# Siril

*Release 1.4.4*

**Free-Astro Team**

**Oct 01, 2026**

# GENERAL


**1 Installation** **3**
1.1 Minimum System Requirements . . . 3
1.2 OS specific installation . . . 4

**2 Graphical User Interface** **15**
2.1 Main interface . . . 15
2.2 Main Toolbar . . . 23
2.3 Burger Menu . . . 26
2.4 Tools Menu . . . 26
2.5 Image Processing Menu . . . 27
2.6 Region of Interest Processing . . . 27
2.7 Image information window . . . 29
2.8 Shortcuts . . . 30

**3 Working Directory** **33**

**4 Preferences** **35**
4.1 Preferences (GUI) . . . 35
4.2 Preferences (commands) . . . 60

**5 File Formats** **67**
5.1 Bit Depth . . . 67
5.2 Common File Formats . . . 67
5.3 FITS . . . 71
5.4 Astro-TIFF . . . 78
5.5 SER . . . 81
5.6 IRIS PIC . . . 82
5.7 PixInsight XISF . . . 82

**6 Color Management** **83**
6.1 Getting Started . . . 83
6.2 Color Management Status . . . 86
6.3 Color Management Dialog . . . 87
6.4 ISO 12646 Image Assessment Mode . . . 89
6.5 Commands . . . 89
6.6 Color Conversion Matrices . . . 90
6.7 Color Management Theory . . . 92
6.8 Workflow . . . 97
6.9 Color Space Profiles . . . 98
6.10 Soft Proofing . . . 102
6.11 Changes caused by introduction of color management . . . 102

**7 Sequences** **105**
7.1 A set of two or more FITS files . . . 105
7.2 A single SER file . . . 105
7.3 A single FITS file . . . 106
7.4 Loading a sequence . . . 106
7.5 Frame selector . . . 107
7.6 Sequence export . . . 109
7.7 Sequence information . . . 110

**8 Definitions and workflow** **111**

**9 Preprocessing** **113**
9.1 Conversion . . . 113
9.2 Calibration . . . 118
9.3 Registration . . . 142
9.4 Drizzle . . . 163
9.5 Stacking . . . 173

**10 Processing** **181**
10.1 Image stretching . . . 182
10.2 Color Calibration . . . 200
10.3 Other Color Tools . . . 221
10.4 Filters . . . 226
10.5 Star Processing . . . 271
10.6 Geometry . . . 281
10.7 Background Extraction . . . 286
10.8 GraXpert Interface . . . 290
10.9 Channel Extraction . . . 296
10.10 Linear Match . . . 302
10.11 RGB compositing . . . 303
10.12 Merge CFA Channels . . . 305
10.13 Pixel Math . . . 308

**11 Astrometry** **315**
11.1 Platesolving . . . 315
11.2 Annotations . . . 330

**12 Dynamic PSF** **343**
12.1 Initial star candidates . . . 343
12.2 Models . . . 344
12.3 Minimization . . . 348
12.4 Use . . . 348
12.5 Configuration . . . 350
12.6 References . . . 352

**13 FITS header** **353**

**14 Intensity Profiling** **357**
14.1 Basic Intensity Profile . . . 357
14.2 Types of Profile . . . 358
14.3 Precise Coordinate Entry . . . 359
14.4 Measurement . . . 359
14.5 Siril Plot Tool . . . 361
14.6 Commands . . . 363

**15 Image Inspectors** **365**
15.1 Tilt . . . 365
15.2 Aberration Inspector . . . 367

**16 Photometry** **369**
16.1 Principles . . . 369
16.2 Quick photometry . . . 373
16.3 Light curves . . . 381

**17 Plotting Feature** **399**
17.1 Registration Plot . . . 399
17.2 Photometry Plot . . . 404
17.3 Shared options . . . 404
17.4 Plot interactions . . . 405

**18 Statistics** **407**
18.1 Estimators . . . 408

**19 Siril for Scientists** **411**
19.1 How can Siril help me with my research? . . . 411
19.2 Which version of Siril should I install? . . . 411
19.3 How can I get in touch with the developers? . . . 411
19.4 How accurate are Siril results? . . . 411
19.5 How can I cite Siril? . . . 412
19.6 How can I open FITS cubes with Siril? . . . 412
19.7 How do I measure FWHM in an image? . . . 412
19.8 How do I use star detection (source extraction)? . . . 412
19.9 What astrometry tools are available? . . . 413
19.10 Can it handle solar system objects? . . . 413
19.11 Can Siril work with spectra? . . . 413
19.12 Can Siril replace IRAF? . . . 413

**20 Scripting** **415**
20.1 Siril Script Files . . . 415
20.2 Python Scripts . . . 422
20.3 Python Script Key Information for Users . . . 428
20.4 Python Script Key Information for Authors . . . 429
20.5 Script Editor . . . 433
20.6 Python Script Examples . . . 436
20.7 Current list of Python scripts for Siril 1.4 . . . 453
20.8 Get Scripts . . . 456

**21 Headless mode** **459**
21.1 Command Stream (Pipe) . . . 460

**22 Path parsing** **461**
22.1 Syntax example . . . 461
22.2 Finding a file with path parsing . . . 463
22.3 Writing a file with path parsing . . . 464
22.4 Using wildcards . . . 465
22.5 Handling duplicates . . . 465

**23 Livestacking** **467**
23.1 Livestacking (GUI) . . . 467
23.2 Livestacking (Headless) . . . 468

**24 Commands** **471**

**25 Sirilpy Python Module API 1.0.25 Reference** **543**
25.1 Dependencies . . . 543
25.2 Siril Script Coding Guidance . . . 544
25.3 Sirilpy Connection . . . 544
25.4 Sirilpy Data Models . . . 570
25.5 Sirilpy Enums . . . 587
25.6 Sirilpy Plotting . . . 593
25.7 Sirilpy Utility Methods . . . 594
25.8 Sirilpy GPU Helpers . . . 598
25.9 Sirilpy TK GUI . . . 603
25.10 Sirilpy Tkfilebrowser . . . 605
25.11 Sirilpy Exceptions . . . 605

**26 How to report issues** **609**
26.1 Check changelogs and bug trackers . . . 609
26.2 Send us useful information . . . 609
26.3 How to contact us? . . . 610

**Bibliography** **611**

**Python Module Index** **613**

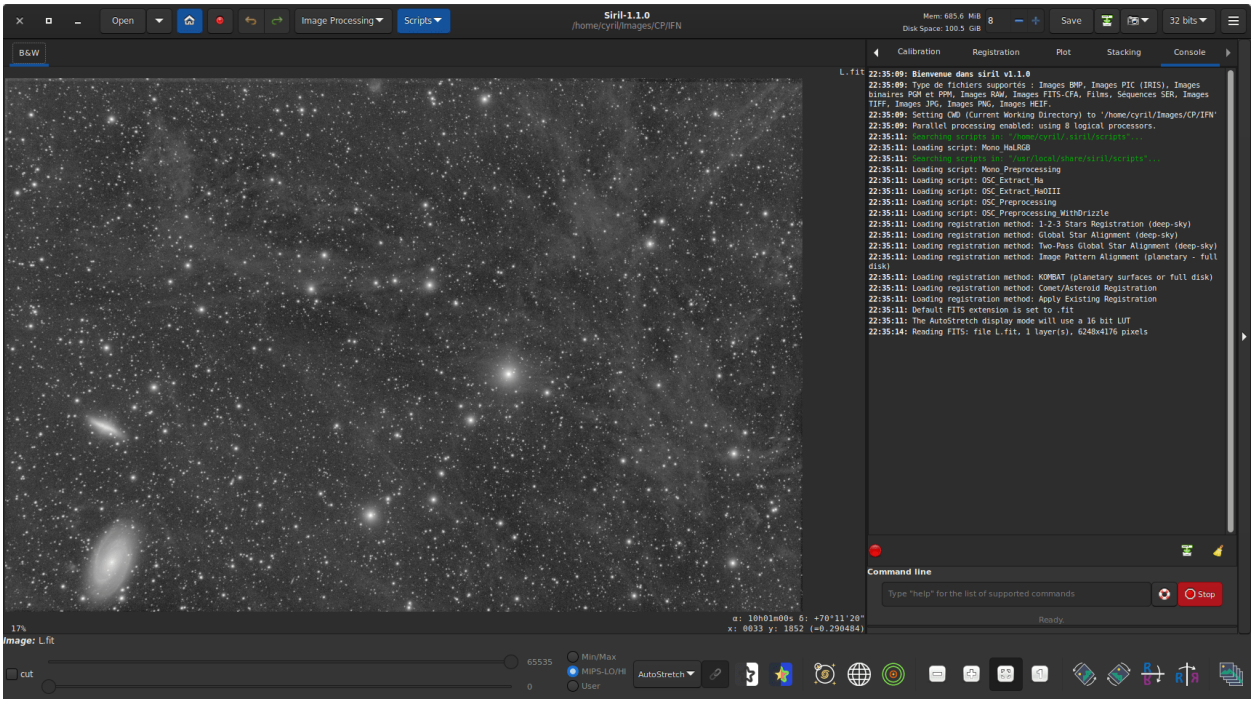
**Commands Index** **615**

**Index** **619**

JOSS 10.21105/joss.07242

This is the documentation of version 1.4.4.

Siril is a deep sky astronomical image processing tool.




Siril can cailbrate, align, stack and enhance pictures from various file formats, even image sequence files (FITS cube, films and SER files). Siril can also analyze images with astrometry and photometry tools.

Programming language is C, with parts in C++. Main development is done with most recent versions of shared libraries on GNU/Linux. Contributors are welcome.

This is the documentation, it tries to describe all Siril functions. If the equivalent of a GUI function exists on the command line, then it is given in an insert. Other useful resources can be found at our main website siril.org.

> **Tip**
>
> This documentation also includes various **tip** and **warning** boxes that highlight important information. Additionally, there are **theory** boxes, which are not essential for understanding the content but are available for those with a mathematical background who are curious to delve deeper into certain concepts.

You can find here an index of Siril commands.

To report any problems in the documentation please open a ticket at the following address: https://gitlab.com/free-astro/siril-doc.

A problem in the translation of the documentation should be reported here: https://gitlab.com/free-astro/siril-localized-doc.

CHAPTER

ONE

# INSTALLATION

Each version of Siril are distributed for the 3 most common platforms (Windows, MacOS, GNU / Linux) and can be downloaded on the Siril website. But of course, as Siril is a free software and you can build the application from the sources.

**Tip**

It may be useful to check the integrity of the binary or package you have just downloaded. The list of SHA checksum is available on this page, in json format.

At the end of the installation you can try the *capabilities* command to learn more about your installation.

**Siril command line**

```
capabilities
```

Lists Siril capabilities, based on compilation options and runtime

### Understanding Siril version numbers

Starting with 1.0 version, stable versions of Siril (such as 1.0, 1.2, etc.) are indicated by even numbers and are designed for everyday use. Development versions, indicated by odd numbers (such as 0.99.0, 1.1.0, etc.), are not usually available as packages or binary executables, and must be compiled by the user. The third and last number, called micro numbering, corresponds to the number of releases that brought bug fixes and other small contributions (such as 1.0.1, 1.0.2, 1.0.3, etc.).

## 1.1 Minimum System Requirements

To ensure optimal performance, it is recommended that your system meets the following minimum specifications:

- **RAM**: At least 8GB of RAM is recommended for all systems to ensure smooth operation. Higher RAM capacity is beneficial as Siril utilizes available RAM extensively for processing tasks.
- **Storage**: An SSD (Solid State Drive) is highly recommended for faster read/write speeds, resulting in **order-of-magnitude faster execution** compared to traditional HDDs

Specific platform requirements are as follows:

- **Windows**: Windows 8.1 or later.
- **macOS**: macOS 13 Ventura or later.

- **GNU/Linux**: No specific minimum requirements stated. Typically, any modern distribution should suffice.

## 1.2 OS specific installation

The following pages explain installation procedures for the different OSes.

### 1.2.1 Installation on GNU/Linux

#### Installation on Debian

The binary package is available on Debian testing and an old version for stable. It can be installed via apt, with superuser privileges:

#### Installation on Ubuntu or Linux Mint

#### Official repositories

As for debian, it is available in the repositories, but the version can be outdated:

```
sudo apt install siril
```

#### PPA repositories

The newer version is thus available in our PPA, which is the preferred way to install Siril on Ubuntu or Linux Mint:

```
sudo add-apt-repository ppa:lock042/siril
sudo apt-get update
sudo apt-get install siril
```

#### Installation of the AppImage binary

For GNU/Linux systems, we also decided to provide bundled binaries with AppImage (x86_64) and flatpak that works on GNU/Linux-like systems. To run the AppImage binary, you just have to download it and allow it to run using the command:

```
chmod +x Path/To/Application/Siril-x.y.z-x86_64.AppImage
```

By replacing with the correct path and x,y and z with the version numbers. Then a simple double-click on the AppImage starts Siril.

#### Installation of the flatpak

Another way to install stable version of siril is to use the flatpak, the utility for software deployment and package management for Linux. To install flatpak, type the following command:

```
flatpak install flathub org.siril.Siril
```

Then, to run the application:

```
flatpak run org.siril.Siril
```

### 1.2.2 Installation on Microsoft Windows

#### Installation with the installer

The recommended way to install Siril is to use the provided installer which will guide you step by step.

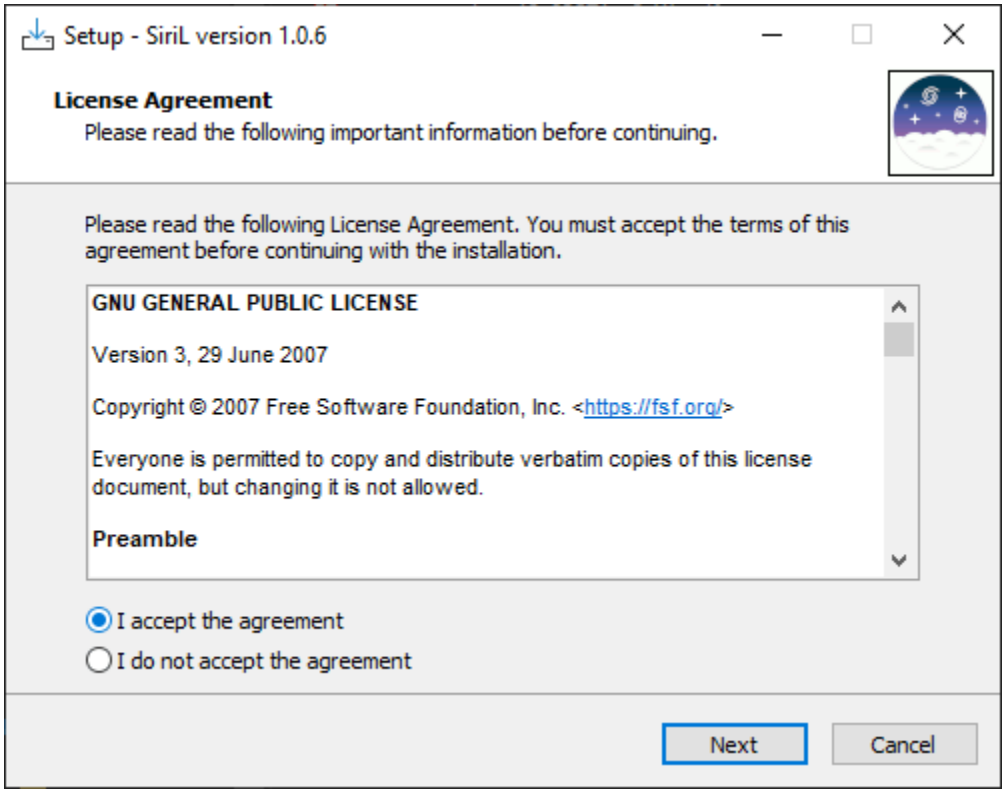



Fig. 1: First screen of the installer, you need to accept the agreement to continue.

The Siril setup wizard will install all the necessary files in the right place and at the end you will have the choice to create or not a shortcut on the desktop.

> **Note**
>
> Siril will be installed in C:\Program Files\Siril. If you don't have the rights to install to this folder, use a portable version instead (see *Installation of the portable binary*.)

#### Installation of the portable binary

If you want to use Siril without installing all kinds of files on your computer (for example if you don't have administrator permissions on the machine), then it is recommended to use the portable version. It comes in the form of a zip file, which you just have to extract to the location of your choice, then go to the `bin` folder to run `siril.exe`. You can also create a shortcut on your desktop to make it easier to launch the application.

> **Warning**
>
> Be careful, under no circumstances you should move the `exe` file, or any other file. Otherwise Siril will not run.

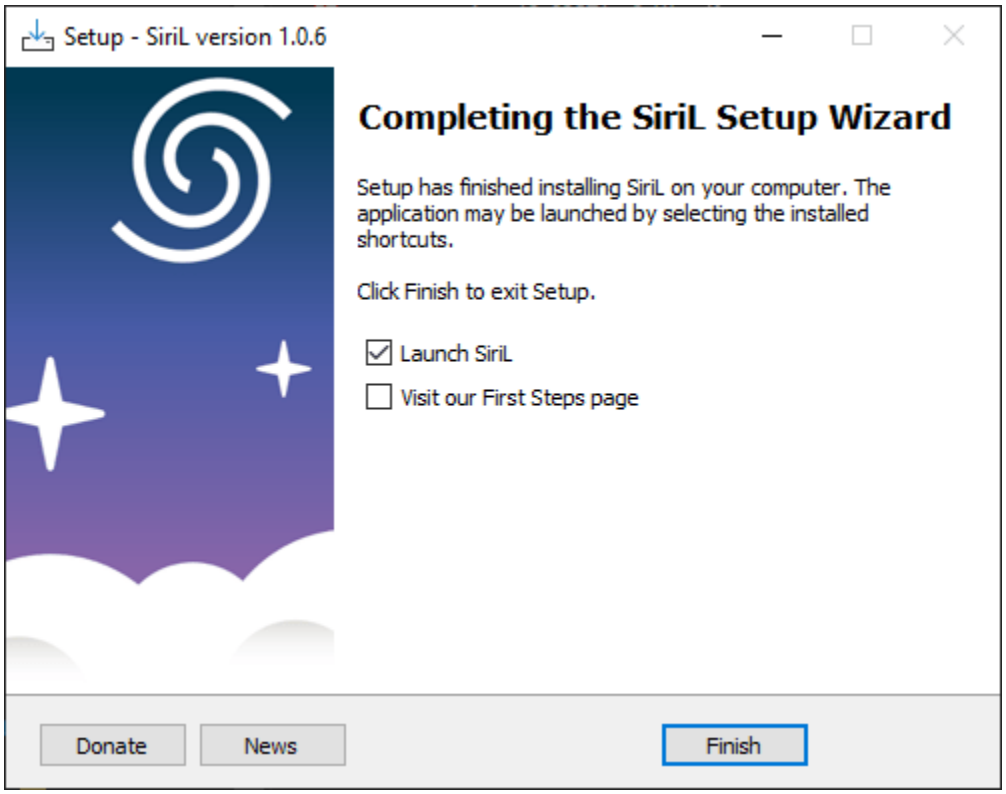



Fig. 2: Last screen of the installer. You can chose to start Siril right after installation, and to open the tutorial explaining the first steps.

### Building on Windows with Msys2

These instructions are made for compiling on Windows with MSYS2 using the UCRT64 toolchain. MSYS2 requires 64-bit Windows 10 or newer, and does not work with FAT filesystems.

> **Note**
>
> Starting with Siril 1.4.2, UCRT64 is the only supported build environment on Windows. The MinGW64 environment, previously supported as an alternative, has been deprecated by the MSYS2 project on March 15th, 2025. Several Siril dependencies have already been removed from the MinGW64 package index, and more are expected to follow. All future releases of Siril will be built exclusively with UCRT64.
>
> If you are using the installer or portable binaries provided on the Siril website, this change is transparent to you. If you are building Siril from sources, you must use the UCRT64 environment.
>
> Using UCRT64 also increases the number of files that can be handled in a sequence to 8192.

Download MSYS2 64bit, a software distribution and building platform for Windows, and run the `x86_64` installer for 64-bit. When asked, specify the directory where MSYS2 will be installed.

Run MSYS2 directly from the installer, or later **MSYS2 MinGW UCRT x64** from the Start menu or a shortcut.

> **Warning**
>
> Make sure to launch the UCRT x64 shell (the icon at the top of the terminal window should be yellow).

First, update the package database and core system packages by typing (for more information about pacman, see this page):

```
pacman -Syu
```

### Installing dependencies

To install the dependencies, enter the following command:

```
curl -s -o siril-deps.sh https://gitlab.com/free-astro/siril/-/raw/master/build/windows/
↪native-gitlab-ci/siril-deps.sh
bash siril-deps.sh
```

> **Note**
>
> The link above points to the latest version of the dependencies installation script. We have had a tendency in the past not to remove any dependencies (rather to add more), so even if you want to build the stable branch, you should be okay using the latest dependencies script. In case a dependency is missing for a specific version, you can always check the script from the corresponding tag in the GitLab repository.

### Building from source

The source code is hosted on GitLab. Download it with the following command:

```
git clone https://gitlab.com/free-astro/siril.git
cd siril
git submodule update --init
```

Generate the build system and compile the code:

```
meson setup _build --buildtype release
ninja -C _build install
```

To launch your build of Siril, open an MSYS2 UCRT64 shell and type:

```
siril
```

You can also create a shortcut to `siril.exe`, located by default at `C:\msys2\ucrt64\bin\`.

To update your version, open an MSYS2 UCRT64 shell, then:

```
pacman -Syu
cd siril
git pull --recurse-submodules
meson setup _build --reconfigure
ninja -C _build && ninja -C _build install
```

If `git pull` does not show any changes, there is no need to rebuild. Otherwise, the above commands will update your build. Then launch Siril with:

```
siril
```

**Warning**

Starting with Siril 1.3.6, you will need to have Python installed on your system if you build from sources. The first time you launch a self-built Siril, you must do so from a native Windows environment, not from the MSYS2 shell, in order for Siril to detect the Windows Python installation rather than the MSYS2 one (a notification will appear in the console if this step is skipped, and the sirilpy module will not be initialized). This is a one-time requirement. Once the virtual environment has been set up, you can start Siril from the MSYS2 shell as usual.

To launch Siril in a Windows environment, either open a Windows terminal and run `C:\msys2\ucrt64\bin\siril.exe`, or locate this file in Explorer and double-click it.

### 1.2.3 Installation on macOS

#### Installation of our app

The macOS app is provided per architecture:

- Intel (macOS 13 Ventura+)
- Apple Silicon (macOS 13 Ventura+)

Choose the link corresponding to your processor architecture and download the disk image. Once downloaded, double-click to open it.

A new window pops up. Now drag the `Siril` icon and drop it on the `Applications` one.

Congratulations, Siril is now installed.

Siril

Siril
1.1.0 (24)
Siril
Applications

#### Installation from Homebrew

Homebrew is similar to MacPorts and provides packages (aka formulae) to install, either by compiling them from source or by using pre-compiled binaries (aka bottles). To install Homebrew, click here. Siril can be installed with:

```
brew install siril
```

> **Note**
>
> Please be aware that it was announced that Homebrew is using analytics. To turn this off, run: `brew analytics off` You can read more about this on Brew Analytics.

### 1.2.4 Installation from source code

Installation from source code is required if you want the latest features, if the past release is getting old, if you want to participate in improving Siril or not use all the dependencies.

#### Getting the source code

The source code is stored on a git repository, which you can download with this command the first time:

```
git clone --recurse-submodules https://gitlab.com/free-astro/siril.git
```

And update it the following times with these commands in the base siril directory:

```
git pull
git submodule update --recursive
```

#### Dependencies

Siril depends on a number of libraries, most of which should be available in your Linux distribution or package manager of choice. The names of the packages specific to operating systems are listed in each section below. Mandatory dependencies are:

- gtk+3 (Graphical user interface library), at least version 3.20.
- adwaita-icon-theme (icons) to support gtk's look and feel.
- cfitsio (FITS images support).
- fftw (Discrete Fourier Transform library).
- gsl (The GNU Scientific Library), version 1 or 2 starting with release 0.9.1 or SVN revision 1040.
- OpenCV and a C++ compiler for some image operations.
- Little CMS an open-source color management system
- wcslib for world coordinate system management, annotations and the photometric color calibration.
- gtksourceview4 for multiline text editing. GtkSourceView adds support for syntax highlighting, undo/redo, file loading and saving, search and replace, a completion system, printing, displaying line numbers, and other features typical of a source code editor.

> **Note**
>
> Even if Siril can run in console since version 0.9.9, it is still linked against the graphical libraries, so you still need GTK+ to compile and run it.

Optional dependencies are:

- openmp for multithreading. Although optional, this dependency is highly recommended as the performance will be much better. The flag of this option is set to true by default. That means if openmp is not installed on your machine, you must add `-Dopenmp=false` in the meson setup.
- libraw, libtiff, libXISF, libjpeg, libjxl, libpng, libheif for RAW, TIFF, XISF, JPEG, JPEG XL, PNG, HEIF and AVIF images import and export. The libraries are detected at compilation-time.
- FFMS2 for film native support as image sequences. It also allows frames to be extracted from many kinds of film, for other purposes than astronomy. Versions < 2.20 have an annoying bug. It is recommended to install the latest version.
- ffmpeg (or libav), providing libavformat, libavutil (>= 55.20), libavcodec, libswscale and libswresample for mp4 sequence export.
- wcslib for world coordinate system management, annotations and the photometric color calibration.
- Exiv2 to manage image metadata.
- libcurl for astrometry and photometry requests.
- libgit2 to maintain a local clone of the siril-scripts repository.

### Build dependencies

To install from source code, you will have to install the base development packages:

```
git, autoconf, automake, libtool, intltool, pkg-tools, make, cmake, gcc, g++
```

The compilers gcc and g++ from this list can be replaced by clang and clang++ (we use them for development), probably others as well.

The autotools packages (autoconf, automake, probably some others) can be replaced by meson.

### Generic build process

Siril can be compiled either using autotools or meson.

### Meson

The recommended way is to use meson and ninja:

```
meson setup _build --buildtype release

cd _build
ninja
ninja install
```

To disable some dependencies or features, use meson options `-Dfeature=false` or `-Denable-feature=yes` depending on the case.

Table below lists all configurable options.

| Option | Type | Value | Choices | Description |
|---|---|---|---|---|
| relocatable-bundle | combo | platform-default | [ 'yes', 'no', 'platform-default' ] | build with resources considered bundled under the same prefix |
| openmp | boolean | true | N/A | build with OpenMP support |
| exiv2 | boolean | true | N/A | build with exiv2 support |
| libgit2 | boolean | true | N/A | build with libgit2 git integration |
| libraw | boolean | true | N/A | build with LibRaw support |
| libtiff | boolean | true | N/A | build with TIFF support |
| libjpeg | boolean | true | N/A | build with JPEG support |
| libjxl | boolean | true | N/A | build with JPEG XL support |
| libpng | boolean | true | N/A | build with PNG support |
| libheif | boolean | true | N/A | build with HEIF support |
| libXISF | boolean | true | N/A | build with XISF support |
| ffms2 | boolean | true | N/A | build with FFMS2 support |
| ffmpeg | boolean | true | N/A | build with FFmpeg support |
| libcurl | boolean | true | N/A | Use libcurl for web access |
| criterion | boolean | false | N/A | build with criterion support |
| flatpak-id | string | | N/A | set by flatpak-builder if building a flatpak |

### Autotools

The autotools ways is well known in the unix world, once the source code has been downloaded and the prerequisites have been installed, the general way to build it is as such:

```
./autogen.sh
make
make install
```

possibly with superuser privileges for the last line.

You may want to pass specific options to the compiler, for example like that if you want optimisation and installation in `/opt` instead of the default `/usr/local`:

```
CFLAGS='-mtune=native -O3' ./autogen.sh --prefix=/opt
```

To launch Siril, the command name is **`siril`** or **`siril-cli`**.

### Installation on Debian-like systems

You may want to build a `.deb` package instead of using a non-packaged version, in that case see this help. In particular, to install dependencies, you can use the command:

```
apt build-dep siril
```

Otherwise, here is the list of packages for the current version:

- Packages required for the build system:

```
autoconf automake make gcc g++ libtool intltool pkg-config cmake
```

- List of packages for mandatory dependencies:

```
libgtk-3-dev libcfitsio-dev libfftw3-dev libgsl-dev libopencv-dev
liblcms2-dev wcslib-dev libgtksourceview-4-dev
```

- List of packages for optional dependencies:

```
libcurl4-gnutls-dev libpng-dev libjpeg-dev libtiff5-dev
libraw-dev gnome-icon-theme libavformat-dev libavutil-dev libavcodec-dev
libswscale-dev libswresample-dev libgit2-dev libheif-dev
libexiv2-dev libjxl-dev liblz4-dev libpugixml-dev libxisf-dev
```

for film input (AVI and others) support:

```
libffms2-dev
```

### Installation on Arch Linux

Two packages are available on AUR: **`siril`** and `siril-git`. Download the `PKGBUILD` or the repository, install dependencies, run makepkg to build the package and `pacman -U` to install it.

Dependencies (mandatory and a few optional):

```
pacman -S base-devel cmake git intltool gtk3 fftw cfitsio gsl opencv
exiv2 libraw wcslib gtksourceview4
```

### LittleCMS version

Siril requires lcms2 >= 2.14 to build. This version is required in order to provide optmisation of the color management code. A sufficient version is available in debian testing and Ubuntu 23.04, however the current Ubuntu LTS release provides an older version, and other OS distributions may do too. If you find yourself in the position of being on an older OS that does not provide lcms2 >= 2.14, you can either manually install packages from a more recent version of your distribution, or you can build and install lcms2 from source code. The lcms2 source is available at their git repository.

### Build Failures

Every commit to Siril git is automatically built in a standard build environment for Linux, Windows and MacOS using the gitlab CI infrastructure. This means that we have high confidence that the master branch, as well as tagged releases, **will** build successfully given a correctly set up build environment with the necessary dependencies installed.

If you experience a build failure it is likely that this indicates a problem with your build environment or incorrectly installed dependencies - remember many distributions require separate installation of development packages that contain the necessary header files. Check the CI report for the git commit you are trying to build. In the unlikely event that there is a build failure shown, rest assured the team is working to fix it. Otherwise, if the CI pipeline shows green ticks, you will need to review and fix any issues with your own build environment.

If you still believe you have found a build issue that has not been flagged by the CI pipeline - for example if you're building on a different platform such as BSD that the developers don't regularly use - then feel free to raise an issue on gitlab.

Note that issues should only be raised against the master branch or tagged releases. If you are testing new features in merge requests, please provide feedback in comments against the relevant merge request.

### System Python

If you are building Siril yourself you need to ensure you have a working system Python installation including pip and venv. Note the last point because on some systems (at least Debian / Ubuntu based systems, possibly others) Python is split into many packages and you must ensure the packages that provide pip and venv are installed.

CHAPTER

TWO

# GRAPHICAL USER INTERFACE

The Graphical User Interface (GUI) allows you to process your images manually, as well as using scripts or typing commands. To learn how to use Siril in headless mode, please refer to this *section*.

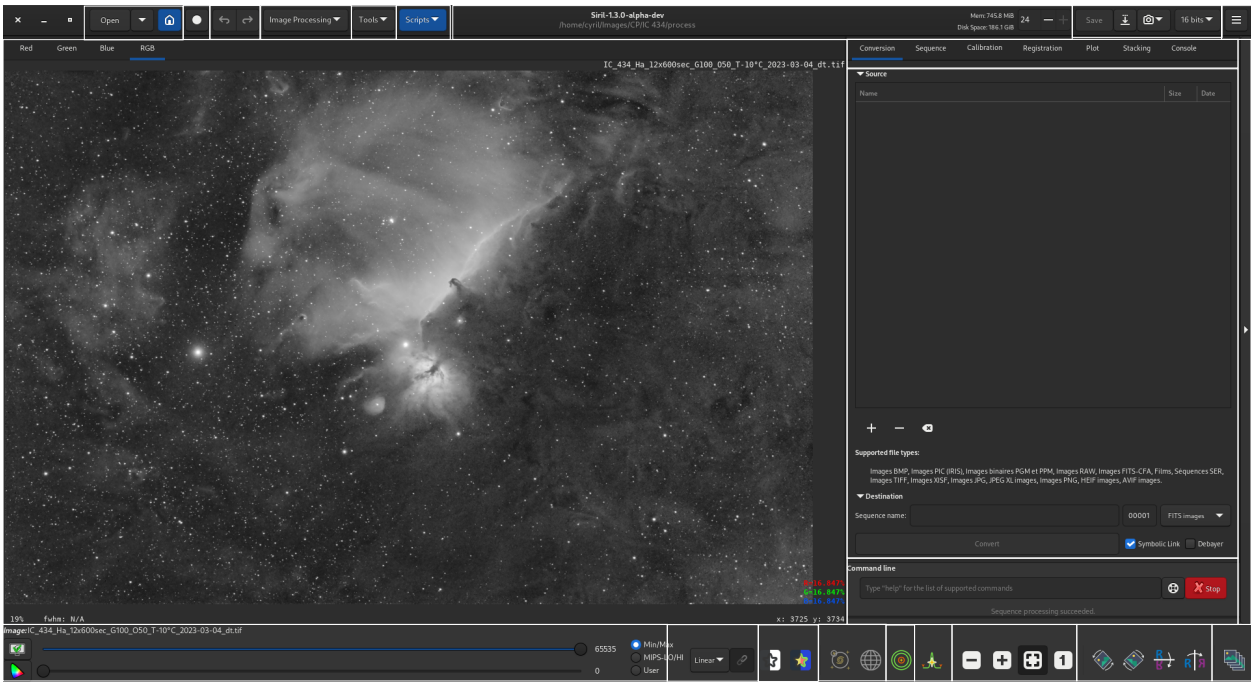
Siril's GUI is written using GTK, a free and open source cross-platform toolkit for GUI creation. Currently, the version used is version 3.

The following subsections take you through the main interface window and useful menus.

## 2.1 Main interface

When launching Siril, the main interface opens.

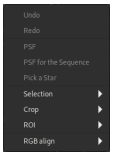
> **Note**
>
> Click anywhere onto the image below to display its functions.



### Image area

This area displays the currently loaded image. Click on *Red*, *Green* or *Blue* to switch between the different layers (color images only, a single *BW* tab is available for mono images).

Right-click on the image to display a contextual menu:




> **Tip**
>
> When no image is loaded, double-clicking on the image area pops the *Open* dialog.

## 2.1.1 Undo

Undoes the most recent operation.

## 2.1.2 Redo

Redoes the most recently undone operation.

## 2.1.3 PSF

Computes a PSF for the current selection and carries out quick photometry.

## 2.1.4 PSF for the Sequence

Computes a PSF for the current selection for all images in the current sequence.

## 2.1.5 Pick a Star

Applies star detection to the current selection. The result is opened in the *Dynamic PSF* dialog. Note this operation is more lenient than the full image star finder routine as it assumes that you have picked a selection containing a star: it can therefore be used to pick stars that have been missed by the full image star finder function.

## 2.1.6 Selection

Applies constraints to the selection box.

## 2.1.7 Crop

Provides an interface to the *Crop* and *Crop & rotate* functions.

> **Tip**
>
> If a CFA image is loaded, the crop will be constrained to a CFA boundary so that the effective CFA pattern is unchanged for future operations. For Bayer CFA patterns this means the crop selection will snap to a multiple of 2x2 and may shift by up to 1 pixel in x and y directions in order to align with the Bayer pattern start. For X-Trans patters the crop selection will snap to a multiple of 6x6 and may shift by up to 5 pixels in each direction.

> **Warning**
>
> Rotating a CFA image by arbitrary angles will destroy the CFA pattern: you should progress your workflow to the point where the CFA pattern has been debayered or Bayer drizzled before doing anything that applies rotation or scaling.

### 2.1.8 ROI

Provides an interface to set or clear the Region of Interest for ROI processing. Note: this menu entry is not available if the ROI mode is set to "Automatic from selection" in the UI preferences tab.

### 2.1.9 RGB align

Aligns the image RGB channels. This supports a range of registration methods for alignment:

- **2-pass global star alignment** This uses the 2-pass global star alignment routine with COG framing (so the image dimensions are preserved). This algorithm adjusts with 8 degrees of freedom so it can correct for image shift, rotation and skew. This is the recommended algorithm for images containing stars. This algorithm is global and does not require a selection to be made.
- **KOMBAT alignment** This uses the KOMBAT alignment method. It is primarily intended for planetary alignment, but can also work for deep space images. It provides shift registration only, but is fast. This algorithm requires a selection to be made.
- **One star registration** This is an older algorithm that provides shift-only registration based on the movement of a single star. This is very fast but the registration cannot cope with as many degrees of freedom as the 2-pass global star alignment. It may be suitable if you are certain you only need to correct for image shift. This algorithm requires a single star to be selected.
- **Image pattern alignment** This method performs alignment using the image Fourier transforms. It works for deep space and planetary images but provides only shift registration and is quite slow. This algorithm uses a square selection: a selection must be made, and non-square selections will be corrected to be square.

> **Warning**
>
> The tool is **RGB** align. It is not intended for, and will not work for, aligning unaligned **LRGB** compositions. In order to do this you must either align the layers to be composed before composition, or align using the RGB alignment tool.

*Back to figure*

#### Open

Click on these icons (from left to right) to:

- open a file
- open a recent file
- change the *working directory*

*Back to figure*

### Livestack

Click on this button to start a *Livestacking* session.

*Back to figure*

### Undo/Redo

Use these buttons to undo/redo last actions. This is only available if last action has been done through GUI, not by typing a command.

*Back to figure*

### Image processing

Click on this button to display the *Processing* menu.

*Back to figure*

### Tools

Click on this button to display the *Tools* menu.

*Back to figure*

### Scripts

Click on this button to display and launch the *scripts*.

*Back to figure*

### Information bar

This bar displays the current version of Siril and the path to the current *working directory*.

- To the right, informationabout available RAM and disk space id also given.
- You can change the available number of threads used by Siril using the +/- signs.

*Back to figure*

### Save

These buttons are used to save your results:

- save (overwrites) the current image.
- save with a different name and/or extension.

  The drop-down list at the bottom right allows you to choose the type of image recorded. It automatically adds the extension to the file name. However, by staying in the *Supported Image Files* mode, it is possible to add any extension supported by Siril by hand and it will save in the correct file format.
- take a snapshot of the current view (as seen on the screen, meaning preview strecthing, if any, is applied). There are two possible options. Either the snapshot is saved in the clipboard, or directly copied to the disk in the *working directory*.
- change the bitdepth of the current image. The choice is between 16-bit and 32-bit.

*Back to figure*

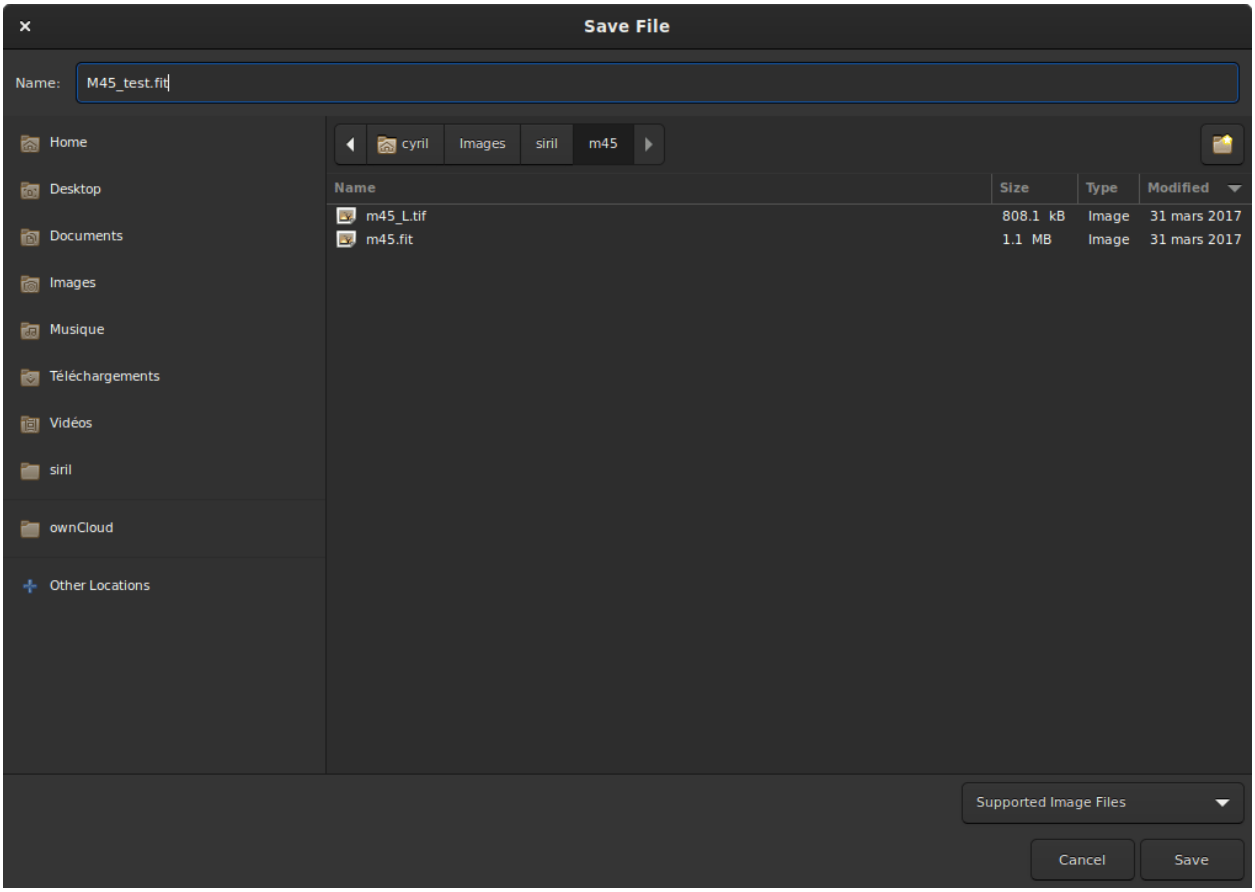



Fig. 1: Save dialog box.

### Burger menu

Opens the main menu, also called *burger menu*. Gives access to *Preferences*, the documentation and much more.

*Back to figure*

### Tabs

Selects one of the tabs. You can also switch between the different tabs using `F1` to `F7` shortcuts.

More details can be found there:

| Tabs | Keys |
|---|---|
| *Conversion* | `F1` |
| *Sequence* | `F2` |
| *Calibration* | `F3` |
| *Registration* | `F4` |
| *Plot* | `F5` |
| *Stacking* | `F6` |
| Console | `F7` |

*Back to figure*

### Tab window

Displays the specifics of the currently selected tab.

*Back to figure*

### Mouse behaviour

Siril uses GTK primary, secondary and middle mouse button mappings. These are usually mapped to left, right and middle (`scroll` wheel button) physical mouse buttons, however they may be mapped differently in specific setups (e.g. left-handed users may have configured swapped left and right buttons).

1. **Primary mouse button**

   The primary mouse button is used for a number of purposes:

   - To select areas of the image
   - With `Ctrl` pressed (or `Cmd` on MacOS), to pan the image
   - When the intensity profiling tool is active, to drag the line along which the intensity profile is to be drawn, and to select points corresponding to known wavelengths / wavenumbers in the spectroscopy setting
   - In the RGB composition tool, to select the rotation center when using manual alignment
   - In the gradient removal tool, to draw samples
   - In photometry mode, to perform photometry on a star
   - To select the positions for the registration previews
   - Double clicking the primary mouse button with no image loaded will bring up the *Open* dialog.

2. **Secondary mouse button**

   The secondary mouse button is used for a number of purposes:

   - To bring up the image context menu (except when in photometry mode)

- In the gradient removal tool, to remove samples
- In photometry mode, to perform photometry on the selected star for all frames of the currently loaded sequence (if no sequence is loaded, this has no effect)

3. **Middle mouse button**

   The middle mouse button is used for the following purposes:

   - To pan the image at the current zoom level
   - With `Ctrl` pressed (or `Cmd` on MacOS), to make a square selection suitably sized for photometry
   - Double clicking adjusts the zoom level to a preset value. The behaviour is configurable in the *User Interface* tab in the *Preferences* dialog. The choices are:
     - Always zoom to fit
     - Always zoom to 100% centered on the mouse cursor
     - Toggle between zoom to fit and zoom to 100% centered on the mouse cursor

4. **Scroll wheel**

   - The `scroll` wheel is used to adjust the zoom level.

### Command line

Type in a *command* and press `Enter`.

- You can press the button at the end of the line to get some help on the usage.
- You can also abort the process being currently executed by clicking on the *Stop* button.

*Back to figure*

### Expand

Click on this bar to expand/retract the whole tab/tab window area.

*Back to figure*

### Image sliders

Use the top and bottom sliders to adjust the white and black points of the previewed image (in Linear mode).

> **Tip**
>
> Click on the name of the *Image* or *Sequence* loaded to copy its name to the clipboard (useful to paste in a command).

*Back to figure*

### Preview mode

Select the preview mode for the loaded image, between the following choices:

- Linear
- Logarithm
- Square root
- Squared

- Asinh
- Autostretch (tick the High Definition box to use a deeper (up to 24-bit, configurable in Preferences) LUT instead of the default 16-bit one)
- Histogram

In Autostretch mode with color images, the toggle to the right activates/deactivates channel linking. When unlinked, the 3 layers are stretched independantly so as to give a more balanced image.

> **Warning**
>
> This is just a preview of the image, not the actual data (except if Linear mode is selected). Do not forget to stretch your images before saving them.

*Back to figure*

### Special views

Use these toggles to show previewed images:

- in inverted colors
- in false colors

*Back to figure*

### Astrometry tools

Use these toggles to show:

- astrometric *annotations*
- celestial grid

> **Warning**
>
> The loaded image needs to be plate-solved for these buttons to be active.

*Back to figure*

### Quick photometry

Use this toggle to trigger the *quick photometry* mode.

*Back to figure*

### Intensity profile

Use this toggle to trigger the *intensity profile* mode.

*Back to figure*

### Zoom

Use these buttons to:

- Zoom out
- Zoom in
- Zoom to fit the avaialble window space
- Zoom to actual size

> **Tip**
>
> `Ctrl+left clic` will allow to navigate into the picture

> **Tip**
>
> `Ctrl+mouse scroll` will zoom in/out and `Ctrl` + `0` / `1` will zoom to fit/100%.

> **Tip**
>
> `Ctrl+Shift` and drag with the primary mouse button will measure the distance between two points. If sufficient metadata is available the measurement will be given in degrees, minutes and arcseconds, otherwise it will be given in pixels.

*Back to figure*

### Geometric transformations

Use these buttons to:

- Rotate left
- Rotate right
- Mirror about horizontal axis
- Mirror about vertical axis

*Back to figure*

### Frame selection

Click on this button to open the *frame selector*.

*Back to figure*

## 2.2 Main Toolbar

The main toolbar is located at the bottom of Siril's main window. It provides quick access to display controls, analysis tools and basic image transformation commands.

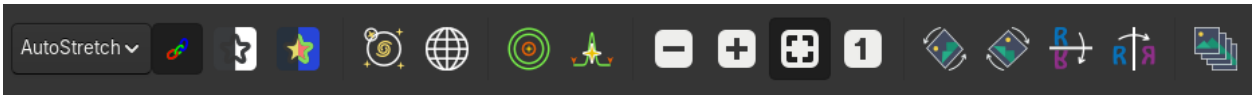



Fig. 2: The Siril main toolbar.

> **Note**
>
> The toolbar is only active when an image is loaded in memory.

### 2.2.1 Display Group

#### Display mode

The first control is a drop-down menu button showing the current display mode (`Linear` by default). Clicking it opens the list of all visual modes available in Siril:

- **Linear** — direct rendering of pixel values. This is the reference mode.
- **Logarithm** — applies a logarithmic transfer function, compressing bright areas and bringing out faint details.
- **Square root** — applies a square root transfer function, a gentler alternative to the logarithm.
- **Squared** — applies a quadratic transfer function, boosting bright areas at the expense of the shadows.
- **Asinh** — applies an inverse hyperbolic sine transfer function, offering a smooth and natural-looking stretch.
- **AutoStretch** — automatically computes and applies a stretch for a visually pleasing result. A **High definition** sub-option is available to increase the precision of the stretch computation.
- **Histogram** — applies a stretch based on the image histogram for a balanced display.

> **Warning**
>
> Only **Linear** mode (with both visualisation sliders at their minimum and maximum positions) reflects the true pixel values. All other modes are purely visual aids. Stretch the histogram before exporting to an external tool.

#### Channel linking

Immediately to the right of the mode selector is a toggle button showing a chain link icon.

- **Closed chain** — the R, G and B channels are linked in AutoStretch mode.
- **Broken chain** — the channels are stretched independently.

### 2.2.2 Visual Rendering Group

#### Negative view

Toggles the display between the normal view and the **negative view** of the image. Useful for detecting subtle artefacts or faint gradients.

#### False colour / rainbow colormap

Toggles between normal display and a **false colour rendering** (rainbow palette). This mode helps visualise intensity variations across the image more clearly.

### 2.2.3 Astrometry Group

#### Object annotation

- **Left-click** — displays the names of celestial objects if WCS (World Coordinate System) information is available in the image.
- **Right-click** — opens the list of available astrometric catalogues to select the annotation source.

#### Celestial grid

Shows or hides the **celestial coordinate grid** (right ascension / declination) if WCS information is available in the image.

### 2.2.4 Analysis Group

#### Photometry / PSF

Switches to *photometry / PSF mode*. In this mode:

- A click on a star performs a PSF (Point Spread Function) fit and displays the photometric parameters.
- If a sequence is loaded, a **right-click** on the displayed image applies the PSF/photometry analysis to the entire sequence.

#### Intensity profile cut

Activates the *intensity profile cut tool*: draw a segment between two points on the image to display a graph of the pixel values along that line.

### 2.2.5 Zoom Group

#### Zoom out

Reduces the zoom factor (shrinks the displayed image). Usual keyboard shortcut: `-`.

#### Zoom in

Increases the zoom factor (enlarges the displayed image). Usual keyboard shortcut: `+`.

#### Fit to window

Toggle button. When active, the image is **automatically scaled** to fill the available area of the window.

#### Actual size — 1:1

Displays the image at its **actual size** (1 image pixel = 1 screen pixel).

### 2.2.6 Geometric Transformations Group

#### Rotate 90° counter-clockwise

Applies a **90° counter-clockwise rotation** to the image.

#### Rotate 90° clockwise

Applies a **90° clockwise rotation** to the image.

#### Horizontal mirror

Applies a **horizontal flip** (reflection about the vertical axis).

#### Vertical mirror

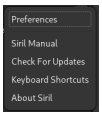
Applies a **vertical flip** (reflection about the horizontal axis).

### 2.2.7 Sequence Group

#### Sequence image list

Shows or hides the **side panel listing the images in the current sequence**, along with the registration data associated with each frame.

## 2.3 Burger Menu




#### Preferences

Opens the *Preferences* menu.

#### Siril Manual

Opens the online documentation.

#### Check for updates

Checks if a newer version is available.

#### Keyboards shortcuts

Opens a panel reminding all the available *Shortcuts*.

#### About Siril

Opens a dialog showing the version information and Credits.

## 2.4 Tools Menu

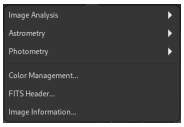



Fig. 3: Tools menu

Introduced with version 1.4, this menu brings together Siril's various tools, such as image analysis, astrometry, photometry, *etc*.

It consists of several nested menus and sub-menus, organized as follows:

- **Image Analysis**
  - *Statistics*
  - *Noise Estimation*
  - *Aberration Inspector*
  - *Show Tilt*
  - *Show Distortions*
  - *Dynamic PSF*
- **Astrometry**
  - *Image Plate Solver*
  - *Annotate*
- **Photometry**
  - *Create Comparison Star File*
  - *Automated Light Curve*
  - **Create Output**
    - *Light Curve (ETD format)*
    - *AAVSO file format*
- *Color Management*
- *FITS Header*
- *Image Information*

For detailed information on each item, click on the respective entry.

## 2.5 Image Processing Menu

This menu is explained in detail in the following *page*.

## 2.6 Region of Interest Processing

Siril offers Region of Interest (ROI) processing for some functions. ROI processing serves two purposes:

- For image operations that are slow to compute, such as deconvolution, it allows processing only a small portion of the image while experimenting to find the ideal parameters. This means the operation completes much quicker on the small ROI and allows the desired parameters to be chosen much more quickly.
- For image operations that affect the look of the image such as stretches, saturation tools etc., it can be convenient to use ROI processing to compare side-by-side an unprocessed and a processed area.

ROI processing is not universal to all image operations in Siril. In fact, there are a great many where it wouldn't make sense, for example geometry operations where the contents of the ROI would be mapped to a different part of the image. ROI processing is not supported for image operations that take multiple input images, i.e. *Star Recomposition*, *RGB Composition* or *Pixel Math*.

### 2.6.1 Operations supporting ROI Processing

The following image operations support ROI processing:

- **Deconvolution** Deconvolution can be very slow, so ROI processing allows for quicker trialling of options on a small part of the image in order to find the best settings for your image.
- **Noise Reduction** Noise reduction can also be slow to process, so ROI processing allows for quicker trialling of options.
- **Asinh Stretch** A user may wish to examine the effect of asinh stretch, or any of the following image operations, side by side with an unprocessed area. ROI processing allows this.
- **Generalized Hyperbolic Stretch**
- **Histogram Stretch**
- **Curves Transformation**
- **Color Saturation**
- **Median Filter**
- **Remove Green Noise (SCNR)**
- **Edge Preserving Filters**
- **Unpurple Filter**

No other image operations support ROI processing.

> **Note**
>
> ROI processing is not generally intended as a means of processing only part of an image.
>
> In image operations with an automatically updating preview, the preview will appply to the ROI but when *Apply* is clicked, the entire image will be updated.
>
> In image operations that do not have an automatically updating preview, when a ROI is set a *Preview in ROI* button will become available. This serves as a manual preview button, so you can preview a range of settings in the ROI until you are happy. Once the settings are as desired, clicking Apply will apply the operation to the whole image.

### 2.6.2 Setting the ROI

There are two options for setting the ROI:

- **Manual** This option requires the ROI to be set and cleared manually using the right mouse button menu, in the *ROI → Set ROI to selection* sub-menu.
- **Automatic from selection** This option automatically synchronizes the ROI with the current selection, and automatically clears it when the selection is cleared.

These options are mutually exclusive and the preferred option can be configured in the *User Interface* tab of the *Preferences* dialog.

### 2.6.3 Clearing the ROI

In manual ROI setting mode, the option to clear the ROI is located in the right mouse button menu, in the *ROI → Clear ROI* sub-menu.

In automatic ROI setting mode, simply clear the current selection and the ROI will clear.

The ROI will also automatically clear on the following occasions:

- When the current image is closed,
- When a new image is opened,
- When a sequence is open and the visible frame is changed.

### 2.6.4 Visualisation

When a ROI is set, it is shown in the image window as a red dashed box in the same style as the selection box. However, when an image operation dialog is opened and the image operation supports ROI processing (or if a ROI is set when such a dialog is already open), the ROI outline will change color to green, indicating that ROI processing is enabled for the current operation.

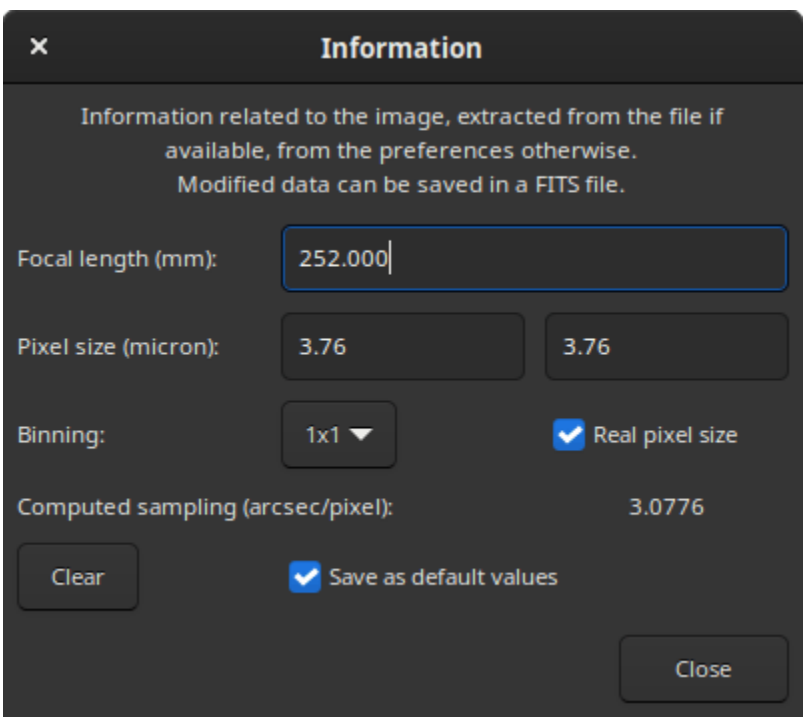
### 2.6.5 Siril Commands

Siril commands ignore the ROI, even if run in GUI mode from the command line, and even if the same image operation does support ROI processing when called from the GUI. The ROI is intended as a quick preview, whereas commands are intended to operate on the whole image. If a ROI is in place when a command runs it will be ignored, but not cleared, so it will still be available for subsequent operations that do support ROI processing.

### 2.6.6 Color Space Transforms

Any operation to remove, assign or convert the image's ICC profile will clear the ROI, if one is set.

## 2.7 Image information window




This window provides information about what is known about the sampling of the opened image. The sampling, sometimes also called resolution or scale, indicates how much angle of the sky is seen in one pixel, as seen through the instrument. It depends on two things: the focal length of the instrument and the pixel size of the sensor, itself depending on the binning mode.

FITS headers can contain this information if it was given to the acquisition software. In that case this is the values that are shown in this window. If not available in the image metadata, because it was unknown to the acquisition software, or more simply because the file format does not support it, this dialog will still be available and populated with *default values*. They can be modified and used for various operations of Siril that require sampling information, for example displaying FWHM in arc seconds instead of pixels.

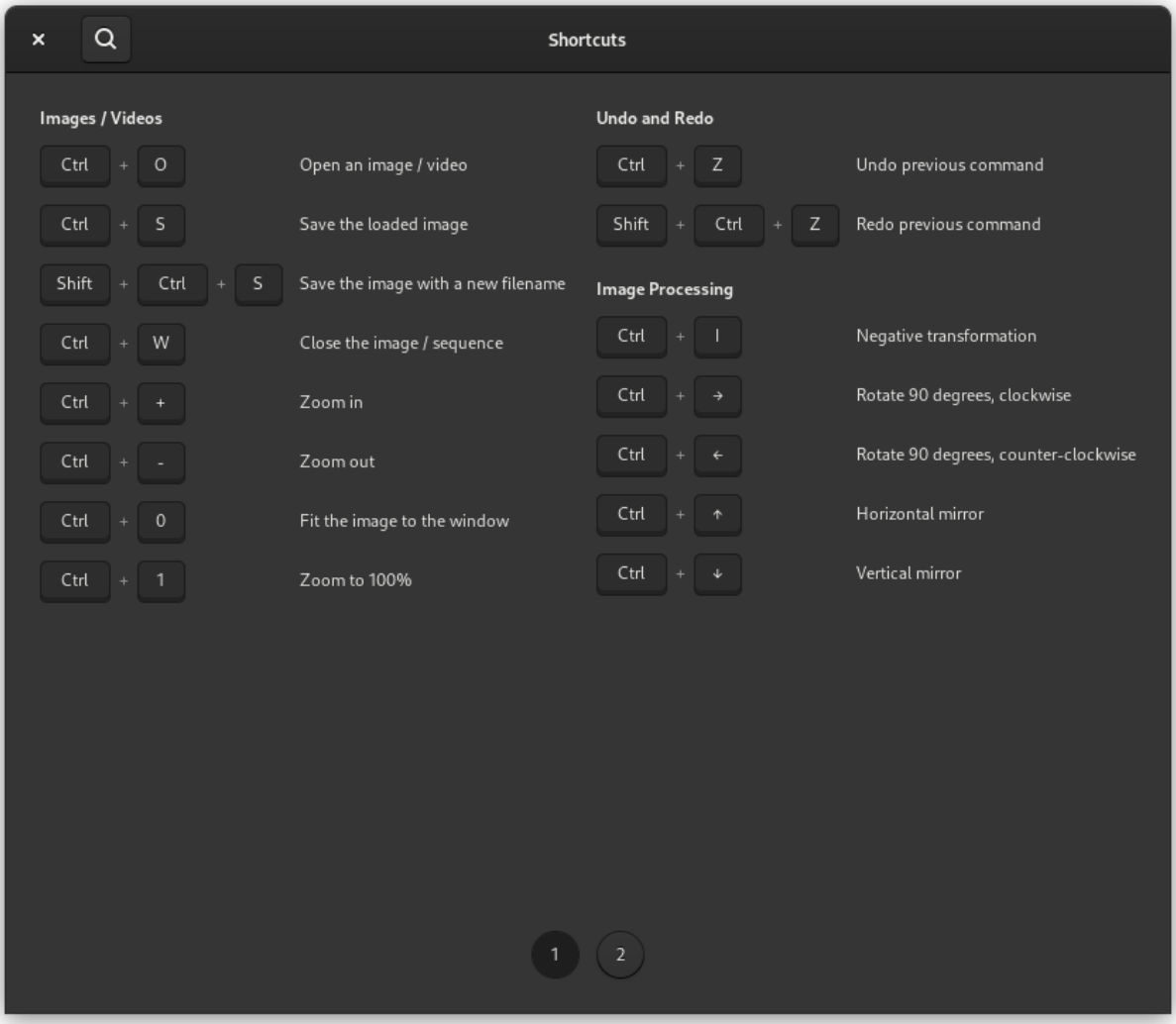
Default values are no binning (1x1), and a focal length and a pixel size stored in the settings. The values stored in the settings can be set from this dialog by activating the *Save as default values* button before clicking on *Close*. They can also be set by performing an astrometric resolution on the image, also called *plate solving*, if the option to update the default values when a result is found is enabled *in the preferences*.

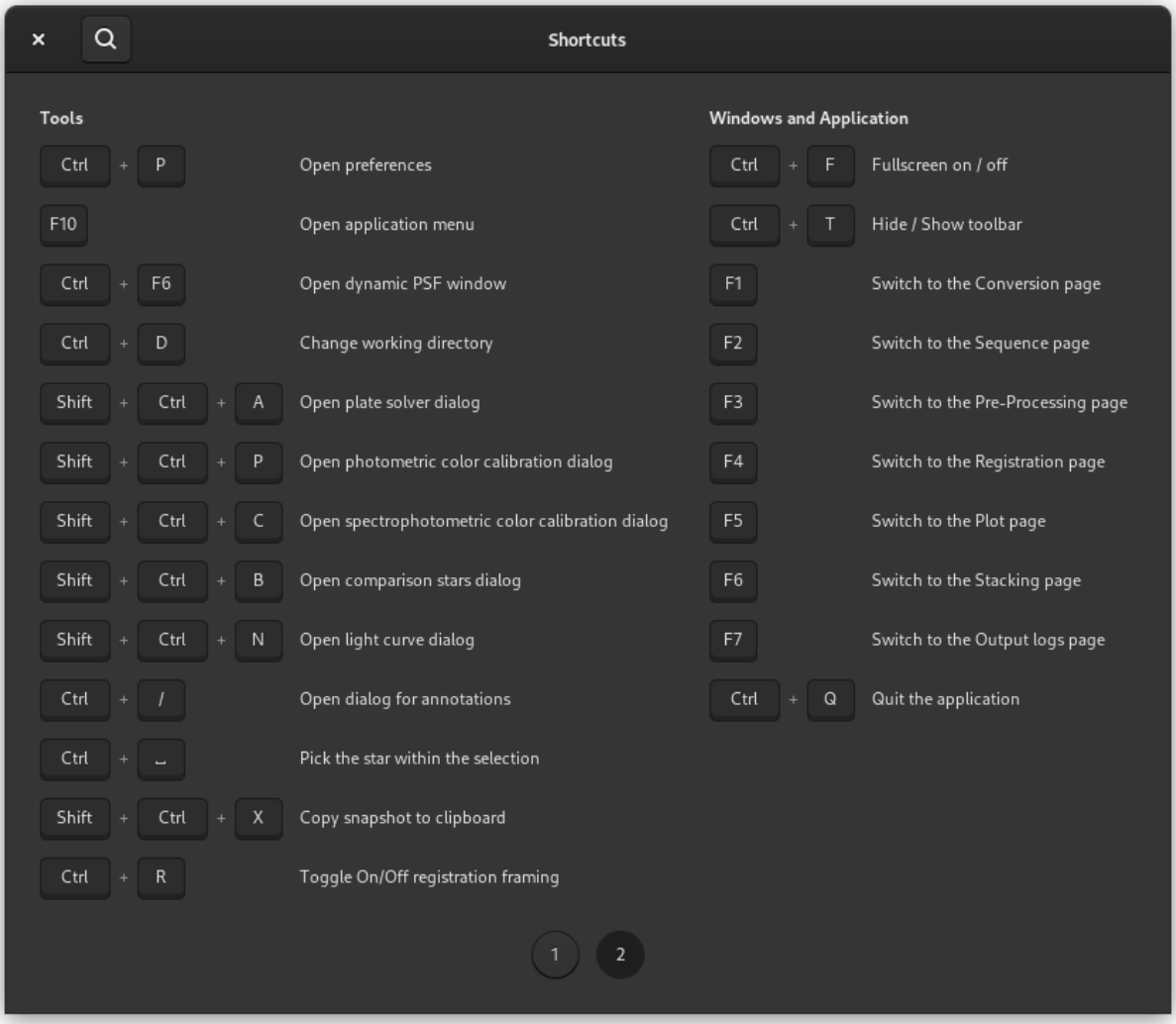
The values displayed in this window will be stored in the current loaded image and if this image is saved as FITS, they will be stored in the FITS header.

Binning management can take two forms depending on the acquisition software: the real pixel size is given but has to be multiplied by the binning (when *Real pixel size* is checked), the already multiplied pixel size is given (when unchecked).

## 2.8 Shortcuts

Siril uses several shortcuts to access processing tools or to manipulate the application and/or the images. These shortcuts are all detailed in the *Keyboard Shortcuts* dialog accessible via the burger menu ☰.


Shortcuts
Images / Videos
Ctrl + O
Open an image / video
Ctrl + S
Save the loaded image
Shift + Ctrl + S
Save the image with a new filename
Ctrl + W
Close the image / sequence
Ctrl + +
Zoom in
Ctrl + -
Zoom out
Ctrl + 0
Fit the image to the window
Ctrl + 1
Zoom to 100%
Undo and Redo
Ctrl + Z
Undo previous command
Shift + Ctrl + Z
Redo previous command
Image Processing
Ctrl + I
Negative transformation
Ctrl + →
Rotate 90 degrees, clockwise
Ctrl + ←
Rotate 90 degrees, counter-clockwise
Ctrl + ↑
Horizontal mirror
Ctrl + ↓
Vertical mirror
1
2


Shortcuts
Tools
Ctrl + P Open preferences
F10 Open application menu
Ctrl + F6 Open dynamic PSF window
Ctrl + D Change working directory
Shift + Ctrl + A Open plate solver dialog
Shift + Ctrl + P Open photometric color calibration dialog
Shift + Ctrl + C Open spectrophotometric color calibration dialog
Shift + Ctrl + B Open comparison stars dialog
Shift + Ctrl + N Open light curve dialog
Ctrl + / Open dialog for annotations
Ctrl + ␣ Pick the star within the selection
Shift + Ctrl + X Copy snapshot to clipboard
Ctrl + R Toggle On/Off registration framing
Windows and Application
Ctrl + F Fullscreen on / off
Ctrl + T Hide / Show toolbar
F1 Switch to the Conversion page
F2 Switch to the Sequence page
F3 Switch to the Pre-Processing page
F4 Switch to the Registration page
F5 Switch to the Plot page
F6 Switch to the Stacking page
F7 Switch to the Output logs page
Ctrl + Q Quit the application
1
2

CHAPTER

THREE

# WORKING DIRECTORY

The Working Directory (`wd`), also known as the Current Working Directory (`cwd`), is the directory in which Siril works. Its choice is a crucial step, especially when using scripts. The wrong choice of `cwd` is responsible for 90% of *script* failures. This folder is selected by clicking on the *Home* button, in the shape of a house: . This is the directory where Siril saves images by default (if no other path is specified) and also the directory where it searches for *sequences*.

Once the directory has been selected, its path can be easily checked in the title bar of the application window, below the version in use as illustrated in the figure below.

Fig. 1: Working Directory path shown in the title bar. Here, this is a Linux path.

CHAPTER

FOUR

# PREFERENCES

Preferences are settings which are persistent for every session of Siril, and which define your prefered choices for many of the tools.

Since version 1.2.0, they are accessible both from the *user interface* or programatically, by using *set/get commands*.

By default, the preference file is located at:

- `~/.config/siril/configX.Y.ini` (Linux)
- `%LOCALAPPADATA%\siril\configX.Y.ini` (Windows)
- `~/Library/Application Support/org.free-astro.Siril/siril/configX.Y.ini` (MacOS)

Where *X* and *Y* are major and minor release numbers.

If you wish to have multiple configuration files, you can choose which one to start from by opening a terminal and typing:

```
siril -i path/to/my_other_config.ini
```

> **Warning**
>
> Siril needs to be in your path to use **`siril`** as in the line above. Otherwise, use the full path to Siril binary.

## 4.1 Preferences (GUI)

Preferences are accessible from the Burger menu or with the shortcut `Ctrl` + `P`. There are 10 pages and each page representing a theme. The preferences allow more or less advanced users to optimize Siril to best suit their needs. Some settings can have a negative impact on Siril's performance, so it is advisable to change the settings only when you know what you are doing. There are three buttons at the bottom of the preferences dialog box. *Reset* restores all settings to the default value, *Cancel* cancels the current changes and *Apply* will close the dialog box and save the settings.

### 4.1.1 FITS/SER debayer

The **FITS/SER debayer** tab allows the user to define the debayer settings for FITS, SER or TIFF files. Consequently, this tab is only usable for a user with an OSC camera. It is advisable to leave the default settings as Siril will automatically define the correct settings to use. However, in the case of a TIFF file that is not an AstroTIFF, or a file that does not have all the required keywords, it may be necessary to adjust the settings manually.

- **Row-order preference**: You can choose to use header *ROW-ORDER* keywword if present and fallback to either Top-Down or Bottom-Up if not present. Or you can force a preferred orientation. More explanation on this keyword can be found *here*.

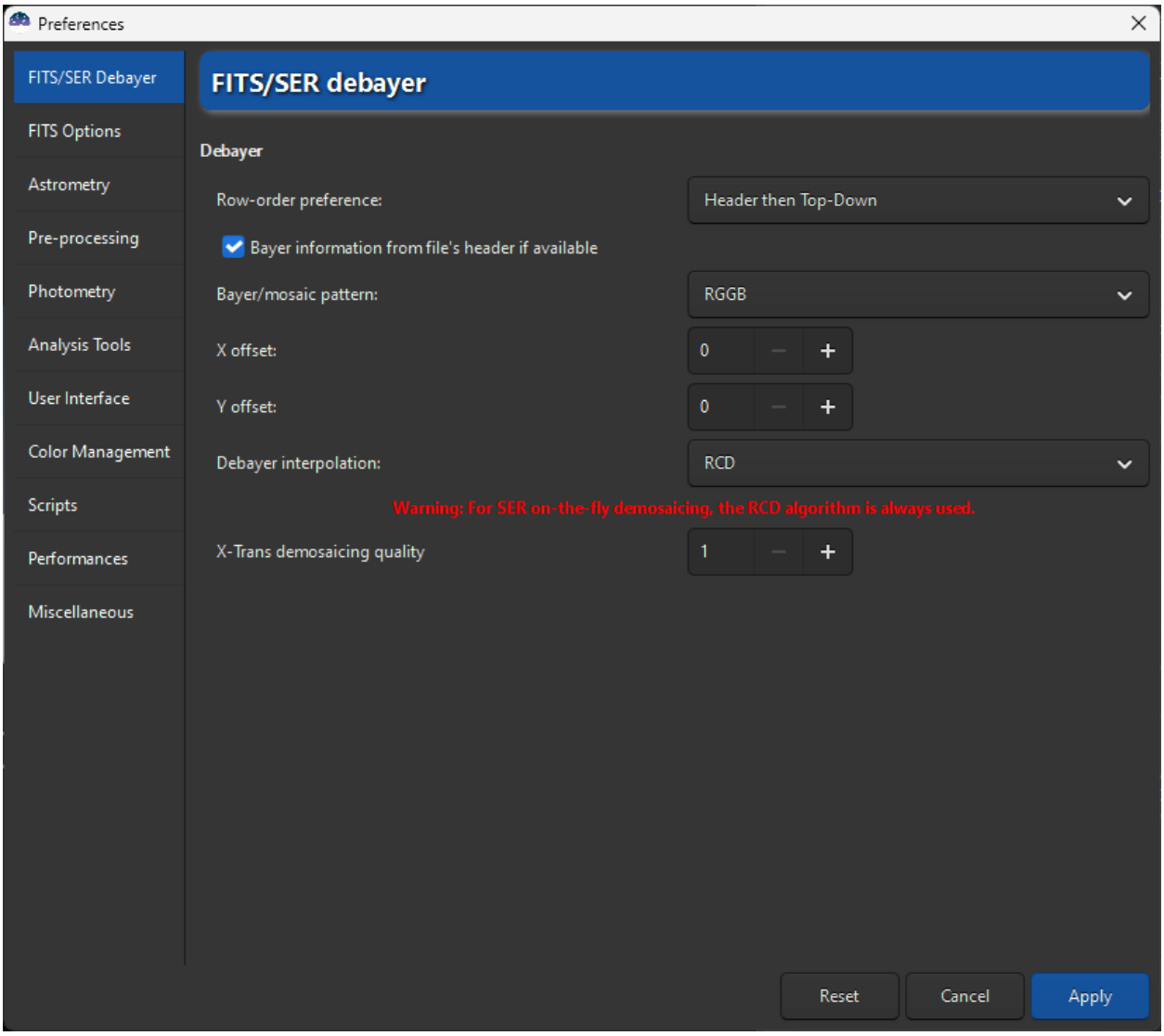



Fig. 1: Page 1 of preferences dialog

- **Bayer/mosaic pattern**: This drop-down menu allows you to choose the type of Bayer matrix used by the camera. It is generally indicated in the manufacturer's information. If *Bayer information from file's header if available* is ticked, the value specified in this dropdown will only be used in case the information is not present in the header. If the option is unticked, it is always used.
- **X offset**: In rare cases, files are recorded with a Bayer array shift. We can define an offset of 1 on the X axis, and an offset of 1 on the Y axis. Here the value defines if there is an offset in X. Note that this is used only when *Bayer information from file's header if available* is unticked or when the header does not contain this information.
- **Y offset**: Y offset of Bayer array.

Changing these settings will involve a different demosaicing each time. That's why it is strongly advised to leave the settings in their default state, unless you are really sure of what you are doing.

Another option that has less impact on the final result is the choice of the demosaicing algorithm proposed in **Debayer interpolation**. The choices are the following:

- **Fast Debayer** is the fastest algorithm available in Siril. However, other algorithms listed below are often quite better.
- **VNG4**, Threshold-Based Variable Number of Gradients, works on a 5x5 pixel neighborhood around each source pixel. It is a very good algorithm for flat areas of the image (like sky background) but produces artifacts in high contrast areas (like stars).
- **AHD**, Adaptive Homogeneity-Directed, is another well known debayer algorithm. However it usually shows artefacts in the background and bad star shapes.
- **AMaZE**, Aliasing Minimization and Zipper Elimination, is an algorithm that gives good results specially on low noise captures.
- **DCB**, Double Corrected Bilinear, a more recent algorithm, can show some artifacts in the background like AHD.
- **HPHD**, Heterogeneity-Projection Hard-Decision, is an old algorithm giving some nice results but that is quite slow.
- **IGV** and **LMMSE** are very good when working with very noisy images. However, IGV tends to lose some chromatic information, while LMMSE is one of the most computational expensive demosaicers and needs a lot of memory.
- **RCD**, Ratio Corrected Demosaicing, intends to smooth the colour correction errors that are common in many other interpolation methods. It does an excellent job for round edges, for example stars, and is therefore the default algorithm used in Siril.

For the X-Trans sensor, a special algorithm called **Markesteijn** is used regardless of the method selected in the preferences. For the latter, it is possible to define the requested quality with the **X-Trans demosaicing quality** option. It defines the number of passes for the X-Trans Markesteijn demosaicing algorithm, 1 is the default, 3 may be slightly better but slower.

> **Warning**
>
> For on-the-fly demosaicing of **SER** files, the RCD algorithm is always used regardless of the choice made in the drop-down menu. This allows Siril to be more efficient in terms of execution speed and to offer a good quality.

### 4.1.2 FITS Options

The **FITS Options** page groups all the settings related to the native format used by Siril.

- **FITS extension**: By default, the value is set to **.fit** However, many capture programs use the **.fits** extension. In this case we advise you to update the value. All files created by Siril will have the extension defined here.

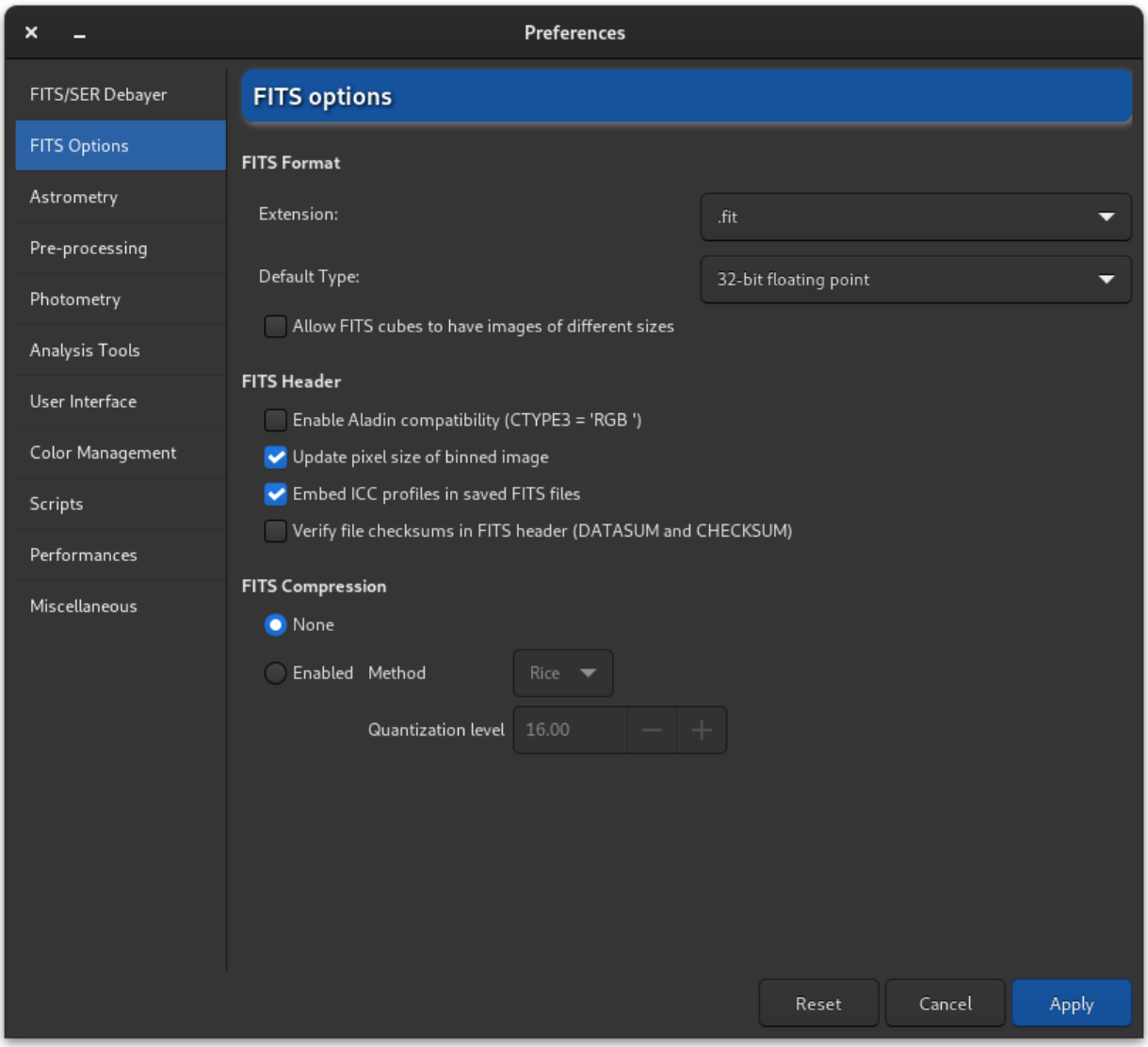



Fig. 2: Page 2 of preferences dialog

Moreover, only sequences with the extension defined in the preferences can be loaded. It is therefore not possible to open a **.fits** sequence and a **.fit** sequence without updating this value. Supported extensions are:

- **.fit**
- **.fits**
- **.fts**

All of them can be appended by the **.fz** extension if the files are compressed.

**Siril command line**

```
setext extension
```

Sets the extension used and recognized by sequences.

The argument **extension** can be "fit", "fts" or "fits"

- **Default Type**: By default, Siril works in 32-bit floats in the range [0, 1]. This is the best way to keep a high precision. However, for hard disk space considerations a user may decide to work in 16-bit unsigned (in the range [0, 65535]). Be careful though, a 16-bit stacking can lose a lot of information.
- **Allow FITS cubes to have images of different sizes**: This can be useful to open scientific FITS files that were not created by Siril and that contain multiple images of different dimensions, which would otherwise considered as invalid Siril FITSEQ files. The JWST images are a good example of the use of this option. An example is given *here*.
- **Enable Aladin compatibility (CTYPE3 = 'RGB ')**: Aladin considers a 3D FITS cube as a RGB image (Red, Blue and Green components) if the FITS keyword `CTYPE3 = 'RGB '` is specified in the header. In this case any `BITPIX` value are supported. Without the FITS keyword `CTYPE3 = 'RGB '` set, only FITS cube with 3 frames sharing the same dimension and with a `BITPIX=8` will be automatically detected as RGB FITS.

**Warning**

This option may conflict with the astrometry feature and should only be activated if it is really necessary.

- **Update pixel size of binned images**: Used for image sampling computation, pixel size can be given in two different ways: the real pixel size is given but has to be multiplied by the binning (when checked), the already multiplied pixel size is given (when unchecked). It depends on the acquistion software used to create the FITS.
- **Embed ICC profiles in saved FITS files**: This should generally be left on, unless you need interoperability with a program that fails to handle FITS with embedded ICC profiles (none are currently known).
- **Verify file checksums in FITS header (DATASUM and CHECKSUM)**: Verify file checksums if they exist. The verification occurs each time the file is read, which may significantly impact Siril's performance when using a sequence.
- **FITS compression**: La compression peut être intéressante dans certains cas où l'espace sur le disque dur est un point clé du traitement. You can find more information in the section dedicated to the FITS format, *here*.

  The compression adds the extension `.fz` to the files created. Siril is able to open a sequence with the fz extension without having to change any value in the preferences.

**Siril command line**

```
setcompress 0/1 [-type=] [q]
```

Defines if images are compressed or not.

**0** means no compression while **1** enables compression.
If compression is enabled, the type must be explicitly written in the option **-type=** ("rice", "gzip1", "gzip2").
Associated to the compression, the quantization value must be within [0, 256] range.

For example, "setcompress 1 -type=rice 16" sets the rice compression with a quantization of 16

### 4.1.3 Astrometry

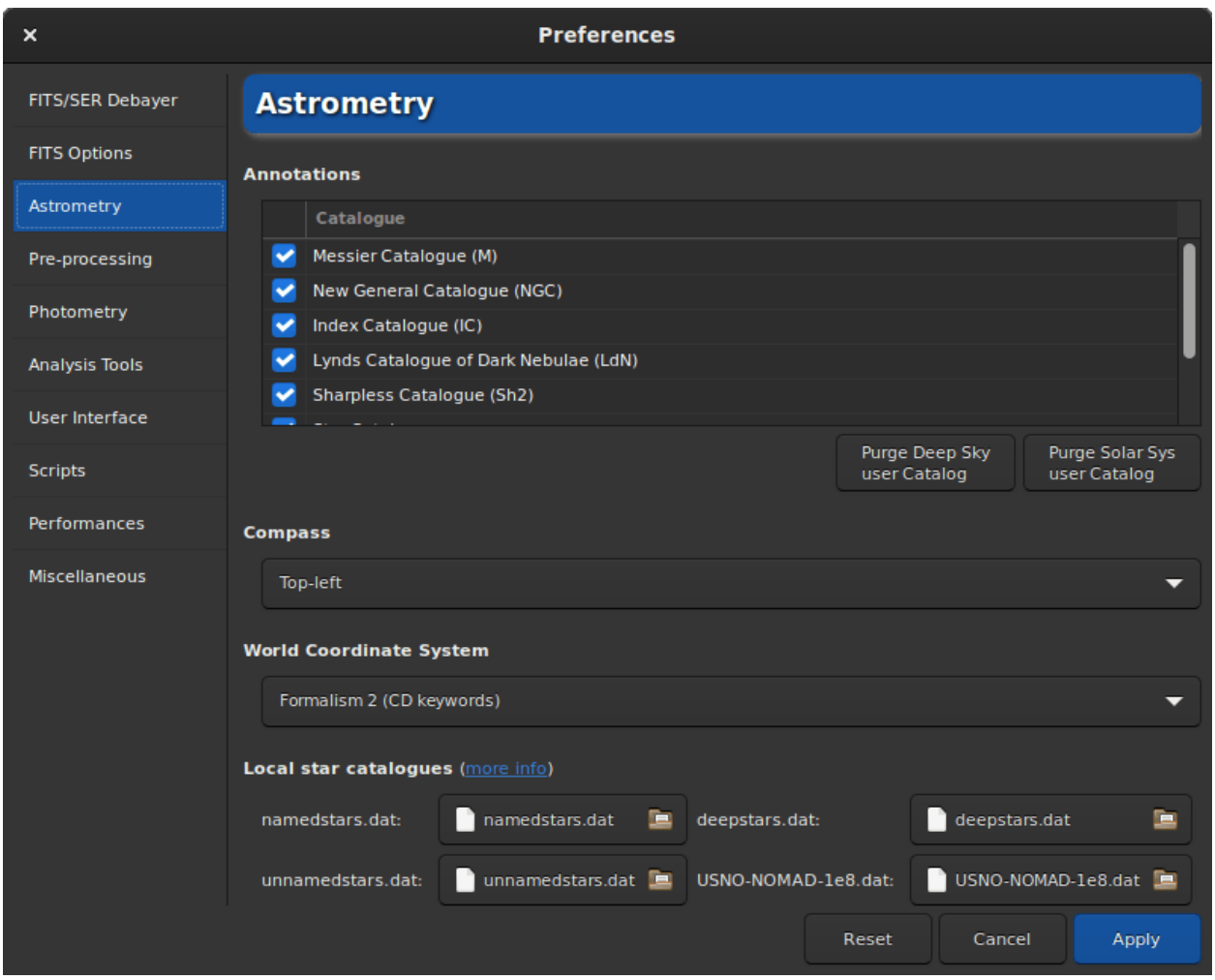



Fig. 3: Page 3 of preferences dialog

> **Tip**
>
> Don't forget that you can scroll the window as there are options at the bottom.

This tab contains all the options related to astrometry. Astrometry is a feature strongly implemented in Siril. When the image is solved (i.e. when the astrometry has been successful), it is possible to display the names of the known objects. In particular those listed in the large astronomical catalogs. The annotation part allows to define which catalogs can be used for the display of object names. Currently there are 8 of them, and they can be deselected to be ignored:

- Messier catalog
- New General Catalogue
- Index Catalogue
- Lynds' Catalogue of Bright Nebulae
- Sharpless Catalogue
- Catalogue of Brightest stars
- IAU constellations lines
- IAU constellations names (positions collected from this page)

In addition to this list, there are two more catalogs that are filled in by the user. One concerning the deep sky objects, the other concerning the solar system. They can be better described in the *annotations* section of this documentation.

Underneath the list of astronomical catalogues is an option to set a preferred IAU Observatory Code. This is a 3-character code that can be used to specify a nearby IAU observatory to improve location accuracy when running *conesearch -cat=solsys*. If the *-obscode=* option is not provided, this default is used in preference to the geocentric default which is used otherwise. The option text provides a hyperlink to the Minor Planets Center website which maintains an up-to-date list of the IAU observatory codes, so that you can easily find the closest code to your location.

By clicking on the button *Show object names* (only if the image has been plate-solved), the annotations are displayed on the image. It is also possible to click on the button that displays the celestial grid . The latter, by default, adds a compass to the center of the image. The **Compass** section allows you to define the desired location for the display of the compass.

The **World Coordinate System** section allows you to choose

- **Formalism 1**: In the `PC i_j` formalism, the matrix elements $m_{ij}$ (linear transformation matrix) are encoded in `PC i_j` (floating-valued) header cards, and si as `CDELT i`. The *i* and *j* indices are used without leading zeroes, e.g. `PC 1_1` and `CDELT 1`. The default values for `PC i_j` are 1.0 for $i = j$ and 0.0 otherwise. The `PC i_j` matrix must not be singular; it must have an inverse. Furthermore, all `CDELT i` must be non-zero.
- **Formalism 2**: The `CD i_j` (floating-valued) keywords encode the product $s_i m_{ij}$. The *i* and *j* indices are used without leading zeroes, e.g. `CD 1_1`. The `CD i_j` matrix must not be singular; it must have an inverse. `CDELT i` and `CROTA i` are allowed to coexist with `CD i_j` as an aid to old FITS interpreters, but are to be ignored by new readers.

The **Local star catalogues** part of the dialog window concerns the use of local catalogs to platesolve images. This feature is described in details in the *local star catalogues* section of this documentation.

Under the **General Platesolving** section, a number of options are available to control both Siril internal solver and astrometry.net.

- the first option determines if the computed focal length and input pixel size are stored in the settings as default values for images that don't have the corresponding metadata, when an astrometric solution is found.

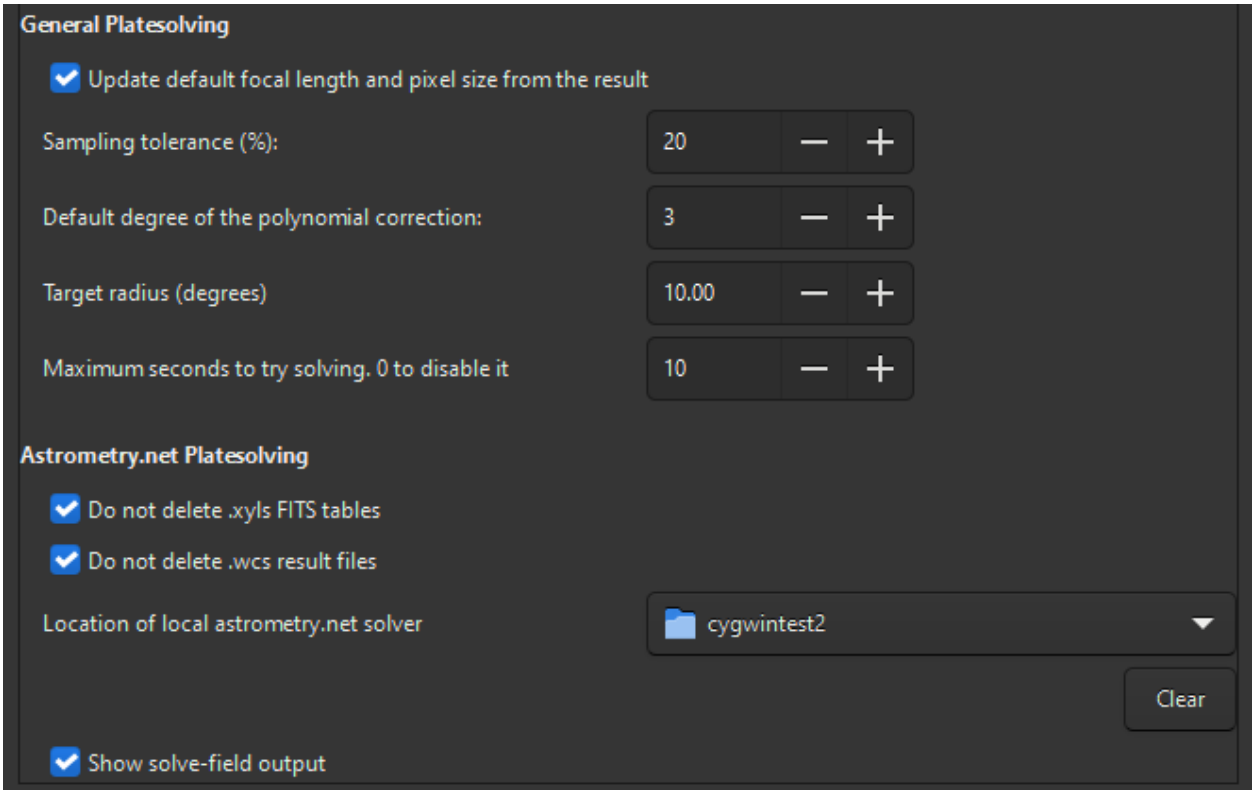



Fig. 4: Bottom of page 3 of preferences dialog

- **Sampling tolerance**: percent below and above the expected sampling to allow. Given sampling is multiplied or divided by 1 + this / 100.
- **Degrees of the polynomial correction**: solvers can use a polynomial correction (SIP) to work with optical aberrations, this is the order of the polynomial model. 0 disables it.
- **Target radius**: allowed search radius around the target coordinates for the solve (degrees). Unused for blind (no target passed) solves.
- **Maximum seconds to try solving**: allowed time for the solve for each catalog file. It can be used as the total solve time only if solve-field is configured to do so in its configuration file.

The last section is dedicated to the solve-field plate solver from the astrometry.net suite.

- **Do not delete .xyls**: the list of stars is passed to solve-field as a FITS table, check this to keep the file in the working directory.
- **Do not delete .wcs**: the results from solve-field are stored in a FITS header with a file name ending in .wcs. Check this to not remove this file.
- **Location of local astrometry.net solver**: In order to use Astrometry.net locally in Siril, it can be necessary to tell to Siril the path where it is located. On UNIX-based systems, it is generally in the `PATH` variable and not necessary. For Windows, if you did not modify the default installation directory, that is *%LOCALAPPDATA%\cygwin_ansvr*, Siril will search for it without additional setup. If you have cygwin and have build astrometry.net from the sources, you must specify the location of cygwin root here.
- **Show solve-field output**: print the output of the solver in Siril's main log window, otherwise, only the outcome will be given.

### 4.1.4 Pre-processing

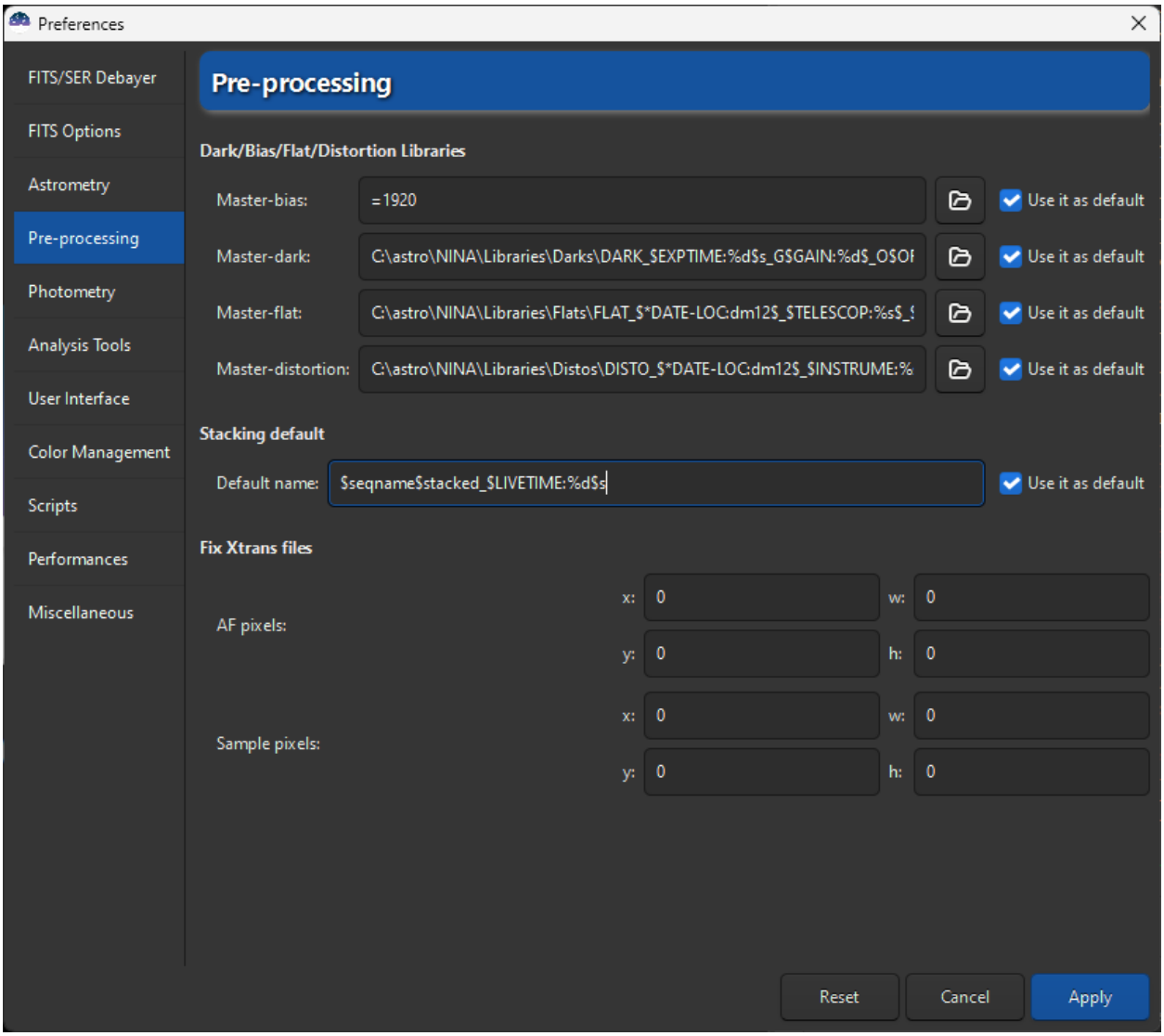



Fig. 5: Page 4 of preferences dialog

The pre-processing tab contains all the elements related to the steps that are executed until the stacking. Here it is possible to manage a library of offsets, darks, flats and distortions, the output name of the stacked file or specific corrections for cameras that use the X-Trans sensor.

- **Dark/Bias/Flat/distortion Libraries**: In this section it is possible to load an offset, a dark and a flat that will be used by default in the pre-processing if the button to the right of the text box, *Use it as default* is checked. Each path will also be stored in the reserved keywords `$defbias`, `$defdark`, `$defflat` and `$defdisto` (one token `$`) which can be used when saving a stacking/platesolving result. As far as bias is concerned, it is possible to use more than just a file path. Indeed, in the Siril team we encourage users to use synthetic bias as explained in this tutorial. Several values are then possible as long as the first character entered is the = sign. It is possible to use a fixed integer value like `=500` or a multiplication involving the keyword `$OFFSET` (one token `$`) as long as the latter is actually registered in the FITS file header, like `10*$OFFSET`. More details are given in the tutorial.
- **Stacking default**: Here we define the default name that we want to give to the stacking results. It is possible to use any value given in the FITS header as a keyword and surround it with `$` tokens. If the key-

word does not exist the variable will be used, otherwise it is its value. Another reserved keyword that can be used is `$seqname$`. It contains the name of the loaded sequence. For example, the following default name, `$seqname$stacked_$LIVETIME:%d$s` with a sequence name `r_pp_light_` and the following header:

```
...
DATE    = '2022-12-08T22:21:14' / UTC date that FITS file was created
DATE-OBS= '2015-08-21T22:18:25' / YYYY-MM-DDThh:mm:ss observation start, UT
STACKCNT=                   13 / Stack frames
EXPTIME =                 300. / Exposure time [s]
LIVETIME=                3900. / Exposure time after deadtime correction
EXPSTART=     2457256.42945602 / Exposure start time (standard Julian date)
EXPEND  =     2457256.51666667 / Exposure end time (standard Julian date)
...
```

will output `r_pp_light_stacked_3900s.fit`.

**Note**

Both library paths and stacking default support *path parsing*.

- **Fix Xtrans files**: This setting field is very specific and only concerns possessors of certain X-Trans sensors. Indeed, some images from these cameras show a large square in the center of the darks and bias images due to the position of the autofocus (AF). Siril has an algorithm to eliminate it for the following cameras:
  - Fujifilm X-T1
  - Fujifilm X-T2
  - Fujifilm X-T20
  - Fujifilm X-Pro2
  - Fujifilm X-E3
  - Fujifilm X-H1

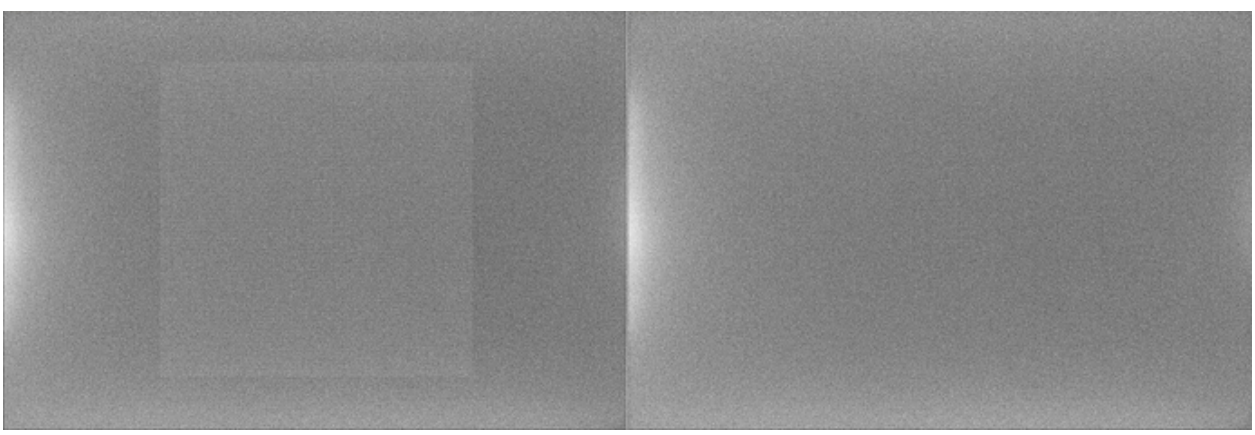


Fig. 6: X-Trans artifact fixed by the algorithm of Siril

In the unlikely event that your camera contains this artifact and is not supported, then it is possible to define the correction to be applied here. The best thing to do is to contact the dev team in order to have the values to enter that would correspond to your camera.

### 4.1.5 Photometry

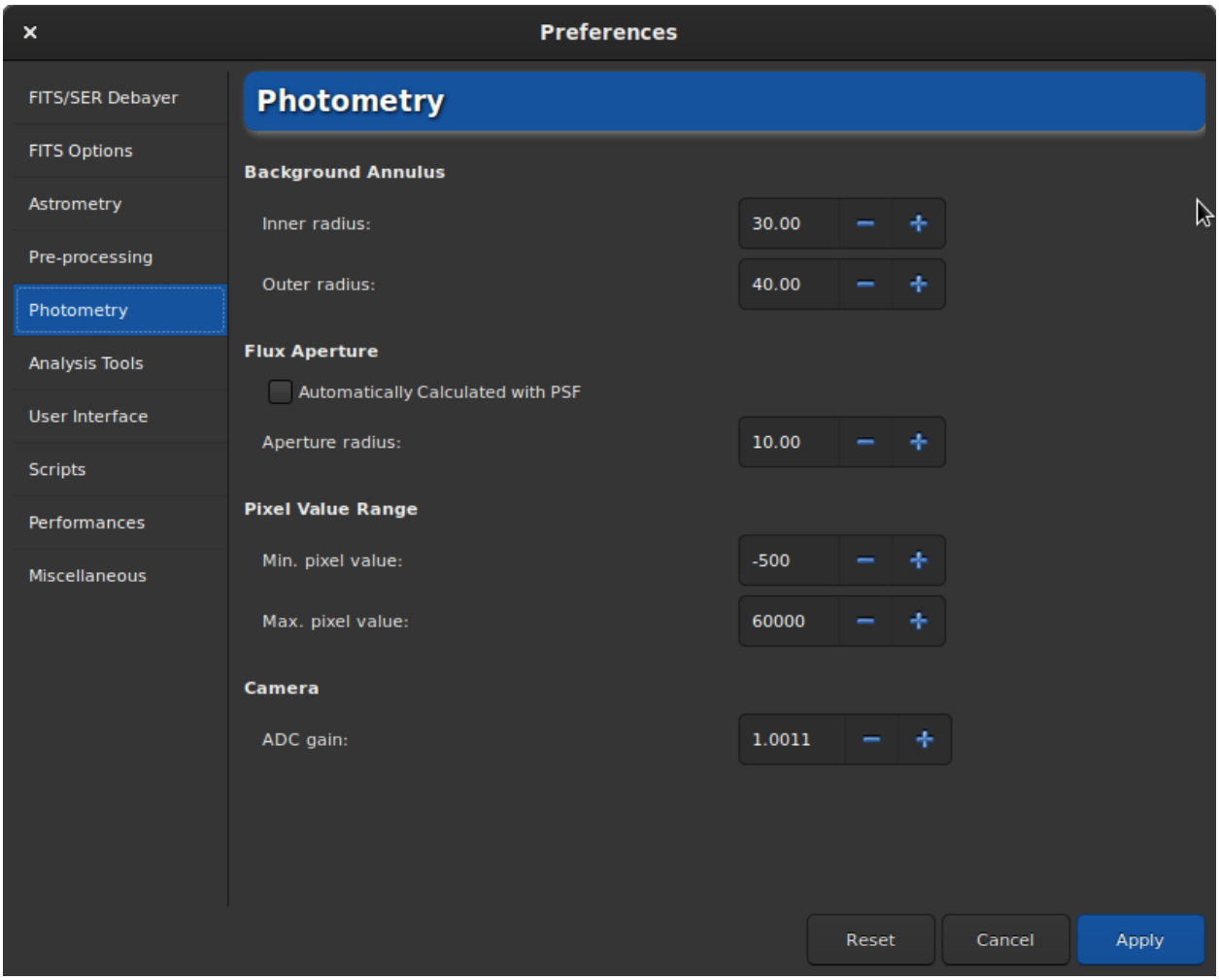



Fig. 7: Page 5 of preferences dialog

Photometry, which is the study of light, is another feature very present in Siril. This section of the preferences allows you to define the settings associated with this tool.

The basic principle of aperture photometry is to sum-up the observed flux in a given radius from the center of an object, then subtract the total contribution of the sky background in the same region (calculated in the ring between the inner and outer radius), leaving only the flux of the object to calculate an instrumental magnitude. This is described in more detail in the *Photometry* section of this documentation.

- It is then possible to modify the **inner radius** and the **outer radius** to define a size that optimizes the calculated sky value, trying to avoid the stars inside the ring. Outer **must** always be **greater** than inner. By default, the **flux aperture** radius is set as twice the PSF's FWHM, however it is possible to disable this feature and define a fixed value manually.
- **Pixel range value** allows users to set a limit for which the pixel is considered bad for photometry. Indeed, doing photometry on saturated data will never give good results, but even getting close to high values may not be suitable because it may be in the non-linear regime of sensors. A default value of 50000 ADU is set to avoid this region, but it may vary from sensor to sensor. Negative values are also allowed because noise can average around a positive value but still provide a few pixels with negative values.
- Finally, if known, it is highly recommended to put the value of the A/D **converter gain** in electrons per ADU: it

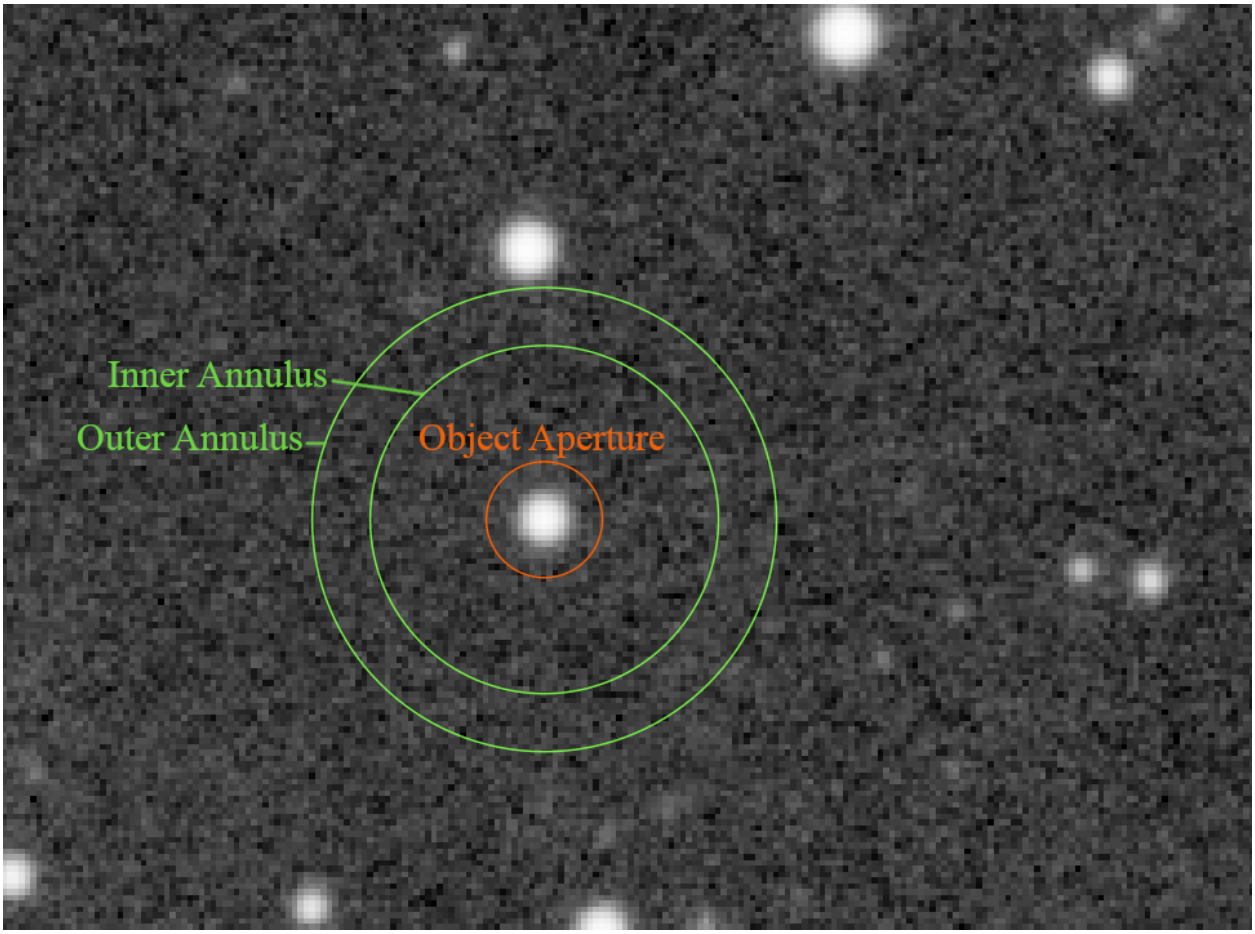



Fig. 8: Circle of the aperture photometry

is used in the uncertainties computations, if not already provided in the headers of the processed images.

### 4.1.6 Analysis Tools

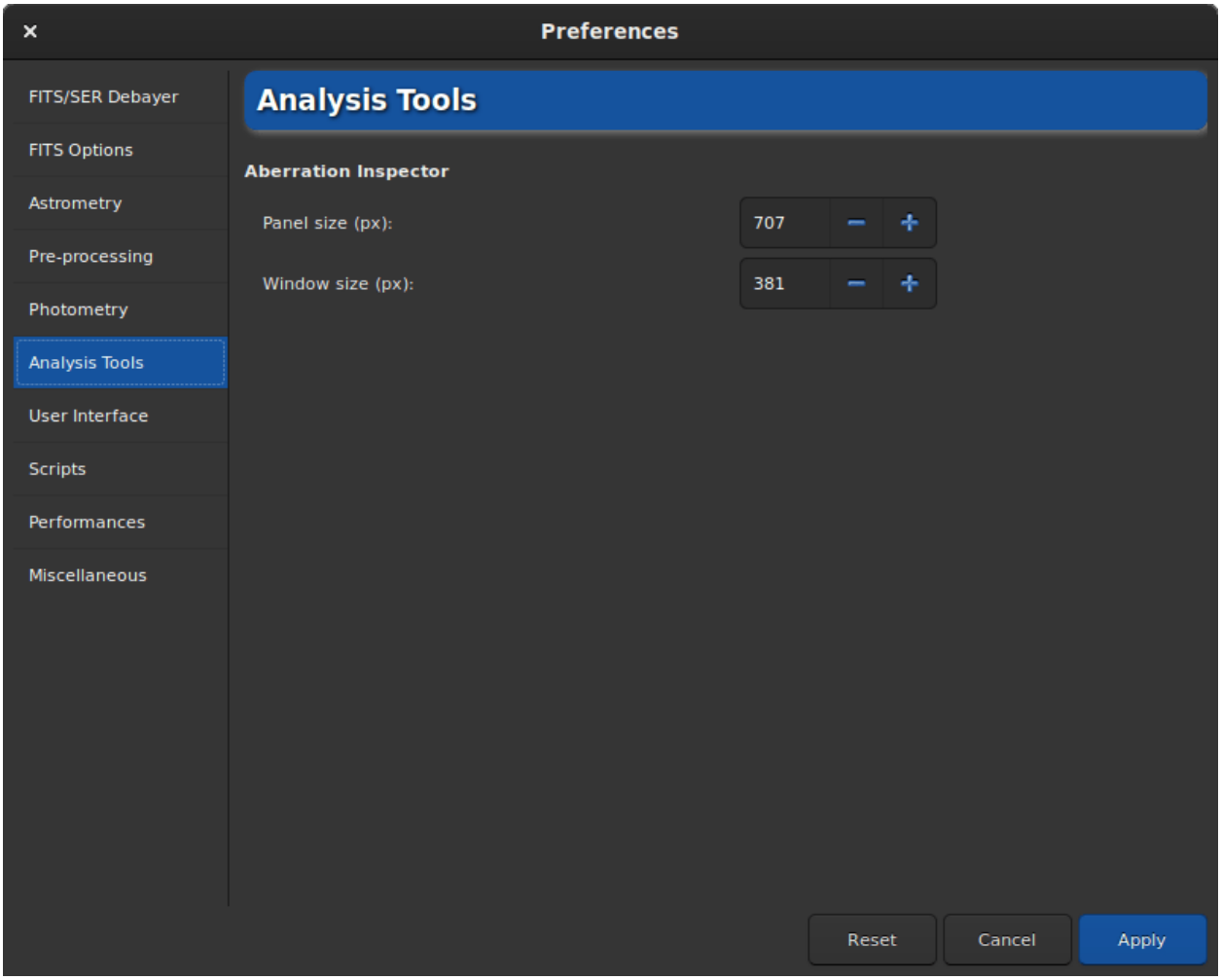



Fig. 9: Page 6 of preferences dialog

So far, only one image analysis tool requires adjustment parameters. It is the aberration inspector tool. In this tab you can adjust:

- The **panel size**, in pixels, which defines the size of the image that will be placed in a panel. The larger the value, the larger the size of the image in a panel. A value that is too high may prevent from seeing the defects of the stars.
- The **window size**, also in pixels, which defines the size of the dialog. It is usually a good idea to increase this value when using a 4K screen.

### 4.1.7 User Interface

In this tab are listed all the adjustments related to the user interface. These are not settings that have an impact on the processes, but on the feel and look and needs of the user.

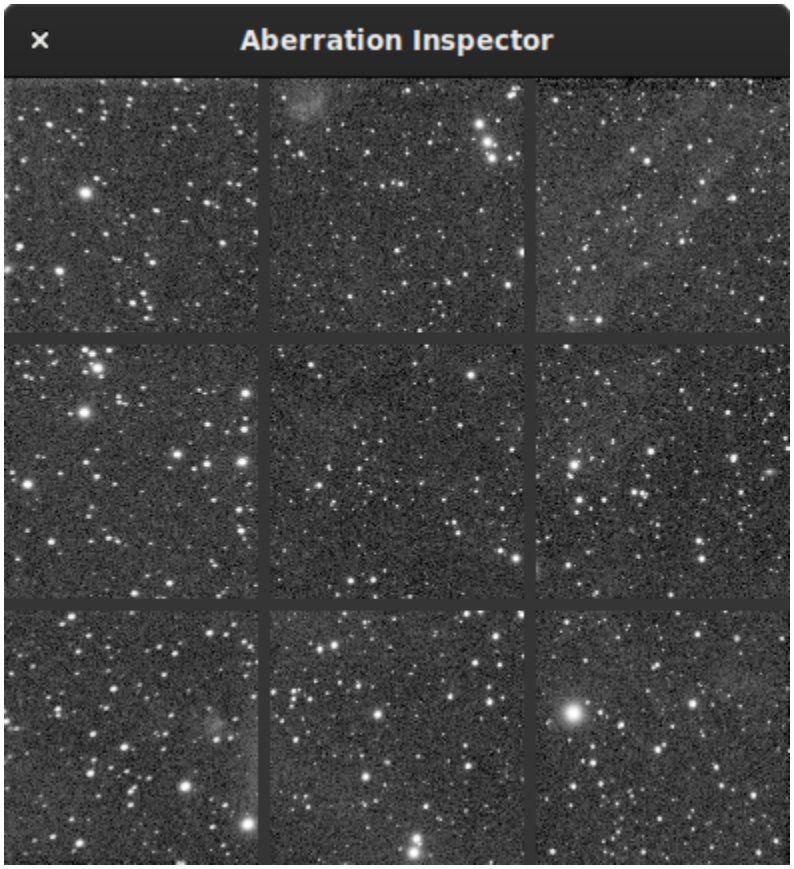



Fig. 10: Aberration inspector window

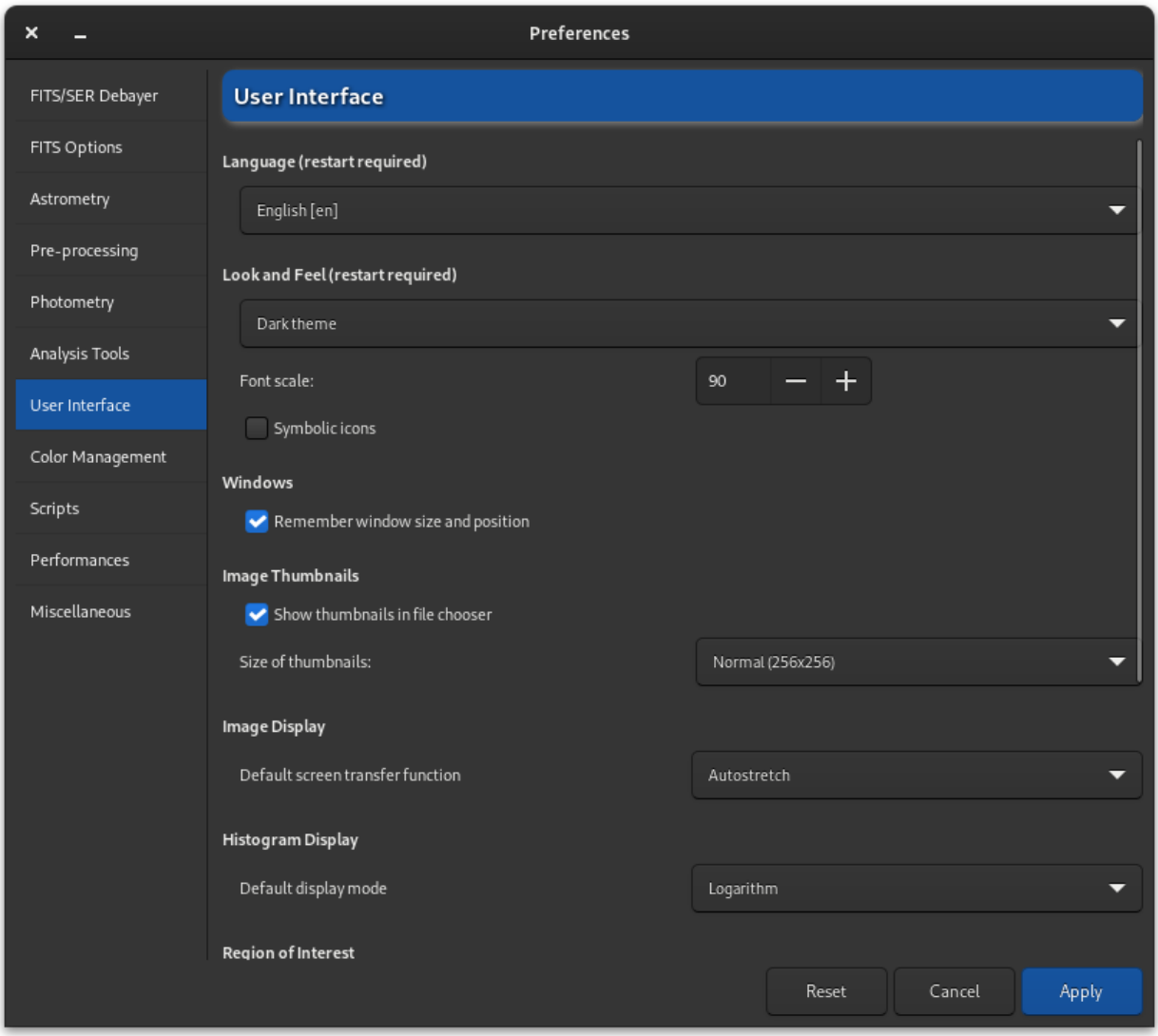



Fig. 11: Page 7 of preferences dialog

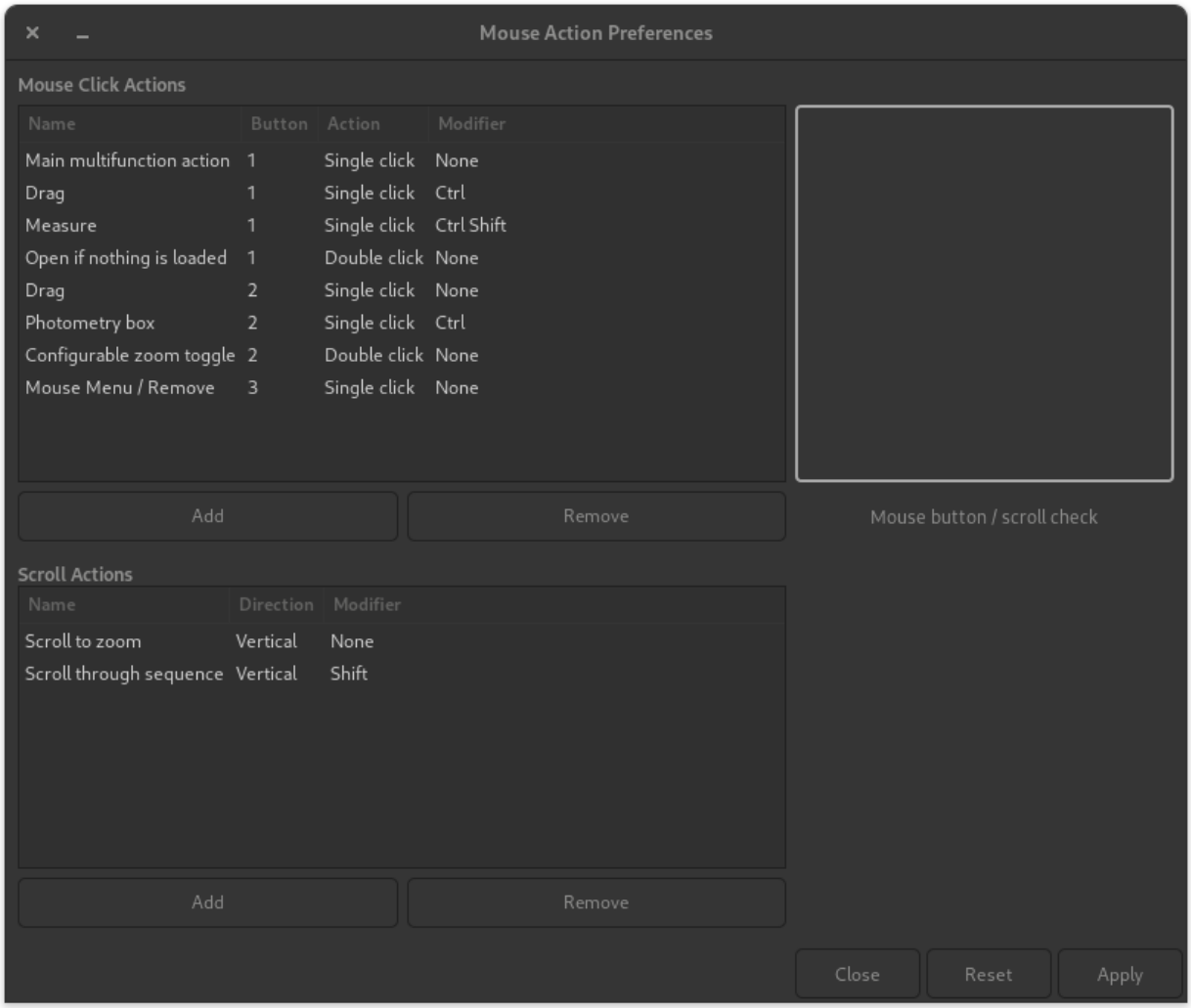
> **Tip**
>
> Don't forget that you can scroll the window as there are options at the bottom.

- By default, the **language** of Siril is defined according to the system language. However it is possible to change the language and define it to your needs, as long as it exists. However, keep in mind that Siril is developed in English.
- Two themes are available:
  - The dark theme (default theme)
  - The light theme

  Changing the theme requires a restart of the application to be fully operational.
- It is possible to adjust the **font scale** for users with a 4K Ultra-HD screen, or to use **symbolic icons** for some icons. Theses settings also require a restart of the application.
- By default Siril remembers the size and the position of the application window each time you close it. By checking on the *Remember window size and position* button you can disable this behavior.
- The thumbnails of the images are usually visible in the open dialog boxes. The preferences allow you to not display them if the computer has limited performance and the user does not see the need. You can also change the size of the thumbnail display with the drop-down list.
- **Default screen transfer function** is the setting that allows images to be displayed according to the user's preference. By default, this is set to *Linear*. As this really represents what the image is, it is recommended that beginners leave this setting at default. It is easy to forget that you are in auto-adjusted viewing and not understand why the saved images are not as they appear on the screen. However you can always adjust the visualization in the main window.
- According to the same principle, the histograms can be displayed in two modes with **Default display mode**. Either the *Linear* or the *Logarithmic* mode. The latter can be very useful with the Generalized Hyperbolic Stretch tool. You can however change the mode in each window with a histogram. In the preferences, this is a matter of setting the default behavior.
- **Region of Interest** is the setting that allows to chose the type of selection you want to apply to the ROI: *Manual* or *Automatic from selection*. These options are described in details in the dedicated section *Setting the ROI*.
- **Mouse Behaviour** provides configuration of mouse behaviour. Since 1.3.0 this allows configuring the behaviour of double middle-click to set a preset zoom level. The preset zoom level can be set either to zoom to fit, zoom to 100% centered on the mouse cursor, or to toggle between zoom to fit and zoom to 100% centered on the mouse cursor. Together with the other improvements to pan and zoom control using the middle mouse button, this improves one-handed operability of Siril.
- The **Mouse Configuration** button opens the mouse configuration dialog. This allows you to completely customise your mouse behaviour for Siril by changing default mouse behaviour or adding or removing mouse functions. This can be particularly useful for disabled users to remap existing mouse functions to other buttons or so that they don't need modifier keys, and it can be useful for any user with a mouse with lots of buttons (or two-way scroll wheels) to assign new actions (such as save, undo, scroll through the active sequence) to mouse buttons or scroll axes in a way that has not previously been possible, or with two-way scroll wheels.

  You need to click through a reminder that this tool can remove or change mouse behaviour so that your settings no longer match the documentation. The tool has a certain amount of sanity checking built in (it will reject duplicate assignments, it will insist that the main mouse action used for interacting with the image is included, and it will check for clashes between single click and double click actions assigned to the same button and modifier. If you really mess up your configuration there is a reset button that will reset all mouse behaviour to the defaults.

Mouse button numbering is not always obvious, especially with mice with many buttons, as the scroll wheel directions are accounted for in button numbering, so a scratchpad area is provided to the right of the dialog. If you click any mouse button here, its number will be shown in the label. This is the same number you need to use when assigning functions to that mouse button.

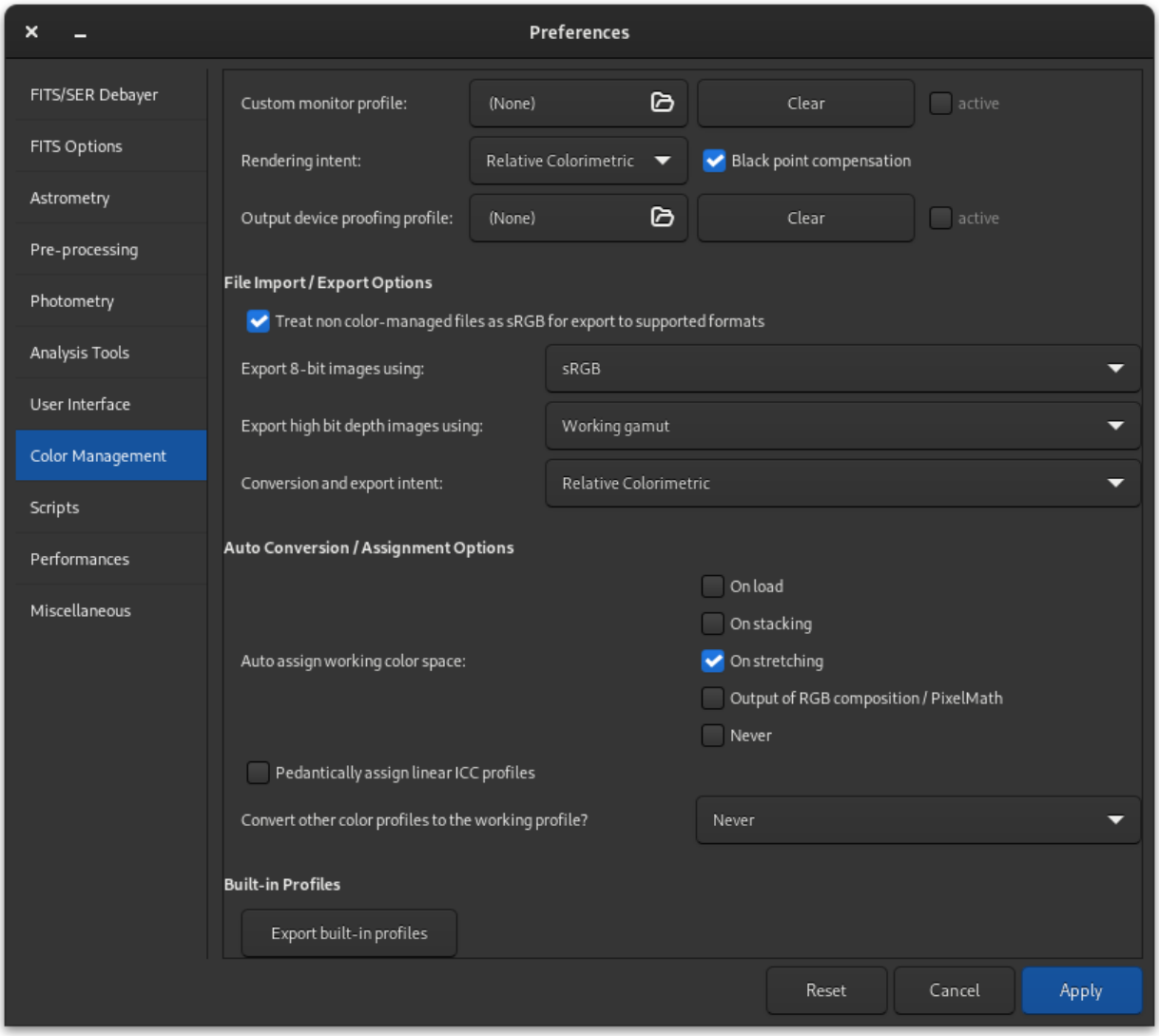



> **Tip**
>
> Note that this configuration applies only to mouse events on the main drawing area (where the image is displayed). Regardless of any mouse actions configured here, the mouse behaviour will still be the same elsewhere in the application. So for example if you want to universally swap left and right mouse buttons, you can't do that here, but you may be able to do so in your Operating System settings or mouse driver config.

- You can configure some texts and drawings colors to be displayed on the image in the **Colors** section. To do so, simply click on the color button and choose the color of your choice. This choice concerns 5 items:
  - Background extraction samples
  - Standard annotations
  - Deep Sky Object annotations

- Solar System Object annotations
- Temporary Object annotations

### 4.1.8 Color Management




#### Working Color Spaces

#### Preferred Color Space

This option selects the set of working color spaces, namely a linear RGB working space, a nonlinear RGB working space with the color space's standard tone response curve (TRC), and a Gray working space with a TRC matched to the RGB working space. The options are sRGB, which is most suitable for working with images for WWW use; Rec2020, which is Siril's recommended wide gamut color space; and Custom, which allows you to define your own set of profiles using the file choosers.

### External Color Profiles

These filechoosers are only of use to gether with the *From Files* option selected above for your Preferred Color Space.

- **RGB ICC profile**: This file chooser lets you pick the RGB ICC profile for your preferred color space.
- **Gray ICC profile**: This file chooser lets you choose a Gray ICC profile. The profile should have a tone response curve (TRC) matched to that of your chosen RGB color space.

An example of how to set a custom RGB gamut corresponding to Pro Photo RGB is available on the *profiles page*.

### Gamut Visualisation

This button opens a reference image comparing the gamuts offered by the built-in sRGB and Rec2020 color spaces with the gamuts of some other well-known RGB color spaces.

### Custom Profiles

- **Custom monitor profile**: This option allows you to choose a custom monitor profile, for example if you have calibrated your monitor or are using a wide gamut monitor. There is also a button to clear the current selection, and a check box to set whether or not the custom monitor profile is active. If it is not active, the display transform will target a standard sRGB display.
- **Rendering intent**: This option allows you to choose your preferred ICC rendering intent, which is used for the color transform from the working space to the monitor profile. A description of the available intents is available *on the Theory page*. A check button is also provided to set whether or not to use Black Point Compensation.
- **Output device proofing profile**: This option allows you to choose a soft proofing profile for an output device. If no profile is set here, soft proofing will target the display profile. There is also a button to clear the current selection and a check box to set whether or not the soft proofing profile is enabled. If it is not enabled, the soft proofing display mode will not be available.

### File Import / Export Options

- **Treat non color-managed files as sRGB for export to supported formats**: This option is for images that do not have a color profile assigned. If the option is checked, when an image without an ICC profile is exported to a format that supports ICC profiles, it will be saved with a sRGB ICC profile or a Gray profile with a TRC that matches the standard sRGB TRC. This is essentially the behaviour of previous versions of Siril. It will often work OK, but does not represent a properly color managed workflow. The option is on by default, but it is better to avoid the issue by using the auto conversion / assignment options to ensure that images will have an ICC profile assigned and be properly color managed at the time you wish to save them.
- **Export 8-bit images using**: This option sets the color space to export 8-bit images that support embedding ICC profiles: the choice is between sRGB and your working color space as chosen with the Working RGB Gamut option above.
- **Export high bit depth images using**: This option sets the color space to export 16-bit images that support embedding ICC profiles: the choice is between sRGB and your working color space as chosen with the Preferred Color Space option above.
- **Export intent**: this sets the intent for transforming images from their assigned color space to the color space they are to be saved in. Leave this set to Relative Colorimetric to ensure your saved images look consistent with how you saved them. Other intents should probably only ever be chosen here for debugging purposes and similar.

One preference is in the FITS Options tab:

- **Embed ICC profiles in FITS images and FITSEQ sequences**: This option enables ICC profile embedding in FITS files: the profiles are embedded within an extra HDU inside the FITS file, with the name "ICCProfile", which matches the embedding method used by other major astrophotography software. *It is strongly recommended to leave this option enabled.*

> **Tip**
>
> Export of images to 8-bit and 16-bit formats that do not support embedded ICC profiles will always be in the sRGB color space.

### Auto Conversion / Assignment Options

#### Auto assign working color space

This set of check boxes sets the occasions on which an image will automatically be assigned the working color space. The possibilities are: on load, on stacking, on stretching and on the output of compositing tools (pixelmath / RGB composition).

The default is to assign the working color space on stretching, as this is the point at which you typically switch the viewer mode to Linear and start to stretch your image to look the way you want it to. This is when color management starts to matter.

However, you can also assign the color space at earlier stages in processing if you want to. There is little benefit or drawback to doing so in most circumstances, but if you expect to be exporting images before stretching them then selecting the other check boxes will ensure that a color profile is always assigned.

> **Tip**
>
> A color profile will **never** be automatically assigned if Siril believes it is a single channel extracted from a 3-color image using RGB, HSV, HSL, CIE L*a*b* or Green extraction. There is no ICC profile that can sensibly be assigned in this case. You should consider the file to be raw data until you are ready to either re-composite it with other data or process it to look good in its own right, at which point you should assign your preferred working color space.
>
> Split-out CFA, Ha and OIII channels can safely be assigned a linear version of the working space, and this will be done if the pedantic option described below is active.

If you want to avoid color profiles completely you can select "Never", but this will result in a non-color managed workflow and your images may not look consistent when viewed in other software or when output to different output devices.

#### Pedantically assign linear ICC profiles

When auto-assigning ICC profiles to:

- loaded images that lack an ICC profile and show no evidence of previous stretching,
- newly stacked images,
- newly composited images

Siril will assign a linear ICC profile. This is usually technically correct, as such images will still contain linear light data. The option is made available should you want it, however it is generally unnecessary: see the *profiles page* for an explanation of why. You may wish to use this in order to benefit from the display transform by applying your stretches to the original linear data while previewing the effect in sRGB (or your custom monitor profile): in this case you should deselect the following auto-conversion box, so that the profile remains linear during stretching, and then manually convert the data to your working profile on completion of stretching.

#### Convert other color profiles to the working profile?

This drop-down sets Siril's behaviour if an image is loaded with an ICC profile that is different to the selected working profile. The optons are: Never, Always and Ask every time. The default is Never, as this avoids unrequested change to the image, however for a consistent workflow you may wish to select Always, so that you are always editing in your preferred color space. Ask every time is an option, but it can be annoying especially as it regards the v2 export profiles as being different to the v4 built-in working profiles, so although it is provided for users who wish to manage every detail, it is not particularly recommended.

#### Built-in Profiles

#### Export Built-in Profiles

Siril uses profiles from the Elle Stone collection of color profiles to provide its built-in profiles. As these are distributed under a Creative Commons BY-SA-3.0 licence which requires sharing the profiles, a button is provided to export all the built-in profiles to the working directory.

In order to provide a profile capable of the Perceptual sRGB intent, the ICC Preference profile with perceptual lookup tables is provided. To export this you will need to accept the same terms and conditions text as you do to download it from the ICC website.

### 4.1.9 Scripts

- The Scripts tab essentially contains the locations where Siril should look for scripts. Indeed, by default and depending on the OS used, the scripts are installed in a specific place:
  - `/usr/local/share/siril/scripts` or `/usr/share/siril/scripts` on GNU/Linux.
  - `C:\Program Files\Siril\scripts` on Windows.
  - `/Applications/Siril.app/Contents/Resources/share/siril/scripts` on MacOS, if the application has been installed in the Applications folder.

> **Warning**
>
> On macOS, as the application is signed and notarized, it is impossible to modify the scripts inside the bundle. Otherwise, the application will not start. So you have to define another path pointing to a folder where you have write permissions.

- The **Script Storage Directories** field allows you to define custom folder paths to place scripts that you have created and/or modified. Clicking on the button just below will rescan the folders and update the list of scripts in the dedicated menu.
- The **Warning Dialogs** section proposes to disable:
  1. The warning text that is displayed before a script is executed.
  2. The check of the keyword `requires` which must be at the very beginning of the script in order to check if the script is compatible with the version of Siril. We recommend to not uncheck this option.
- The **Siril Script Repository** field is where scripts that have been validated by the Siril team are automatically uploaded. However, these may have been written by third-party developers. You can enable/disable this feature with the *Enable use of the siril-scripts online repository* button, and define automatic or manual updates with *Automatic updates* and *Manual update*. To display a script in the script menu, simply select it.
- The **Reset python venv** will kill all running python scripts and reset the virtual environment "venv" directory to an as-new state. Any modules that have been installed by scripts will require re-installation. This is normally not necessary but is provided as a last resort to restore the venv to a pristine state as an aid to troubleshooting.

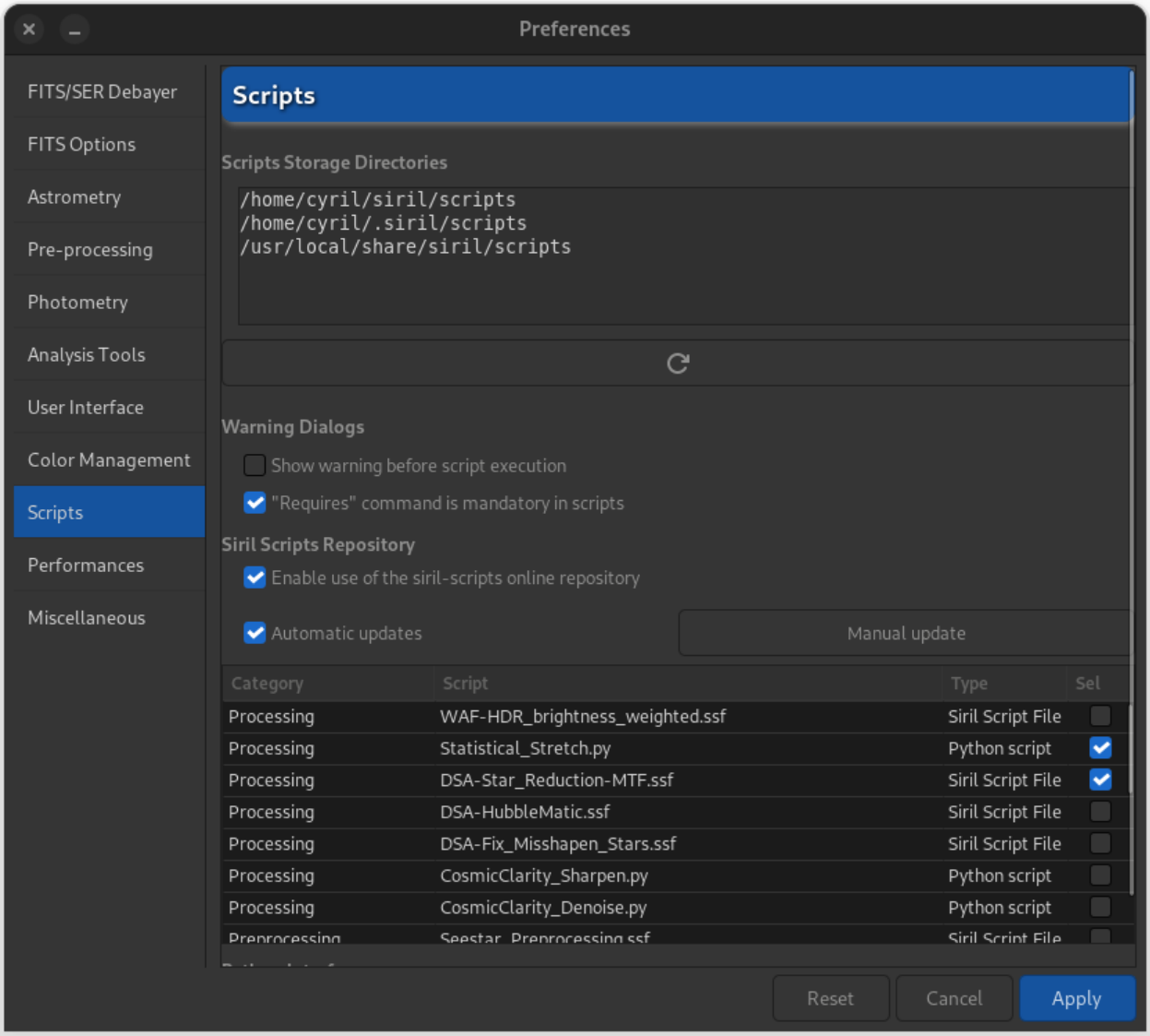



Fig. 12: Page 8 of preferences dialog

### 4.1.10 Performances

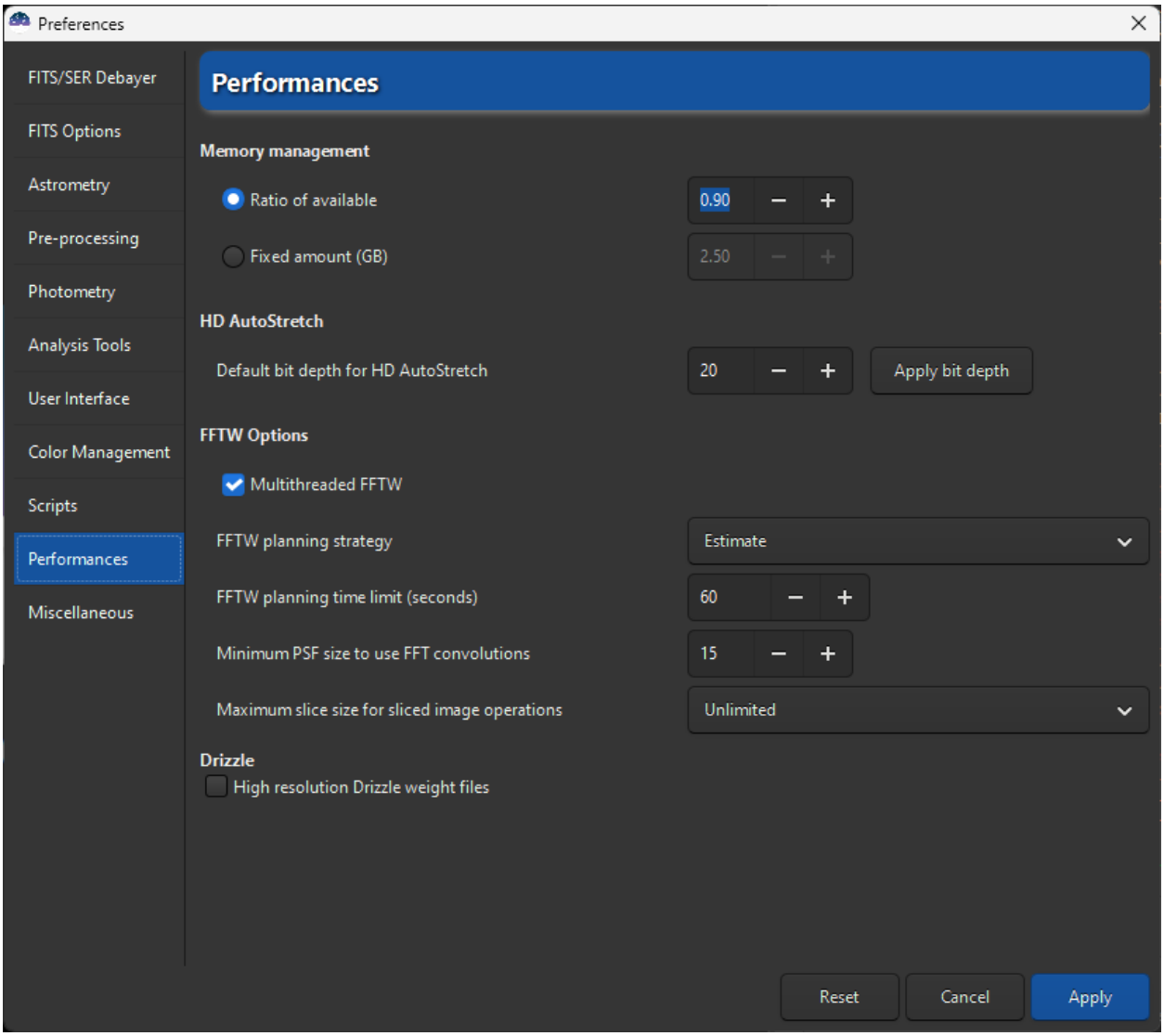



Fig. 13: Page 9 of preferences dialog

Astronomical image processing software, such as Siril, uses a lot of resources and usually requires quite powerful computers. It is not impossible, when the computer is very busy, that it freezes completely. It is not at all recommended to do anything else on the computer during the processing, especially Internet browsing, because browsers are very greedy for RAM. However, it is possible to manage the maximum percentage of RAM that Siril can use.

- **Ratio of available**: Siril will limit itself to a ratio of the amount of free physical memory and will decrease the size of work tasks if needed. A value above 1 means that some memory paged on a configured storage will be used and that the overall process will be slower and the system will likely be unresponsive during some operations. If you don't have paged memory configured on some storage, a value of 1 or above will likely result in a crash of siril or of the operating system.
- **Fixed amount (GB)**: Siril will limit itself to a fixed amount of memory and will decrease the size of work tasks if needed. Configuring a memory amount larger than what is available on your system may result in a crash of siril or of the operating system.
- The **Default bit depth for HD AutoStretch** option sets default bit depth for the HD AutoStretch display mode.

Higher bit depths require exponentially more memory for the LUT and take longer to recalculate it, but do a much better job of smoothing quantization artefacts in displaying images with very narrow histogram peaks. The default bit depth will apply from the next viewer mode change, and can be applied now using the button to the right. Click on the *Apply bit depth* button to set the selected HD AutoStretch bit depth now.

- **Multithreaded FFTW**: this toggle button sets FFTW to use multiple threads. This can be faster (though performance does not increase linearly with the number of processors, due to synchronization overhead), but FFTW's planning stage takes longer for multi-threaded systems so particularly the first FFT for a given image size may be considerably slower using multiple threads.
- **FFTW planning strategy**: this combobox sets the FFTW planning strategy. FFTW has multiple algorithms for calculating a FFT and will plan a given FFT to optimize speed. It saves the results of these plans for later reuse in a cache file called "Wisdom", so some extra time spent up-front planning can reap rewards if you calculate a lot of FFTs of the same size. Note that wisdom is specific to a given machine: it should not be shared between machines and should be deleted and regenerated from scratch following a memory or processor upgrade or a major change of software environment (major OS changes, Siril major version changes). In order of speed, Estimate is fastest: this strategy does not actually do any measurement but does planning based on a set of heuristics. Measure is next fastest: this method actually compares the speed of different internal FFTW methods of calculating the FFT and picks the fastest. As a result, the planning step takes longer. Patient considers even more possible plans, and Exhaustive considers even more. If you always process images of a specific size then the more expensive planning strategies may be worthwhile becaues of Wisdom, but if you work with images of lots of different sizes then a cheaper planning algorithm may be more suitable.
- **FFTW planning time limit**: this time limit halts FFTW planning after the specified time limit. This will override the planning strategy. Note that the time limit is not strictly enforced: FFTW will finish any non-interruptible calculation it is performing at the time the limit is reached, and if set to zero FFTW will always as a minimum carry out Estimate planning.
- **Drizzle: High resolution Drizzle weight files**: When this button is toggled, weight files produced by the drizzle algorithm will be of the same bitdepth than the images. This can be important when trying to get the best possible FWHM on the stacked image, but it is very much space consuming. By default, it is unticked and the weight files are saved in 8-bit.

### 4.1.11 Miscellaneous

The last tab contains everything that does not fit in the other themes.

- Using the *Undo*/*Redo* buttons requires disk space. Lots of space in some cases. The folder containing the swap files (which are the files necessary for the proper functioning of the undo/redo pair) can be defined in the **Swap Storage Directory** section. The disk space is listed to the right of the file chooser. We advise **to not change** the default settings unless you have a good reason to do so. As the choice of a good folder is critical, it is possible to return to the default folder by clicking on *Restore to Default*.
- The **Warnings Dialogs** allows to disable some warning popups that are here to help beginners.
- **Introduction Tips**: At the very first start of Siril, it is possible to see a little animation showing what's new in the application. This animation can be replayed by clicking on *Play introduction*.
- **StarNet Executable Location**: In order to use StarNet in Siril, it is necessary to tell Siril the path where the StarNet executable is located. For old StarNet++ v1 installations that use separate executables to process mono and RGB files, either can be chosen - Siril will autodetect the other one if both are installed. Note that for these old installations, the original executable names **rgb_starnet++** and **mono_starnet++** MUST be kept. For all newer single-executable versions of StarNet, Siril will determine the version heuristically and interface with it accordingly.
- **StarNet Weights Location**: New Torch-based versions of StarNet provide the option to provide the location of a neural network weights file: it need not be in the same directory as the executable. This preference can be used to set the location of a weights file to pass to StarNet, and it can be reset using the associated button. Note: this

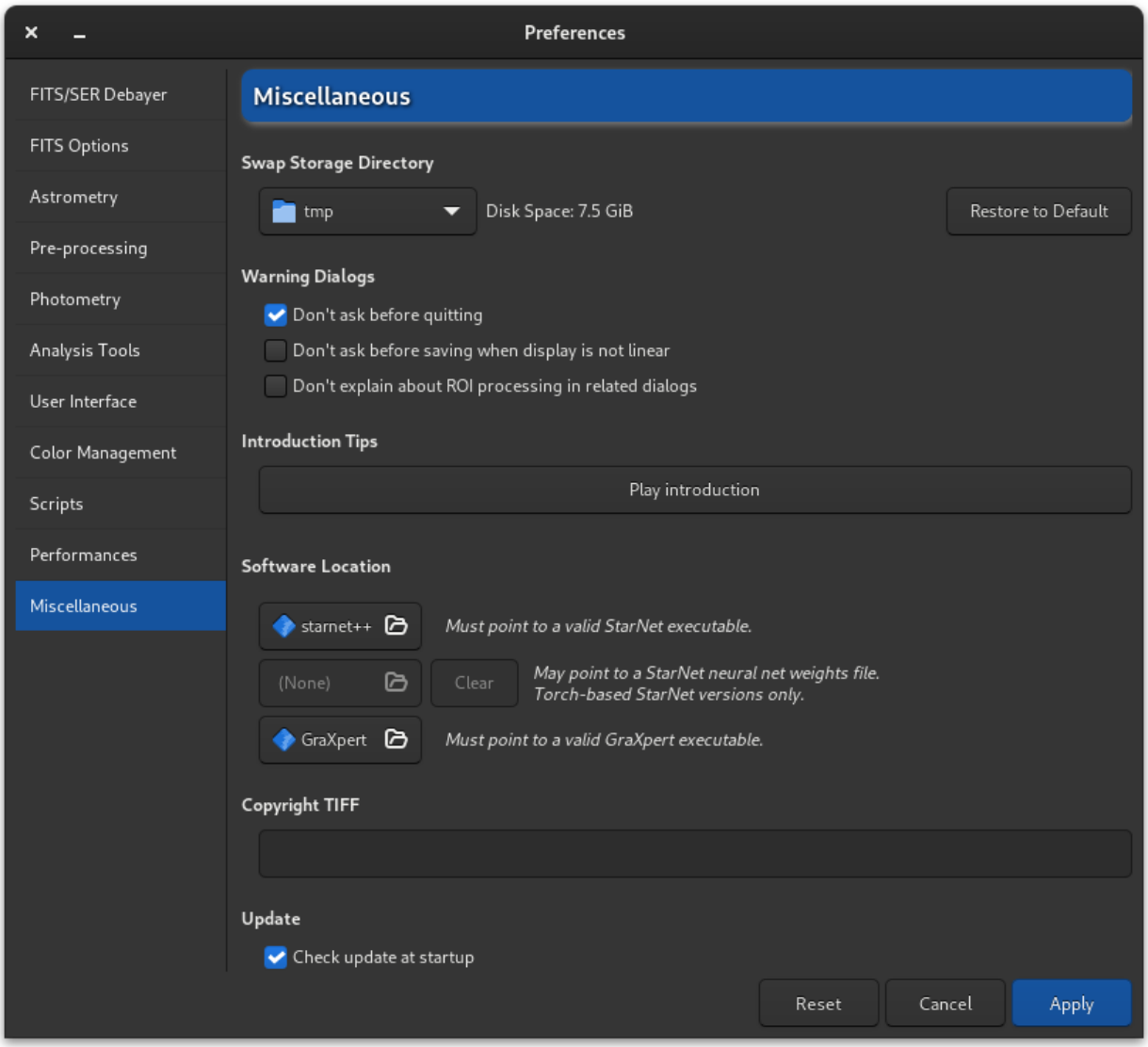



Fig. 14: Page 10 of preferences dialog

option only works with Torch-based StarNet installations. With older StarNet installations the weights file must be in the same directory as the executable.

> **Warning**
>
> This is the location of the command line version of StarNet that need to be given, not the GUI one.

> **Warning**
>
> On macOS, it can be difficult to find the directory path because Apple does not make browsing easy for some folders. A trick is to type `Shift` + `Cmd` + `g` on the open File Chooser Dialog, then directly enter the installation path, which is usually the one set by Homebrew. Usually it is `/usr/local/bin` on intel computers and `/opt/homebrew/bin/` on Apple Silicon versions.

- **GraXpert Executable Location**: In order to use GraXpert in Siril, it is necessary to tell Siril the path where the GraXpert executable is located. If you find any difficulty in getting GraXpert working with Siril, check that it works by itself before submitting a bug report: all Siril does is execute the third party software, so if it doesn't work by itself then it will not work when called from Siril either.
- **Copyright TIFF**: When saving TIFF files it is possible to customize the copyright of the dedicated EXIF metadata.
- **Update**: By default Siril checks updates at startup. You are free to disable this behavior if you don't want the application queries our website.

## 4.2 Preferences (commands)

Starting from v1.2, most preferences can also be set through commands, meaning either from direct input in the command line, through scripting or in headless mode.

To get a list of all the available variables, through siril command line:

```
get -A
```

This will print a list of all the variables to the Console, with their current value and a short description (use lowercase option *-a* to omit description).

The table below lists them:

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| core.wd | (not set) | directory | current working directory |
| core.extension | .fit | string | FITS file extension |
| core.force_16bit | false | boolean | don't use 32 bits for pixel depth |
| core.fits_save_icc | true | boolean | embed ICC profiles in FITS when saving |
| core.allow_heterogeneous_fitseq | false | boolean | allow FITS cubes to have different sizes |
| core.mem_mode | 0 [0, 1] | integer | memory mode (0 ratio, 1 amount) |
| core.mem_ratio | 0.9 [0.05, 4] | double | memory ratio of available |
| core.mem_amount | 10 [0.1, 1e+06] | double | amount of memory in GB |

continues on next page

Table 1 – continued from previous page

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| core.hd_bitdepth | 20 [17, 24] | integer | HD AutoStretch bit depth |
| core.script_check_requires | true | boolean | need requires cmd in script |
| core.pipe_check_requires | false | boolean | need requires cmd in pipe |
| core.check_updates | false | boolean | check update at start-up |
| core.lang | (not set) | string | active siril language |
| core.swap_dir | os dependant | directory | swap directory |
| core.binning_update | true | boolean | update pixel size of binned images |
| core.wcs_formalism | 1 [0, 1] | integer | WCS formalism used in FITS header |
| core.catalogue_namedstars | ~/.local/share/kstars/ | string | Path of the namedstars.dat catalogue |
| core.catalogue_unnamedstars | ~/.local/share/kstars/ | string | Path of the unnamedstars.dat catalogue |
| core.catalogue_tycho2 | ~/.local/share/kstars/ | string | Path of the deepstars.dat catalogue |
| core.catalogue_nomad | ~/.local/share/ks NOMAD-1e8.dat | string | Path of the USNO-NOMAD-1e8.dat catalogue |
| core.catalogue_gaia_astro | ~/.local/share/siril/ga | string | Path of the local Gaia astrometric catalogue |
| core.catalogue_gaia_photo | ~/.local/share/siril/ga | string | Path of the local Gaia photometric catalogue |
| core.rgb_aladin | false | boolean | add CTYPE3='RGB' in the FITS header |
| core.use_checksum | false | boolean | Verify file checksums if they exist |
| core.copyright | (not set) | string | user copyright to put in file header |
| core.starnet_exe | (not set) | string | location of the StarNet executable |
| core.starnet_weights | (not set) | string | location of the StarNet-torch weights file |
| core.graxpert_path | (not set) | string | location of the GraXpert executable |
| core.asnet_dir | (not set) | string | directory of the asnet_ansvr installation |
| core.fftw_timelimit | 60 | double | FFTW planning timelimit |
| core.fftw_conv_fft_cutoff | 15 | integer | Convolution minimum kernel size to use FFTW |
| core.fftwf_strategy | 0 | integer | FFTW planning strategy |
| core.fftw_multithreaded | true | boolean | multithreaded FFTW |
| core.max_slice_size | 32769 [512, 32769] | integer | Maximum slice size for automated slice processing |
| starfinder.focal_length | 0 [0, 999999] | double | focal length in mm for radius adjustment |
| starfinder.pixel_size | 0 [0, 99] | double | pixel size in µm for radius adjustment |
| debayer.use_bayer_header | true | boolean | use pattern from the file header |
| debayer.pattern | 0 [0, 7] | integer | index of the Bayer pattern |
| debayer.interpolation | 8 [0, 9] | integer | type of interpolation |
| debayer.orientation | 0 [0, 3] | integer | row-order preference |
| debayer.offset_x | 0 [0, 1] | integer | Bayer matrix offset X |
| debayer.offset_y | 0 [0, 1] | integer | Bayer matrix offset Y |
| debayer.xtrans_passes | 1 [1, 4] | integer | Number of passes for the X-Trans Markesteijn algorithm |
| photometry.gain | 2.3 [0, 10] | double | electrons per ADU for noise estimation |
| photometry.inner | 20 [2, 100] | double | inner radius for background annulus |
| photometry.outer | 30 [3, 200] | double | outer radius for background annulus |
| photometry.inner_factor | 4.2 [2, 50] | double | factor for inner radius automatic computation |

continues on next page

Table 1 – continued from previous page

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| photometry.outer_factor | 6.3 [2, 50] | double | factor for outer radius automatic computation |
| photometry.force_radius | true | boolean | force flux aperture value |
| photometry.auto_aperture_factor | 4 [1, 5] | double | Radius/halfFWHM ratio |
| photometry.aperture | 10 [1, 100] | double | forced aperture for flux computation |
| photometry.minval | -1500 [-65536, 65534] | double | minimum valid pixel value for photometry |
| photometry.maxval | 60000 [1, 65535] | double | maximum valid pixel value for photometry |
| photometry.discard_var_catalogues | 4 [0, 7] | integer | catalogues to be used to discard the variable stars from the comparison stars list |
| photometry.redpref | (not set) | string | preferred SPCC red filter |
| photometry.greenpref | (not set) | string | preferred SPCC green filter |
| photometry.bluepref | (not set) | string | preferred SPCC blue filter |
| photometry.lpfpref | (not set) | string | preferred SPCC DSLR LPF filter |
| photometry.oscfilterpref | No filter | string | preferred SPCC OSC filter |
| photometry.monosensorpref | (not set) | string | preferred SPCC mono sensor |
| photometry.oscsensorpref | (not set) | string | preferred SPCC OSC sensor |
| photometry.is_mono | true | boolean | is the SPCC sensor mono? |
| photometry.is_dslr | false | boolean | is the SPCC OSC sensor a DSLR? |
| photometry.nb_mode | false | boolean | are we in narrowband mode? |
| photometry.r_wl | 656.28 | double | red NB filter wavelength |
| photometry.r_bw | 6 | double | red NB filter bandwidth |
| photometry.g_wl | 500.7 | double | green NB filter wavelength |
| photometry.g_bw | 6 | double | green NB filter bandwidth |
| photometry.b_wl | 500.7 | double | blue NB filter wavelength |
| photometry.b_bw | 6 | double | blue NB filter bandwidth |
| astrometry.asnet_keep_xyls | false | boolean | do not delete .xyls FITS tables |
| astrometry.asnet_keep_wcs | false | boolean | do not delete .wcs result files |
| astrometry.asnet_show_output | false | boolean | show solve-field output in main log |
| astrometry.sip_order | 3 [1, 5] | integer | degrees of the polynomial correction |
| astrometry.radius | 10 [0.01, 30] | double | radius around the target coordinates (degrees |
| astrometry.max_seconds_run | 30 [0, 100000] | integer | maximum seconds to try solving |
| astrometry.update_default_scale | true | boolean | update default focal length and pixel size from the result |
| astrometry.percent_scale_range | 20 [1, 50] | integer | percent below and above the expected sampling to allow |
| astrometry.default_obscode | (not set) | string | default IAU observatory code |
| analysis.panel | 256 [127, 1024] | integer | panel size of aberration inspector |
| analysis.window | 381 [300, 1600] | integer | window size of aberration inspector |
| compression.enabled | false | boolean | FITS compression enabled |
| compression.method | 0 [0, 3] | integer | FITS compression method |
| compression.quantization | 16 [8, 256] | double | quantization factor for 32-bit float |
| compression.hcompress_scale | 4 [0, 256] | double | Hcompress scale factor |
| gui_prepro.cfa | false | boolean | type of sensor for cosmetic correction |
| gui_prepro.equalize_cfa | true | boolean | equalize flat channels |
| gui_prepro.fix_xtrans | false | boolean | enable correction for X-Trans sensor |

continues on next page

Table 1 – continued from previous page

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| gui_prepro.xtrans_af_x | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_af_y | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_af_w | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_af_h | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_sample_x | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_sample_y | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_sample_w | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.xtrans_sample_h | 0 | integer | if no X-Trans model found, use this |
| gui_prepro.bias_lib | (not set) | string | default master bias |
| gui_prepro.use_bias_lib | false | boolean | use default master bias |
| gui_prepro.dark_lib | (not set) | string | default master dark |
| gui_prepro.use_dark_lib | false | boolean | use default master dark |
| gui_prepro.flat_lib | (not set) | string | default master flat |
| gui_prepro.use_flat_lib | false | boolean | use default master flat |
| gui_prepro.disto_lib | (not set) | string | default distortion master |
| gui_prepro.use_disto_lib | false | boolean | use default master distortion |
| gui_prepro.stack_default | $seq-name$stacked | string | default stack name |
| gui_prepro.use_stack_default | true | boolean | use preferred stack name |
| gui_registration.method | 0 [0, 7] | integer | index of the selected registration method |
| gui_registration.interpolation | 4 [0, 5] | integer | index of the selected interpolation method |
| gui_registration.clamping | true | boolean | use clamping method with Lanczos and Cubic interpolation |
| gui_registration.drizz_weight_ma | false | boolean | Match Drizzle weights bitpix to sequence |
| gui_stack.method | 0 [0, 4] | integer | index of the selected method |
| gui_stack.normalization | 3 [0, 4] | integer | index of the normalization method |
| gui_stack.rejection | 5 [0, 7] | integer | index of the rejection method |
| gui_stack.weighting | 0 [0, 4] | integer | index of the weighting method |
| gui_stack.sigma_low | 3 [0, 20] | double | sigma low value for rejection |
| gui_stack.sigma_high | 3 [0, 20] | double | sigma high value for rejection |
| gui_stack.linear_low | 5 [0, 20] | double | linear low value for rejection |
| gui_stack.linear_high | 5 [0, 20] | double | linear high value for rejection |
| gui_stack.percentile_low | 3 [0, 100] | double | percentile low value for rejection |
| gui_stack.percentile_high | 3 [0, 100] | double | percentile high value for rejection |
| gui.first_start | (not set) | string | first start of siril |
| gui.silent_quit | false | boolean | don't confirm quit when exiting |
| gui.silent_linear | false | boolean | don't confirm save when non linear mode |
| gui.remember_windows | true | boolean | remember window position |
| gui.main_win_pos_x | 0 | integer | main window position |
| gui.main_win_pos_y | 0 | integer | main window position |
| gui.main_win_pos_w | 0 | integer | main window position |
| gui.main_win_pos_h | 0 | integer | main window position |
| gui.pan_position | -1 | integer | position of the two sides separator |
| gui.extended | true | boolean | main window is extended |
| gui.maximized | false | boolean | main window is maximized |
| gui.theme | 0 [0, 1] | integer | index of the selected theme |
| gui.font_scale | 100 | double | font scale in percent |
| gui.icon_symbolic | false | boolean | icon style |

continues on next page

Table 1 – continued from previous page

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| gui.script_path | | list of strings | list of script directories |
| gui.use_scripts_repository | true | boolean | use and sync online scripts repository |
| gui.use_spcc_repository | true | boolean | use and sync spcc-database repository |
| gui.auto_update_scripts | true | boolean | auto sync online scripts repository |
| gui.auto_update_spcc | true | boolean | auto sync spcc-database repository |
| gui.selected_scripts | | list of strings | list of scripts selected from the repository |
| gui.warn_scripts_run | true | boolean | warn when launching a script |
| gui.show_thumbnails | true | boolean | show thumbnails in open dialog |
| gui.thumbnail_size | 256 | integer | size of the thumbnails |
| gui.selection_guides | 0 | integer | number of elements of the grid guides |
| gui.show_deciasec | false | boolean | show tenths of arcseconds on hover |
| gui.default_rendering_mode | 0 [0, 6] | integer | default display mode |
| gui.display_histogram_mode | 0 [0, 1] | integer | default histogram display mode |
| gui.roi_mode | 0 | integer | ROI selection mode |
| gui.roi_warning | true | boolean | enable ROI dialog warning |
| gui.mmb_zoom_action | 0 | integer | Middle mouse button double click zoom action |
| gui.mouse_speed_limit | 0 | double | Mouse smooth scroll speed limit |
| gui.color_bkg_samples | rgba(255, 51, 26, 1.0) | string | configure background samples color |
| gui.color_std_annotations | rgba(128, 255, 77, 0.9) | string | configure standard annotation color |
| gui.color_dso_annotations | rgba(255, 128, 0, 0.9) | string | configure dso annotation color |
| gui.color_sso_annotations | rgba(255, 255, 0, 0.9) | string | configure sso annotation color |
| gui.color_tmp_annotations | rgba(255, 0, 0, 0.9) | string | configure tmp annotation color |
| gui.custom_monitor_profile | (not set) | string | path to custom monitor ICC profile |
| gui.soft_proofing_profile | (not set) | string | path to soft proofing ICC profile |
| gui.icc_custom_monitor_active | false | boolean | custom monitor profile active |
| gui.icc_soft_proofing_active | false | boolean | output proofing profile active |
| gui.custom_RGB_ICC_profile | (not set) | string | path to custom RGB ICC profile |
| gui.custom_gray_ICC_profile | (not set) | string | path to custom gray ICC profile |
| gui.rendering_intent | 1 | integer | color management rendering intent |
| gui.proofing_intent | 1 | integer | color management soft proofing intent |
| gui.export_intent | 1 | integer | color management export intent |
| gui.default_to_srgb | true | boolean | default to sRGB when exporting non color managed images |
| gui.working_gamut | 0 | integer | color management working gamut |
| gui.export_8bit_method | 0 | integer | color management export profile for 8bit files |
| gui.export_16bit_method | 1 | integer | color management export profile for 16bit files |
| gui.icc_autoconversion | 0 | integer | autoconvert images with an ICC profile to the working color space |
| gui.icc_autoassignment | 4 | integer | encodes ICC profile auto-assignment options |
| gui.icc_rendering_bpc | true | boolean | enable rendering BPC |
| gui.icc_pedantic_linear | false | boolean | pedantically assign linear ICC profiles |

continues on next page

Table 1 – continued from previous page

| Variable | Default ([Range]) | Type | Comment |
|---|---|---|---|
| gui.mouse_actions | | list of strings | mouse actions config. Opaque data structure, edit this using the GUI |
| gui.scroll_actions | | list of strings | scroll actions config. Opaque data structure, edit this using the GUI |
| gui_astrometry.compass_position | 1 [0, 5] | integer | index of the compass position over grid |
| gui_astrometry.cat_messier | true | boolean | show Messier objects in annotations |
| gui_astrometry.cat_ngc | true | boolean | show NGC objects in annotations |
| gui_astrometry.cat_ic | true | boolean | show IC objects in annotations |
| gui_astrometry.cat_ldn | true | boolean | show LDN objects in annotations |
| gui_astrometry.cat_sh2 | true | boolean | show SH2 objects in annotations |
| gui_astrometry.cat_stars | true | boolean | show stars in annotations |
| gui_astrometry.cat_const | true | boolean | show constellations in annotations |
| gui_astrometry.cat_const_names | true | boolean | show constellations names in annotations |
| gui_astrometry.cat_user_dso | true | boolean | show user DSO objects in annotations |
| gui_astrometry.cat_user_sso | true | boolean | show user SSO objects in annotations |
| gui_pixelmath.pm_presets | | list of strings | list of pixel math presets |
| script_editor.highlight_syntax | true | boolean | highlight syntax in the script editor |
| script_editor.highlight_bracketma | true | boolean | highlight matching brackets in the script editor |
| script_editor.rmargin | true | boolean | show the right margin in the script editor |
| script_editor.rmargin_pos | 80 | integer | position of the right margin in the script editor |
| script_editor.show_linenums | true | boolean | show line numbers in the script editor |
| script_editor.show_linemarks | false | boolean | show line marks in the script editor |
| script_editor.highlight_currentlin | true | boolean | highlight the current line in the script editor |
| script_editor.autoindent | true | boolean | automatically indent new lines |
| script_editor.indentontab | true | boolean | indent selected blocks of lines in the script editor using the tab key |
| script_editor.smartbs | true | boolean | Smart Backspace behaviour in the script editor |
| script_editor.smarthomeend | true | boolean | Smart Home / End behaviour in the script editor |
| script_editor.showspaces | false | boolean | Show visible space and tab characters in the script editor |
| script_editor.shownewlines | false | boolean | Show visible newline characters in the script editor |
| script_editor.minimap | false | boolean | Show a minimap in the script editor |

(*). For kstars catalogues, this will default to `~/.local/share/kstars/`, irrespective of your OS.
In any case, you will need to download them and set the path you've chosen.
See section about using *local star catalogues*.

The values can be fetched with `get` command:

**Siril command line**

```
get { -a | -A | variable }
```

Gets a value from the settings using its name, or list all with **-a** (name and value list) or with **-A** (detailed list)

See also SET to update values

Links: *set*

The values can be modified with `set` command:

**Siril command line**

```
set { -import=inifilepath | variable=value }
```

Updates a setting value, using its variable name, with the given value, or a set of values using an existing ini file with **-import=** option.
See GET to get values or the list of variables

Links: *get*

CHAPTER

FIVE

# FILE FORMATS

There are several file formats that Siril can open and work on. However, only two are read natively and allow to build *sequences*: the FITS and SER formats.

Here we will look at the different file formats read by Siril and understand the limitations of some and the strengths of others.

## 5.1 Bit Depth

The bit depth, defines the number of bits used to indicate the color of a single pixel, or the number of bits used for each color component of a single pixel.

For images of daily life, 8-bit is more than enough. This means that a pixel is encoded on values in the range [0, 255]. However, photographing astronomical objects is more demanding and usually requires working on images with a bit depth of at least 16-bit: *i.e.* in the range [0, 65535]. Even better, 32-bit precision allows the most subtle information to be retained. On this last type, the pixels are either encoded in the interval [0, 4294967295], or, as used in Siril, between the floating values [0, 1]. It is possible to find formats encoding pixels on 64-bit (in the range [0, 1], but they are rare and have a very specific use. In particular the FITS format allows this.

However, not all image file formats support 16-bit, let alone 32-bit. This must therefore be taken into account when choosing a format to work with.

## 5.2 Common File Formats

The image file formats presented here are standard formats, readable by all image manipulation software. These formats were designed to meet the needs some time ago and may be obsolete. Moreover, none of these formats have been designed to handle astronomical data. They must therefore generally be used at the end of the processing chain.

### 5.2.1 AVIF Format

AVIF is a modern image format based on the AV1 video codec. This format uses the HEIC container format. Siril supports reading AVIF images up to 12 bits bitdepth. Lossless support may be available depending on the codecs compiled into the installed libheif. AVIF supports color profiles using both embedded ICC profiles and NCLX identifiers: embedded ICC profiles are fully supported, and there is partial support for NCLX identifiers (they are converted into the equivalent ICC profile, although currently only the most likely NCLX profiles are supported).

The filename extensions for this format are **.avif**.

Fig. 1: Linear image saved in 16-bit

Fig. 2: The same linear image saved in 8-bit. Almost all data have been lost

### 5.2.2 BMP Format

Files with the **.bmp** extension are bitmap image files used to store digital bitmap images. These images are independent of the graphics card and are also called Device Independent Bitmap (DIB) file format. This independence allows the file to be opened on multiple platforms such as Microsoft Windows and Mac. The BMP file format allows data to be stored as two-dimensional digital images, both in monochrome and in color, with different color depths.

Nowadays, this format is not really used anymore and other file types are preferred.

### 5.2.3 HEIF Format

HEIF is a modern image format based on the x265 video codec. This format uses the HEIC container format. Siril supports reading HEIC images up to 12 bits bitdepth. Lossless support may be available depending on the codecs compiled into the installed libheif. HEIC supports color profiles using both embedded ICC profiles and NCLX identifiers: embedded ICC profiles are fully supported, and there is partial support for NCLX identifiers (they are converted into the equivalent ICC profile, although currently only the most likely NCLX profiles are supported).

The filename extensions for this format are **.heic** and **.heif**.

### 5.2.4 JPEG Format

Probably the most used file format for sharing images on forums, by e-mail or usb sticks. This format allows a more or less strong (destructive) compression which gives ideal file sizes for exchanges. The extension of this type of file is **.jpg** or **.jpeg**.

The JPEG format is however only coded in 8-bit. With the compression that produces artifacts, this format is not very suitable for astronomy images and we generally prefer the PNG format.

### 5.2.5 JPEG XL Format

Following several previous updates to the JPEG format (JPEG 2000, JPEG XR...) JPEG XL represents a major jump in capability with the stated intention of replacing all raster formats. Whether or not it achieves this, it has several features that make it of interest for astrophotography and Siril has therefore added read and write support. It supports high bit depths up to 32 bit float. It supports lossy and lossless compression: for lossy compression, it achieves much better compression ratios than JPEG for the same quality setting, and for lossless compression, it achieves much better compression ratios than PNG. Note that FITS metadata will not be preserved when saving as JPEG XL: its primary use within Siril is likely to be as an archival format for final renders of images. It may also be useful for web export: the format is not yet widely supported by browsers, but Safari supports it natively and support is available experimentally or using add-ons in Firefox and Chrome so we may see support become more widely available in the future.

JPEG XL has full support for embedded ICC profiles, however to improve compression it will save profiles it recognises with a coded identifier instead of the full profile. This means that the ICC profile description you see after opening a saved JPEG XL file may not be the same as it was originally, however the ICC profile will be functionally identical.

The filename extension for this format is **.jxl**.

### 5.2.6 PNG Format

Portable Network Graphics is a raster-graphics file format that supports lossless data compression. The extension of the format is **.png**. PNG grayscale images support the widest range of pixel depths of any image type. Depths of 1, 2, 4, 8, and 16-bit are supported, covering everything from simple black-and-white scans to full-depth medical and raw astronomical images.

Calibrated astronomical image data is usually stored as 32-bit or 64-bit floating-point values, and some raw data is represented as 32-bit integers. Neither format is directly supported by PNG.

However, this format is an excellent choice for saving the final image, after processing.

### 5.2.7 TIFF Format

TIFF or TIF, Tagged Image File Format, represents raster images for use on various devices that conform to this file format standard. It is capable of describing bi-level, grayscale, palette color and full color image data in multiple color spaces. It supports both lossless and lossy compression schemes to allow applications that use this format to choose between space and time. The extension is either **.tiff** or **.tif**.

The advantages of the TIFF format are multiple. It supports encoding up to 32-bit per pixel and offers a wide variety of possible fields in the metadata making it a good candidate for storing astronomical data.

Using the TIFF format, and in collaboration with other developers, we have set up a pseudo standard, *Astro-TIFF*.

### 5.2.8 NetPBM Format

Several graphics formats are used and defined by the Netpbm project. The portable pixmap format (PPM), the portable graymap format (PGM) and the portable bitmap format (PBM) are image file formats designed to be easily exchanged between platforms. Possible file extension are **.pbm**, **.pgm** (for gray scale files) and **.ppm**.

These formats, supporting up to 16-bit per channel, are marginally used and should only be used for final image storage.

### 5.2.9 AVI format

It's a film container, able to contain data with various audio and video codecs. Some lossless video codecs exist and they have been used for astronomy imaging purposes in the past, but it's a format that does not contain metadata usable for astronomy, that is limited to 8-bit images and that does not give any warranty that the data it contains is raw.

> **Warning**
>
> This input file format is now deprecated. We recommend to use *SER* format instead.

## 5.3 FITS

### 5.3.1 Specification

FITS stands for **Flexible Image Transport System** and is the standard astronomical data format used by professional scientists such as NASA. FITS is much more than an image format (such as JPG or TIFF) and is primarily designed to store scientific data consisting of multidimensional arrays.

A FITS file consists of one or more header and data units (HDUs), with the first HDU referred to as the "primary HDU" or "primary array." Five primary data types are supported: 8-bit unsigned bytes, 16 and 32-bit signed integers, and 32 and 64-bit single and double-precision floating-point reals. The FITS format can also store 16 and 32-bit unsigned integers.

Each header unit consists of any number of 80-character keyword records which have the general form:

```
KEYNAME = value / comment string
```

The keyword names may be up to 8 characters long and can only contain uppercase letters, the digits 0-9, the hyphen, and the underscore character. The keyword name is (usually) followed by an equals sign and a space character (= ) in columns 9 - 10 of the record, followed by the value of the keyword which may be either an integer, a floating point number, a character string (enclosed in single quotes), or a boolean value (the letter **T** or **F**).

The last keyword in the header is always the `END` keyword which has no value or comment fields.

Each header unit begins with a series of required keywords that specify the size and format of the following data unit. A 2-dimensional image primary array header, for example, begins with the following keywords:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  440 / length of data axis 1
NAXIS2  =                  300 / length of data axis 2
```

**Note**

In Siril, 64-bit FITS files are not supported. Siril reads them but converts them to 32-bit files.

### 5.3.2 Filenames

FITS files should have an extension ".fit", ".fits" or ".fts"; uppercase variations are accepted but not recommended. It is possible to select your preferred extension in the *Preferences* dialog, and this will be used for saving FITS files, but all permitted extensions will be accepted for reading FITS images or sequence frames.

**Warning**

The FITS file library cfitsio interprets square brackets in filenames as providing filtering options. Therefore regardless of whether square bracket characters are permitted in filenames by your operating system, they **must not** be used in FITS files.

### 5.3.3 Compression

Compression is the way to reduce the size of the image. There are many methods of compression depending on the type of images used. This compression can be destructive, as with the JPEG, or lossless as proposed by the PNG.

It is possible to work with compressed FITS files. At the cost of a longer calculation time, the size of the images can be reduced considerably. Siril offers several compression algorithms which are the following:

- **Rice**: The Rice algorithm is simple and very fast
- **GZIP 1**: The gzip algorithm is used to compress and uncompress the image pixels. Gzip is the compression algorithm used in the free GNU software utility of the same name.
- **GZIP 2**: The bytes in the array of image pixel values are shuffled into decreasing order of significance before being compressed with the gzip algorithm. This is usually especially effective when compressing floating-point arrays.

One option is associated to these algorithms, the **Quantization level**:

While floating-point format images may be losslessly compressed (using gzip, since Rice only compresses integer arrays), these images often do not compress very well because the pixel values are too noisy; the less significant bits in the mantissa of the pixel values effectively contain incompressible random bit patterns. In order to achieve higher compression, one needs to remove some of this noise, but without losing the useful information content. If it is too large, one undersamples the pixel values resulting in a loss of information in the image. If it is too small, however, it preserves too much of the noise (or even amplifies the noise) in the pixel values, resulting in poor compression.

**Note**

The supported image compression algorithms are all **loss-less** when applied to integer FITS images; the pixel values are preserved exactly with no loss of information during the compression and uncompression process. Floating point FITS images (which have `BITPIX = -32` or `-64`) are first quantized into scaled integer pixel values before

being compressed. This technique produces much higher compression factors than simply using GZIP to compress the image, but it also means that the original floating value pixel values may not be precisely returned when the image is uncompressed. When done properly, this only discards the 'noise' from the floating point values without losing any significant information.

### 5.3.4 Particular Cases of Scientific FITS Files

FITS files produced by major astronomical missions such as those from NASA's Hubble Space Telescope (HST) or the James Webb Space Telescope (JWST) are often more complex than typical FITS files used by amateur astronomers. These files are generally structured as FITS sequences (also referred to as FITS cubes), but with an important difference: the individual images within these sequences may have different dimensions.

This feature allows for storing multi-dimensional data collected under varying conditions, where each frame or image in the sequence might not have the same resolution or size. For example, data from the JWST often involves such heterogeneous datasets due to the nature of its instruments and observing modes.

In Siril, when processing these types of scientific FITS files, it is crucial to enable the option that allows handling images of varying dimensions within a FITS cube. This can be done by selecting the option *Allow FITS cubes to have images of different size* in the *FITS Options* tab of the preferences.

**Warning**

Currently, Siril is unable to process FITS files with `NAXES[2] > 3`. This feature is planned for future versions of the software, so FITS cubes with more than three axes are not supported at the moment.

### 5.3.5 Orientation of FITS images

The FITS standard is a container that describes how to store image data and metadata. Professional tools, from the early age of the FITS format, like ds9 (Harvard Smithsonian Center for Astrophysics), fv (FITS viewer from NASA), store images **bottom-up**. We might be tempted to say that it does not really matter, but when demosaicing or astrometry is involved, problems arise. For example, the usual **RGGB** Bayer pattern becomes **GBRG** if the image is upside-down.

Nowadays, despite this, most camera drivers are writing data in the top-down order and we have to cope with it.

For these reasons, we recently have introduced, together with P. Chevalley of CCDCiel, a **new FITS keyword**. We encourage all data producers, INDI and ASCOM developers, to use it in order to make things easier for everybody.

This keyword is `ROWORDER` of type `TSTRING`. It can take two values: `BOTTOM-UP` and `TOP-DOWN`.

Siril will always read and display images in the bottom-up order, however if the top-down information is specified in the keyword, then Siril will demosaic the image with the corrected pattern.

#### Why would some programs write images bottom-up in the first place?

The reason is: mathematics do it that way.

Also, the FITS specification says:

***5.1. Image display conventions***

*It is very helpful to adopt a convention for the display of images transferred via the FITS format. Many of the current image processing systems have converged upon such a convention. Therefore, we recommend that FITS writers order the pixels so that the first pixel in the FITS file (for each image plane) be the one that would be displayed in the lower-left corner (with the first axis increasing to the right and the second axis increasing upwards) by the*

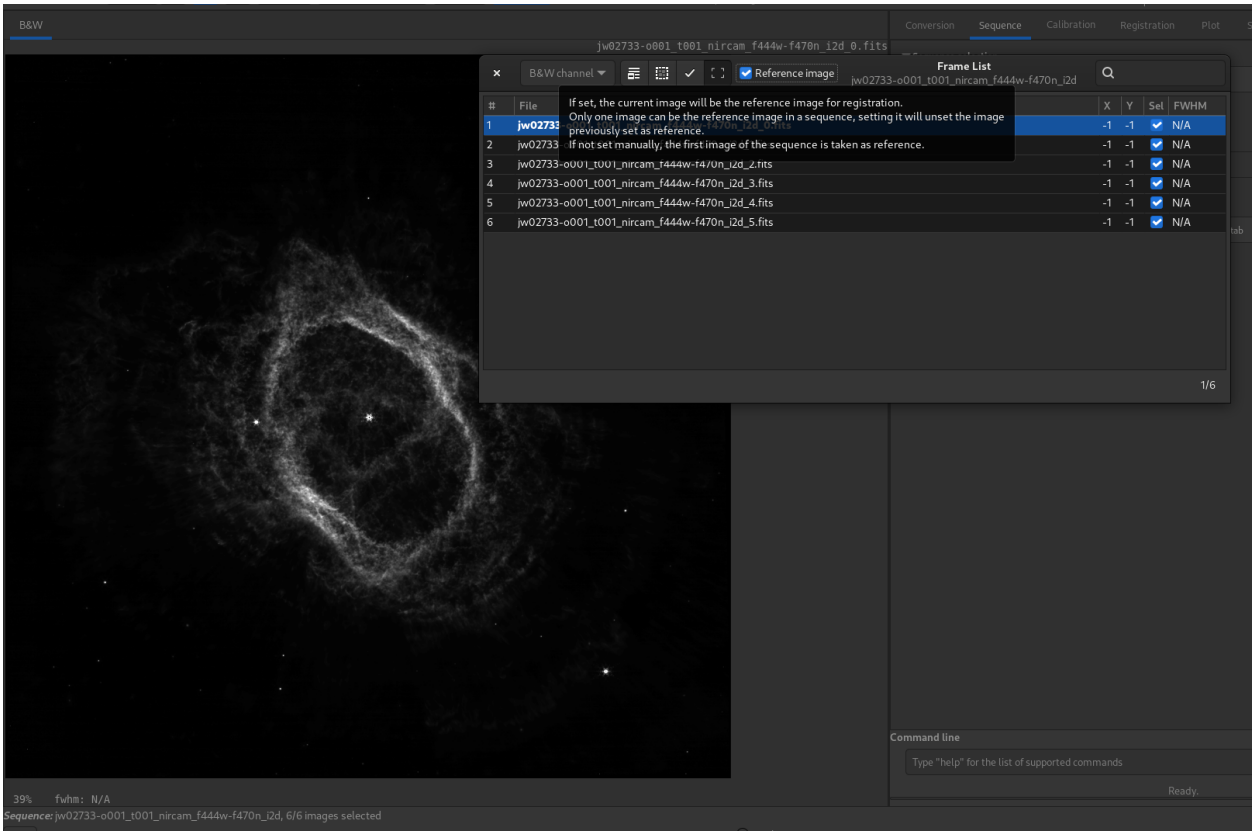



Fig. 3: Example of a JWST image loaded in Siril

*imaging system of the FITS writer. This convention is clearly helpful in the absence of a description of the world coordinates. It does not preclude a program from looking at the axis descriptions and overriding this convention, or preclude the user from requesting a different display. This convention also does not excuse FITS writers from providing complete and correct descriptions of the image coordinates, allowing the user to determine the meaning of the image. The ordering of the image for display is simply a convention of convenience, whereas the coordinates of the pixels are part of the physics of the observation.*

**Warning**

`ROWORDER` keyword can be used for:

1. Displaying the image with the intended orientation (unflip the display).
2. Unflip the Bayer demosaic pattern. So the demosaic pattern can be specified conform the sensor supplier.

**BUT**

1. `ROWORDER` shall not be used to unflip the image data for stacking. Otherwise new images would become incompatible with older darks and flats.
2. `ROWORDER` shall not be used to unflip the image data for astrometric solving. This would make the astrometric solution incompatible with other programs.

### 5.3.6 Software using `ROWORDER` keyword

- Siril (since version 0.99.4)
- CCDCiel (since version 0.9.72)
- Indi (since Jul. 2020)
- KStars (since 3.4.3)
- SharpCap (since version 3.3)
- FireCapture (since version 2.7)
- N.I.N.A (since version 1.10)
- MaxImDL (since version 6.23)
- INDIGO (since Jul. 2020)
- PixInsight (since version 1.8.8-6)
- ASTAP (since version ß0.9.391)
- APT (since version 3.86.3)
- AstroDMx Capture (since version 0.80)
- Astroart (since version 8.0)

### 5.3.7 Retrieving the Bayer matrix

Image row order changes the way the Bayer matrix should be read, but there are also two optional FITS header keywords that have an effect on this: `XBAYROFF` and `YBAYROFF`. They specify an offset to the Bayer matrix, to start reading it on first column or first row.

To help developers integrating the ROWORDER, XBAYROFF and YBAYROFF keywords in their software, some test images were created by Han Kleijn from hnsky.org, one for each combination of the three keywords. Download them here: Bayer_test_pattern_v6.tar.gz.

### 5.3.8 List of FITS keywords

Siril can read and interpret a wide range of keywords. The following list illustrates the non-standard keywords that Siril registers if necessary. Some keywords read by Siril may not appear in this list. For example, the keywords CCDTEMP or TEMPERAT, that indicate the temperature of the sensor, are correctly read, but are propagated under the keyword CCD-TEMP.

**Tip**

Siril is able to read and compare checksums if they are present in the FITS header. However, by default and for software performance reasons, CHECKSUM and DATASUM cards are automatically removed from HDU headers when a file is opened, and any CHECKSUM or DATASUM cards are stripped from headers when an HDU is written to a file. Nevertheless, they can be saved at the user's request, using the **-chksum** option of the *save* command, or via the graphical user interface.

Table 1: FITS keywords saved by Siril. For reasons of clarity, astrometry SIP keywords are not listed.

| FITS Keyword | Type | Comment |
|---|---|---|
| BZERO | Double | Offset data range to that of unsigned short |
| BSCALE | Double | Default scaling factor |
| MIPS-HI | Unsigned short | Upper visualization cutoff |
| MIPS-LO | Unsigned short | Lower visualization cutoff |
| MIPS-FHI | Float | Upper visualization cutoff |
| MIPS-FLO | Float | Lower visualization cutoff |
| PROGRAM | String | Software that created this HDU |
| FILENAME | String | Original Filename |
| DATE | String | UTC date that FITS file was created |
| DATE-OBS | String | YYYY-MM-DDThh:mm:ss observation start, UT |
| IMAGETYP | String | Type of image |
| ROWORDER | String | Order of the rows in image array |
| EXPTIME | Double | [s] Exposure time duration |
| TELESCOP | String | Telescope used to acquire this image |
| OBSERVER | String | Observer name |
| FILTER | String | Active filter name |
| APERTURE | Double | Aperture of the instrument |
| ISOSPEED | Double | ISO camera setting |
| FOCALLEN | Double | [mm] Focal length |
| CENTALT | Double | [deg] Altitude of telescope |
| CENTAZ | Double | [deg] Azimuth of telescope |
| XBINNING | Unsigned int | Camera binning mode |
| YBINNING | Unsigned int | Camera binning mode |
| XPIXSZ | Double | [um] Pixel X axis size |
| YPIXSZ | Double | [um] Pixel Y axis size |
| INSTRUME | String | Instrument name |
| CCD-TEMP | Double | [degC] CCD temperature |
| SET-TEMP | Double | [degC] CCD temperature setpoint |

continues on next page

Table 1 – continued from previous page

| FITS Keyword | Type | Comment |
|---|---|---|
| GAIN | Unsigned short | Sensor gain |
| OFFSET | Unsigned short | Sensor gain offset |
| CVF | Double | [e-/ADU] Electrons per A/D unit |
| BAYERPAT | String | Bayer color pattern |
| XBAYROFF | Int | X offset of Bayer array |
| YBAYROFF | Int | Y offset of Bayer array |
| FOCNAME | String | Focusing equipment name |
| FOCPOS | Int | [step] Focuser position |
| FOCUSSZ | Int | [um] Focuser step size |
| FOCTEMP | Double | [degC] Focuser temperature |
| STACKCNT | Unsigned int | Stack frames |
| LIVETIME | Double | [s] Exposure time after deadtime correction |
| EXPSTART | Double | [JD] Exposure start time (standard Julian date) |
| EXPEND | Double | [JD] Exposure end time (standard Julian date) |
| OBJECT | String | Name of the object of interest |
| AIRMASS | Double | Airmass at frame center (Gueymard 1993) |
| SITELAT | Double | [deg] Observation site latitude |
| SITELONG | Double | [deg] Observation site longitude |
| SITEELEV | Double | [m] Observation site elevation |
| DFTTYPE | String | Module/Phase of a Discrete Fourier Transform |
| DFTORD | String | Low/High spatial freq. are located at image center |
| DFTNORM1 | Double | Normalisation value for channel #1 |
| DFTNORM2 | Double | Normalisation value for channel #2 |
| DFTNORM3 | Double | Normalisation value for channel #3 |
| CTYPE3 | String | RGB image |
| OBJCTRA | String | Image center Right Ascension (hms) |
| OBJCTDEC | String | Image center Declination (dms) |
| RA | Double | Image center Right Ascension (deg) |
| DEC | Double | Image center Declination (deg) |
| CTYPE1 | String | TAN (gnomic) projection |
| CTYPE2 | String | TAN (gnomic) projection |
| CUNIT1 | String | TAN (gnomic) projection + SIP distortions |
| CUNIT2 | String | TAN (gnomic) projection + SIP distortions |
| EQUINOX | Double | Equatorial equinox |
| CRPIX1 | Double | Axis1 reference pixel |
| CRPIX2 | Double | Axis2 reference pixel |
| CRVAL1 | Double | Axis1 reference value (deg) |
| CRVAL2 | Double | Axis2 reference value (deg) |
| LONPOLE | Double | Native longitude of celestial pole |
| CDELT1 | Double | X pixel size (deg) |
| CDELT2 | Double | Y pixel size (deg) |
| PC1_1 | Double | Linear transformation matrix (1, 1) |
| PC1_2 | Double | Linear transformation matrix (1, 2) |
| PC2_1 | Double | Linear transformation matrix (2, 1) |
| PC2_2 | Double | Linear transformation matrix (2, 2) |
| CD1_1 | Double | Scale matrix (1, 1) |
| CD1_2 | Double | Scale matrix (1, 2) |
| CD2_1 | Double | Scale matrix (2, 1) |
| CD2_2 | Double | Scale matrix (2, 2) |
| PLTSOLVD | Logical | Siril solver |

## 5.4 Astro-TIFF

In 2022, Han Kleijn, developer of the ASTAP software, offered to contribute to the development of a new pseudo-standard using the TIFF format and taking advantage of the power of FITS file headers. This is how the Astro-TIFF format was born.

### 5.4.1 Why a new standard among all the others?

Currently, the most used format for astrophotography is the FITS format. This one, developed by professional scientists, meets all the expectations of amateurs. And although its great flexibility causes some concerns of compatibility it remains the format to be preferred.

Other specialized formats exist but are usually associated with a specific software. Like the XISF format developed by the PixInsight team. This last format, although often requested in Siril, is a format dedicated to PixInsight, a proprietary software. So the interest in developing compatibility with Siril is minimal, and we've only done it for reading in the 1.4.x cyle.

Developing Astro-TIFF appears then as a good alternative, because based on the TIFF format, it is possible to open the files on any image processing software.

Finally, the TIFF format supports compression (as does the FITS format) which allows for smaller image sizes.

### 5.4.2 Specification 1.0

#### Dated 2022-06-21

- Files are following the TIFF 6.0 specification including supplement 2 fully.
- The FITS header is written to the TIFF baseline tag **Image Description**. Code 270, Hex 010E.
- The header is following the FITS specification except that that the lines can be shorter then 80 characters and lines are ending with either CR+LF (0D0A) or LF (0A).
- First line in the description is the first header line and starts with `SIMPLE`. The last line of the header starts with `END`.

#### Recommendations

- `TIFFtag_orientation=1` (left-top) Orientation is following the conventions. Pixel `FITS_image[1,1]` is left-bottom. `TIFF_image[0,0]` is left-top. These pixels are first written or read from the file. So when writing a FITS image into TIFF preserving the orientation for the user, the first pixel to write is `FITS_image[1,NAXIS2]`.
- `TIFFtag_compression=8` (Deflate) or 5 (LZW).
- For greyscale images `TIFFtag_PhotometricInterpretation = 1` (minimum value is black, maximum is white).
- Write all available header keywords.

**Notes**

- This use of TIFF format is intended for 16-bit lights, darks, flats and flat-darks (astronomical images), but can also be used in the 32-bit format. It is possible to convert FITS to TIFF and backwards but the application programmer can decide to export only (write) or only import (read) in Astro-TIFF format.
- If an astrometrical (plate) solution is included then it should match with the image orientation.
- Some header keywords are redundant like `NAXIS1`, `NAXIS2`, `BZERO` and `BITPIX` and are not required. TIFF image dimensions and type are leading.

- The de-mosaic pattern specified in the header should match with the image orientation.
- The header will be visible in many image manipulation programs.

Example of an Astro-TIFF header that looks just like a FITS file header:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 6248 / length of data axis 1
NAXIS2  =                 4176 / length of data axis 2
NAXIS3  =                    1 / length of data axis 3
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
BZERO   =                    0 / offset data range to that of unsigned short
BSCALE  =                    1 / default scaling factor
DATE    = '2022-12-14T16:05:47' / UTC date that FITS file was created
DATE-OBS= '2022-05-06T20:29:20.019000' / YYYY-MM-DDThh:mm:ss observation start,
INSTRUME= 'ZWO CCD ASI2600MM Pro' / instrument name
OBSERVER= 'Unknown '           / observer name
TELESCOP= 'iOptron ZEQ25'      / telescope used to acquire this image
ROWORDER= 'TOP-DOWN'           / Order of the rows in image array
XPIXSZ  =                 3.76 / X pixel size microns
YPIXSZ  =                 3.76 / Y pixel size microns
XBINNING=                    1 / Camera binning mode
YBINNING=                    1 / Camera binning mode
FOCALLEN=              370.092 / Camera focal length
CCD-TEMP=                 -9.8 / CCD temp in C
EXPTIME =                  120 / Exposure time [s]
STACKCNT=                  126 / Stack frames
LIVETIME=                15120 / Exposure time after deadtime correction
FILTER  = 'Lum     '           / Active filter name
IMAGETYP= 'Light Frame'        / Type of image
OBJECT  = 'Unknown '           / Name of the object of interest
GAIN    =                  100 / Camera gain
OFFSET  =                   50 / Camera offset
CTYPE1  = 'RA---TAN'           / Coordinate type for the first axis
CTYPE2  = 'DEC--TAN'           / Coordinate type for the second axis
CUNIT1  = 'deg     '           / Unit of coordinates
CUNIT2  = 'deg     '           / Unit of coordinates
EQUINOX =                 2000 / Equatorial equinox
OBJCTRA = '09 39 54.932'       / Image center Right Ascension (hms)
OBJCTDEC= '+70 00 10.118'      / Image center Declination (dms)
RA      =              144.979 / Image center Right Ascension (deg)
DEC     =              70.0028 / Image center Declination (deg)
CRPIX1  =               3123.5 / Axis1 reference pixel
CRPIX2  =               2088.5 / Axis2 reference pixel
CRVAL1  =              144.979 / Axis1 reference value (deg)
CRVAL2  =              70.0028 / Axis2 reference value (deg)
CD1_1   =         -0.000580606 / Scale matrix (1, 1)
CD1_2   =         -4.12215e-05 / Scale matrix (1, 2)
CD2_1   =         -4.11673e-05 / Scale matrix (2, 1)
```

(continues on next page)

(continued from previous page)

```
CD2_2   =          0.000580681 / Scale matrix (2, 2)
PLTSOLVD=                    T / Siril internal solve
HISTORY Background extraction (Correction: Subtraction)
HISTORY Plate Solve
END
```

### 5.4.3 Saving Astro-TIFF in Siril

In Siril you can save Astro-TIFF files by choosing the TIFF format in the save dialog when you click on *Save As*. The drop-down list in the TIFF dialog allows you to choose between saving in standard TIFF format or in Astro-TIFF format. The latter is the default format.

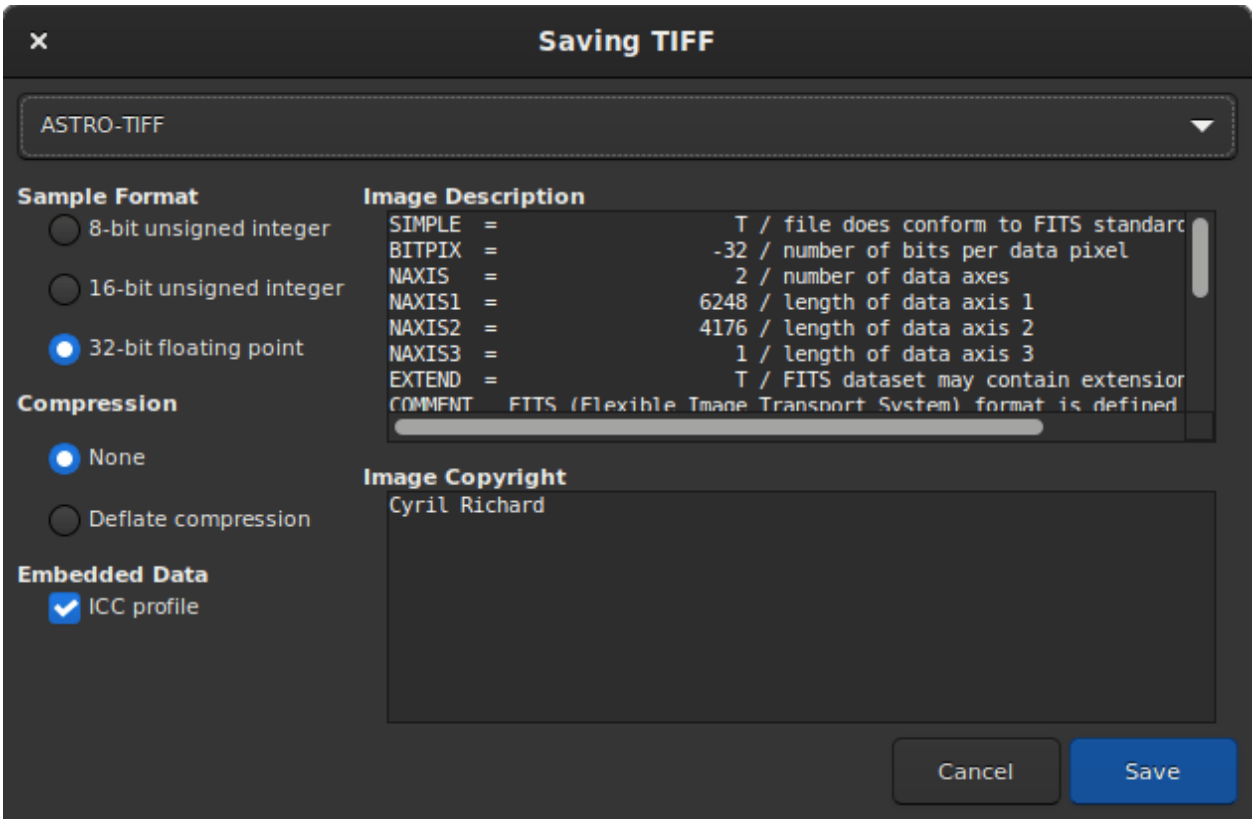



Fig. 4: Save Dialog with Astro-TIFF option

**Siril command line**

```
savetif filename [-astro] [-deflate]
```

Saves current image under the form of a uncompressed TIFF file with 16-bit per channel: **filename**.tif. The option **-astro** allows saving in Astro-TIFF format, while **-deflate** enables compression.

See also SAVETIF32 and SAVETIF8

### 5.4.4 Sample files

- Astro-TIFF file created by Siril (32-bit, uncompressed).
- Astro-TIFF file created by Siril (32-bit, compressed).

## 5.5 SER

SER file format is a simple image sequence format, similar to uncompressed films. Documentation can be found on the official page. The latest PDF document is mirrored on free-astro too.

With improvements of version 2 and 3, SER handles colour images, which makes it perfect as replacement for the usual AVI or other film format produced by older capture programs in all astronomy situations. Compressed images should not be used for astronomy but can still be converted to SER, which will make the files bigger for the same quality, but easier to work with.

Siril can convert any image sequence and many film formats into SER files. Ser-player is a great tool that allows SER files to be visualised just like any film, with many options and works on most operating systems.

The main issue with AVI and other film containers is that it is designed to work with many codecs and pixel formats, which it good for general purpose films, but requires astronomy software to handle a large array of actually different file formats. General purpose film software are often not well equipped to handle 16-bit per pixel values or some uncompressed data formats. With SER, only one file format handles it all, that's why Siril for example is now developing processing only for SER.

### 5.5.1 File structure

A SER file has three parts:

- a 178-byte header containing images and observation information
- image data, raw pixel data
- an optional trailer containing dates for all images of the sequence

### 5.5.2 Handling colours

In version 3 (2014), there are two ways of handling coloured images in SER. If data comes directly from a sensor, the preferred way is probably to use one-plane images and interpolating data from the colour filter array (similarly to CFA file formats used in astronomy software).

The other way, added in version 3, is to use three planes to represent RGB image data. SER v3 supports RGB/BGR 8/16-bit data. This can be useful if data is converted from a source with an unknown colour filter array or for general purpose conversion.

### 5.5.3 Specification issue with endianness

Since SER files can contain 16-bit precision images, endianness must be well specified. The specification allows endianness to be either big-endian or little-endian, to facilitate file creation on various systems, as long as the used endianness is documented in the file's header.

For an unknown reason, several of the first programs to support SER disrespect the specification regarding the endianness flag. The specification states that a boolean value is used for the LittleEndian header, and they use it as a BigEndian header, with 0 for little-endian and 1 for big-endian. Consequently, to not break compatibility with these first implementations, later programs, like Siril, GoQat, Ser-player and many others, have also decided to implement this header in opposite meaning to the specification.

## 5.6 IRIS PIC

The PIC format is a proprietary image format created for Christian Buil's IRIS software. To ensure compatibility with the latter, Siril is able to read this type of file. However, because the format is proprietary and the specifications are not known, not all header information will be saved when converting to FITS.

Siril cannot record in PIC format.

## 5.7 PixInsight XISF

PixInsight introduced a file structure called Extensible Image Serialization Format or XISF. This format was written to replace the FITS format, commonly used in astrophotography. However, this format was essentially designed by, and for, the PixInsight software, and many important types of metadata are stored in private namespaces that the Pixinsight developers advise cannot be relied on and may change arbitrarily between XISF specification versions. Moreover, the way a keyword is used to describe a given feature can be very opaque. As a result, the advantages it could bring for use within Pixinsight are generally nil for other software. Moreover, the FITS file format is a recognized and widely used format in the community. NASA itself uses it for its own images. It's very flexible and powerful, and contains all the features needed for modern astrophotographic processing.

We have therefore decided to enable Siril to read this type of file, but in read-only mode. Saving in XISF format is not, and will not be, implemented.

CHAPTER

SIX

# COLOR MANAGEMENT

Siril now offers a fully color managed workflow using ICC profiles. By ensuring accurate and consistent color reproduction across different devices, Siril enables you to edit your images with confidence, knowing that the colors you see on your screen will translate faithfully to prints or other displays. You can edit your images in color spaces better matched to the capabilities of a wide gamut professional photo printer, thus allowing production of prints with richer colors than are possible when working in the standard sRGB color space, and you can produce images that your audience will be able to view consistently with how you see them during editing, regardless of their display device.

There are several pages in this section of the documentation: between them they aim to provide a basic outline of what color management does, how it can help you and how it works in Siril.

## 6.1 Getting Started

Welcome to Siril's new color management system! Color management can seem complex, so this page aims to provide the absolute basics you need to know to get started. It will cover two setups to make your life easier.

### 6.1.1 Default Setup

The default settings provide similar behaviour to previous versions of Siril.

The monitor is set as sRGB and the preferred color space is also set as sRGB. This means that there is no transform required between them and therefore no processing overhead for images using the default profile.

When images are loaded, if they have an associated color profile that profile will be preserved. This does mean that a display transform is required and thus causes a small processing overhead, but minimizes change to the loaded file.

If images have no color profile, they are assigned the preferred color space (sRGB) when first stretched. This is the point at which you should be using the linear viewer mode to make the image appear on the screen the way you want it to look, and therefore is the point at which color management becomes important.

Images will be exported to all 8-bit and 16-bit formats using the sRGB color space.

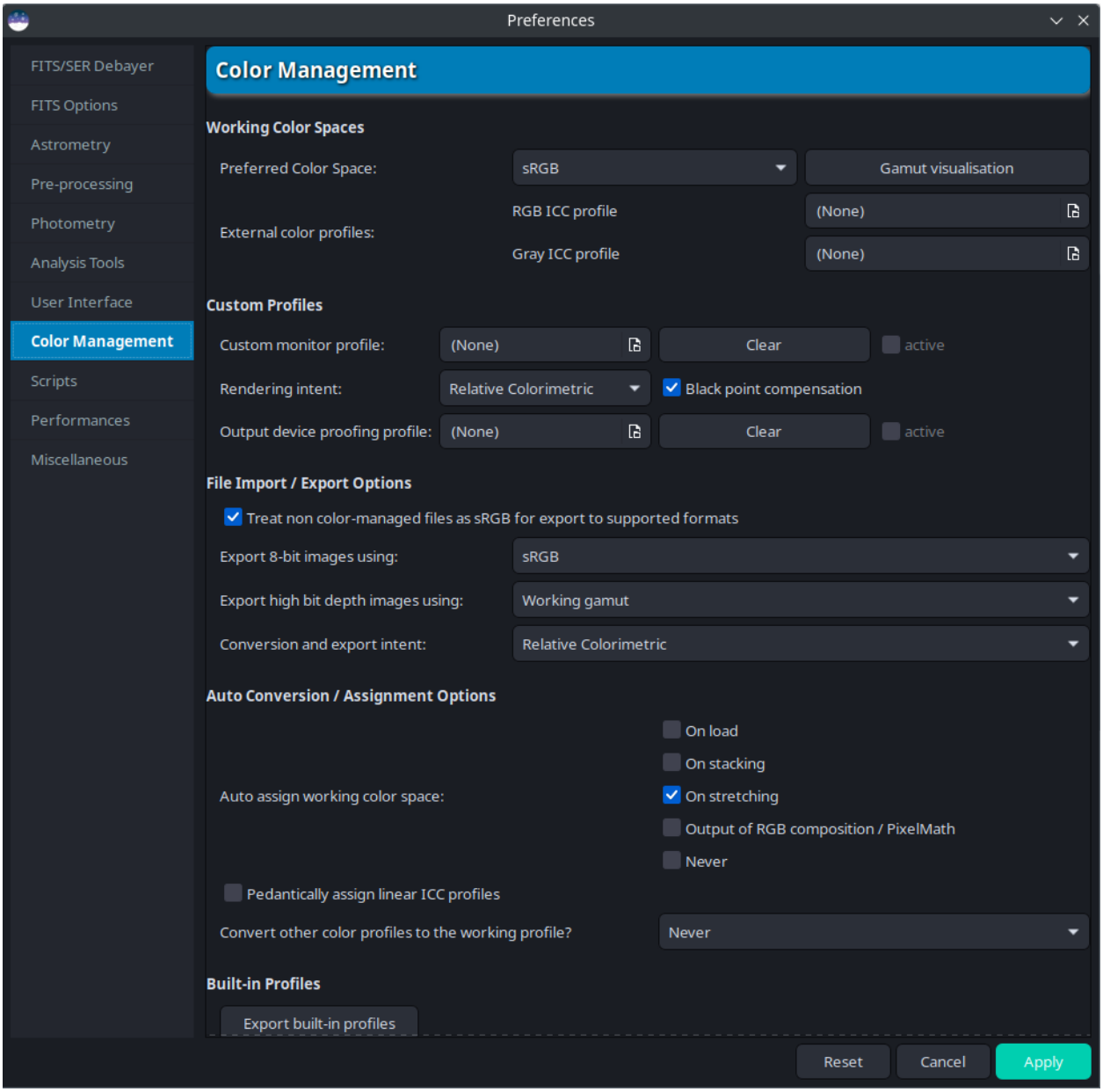
This default setup is perfectly fine if your main target is image production for the web, or for viewing on most typical computer monitors.

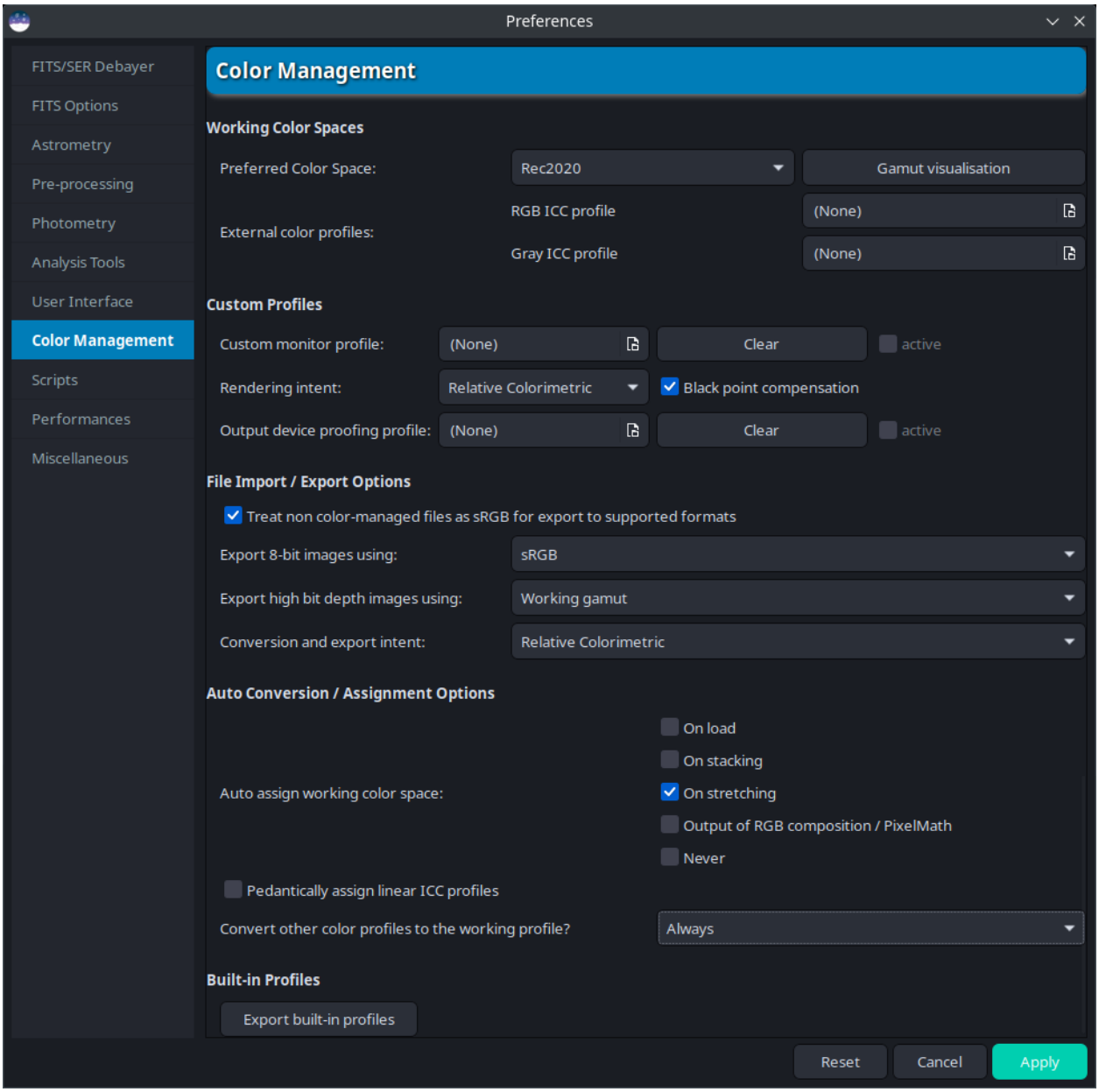
### 6.1.2 Photo Print Setup

If you intend to produce photo prints of your work, consider the following setup.

The monitor remains sRGB in the screenshot as no custom monitor profile is set, but if you have access to the necessary hardware to calibrate your monitor then you could set a custom monitor profile tailored to your display. (How to do this is outside the scope of Siril, but you may wish to look into the Free Software product ArgyllCMS.)

The preferred color space is Rec2020. This wide gamut color space is better suited to a workflow targeted at printed output as the RGB to CMYK conversion done by the printer is less likely to clip the CMYK profile.


Preferences
FITS/SER Debayer
FITS Options
Astrometry
Pre-processing
Photometry
Analysis Tools
User Interface
Color Management
Scripts
Performances
Miscellaneous
Color Management
Working Color Spaces
Preferred Color Space:
sRGB
Gamut visualisation
External color profiles:
RGB ICC profile
(None)
Gray ICC profile
(None)
Custom Profiles
Custom monitor profile:
(None)
Clear
active
Rendering intent:
Relative Colorimetric
Black point compensation
Output device proofing profile:
(None)
Clear
active
File Import / Export Options
Treat non color-managed files as sRGB for export to supported formats
Export 8-bit images using:
sRGB
Export high bit depth images using:
Working gamut
Conversion and export intent:
Relative Colorimetric
Auto Conversion / Assignment Options
Auto assign working color space:
On load
On stacking
On stretching
Output of RGB composition / PixelMath
Never
Pedantically assign linear ICC profiles
Convert other color profiles to the working profile?
Never
Built-in Profiles
Export built-in profiles
Reset
Cancel
Apply


Preferences
FITS/SER Debayer
FITS Options
Astrometry
Pre-processing
Photometry
Analysis Tools
User Interface
Color Management
Scripts
Performances
Miscellaneous
Color Management
Working Color Spaces
Preferred Color Space:
Rec2020
Gamut visualisation
External color profiles:
RGB ICC profile
(None)
Gray ICC profile
(None)
Custom Profiles
Custom monitor profile:
(None)
Clear
active
Rendering intent:
Relative Colorimetric
Black point compensation
Output device proofing profile:
(None)
Clear
active
File Import / Export Options
Treat non color-managed files as sRGB for export to supported formats
Export 8-bit images using:
sRGB
Export high bit depth images using:
Working gamut
Conversion and export intent:
Relative Colorimetric
Auto Conversion / Assignment Options
Auto assign working color space:
On load
On stacking
On stretching
Output of RGB composition / PixelMath
Never
Pedantically assign linear ICC profiles
Convert other color profiles to the working profile?
Always
Built-in Profiles
Export built-in profiles
Reset
Cancel
Apply

When images are loaded, if they have an associated color profile they will be converted to Rec2020. If images have no color profile, they are assigned the preferred profile (Rec2020) when first stretched.

### 6.1.3 Rendering Intent

In both these suggested setups the rendering intent is set to Relative Colorimetric. This is what you should always use for editing your image, as with Relative Colorimetric the in-gamut colors stay true and this provides the most accurate view of what your image looks like overall.

However the Relative Colorimetric intent clips out-of-gamut colors. You won't see details of coloration that are out-of-gamut.

You may therefore sometimes wish to change the rendering intent to Perceptual. This will smoothly remap colors from the working space to the display. All coloration detail is shown, but at the cost of a loss of overall saturation. This is unavoidable, as the larger gamut gets squashed to fit into the smaller gamut. Therefore the Perceptual intent does not provide a true visualisation of your image: to repeat, it should **not** be used for editing, **only** for exploring detail in your images that sits outside the sRGB gamut. See the discussion here.

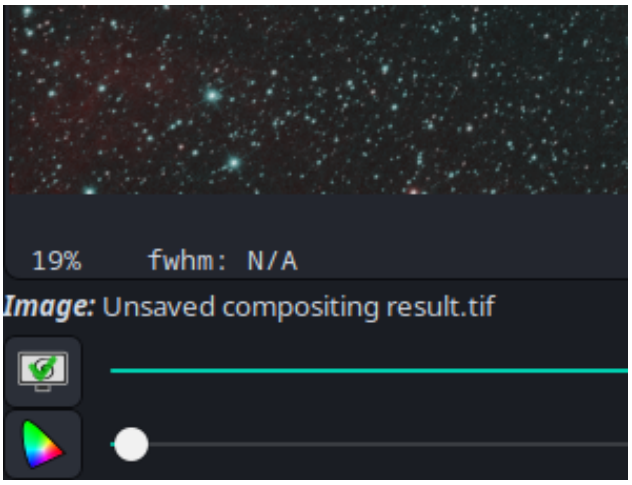
Images will be exported to 8-bit formats as sRGB, but to high bit depth formats as Rec2020 (and you should save your final version in a printing industry standard 16-bit format, i.e. probably TIFF).

## 6.2 Color Management Status

The color management status of the loaded image is indicated by the Color Management status icon in the bottom left hand corner of the window. The icon is fully saturated when the image is color managed, and desaturated when the image is not color managed. The tooltip provides more detailed status information if the image is color managed: it shows the working space and monitor profiles.

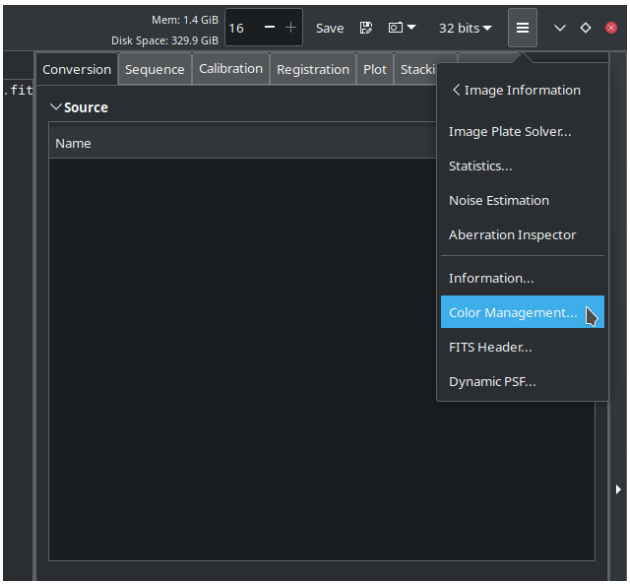
Left clicking the status button accesses the Color Management Dialog. Right clicking the tool toggles the ISO 12646 color assessment display mode. Both are described in more detail below.

Just above the color management status button is a new menu for image checks. This includes the existing cut check for highlighting pixels that exceed the gui hi slider, and adds a new gamut check, which highlights pixels that are out of gamut for the soft proofing profile in bright magenta.



## 6.3 Color Management Dialog

The Color Management tool is accessed by clicking on the Color Management status icon. (It can also be accessed via the *Tools* menu.)

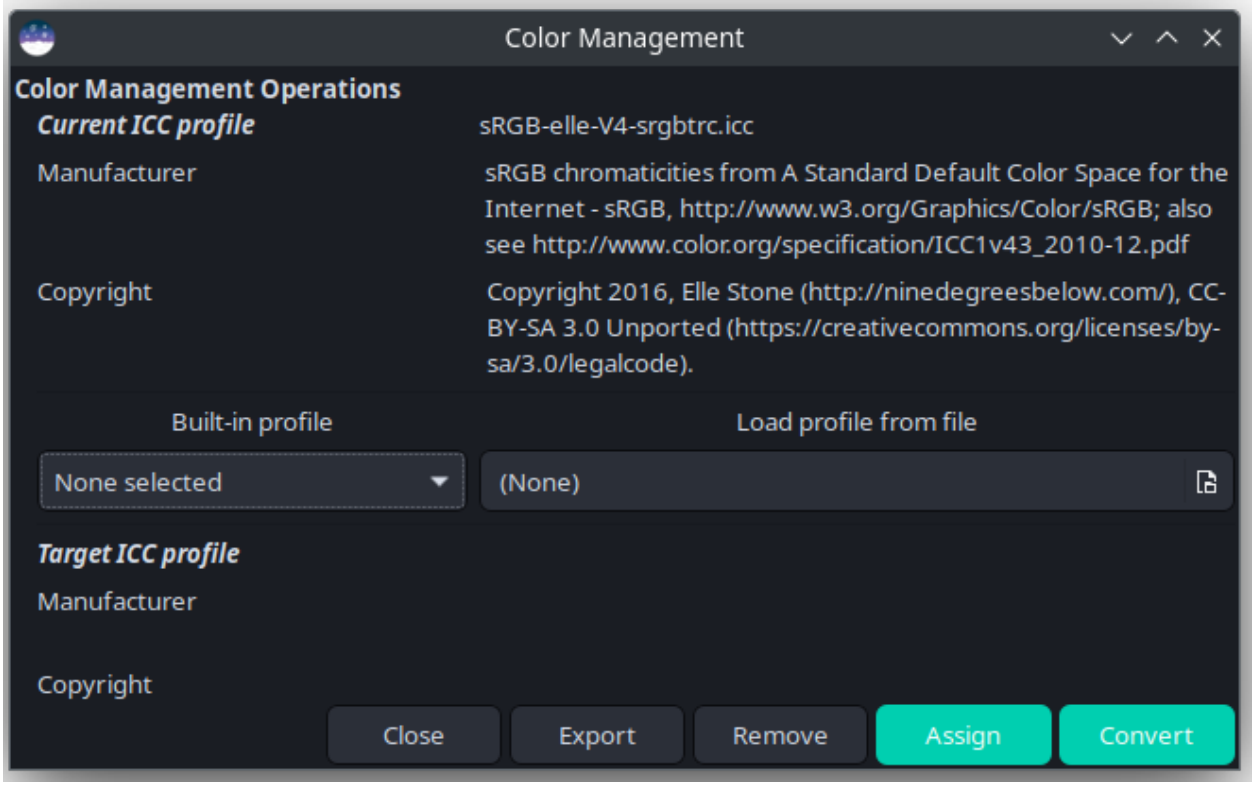



The tool itself is shown below.

The top part of the tool shows information about the current ICC profile assigned to the image. The description, manufacturer details and copyright notice are shown.

Underneath this are choosers that allow you to choose a different profile, either one of the built-in profiles using the drop-down menu on the left or any ICC profile you wish to load from a file using the filechooser on the right.

> **Warning**
>
> When loading a profile from a file, the profile type must be either RGB or Gray to match the loaded image. Assigning or converting images to other color spaces such as XYZ, CIE La*b* or CMYK is not supported, however color spaces such as CIE La*b* may be used internally by some image operations where required.




**Tip**

You can convert a RGB image to a Gray profile. This will result in a mono image with the appropriate tone response curve (TRC) transform and contributions from the 3 R, G and B channels determined by the white points of the two profiles. Converting a Gray image to a RGB profile will result in a 3-channel image with the appropriate TRC transform and all pixel components equal.

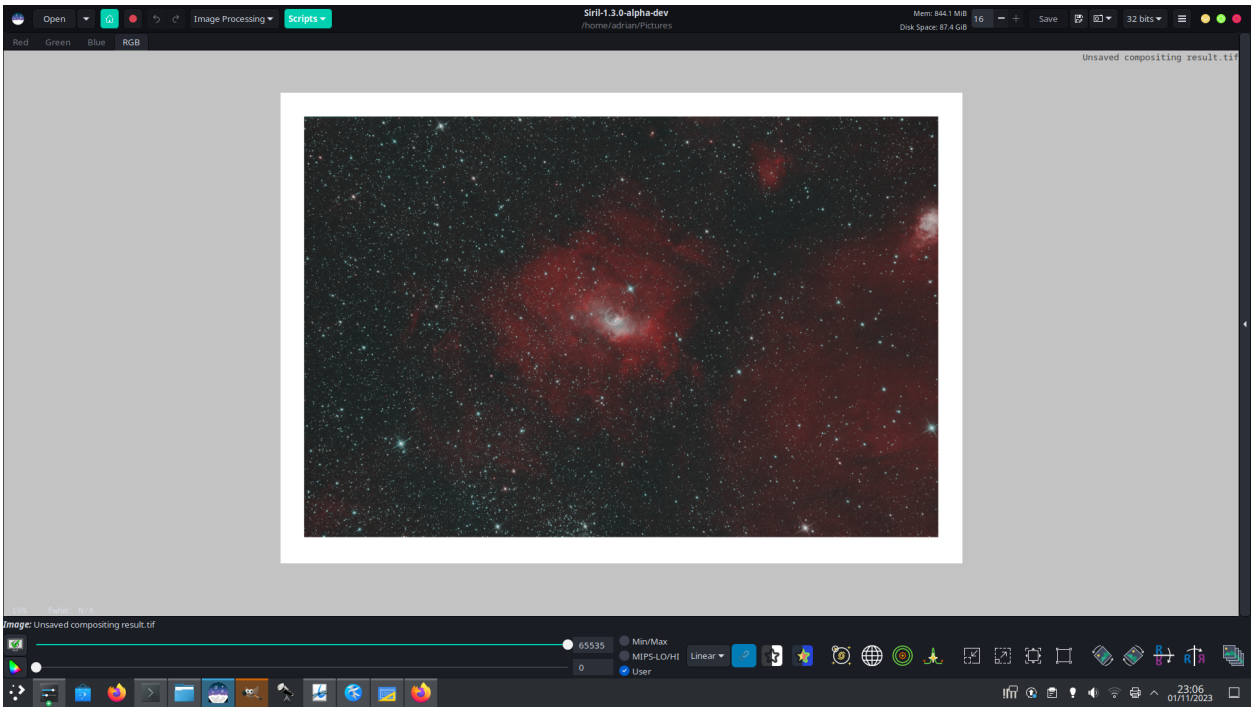
At the bottom of the window, the tool shows the same kind of information for the target profile that it does for the original profile at the top of the window. This helps in checking you've loaded the right one.

### 6.3.1 Toolbar

- The *Export* button simply exports the current image profile to a file in the working directory.
- The *Remove* button removes the ICC profile from an image.
- The *Assign* button assigns the selected ICC profile to the image, without performing a color space conversion. This is useful if the image has an incorrect color profile embedded.
- The *Convert* button converts the image to the selected ICC profile and assigns the profile to the image. This is useful if you wish to transform the image from its current color space to a different one.

**Warning**

Use of the *Remove*, *Assign* or *Convert* buttons (or the Siril commands *icc_remove*, *icc_assign* or *icc_convert_to*) will clear any ROI that is set.




## 6.4 ISO 12646 Image Assessment Mode

ISO 12646 defines optimum viewing conditions for assessing color. In summary, to get the best assessment of the colors in your image the best viewing conditions are provided by a uniform neutral gray background called D50. Technically this should even extend to the walls in your room, but Siril can't control that! Siril does provide an image assessment mode that approximates ISO 12646 recommendations, however. It is available by right clicking on the color management button in the bottom-left of the screen, and does not require the image to be color managed to use it.

This mode is intended as a final check of the colors of your image, so it sets the preview mode to linear with the sliders at 0 and 65535 (full range). It also hides the panel and sets the zoom so that the full image is visible and centered with an ample border around it, and sets a moderate white border around the image to provide a visual white reference, surrounded by a background of D50 gray.

Ideally you should also choose a neutral gray GTK theme, such as the excellent Equilux theme.

This mode automatically disables itself if the zoom is changed or the panel made visible again, and it can be toggled off by right clicking a second time on the color management button. The view will revert to its previous zoom settings and panel state.

## 6.5 Commands

Most Siril commands do not engage with color management except that they do not prevent existing color profiles from being preserved in an image.

The exceptions are three specific color management commands that can be used to assign, convert or delete the ICC profile in an image.

**Siril command line**

```
icc_assign profile
```

Assigns the ICC profile specified in the argument to the current image.
One of the following special arguments may be provided to use the respective built-in profiles: **sRGB**, **sRGBlinear**, **Rec2020**, **Rec2020linear**, **working** to set the working mono or RGB color profile, (for mono images only) **linear**, or the path to an ICC profile file may be provided. If a built-in profile is specified with a monochrome image loaded, the Gray profile with the corresponding TRC will be used

**Siril command line**

```
icc_convert_to profile [intent]
```

Converts the current image to the specified ICC profile.
One of the following special arguments may be provided to use the respective built-in profiles: **sRGB**, **sRGBlinear**, **Rec2020**, **Rec2020linear**, **graysrgb**, **grayrec2020**, **graylinear** or **working** to set the working mono or RGB color profile, (for mono images only) **linear**, or the path to an ICC profile file may be provided. If a built-in profile is specified with a monochrome image loaded, the Gray profile with the corresponding TRC will be used.

A second argument may be provided to specify the color transform intent: this should be one of **perceptual**, **relative** (for relative colorimetric), **saturation** or **absolute** (for absolute colorimetric)

**Siril command line**

```
icc_remove
```

Removes the ICC profile from the current image, if it has one

## 6.6 Color Conversion Matrices

### 6.6.1 Introduction

ICC profiles do not cover everything a user may wish to do in terms of color manipulation, so additional tools are provided as well. Of course, *pixel math* is a powerful general purpose tool for manipulating pixels but a common use case is to apply a color correction matrix to data, for example for manual conversion of camera chromaticities.

**Warning**

Application of CCMs is an advanced technique. Description of techniques involving the use of CCMs is outside the scope of Siril documentation. You should understand how CCMs operate and how to apply them within your workflow in order to use this tool successfully.

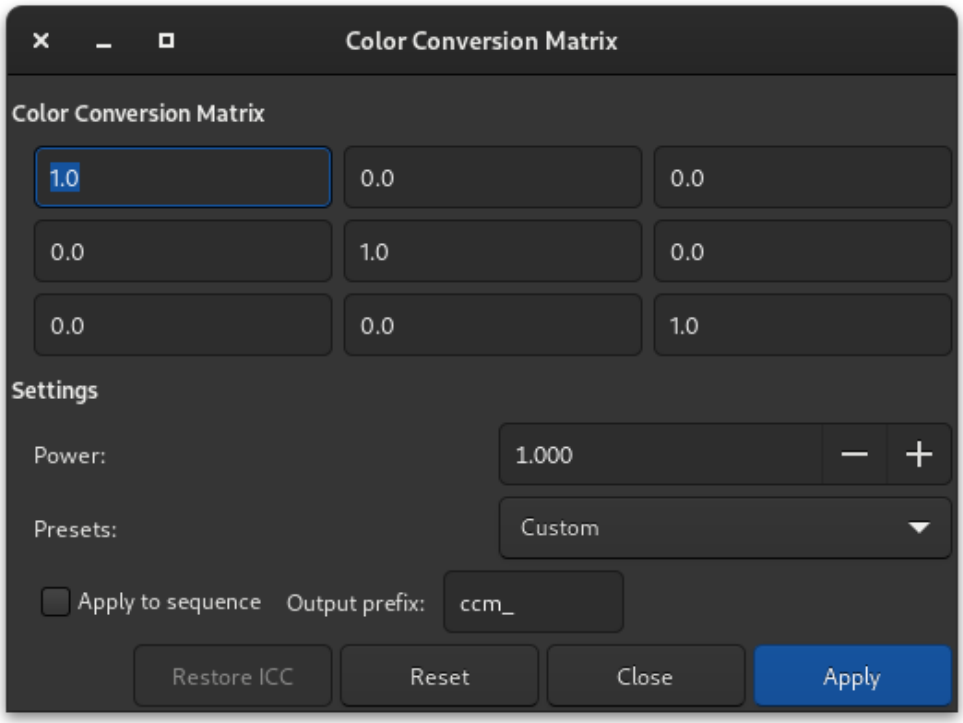



Fig. 1: Color Conversion Matrices dialog.

### 6.6.2 Color Conversion Matrix Tool

The *Tools* → *Color Conversion Matrix* tool allows direct application of a color conversion matrix (CCM) to pixels. The CCM is specified by 9 elements:

$$\begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

Several common preset are provided in a drop-down selector. The tool additionally offers the ability to scale by a $\gamma$ factor.

This is applied to pixels as follows:

$$r' = (m_{00} \cdot r + m_{01} \cdot g + m_{02} \cdot b)^{(-1/\gamma)} \tag{6.1}$$

$$g' = (m_{10} \cdot r + m_{11} \cdot g + m_{12} \cdot b)^{(-1/\gamma)} \tag{6.2}$$

$$b' = (m_{20} \cdot r + m_{21} \cdot g + m_{22} \cdot b)^{(-1/\gamma)} \tag{6.3}$$

**Warning**

If a CCM is applied to an image that has an embedded ICC profile, the ICC profile will no longer be a valid description of the image data. The profile is therefore temporarily disabled and the color management icon will show as inactive. It is assumed that your workflow will involve low-level colorspace transforms and image operations and at some point you will end up transforming the data back to the color space described by the ICC profile. At this point the ICC profile can be reactivated using the bottom of the dialog. If however your workflow involves manual conversion of the image to a different final color space you will need to apply the target ICC profile using the *Color Management* dialog.

Note that this does not apply to the command line *ccm* command. By policy, Siril commands do not interact with ICC profiles therefore the *ccm* command will not disable an ICC profile attached to an image: it is your responsibility to do this using the *icc_remove* command if required.

This operation can be applied to sequences. Open a sequence and prepare the settings you want to use, then check the *Apply to sequence* button and define the output prefix of the new sequence (*ccm_* by default).

**Siril command line**

```
ccm m00 m01 m02 m10 m11 m12 m20 m21 m22 [gamma]
```

Applies a color conversion matrix to the current image.

There are 9 mandatory arguments corresponding to the 9 matrix elements:

m00, m01, m02
m10, m11, m12
m20, m21, m22

An additional tenth argument **[gamma]** can be provided: if it is omitted, it defaults to 1.0.

These are applied to each pixel according to the following formulae:

r' = (m00 * r + m01 * g + m02 * b)^(-1/gamma)
g' = (m10 * r + m11 * g + m12 * b)^(-1/gamma)
b' = (m20 * r + m21 * g + m22 * b)^(-1/gamma)

**Siril command line**

```
seqccm sequencename [-prefix=]
```

Same command as CCM but for the the sequence **sequencename**. Only selected images in the sequence are processed.

The output sequence name starts with the prefix "ccm" unless otherwise specified with option **-prefix=**

Links: *ccm*

## 6.7 Color Management Theory

### 6.7.1 Introduction

Color management is the method used to ensure that the colors of an image always look consistent, however the image is viewed. This is done using color profiles (ICC profiles, named for the International Color Consortium). Each display

type and printer / paper combination has its specific color profile. The image also has a defined color space, and by transforming the image between different color spaces when viewed on two different types of display, or when printed, we ensure it always looks the same (or at least as close to "the same" as possible).

### 6.7.2 CIE 1931 Color Space

The CIE 1931 color space (CIE 1931) maps all colors that are discernable by the human eye. As the human eye has three types of cone cell (color receptors), so CIE 1931 has three parameters (X, Y and Z). Note the parameters X, Y and Z don't directly correspond to the response of each type of cone cell, but they do allow 3 degrees of freedom. For more details on this color space, see here: https://en.wikipedia.org/wiki/CIE_1931_color_space

This isn't a color space that is used to store image data most of the time but it matters because it is used as an intermediate color space, and because it defines what are "real" colors (those visible by human eyesight) and what are "imaginary" (those we can't see). CIE 1931 can be visualised as a horseshoe shape.

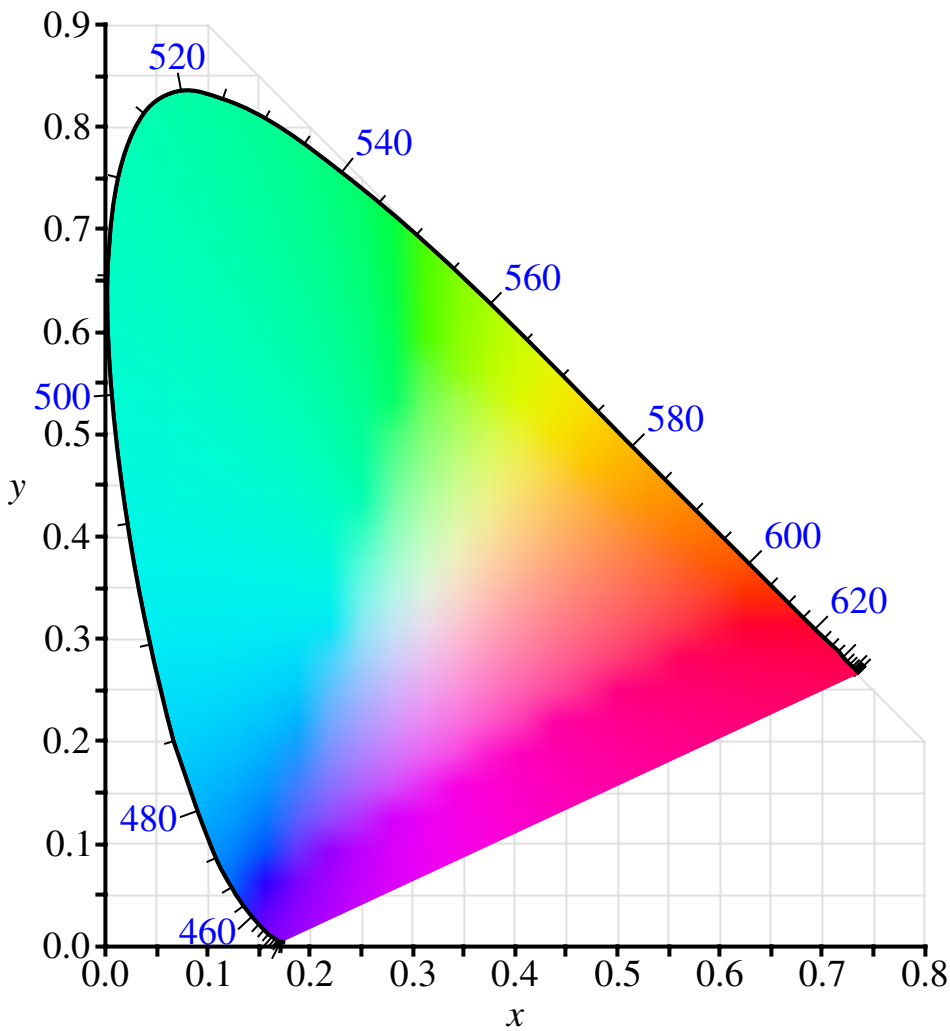



Fig. 2: By BenRG - CIExy1931.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=7889658

Note that the outside of the horseshoe defines the color of pure monochromatic spectral lines. This will be important later.

### 6.7.3 RGB Color Spaces

Our working color spaces are generally based around the color axes Red, Green and Blue. This corresponds roughly to how our eyes work and also to the phosphors in the emitters in a screen. However there are a great many RGB color spaces, each suitable for different purposes, and each with advantages and disadvantages. You can see the variety of color spaces in this diagram, which shows them defined in relation to CIE 1931.

The R, G and B channels of an image are defined in terms of a color space formed by three primary colours located within this CIE 1931 space, plus a white point, plus a tone response curve (TRC) for each channel that sets the gamma of each channel. *(Actually, TRCs can be more complex than a simple gamma curve but most are at least approximately a gamma curve.)* The diagram above shows that some color spaces are considerably bigger than others: that is, they can represent a greater portion of the visible colors.

### 6.7.4 Gray Color Spaces

In astrophotography we quite often deal with monochrome images. These clearly can't have a RGB color profile. Instead they get a Gray color profile. This defines the image's tone response curve (TRC) in exactly the same way as the RGB color profiles define the TRC of each channel. So whatever type of display or printer you're using, monochrome images can still be faithfully reproduced and look the same (or as close to it as possible) in different output media.

### 6.7.5 Display Color Space

The "standard" color space used for computer graphics has traditionally been sRGB. This was created by HP and Microsoft in 1996 and subsequently standardized as IEC 6166-2-1:1999. This color space codifies the color gamut achievable by displays of the time. As can be seen in the figure it can only represent a small portion of the total visible color space defined by CIE 1931. However, many monitors today can still do little better than sRGB and it is the current standard color space for the World Wide Web (WWW), so even if you don't want to use it for any other purpose it is necessary to use it as the export profile for any image you want to display consistently in web browsers. It is also the assumed color space for any image format that lacks color profile support, and for any application that lacks color management.

However it is obvious that there are other RGB color spaces with much larger gamuts, for example Adobe RGB, Adobe ProPhoto, Rec2020. These can represent a much larger portion of the real colors defined in CIE1931. They can also represent a much larger color gamut than most monitors. So why would these be of interest?

Well, for one, monitors are getting better. Wide gamut monitors are still not mainstream but they are becoming less rare, and more affordable. If you edit and view your images on a wide gamut monitor, you will benefit from richer color by using a color space with a bigger gamut. Some modern phones can display the whole of the P3 gamut, which is considerably bigger than sRGB and allows a richer range of colors to be displayed.

### 6.7.6 Print Profiles

Print profiles can extend beyond sRGB in some areas (and may not cover all of sRGB in other areas). By editing in a wide gamut color space, that richness of color - even if it can't be shown in the color-transformed output on your display - is still there and will be evident in good quality prints.

### 6.7.7 Transforms

The point of color spaces is that an image will often appear in outputs with different color spaces at different points in its life. It may be created on a professional-quality high gamut monitor, it may be viewed by the public on basic sRGB monitors, and it may be printed on a range of different printers. Each of these devices is capable of reproducing different color gamuts but we want the image, so far as possible, to look consistent across all of them. This is done by color space transforms. Unfortunately, because color spaces are different, we have to deal with the problem of how to represent in one color space colors that are "out of gamut" having been transformed from a different color space.

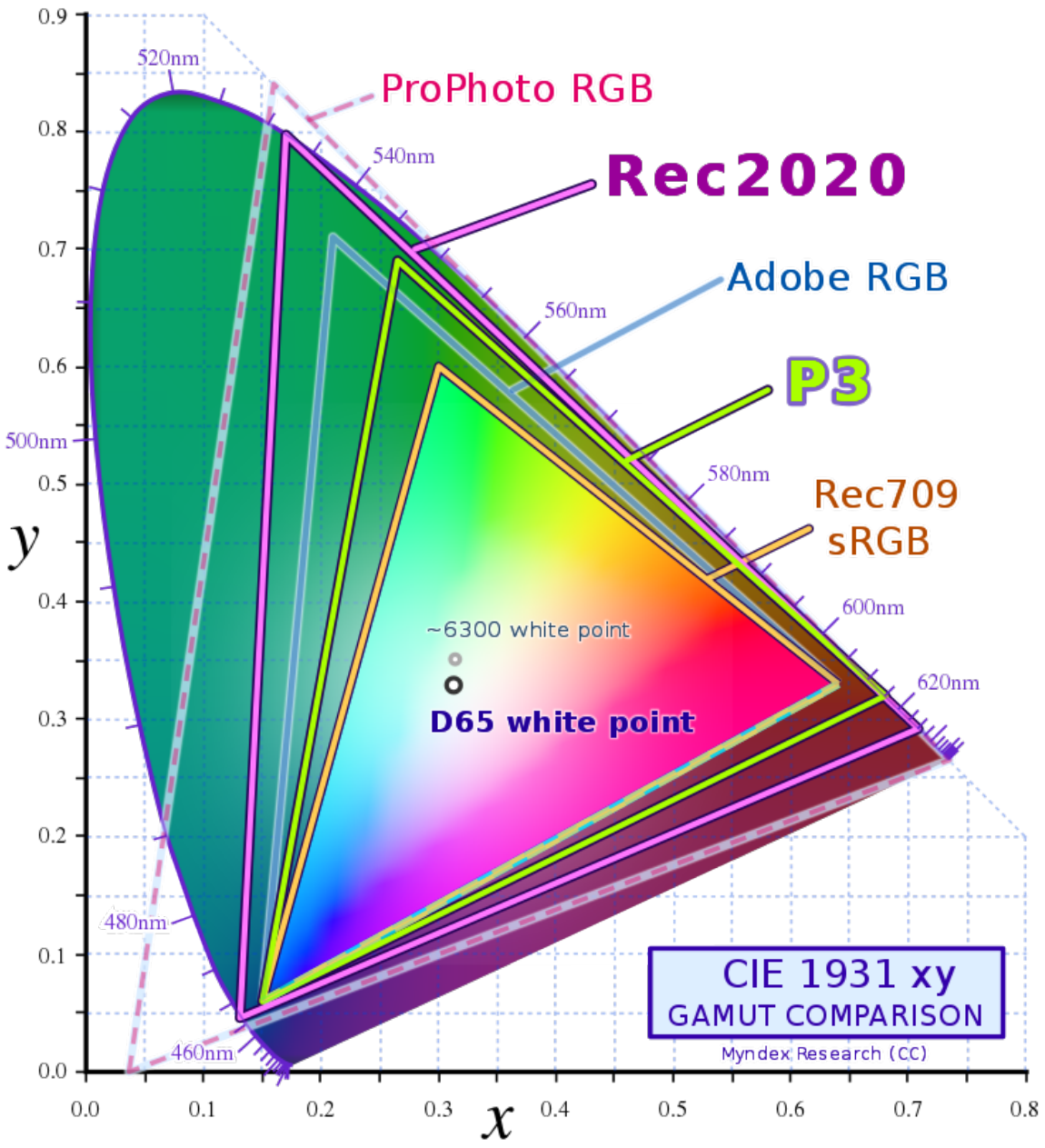



Fig. 3: By Myndex - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=116654642

### 6.7.8 Color Transform Intents

The answer to the problem above is "intents". Whenever you are viewing an image on a device / paper that has a different color space to your working profile it has a color space transform applied to it. This isn't as straightforward as an arbitrary 2-way mapping between two sets of coordinates. Say you're working in Rec2020. Consider the transform to your monitor color profile. Remember, your monitor (let's assume it is approximately a sRGB display) cannot display as many colors as are representable in Rec2020. So the color space has to map all the colors in Rec2020 onto colors in sRGB. How it does this is determined by the Rendering Intent.

You may want to choose one intent for rendering your image for display but a different intent for other purposes. The different intents defined by the ICC and available in Siril are described below.

**Tip**

- **Perceptual** The perceptual intent scales the input color gamut into the output color gamut. All colors are changed, but the relationships between colors are maintained. Generally, color is less saturated in the output color space, but the saturation compared with other colors is maintained.
- **Saturation** The saturation intent similarly scales the input color gamut into the output gamut, but it does so in a way that prioritises saturation. This is usually more suitable for vibrant graphics than for photography.
- **Relative Colorimetric** The relative colorimetric intent reproduces in-gamut colors accuately, however it clips colors that are out of gamut to the closest point on the triangle representing the boundaries of the target color profile.
- **Absolute Colorimetric** The absolute colorimetric intent is really only of use in pre-press proofing.

### 6.7.9 ICC Color Profiles and Intent Availability

The ICC defines the four intents listed above (as well as some others used primarily for ink control in press applications) but not all ICC profiles support all of the intents. Most of the color profiles built into Siril are matrix shaper profiles. These are very good as working color space profiles but mostly they only support the Relative Colorimetric intent. That's fine, because mostly that's the intent we want to use when converting between color spaces. (See below for an explanation of why.) If you set an intent in preferences that your ICC profile doesn't support, Siril will fall back to one that is supported - generally Relative Colorimetric.

### 6.7.10 Display

For display you generally want to use Relative Colorimetric. This makes the image display as consistent as possible with what anyone else with a color managed display will see, or what it will look like when printed to a color managed print workflow.

You *may* occasionally want to switch to Perceptual. This will not give you an accurate representation of color, but it will show the relative differences in color that get clipped when shown on your display. You can check what parts of the image are outside your display gamut using the gamut check tool in the image checking menu - the pixels that are outside the display gamut are shown in bright magenta. Siril provides a built-in sRGB profile with support for the Perceptual intent and will automatically use it if Perceptual intent is selected in the Preferences tab (and if no custom monitor profile is active). If you do wish to use a custom monitor profile and the Perceptual intent, it is up to you to ensure your profile supports this.

### 6.7.11 Conversion and Saving

For converting files and saving them you almost always should use Relative Colorimetric. This preserves color correctly. Relative Colorimetric does typically clip colors to the gamut of the color space it is converting to, but you get a consistent result. In fact, when dealing with 32 bit floating point images Siril uses an unbounded transform - that is, instead of

clipping colors it permits negative values. They will need clipping at some point - when displayed, for example - but they can be saved, and applying a color transform in the reverse direction brings back the original data. Nothing is lost.

### 6.7.12 Soft Proofing

For soft proofing you probably want Relative Colorimetric. Absolute Colorimetric may be of use in simulating exactly what your image will look like on a given printed medium as it simulates one medium's white point in another (so if you're proofing what an image will look like on the dingy yellowish-white of newsprint then Absolute Colorimetric will try to simulate that via your monitor), but for soft proofing a wider gamut color space against your display profile then you likely want Relative Colorimetric.

## 6.8 Workflow

### 6.8.1 Linear Stage

Almost all astrophotographic workflows start off with a linear stage. This is because our imaging sensors are photon counters, and provide essentially a linear response to incoming photons. Double the number of incoming photons during an exposure and the pixel value doubles (neglecting sensor bias, which will be calibrated out).

Many standard photographic workflows jump straight into color spaces with a nonlinear "tone response curve", for example the TRC of sRGB is approximately a gamma function $f(x) = x^{1/g}$, where $g = 2.2$.

However, astrophotography is different. Representing this linear data as linear values in a linear colorspace is vital for a number of reasons. Most importantly some functions rely on the data being linear: star detection matches Gaussian star profiles, and if the data has had a nonlinear TRC applied then the profiles will no longer be Gaussian, and the star finder algorithm will give worse results. StarNet has been trained on linear data with a specific transformation (histogram stretch with automatic parameters) and again, feeding it data that has already had a gamma function applied will make the results worse. Deconvolution, noise reduction... Many functions we apply to our data rely on it still being linear.

If you wish, you may apply a linear profile at this stage. Siril can do this automatically for you when an image is opened, if you set the auto-assign preferences and check the *Pedantically assign linear profiles* button. For monochrome images the linear Gray profile will be applied. For RGB images (or when compositing linear monochrome images into a color image), the profile selected will be the linear profile from the preferred set of color space profiles you set in *Preferences*.

For example if you chose the Rec2020 color space, Siril will assign the Rec2020-V4-g10 ICC profile to your image. "Rec2020" is the name of the gamut; "V4" indicates that this is a Version 4 ICC profile; and "g10" states that the profile has a TRC that is a gamma function with $g = 1.0$ (i.e. a linear response).

**Tip**

Assigning a basic working space profile is enough for most users. Color balance can be addressed later using the color calibration tools. However, if you have a calibrated camera you may wish to apply an input profile. Camera calibration and creation of camera profiles is an advanced topic and is outside the scope of Siril, but you may find the guidance in the articles at Section D of this page of use. The simplest way to apply an input profile is using the Color Management dialog. Select your camera profile from the file chooser and click *Assign*. This is actually enough: Siril will happily edit your image in the camera profile, but you can also select your working profile and click *Convert* to convert the data to your preferred color space.

Another approach to calibrating your input is by using Color Conversion Matrices (CCMs). This is more labor-intensive: as with camera profiles, the CCMs need to be produced outside Siril, and they can then be applied to your linear data using PixelMath.

What you have just done is assigned the color space primaries in CIE 1931 XYZ space, according to the points on the triangle representing the Rec2020 gamut. Although the data is linear, when you see the image on your screen it has the display color space transform applied to it, using your chosen rendering intent.

Having said that, it isn't really all that necessary to assign a color profile at this stage. You will quite likely be using the Autostretch viewer mode when working with linear data, which doesn't require color management anyway.

Now that's explained, you can happily get on with the rest of your linear workflow. Calibration, registration, stacking, star removal, noise reduction... Go at it!

### 6.8.2 Nonlinear Stage

When you're ready to stretch your image, it's time to think about your color space again. Stretching changes the image from linear data to non-linear data so that it looks pleasing to the human eye. You're going to make your data non-linear now, so before stretching is a good time to convert the image to your chosen nonlinear color space, be it sRGB or Rec2020 or another color space of your preference. You can either do it yourself manually, or you can set a preference for Siril either to prompt you to convert the image color space to your preferred color space, or to do it automatically.

You can now carry on and finish any post-stretch editing of your image.

### 6.8.3 Image Saving and Export

When the time comes to save your image, you have a choice of formats:

- FITS is the native image format for Siril, and ICC profiles can be embedded in FITS files. This is done in the same way as other astrophotography software, so images saved with embedded ICC profiles will be interoperable between Siril and PixInsight (and probably other software supporting FITS ICC profile embedding).
- TIFF and PNG are good formats to choose for high quality image export and offer wider compatibility with other software than FITS. Both support ICC profile embedding. Options in the *Preferences* dialog allow setting whether 8-bit and higher bit depth TIFF and PNG images are saved as sRGB, saved in the image's current ICC profile, or saved in the preferred color space set in preferences.
- The *de facto* standard for highly compressed images for the WWW is JPEG. Support for ICC profiles in JPEG images varies according to your JPEG library, but if you are using libjpegturbo version 2.0.0 or higher then ICC profiles will be embedded. Still, many third party programs do not support color management at all and they will assume JPEGs represent a image in the sRGB colorspace with the sRGB standard TRC. This is the default export color space for JPEGs and you should consider carefully before saving them with a different color profile.
- Siril does also support some other image formats: PNM / PGM, BMP etc. These formats are grouped together under the category of "do not support ICC profiles". When exporting to these formats, images will **always** be converted to sRGB.

Siril provides a preference setting for the export / conversion intent. In pretty much every single case you should leave this as "Relative Colorimetric".

## 6.9 Color Space Profiles

### 6.9.1 Built-in Color Profiles

Siril contains a basic set of color space profiles. These are taken from Elle L. Stone's excellent repository of free and well-behaved color profiles. Her website contains a wealth of technical information on color space theory and practice, and a great source of inspiration for applying color processing to images. All the profiles there are released under the Creative Commons Attribution-Share-Alike Unported license, version 3.0.

The built-in profiles are intended to provide a sufficient set of color spaces for most purposes: sRGB for web export and as a default display profile, Rec2020 as a wide gamut profile that looks like it may become the next standard for high quality wide gamut monitors, and of course a range of Gray profiles to match the RGB ones.

- sRGB
  - Linear profile sRGB_elle_V4_g10

  - sRGB TRC profile sRGB_elle_V4_srgbtrc
  - ICC sRGB Preference profile with perceptual lookup tables (for display only)
  - sRGB TRC profile sRGB_elle_V2_srgbtrc (for export)
- Rec2020
  - Linear profile Rec2020_elle_V4_g10
  - Rec709 TRC profile Rec2020_elle_V4_rec709
  - Rec709 TRC profile Rec2020_elle_V2_rec709 (for export)
- Gray
  - Linear profile Gray_elle_V4_g10
  - sRGB TRC profile Gray_elle_V4_srgbtrc
  - sRGB TRC profile Gray_elle_V2_srgbtrc (for export)
  - Rec709 TRC profile Gray_elle_V4_rec709
  - Rec709 TRC profile Gray_elle_V2_rec709 (for export)

Note that the native TRC versions of the profiles are provided in two versions (version 4 and version 2). Version 4 provides better functionality (especially for high bit depth images) including parametric TRCs which avoid quantization. However version 4 profiles are not universally supported yet. So when exporting a file for use in other software it is safest to embed a V2 profile.

The Gray profile is available with TRCs to match the two built-in RGB color spaces, and also with a linear TRC.

### 6.9.2 Linear TRC Profiles

Sensor data starts off as a linear representation of light, so why don't we assign a linear color profile to unstretched images such as a newly stacked stack? Well, technically perhaps we should. Certainly if you wish to do so you can, and nothing bad will happen. You can do this using the color management dialog, and there is also a preference (Pedantically Assign Linear ICC Profiles) that will assign a linearized version of the working ICC profile when a profile is auto-assigned to a newly loaded image (if it shows no sign of having been previously stretched), a stacked image or a newly composited image. However there is generally no benefit in doing so. To understand why, we need to understand a fundamental difference between astrophotography and normal photography processing.

In normal photography the whole image is well exposed, with darker shadow regions and lighter highlights. At the raw stage the image is converted from linear light to a working profile, but this is reversible and in fact is frequently reversed as many editing operations (color blending, noise reduction, etc.) are best done in linear light.

However in astrophotography, all the detail is very deep in the shadows and we have to apply a very strong stretch indeed in order to produce an image that looks pleasing. This stretch is much more extreme than the mild change of gamma from 1.0 to 2.2 going from linear light to sRGB, and **it is not easily reversible** - especially if multiple stretches are applied. So we do still need to do the same kind of operations (color correction, noise reduction, deconvolution etc.) on linear light (and a lot of other things that don't even exist in normal photography should be done to linear images too: star modelling, star removal etc.)

It isn't necessary to set a linear color profile to do these operations - algorithms like noise reduction etc. aren't even aware of color profiles. They just apply themselves to the provided data.

So it is safe to assign the color profile to an image at any stage of editing, though it usually makes most sense to do it just before stretching. The key thing to remember is that with astrophotography, **all the operations that need to be done on linear images should be done before any stretching**. Generally, stretching and any final tweaks to color balance, saturation etc. should be just about the last editing operation that is done to an image. And at that point you should be editing it in your chosen color space, which will give as consistent as possible a look to the image when viewed in any color managed application or output device.

### 6.9.3 Color Space Recommendations

This section explains in a bit more detail why Siril offers the built-in color spaces it does.

sRGB is the *de facto* standard for the web. As mentioned above, if you want to guarantee your images look good in all web browsers, as well as all third party applications that may or may not support color management, you really need to export in sRGB. It's a good fit for most SDR monitors, and in the absence of proper color management images that looked right in previous versions of Siril should be considered to be in sRGB.

However, consider the CIE 1931 color space again. The horseshoe shape forming the boundary of the color space represents *pure spectral tones*. If you have a perfectly monochromatic light source like a laser or one of those sodium yellow street lights, you're looking at the boundary of CIE 1931. As astrophotographers these pure monochromatic sources are actually really important to us, especially if you do narrowband imaging. Those colors can't be represented accurately in sRGB. The proportion of visible colors that can be represented in sRGB is actually rather small. We can do better.

So, if a wide gamut is better why is the wide gamut color space built into Siril Rec2020 rather than an even wider one? AllColorsRGB, ACES 2065 and ProPhoto RGB are all much bigger - ACES 2065 can represent all parts of the visible spectrum.

The problem is, the way they do that is by setting their primary colors - the "100% red", "100% green" and "100% blue" values - *outside* the visible range. This is problematic especially for narrowband imagers who want to assign filters to primary colors: you end up with imaginary colors in your image and always have to rely on the intent of a color space transform to do the right thing in turning them into something visible. Also some of these color spaces are linear, such as ACES 2065. This is good for CGI artists but not so good for us, because of the slow display transforms when working with linear spaces. Siril can optimise some transforms from linear color spaces over and above what lcms2 does, but only if the RGB primaries are the same, e.g. if transforming linear sRGB to g22 sRGB). So Rec2020 was chosen as it provides a nonlinear color space with the widest gamut without having imaginary primaries.

My recommendation, if you aren't already sold on a particular color space, is to try Rec2020 as a color space for editing, as well as for sending images to high-quality printing services that can cope with color managed images, and sRGB for web export. But there are also other good choices available using ICC profile files that you can use instead, such as Adobe RGB and ProPhoto RGB (ROMM), if you prefer.

### 6.9.4 Third Party Color Profiles

**Monitor Profiles** Siril includes the ICC's v4 sRGB monitor profile with support for Perceptual rendering LUTs and uses this by default. A different monitor profile may be used in its place, but note that only the intents supported by a given profile will be available - many widely available sRGB profiles only support the Relative Colorimetric intent.

**Soft Proofing Profiles** There are a wide variety of press standards and papers in the world and each combination requires its own ICC profile. Siril cannot provide all of these. Therefore in order to use the soft proofing display mode you will need to provide the appropriate soft proofing ICC profile in the Preferences window.

You may wish to work in a color space other than the inbuilt ones, for example ProPhoto RGB. This is supported, but you will need to provide ICC profiles yourself. The Preferences window provides controls to set a RGB profile and a Gray profile with a matching TRC (essentially, with the same gamma). If you intend to export files in your chosen color space it is recommended that the profile you provide is a V2 ICC profile, for maximum compatibility with other software.

### 6.9.5 HDR Display Limitations

Wide gamut display color spaces such as Rec2100 with HLG or PQ TRCs may need greater than 8 bit pixel buffers to display smoothly. Unfortunately, Siril uses the Cairo graphics library for display and Cairo cannot yet process pixel buffers wider than 8 bits. The impact of this is likely negligible for most users. However, if you are lucky enough to be using a >1000nit HDR display capable of displaying wide gamut color spaces and are using certain combinations of third party image and custom monitor profiles then for certain images, you may experience minor color staircasing

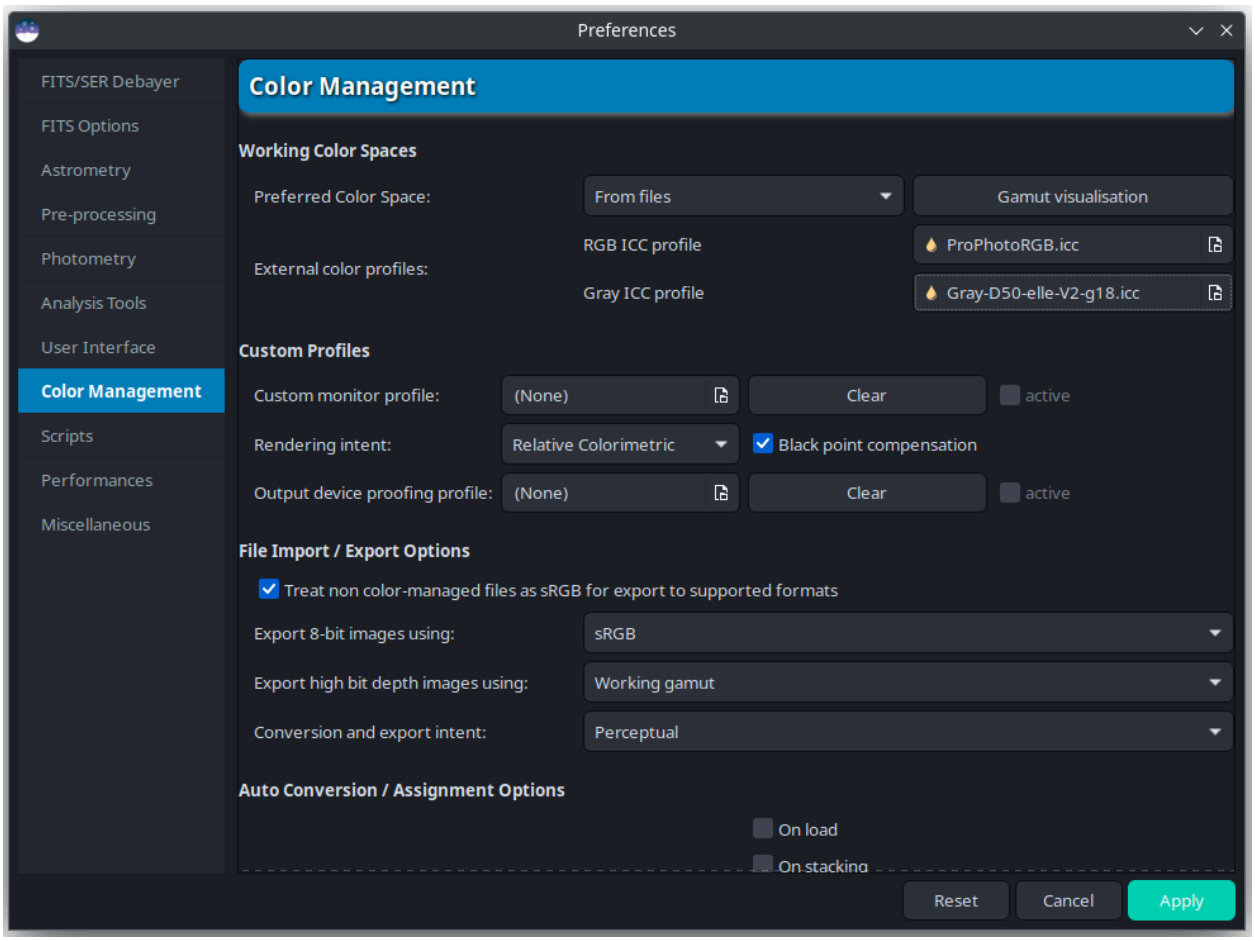



Fig. 4: The image shows how to set up a ProPhoto working space using Elle Stone's profiles. The standard gamma of ProPhoto RGB (ROMM) is 1.8, so as well as the ProPhotoRGB profile, we add the Elle Stone Gray profile with gamma = 1.8. You can download all of Elle Stone's profiles here. If you don't have the ProPhotoRGB profile, you can also use Elle Stone's "LargeRGB-elle-v2-g18" profile, which is exactly the same except she has avoided use of the term "ProPhoto RGB" for possible copyright reasons.

artefacts. These are only a display issue and won't affect the printed appearance of your image, or even its appearance when transformed to a narrower gamut or viewed on operating systems that do support high bit depth pixelbuffers.

For the moment there isn't anything we can do, however if Cairo adds high bit depth support in the future then there may be room for improvement.

## 6.10 Soft Proofing

### 6.10.1 Display Soft Proofing

Siril supports soft proofing. This provides a visualisation in your monitor's color space of what your image will look like when printed on a specified press standard. To use output device soft proofing you must specify a soft proofing profile in the Settings dialog.

If no output device profile is specified, or if it is made inactive using the preference check box, soft proofing will use the display profile as the proofing target. This enables the *gamut* image check in the Image Checks menu, which shows the pixels that are outside your display's color gamut. When gamut checking is active, out-of-gamut pixels will be shown in garish magenta.

### 6.10.2 Press Export

Siril does not support exporting in CMYK formats for print / press purposes. Most photo printing services expect images to be provided in RGB formats and printer drivers do a much better job of converting RGB to a blend of the specific inks used by each printer than a simplistic application-level RGB-to-CMYK conversion. Unless you know exactly why you need to convert an imge to CMYK, you almost certainly don't need to. However, in order to minimize gamut clipping when your printing service converts the image to a CMYK profile for printing, it is worth using a wide gamut RGB profile such as the built-in Rec2020.

## 6.11 Changes caused by introduction of color management

Siril is now doing color management properly.

That means it wasn't really doing it properly in the past. Siril wasn't unusual in this regard: many programs do not provide color management, but rely on assumptions that everything is done in an sRGB color space. This is fine most of the time, but it led to reports of issues when interchanging images between Siril and properly color-managed graphics applications.

Some users reported that in certain circumstances, images saved by Siril looked different in other software than it did in Siril. For images exported from the new color-managed Siril, that should stop. Images will have a consistent look in all software with a correct color management implementation.

However, if you have stretched images saved as FITS in a previous version of Siril and you open them in color-managed Siril, they may in some circumstances look pale and washed out. Don't panic! Here is an explanation of why it happens, and how it can very easily be avoided / fixed.

### 6.11.1 Why does it happen?

When Siril reads a FITS image without an embedded ICC profile, if the "Auto assign ICC profile" preferences are set it tries to work out whether the image has been stretched or not. It can only do that by checking the HISTORY entries in the FITS header. If it can detect stretch functions have been applied, then it assumes the image has been stretched on a sRGB monitor and applies the relevant profile. However older versions of Siril didn't always add the history for every command, and other software may or may not add history but Siril can't recognise all the stretch keywords that might be used by all other software, so it may be unable to detect that an image has been stretched. The fallback assumption we choose to make is that a FITS image without an embedded ICC profile is **linear** data.

However, your stretched FITS isn't really linear data. It has been stretched, and you have in effect been stretching it so that it looks right in your monitor's color space, which is probably approximately sRGB.

When this has a linear ICC profile assigned to it, the display routines then apply a color space transform from linear space to the display profile, in effect stretching it a second time.

### 6.11.2 How to prevent it?

This problem will only occur if you have the option "Auto assign preferred color space on load" enabled. If this option is disabled, Siril will not assign any color profile when the file is loaded. You can then assign one manually using the Color Management dialog.

### 6.11.3 How to fix it?

If you loaded an image with the Auto assign option enabled and the wrong profile is assigned (or if the wrong profile is assigned for any other reason, for example if you loaded a fle saved by other software that has assigned a wrong profile) fixing the problem is also very easy. All you need to do is open the image, go to the Color Management dialog and select "sRGB (standard sRGB TRC)" from the drop down selection (or "Gray (sRGB TRC)" if the image is mono). Click the *Assign* button (**not** *Convert to*) and Siril will assign the selected color profile to your image.

That's it. The display will now look correct and when you save the image, and the appropriate color profile will be embedded.

CHAPTER

SEVEN

# SEQUENCES

Sequences are what Siril uses to represent a set of manipulated files, for example the set of dark images that we'll turn into the master dark. It is a very useful tool for handling a large number of files that need to be linked with each other.

## 7.1 A set of two or more FITS files

Siril uses natively 32-bit floating point data or 16-bit unsigned integer data for the *FITS* images, other formats are automatically converted. To be recognized and detected as a sequence, FITS images file names must respect a particular pattern which is:

```
basename$i.[ext]
```

- **basename** can be anything using ascii characteres. It is usually convenient, but not mandatory, to have it end with the _ character. It will be used as the sequence name.
- **$i** is the index of the image. It must be a positive number and can have several leading zeros.
- **[ext]** is the supported extension as explained in the *settings*, `fit` by default.

> **Note**
>
> The extension used to detect FITS sequences in the current working directory will be the same as the extension configured in the settings and as the files created by Siril.

> **Warning**
>
> Some operating systems limit the number of images that can be opened at the same time, which is required for median or mean stacking methods. For Windows, the limit is 2048 images. If you have a lot of images, you should use another type of sequence, described below.

## 7.2 A single SER file

*SER* is a format meant to contain an acquisition sequence of several contiguous images in a single file. It is a rather simple format that cannot contain as much metadata as FITS, but more than simple films and data is not compressed. SER files can contain images of 8 or 16-bits per channel only. There are three types of SER files, depending on the pixel content: monochrome, CFA or color (3 channels).

> **Note**
>
> An SER file can be opened either via *File* and *Open*, or with the *Search sequences* button.

See *SER* for more information on the SER format and why film formats like uncompressed AVI should not be used for astronomy.

> **Warning**
>
> To some extent, a regular film file such as AVI or any other container are supported too. Film files support is being dropped in favour of SER, but it can still be useful to open a film in Siril, to explore its content, extract some frames or convert them. A few operations can still be done, but in a slower way than with other sequences, like sum stacking. For a complete processing you will face limitations and incompatibilities.

## 7.3 A single FITS file

> **Note**
>
> Also called FITS cubes or FITS sequences, or FITSEQ for short in Siril.

The FITS format is an image and science data container, it can contain several of these in a single file. We can use that to store an entire sequence of FITS images in a single file while preserving the FITS header of each image. It is the file format that professional astronomers use.

It's simpler to manage one file on the disk than 2000, but since it is a single file, some operations on single images of the sequence may not be possible. In particular, it is not currently supported within Siril to change the header of a single image.

This format is an alternative to SER for a single-file sequence, with 32 bits per channel and full header support.

## 7.4 Loading a sequence

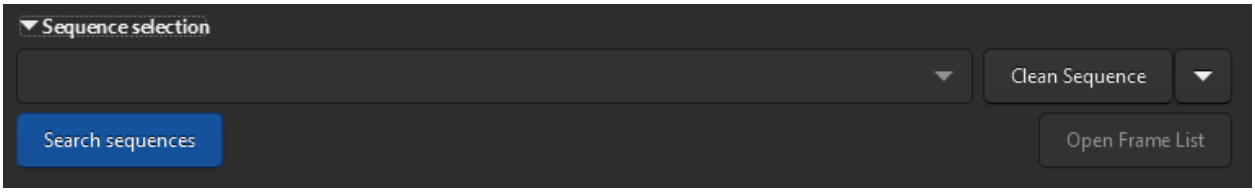



Fig. 1: Sequence Search and Cleaning.

When the working directory is set in the right place, the FITS follow the correct nomenclature, and the extension of the FITS files is also set correctly, then click the *Search Sequence* button of the Sequence tab. A drop-down list opens with all the sequences available in the folder. If only one is found, then it is automatically selected and loaded.

## 7.5 Frame selector

A great strength of Siril is that it easily manipulates image sequences. When a sequence is opened, the reference image (see below) will be displayed, by default this is the first image. However, it can sometimes be useful to inspect individual images of a sequence. This is possible with the frame selector, available via the toolbar with the button or via the sequence tab with the *Open Frame List* button.

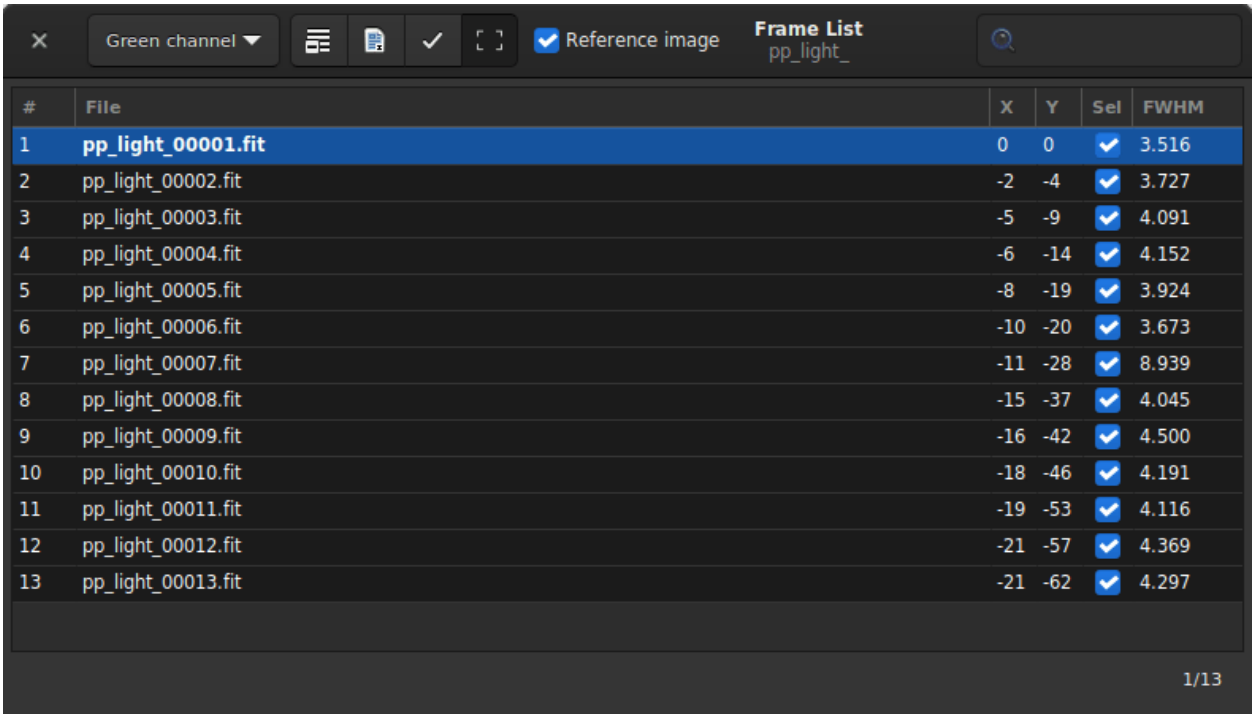



Fig. 2: Frame selector that allows you to choose a frame from the sequence and display it, set it as reference or exclude it.

> **Tip**
>
> Hovering over an image in the frame selector displays the original file name, if saved.

Clicking on an image in the list will load it and display it in the main area, while keeping the sequence as the active object for processing. More than just a image display selector, the tool can also be used to **manually exclude images** from the sequence, or visualize which are still included, visualize the values of image quality and shift between images if they have been computed, and change the reference image. Note that more image quality information can be viewed in the Plot tab.

Excluding an image from the sequence does not mean its data will be permanently deleted, it will just not be used for the subsequent processing operations, if instructed to do so. In most cases, the option to look for is called *Process included images only*.

The **reference image** is the image in the sequence that will serve as target for the registration and for the normalization. Other images will be transformed to look like the reference image, so it should be chosen carefully. Fortunately, since Siril 1.2, a new two-pass registration can automatically select the best image of the sequence as reference image before proceeding to image transformation.

The header bar of the window will provide many controls for these sequence properties:

- The drop-down menu allows the channel for which registration data (quality, shifts) is displayed to be changed, if they exist for other channels.
- The first button of the toolbar sets all images of the sequence as manually excluded.
- The second one, sets them all as included.
- The third includes or excludes the images selected from the list (multiple selections can be done with `Ctrl` or `Shift`) of the sequence.
- The last button can be deactivated to hide the framing indicator over registered images.

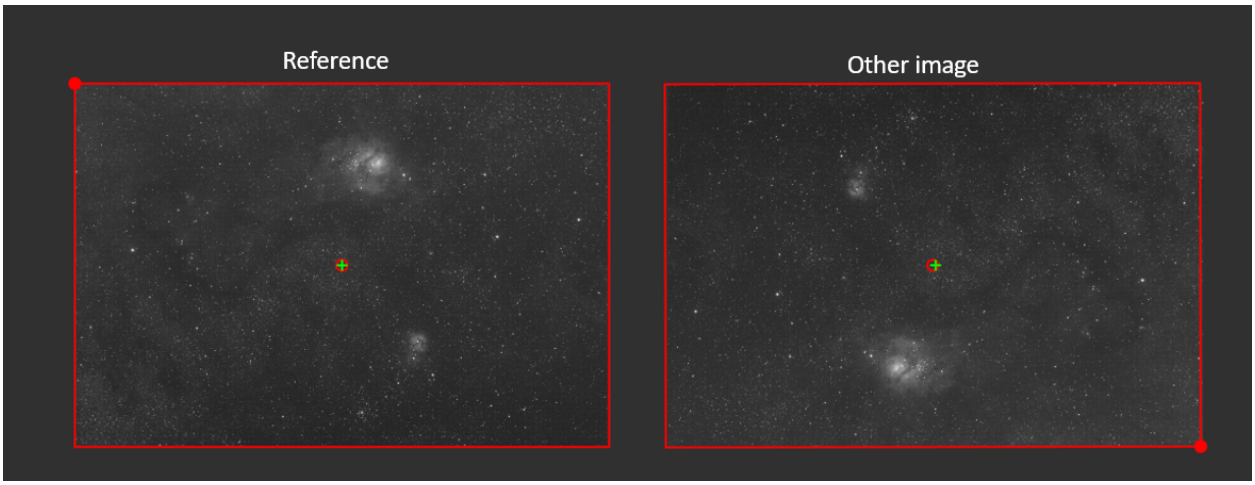



Fig. 3: Framing indicator

The red frame shows the trace of the reference frame over each image of the sequence. The filled circle in one corner indicates the position of the top-left corner of the reference image. For instance, for the right image above, the framing indicator circle is at the bottom right, showing that there has been a meridian flip between this image and the reference shown on the left.
The red circle/green cross close to the center show the position of the center of the reference image and of the current image respectively. This enables to visualize the shift between the two images.

**Note**

Since Siril 1.3.1, the color of the framing indicator is different whether the registration information is just pure shift (in blue) or more *complex transformations* (in red for linear transformations, in green when distortions are included). For previous versions, it was always red, irrespective of the transformation.

- The button *Reference image* is used to select the reference image for the sequence. All sequences must have one, it will be the first image if unset or by default.
- Finally, the search field allows you to find the images by name.

It is also possible to sort all the images by clicking on the column headers. Thus you can sort the images by their number or their FWHM. The latter is very useful to have a look at the best and worst images.

### 7.5.1 Sequence manipulation commands

**Siril command line**

```
setref sequencename image_number
```

Sets the reference image of the sequence given in first argument. **image_number** is the sequential number of the image in the sequence, not the number in the filename, starting at 1

**Siril command line**

```
select sequencename from to
```

This command allows easy mass selection of images in the sequence **sequencename** (from **from** to **to** included). This is a selection for later processing.
See also UNSELECT

Links: *unselect*

**Siril command line**

```
unselect sequencename from to
```

Allows easy mass unselection of images in the sequence **sequencename** (from **from** to **to** included). See SELECT

Links: *select*

## 7.6 Sequence export

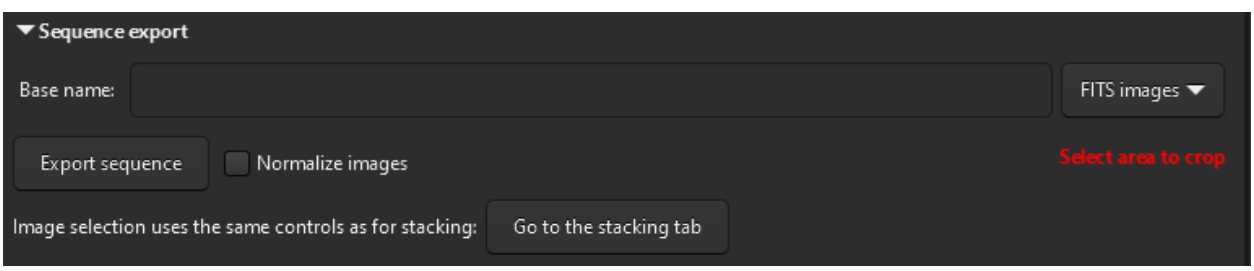



Fig. 4: Give a name and specify output format to export a sequence.

The **Sequence Export** tool allows you to export a sequence of images in a variety of formats. It is particularly useful if you want to export the images taking into account the registration information contained in the `seq` file, with optional cropping and normalization.

With the sequence export function, you can select a sequence to export, choose the file format and compression level for video formats. Siril's sequence export function supports a wide range of image file formats, including *FITS* (single FITS file or sequence FITS file), *TIFF*, *SER* , AVI, MP4 and WEBM and can come in handy when building timelapse.

The button *Normalize images* allows you to normalize the images with respect to the reference image. The normalization is the same as the one done *during the stacking*, with the following settings: *Additive with scaling*, *Faster normalization* disabled.

Moreover, it is possible to play with the *image filtering* criteria to exclude or not images according to their quality. A button *Go to the stacking tab* has been added here, to easily go to the tab that exposes them.

## 7.7 Sequence information

All sequence information, registration transformation, statistics and frame selection are stored in a `.seq` file saved next to the sequence files. It is strongly recommended never to edit this file manually because Siril writes continuously inside it and one wrong character could make the reading of the sequence corrupt.

One way to clean the content of this sequence file is to go in the **Sequence** tab and click on *Clean Sequence*. The choice of what will be cleaned can be defined by clicking on the small arrow next to it.

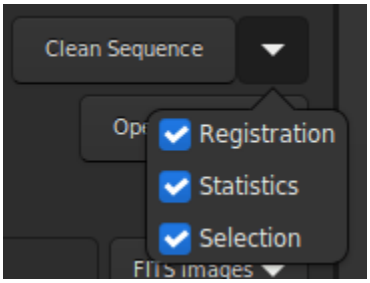



Fig. 5: Menu to clean the sequence file.

**Siril command line**

```
seqclean sequencename [-reg] [-stat] [-sel]
```

This command clears selection, registration and/or statistics data stored for the sequence **sequencename**.

You can specify to clear only registration, statistics and/or selection with **-reg**, **-stat** and **-sel** options respectively. All are cleared if no option is passed

CHAPTER

EIGHT

# DEFINITIONS AND WORKFLOW

Astrophotography is the process of capturing images of celestial objects. It involves several steps, including preprocessing and processing, which are distinct but related.

*Preprocessing* is the initial step of working with raw astrophotographic data. It involves preparing these data for further processing. This step typically involves dark current subtraction, flat field correction, and correction of other basic problems such as removing hot and cold pixels.

*Processing* refers to the post-processing of the preprocessed data, generally after stacking. This is where the astrophotographer applies various techniques to enhance the final image and bring out details and features. These may include sharpening (deconvolution), color calibration, noise reduction, and stretching the image to increase the visibility of faint details.

In short, preprocessing sets the stage for processing by ensuring that the data is in an appropriate form and cleaned of unwanted signal, while processing is about bringing out the best in the signal to create the final image. Both steps are important in the astrophotography process, and the quality of the result depends on the skills and techniques applied at both stages.

In Siril, the main preprocessing is done following the order of the tabs in the right pane and requires the use of master files. This is a process that can be automated quite easily and the scripts provided in Siril perform this task. The image processing is processed via the dedicated menu *Image Processing*. This process is more difficult to automate because it is specific to each image and consists of an iterative work.

CHAPTER

NINE

# PREPROCESSING

This section takes you through the different steps of pre-processing your images, from import into Siril to obtaining a stacked image.

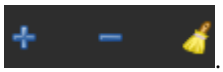
The right pane contains the tabs that are useful during preprocessing. They have been designed to be used from left to right throughout the process, with some exception for the creation of masters. These tabs are also accessible via the keys `F1` to `F7`.

Pre-processing is the step that starts with the conversion to the stacking of the images. The goal is to remove all unwanted signals and to reduce the noise present on all the subs.

## 9.1 Conversion

Siril supports the FITS 32bits format as well as the SER format in a native way. Therefore, any other file format must first be converted to these formats in order to be supported and to generate a *sequence*. The type of supported files is indicated in the tab and depends on how Siril was compiled.

Siril provides a conversion tab which is divided into 2 panels. The upper panel allows you to load the **source** files you wish to convert.

The management of these files is done from the mini toolbar  .

- The first button, the + button, is the one that allows to load all the source files. It opens a dialog window allowing you to choose all the files to be converted on your computer. Only the formats supported by Siril are visible.

> **Tip**
>
> It is possible to drag and drop files directly into the **sources** area. The drop zone is highlighted when the files are over it.

- The second button, the - button, allows to delete the selected files. Several files can be deleted at the same time. They are not deleted from the hard disk, but only from the conversion area.
- The last button allows you to delete all loaded files at once.

The number of loaded and selected files are reported in the status bar, to the right of the toolbar.

In the **destination** section it is possible to choose the name of the sequence that will be generated after the conversion of the files.

Thus, for a sequence name `basename`, the converted files will be of the form

```
basenameXXXXX.[ext]
```

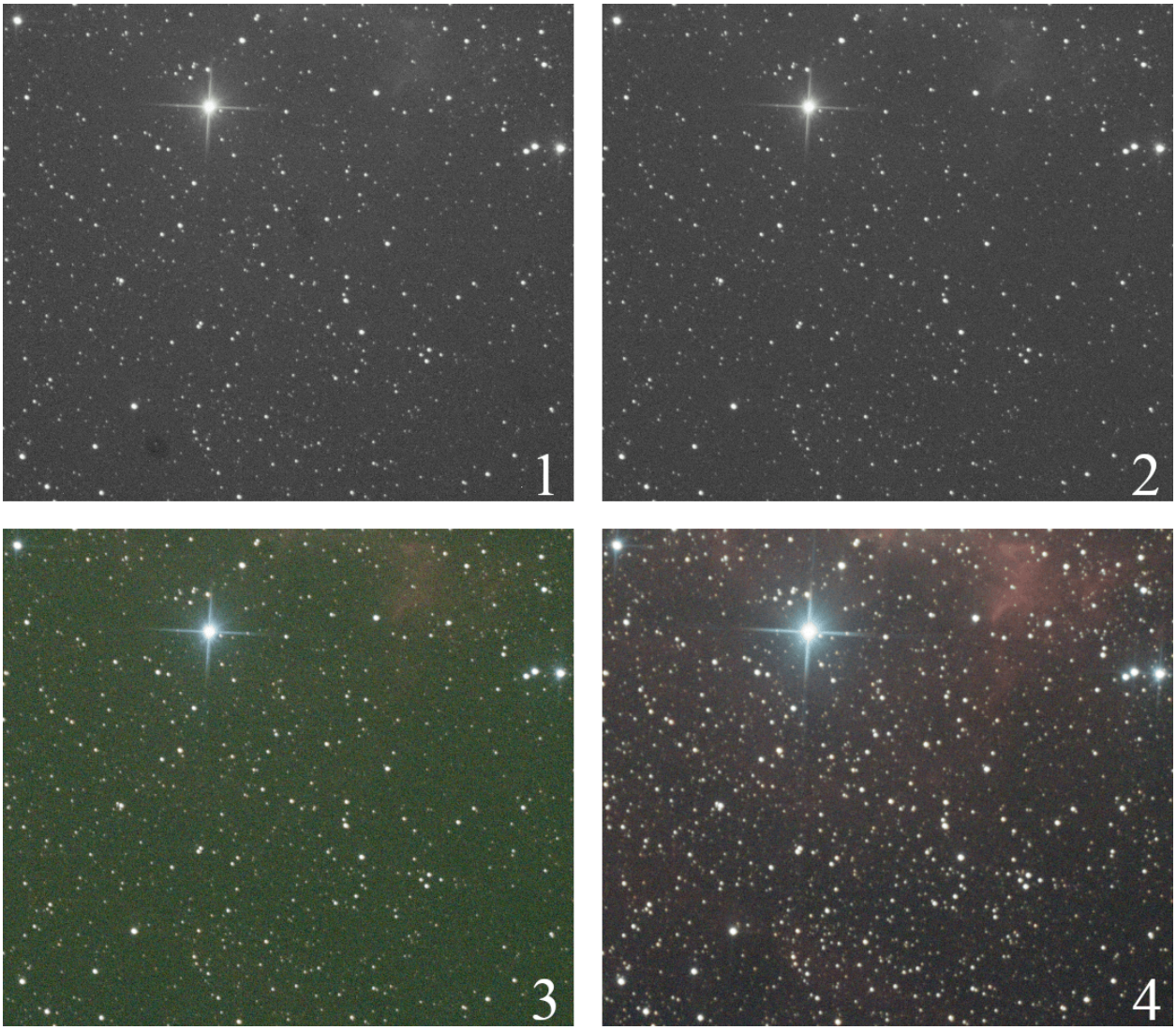



Fig. 1: **Image 1** shows the result of the conversion of a raw digital camera image. You can see visible dust, similar to dark spots. **Image 2**, after calibration of the images by the master darks, bias and flats shows the complete removal of these spots and a cleaner signal. **Image 3** is the same after demosaicing, showing color and a very large green cast due to the higher sensitivity of the green photosites on the sensors. Finally, **image 4** is the stacking output, with channel balancing.

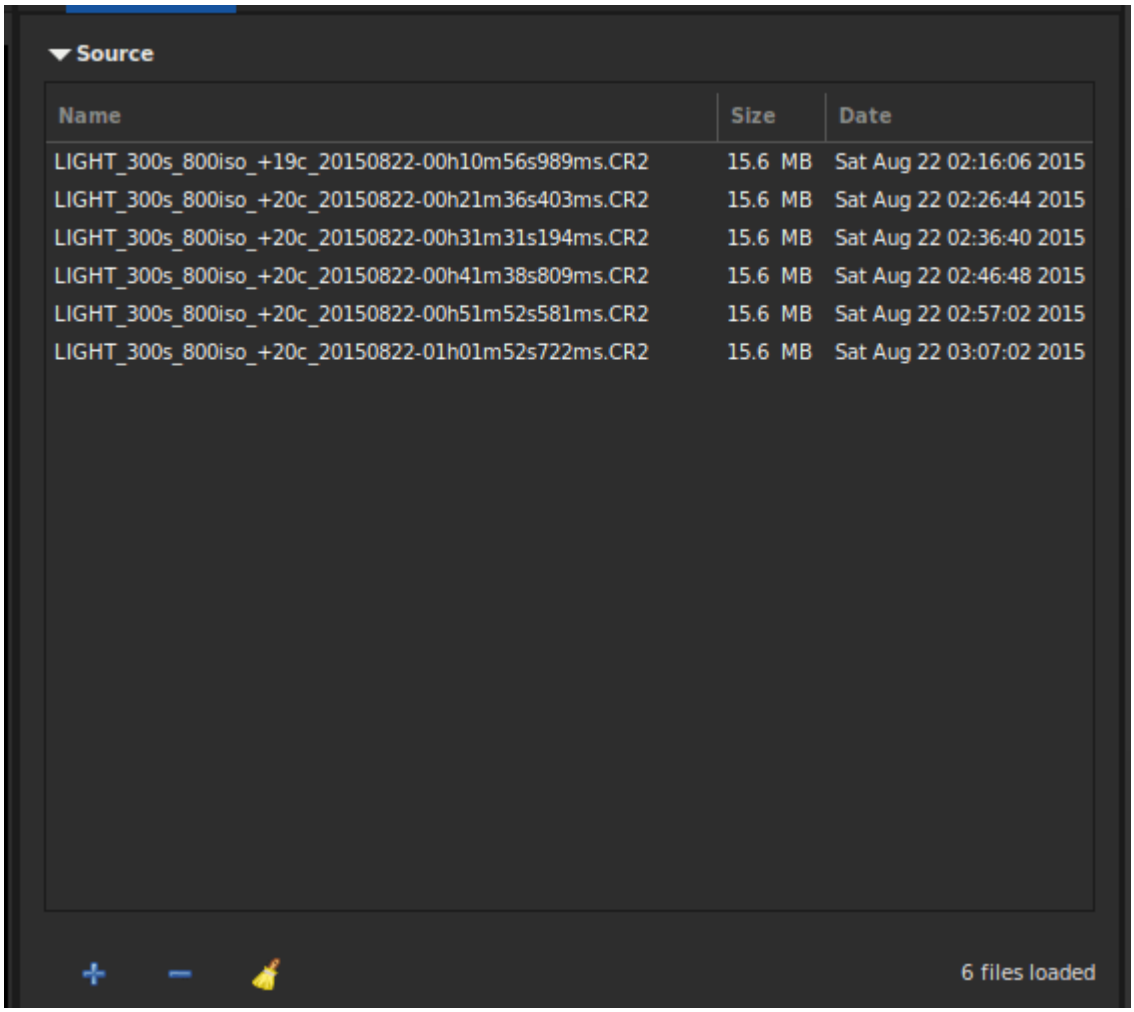



Fig. 2: Source panel of the conversion tab.

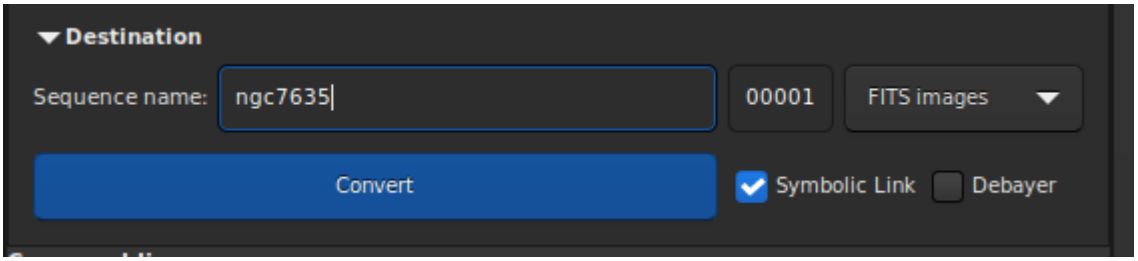



Fig. 3: Destination panel of the conversion tab.

The extension is as defined in the *preferences*. The `XXXXX` index starts by default at `00001` with the first image, however it is possible to define a different starting index. This can be useful in the case of a multi-session that shares the same master files. Three types of outputs are possible, to choose from a drop-down menu:

- FITS images
- SER sequence
- FITS sequence

These file formats are explained in the *sequence* section of this documentation.

Technically, when the input files are in FITS format, there is no need to convert them. However, you may want to do so so that the files are renamed to create a sequence and can be processed in Siril. In order not to fill the hard disk unnecessarily, it is then possible to choose the option *Symbolic link*. This option creates a symbolic link for the FITS files instead of copying them. This option is therefore only available when the output files are FITS images.

> **Note**
>
> When symbolic links are enabled, this disables compression.

> **Warning**
>
> For Microsoft Windows, the use of symbolic links requires the activation of the developer mode in Windows.

> **Warning**
>
> If on GNU/Linux you see the error **Symbolic link Error: Function not implemented** could be because you try making a symlinked sequence in a directory on a filesystem that doesn't permit symbolic links.

When the output formats are SER, or FITS sequence, then the *Multiple sequences* option becomes visible. Tick this to create several sequence files instead of a single SER or FITS file for all input elements. Use this if input elements (sequence files such as films, SER or FITS cubes) don't share the same image size or must not be processed together.

The last option *Debayer* allows the user to demosaic the images during the conversion. This option should generally not be used if the images are bias, dark and flat images, or light images intended to be pre-processed. Indeed, due to Bayer matrix consideration, the RGB result of your RAW image is an interpolated picture. In consequence pre-processing interpolated data will give wrong results. Converting RAW files of an OSC sensor gives Color Filter Array (CFA) monochrome FITS pictures. Contrary to RGB image, CFA image represent the entire sensor data with the Bayer pattern. The following image shows you a crop of a CFA image. Note that the Bayer pattern (RGGB on this example) is visible.

Finally, the button *Convert*, allows, as its name indicates, to start the conversion of files.

> **Note**
>
> The raw images of digital SLRs depend on the manufacturer and are generally closed source formats. Therefore the decoding of such files is a complex task that must be done by a dedicated code. For Siril, the task of converting raw files is performed by LibRaw. In fact, if a file format, usually a recent one, does not read, you have to look on the LibRaw website if it is supported. If it is not, providing them with a raw file can help the dev team to do so. However, it is also possible that the version of LibRaw embedded in the Siril package is not the most recent version. In this case, you must either wait for a new release or compile the sources directly.

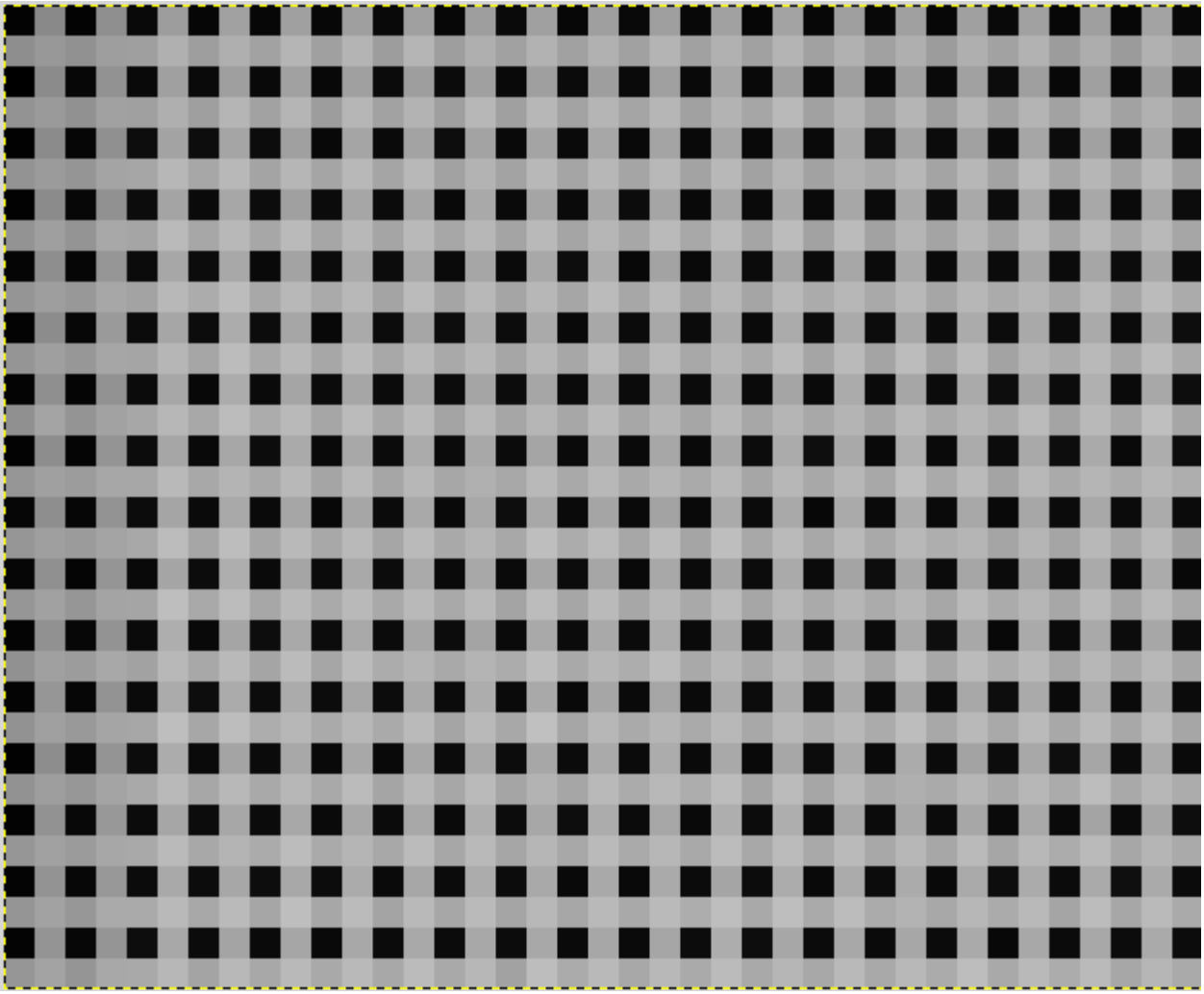


Fig. 4: Bayer pattern showed on a CFA (Color Filter Array) image.

### 9.1.1 Correspondence file

After each conversion, a file ending with `_conversion.txt` is created. It contains the correspondence between the input images and the images of the sequence obtained during the conversion.

**Siril command line**

```
convert basename [-debayer] [-fitseq] [-ser] [-start=index] [-out=]
```

Converts all images of the current working directory that are in a supported format into Siril's sequence of FITS images (several files) or a FITS sequence (single file) if **-fitseq** is provided or a SER sequence (single file) if **-ser** is provided. The argument **basename** is the base name of the new sequence, numbers and the extension will be put behind it.

For FITS images, Siril will try to make a symbolic link; if not possible, files will be copied. The option **-debayer** applies demosaicing to CFA input images; in this case no symbolic link is done.

**-start=index** sets the starting index number, useful to continue an existing sequence (not used with -fitseq or **-ser**; make sure you remove or clear the target .seq if it exists in that case).

The **-out=** option changes the output directory to the provided argument.

See also CONVERTRAW and LINK

Links: *convertraw*, *link*

**Siril command line**

```
convertraw basename [-debayer] [-fitseq] [-ser] [-start=index] [-out=]
```

Same as CONVERT but converts only DSLR RAW files found in the current working directory

Links: *convert*

## 9.2 Calibration

Once a sequence is loaded, images can be calibrated, registered and stacked. The calibration is an optional, yet important, step and involves bias, dark and flat frames. Calibrating a sequence in Siril can only be done with master bias, dark and flat frame, which have to be created from their sequences first.

### 9.2.1 Master files

#### Bias

Citing from A Glossary of CCD terminology, to explain what a bias image is:

*The bias level of a CCD frame is an artificially induced electronic offset which ensures that the Analogue-to-Digital Converter (ADC) always receives a positive signal. All CCD data has such an offset which must be removed if the data values are to be truly representative of the counts recorded per pixel.*

To use master-bias in Siril, click on the button to the right of the text entry and browse your files to select the right master. You can even use master-bias from a library as defined in the *preferences*.

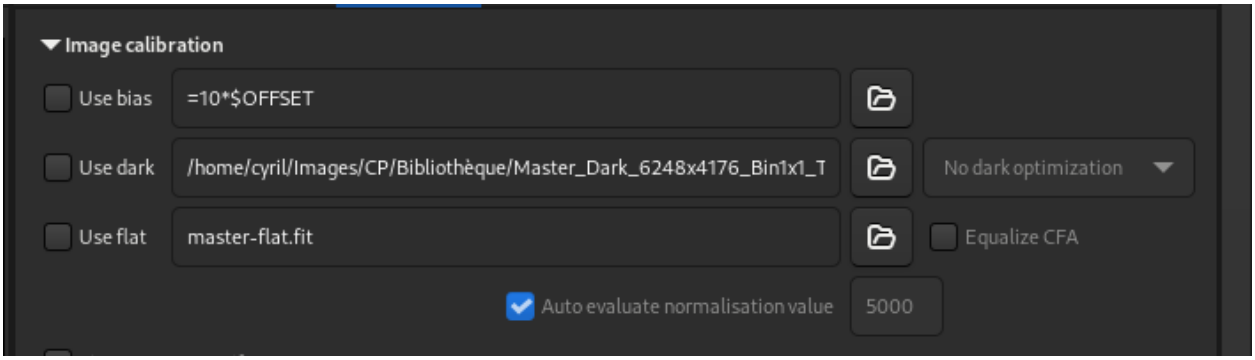



Fig. 5: Masters settings of the Calibration Tab

> **Tip**
>
> The bias frame must be taken with the shutter closed and the shortest possible exposure time. Basically it corresponds to an exposure of 1/4000s with modern DSLRs.

### Synthetic bias

Since the offset signal is very uniform on modern sensors, we recommend processing it as a constant level image. This has the advantage of saving disk space and minimizing noise in the final image. For this purpose, Siril has a feature that makes it very easy to do.

During preprocessing of your flats, instead of specifying a masterbias, you can directly type expressions in the folder selector such as:

```
=2048
```

or, if the FITS header contains the `OFFSET` keyword,

```
=64*$OFFSET
```

The = and $ tokens are mandatory. The level must be given in ADU (not float, even if you are working in 32-bit).

Translated into the scripting language, this is written:

```
preprocess flat -bias="=64*$OFFSET"
```

The value 2048 is here an example taken for cameras whose master-bias would have a median value of 2048. Generally, for DSLRs, the value is proportional to a root of 2. In our example, $2048 = 2^{11}$.

Fore more details, please refer to the tutorial on the Synthetic biases.

### Dark

Dark frames contain the thermal noise associated with the sensor, the noise being proportional to temperature and exposure time. Hence, they should be made at approximately the same temperature as the light frames, this is the reason why we make dark frames at the end, or in the middle of the imaging session.

To use master-dark in Siril, click on the button to the right of the text entry and browse your files to select the right master. You can even use master-dark from a library as defined in the *preferences*.

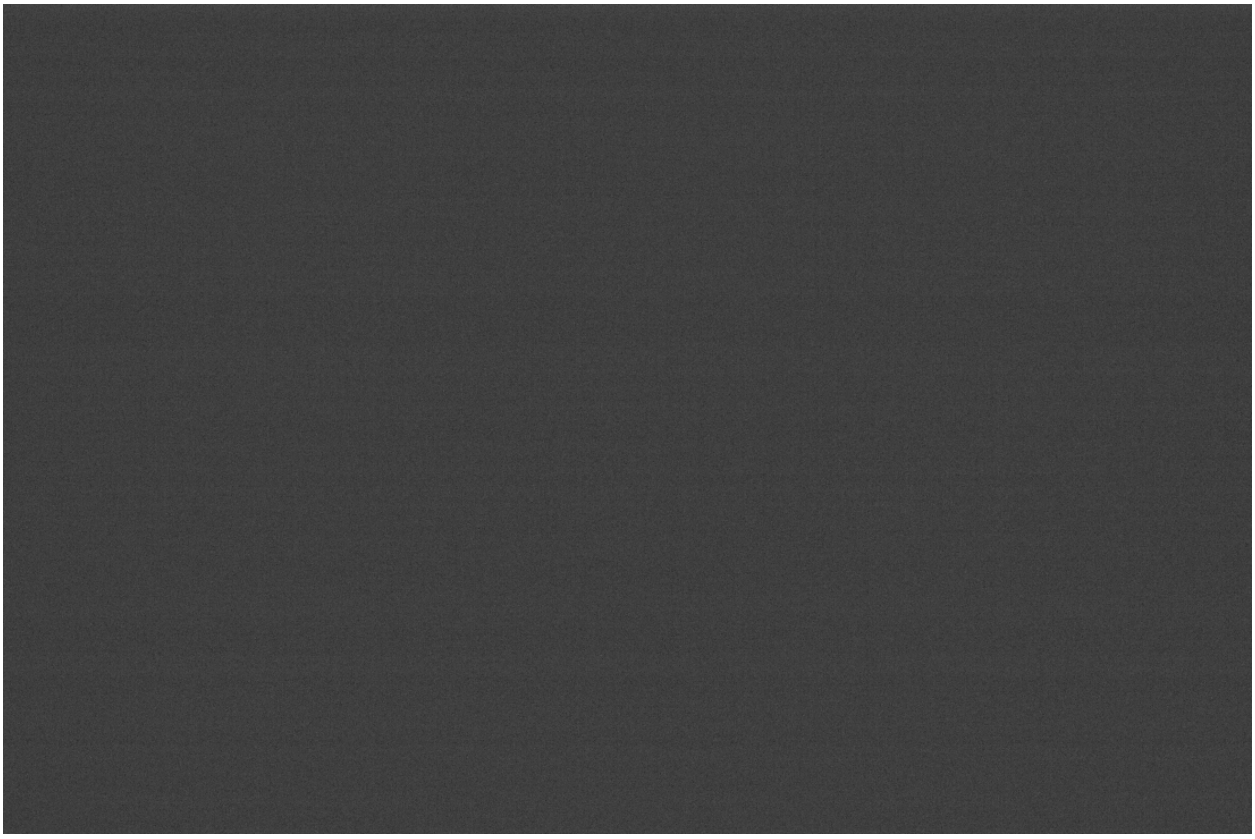


Fig. 6: Example of a bias frame that was shot with a Canon EOS 1100D. Do not rely on the slightly visible bias signal, the image is auto-stretched and the differences in signal amplitudes are very exaggerated.

**Tip**

Dark frames are made at the same exposure time and ISO/Gain than the subject light frames but with the shutter closed.

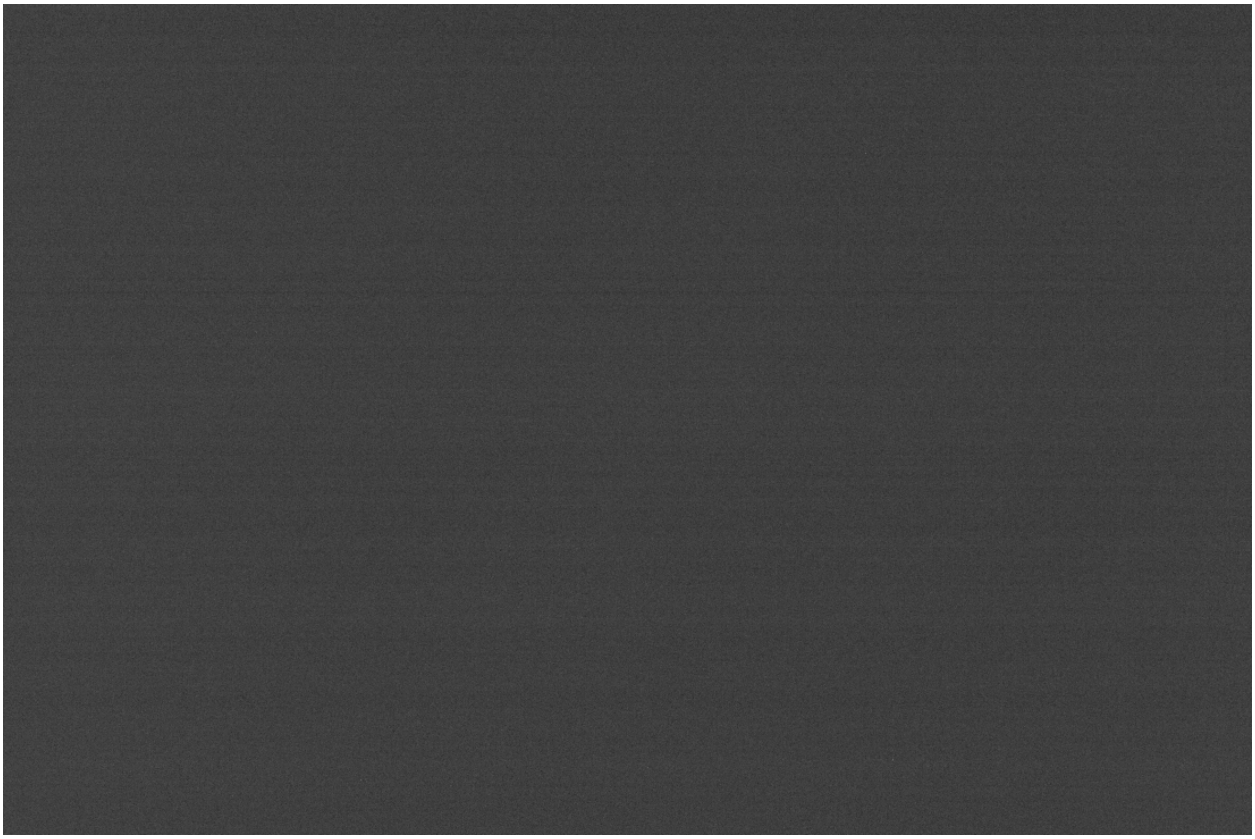


Fig. 7: Example of a dark frame taken with a Canon EOS 1100D with an 300s exposure and at ISO 800.

Fig. 8: An animation showing the removal of the thermal signal thanks to the dark subtraction.

### Dark optimization

The dark optimization option is useful when dark images have not been taken under optimal conditions. Siril offers two methods with different approaches, accessible via a drop-down list.

1. With the option *Auto-evaluation*, dark subtraction can be optimized so that the noise of the resulting picture (light frame minus dark frame) is minimized by applying a coefficient to the dark frame.
2. The second option, *Use exposure*, is based on the exposure times of the images if they have been registered in the FITS keywords.

Here's an example of a situation where the use of this option is necessary. The images were taken with a **FLI ProLine 4240 camera**. The master-dark used comes from a library and was made with an exposure of 600s. Each subs, on the other hand, have exposures of 60s. The master-dark has a very distinctive and rather unsightly signal signature: the presence of 4 preamplifiers in the camera is the cause of such a signal. This defect is obviously also present in the galaxy image, and calibration of the dark must be meticulously carried out to obtain an image free of all defects.

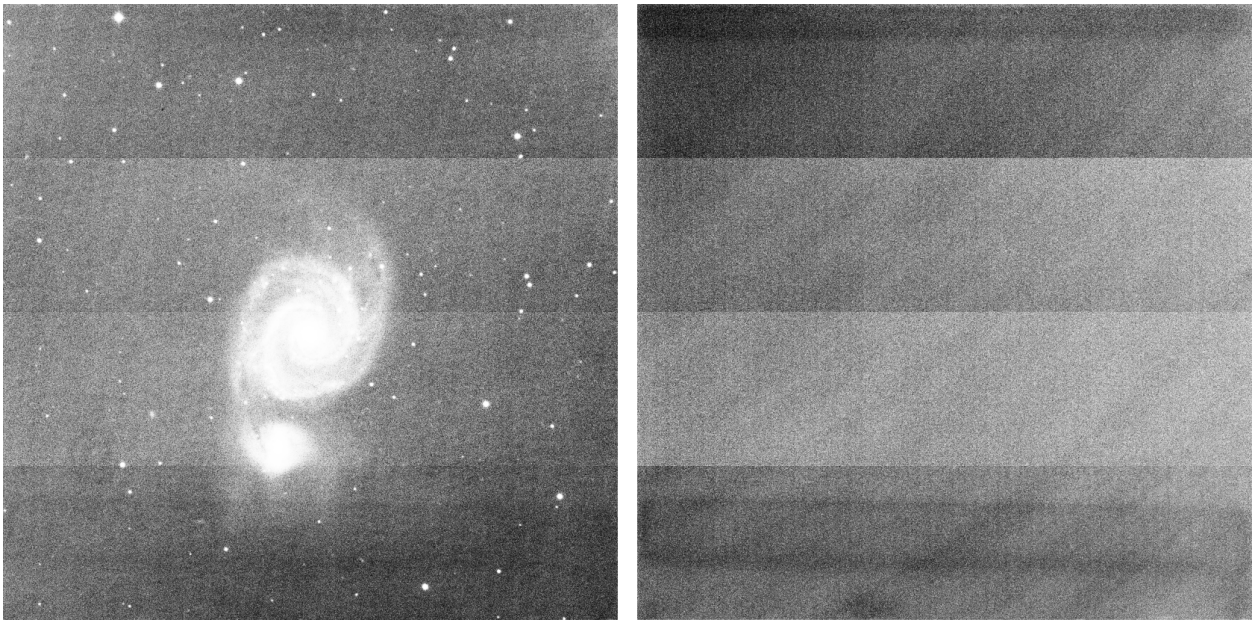


Fig. 9: This is a light frame and the master-dark of the **FLI ProLine 4240 camera**. You can see 4 very distinctive bands caused by the preamplifiers visible on both, lights and master-dark. The images are displayed in histogram equalization mode, to highlight any defects.

However, in this case, if we use the usual workflow, the calibration result will be very poor. This is because the dark master was not taken under the same exposure conditions.

The solution is therefore to subtract the bias to the dark, then integrate the bias subtraction with those of the images, and check the dark optimization box. Siril will automatically calculate a coefficient to be applied to the dark. Here, it calculates **0.110**, which is very coherent, as it corresponds to the 10-fold difference between darks and images ($60/600 = 0.1$).

```
10:34:58: Preprocessing...
10:34:58: Normalisation value auto evaluated: 0.313
10:34:58: 13230 corrected pixels (0 + 13230)
10:34:59: Dark optimization of image 0: k0=0.110
10:34:59: Saving FITS: file pp_M51SDSSr_00002.fit, 1 layer(s), 2048x2048 pixels, 32 bits
```

### Flat

Telescopes generally do not illuminate the detector evenly. In addition, dust on the optical surfaces and the sensor causes darker patterns in the resulting image, and the sensor itself reacts differently to the number of photons striking different photosites. To correct for these effects, each bright image must be divided by the master flat, which should be the median of the single exposures of a homogeneous and unsaturated area.

To use master-flat in Siril, click on the button to the right of the text entry and browse your files to select the right master. You can even use master-flat from a library as defined in the *preferences*.

### CFA equalization

The *Equalize CFA* option equalizes the mean intensity of RGB layers in a CFA flat image. This is equivalent to apply the *grey_flat* command.

Fig. 10: Using the classic workflow, calibration is poor and defects are not corrected. The image is displayed in histogram equalization mode, to highlight any defects.

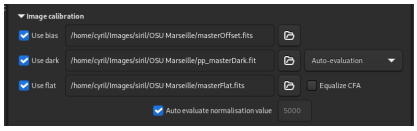



Fig. 11: The calibration tab as it should be completed in such a case. Master-flat and master darks have been calibrated by biases.

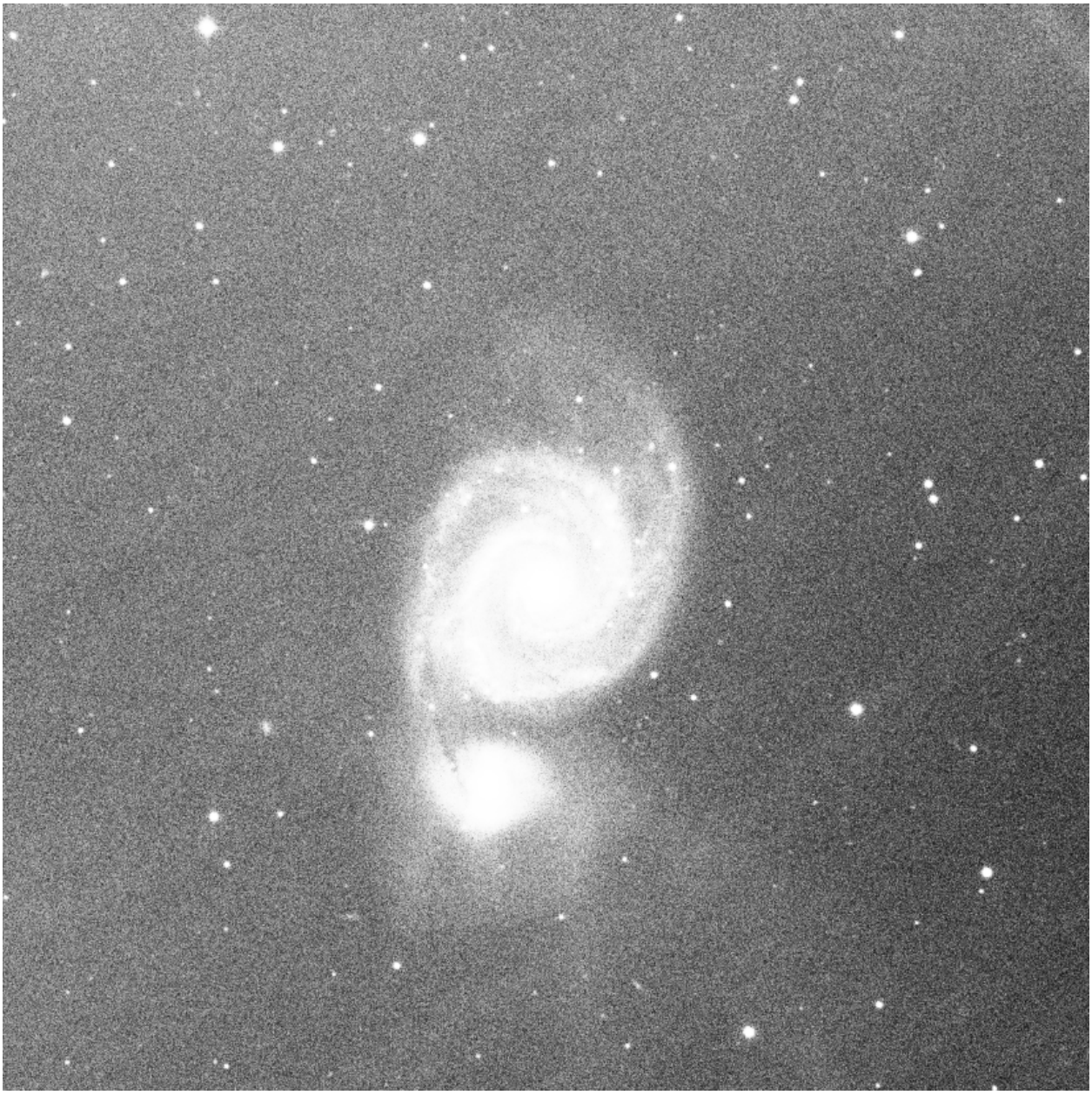


Fig. 12: Thanks to dark optimization, calibration is correct. The only residue visible is CCD fringing in the near IR range, which cannot be removed by calibration. The image is displayed in histogram equalization mode, to highlight any defects.

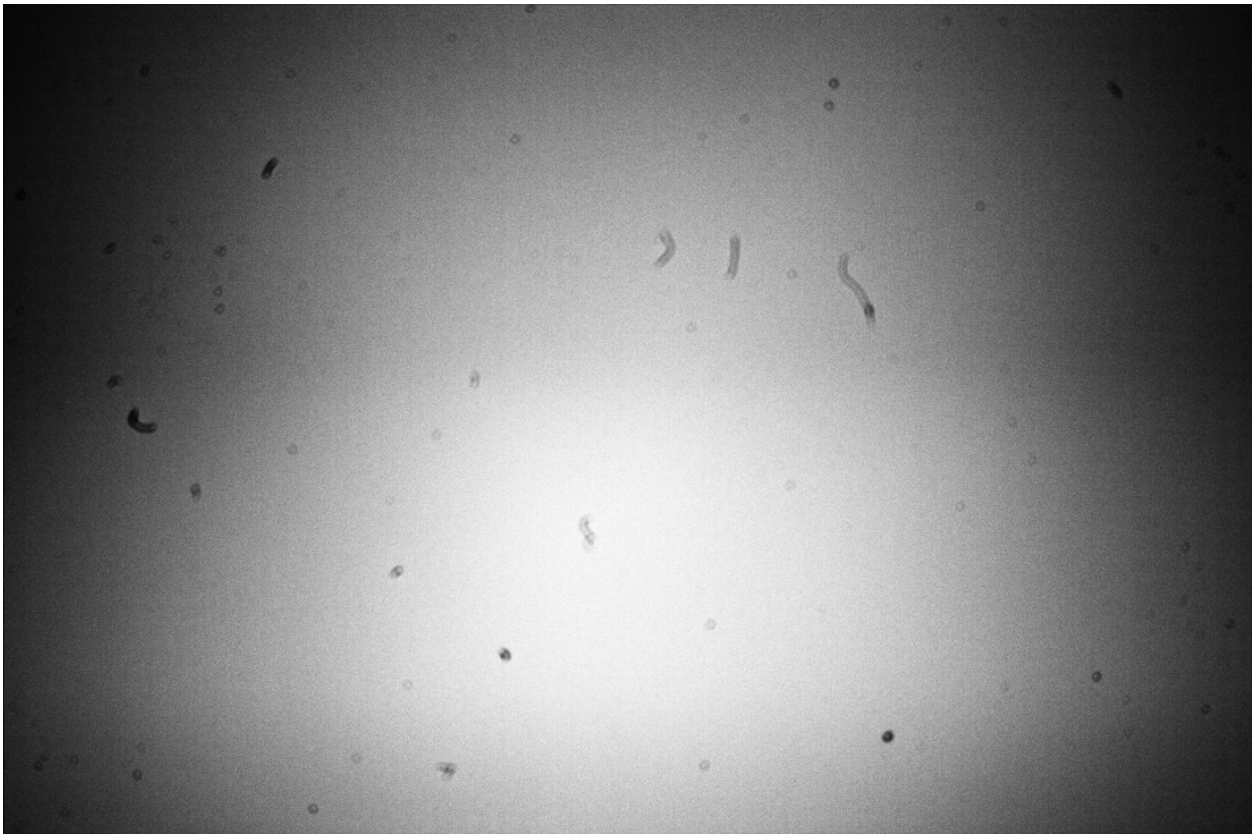


Fig. 13: Example of a flat frame that was shot with a Canon EOS 1100D. The dust present on the optical path, and especially on the sensor, are clearly visible. The vignetting (darkening of the corners of the image) is also very visible. The defects are exaggerated by the viewing mode. Also, the *grey_flat* command was used on this image to get rid of the Bayer pattern.

**Siril command line**

```
grey_flat
```

Equalizes the mean intensity of RGB layers in the loaded CFA image. This is the same process used on flats during calibration when the option equalize CFA is used

#### Auto evaluate normalisation value

If the *Auto evaluate normalisation value* option is checked, Siril will auto-evaluate the normalisation value. This value is the mean of the master-flat calibrated with the master-bias. Otherwise, the value indicated in the text box will be taken into account.

### 9.2.2 Calibration of light images

The calibration of the light images consists in applying the master bias, dark and flat to the astronomical images in order to remove the unwanted signal.

**Warning**

In no case does the calibration reduce the noise of the images. On the contrary, it increases it. This is why it is important to take as many calibration images as possible, such as darks, in order to minimize the amount of noise in the images.

#### Fix X-Trans AF artifact

This *Fix X-Trans AF artifact* option helps to fix the Fujifilm X-Trans Auto Focus pixels. Indeed, because of the phase detection auto focus system, the photosites used for auto focus get a little less light than the surrounding photosites. The camera compensates for this and increases the values from these specific photosites giving a visible square in the middle of the dark/bias frames. This option has no effect on Bayer pattern. The option is only enabled if a master-bias or master-dark is loaded and used.

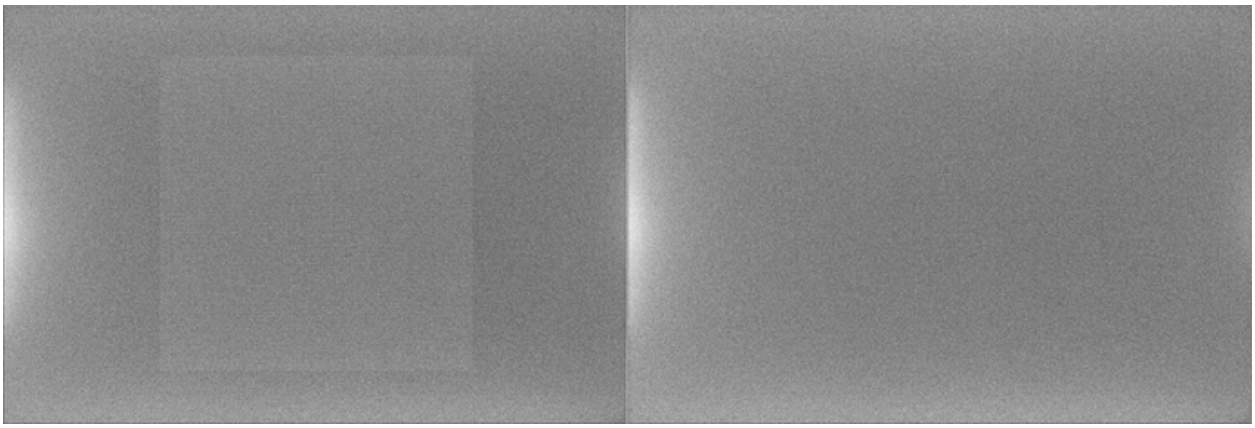


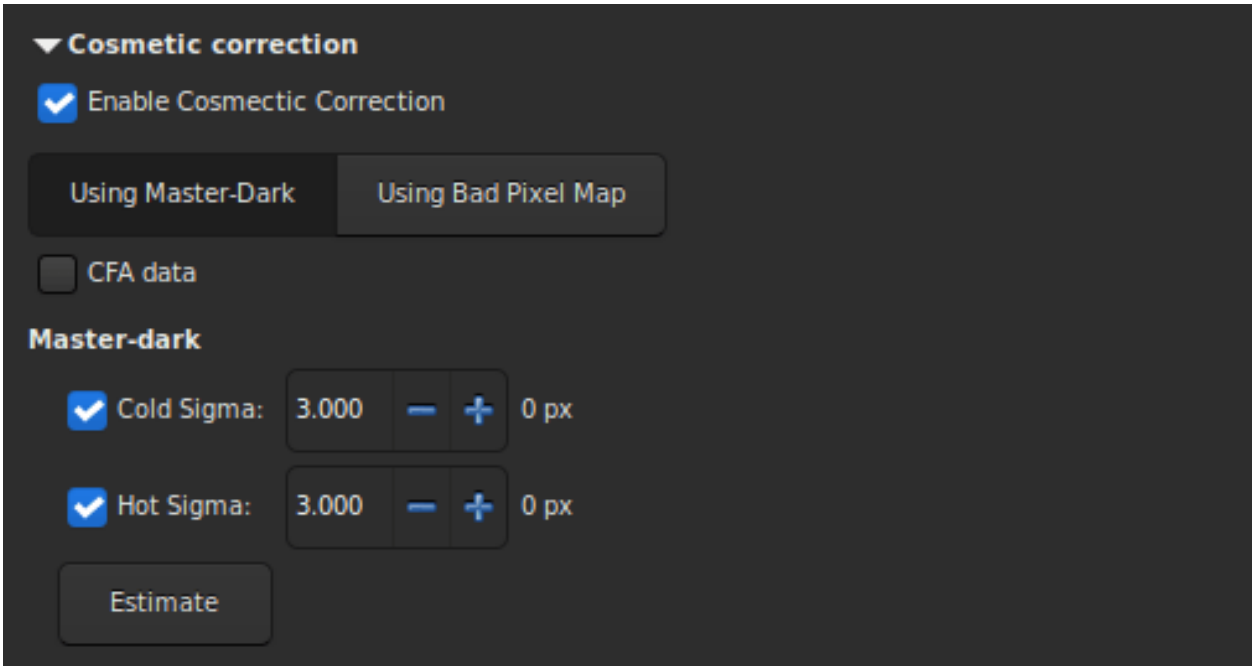
Fig. 14: X-Trans artifact fixed by the algorithm of Siril

### Cosmetic Correction

Cosmetic correction is the technique used to correct defective pixels in images. Indeed, any camera sensor has photosites that do not react correctly to photon reception. This is visible in the image with pixels with values very different from those of their nearest neighbors. These pixels are called **hot pixels**, if the value is much higher, or **cold pixels** when it is much lower. Siril offers two algorithms to correct these defective pixels if the option *Enable Cosmectic Correction* is checked.

### Using Master-Dark

This method requires the presence of a master-dark. Siril will search for pixels whose deviation from the median exceeds x times the standard deviation $\sigma$. This value is adjustable for both hot and cold pixels.




It is possible to estimate the number of pixels that will be corrected in the calibrated image by pressing the *Estimate* button. If the corrected pixel value is displayed in red, it means that this number exceeds 1% of the total number of pixels in the image. In this case you should increase the value of the coefficient, or uncheck the corresponding correction. If the images are from a color sensor then it is necessary to check the *CFA* option.

### Using Bad Pixel Map

The other method uses a file that contains the coordinates of the defective pixels. This file is a simple text file and can initially be created with the command *find_hot*. The last line has been added by hand and corrects a damaged column located at position $x = 1527$.

```
P 325 2855 H
P 825 2855 C
P 838 2855 H
P 2110 2855 H
P 2702 2855 H
P 424 2854 H
C 1527 0 H
```

**Siril command line**

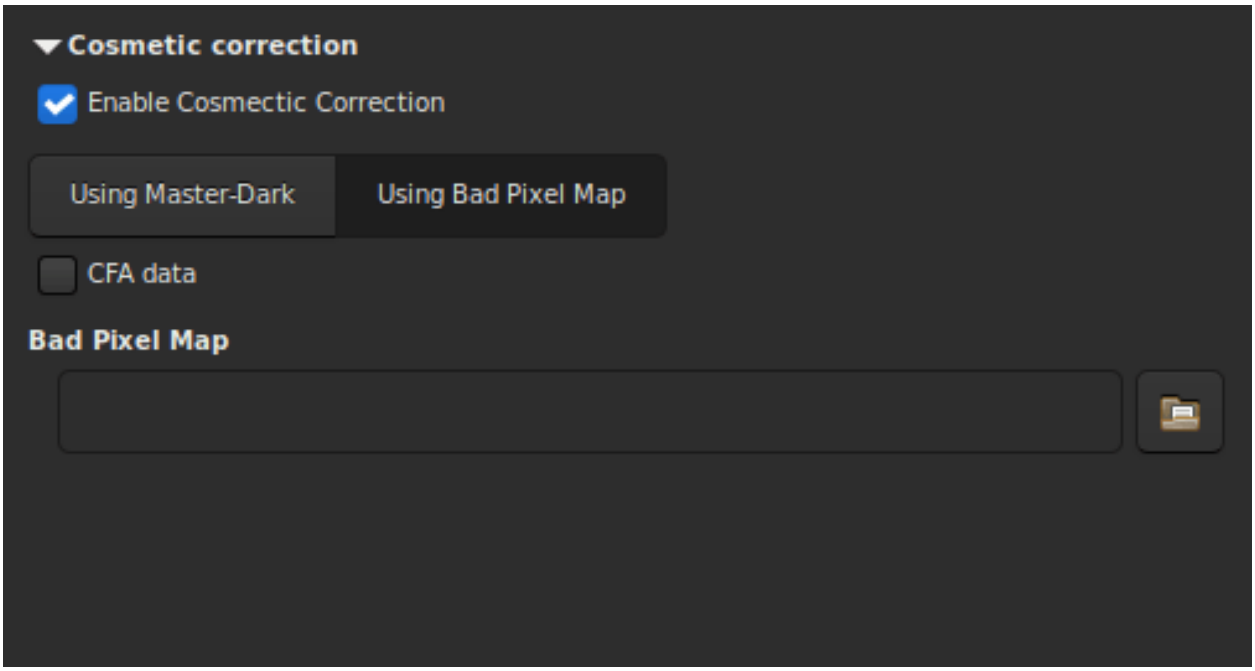
```
find_hot filename cold_sigma hot_sigma
```

Saves a list file **filename** (text format) in the working directory which contains the coordinates of the pixels which have an intensity **hot_sigma** times higher and **cold_sigma** lower than standard deviation, extracted from the loaded image. We generally use this command on a master-dark file. The COSME command can apply this list of bad pixels to a loaded image, see also SEQCOSME to apply it to a sequence

Links: *cosme*, *seqcosme*

Lines `P x y type` will fix the pixel at coordinates (x, y) type is an optional character (C or H) specifying to Siril if the current pixel is cold or hot. This line is created by the command FIND_HOT but you also can add some lines manually:
Lines `C x 0 type` will fix the bad column at coordinates x.
Lines `L y 0 type` will fix the bad line at coordinates y.

This file, which can be edited by hand, is to be loaded as a Bad Pixel Map.

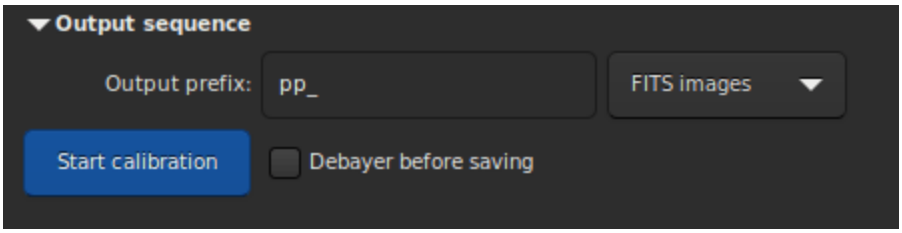



Finally, if the images are from a color sensor then it is necessary to check the *CFA* option.

### Output sequence

This section groups the options that can be applied to the output.

- The *Output Prefix* entry box adds a prefix to the output images, to easily identify them. By default, the prefix is `pp_`, which means pre-processed.
- The drop-down list defines the type of destination sequence.




  - FITS images: one FITS file per image.
  - SER sequence: one SER file for the whole sequence (limited to 16 bits per channel).
  - Sequence FITS: one FITS file for the entire sequence.

- Last option, *Debayer before saving*. Check this option if you want to apply a demosaicing algorithm to your frames right after they were calibrated. By doing this, you skip one manual step which can take some time.

### Command lines

**Siril command line**

```
calibrate sequencename [-bias=filename] [-dark=filename] [-flat=filename] [-cc=dark␣
↪[siglo sighi] || -cc=bpm bpmfile] [-cfa] [-debayer] [-fix_xtrans] [-equalize_cfa] [-
↪opt[=exp]] [-all] [-prefix=] [-fitseq]
```

Calibrates the sequence **sequencename** using bias, dark and flat given in argument.

For bias, a uniform level can be specified instead of an image, by entering a quoted expression starting with an = sign, such as -bias="=256" or -bias="=64*$OFFSET".

By default, cosmetic correction is not activated. If you wish to apply some, you will need to specify it with **-cc=** option.
You can use **-cc=dark** to detect hot and cold pixels from the masterdark (a masterdark must be given with the **-dark=** option), optionally followed by **siglo** and **sighi** for cold and hot pixels respectively. A value of 0 deactivates the correction. If sigmas are not provided, only hot pixels detection with a sigma of 3 will be applied.
Alternatively, you can use **-cc=bpm** followed by the path to your Bad Pixel Map to specify which pixels must be corrected. An example file can be obtained with a *find_hot* command on a masterdark.

Three options apply to color images (in CFA format): **-cfa** for cosmetic correction purposes, **-debayer** to demosaic images before saving them, and **-equalize_cfa** to equalize the mean intensity of RGB layers of the master flat, to avoid tinting the calibrated image.
The **-fix_xtrans** option is dedicated to X-Trans images by applying a correction on darks and biases to remove a rectangle pattern caused by autofocus.
It's also possible to optimize dark subtraction with **-opt**, which requires the supply of bias and dark masters, and automatically calculates the coefficient to be applied to dark, or calculates the coefficient thanks to the exposure keyword with **-opt=exp**.
By default, frames marked as excluded will not be processed. The argument **-all** can be used to force processing of all frames even if marked as excluded.
The output sequence name starts with the prefix "pp_" unless otherwise specified with option **-prefix=**.
If **-fitseq** is provided, the output sequence will be a FITS sequence (single file)

**Siril command line**

```
calibrate_single imagename [-bias=filename] [-dark=filename] [-flat=filename] [-
↪cc=dark [siglo sighi] || -cc=bpm bpmfile] [-cfa] [-debayer] [-fix_xtrans] [-
↪equalize_cfa] [-opt[=exp]] [-prefix=]
```

Calibrates the image **imagename** using bias, dark and flat given in argument.

For bias, a uniform level can be specified instead of an image, by entering a quoted expression starting with an = sign, such as -bias="=256" or -bias="=64*$OFFSET".

By default, cosmetic correction is not activated. If you wish to apply some, you will need to specify it with **-cc=** option.
You can use **-cc=dark** to detect hot and cold pixels from the masterdark (a masterdark must be given with the **-dark=** option), optionally followed by **siglo** and **sighi** for cold and hot pixels respectively. A value of 0 deactivates the correction. If sigmas are not provided, only hot pixels detection with a sigma of 3 will be applied.
Alternatively, you can use **-cc=bpm** followed by the path to your Bad Pixel Map to specify which pixels must be corrected. An example file can be obtained with a *find_hot* command on a masterdark.

Three options apply to color images (in CFA format): **-cfa** for cosmetic correction purposes, **-debayer** to demosaic images before saving them, and **-equalize_cfa** to equalize the mean intensity of RGB layers of the master flat, to avoid tinting the calibrated image.
The **-fix_xtrans** option is dedicated to X-Trans images by applying a correction on darks and biases to remove a rectangle pattern caused by autofocus.
It's also possible to optimize dark subtraction with **-opt**, which requires the supply of bias and dark masters, and automatically calculates the coefficient to be applied to dark, or calculates the coefficient thanks to the exposure keyword with **-opt=exp**
The output filename starts with the prefix "pp_" unless otherwise specified with option **-prefix=**

### 9.2.3 Understanding how the flats correct the lights

The point of this section is to give a bit more insight in how the different levels play a role in the correction of the lights by the flats.

We will disregard here any considerations about noise (again, noise does not vanish with masters subtraction or division, it decreases by averaging over many realizations of the same random process). We also disregard particular spatial patterns such as ampglow or dust.

If we try to quantify the intensity of background pixels in the different frames we have, we can write the following expressions:

$$L = a - b \times \left(x - \frac{W}{2}\right)^2 + d_{\text{rate}} \times t_{\text{lights}} + o \tag{9.1}$$

$$D = d_{\text{rate}} \times t_{\text{lights}} + o \tag{9.2}$$

$$F = K\left(a - b \times \left(x - \frac{W}{2}\right)^2\right) + o \tag{9.3}$$

$$O = o \tag{9.4}$$

with, $L$ for Lights, $D$ for Darks, $F$ for Flats and $O$ for Bias.

For the lights $L$, the first part is a spatial illumination component, *i.e.*, $a - b(x - \frac{W}{2})^2$. We have chosen here a quadratic variation with a maximum value $a$ in the middle of the frame of width $W$, even about the center of the sensor. This is not the exact spatial shape of vignetting but it is a good enough approximation to understand how it works. In addition to this spatial illumination term, there is a term varying with exposure time which is usually named dark current ($d_{\text{rate}} \times t_{\text{lights}}$) but which does not depend on the position of the pixel on the sensor. And finally there is a pedestal, *i.e.* the offset. This offset is present in any frame which is shot, so that we find it in all the expressions.

The darks $D$ not being illuminated, they only bear the dark current term, with same intensity as lights as they are shot for the same amount of time, and the offset term.

The flats $F$ also have a spatial term, proportional to the term found in the lights. The factor $K$, larger than 1, simply shows that their intensity is larger. To write this, we only need to assume that the pixels respond linearly to the number of photons they gather, which is sensible. We could also have written a dark current term, proportional to the exposure time of flats. But unless this time is significant, we can assume it is negligible. If it is not the case, then it means you need to shoot dark flats, or at least to assess their level.

And finally the offsets $O$ only measure the offset level.

To visualize these levels, we have plotted here-below these expressions as curves wrt. position on a frame and we encourage you to do the same and to play around with the inputs.

- $a = 200$[ADU]
- $b = 0.0003$[ADU/px$^2$]
- $d_{\text{rate}} = 1$[ADU/s]
- $t_{\text{lights}} = 10$[s]
- $o = 2048$[ADU]
- $W = 1000$[px]

$L$, $D$ and $O$ values in ADU are given on the left scale while $F$ are on the scale reported to the right.

Now what does calibrating your lights mean? When you calibrate your lights, you perform the following operation:

$$L_c = \frac{L - D}{F - O}.$$

The term $F - O$ is a flat from which you have subtracted the offset level (whether it is a masterbias or simply a level). This is the operation performed prior to stacking your masterflat. And the term $L - D$ represents a light from which you have subtracted the dark current level and the offset, *i.e.* a masterdark. If you replace with the expressions shown above, you end up with the following:

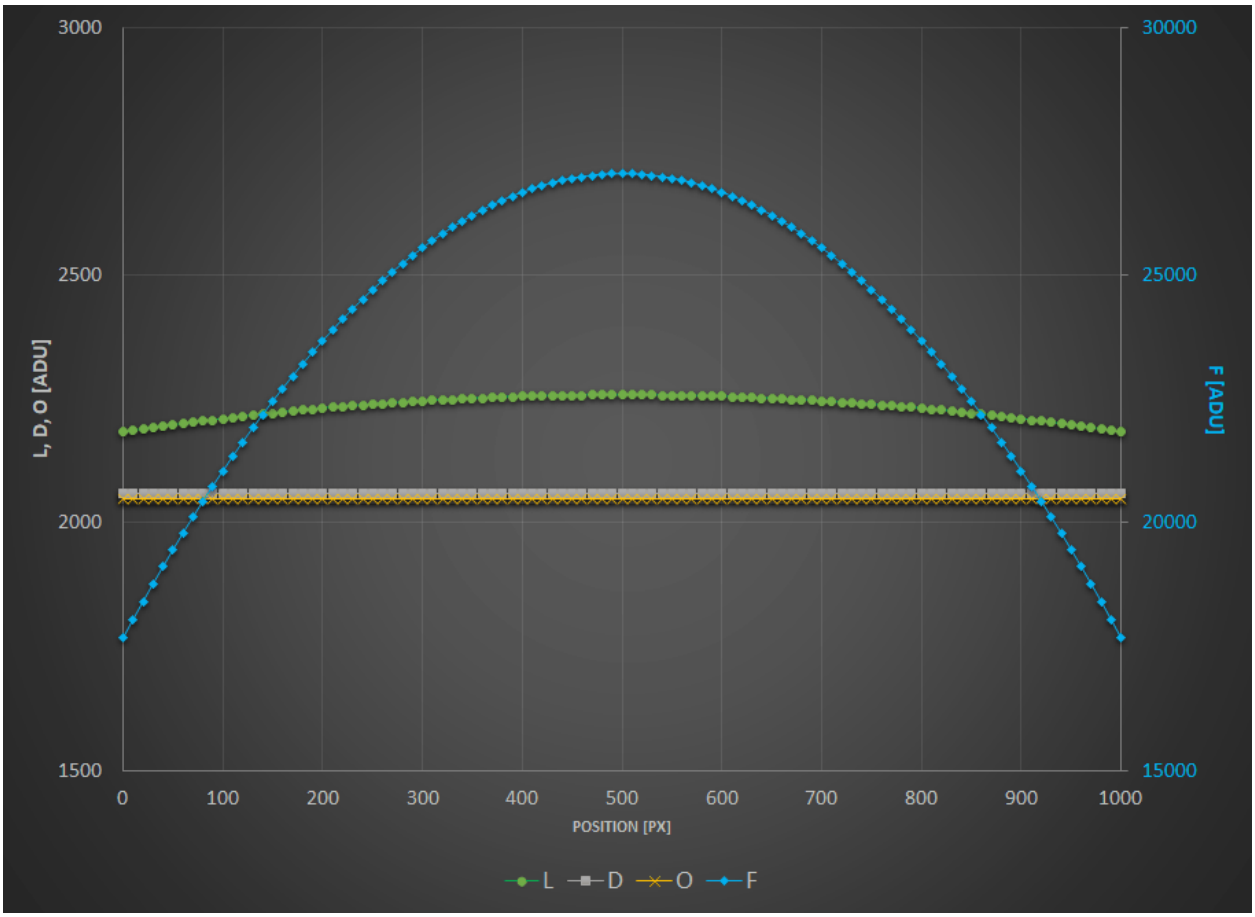
$$L_c = \frac{1}{K}.$$

No spatial variation term is left, you have flat-fielded your lights! Getting a sensible value in ADU (and not $1/K$) is what Siril does when you check *Auto evaluate normalisation value* in the *Calibration* tab.

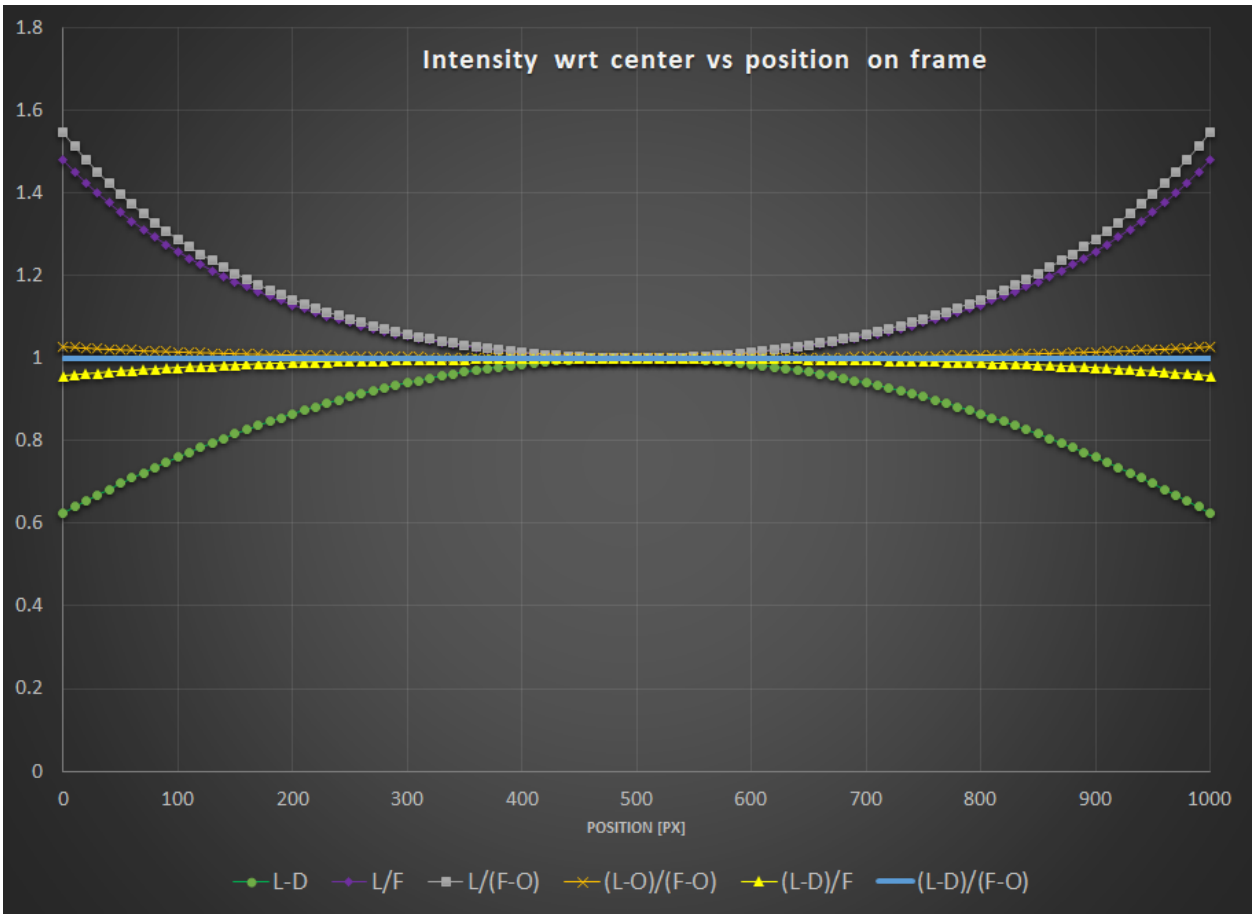
And you can try with any other combination, no other will get rid of spatial variations.

Just to illustrate this, We have plotted below the result of different combinations. To put everything on the same scale, all the results are normalized to have the same intensity of 1 in the middle of the frame. The following tests are shown:

- $L - D$ : you have just shot darks.
- $L/F$ : you have just shot flats.
- $L/(F - O)$ : you have shot flats and corrected them by an offset (either a master or a synthetic one).
- $(L - O)/(F - O)$ : you have just flats corrected by offset. But you have subtracted the offset from your lights as well.


3000
2500
2000
1500
30000
25000
20000
15000
L, D, O [ADU]
F [ADU]
0
100
200
300
400
500
600
700
800
900
1000
POSITION [PX]
L
D
O
F

- $(L-D)/F$ : you have shot flats and darks but no offsets.
- $(L-D)/(F-O)$ : you have done everything by the book.




Interestingly, you can notice that:

- $L-D$ shows obviously no correction for vignetting.
- Both $L/F$ and $L/(F-O)$ show overcorrection or inverse vignetting.
- Getting very close to the optimal result, $(L-D)/F$ and $(L-O)/(F-O)$ shows a field almost flat. This, of course, will depend how much your sensor has dark current and how much vignetting your optical train shows.
- The reference calibration gives a flat field.

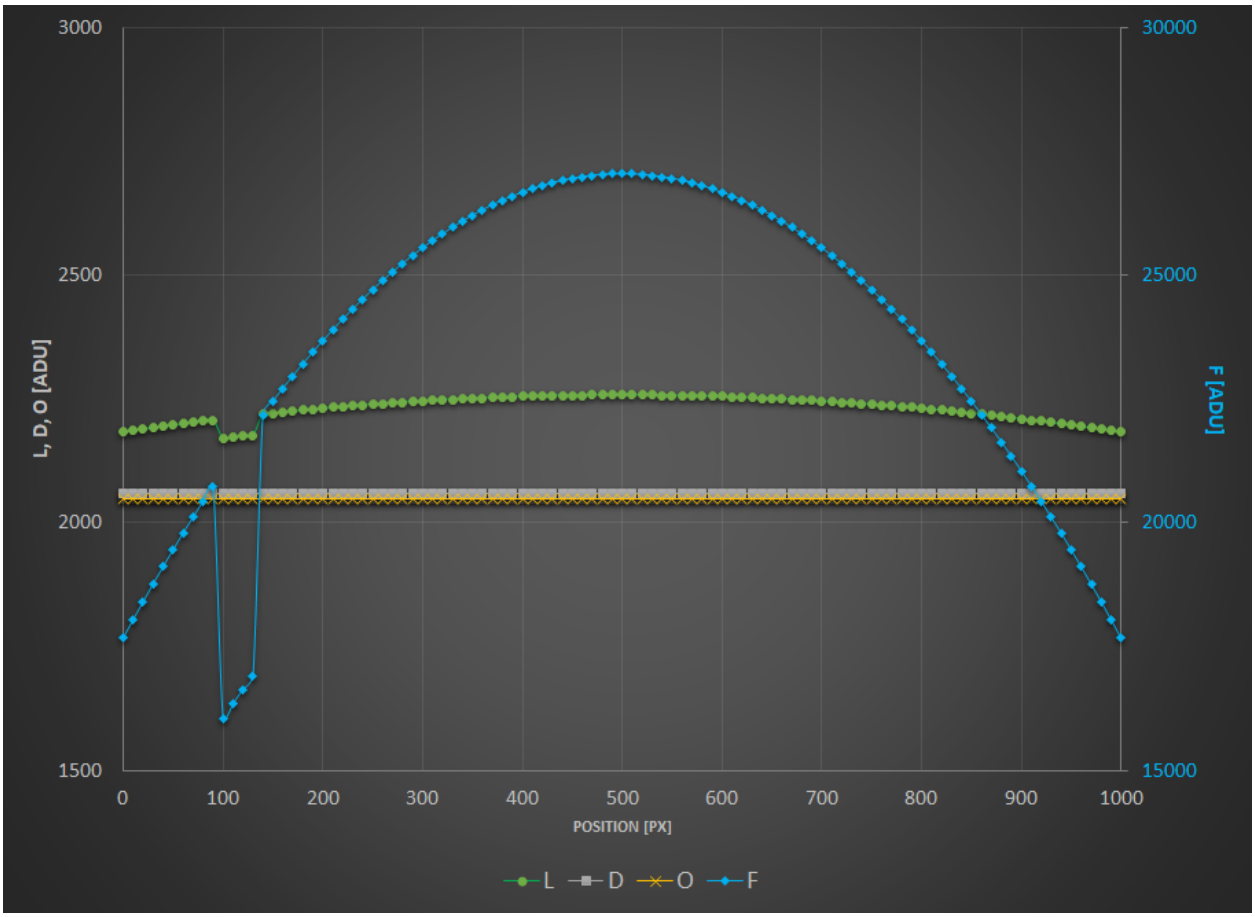
The conclusions that you can draw from the above are:

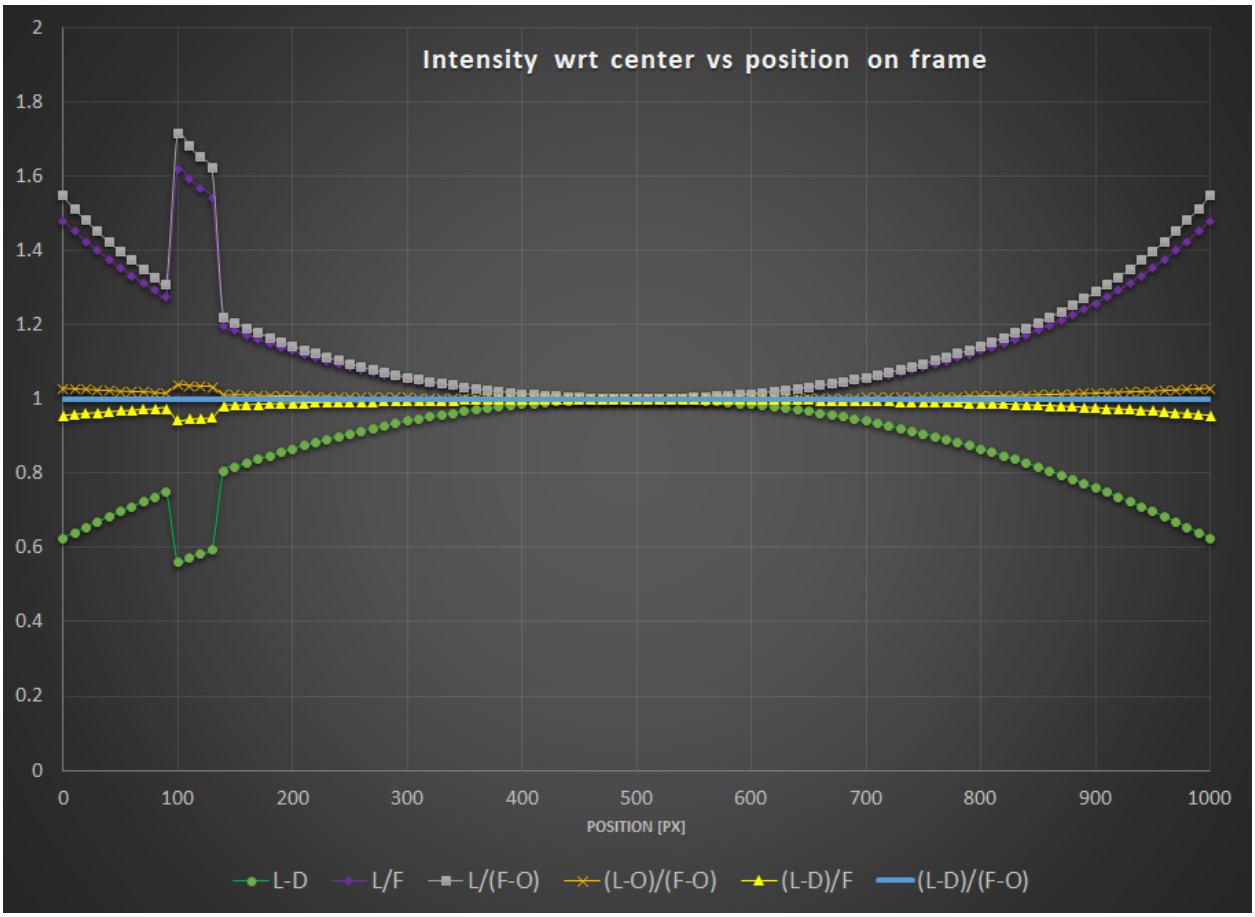
- You are better off correcting your lights with offset (masterbias or simply a level) if you have not shot darks.
- Even better, if you don't have time to shoot a series of darks, it is probably worth shooting at least one dark, measure its median, and subtract this (synthetic) dark from your lights. It will of course not correct for ampglow or enable hot pixel correction, but your lights will at least be flat!

### Now what about dust...?

In order for your flats to also correct for these nasty spots, the sad news is you also need to get all the calibration frames in the equation. We have added a small local ADU deficit in the lights and flats to illustrate this effect.

As you can see, only the combination $(L-D)/(F-O)$ can get rid of it.


3000
2500
2000
1500
30000
25000
20000
15000
L, D, O [ADU]
F [ADU]
0
100
200
300
400
500
600
700
800
900
1000
POSITION [PX]
L
D
O
F


Intensity wrt center vs position on frame
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
600
700
800
900
1000
POSITION [PX]
L-D
L/F
L/(F-O)
(L-O)/(F-O)
(L-D)/F
(L-D)/(F-O)

To further illustrate the equations and curves above, nothing is better than a real-life example. All pictures below are shown courtesy of G. Attard.

Fig. 15: $L - D$

### 9.2.4 Troubleshooting calibration

Calibration is a very simple step arithmetically, and cannot fail if the input data conforms to what is expected for astronomical images.

However, users are regularly confronted with situations where the calibrated images are not correct. In this section, we'll give you an overview of the problems encountered and how to avoid them.

First of all, the *statistic tool* is an invaluable aid to understanding problems, and is used in the majority of cases to fix issues.

- When analyzing the statistics of a master-dark, it must first be black. This is because these images are taken with the cap closed, and there's no reason why one of the photosites should be privileged. The image must look as if it had been taken by a monochrome sensor, with the Bayer matrix not visible. Below, here's an example where the master-dark has undergone an unwanted color balance for this type of image. As a result, it is no longer black and the Bayer pattern is visible. Such a dark is unfit for use and must be remade.
- During the night session, it's very important to set the OFFSET value to the same value for all images. Particularly it is mandatory to have the same setting for the pairs darks/lights and bias/flats. Failing to meet the first condition may end up in losing significant data (clipping pixels to the left hand side of the histogrram). Failing to meet both conditions will most probably prevent your images to be correctly flat-fielded (see *section above*).
- Check darks and lights levels: the median value of lights images must be sufficiently higher than that of darks to avoid generating images full of pixels with negative values.

Fig. 16: $L/F$

Fig. 17: $L/(F-O)$

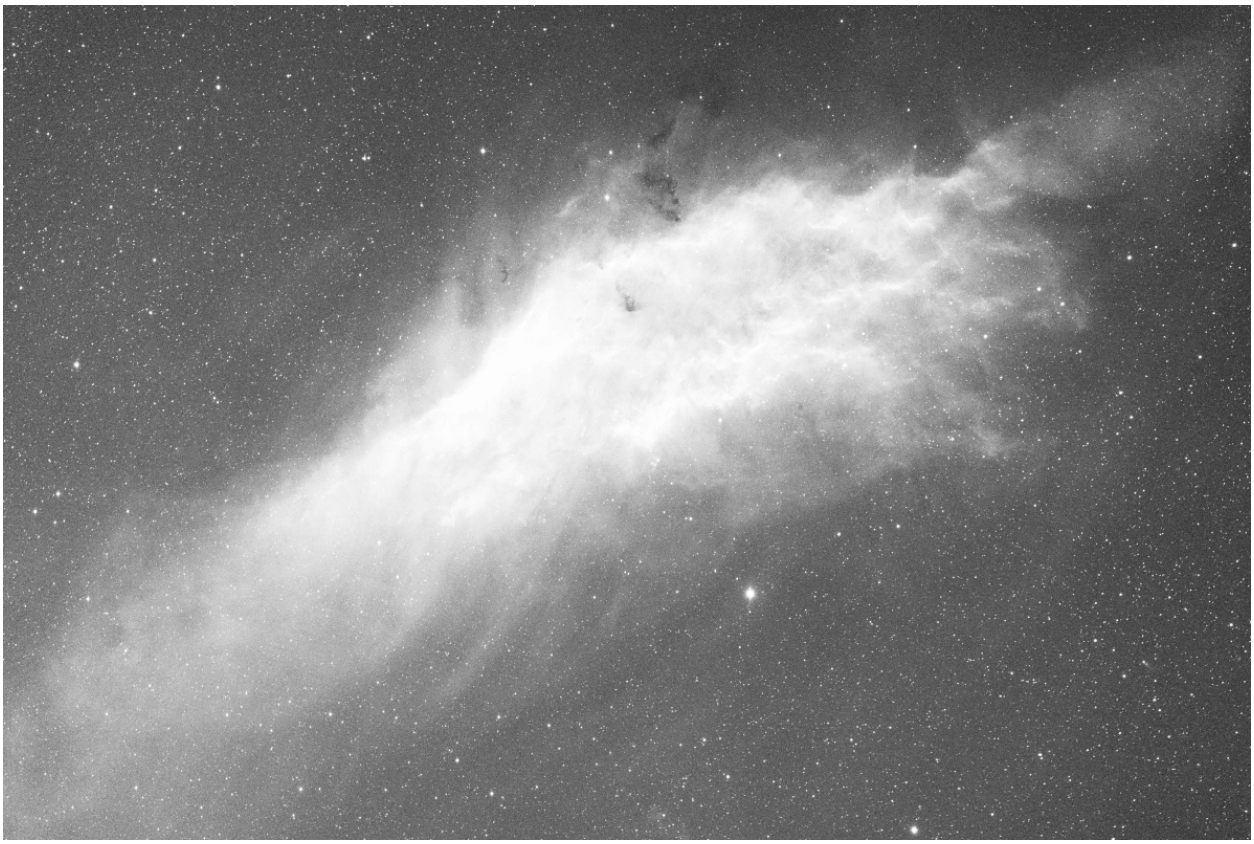


Fig. 18: $(L - O)/(F - O)$

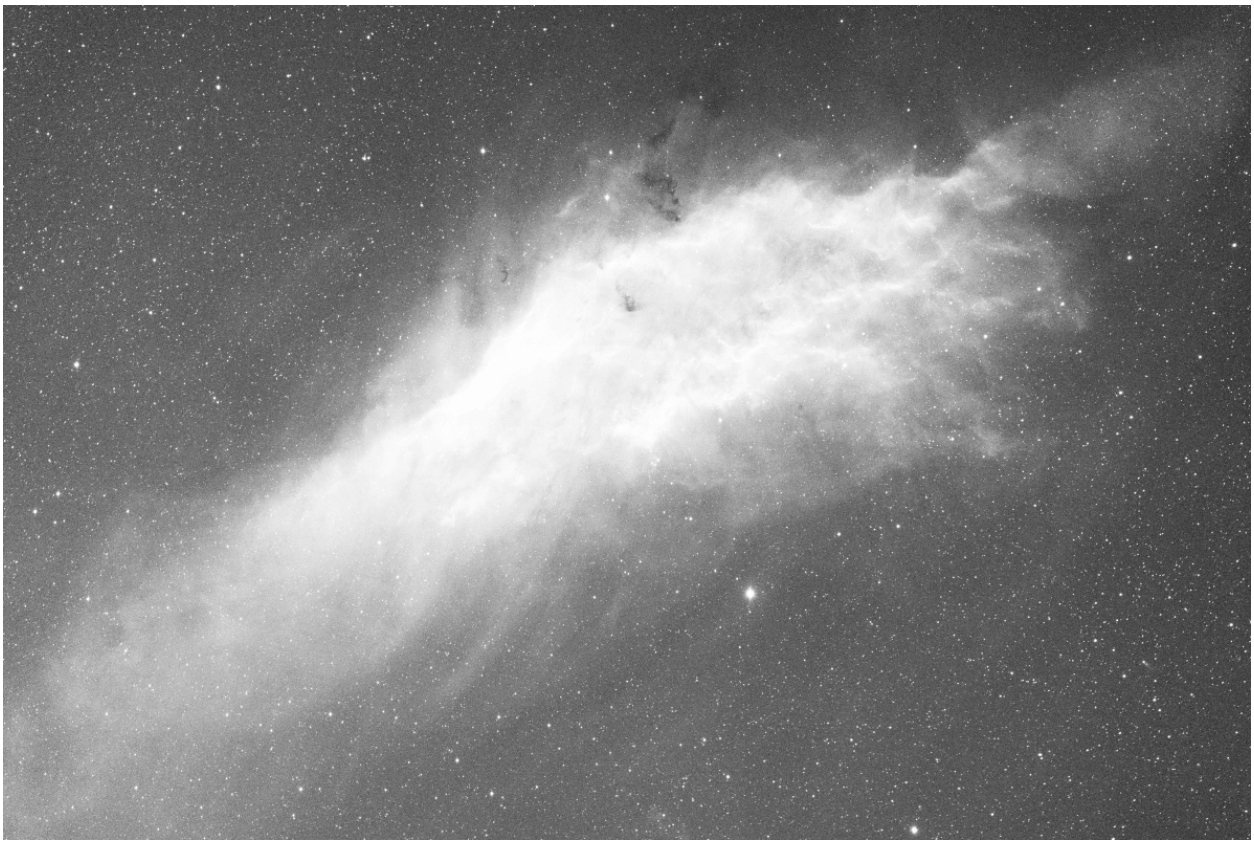


Fig. 19: $(L - D)/(F - O)$

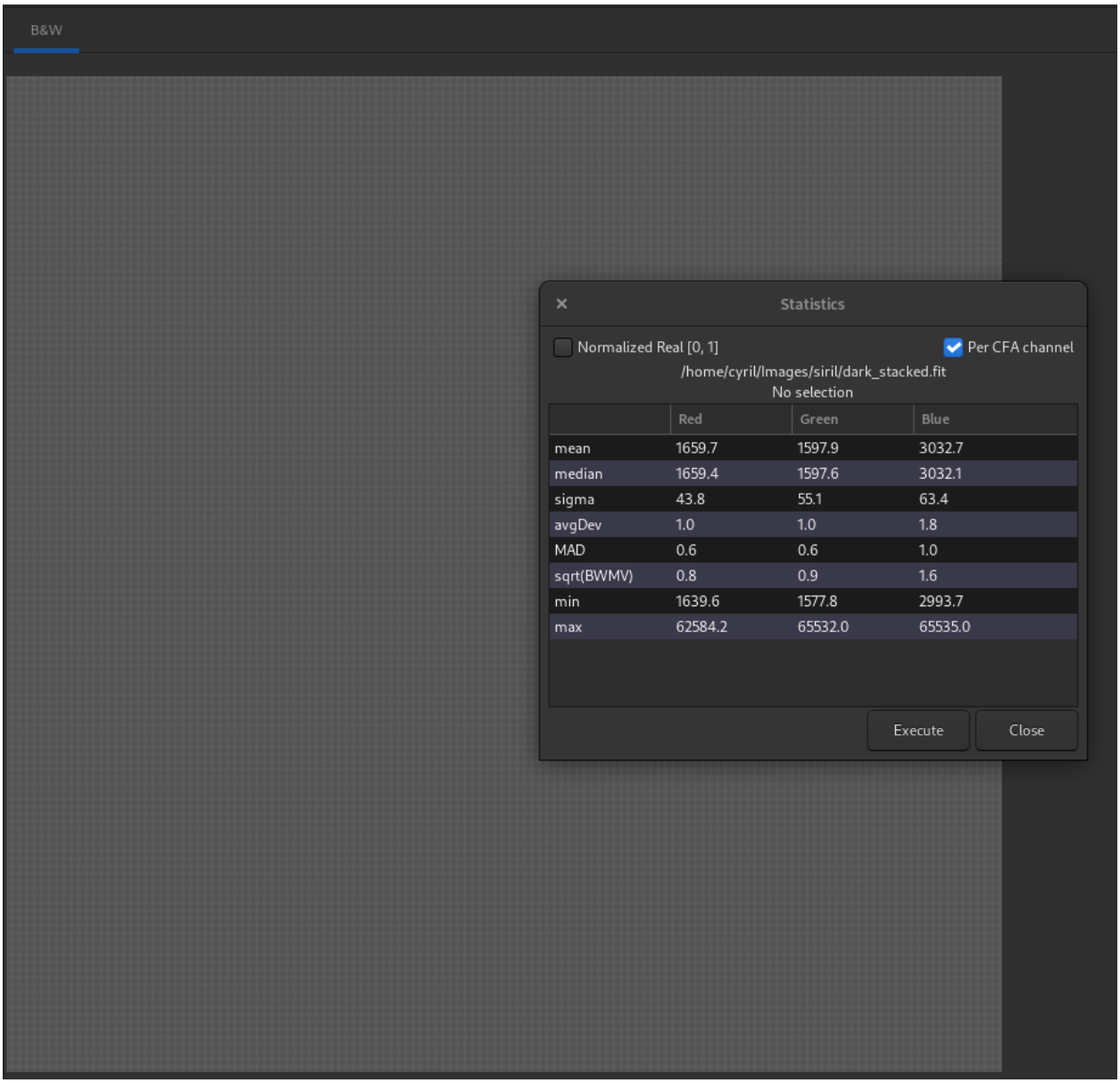



Fig. 20: A close look at the statistics shows that the median value of each channel is different, whereas they should be identical (or almost). Also, Bayer's pattern is clearly visible.

- If you have used the same settings for darks and biases, their median values should be very close to each other (at least with a cooled camera). Otherwise, it may mean you have you have a light leakage that has affected your darks (biases are less senstive as they are exposed for a much shorter time). So always inspect your master-dark to see whether there's a gradient or a brighter patch in the center. This is not to be confused with ampglow which is normal for certain cameras.
- We strongly recommend that you shoot your images in the same way: same software / same computer or astrobox /same image format. In fact, each program may use its own writing conventions, and images may no longer be compatible with each other. We often hear of users making all their images with an astrobox, and making the flats the next day directly with their DSLR. In this case, the images are often of different sizes, making calibration impossible.
- An error often encountered when running a script is the presence of JPG images in one of the input folders (darks/biases/flats/lights), most times snapshots saved by the acquisition software for faster browsing. The consequence of this kind of error is that calibration fails and stops, complaining that the images are not of the same size. In fact, since JPG images are already demosaiced, they have three channels, while RAW images have only one. Remove all JPG images from the input folders to fix this issue.
- Check that the flat is not overexposed. Flat frames are used to correct for pixel-to-pixel sensitivity variations in the sensor. If some pixels are overexposed, their true sensitivity may not be accurately represented, leading to incorrect corrections during the calibration process. An overexposed flat is the guarantee of a failed calibration.

  To check for overexposed pixels, you can load a flat subframe and use the *Image Processing → Histogram Transformation...* to show the histogram of the image. The snapshot below shows that one of the peaks is clipped to the right. As a prudent approach, you should always check that the right tail of the peak furthest to the right is not above 80%, to avoid going in a zone where your sensor may become non-linear.

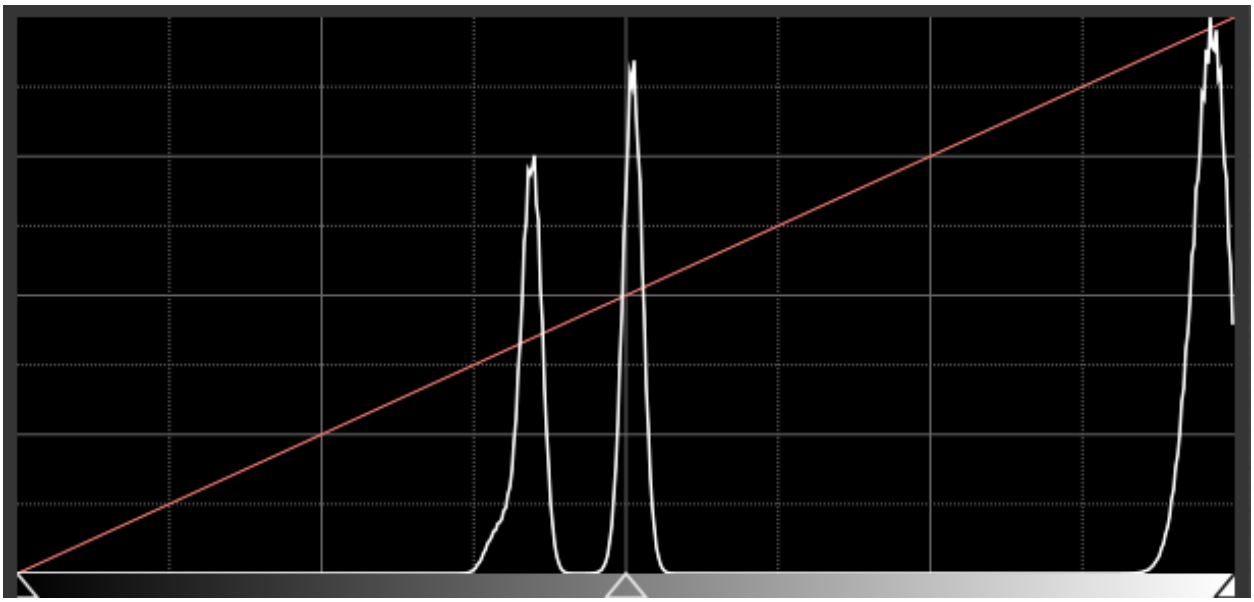


Fig. 21: White-clipping of a flat. When this happens, it means you should lower the gain or the exposure time.

## 9.3 Registration

Registration is basically the process of aligning the images from a sequence to be able to process them afterwards. All the processes described hereafter calculate the transformation to be applied to each image in order to be aligned with the reference image of the sequence.

Siril's strength lies in the wide variety of recording algorithms offered. Each method is explained below. Pressing the *Go register* button starts the registration of the sequence.

It is possible to choose the **registration channel**. Green is the default for color images, Luminance for monochrome. The (*) sign appearing after the channel's name means that registration data is already available for this layer. When processing images, registration data is taken from the default layer if available (for RGB images: Green, else fallback to Blue then Red).

### 9.3.1 Theory

#### Registration process

What we call Registration is in fact a three-step process:

1. Detect features to be matched in all the images
2. Compute the transformations between each image and the reference image
3. Apply the computed transformation to each image to obtain new images

Depending on the registration method selected, the 3 steps occur (or not) into a single process. Siril uses the most sensible defaults (choosing whether or not to apply the computed transformation) depending on the registration method selected, but understanding the internal machinery may help you to change this behavior to better suit your needs.

#### Algorithms

The table here below details the different algorithms used for the first 2 steps (detection and transformation calculation).

| Registration method | Feature detection | Transformation calculation | New seq |
|---|---|---|---|
| Global | *Dynamic PSF* | Triangles matching + RANSAC | Y |
| Global with 2 pass | | | N |
| 1-2-3 stars | PSF *minimization* in selection box | Singular Value Decomposition (2-3 stars) Difference (1 star) | N |
| Image Pattern alignment | cross correlation on selection box | | N |
| KOMBAT | Max of convolution in spatial domain on selection box | | N |
| Comet | PSF *minimization* in selection box | Shifts from velocity vector using timestamps | Y(*) |
| Manual | Your eyes | Your hand | N |
| Astrometry | *Dynamic PSF* | From astrometric solution | N |

**Note**

(*) Comet registration creates a new sequence file (with the prefix *comet_* by default) and creates "new" images. If registration data was present in the input sequence, the comet shifts are composed with this pre-existing registration data. The "new" images created are simply symlinks to the input images, so that they do not use up additional storage space. This change introduced in version 1.4 was made to clarify whether or not a sequence containing registration data, had only star alignment or star+comet shift alignment. Exporting a new sequence now makes it obvious that additional shifts are included. It also enables to use the same input sequence and generating multiple sequences aligned on different moving objects.

The table below lists the transformations compatible with each method, and if undistortion is available as well.

| Registration method | Shift | Euclidean | Similarity | Affine | Homography | Undistort |
|---|---|---|---|---|---|---|
| Global | subpixel | | x | x | x | x |
| 2 pass | subpixel | | x | x | x | x |
| 1-2-3 stars | subpixel (1-2-3) | (2-3) | | | | |
| Image Pattern alignment | pixel | | | | | |
| KOMBAT | pixel | | | | | |
| Comet | subpixel | | | | | |
| Manual | pixel | | | | | |
| Astrometry | | | | | x | x |

It is also important to keep in mind how the registered sequence is fed into the stacking process that is generally used right after registration:

- if the transformation consists only of pixel-wise shifts, the stacking algorithm can use these shifts on-the-fly when reading the images. It means you do not need to generate "registered images". This saves storage space and skips interpolation. It is, of course, at the expense of less accurate registration (i.e. subpixel accuracy) but is generally used on planetary/lucky imaging images where sampling is small. This can also be applied with a registration method which computes subpixel shifts. During the stacking process, the shifts will be rounded off to pixel precision. In any other case, meaning the stacking is fed with a sequence where the registration has computed transformations more complex than just shifts but the registered images have not been saved, Siril will emit a warning inviting you to export the registered images before proceeding to stacking.
- In all other cases, once the transformations have been computed, the transformed images need to be saved before proceeding to stacking, generally named with `r_` prefix.

### Image transformations

### Linear transformations

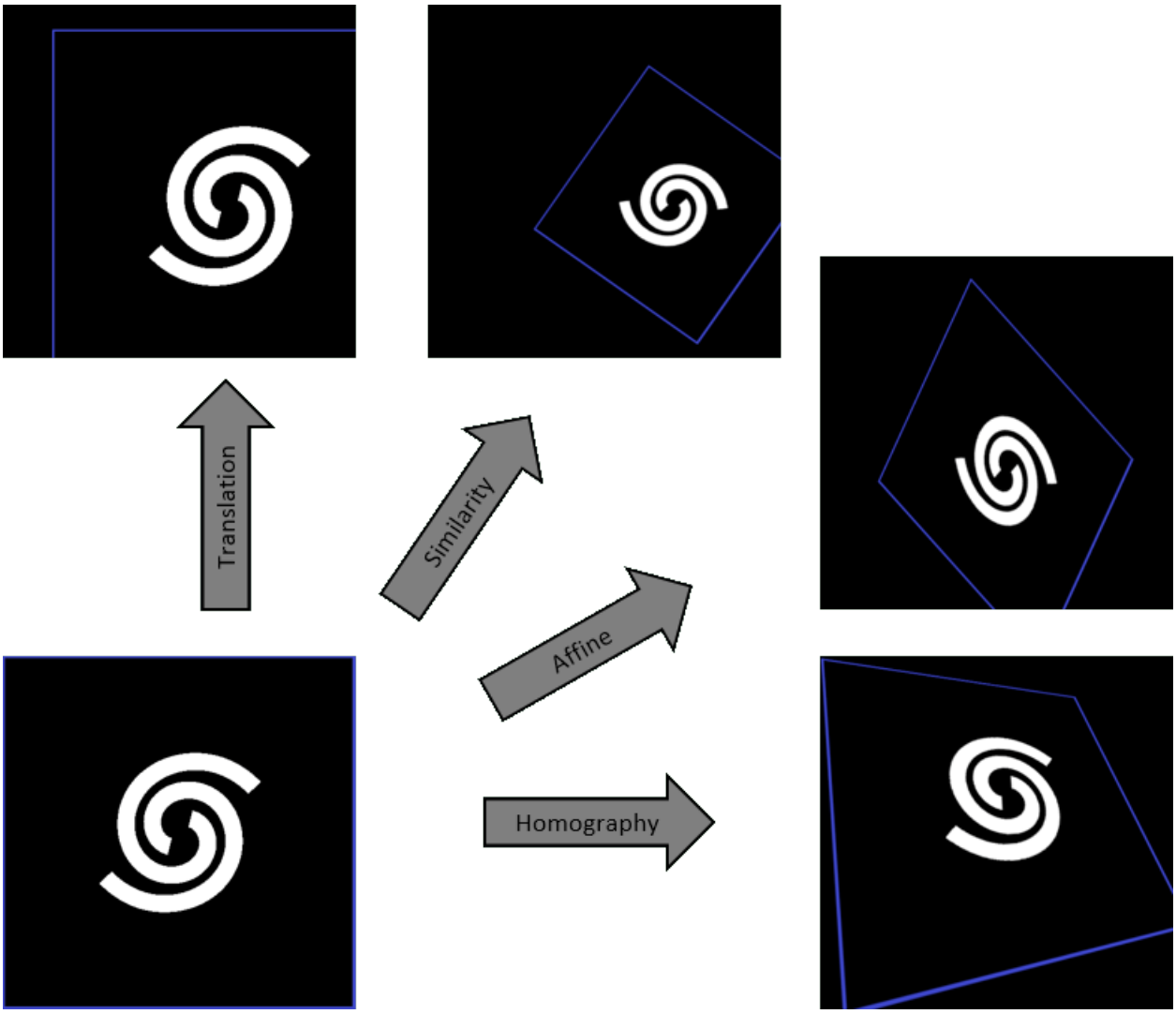
Siril uses linear transformations, with different degrees-of-freedoms, to map an image to the reference image:

- **Shift** is a 2 degree-of-freedom (x/y shifts) rigid mapping, well-suited for images with no distortion, no scaling and no field rotation. It needs only 1 pair of stars (or feature) to be matched to define the transformation.
- **Euclidean** is a 3 degree-of-freedom (x/y shifts + one rotation) rigid mapping, for images with no distortion, no scaling. It needs at least 2 pairs of stars to be matched to define the transformation.
- **Similarity** is a 4 degree-of-freedom (one scale, one rotation and x/y shifts) more rigid mapping than homography, well-suited for images with no distortion. It needs at least 2 pairs of stars to be matched to define the transformation.
- **Affine** is a 6 degree-of-freedom (two scales, one shear, one rotation and x/y shifts) more rigid mapping than homography, well-suited for images with little distortion. It needs at least 3 pairs of stars to be matched to define the transformation.
- **Homography** is the default transformation which uses an 8-degree-of-freedom transform to warp the images onto the reference frame. This is well-suited for the general case and strongly recommended for wide-field images. It needs at least 4 pairs of stars to be matched to define the transformation.

### Distortions

Since version 1.3, Siril can also account for distortions, for some of the registrations methods as listed in this *table*. Distortion coefficients handled by Siril follow SIP convention. This convention assumes that the pixel coordinates need to be corrected BEFORE trying to map them through a linear transformation. In WCS jargon, this is called a *prior* distortion (as opposed to a *sequent* distortion).

Those coefficients are used twice during the registration process:


Translation
Similarity
Affine
Homography

- the stars positions detected in "distorted" conditions are first corrected, both in the image to be aligned and the reference image (or the stars projected positions for astrometric registration). The *linear transformation* that maps the stars from the current image to the reference is then computed.
- When exporting the registered image, it is first corrected for distortion and then linearly projected to be aligned to the reference image. Note that this actually occurs in a single operation (the pixel mapping is computed as the composition of this non-linear correction and then the linear projection) so as to avoid interpolating pixel values twice. The reference image also undergoes this correction without the linear preojection.

#### Reference image

This is the image which is used as a common reference to compute the transformations that send all the images of the sequence onto this particular one.

If not set manually, the reference image is chosen with the following criteria:

- if the sequence has already been registered, it is the best image, in term of lowest FWHM or highest quality depending on the type of registration
- Otherwise, it is the first image of the sequence that is not excluded.

To specify an image as the reference, you can:

- Open the *Frame selector*, select the image to be set as the new reference and click the button *Reference Image*.
- Use the command *setref*. For instance, if you want to set image #10 as the reference:

```
setref 10
```

**Siril command line**

```
setref sequencename image_number
```

Sets the reference image of the sequence given in first argument. **image_number** is the sequential number of the image in the sequence, not the number in the filename, starting at 1

During stacking, the reference image is used as the normalization reference as well, if normalization is activated.

### 9.3.2 Registration methods

#### Global registration

This is the preferred algorithm to align deep-sky images with sufficient overlap.

The global matching is based on triangle similarity method for automatically identify common stars in each image [Valdes1995]. Our implementation is based upon the program match from Michael Richmond. Then, RANSAC [Fischler1981] algorithm is used on the star lists to further reject outliers and determine the projection matrix. The robustness of the algorithm depends on the ability to detect the stars while avoiding false detections. Siril has a very elaborate star detection algorithm that avoids as much as possible to select objects that are not stars in the fastest possible time. The detection of the brightest stars is usually the most important. However, if there is a need to detect fainter stars, then the *Dynamic PSF* window can be used to adjust the detection parameters.

There are some options associated with this alignment method.

The *Transformation* dropdown menu allows to choose between different transformations.

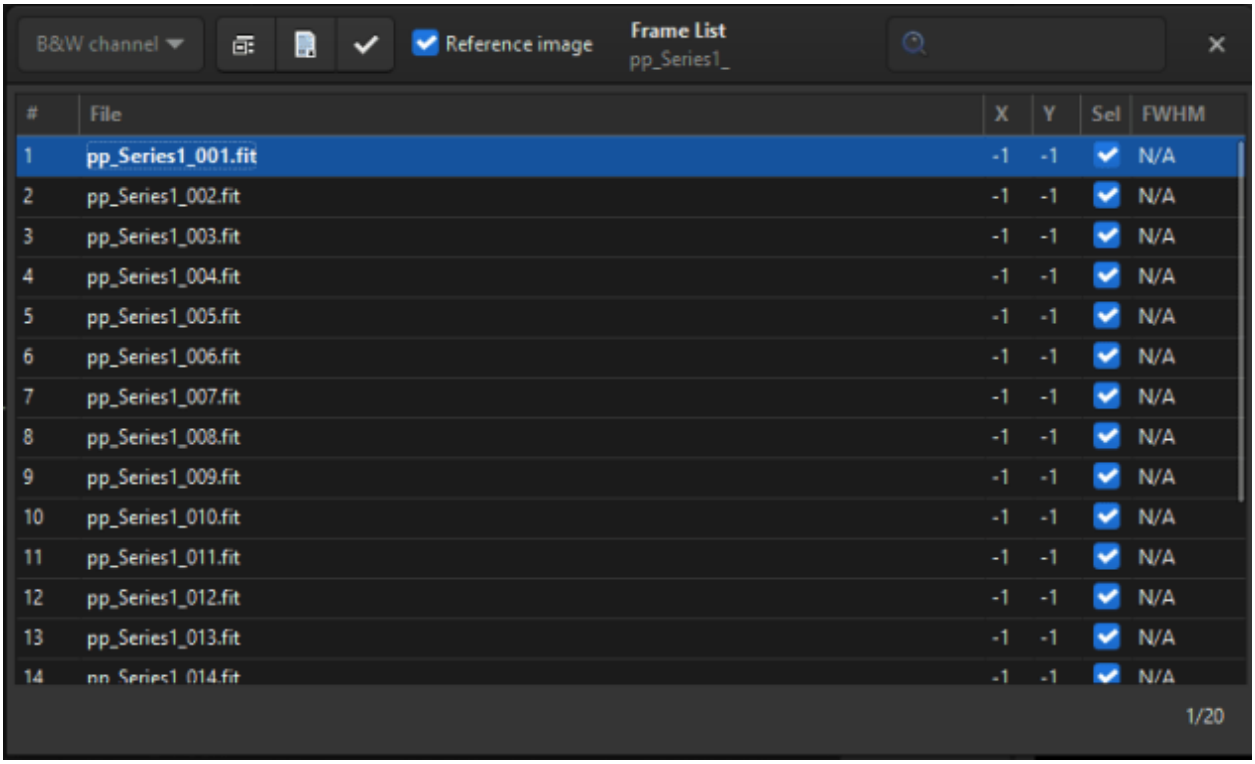



Fig. 22: The frame list dialog. You can browse all images in the sequence.

> **Warning**
>
> The initial star matching uses triangle similarity algorithm, in consequence the minimum of star pairs must be at least of 3 for **Shift**, **Similarity** and **Affine** and of 4 for **Homography**.

Other registration options are:

- The *Minimum Star Pairs* button sets the minimum number of star pairs a given frame can have in relation with the reference frame. If a given light frame has less star pairs, it will not be registered. To the right of this option is a button that opens the Dynamic PSF tool.
- The option *Maximum Stars Fitted* defines the maximum number of stars to be searched for in each frame (default 2000). The larger this value, the more stars will potentially be detected, resulting in a longer detection but more accurate registration.
- Use the option, *Match stars in selection*, if you want to perform the Global Star Alignment algorithm within the selected area in the reference image. If no selection is done, this option is ignored.
- The *Undistortion* dropdown menu allows to choose between different corrections:
  - None
  - From image
  - From FITS/WCS file
  - From masters

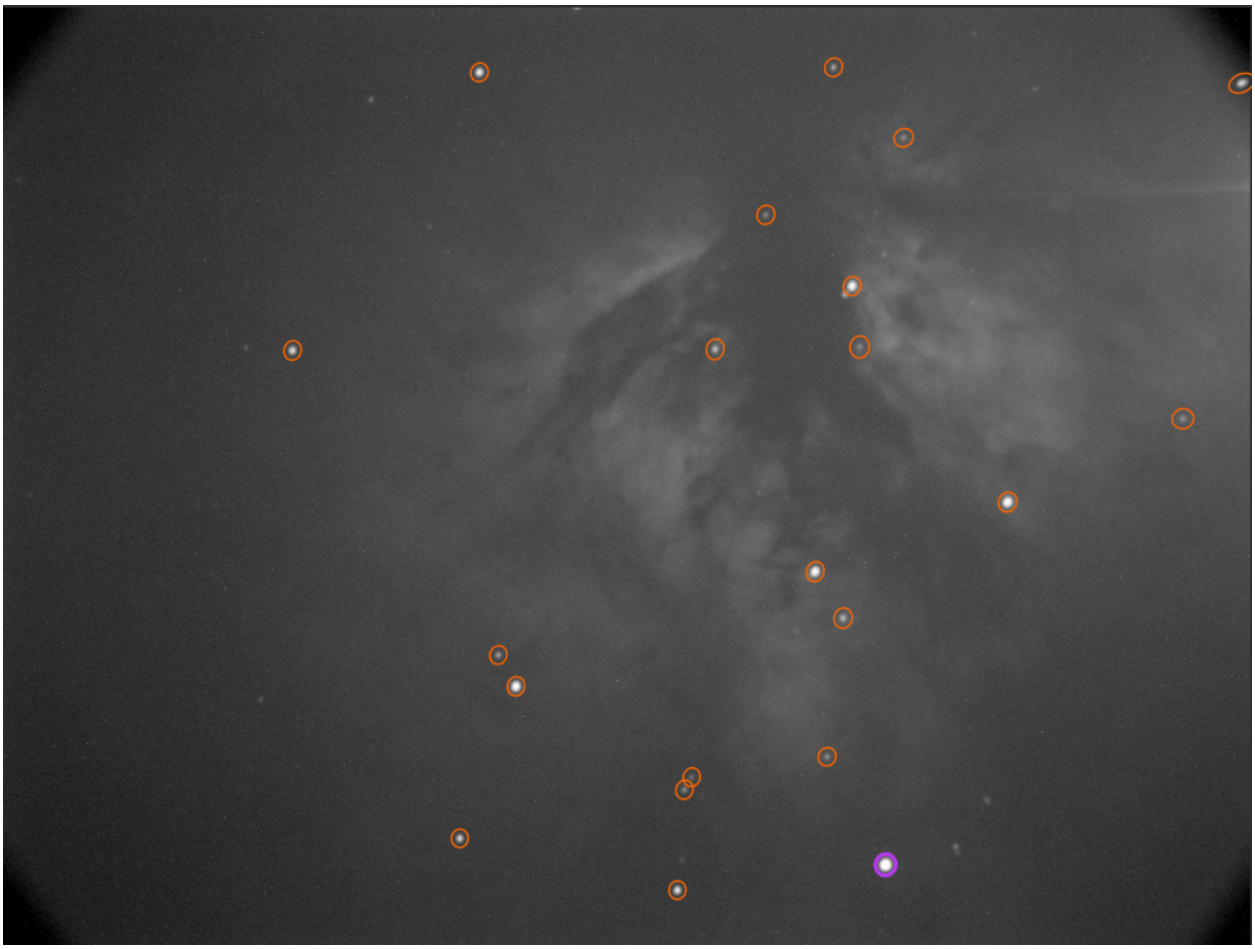


Fig. 23: Automatic detection of stars in a single frame

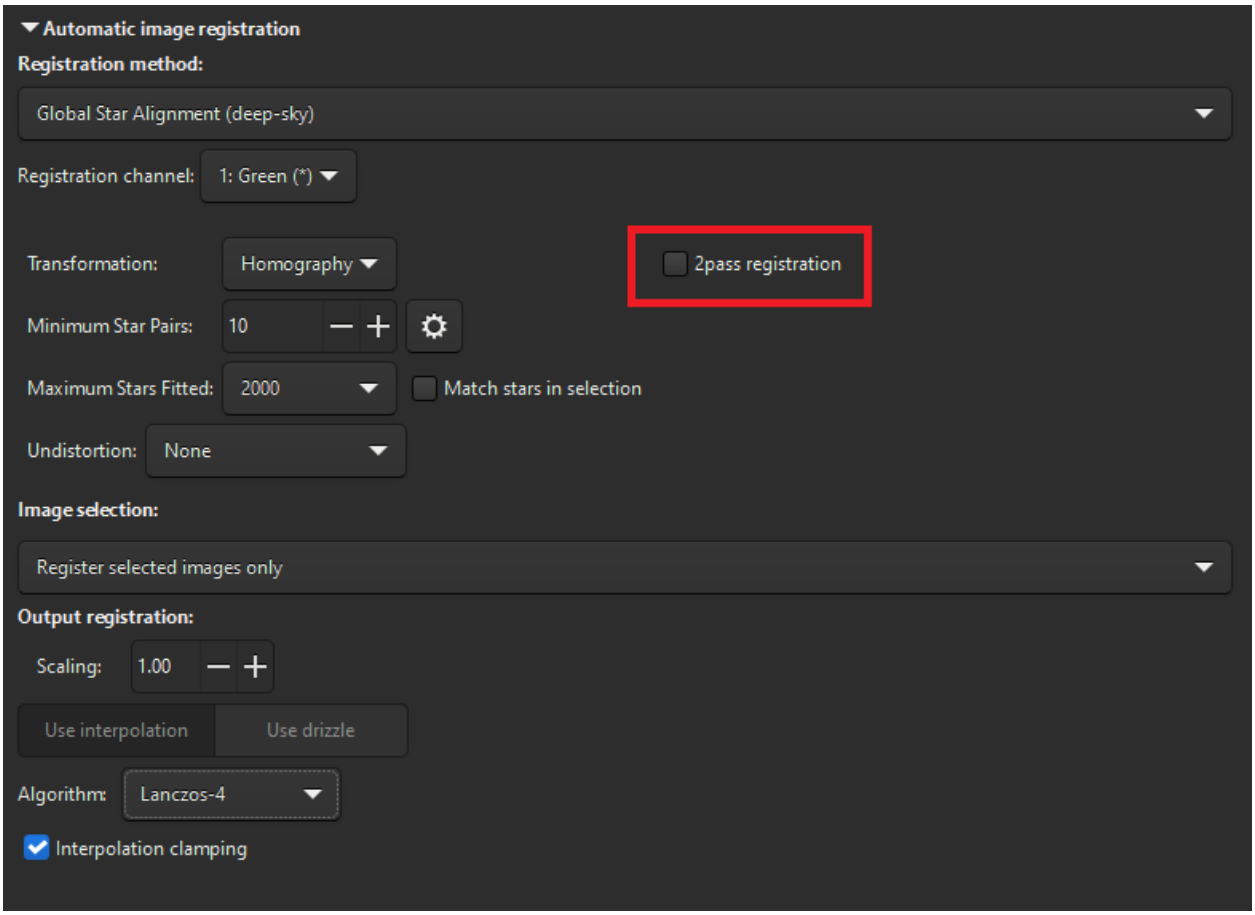



Fig. 24: Global registration options

**From image** uses the astrometric solution from the currently loaded image.
**From FITS/WCS file** will open a file chooser to select a FITS image or a `*.wcs` file that has distortion coefficients.
**From masters** will fetch the master distortion file corresponding to each image. This option can be useful if you are aligning images from two or more sessions and the sensor has been moved wrt. the optics, in which case distortion coefficients may not be the same.

**Tip**

For this last option *From masters* to work, you will need to have set a master distortion path in the *preferences*. To create a master file, go *here*.

The options at the bottom let you:

- Filter out images that have been unselected from the sequence.
- Choose between interpolation and drizzle for exporting the images. Those are the same as in the *Output registration* section, they are not further explained here.

**Siril command line**

```
register sequencename [-2pass] [-selected] [-prefix=] [-scale=]
register sequencename ... [-layer=] [-transf=] [-minpairs=] [-maxstars=] [-
↪nostarlist] [-disto=]
register sequencename ... [-interp=] [-noclamp]
register sequencename ... [-drizzle [-pixfrac=] [-kernel=] [-flat=]]
```

Finds and optionally performs geometric transforms on images of the sequence given in argument so that they may be superimposed on the reference image. Using stars for registration, this algorithm only works with deep sky images. Star detection options can be changed using **SETFINDSTAR** or the *Dynamic PSF* dialog.

All images of the sequence will be registered unless the option **-selected** is passed, in that case the excluded images will not be processed.
The **-2pass** option will only compute the transforms but not generate the transformed images, **-2pass** adds a preliminary pass to the algorithm to find a good reference image before computing the transforms, based on image quality and framing. To generate transformed images after this pass, use SEQAPPLYREG.
If created, the output sequence name will start with the prefix "r_" unless otherwise specified with **-prefix=** option. The output images can be rescaled by passing a **-scale=** argument with a float value between 0.1 and 3.

**Image transformation options:**

The detection is done on the green layer for colour images, unless specified by the **-layer=** option with an argument ranging from 0 to 2 for red to blue.
**-transf=** specifies the use of either **shift**, **similarity**, **affine** or **homography** (default) transformations respectively.
**-minpairs=** will specify the minimum number of star pairs a frame must have with the reference frame, otherwise the frame will be dropped and excluded from the sequence.
**-maxstars=** will specify the maximum number of stars to find within each frame (must be between 100 and 2000). With more stars, a more accurate registration can be computed, but will take more time to run.

**-nostarlist** disables saving the star lists to disk.

**-disto=** uses distortion terms from a previous platesolve solution (with a SIP order > 1). It takes as parameter either **image** to use the solution contained in the currently loaded image, **file** followed by the path to the image containing the solution or **master** to load automatically the matching distortion master corresponding to each image. When using this option, the polynomials are used both to correct star positions before computing the transformation and to undistort the images when output images are exported.

**Image interpolation options:**

By default, transformations are applied to register the images by using interpolation.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument.

**Image drizzle options:**

Otherwise, the images can be exported using HST drizzle algorithm by passing the argument **-drizzle** which can take the additional options:

**-pixfrac=** sets the pixel fraction (default = 1.0).

The **-kernel=** argument sets the drizzle kernel and must be followed by one of **point**, **turbo**, **square**, **gaussian**, **lanczos2** or **lanczos3**. The default is **square**.

The **-flat=** argument specifies a master flat to weight the drizzled input pixels (default is no flat).

Note: when using **-drizzle** on images taken with a color camera, the input images must not be debayered. In that case, star detection will always occur on the green pixels

Links: *setfindstar*, *psf*, *seqapplyreg*

### 2pass registration

To activate this option, simply tick the *corresponding box* after choosing *Global registration* in the method drop-down.

This performs only the first (of 2 passes), allowing the reference frame to be chosen from detected star information instead of automatically choosing the first frame of the sequence. The proposed options are similar to the *Global Registation* algorithm but this method does not create any sequences and all alignment information is saved in the `seq` file.

During star detection, Siril sets a maximum of 2000 stars to be found (this can also be changed with the appropriate option). In case more than one image has reached the maximum star limits, the star lists of all images are screened again. A new minimum detection threshold is defined to be able sort the images by both number of stars detected and FWHM.

The star lists of all images are saved, the `.fit(s)` extension being replaced by `.lst`. This allows to re-run the 2pass algorithm very quickly with different parameters, say different transformation. In case the star detection have been modified, the process detects these changes and re-run the analysis as required.

This registration must generally followed by *Apply Existing Registation* in order to apply the transformation and build a new sequence, unless you have chosen to compute `Shift`.

These lines perform a 2pass registration on a sequence named *pp_light* and applies it. The output is a sequence *pp_light*.

```
# Align lights in 2 passes
register pp_light -2pass
seqapplyreg pp_light
```

These lines perform a 2pass registration on a sequence named *colors* and applies it while cropping the output images to the minimum coommon area. The output is a sequence *pp_colors*. This can be useful before compositing mono images (the areas which are not common to all images are cropped).

```
# Align layers in 2 passes and crop away borders
register colors -2pass
seqapplyreg colors -framing=min
```

### 1-2-3 stars registration

When the images contain few stars, for example in the case of DSO Lucky Imaging images where the frame exposure is less than one second. It is possible that the global registration algorithm fails, even if you change the detection parameters in the *Dynamic PSF* window. It may then be interesting to make a manual detection of the stars you want to align. This is the interest of the 1, 2 or 3 star registration algorithm.

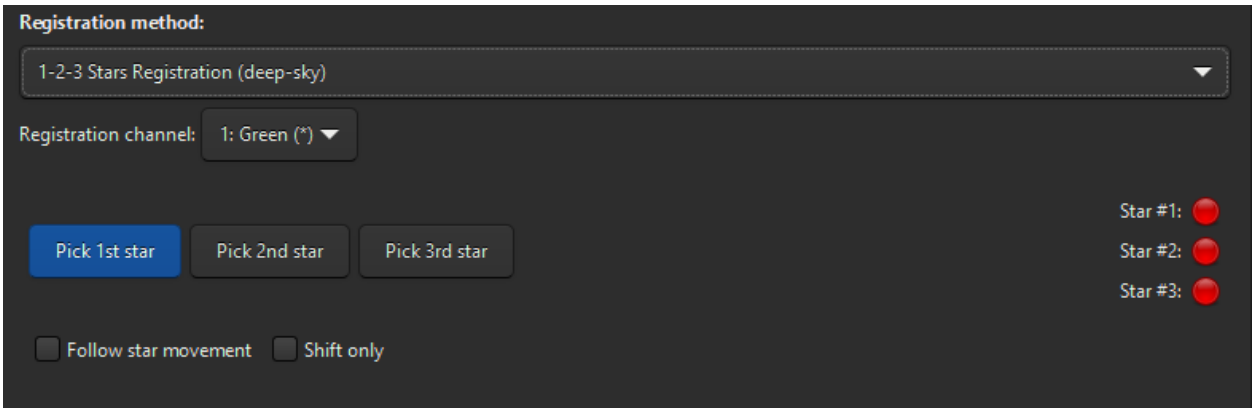



Fig. 25: 1-2-3 stars registration options

The principle of this method is to draw a selection area around a star and click on the *Pick 1st star* button, then so on.

- If only one star is selected, only the translation between the images will be calculated. Therefore the *Shift only* button is automatically selected. The shift values are then storred in the `seq` file.
- If two or three stars are selected, then the rotation can be calculated and applied to create a new sequence. However, if the *Shift only* option is selected, which is not mandatory, only the shifts will be calculated.

The option *Follow star movement* use the position of the star(s) found in the previous image as new centre for the current image registration. This allows the selection area to be smaller, registration faster, and accounts for drift or images with a large number of stars.

> **Warning**
>
> Enabling this option requires the registration to not be parallelized, it will run on one CPU core only.

### Image Pattern alignment (planetary-full disk)

This is a simple registration by translation method using cross correlation in the spatial domain.

This method is fast and is used to register *planetary* movies, in which constrasted information can be seen on large areas of the image. It can also be used for some deep-sky images registration. Nevertheless keep in mind that it is a single point alignment method, which makes it poorly suited for high definition planetary alignment. But, it does effectively anchor the images to stabilize the sequence. Simply draw a selection around the object (the planet for example) and make sure that its movement during the sequence is contained within the selection. Only the translation can be calculated with this method.

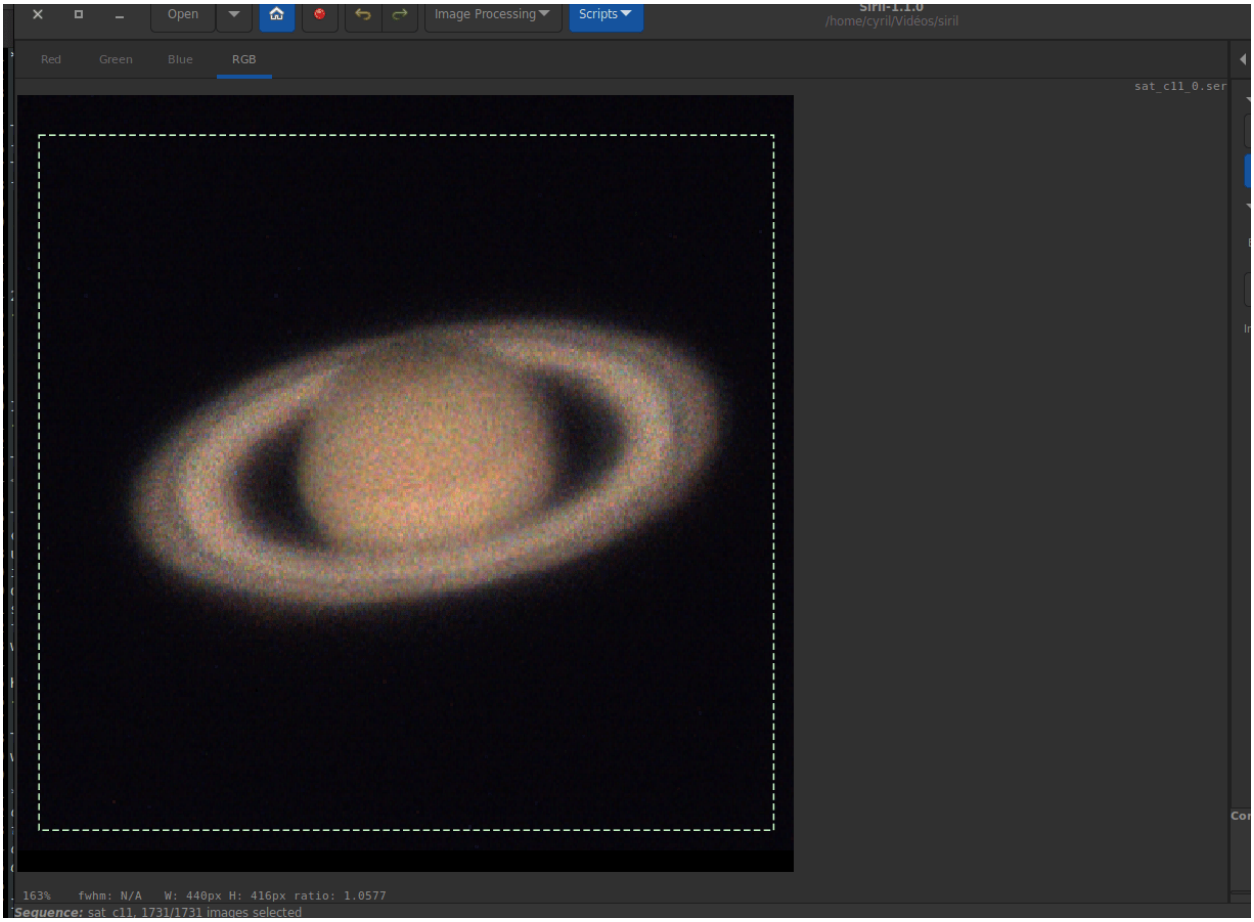



Fig. 26: Pattern alignment options

### KOMBAT

This method comes from the OpenCV library, a library widely used in Siril. They explain:

*It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. (You can check docs for more details). It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.*

In practice, simply draw a selection around the object (the planet for example) and make sure that its movement during the sequence is contained within the selection. Only the translation can be calculated with this method.

### Comet/Asteroid registration

The cometary registration tool works in a very simple way, in two steps.

1. With the frame selector, select the first image of the sequence, surround the comet nucleus, then click on the button *Pick object in #1*.
2. Then select the last image of the sequence, surround the nucleus of the comet, then click on the button *Pick object in #2*.

The comet velocity $\Delta x$ and $\Delta y$ is computed in pixel per hour if everything is ok.

**Warning**

The alignment of the comet must be done on images whose stars have been previously aligned. Either via a new sequence, with the global alignment, or by having saved the registration information in the `seq` file, via a 2-pass or astrometric registration.

**Note**

To fully function, the images must have a timestamp.

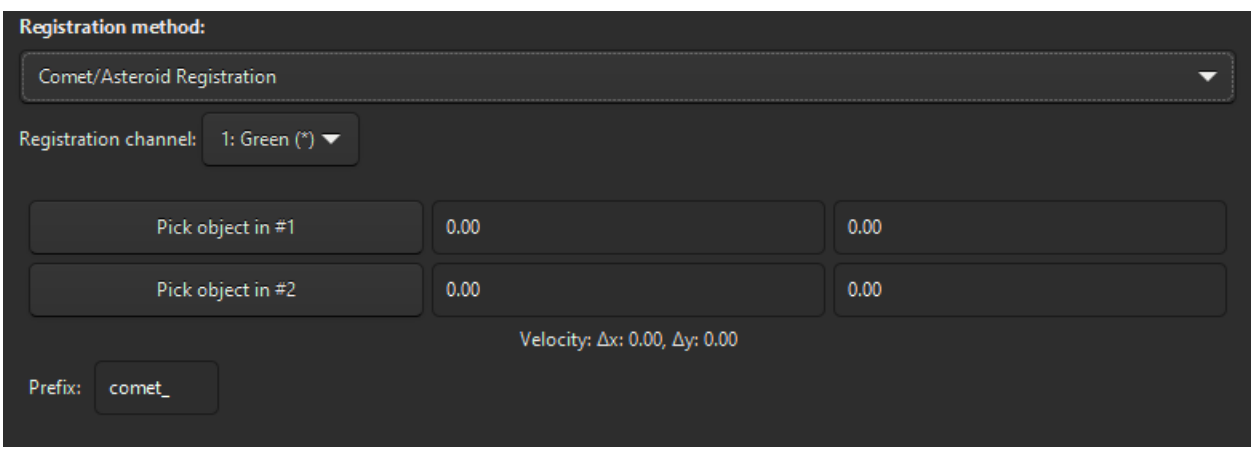



Fig. 27: Comet registration options

**Tip**

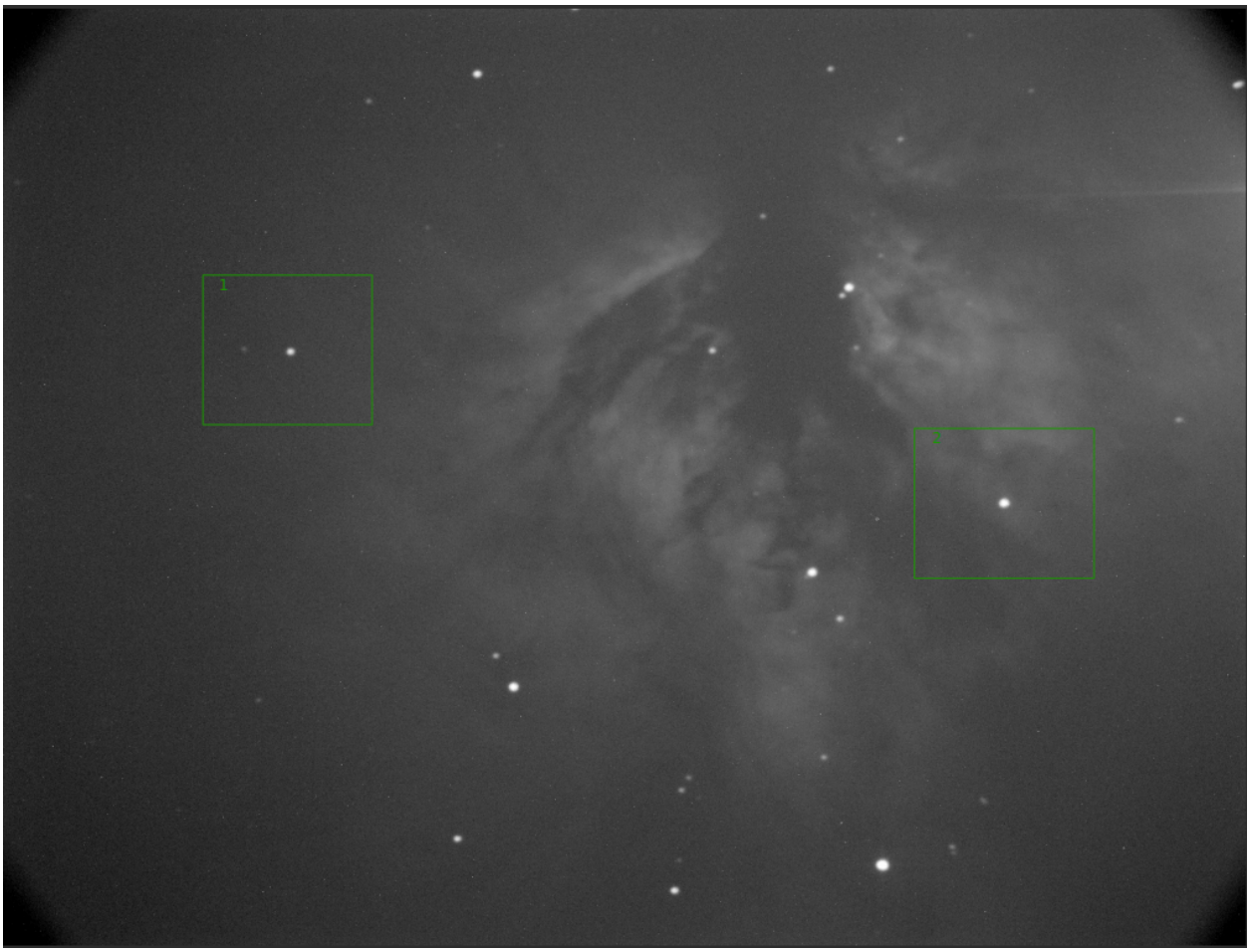
If the PSF detection fails to detect an object, it will return the center of the box that was drawn. This can be handy if you want to align on an object that is not visible on your subs. Use the *solar system object* annotations to plot the position of an asteroid and draw a box around the marker to pick its position.

This method will ouput a new sequence file, with the sequence name prepended with the *prefix* defined (`comet_` by default). This does not however create the new images but symlinks to the original images. For Windows users, please make sure you have enabled developer mode, otherwise, it will make hard copies. See also the note below *this table*.

### Manual Registration

This last method of registration is very particular, which explains its separate position, and allows to align images manually. Of course, only the translation between images is allowed.

The first thing to do is to define two previews in the image. Clicking on the button *Set first preview* will initialize the first preview. You then need to click on an area of the image, ideally a star in the vicinity of an edge of the image to set the preview area. A click on the second button *Set second preview* allows to do the same on a second point.




It is very important to have a reference image already set with the *Frame selector*. By default, it is the first image. The user is free to choose the one he wants. It will be used as a reference layer, seen by transparency, to align the images manually with the numerical buttons. Then, browse the image one by one to apply the same method to the whole sequence.

### Astrometric registration

Introduced in version 1.3, this is the prefered mode for assembling mosaics or images with little overlap. It can also be useful to register stacks issued from different set-ups (different optics, different cameras, different field of views etc...).

It does not have an entry in the registration method drop-down as the information to export the registered images has already been computed when *platesolving the sequence*. All you need to do is to *Apply Existing Registation*.

Undistortion will be applied as defined when platesolving the sequence, meaning if the images were plate-solved using a SIP order larger than 1, then undistortion will automatically be included. Unless you have a perfectly optically flat field, it is usualy a good idea to platesolve using SIP, as shsown below, with and without distortion correction.

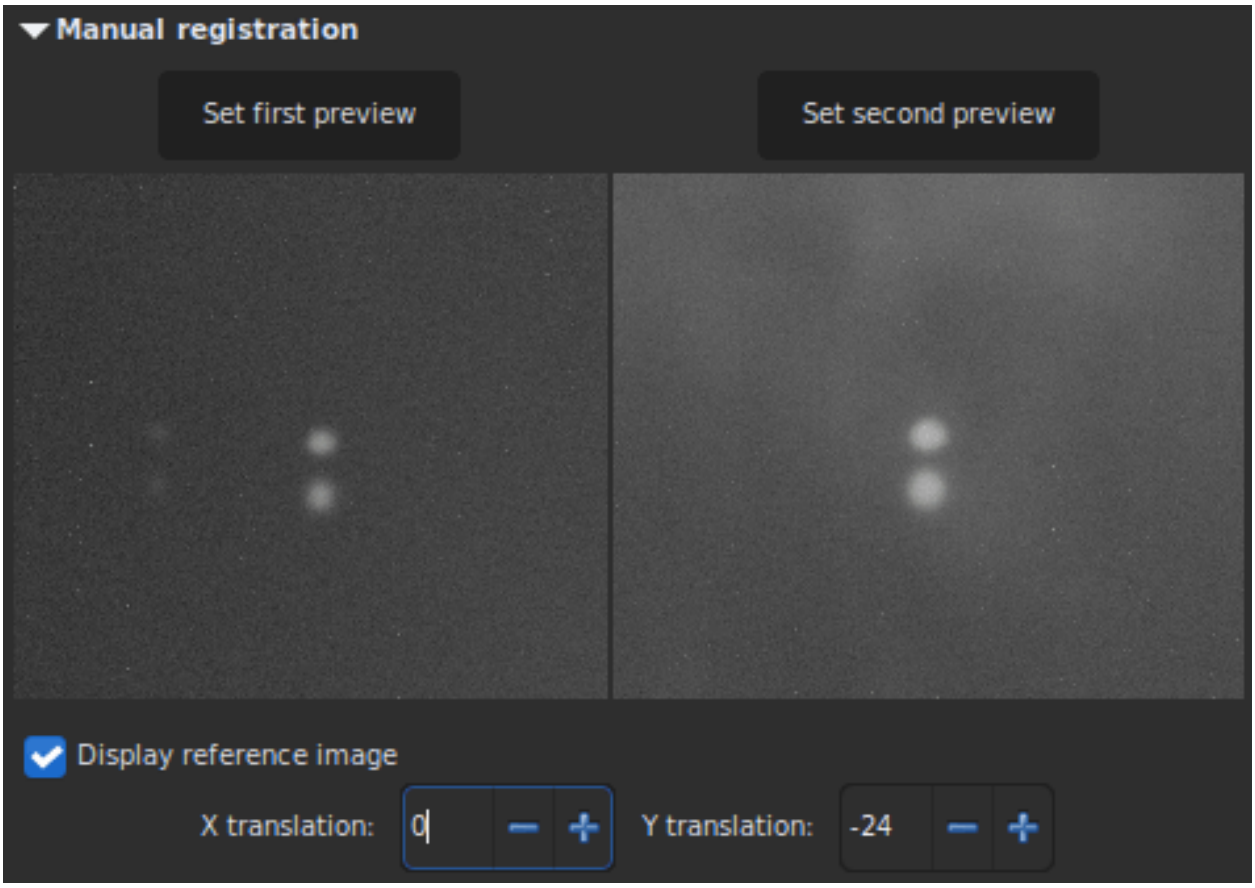



Fig. 28: The Y-shift is too large, same stars on different frames do not overlap.

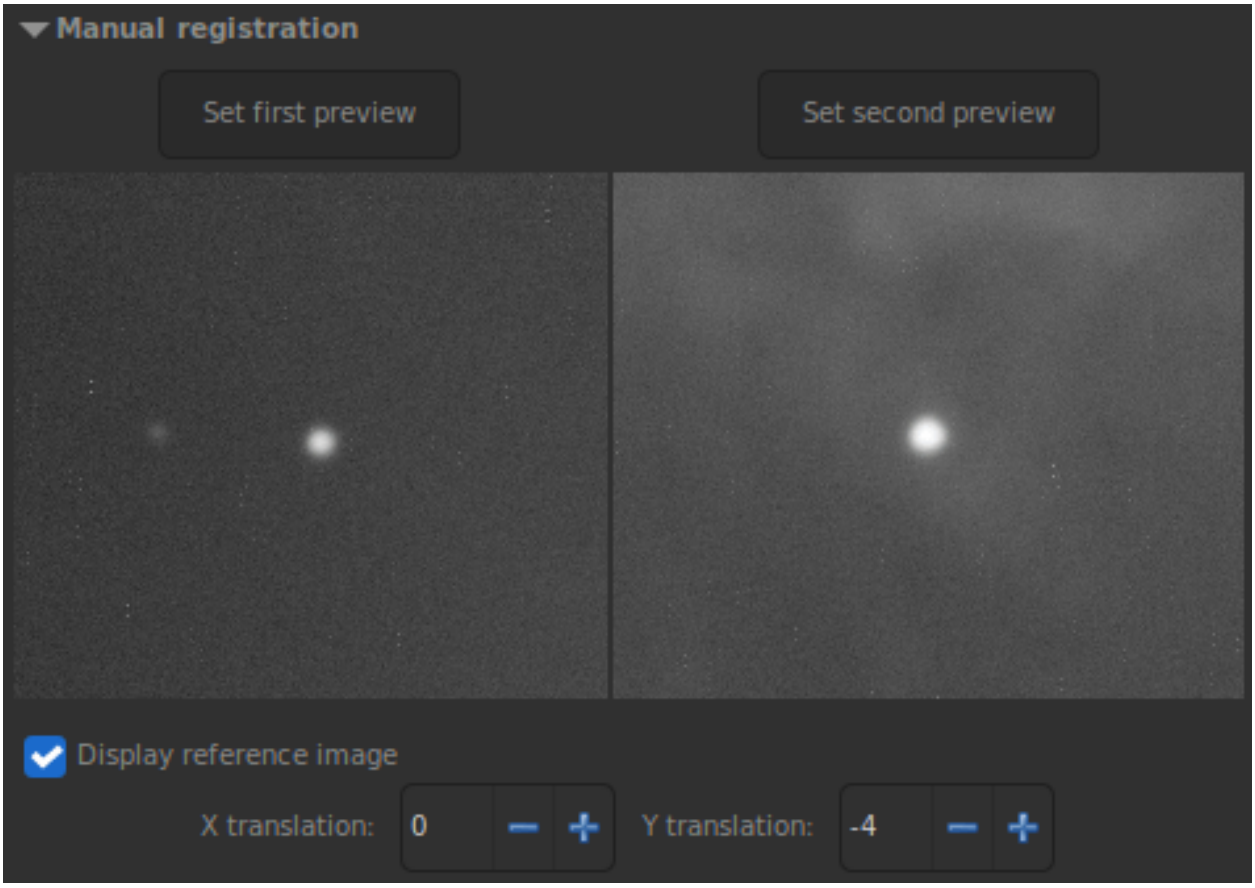



Fig. 29: X- and Y-shift look fine. The current image is aligned to the reference one.

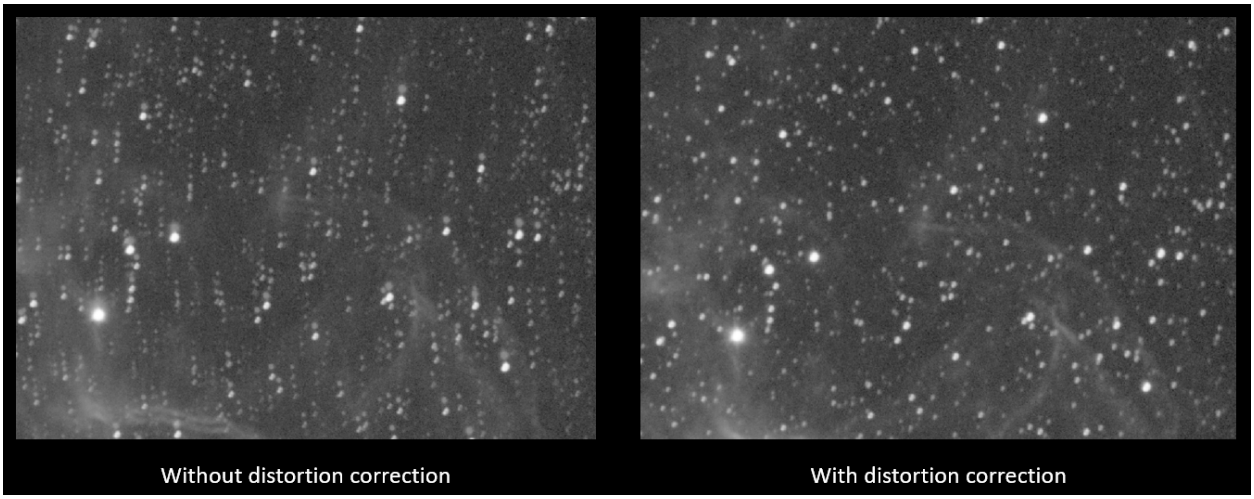
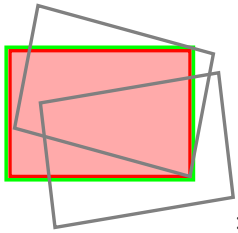
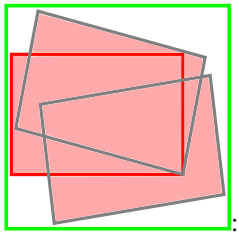
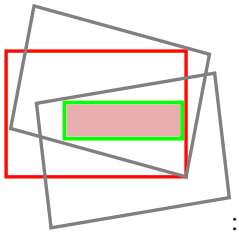
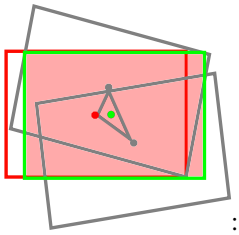



Fig. 30: Effect of undistortion on two overlapping panels after registration

### Apply Existing registration

This is not an algorithm but rather a commodity to apply previously computed registration data stored in the sequence file. The interpolation method can be selected in the *Output Registration* section. You can also use image filtering to avoid saving unnecessary images, as in stacking *Image rejection*. There is also a *Drizzle* option to apply registration using Drizzle instead of interpolation. See the *Drizzle section* for details.

Four framing methods are available:

-  : **current** uses the current reference image. This is the default behavior.
-  : **maximum** (bounding box) adds a black border around each image as required so that no part of the image is cropped when registered.
-  : **minimum** (common area) crops each image to the area it has in common with all images of the sequence.
-  : **center of gravity** determines the best framing position as the center of gravity (cog) of all the images.

> **Tip**
>
> Introduced in Siril 1.3, `max` mode does not export images with black borders, that encompass the full resulting image. The images are exported with the necessary projection and the relative shifts required to compose the final image is kept in the the resulting seq file.

*Estimate* button will launch the framing computation, without actually exporting the images. This information can be interesting to know in advance the size of the exported images. This accounts for the **framing** method selected and the **scaling** factor chosen in the *Output Registration*.

When pressing *Estimate*, the console will show an output like this:

```
Output image: 7893 x 5254 pixels (assuming a scaling factor of 1.30)
```

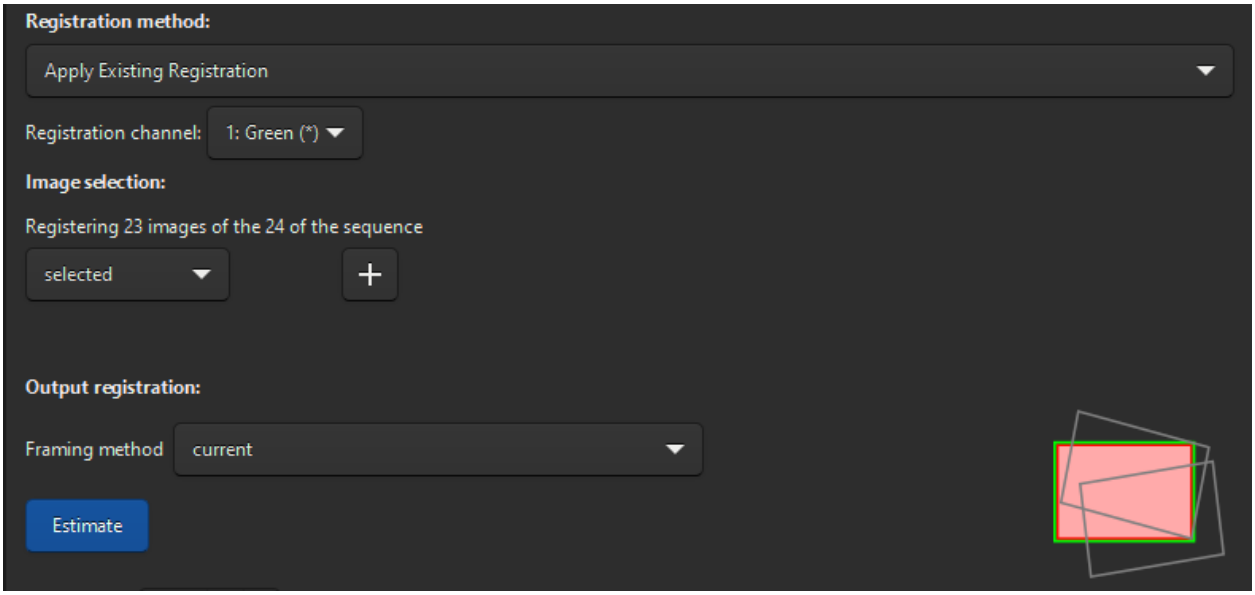



Fig. 31: Apply Existing registration options

**Siril command line**

```
seqapplyreg sequencename [-prefix=] [-scale=] [-layer=] [-framing=]
seqapplyreg sequencename ... [-interp=] [-noclamp]
seqapplyreg sequencename ... [-drizzle [-pixfrac=] [-kernel=] [-flat=]]
seqapplyreg sequencename ... [-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-
↪filter-round=value[%|k]] [-filter-bkg=value[%|k]] [-filter-nbstars=value[%|k]] [-
↪filter-quality=value[%|k]] [-filter-incl[uded]]
```

Applies geometric transforms on images of the sequence given in argument so that they may be superimposed on the reference image, using registration data previously computed (see REGISTER).

The output sequence name starts with the prefix **"r_"** unless otherwise specified with **-prefix=** option.

The registration is done on the first layer for which data exists for RGB images unless specified by **-layer=** option (0, 1 or 2 for R, G and B respectively).

The output images can be rescaled by passing a **-scale=** argument with a float value between 0.1 and 3.

Automatic framing of the output sequence can be specified using **-framing=** keyword followed by one of the methods in the list { current | min | max | cog } :

**-framing=max** (bounding box) will project each image and compute its shift wrt. reference image. The resulting sequence can then be stacked using option **-maximize** of STACK command which will create the full image encompassing all images of the sequence.

**-framing=min** (common area) crops each image to the area it has in common with all images of the sequence.

**-framing=cog** determines the best framing position as the center of gravity (cog) of all the images.

**Image interpolation options:**

By default, transformations are applied to register the images by using interpolation.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument.

**Image drizzle options:**

Otherwise, the images can be exported using HST drizzle algorithm by passing the argument **-drizzle** which can take the additional options:

**-pixfrac=** sets the pixel fraction (default = 1.0).

The **-kernel=** argument sets the drizzle kernel and must be followed by one of **point**, **turbo**, **square**, **gaussian**, **lanczos2** or **lanczos3**. The default is **square**.

The **-flat=** argument specifies a master flat to weight the drizzled input pixels (default is no flat).

**Filtering out images:**

Images to be registered can be selected based on some filters, like those selected or with best FWHM, with some of the **-filter-*** options.

Links: *register*, *stack*

With filtering being some of these in no particular order or number:

```
[-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-filter-round=value[%|k]] [-
↪filter-bkg=value[%|k]]
[-filter-nbstars=value[%|k]] [-filter-quality=value[%|k]] [-filter-incl[uded]]
```

Best images from the sequence can be stacked by using the filtering arguments. Each of these arguments can remove bad images based on a property their name contains, taken from the registration data, with either of the three types of argument values:

- a numeric value for the worse image to keep depending on the type of data used (between 0 and 1 for roundness and quality, absolute values otherwise),

- a percentage of best images to keep if the number is followed by a % sign,

- or a k value for the k.sigma of the worse image to keep if the number is followed by a k sign.

It is also possible to use manually selected images, either previously from the GUI or with the select or unselect commands, using the **-filter-included** argument.

### 9.3.3 Output Registration

This frame contains all the output elements for the sequence. You can choose between Interpolation and Drizzle to export the images.

> **Warning**
>
> They may not be both available depending on the nature of the input sequence:
>
> - for mono sequences, both Interpolation and Drizzle may be selected
> - for CFA sequences (undebayered color images), only Drizzle is available
> - for RGB sequences (debayered color images), only Interpolation is available

Both methods share the options:

- *Scaling*, a value between 0.1 and 2, that is used to rescale the output images.
- *Prefix*, the prefix that will be preprended to form the exported sequence name (defaults to `r_`).

### Interpolation

Tthe pixels of the resulting images are interpolated by an algorithm that is left to the user's choice. There are 5 possible interpolation algorithms, plus a **None** option:

- Nearest Neighbor
- Bilinear
- Bicubic
- Pixel Area Relation
- Lanczos-4
- None

The best efficient interpolation methods are generally bicubic and Lanczos-4 (used by default). However, they usually require the *Clamping interpolation* option to be enabled to avoid ring artifacts around the stars, which makes the interpolation operation twice as slow. We still enable it by default because it gives a significant improvement in resolution, with often a 8% smaller in FWHM compared to Siril 1.0 default method of pixel area. It is possible the clamping is not useful for your images, we recommend you to compare the results with your images.

The special case of **None** is reserved for the case of global registration and Apply Existing registration. If you want to export or save a sequence that contains only translation, without using interpolation (so as not to modify the pixel values), you should select **None**.

### Drizzle

The button *Drizzle* activates the drizzle algorithm for the processing of this sequence. See the *Drizzle section* for details.

> **Warning**
>
> The counterpart of this technic is that the amount of memory and disk space needed to create and process drizzled images is multiplied by the square of the Drizzle scale factor.

### Astrometric solution of the registered images

When registered images are exported, they inherit the astromertic soluion of the reference image if any. Otherwise, their previous solution is wiped. Obviously, the new solution for each image accounts for the transformations it has undergone through the registration process.

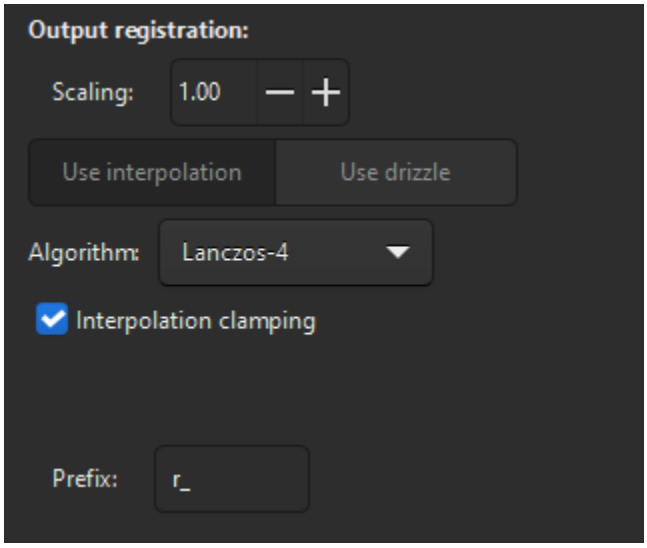



Fig. 32: Output options for interpolation

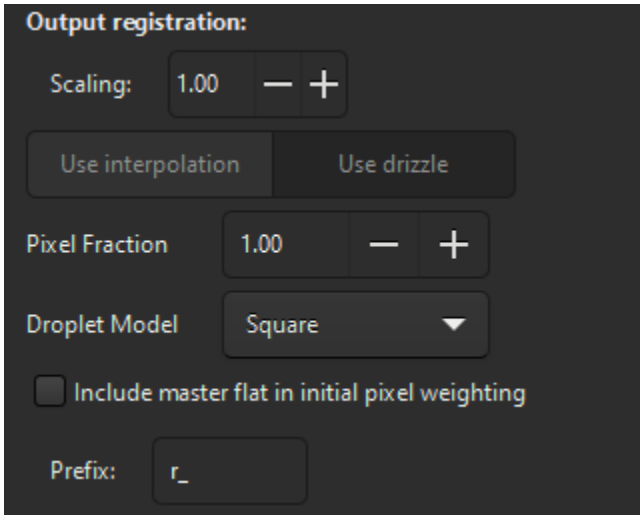



Fig. 33: Output options for drizzle

In case a distortion solution is found in the reference image but no undistortion was applied when computing the transformations - for instance if you have not selected any `Distortion` from the drop-down in global or 2-pass methods, a warning will be displayed in the Console. The distortion information will be kept in the registered images. In case they are significant, you may see their effect when stacking. In which case, you will need to register again using a distortion specification.

### 9.3.4 References

## 9.4 Drizzle

Variable-pixel linear reconstruction, and more commonly referred to as Drizzle, was developed for Hubble Space Telescope (HST) work by Andy Fruchter and Richard Hook [FruchterHook1997], initially for the purposes of combining dithered images of the Hubble Deep Field North (HDF-N). This algorithm can be thought of as a continuous set of linear functions that vary smoothly between the optimum linear combination technique (interlacing) and shift-and-add. This often allows an improvement in resolution and a reduction in correlated noise, compared with images produced by only using shift-and-add.

There is an excellent page summarising the technique and providing a good graphical representation of how the pixel data "drizzles" down from the coarse input pixel grid onto a finer output pixel grid here.

The degree to which the algorithm departs from interlacing and moves towards shift-and-add depends upon how well the PSF is subsampled by the shifts in the input images. In practice, the behavior of the Drizzle algorithm is controlled through the use of a parameter called the pixel fraction, that represents the amount by which input pixels are shrunk before being mapped onto the output image plane. At a pixel fraction of 0, drizzle is equivalent to pure interlacing; at a pixel fraction of 1, it is equivalent to shift-and-add.

To understand the difference between drizzle and the interpolation methods of applying registration, firstly consider how the standard interpolation method works. The registration data takes the form of a 3x3 Homography matrix, which encodes an 8 degrees of freedom linear transformation from one set of coordinates to another (i.e. from each frame to the reference frame). This is used to map the values from each pixel in each input image onto the correct place in the output image, aligning the output with the reference image. The actual alignment uses an interpolation method, which can be selected in the registration options. Interpolation results in a smearing of the point spread function, especially when upscaling images. It can also result in artefacts, although Siril implements a clamping mechanism to minimize this.

Drizzle, by contrast, turns each pixel in the input image into a droplet, and drizzles it through a grid pattern onto the output reference frame. Each droplet has a size, and by choosing a scaled-up output pixel grid but smaller droplet size you can achieve improved resolution **if** your imaging train is undersampled. (If your sampling is correct for the resolving power of your telescope, Drizzle can't produce detail beyond the diffraction limit.) This comes at the cost of increased image noise: because each droplet "paints" a smaller area in the output image, the average coverage of droplets per output pixel in the final stack is reduced.

Note that Drizzle doesn't replace all of the registration process: you still use the Global Star Alignment, 1-2-3 Star Alignment, Comet registration or whichever registration method you wish prior to drizzling: it is an alternative only to the interpolation method used when applying registration.

**Warning**

The Drizzle process involves taking multiple frames and drizzling them into an output frame. The input is the set of frames and their WCS solutions, and the output is the single drizzled image. In Siril the process is split into Drizzle registration and subsequent stacking, however the intermediate artefact (the set of images representing the individual images drizzled onto the common output grid but not yet stacked) is not something that has significance in its own right. Individual frames in the drizzled sequence can **and will** look weird: the reference frame often looks different to the others, especially when drizzling CFA data, as a result of its special property of having zero shift with respect to the output frame, and other frames may show Moiré patterns. **Don't panic!** This is simply

an intermediate stage in the overall Drizzle process, and all these apparent oddities will be resolved in the stacking stage.

### 9.4.1 Uses of Drizzle

There are 3 main reasons you may prefer Drizzle to using an interpolation method to apply registration.

- **Resolution enhancement** If your image is significantly undersampled, you may be able to achieve an improvement in resolution using Drizzle that you could not achieve using the *Interpolation upscaling x2* registration option.
- **CFA imaging** If your images have a CFA pattern (i.e. if you use a one-shot camera or dSLR), Drizzle provides a significant improvement over debayering. This is sometimes referred to separately to Drizzle as *Bayer Drizzle*, but really it's exactly the same process. When drizzling a CFA image the CFA color of the current droplet sets which channel of the output image it lands on, whereas when drizzling a mono image all droplets land in the (only) mono output channel. Drizzling CFA images avoids the artefacts that occur with all debayering algorithms, which gives improved noise characteristics when strongly stretching images. This supports improved noise reduction and deconvolution for drizzled CFA sequences compared with debayered and registered CFA images, and makes the shadows look better.
- **Avoid artefacts** It is possible to drizzle a sequence using scale = 1, pixel fraction = 1 and achieve essentially the same result as applying registration with one of the interpolated methods. You might wish to consider this if you see interpolation artefacts from the standard interpolated method (though these are generally effectively suppressed by the clamping feature). Note that drizzle can produce different artefacts of its own (see the "Some Common Issues" heading below) however these can be completely avoided by choice of drizzle kernel or by having a greater number of input frames, and are generally handled perfectly by stacking.

### 9.4.2 Limitations of Drizzle

- Drizzle is a little slower compared to interpolation, particularly the preferred square kernel. If you are using older or slower hardware, you may prefer the legacy method.
- When used for upsampling, Drizzle achieves its resolution improvements at the cost of increased image noise. Therefore you may wish to collect more integration time when drizzling than when using an interpolation-based upsampling method.
- The above issue is *especially* true for CFA images. Consider that only 1 in 2 of the pixels are green, and only 1 in 4 are red or blue. Therefore for the red or blue channels, CFA drizzle effectively already involves the same level of reduction in droplet coverage as a 2x upscale drizzle. If you upscale on top of that, you need as much droplet coverage as you would for a 4x upscale drizzle! Therefore it is generally recommended to drizzle CFA images at scale = 1.

### 9.4.3 Comparison

The following image shows a comparison between drizzle and the legacy upscaling method. The image is Ha extracted from an OSC session with a dual band filter. On the left you can see the result of the legacy OSC_Extract_HaOIII script, which extracts the Ha data captured by the red pixels in the OSC Bayer matrix as a half-size image and uses OpenCV upscaling with lanczos4 interpolation to produce an image tat matches the size of the OIII image.

On the right you can see the result of the updated script (available through the siril-scripts repository), which extracts the Ha data captured by the red pixels in the OSC Bayer matrix as a half-size image and drizzles it using scale = 2.0, pixel fraction = 0.5, to produce an image that matches the size of the OIII image.

Viewing it at 100% scale it is clear that the drizzled stack restores much of the resolution of the optical system that is undersampled by the spaced-out red pixels in the Bayer matrix: it looks much sharper, and the numbers confirm it: the average fwhm in the left-hand image is 3.59, whereas in the right-hand image it is 3.25.

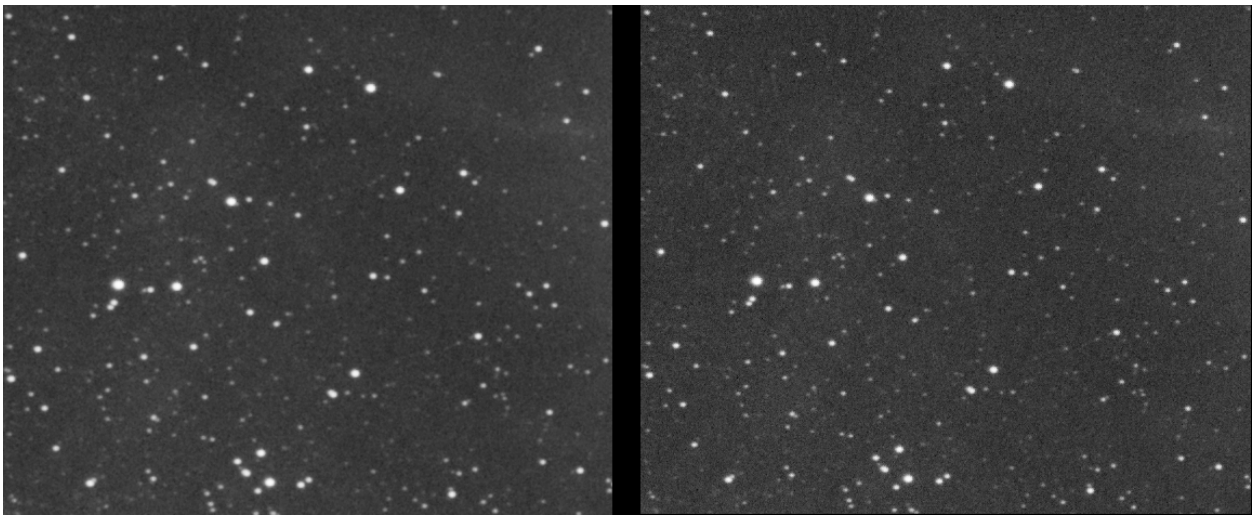


Fig. 34: Comparison between interpolation upscaling and drizzle

### 9.4.4 Workflow and User Interface

### 9.4.5 Mono Workflow

#### Calibration

For mono images nothing changes in the calibration tab. Calibrate as you normally would.

#### Registration

#### Drizzling

#### Scale

Scale sets the scale of the drizzle output image with regard to the input image. A typical drizzle scale for an undersampled mono image is 2.0. This means that the input will be drizzled onto an output pixel grid with twice the resolution. (If your input reference image was 1024 x 512 pixels, your output image would be 2048 x 1024 pixels.) Note: becase the image represents the same area of the sky, although there are **twice as many** pixels along each axis in the output image, effectively each output pixel is **half as wide** and **half as tall**.

> **Tip**
>
> The greater the scale, the sparser each drizzled output image and the fewer pixels end up getting stacked into each output pixel. This results in a noisier image: **the resolution gain provided by drizzle comes at the expense of noise**. This has to be mitigated by using a greater overall integration time than you would need without drizzling to a greater resolution.

#### Pixel Fraction

Pixel Fraction sets the size of the droplet taken from the input grid. Consider a drizzle scale of 2.0: since the output pixels are half as wide and half as tall, that means that in order for each input pixel "droplet" to be the same size as an output pixel it should be shrunk to half the linear dimensions. This is a pixel fraction of 0.5. A good rule of thumb is that the pixel fraction should be roughly the reciprocal of the drizzle scale (with some kernels it helps to set it a little bigger than this, in order to reduce pixels that receive zero input from any drizzled droplets).

There is scope for experimentation with the pixel fraction: setting a larger pixel fraction means that each input droplet

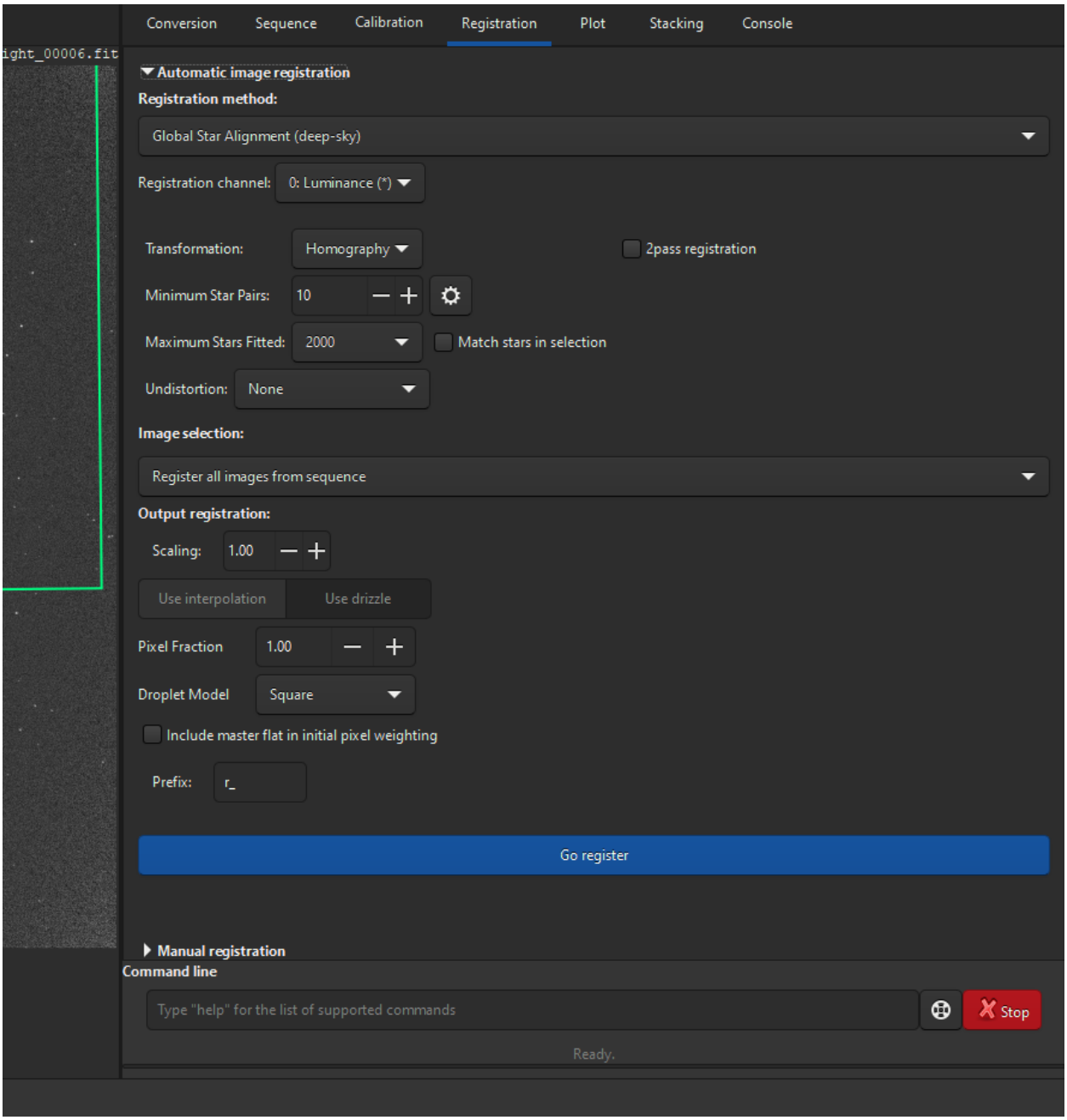



Fig. 35: Registration tab showing drizzle settings

will influence more output pixels. On the other hand, setting a smaller pixel fraction means that each input droplet will influence fewer output pixels. The "point" kernel is a special case where the pixel fraction is zero (and with this kernel selected, the pixel fraction setting has no effect).

### Droplet Model

Siril's Drizzle implementation provides several droplet models:

- **Square**. This models the droplet as a square droplet aligned exactly with the input pixel. It is accurately mapped to the output reference frame, and it works well at any scale and pixel fraction. This is the default kernel.

  > **Tip**
  >
  > The Square kernel is mathematically flux preserving by construction: each input pixel's flux is distributed into output pixels in exact proportion to area overlap. Total counts are preserved identically. This makes it both robust and ideal for applications where you need strict surface brightness conservation (e.g. diffuse structures or extended sources).

- **Point**. This models the droplet as a point at the center of the input pixel. It is the limit of the square kernel with a very small pixel fraction. This will prodce the lowest correlated noise but is prone to leaving holes unless you have very large numbers of well-dithered frames. If you find holes or artefacts in your final stack, use the Square kernel instead. This kernel is very fast.
- **Turbo**. This is a simplification of the Square kernel. It assumes that rotation between the input and output reference is negligible. This results in a much faster computation, but is approximate. It is a "quick and dirty" kernel originally intended for generating intermediate products within the HST workflow, but it may be of some use in lucky imaging due to its speed relative to Square.
- **Gaussian**. This models the droplet as a Gaussian centered on the center of the input pixel with a FWHM equal to *pixfrac*. The droplets in the Gaussian model are round and are recommended where improved preservation of PSFs (star shapes) is important.

  > **Tip**
  >
  > [Avila2015] recommends the Gaussian kernel for applications involving point source photometry. Despite not being mathematically flux preserving, the Gaussian kernel reduces the blockiness typical of square kernels (and avoids the high frequency ringing characteristic of Lanczos kernels, producing smooth, centrally peaked PSFs that are closer to analytical forms used in photometry. While Gaussians don't preserve flux perfectly the trade-off is much better behaved PSFs, which is more important for extracting accurate stellar photometry. It may also be more aesthetically pleasing in a final image.

- **Lanczos2** and **Lanczos3**. These kernels model the droplet as a Lanczos function centered on the center of the input pixel.

  > **Warning**
  >
  > The Lanczos kernels are specially designed for resampling an image from one World Coordinate System (WCS) onto another **at the same scale**. They should only be used with scale == pixfrac == 1.0.

  Lanczos kernels provide a good option for single-image registration at native scale: Lanczos3 produces tigher average FWHM than any other kernel or OpenCV interpolation, but it is significantly slower than other kernels.

#### Initial Pixel Weighting

When a droplet lands on the output pixel grid, it may cover more than one output pixel. In fact, one output pixel may be covered by multiple droplets, by only a fraction of a droplet or even by no droplets at all. The contribution of each input pixel can be weighted by the master flat, so that pixels from areas with higher SNR (less vignetting) are weighted more highly. Unless you have peculiar flats this generally makes no practical difference.

To enable the master flat, check the *Include master flat in initial pixel weighting* checkbox.

> **Warning**
>
> The master flat must be set in the *Calibration* tab!

#### Let's Drizzle!

Once all the options are set, click the *Go register* button.

#### Stacking

You can now stack your drizzled sequence as normal. Note that for some combinations of drizzle scale and droplet size, some rejection models will work better than others. In particular, if you have significant numbers of "zero input" or null pixels, there will be fewer values to use in rejection. MAD may be a good one to try if your usual rejection method struggles.

The GIF below shows a comparison of a stack of 37 images, in one case with registration applied using interpolation and in the other case with registration applied using drizzle. It is clear that the stack made with drizzled data is significantly sharper than the one using data registered using interpolation.

Fig. 36: Comparing registration applied with drizzle and with interpolation. Click to enlarge view.

### 9.4.6 CFA Workflow

#### CFA Calibration

For one-shot color (OSC) images, uncheck the *Debayer before saving* checkbox. This represents a change to previous workflows, but for drizzling it is essential that the CFA pattern is preserved in the drizzle input sequence.

#### CFA Registration with Drizzling

You can use drizzle directly after a *Global Registration* or by *applying existing registration* for all the other registration methods that do not export images directly.

#### CFA Scale

Scale sets the scale of the drizzle output image with regard to the input image. In OSC camera images each pixel only records a single color: red, green or blue. The pixels have a color filter array (CFA) applied to them and this determines which pixels respond to red, green and blue wavelengths. Thus all the pixels are sparsely distributed compared with a mono sensor in which all pixels are sensitive to whatever light passes through the filter. In both Bayer pattern and X-Trans CFAs the red and blue pixels are particularly sparse in the input frames, therefore increasing the drizzle scale above 1.0 will require even more frames to provide enough drizzle coverage and reach an acceptable level of noise.

For a typical OSC sensor application where the seeing is well matched to the nominal sampling of the sensor it is recommended to apply CFA drizzle with scale = 1.0 and pixfrac = 1.0. This will restore resolution in each color channel (which is effectively being undersampled because of the spacing of the colored pixels in the CFA) and avoid conventional debayering artefacts. If you wish to upsample the image as well by using scale > 1.0, be aware that the

pixels available in each channel will be getting even more sparse, and you will need even more data to ensure adequate coverage and contain noise to an acceptable level.

> **Tip**
>
> For OSC drizzle, start with scale = pixel fraction = 1.0.

### CFA Pixel Fraction

Pixel Fraction sets the size of the droplet taken from the input grid. The same comments apply here as are described above for the mono workflow.

### CFA Droplet Model

The same choice of drizzle kernels applies for CFA drizzle as for mono drizzle. Note that the kernels that are particularly prone to generating null pixels can be tricky when used for CFA drizzle. If you have tens of thousands of frames as in a planetary video, turbo may work fine (and will be fast!) however for deep sky sequences with smaller numbers of frames it is recommended to stick to the *square* or *Gaussian* kernels (and bear in mind as mentioned above that Gaussian is not flux preserving, so if you intend to do anything involving photometric techniques *square* is preferred).

### CFA Initial Pixel Weighting

As with mono drizzling a master flat may be specified. To enable the master flat, check the *Include master flat in initial pixel weighting* checkbox.

> **Warning**
>
> The master flat must be set in the *Calibration* tab!

### Let's Bayer Drizzle!

Once all the options are set, click the *Go register* button.

### Stacking your CFA data

You can now stack your drizzled sequence as normal, noting the same comments on rejection as for mono drizzle (these may be more apparent with CFA drizzle if you have inadequate coverage to support some of the outlier rejection algorithms, owing to the greater sparseness of input pixels in each channel).

> **Tip**
>
> If you are drizzling your CFA data to gain resolution, it is possible you may be disappointed when comparing results with stacked debayered images. There *are* generally gains, but they may be marginal (e.g. a few percent improvement in fwhm) and generally will not be nearly as impressive as the resolution gains to be had from drizzling undersampled mono data.
>
> The reason for this is that debayering already restores some of the lost resolution. The various debayering algorithms work differently but they generally all rely on principles of spatial and spectral correlation to infer some of the resolution missing in one channel based on information obtained from the other channels. [Losson2010]
>
> **The real reason to drizzle CFA data is that the drizzled result has much cleaner noise.** It looks less "grainy" (i.e. it lacks the structure that can be seen in the background of a typical debayered CFA stack) and is thus easier to reduce using noise reduction techniques and gives more consistent data for photometric applications such as color

calibration. When stretched hard to bring out faint features just above the background, the resulting background looks more natural.

### Bayer Drizzle Comparison

The animation below shows a comparison between CFA drizzle with two different pixel fractions and two of the classical debayering algorithms.

Fig. 37: Comparison of CFA drizzle (here captioned as Bayer Drizzle) with classical debayering algorithms

- VNG is provided as a basic reference: note the color artefact around the brighter stars.
- RCD is quite good with round objects like stars.
- Bayer Drizzle 1.0 gives results very close to RCD but with a better noise and background
- Bayer Drizzle 0.5 gives better resolution at the cost of more noise. The trade-off that pixel fraction gives between resolution and noise is evident. With a smaller pixel fraction CFA drizzle needs more data to achieve the same noise performance.

## 9.4.7 Some Common Issues

**Tip**

**DON'T PANIC** - the following results may look a bit weird when you view an individual drizzled sub, but they are not bugs - the algorithm is functioning as intended. In most cases they naturally resolve themselves during stacking, in the remaining cases they can be resolved by changing the drizzle parameters or by including more frames of data.

### Moiré Patterns

Due to the nature of the drizzle algorithm, when upscaling some output pixels may not receive any input. These are referred to as "null pixels" and they have a zero value. Some kernels compensate for this, effectively by limiting the pixel fraction, so that all output pixels receive some input, but others do not.

Output pixels that don't receive any input are black: since they typically occur in patterns based on the geometry of the transformation from the input frame, they typically look like Moiré patterns, as shown below:

Don't worry about this! Siril ignores pixels that are exactly 0 in stacking, so as long as you have enough input frames and the dither positions are suitably scattered, all the pixels will receive coverage from enough pixels and the output stack will be fine. However if you are stacking with a lower number of input frames and this is causing problems, try a different drizzle kernel. Here is exactly the same image drizzled with exactly the same scale and pixel fraction, but with the square kernel instead of the turbo kernel. The result is different, and the patterns are no longer evident.

### Patchy Stacks

One issue you may see when stacking drizzled data, if there are too many null pixels, is that you may get an odd patchy appearance in the final result:

This typically occurs with the point, turbo or lanczos kernels. You can fix it by using the square or Gaussian kernels or by having more input frames.

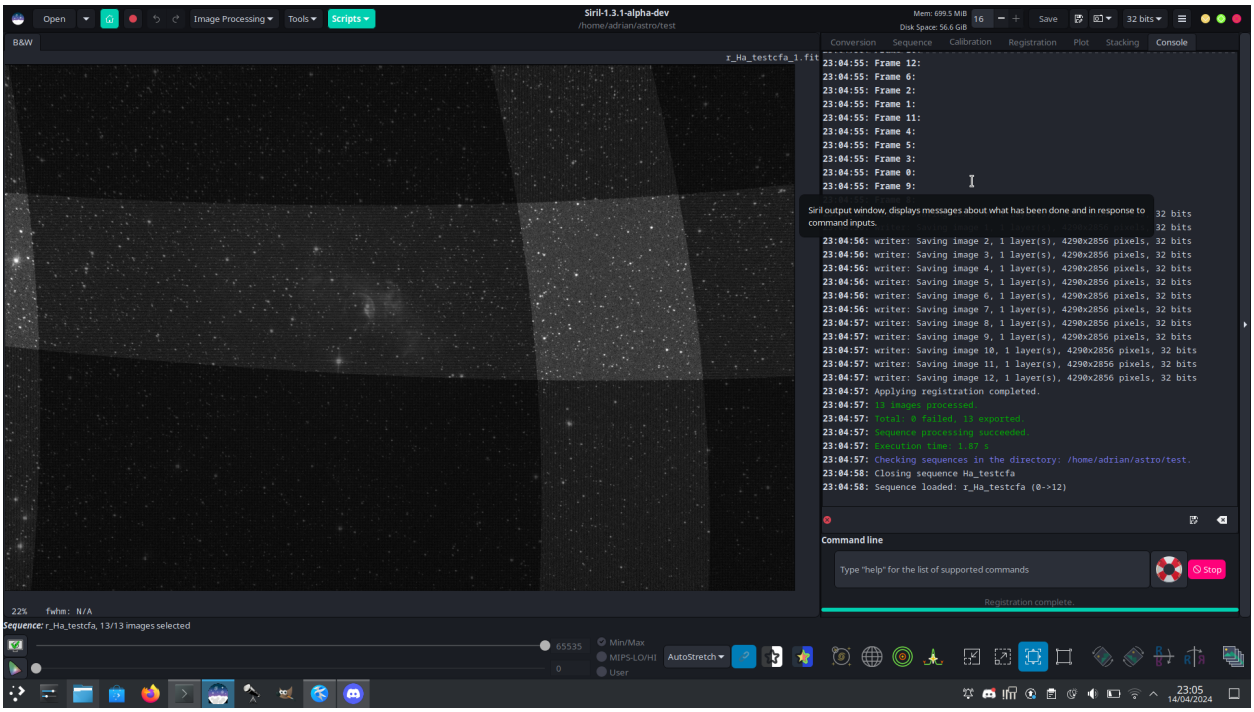



Fig. 38: Showing patterns that result from null pixels in a drizzled image

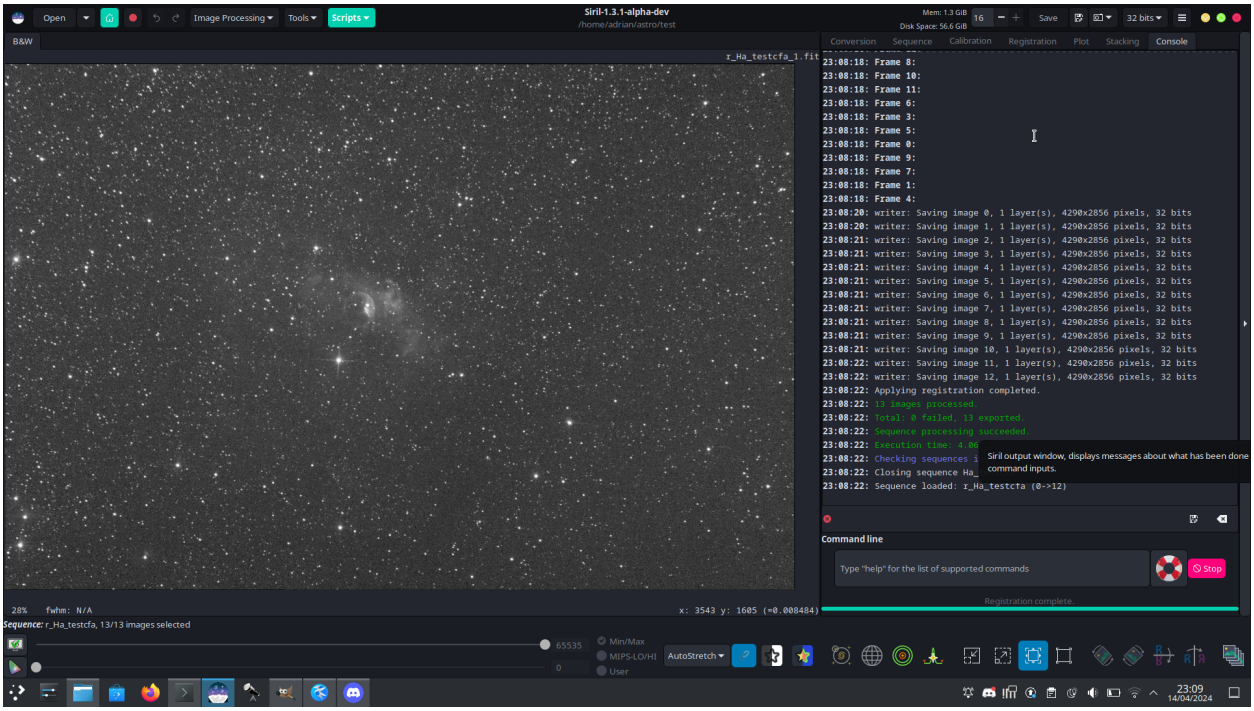



Fig. 39: Using a different drizzle kernel can eliminate patterns of null pixels

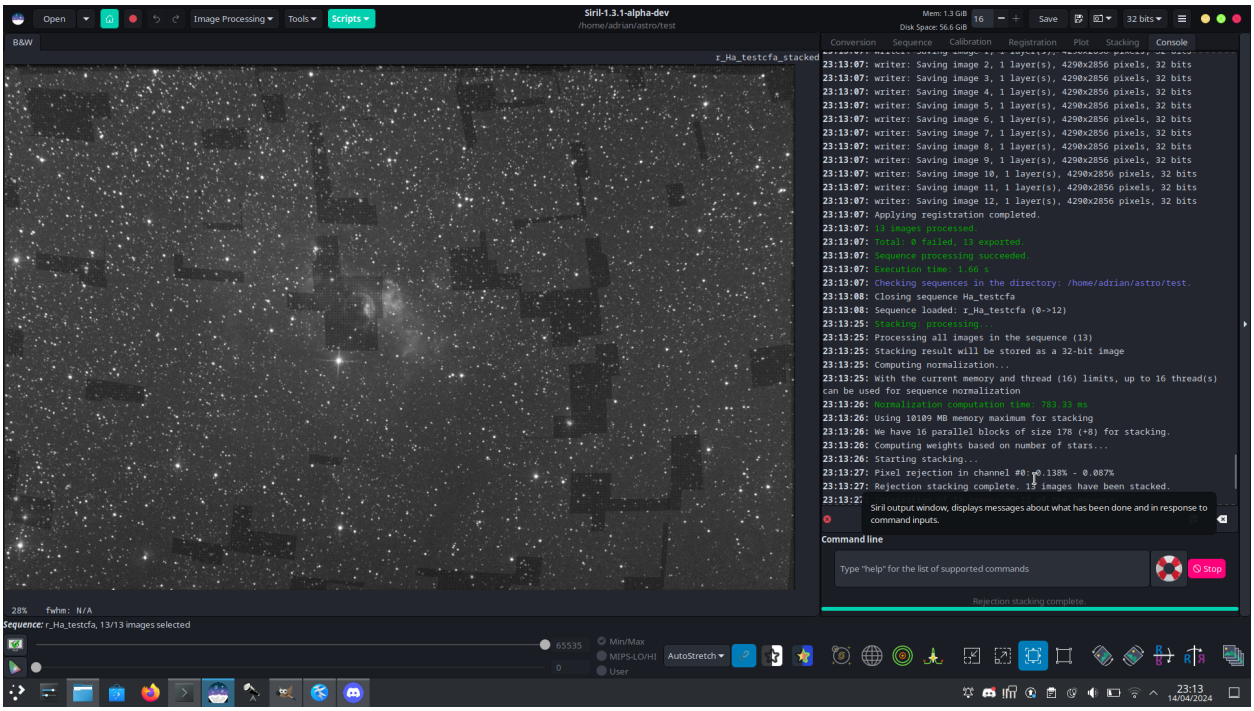



Fig. 40: Typical patchy appearance of a stack of drizzled data with too many null pixels / not enough frames

## Commands

**Siril command line**

```
seqapplyreg sequencename [-prefix=] [-scale=] [-layer=] [-framing=]
seqapplyreg sequencename ... [-interp=] [-noclamp]
seqapplyreg sequencename ... [-drizzle [-pixfrac=] [-kernel=] [-flat=]]
seqapplyreg sequencename ... [-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-
↪filter-round=value[%|k]] [-filter-bkg=value[%|k]] [-filter-nbstars=value[%|k]] [-
↪filter-quality=value[%|k]] [-filter-incl[uded]]
```

Applies geometric transforms on images of the sequence given in argument so that they may be superimposed on the reference image, using registration data previously computed (see REGISTER).

The output sequence name starts with the prefix **"r_"** unless otherwise specified with **-prefix=** option.

The registration is done on the first layer for which data exists for RGB images unless specified by **-layer=** option (0, 1 or 2 for R, G and B respectively).

The output images can be rescaled by passing a **-scale=** argument with a float value between 0.1 and 3.

Automatic framing of the output sequence can be specified using **-framing=** keyword followed by one of the methods in the list { current | min | max | cog } :

**-framing=max** (bounding box) will project each image and compute its shift wrt. reference image. The resulting sequence can then be stacked using option **-maximize** of STACK command which will create the full image encompassing all images of the sequence.

**-framing=min** (common area) crops each image to the area it has in common with all images of the sequence.

**-framing=cog** determines the best framing position as the center of gravity (cog) of all the images.

**Image interpolation options:**

By default, transformations are applied to register the images by using interpolation.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument.

**Image drizzle options:**

Otherwise, the images can be exported using HST drizzle algorithm by passing the argument **-drizzle** which can take the additional options:

**-pixfrac=** sets the pixel fraction (default = 1.0).

The **-kernel=** argument sets the drizzle kernel and must be followed by one of **point**, **turbo**, **square**, **gaussian**, **lanczos2** or **lanczos3**. The default is **square**.

The **-flat=** argument specifies a master flat to weight the drizzled input pixels (default is no flat).

**Filtering out images:**

Images to be registered can be selected based on some filters, like those selected or with best FWHM, with some of the **-filter-*** options.

Links: *register*, *stack*

Note that the introduction of true drizzle has necessitated some changes to existing command arguments for clarity.

*register* and *seqapplyreg* have a new argument **-drizzle** which, together with some related arguments, activates true drizzle.

#### References

## 9.5 Stacking

The final preprocessing step to do with Siril is to stack the images. Image stacking is a technique used in astrophotography to increase the quality and detail of an image by combining multiple photographs into a single, composite image. The process involves taking multiple images of the same object and then align and average the frames together to reduce the noise and increase the signal-to-noise ratio. This results in a final image that has less noise, greater detail and greater dynamic range than a single exposure.

### 9.5.1 Stacking methods

#### Sum stacking

This is the simplest algorithm: each pixel in the stack is summed. The increase in signal-to-noise ratio (SNR) is proportional to $\sqrt{N}$, where $N$ is the number of images. Because of the lack of normalisation and rejection, this method should only be used for planetary processing.

For 8 or 16 bit per channel input images, the sum is done in a 64 bit integer before being normalized to the maximum pixel value and saved as a 16 bit unsigned integer or 32 bit floating point image.

This stacking method should be used for 8-bit input images because it will increase the dynamic of the images while stacking them, making features discernable. Stacking with an mean or median method such a sequence would only decrease the noise but not improve the dynamic of the image, the result would still be 8 bits deep.

### Average Stacking With Rejection

This method of stacking computes a mean of the pixels in a stack after having excluded deviant pixels and an optional normalisation of the images against the reference image. As for sum stacking, the improvement in SNR is proportional to $\sqrt{N}$. There are several ways to normalize the images and several ways to detect and replace or exclude deviant pixels, explained below.

> **Warning**
>
> Some operating systems limit the number of images that can be opened at the same time, which is required for median or mean stacking methods. For Windows, the limit is 2048 images. If you have a lot of images, you should use another type of sequence, described *here*.

Fig. 41: Stacking pixel rejection panel

### Rejection methods

- **Percentile Clipping**: This is a one step rejection algorithm ideal for small sets of data (up to 6 images).
- **Sigma Clipping**: This is an iterative algorithm which will reject pixels whose distance from median will be farthest than two given values in sigma units($\sigma$ low, $\sigma$ high).
- **MAD Clipping**: This is an iterative algorithm working as Sigma Clipping except that the estimator used is the Median Absolute Deviation (MAD). This is generally used for noisy infrared image processing.
- **Median Sigma Clipping**: This is the same algorithm as Sigma Clipping except than the rejected pixels are replaced by the median value of the stack.
- **Winsorized Sigma Clipping**: This is very similar to Sigma Clipping method, except it is supposed to be more robust for outliers detection, see Huber's work [Peter2009].
- **Generalized Extreme Studentized Deviate Test** [Rosner1983]: This is a generalization of Grubbs Test that is used to detect one or more outliers in a univariate data set that follows an approximately normal distribution. This algorithm shows excellent performances with large dataset of more 50 images.
- **Linear Fit Clipping** [ConejeroPI]: It fits the best straight line ($y = ax + b$) of the pixel stack and rejects outliers. This algorithm performs very well with large stacks and images containing sky gradients with differing spatial distributions and orientations.

Note that for drizzled CFA images, you may find difficulty in achieving satisfactory rejection in the Red and Blue channels with some methods. In these cases the MAD Clipping method appears to be the most effective.

### Rejection maps

The option *Create rejection maps* computes and creates rejection maps during stacking. These are images showing how many images were rejected for each pixel of the result image, divided by the number of stacked images. If *Merge L+H* is checked, Siril creates only one rejection map that will be the sum of the low and high maps.

Fig. 42: Example of a rejection map (L+H). We can very clearly see the trace of a satellite that has been removed.

### Images filtering/weighting

The weighting allows to put a statistical weight on each image. In this way, the images considered to be the best will contribute more than those considered to be the worst. Four methods of weighting are available:

- **Number of stars** weights individual frames based on number of stars computed during registration step.
- **Weighted FWHM** weights individual frames based on wFWHM computed during registration step. This is a FWHM weighted by the number of stars in the image. For the same FWHM measurement, an image with more stars will have a better wFWHM than an image with fewer stars.
- **Noise** weights individual frames based on background noise values.
- **Number of images** weights individual frames based on their integration time.

### Image stitching

Available since Siril 1.3, this submenu allows to perform a smoother stitching between images with little overlaps. It is the first step of mosaic stitching which we plan to expand further in next releases.

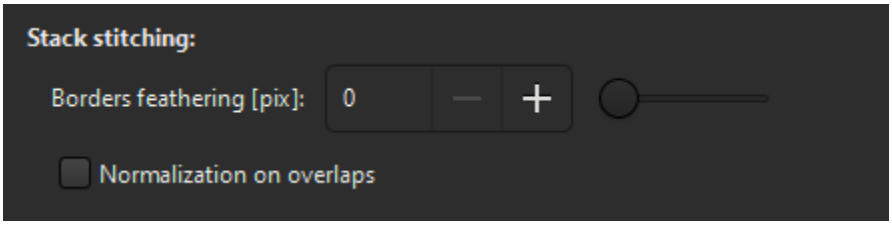



Fig. 43: Stack stitching panel

- *Borders feathering* applies a feathering mask on each image during stacking. The smoothing is applied over X pixels distance, X between the value you set with this interface. The masks are cached in the `./cache` subfolder.

Fig. 44: Borders feathering applied to a 3x2 mosaic (negative). Images courtesy of G. Attard

- *Normalization on overlaps* will compute the normalization factors on the images overlaps instead of whole images. This is useful when images have little overlap and when the nature of the data in each tile is very different accross all images (some images with mostly nebulosity and others with mainly black sky). As it is longer to compute than regular normalization, you should try without ticking this option first. This feature is intended for stitching stacked mosaic tiles so it will throw a warning if you try to compute it for a long sequence (*i.e.* for subs acquired with a smart telescope in mosaic mode). It is only available if *maximize framing* is enabled.

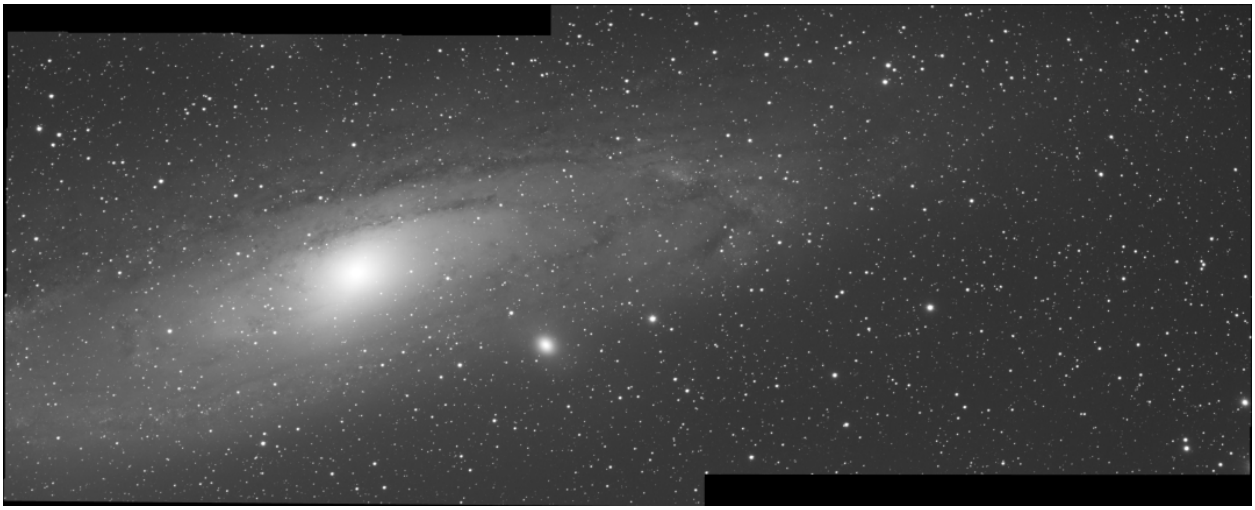


Fig. 45: Global vs overlap normalization on a 2x1 mosaic. Images courtesy of D. Huber

**Theory**

The normalization on overlaps has been derived from the algorithm for gain compensation from [Brown2007]. The full write-up of the modifications is described in the contributors documentation.

### Median stacking

This method is mostly used for dark/flat/bias stacking. The median value of the pixels in the stack is computed for each pixel.

The increase in SNR is proportional to $0.8\sqrt{N}$ and is therefore worse than stacking by average which is generally preferred.

### Pixel Maximum stacking

This algorithm is mainly used to construct long exposure star-trails images. Pixels of the image are replaced by pixels at the same coordinates if intensity is greater.

### Pixel Minimum stacking

This algorithm is mainly used for cropping sequence by removing black borders. Pixels of the image are replaced by pixels at the same coordinates if intensity is lower.

## 9.5.2 Input normalisation methods

Normalisation will adjust the levels of each image against the reference image. This is particularly useful for mean stacking with rejection, because rejecting pixels if the images show differences of levels is not very useful. They can be caused by light nebulosity, light gradient caused by the moon or city lights, sensor temperature variation and so on.

This tends to improve the signal-to-noise ratio and therefore this is the option used by default with the additive normalisation.

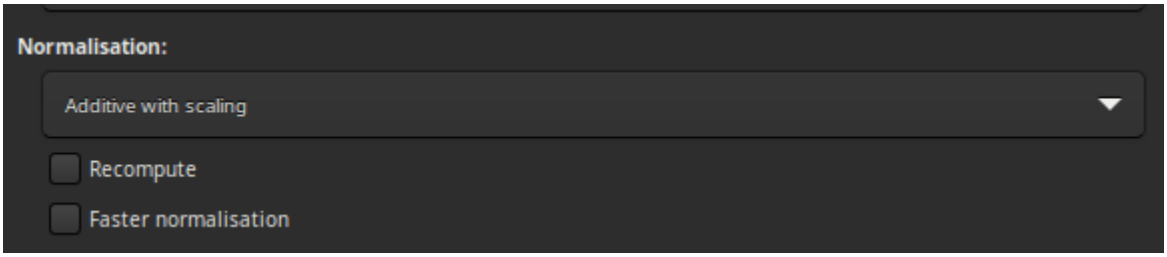



Fig. 46: Stacking input normalization panel

If one of these 5 items is selected, a normalisation process will be applied to all input images before stacking.

- Normalisation matches the mean background of all input images, then, the normalisation is processed by multiplication or addition. Keep in mind that both processes generally lead to similar results but multiplicative normalisation is prefered for image which will be used for multiplication or division as flat-field.
- Scale matches dispersion by weighting all input images. This tends to improve the signal-to-noise ratio and therefore this is the option used by default with the additive normalisation.

| Normalisation | Operation | Use case |
|---|---|---|
| None | No normalisation are applied. | dark/bias frames |
| Additive | Mean background values will be aligned through the application of additive operations. | |
| Multiplicative | Division will be used to align mean background values. | flat frames |
| Additive + Scaling | In combination with additive background through additive matching, the images will be scaled to achieve dispersion matching. | light frames |
| Multiplicative + Scaling | In combination with background matching through division, the images will be scaled to achieve dispersion matching. | |

**Note**

The bias and dark masters should not be processed with normalisation. However, multiplicative normalisation must be used with flat-field frames.

Keep in mind that both processes generally lead to similar results but multiplicative normalisation is preferred for image which will be used for multiplication or division as flat field.

Since the normalisation calculation step is usually a long one, as it requires determining all the statistics of the image, the results are stored in the `seq` file. This way, if the user wants to do another stacking by changing the rejection parameters, it will be executed more quickly. The *Recompute* option allows to force the recalculation of the normalisation.

By default, Siril uses IKSS estimators of location and scale to compute normalisation. For long sequences, computing these estimators can be quite intensive. For such cases, you can opt in for faster estimators (based on median and median absolute deviation) with the option *Faster normalisation*. While less resistant to outliers in each image, they can still give a satisfactory result when compared to no normalisation at all.

### 9.5.3 Image rejection

It is also possible to reject a certain number of images in order to select only the best ones. This can be very useful for Lucky DSO techniques where the number of images in a sequence is very high. One can choose between % and k-$\sigma$ to either retain a given percentage of images or to calculate the allowable threshold using k-$\sigma$ clipping.

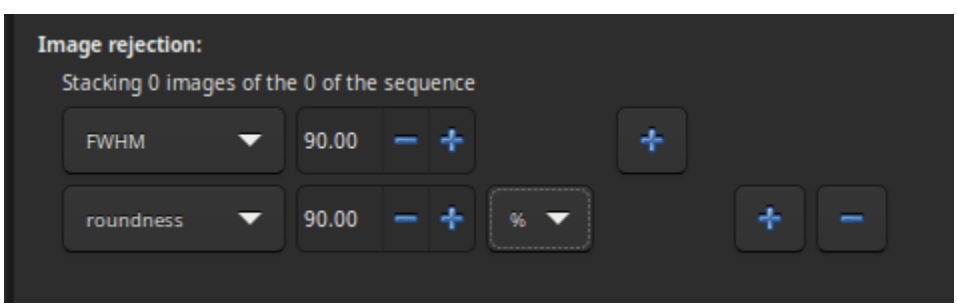



Fig. 47: Stacking image rejection panel

Several critieria are available:

- **all**: all images of the sequence are used in the stack.
- **selected**: only use image that have not been unselected from the sequence.

- **FWHM**: images with best computed FWHM (star-based registration only).
- **weighted FWHM**: this is an improvement of a simple FWHM. It allows to exclude much more spurious images by using the number of stars detected compared to the reference image (star-based registration only).
- **roundness**: images with best star roundness (star-based registration only).
- **background**: images with lowest background values (star-based registration only).
- **nb stars**: images with best number of stars detected (star-based registration only).
- **quality**: images with best quality (planetary DFT or Kombat registrations).

### 9.5.4 Stacking result

- If *Output Normalisation* is checked, the final image will be normalized in the [0, 1] range if you work in 32-bit format precision, or in [0, 65535] otherwise.

> **Warning**
>
> This option should not be checked for master stacking.

- If *RGB equalization* is checked, the channels in the final image will be equalized (color images only).
- If *Maximize framing* is checked, the output image will encompass all the images. Note this option is forced if the images have different sizes.
- If *Interpolation upscaling x2* is ticked, the images will be upscaled during stacking. Note this option is not always available.
- Selecting *Force 32b* forces the stacked image to be saved as a float image irrespective of the bitdepth set in the Preferences.
- The stacking result is saved under the name given in the text field. It is possible to use *path parsing* to build the filename. A click on the *overwrite* button allows the new file created to overwrite the old one if it exists. If the latter is not checked but an image with the same name already exists, then no new file is created.

### 9.5.5 References

CHAPTER

TEN

# PROCESSING

This section takes you through the different processing steps of your images. The drop-down menu is accessible from the header bar using the *Image Processing* button. The tools are grouped in the menus and sub-menus, and in this documentation too, by theme.

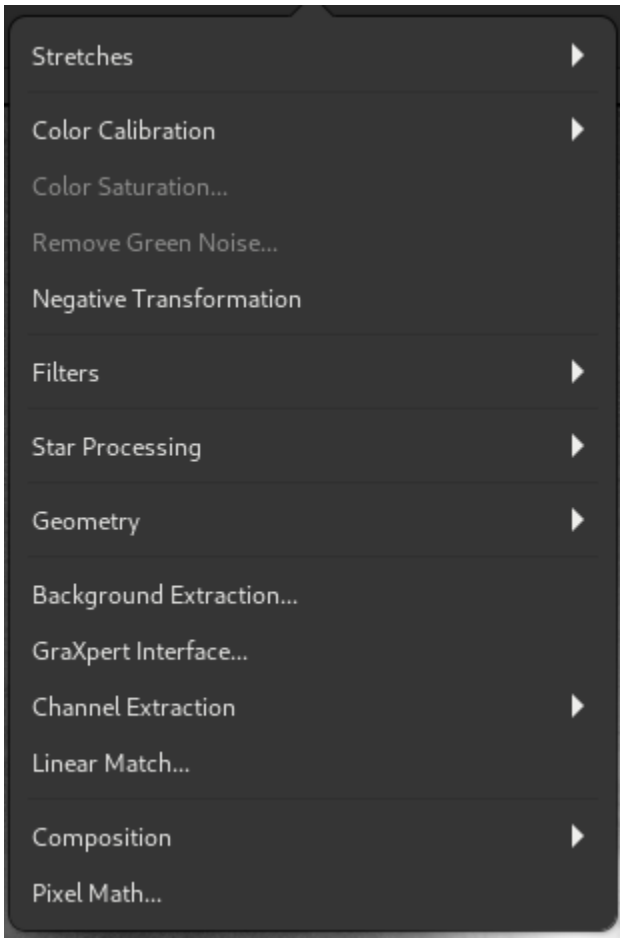



Fig. 1: Image processing menu

## 10.1 Image stretching

Image are stored as pixel values that come from the camera following a quasi-linear law, meaning that for areas of the sky that show no visible feature, the pixel value will be close to zero, but for bright objects like stars it will be close to a maximum value depending on exposure and gain. In between, if a nebula has a surface magnitude half of a star, it will have pixel values half of those of the star and so on. This is what we call the linear pixel mode.

The human eye doesn't quite see photons like that. It amplifies dark areas, so that an object maybe a tenth as bright as another would look half as bright. For astronomy images, we usually display images with a similar pixel value scaling (see display modes from the GUI).

But it is only a display trick, using a screen transfer function, to render the pixel values of the untouched image to better looking images.

Image stretching is about doing something similar but by modifying the pixel values of images instead of just altering their rendering. Siril has three main tools to achieve this.

### 10.1.1 Asinh transformation

The asinh, or inverse hyperbolic sine, transformation will modify image pixel values in a way similar to what can be seen with the asinh display pixel scaling function, which is parametrized by the low and high values cut-off cursors. Here the parameters are the stretch factor and the black point value.

For monochrome images, pixel values are modified using the following function:

$$\text{pixel} = \frac{(\text{original} - \text{blackpoint}) \times \text{asinh}(\text{original} \times \text{stretch})}{\text{original} \times \text{asinh}(\text{stretch})}$$

For color images, the function becomes:

$$\text{pixel} = \frac{(\text{original} - \text{blackpoint}) \times \text{asinh}(\text{rgb_original} \times \text{stretch})}{\text{rgb_original} \times \text{asinh}(\text{stretch})}$$

where rgb_original is computed using the pixel values of the three channels.

A clipping mode can also be set.

**Theory**

As rgb_original is an average of the 3 channels, one or two channel values will be greater than rgb_original and can therefore clip. This can cause color artefacts when bright, strongly-colored regions are stretched. In order to avoid this problem the RGB blend clipping algorithm was developed by the authors of the original GHSastro tool: the same algorithms are available in the Siril implementation and this is the default clipping mode for stretches that require handling of clipping. The $(r, g, b)$ values are stretched first based on the luminance value rgb_original to give $(r', g', b')$. Then the original $(r, g, b)$ values are independently stretched to give $(r'', g'', b'')$. Finally the largest value of $k$ is identified such that

$k \times r' + (1 - k) \times r''1;$

$k \times g' + (1 - k) \times g''1;$

and

$k \times b' + (1 - k) \times b''1$

Then the transformed values are calculated as

$(k \times r' + (1 - k) \times r'', k \times g' + (1 - k) \times g'', k \times b' + (1 - k) \times b'')$

This RGB blend clipping algorithm is also available for the Generalised Hyperbolic Stretch transforms described below.

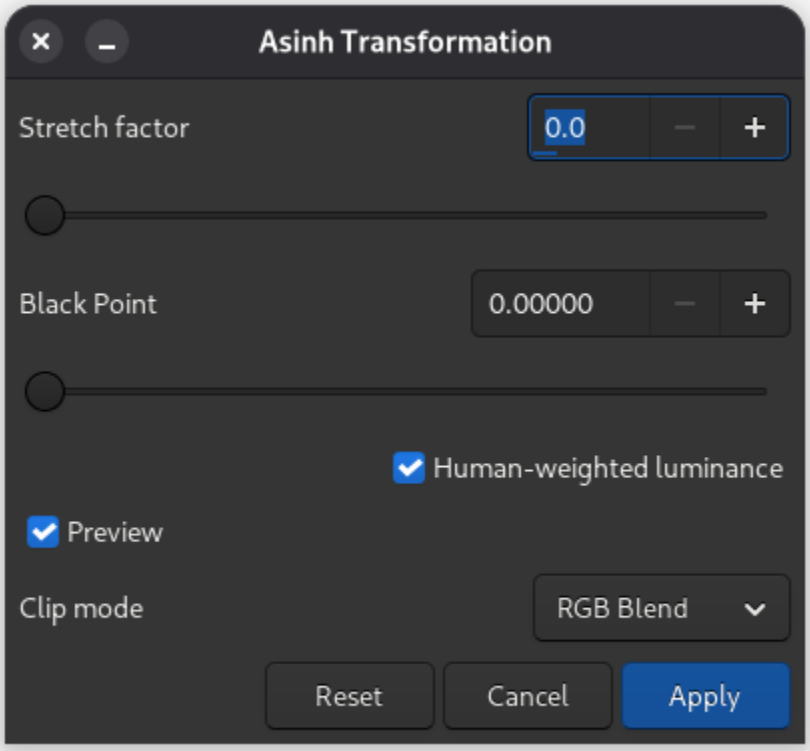



Fig. 2: Dialog box of Asinh Transformation

Other choices of clipping algorithm are available:

- *Clip* - this clipping mode just allows any colour components that clip to clip, but restricts them to values in the range 0.0 to 1.0. This may suffer from coloured artefacts such as fringes around nearly-saturated stars, but it is extremely quick to calculate.
- *Rescale* - this clipping mode checks the R, G and B components of each pixel and if any are > 1.0 it rescales the pixel so that no components are clipped. This method is prone to artefacts and is mainly included for feature equivalence with the GHSastro plugin. It is quick to compute.
- *Global Rescale* - this clipping mode behaves similarly to *Rescale* except that the scaling is computed globally instead of per-pixel. This avoids the kind of artefacts that Rescale can produce, but has a bigger impact on overall image brightness. This is faster than *RGB blending* to compute but slower than *Clip* or *Rescale*.

When the Use Human-weighted Luminance option is not ticked, rgb_original is the mean of the three pixel values; when it is set, weighting changes to 0.2126 for the red value, 0.7152 for the green value and 0.0722 for the blue value, which gets results closer to human perceptual color balance.

**Siril command line**

```
asinh [-human] stretch { [offset] [-clipmode=] }
```

Stretches the image to show faint objects using an hyperbolic arcsin transformation. The mandatory argument **stretch**, typically between 1 and 1000, will give the strength of the stretch. The black point can be offset by providing an optional **offset** argument in the normalized pixel value of [0, 1]. Finally the option **-human** enables using human eye luminous efficiency weights to compute the luminance used to compute the stretch value for each pixel, instead of the simple mean of the channels pixel values. This stretch method preserves lightness from the L*a*b* color space. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

### 10.1.2 Midtone Transfer Function Transformation (MTF)

MTF is one of the most powerful tools for stretching the image. It can be easily automated and that's why the auto-stretched view uses it.

The tool is presented in the form of a histogram with 3 sliders (in the form of a triangle placed underneath) that we must move to transform the image. The triangle on the left represents the shadow signal, the one on the right the highlights and finally, the one in the middle the midtone balance parameter. The values of these sliders are displayed below the histogram, on the left, and can be changed directly by hand. Opposite is the percentage of pixels that are clipped by the transformation: it is important not to clip too many pixels. If only the midtones parameter is changed, then no pixel can be clipped.

**Theory**

The new pixel values are then computed with this function:

$$\mathrm{MTF}(x_p) = \frac{(m-1)x_p}{(2m-1)x_p - m}. \tag{10.1}$$

- For $x_p = 0$, MTF $= 0$,
- for $x_p = m$, MTF $= 0.5$,

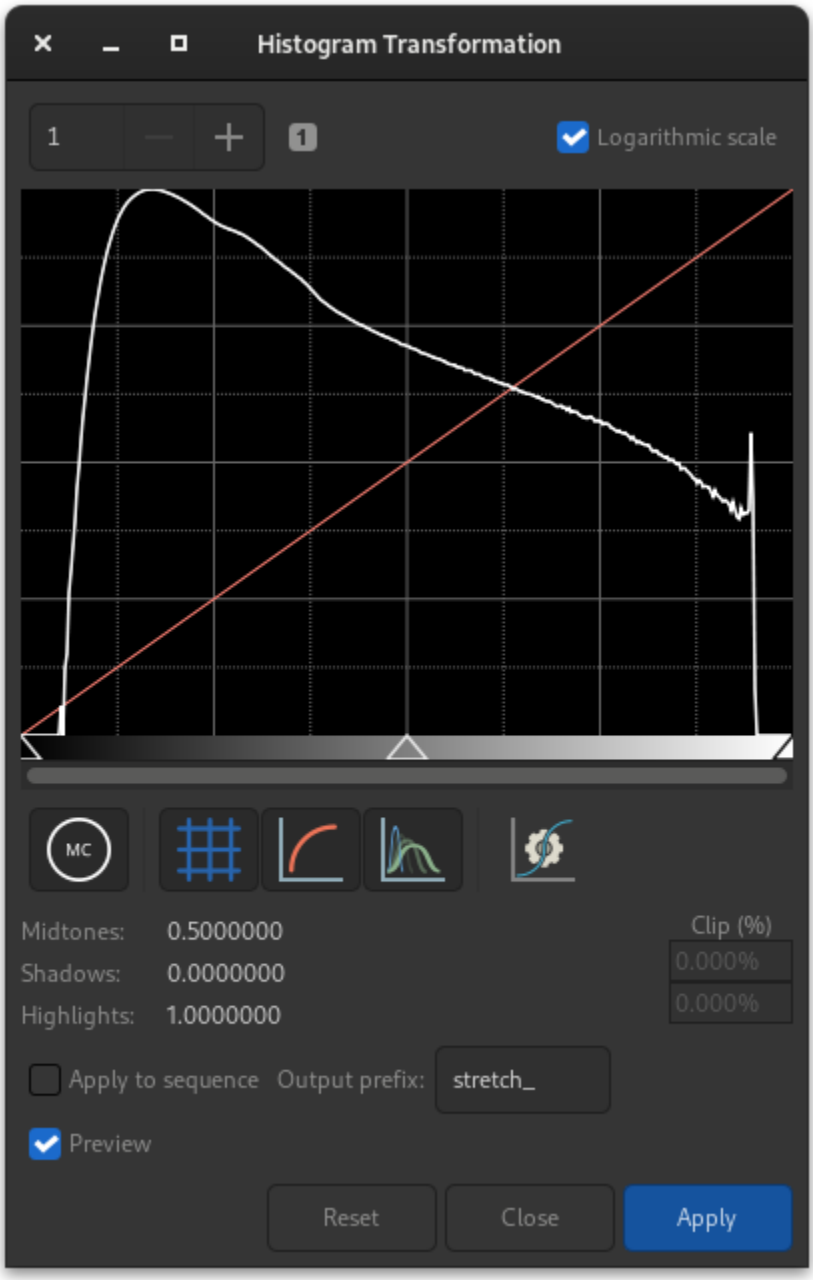



Fig. 3: Dialog box of the Histogram Transformation

- for $x_p = 1$, MTF $= 1$,

where $x_p$ is the pixel value defined as follow

$$x_p = \frac{\text{original} - \text{shadows}}{\text{highlights} - \text{shadows}}. \tag{10.2}$$

**Note**

It is generally not recommended to change the value of the highlights, otherwise they will become saturated and information will be lost.

The toolbar contains many buttons that affect the visualization of the histogram. You can choose to display the input histogram, the output histogram, the transfer curve and the grid. The button allows you to apply the same transformation as the autostretch algorithm. It is rarely advisable to use this button as is. Adjustments are usually necessary to avoid losing information.

**Tip**

When the button is pressed, the sliders and lo, mid and hi entries will become temporarily inactive. You have to apply the autostretch with the *Apply* button and the controls will reactivate. You can then apply adjustments as a second follow-up stretch. This behaviour avoids problems where the monitor ICC profile is different to the image ICC profile.

At the top of the histogram it is also possible to choose to display the histogram in logarithmic view, as in the illustration. This behavior can be made default as explained *here*. Finally a zoom in X is available. This is very useful when all the signal is concentrated on the left of the histogram.

**Tip**

If a ROI is set, the MTF histogram preview will not update to show the impact of the stretch on the ROI. This is because that behaviour could be misleading: if the ROI is not typical of the image overall, adjusting the ROI histogram to a suitable level would result in a badly adjusted histogram for the overall image and potentially a burned-out or excessively dark look to the result. When in ROI mode the stretch parameters should be adjusted by eye. If it is desired to check the histogram for the stretch as applied to the image as a whole, the ROI should be cleared.

**Siril command line**

```
mtf low mid high [channels]
```

Applies midtones transfer function to the current loaded image.

Three parameters are needed, **low**, **midtones** and **high** where midtones balance parameter defines a nonlinear

histogram stretch in the [0,1] range. For an automatic determination of the parameters, see AUTOSTRETCH. Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels

Links: *autostretch*

**Note**

`mtf` is also a function that can be used in the *PixelMath* tool.

**Siril command line**

```
autostretch [-linked] [shadowsclip [targetbg]]
```

Auto-stretches the currently loaded image, with different parameters for each channel (unlinked) unless **-linked** is passed. Arguments are optional, **shadowclip** is the shadows clipping point, measured in sigma units from the main histogram peak (default is -2.8), **targetbg** is the target background value, giving a final brightness to the image, range [0, 1], default is 0.25. The default values are those used in the Auto-stretch rendering from the GUI.

Do not use the unlinked version after color calibration, it will alter the white balance

#### Applying transformation to the sequence

This transformation can easily be applied to a sequence. You just have to define the transformation on the loaded image (with a sequence already loaded), then check the *Apply to sequence* button and define the output prefix of the new sequence (`stretch_` by default), or use the following command:

**Siril command line**

```
seqmtf sequencename low mid high [channels] [-prefix=]
```

Same command as MTF but for the sequence **sequencename**.

The output sequence name starts with the prefix "mtf_" unless otherwise specified with **-prefix=** option

Links: *mtf*

### 10.1.3 Generalised Hyperbolic Stretch transformations (GHS)

This is the most capable and modern tool of Siril, also the most complex to learn. A very detailed tutorial for this tool in Siril was written by the authors of this algorithm: https://siril.org/tutorials/ghs. Here, we will just summarize here the basic operation of this tool.

Simply put, the GHS is able to improve the contrast of a range of brightness levels in an image. For example, if one wanted to better view the details in the medium to high brightness part of a nebula (which is in general very faint in an astronomy image), it would be possible to only select this range for stretching. It is very good at improving the contrast

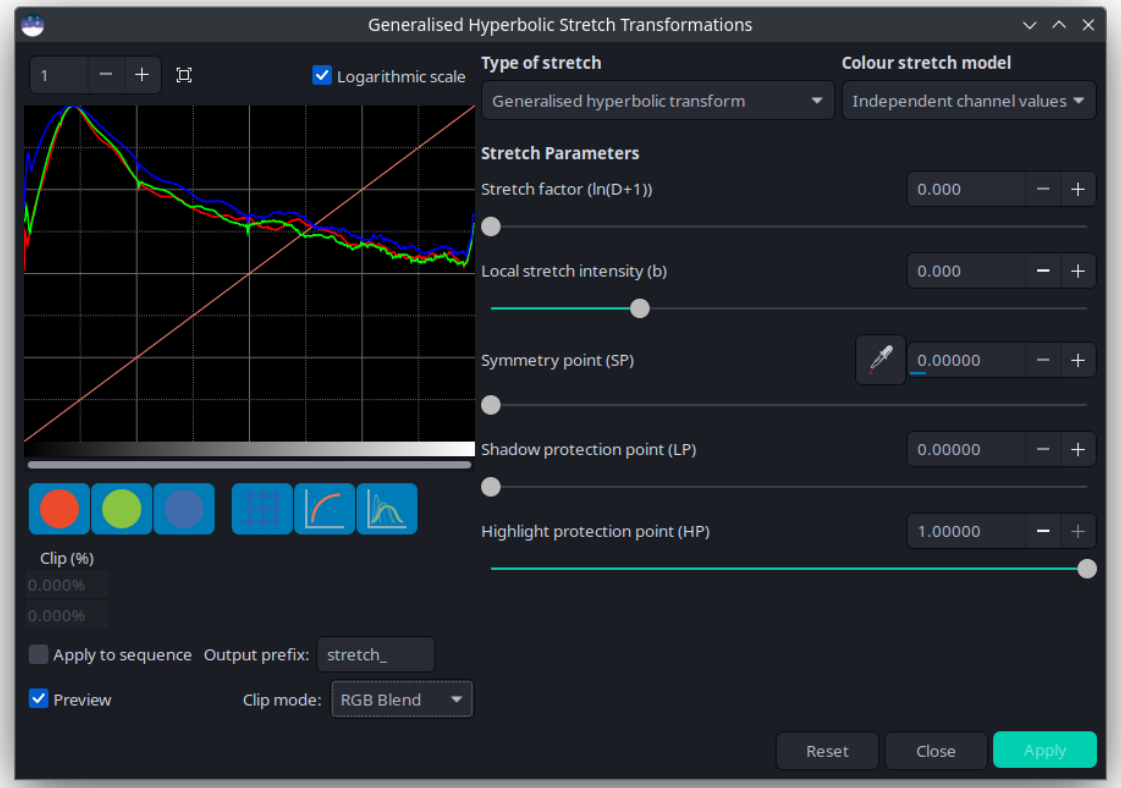



Fig. 4: Dialog box of the Generalized Hyperbolic Stretch

of main objects without making stars too big. The tool is very much based on iterative use, so stretching all the different ranges of brightnses in the image one after the other, by small touches.

To achieve this, the tool relies heavily on histogram display and interaction, for each color channel. The transformation function, shaped like a hyperbole or an 'S', can be altered by moving its center (the **SP** - symmetry point parameter), by flattening either of its ends (with **shadow** and **highlight protections**), and of course its twist (stretch **D** and local stretch **b** factors). Manipulating these parameters on a small (for speed) image with an **SP** value of 0.5 will help you understanding their effect.

There are two main operations to do on each iteration: selecting the range of lights to modify, and actually modifying it. Selecting the range is quite easy, it's a matter of finding a representative value (**SP**) and defining the width of the range (**b**). Setting **SP** can be done in three ways:

- selecting an area of similar brightness in the image and clicking on the picker button
- clicking on the histogram itself with a single left click (it is possible to zoom in the histogram using the + button at the top left)
- using the cursor or its associated plus and minus buttons or direct value.

The width of the range depends on the local stretch. A high value of **b** will make a small range, and increase contrast over a small range of brightnesses in the image.

Modifying the histogram once the location of the change has been set is a more complex operation. One goal given by the algorithm's authors is to make the logarithmic view of the histogram (enabled by checking the box) as close as possible to a decreasing line. To do this, bumps need to be carved out and valleys to be filled. Here is a quick guide of values to use depending on what needs to be achieved:

- **initial stretch from linear:** set **SP** slightly to the left of the main peak, moderate **b** value from 6 and up, increase **D** slightly only to start to see the main object. Do not stretch too much at this point like an autostretch would do, otherwise the stars would grow too big (main tutorial section for this).
- **improving contrast of a range, or filling a valley:** set **SP** to the centre of the valley in the histogram, set **b** as high as how narrow the range or valley is, decrease **HP** to preserve stars, increase **D** slowly until the improvement appears.
- **decreasing contrast of a range, or flattening a peak:** decreasing a peak is not easy to do but will happen as a side effect of valleys being filled. For example, creating a peak, or filling a valley, will decrease what is on the left of **SP**. Another possibility is to use the inverse transformation, from the *Type of stretch* combo box, and a high **LP** value, and **HP** at 1.
- **move curve to the left, making the image darker:** often if we stretched the entire histogram, the peak will move to the right, making the background too bright. There is a simple way to just move everything to the left, select in the *Type of stretch* combo box the last entry, **Linear stretch (BP shift)**. There's only one cursor to move now, controlling how much it will shift.

**Tip**

If a ROI is set, the GHT histogram preview will not update to show the impact of the stretch on the ROI. This is because that behaviour could be misleading: if the ROI is not typical of the image overall, adjusting the ROI histogram to a suitable level would result in a badly adjusted histogram for the overall image and potentially a burned-out or excessively dark look to the result. When in ROI mode the stretch parameters should be adjusted by eye. If it is desired to check the histogram for the stretch as applied to the image as a whole, the ROI should be cleared.

Some operations are also common for **color images**, where we often want to have a similar shape of curve for the three channels, working on each channel independently by unselecting them with the three colored circles below the histogram view:

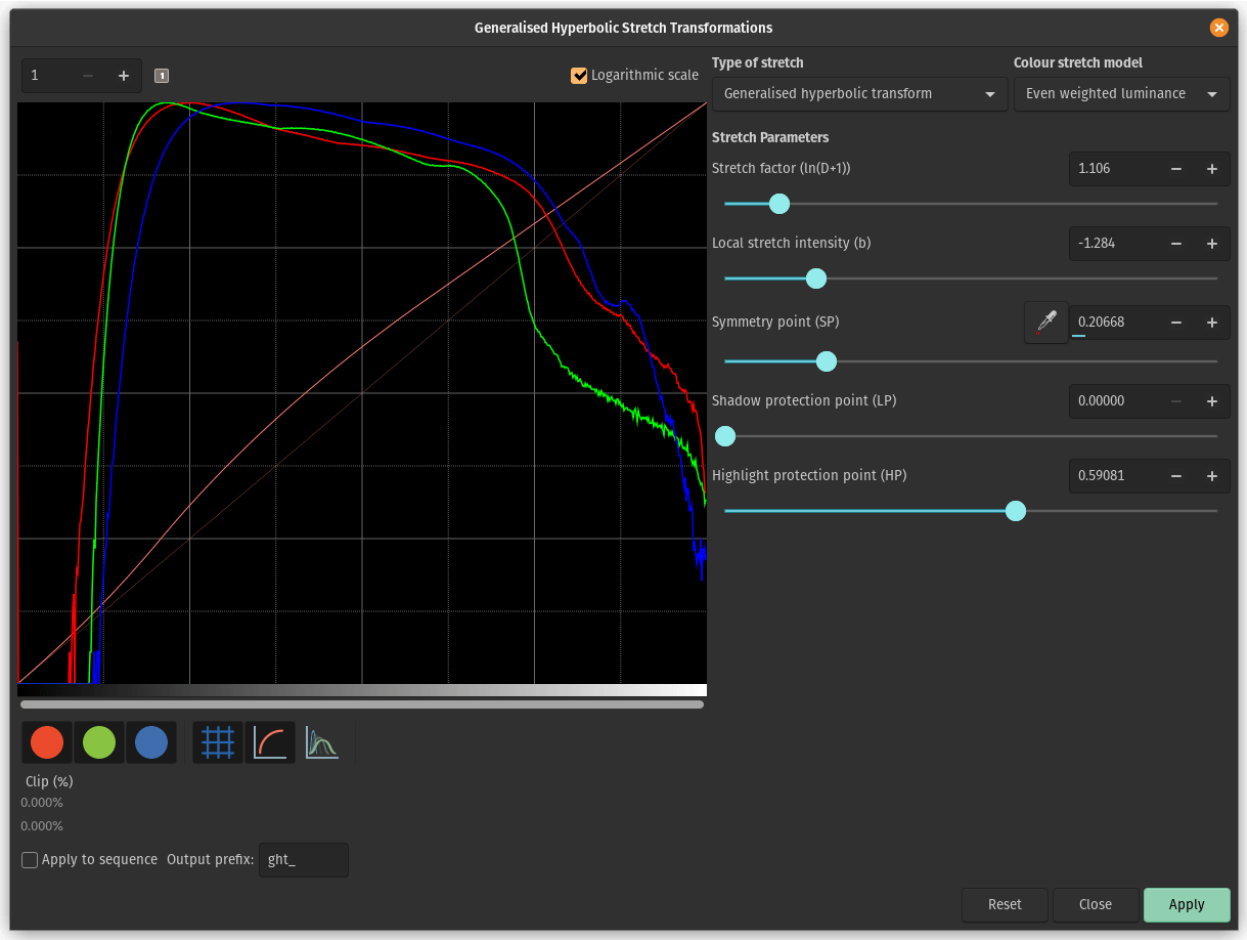



Fig. 5: The Generalized Hyperbolic Stretch with a color image

- **moving the peak to the right:** a simple strech with a **SP** value left of the peak will do that in general, so this should be done as part of a stretch.
- **spreading a peak:** to stretch a channel a bit more and it give it more importance in the final result, without changing the location of the peak too much, set **SP** near the peak or slightly to its right, set **b** depending on how the contribution is expected throughout the channel, between a negative value if the impact shall be felt up to the stars levels (to change their color) and a high value if this is only for a nebula, increase **D** to obtain the target width of the peak, and then offset the peak to the left by decreasing **HP**.
- **moving all channels together:** an alternative luminance mapping stretch exists, see the *Color stretch model* combo box at the top right of the GHS window, using either luminance stretch values will stretch the luminance and reapply colors on it instead of stretching directly the three channels. The luminance modes can be better at preserving colours in the image. These modes use the same RGB blend clipping mode described above to prevent color channel clipping artefacts.
- **remapping image saturation:** the GHS transforms can be applied to the image saturation channel by selecting the Saturation option from the *Color stretch model* combo box. When this mode is selected the pre- and post-stretch saturation histograms will be shown in yellow. All the GHS options are available and this mode can provide highly targeted adjustment of the image saturation channel. A simple method of increasing the saturation in relatively unsaturated regions while preventing oversaturation is to use an **Inverse generalised hyperbolic transform** stretch with **SP** set to around 0.5, and **HP** brought down low enough to flatten the upper end of the saturation histogram.

In order to control partially-clipped highlights, the GHS tool makes available the same range of clipping modes as the asinh stretch. Details can be found *here*.

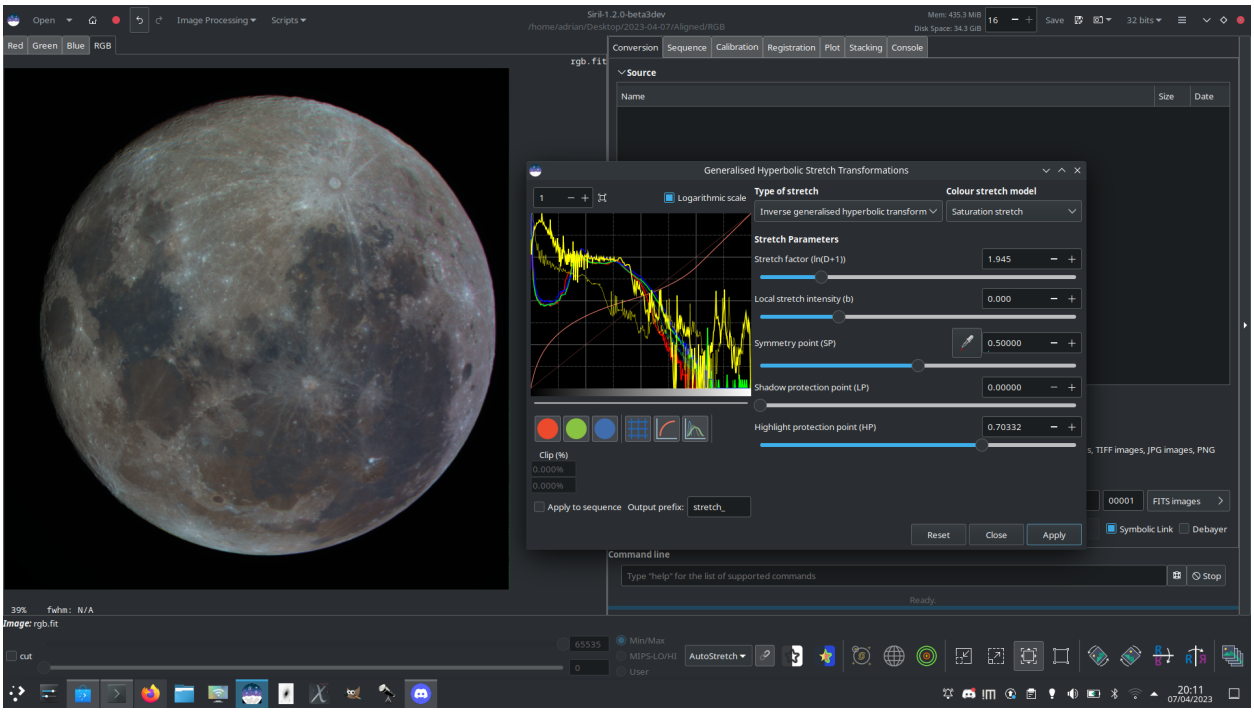



Fig. 6: The image above shows how applying the GHS tool to the saturation channel gives an easy way of strongly enhancing saturation in a low-saturation image while still retaining control of the upper end of the saturation histogram, here used to create a 'Mineral Moon' image highlighting the differing mineral composition of different regions of the lunar surface.

**Siril command line**

```
ght -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -
↪sat] [channels]
```

Generalised hyperbolic stretch based on the work of the ghsastro.co.uk team.

The argument **-D=** defines the strength of the stretch, between 0 and 10. This is the only mandatory argument. The following optional arguments further tailor the stretch:

**B** defines the intensity of the stretch near the focal point, between -5 and 15;

**LP** defines a shadow preserving range between 0 and SP where the stretch will be linear, preserving shadow detail;

**SP** defines the symmetry point of the stretch, between 0 and 1, which is the point at which the stretch will be most intense;

**HP** defines a region between HP and 1 where the stretch is linear, preserving highlight details and preventing star bloat.

If omitted B, LP and SP default to 0.0 ad HP defaults to 1.0.

An optional argument (either **-human**, **-even** or **-independent**) can be passed to select either human-weighted or even-weighted luminance or independent colour channels for colour stretches. The argument is ignored for mono images. Alternatively, the argument **-sat** specifies that the stretch is performed on image saturation - the image must be color and all channels must be selected for this to work.

Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

**Siril command line**

```
invght -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -
↪sat] [channels]
```

Inverts a generalised hyperbolic stretch. It provides the inverse transformation of GHT, if provided with the same parameters, undoes a GHT command, possibly returning to a linear image. It can also work the same way as GHT but for images in negative

Links: *ght*

**Siril command line**

```
modasinh -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -sat]␣
↪[channels]
```

Modified arcsinh stretch based on the work of the ghsastro.co.uk team.

The argument **-D=** defines the strength of the stretch, between 0 and 10. This is the only mandatory argument. The following optional arguments further tailor the stretch:

**LP** defines a shadow preserving range between 0 and SP where the stretch will be linear, preserving shadow detail;

**SP** defines the symmetry point of the stretch, between 0 and 1, which is the point at which the stretch will be most intense;

**HP** defines a region between HP and 1 where the stretch is linear, preserving highlight details and preventing star bloat.

If omitted LP and SP default to 0.0 ad HP defaults to 1.0.

An optional argument (either **-human**, **-even** or **-independent**) can be passed to select either human-weighted or even-weighted luminance or independent colour channels for colour stretches. The argument is ignored for mono images. Alternatively, the argument **-sat** specifies that the stretch is performed on image saturation - the image must be color and all channels must be selected for this to work.

Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

**Siril command line**

```
invmodasinh -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -
↪sat] [channels]
```

Inverts a modified arcsinh stretch. It provides the inverse transformation of MODASINH, if provided with the same parameters, undoes a MODASINH command, possibly returning to a linear image. It can also work the same way as MODASINH but for images in negative

Links: *modasinh*

**Siril command line**

```
linstretch -BP= [-sat] [-clipmode=] [channels] [-clipmode=]
```

Stretches the image linearly to a new black point BP.

The argument **[channels]** may optionally be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels.

Optionally the parameter **-sat** may be used to apply the linear stretch to the image saturation channel. This argument only works if all channels are selected. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

### Applying transformation to the sequence

This transformation can easily be applied to a sequence. You just have to define the transformation on the loaded image (with a sequence already loaded), then check the *Apply to sequence* button and define the output prefix of the new sequence (`stretch_` by default). All of the commands have a sequence processing form too. Each sequence stretching command starts with *seq* and the first argument must be the sequence name, but they are otherwise the same.

**Siril command line**

```
seqght sequence -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as GHT but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *ght*

**Siril command line**

```
seqinvght sequence -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as INVGHT but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *invght*

**Siril command line**

```
seqmodasinh sequence -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as MODASINH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *modasinh*

**Siril command line**

```
seqinvmodasinh sequence -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as INVMODASINH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *invmodasinh*

**Siril command line**

```
seqlinstretch sequence -BP= [channels] [-sat] [-prefix=]
```

Same command as LINSTRETCH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *linstretch*

### 10.1.4 Curves Transformation

Curves Transformation is a highly versatile tool used to adjust the contrast and brightness of an image by modifying the pixel values according to a custom-defined curve. This allows for precise control over the image's stretch.

The curve is defined by a series of points, each of which can be moved to adjust the curve. The curve is interpolated between these points, and the pixel values are transformed based on it. This allows for a wide range of transformations to be applied to the image, from simple linear stretches to complex non-linear adjustments.

There are two interpolation algorithms available in the Curves Transformation dialog: linear and cubic spline.

**Theory**

- Linear interpolation is a simple interpolation that connects the points with straight lines. For each pair of points, the slope of the line connecting the points is calculated:

$$\text{slope} = \frac{y_2 - y_1}{x_2 - x_1}. \tag{10.3}$$

  The pixel values are then transformed by evaluating the line at the original pixel value:

$$\text{pixel} = \text{slope} \times (\text{original} - x_1) + y_1. \tag{10.4}$$

- Cubic spline curves are more complex curves that are defined by a series of control points. For pixel value $x$ between two control points $x_i$ and $x_{i+1}$, the curve is defined by the following equation:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{10.5}$$

For $x_i < x < x_{i+1}$, the coefficients $a_i, b_i, c_i$, and $d_i$ are calculated by solving a system of equations derived from the conditions of continuity and smoothness at each internal point. These conditions are:

- The spline must be continuous at each internal point,
- The first derivative of the spline must be continuous at each internal point,
- The second derivative of the spline must be continuous at each internal point
- Since Curves Transformation uses natural cubic splines, the second derivative at both endpoints is 0.

Fig. 7: Dialog box of the Curves Transformation

**Warning**

Curves Transformation is only available as a GUI tool and cannot be used from the command line.

Curves Transformationcan display the histogram of the image in two modes: linear and logarithmic. You can swap between these modes by clicking the *Logarithmic scale* checkbox at the top of the histogram. Logarithmic scale is useful for images with a wide dynamic range, as it allows you to see the histogram in the shadows and highlights more clearly.

**Tip**

If a ROI is set, the histogram preview will not update to show the impact of the stretch on the ROI. This is because that behaviour could be misleading: if the ROI is not typical of the image overall, adjusting the ROI histogram to a suitable level would result in a badly adjusted histogram for the overall image and potentially a burned-out or excessively dark look to the result. When in ROI mode the stretch parameters should be adjusted by eye. If it is desired to check the histogram for the stretch as applied to the image as a whole, the ROI should be cleared.

Some common uses of the Curves Transformation tool include:

- **"S" curve:** This curve is used to increase the contrast of an image. By increasing the slope of the curve in the middle of the histogram, the contrast is increased.

Fig. 8: An example of an "S" curve

- **BP & WP adjustments:** The black point (BP) and white point (WP) of an image can be adjusted by moving the first and last control points of the curve. This allows for the shadows and highlights of the image to be adjusted independently.

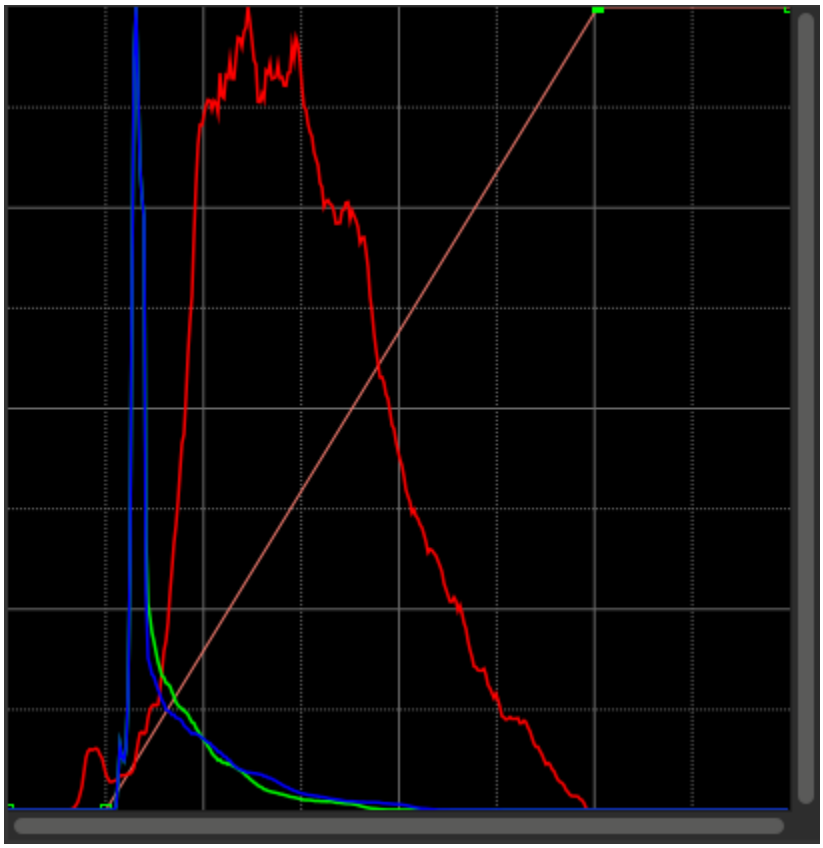


Fig. 9: An example of adjusting the black point and white point of an image

- **Targeted adjustments:** By adding control points at specific locations in the histogram, targeted adjustments can be made to the image. For example, the shadows can be darkened without affecting other parts of the image.

Fig. 10: An example of a targeted adjustment to an image

> **Tip**
>
> The Curves Transformation tool is best used on an image that has already been stretched to some extent. This allows for more precise control over the image's contrast and brightness.

This transformation can easily be applied to a sequence. All you have to do is define the transformation on the loaded image while a sequence is loaded and check the *Apply to sequence* button. Similarly to other stretching tools, you can define the output prefix of the new sequence (`curve_` by default).

## 10.2 Color Calibration

Siril offers three ways to retrieve the colors of your image:

- *Manual calibration*
- *Photometric Color Calibration*, also called PCC.
- *Spectrophotometric Color Calibration*, also called SPCC. This is the most accurate and recommended method

Here, "retrieve" means re-balancing the RGB channels to get as close as possible to the true colors of the shot object.

> **Tip**
>
> PCC and SPCC are two methods that require a *plate-solved image*. It is not possible to access these tools without this prerequisite.

### 10.2.1 Manual Color Calibration

> **Warning**
>
> The color calibration **must** be performed on a linear image whose histogram has not yet been stretched. Otherwise, the colors obtained are not guaranteed to be correct.

The manual way uses the following window:

The first step deals with the **background** of your image. The goal is to equalize the RGB layers in order the background appears as a neutral grey color.

After making a selection in your image (in a not so crowdy nor contrasted area), the area is taken into account by clicking on the *Use current selection* button. The coordinates of the rectangle are displayed. Then *Background Neutralization* will calculate the median of each channel and equalize them.

The second step deals with the **bright objetcs** of the picture. You can modify once again the histogram in two ways:

- Manually, with **White reference** and the 3 R, G and B coefficients, according to your own taste.
- Automatically, by selecting a rectangle area with contrasted objects (the same way as previously)

Two sliders allow you to change the rejection limit for too dark and too bright pixels in the selection.

As this is a trial and error process, you can undo the result with the *Undo* button (up left) and then try with other selections or coefficients until you are satisfied.

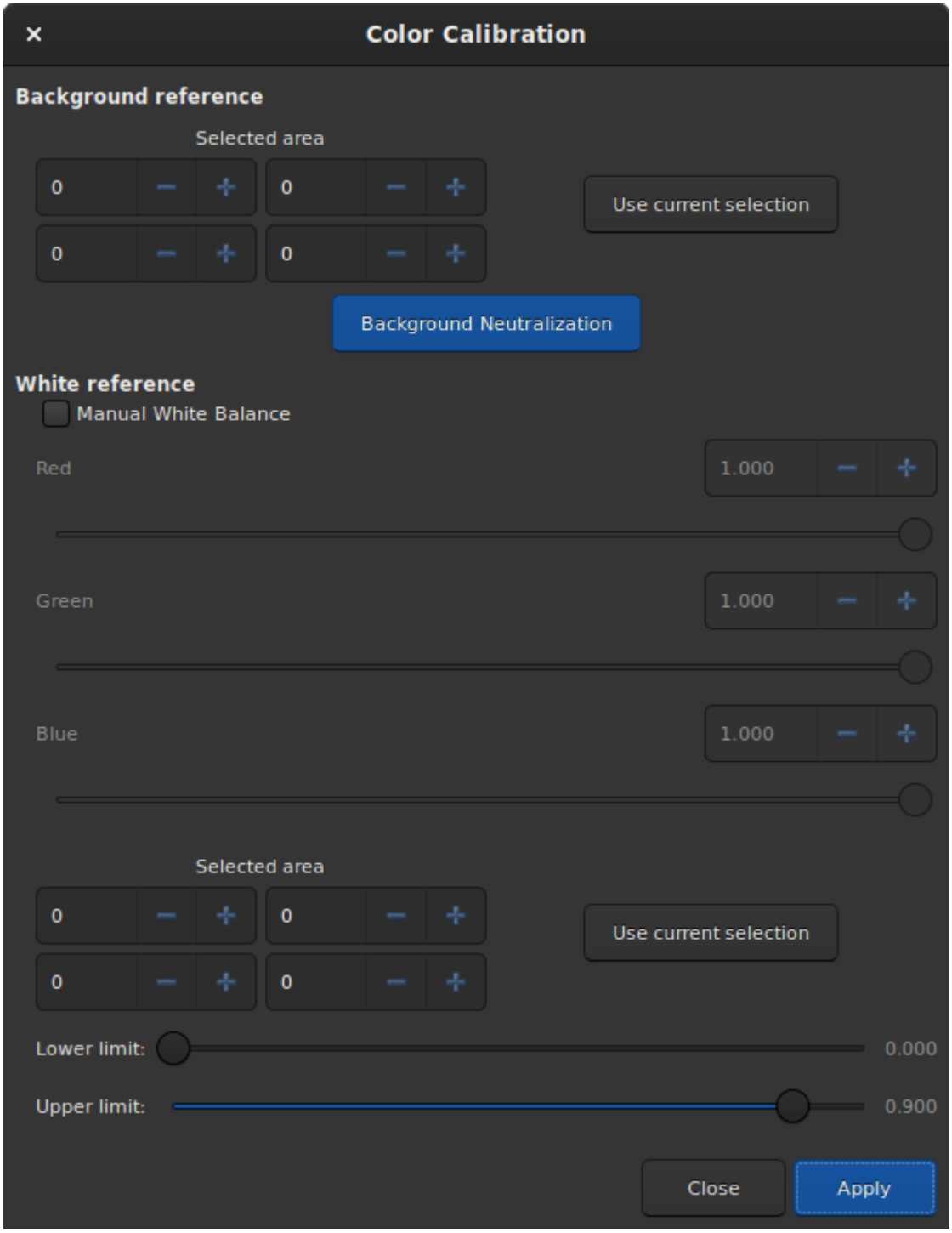



Fig. 11: Manual Color Calibration dialog window.

### 10.2.2 Photometric Color Calibration

> **Warning**
>
> The calibration of the colors by photometry **must** imperatively be carried out on a linear image whose histogram was not yet stretched. Otherwise, the photometric measurements will be wrong and the colors obtained without guarantee of being correct.

Another way for retrieving the colors is to compare the color of the stars in the image with their color in catalogues, to obtain the most natural color in an automatic and non-subjective way. This is the PCC (Photometric Color Calibration) tool (`Ctrl` + `Shift` + `P`). It can only work for images taken with a set of red, green and blue filters for colors, or on-sensor color. To identify stars within the image with those of the catalogue, an astrometric solution is required. Please see the *documentation of the plate solver module*.

This method is less accurate than Spectrophotometric Color Calibration explained below, however it can be performed using local catalogues and is therefore the best option when an internet connection is not available.

> **Note**
>
> This technique is heavily dependent on the type of filter used. Using different kinds of R, G, B filters will not make a large difference, but using a Light pollution filter or no IR-block filters will make the solution deviate significantly and not give the expected colors.

Since version 1.4, the two tools run independently: it is possible to run the photometric analysis and color correction of the image only if the image has been already plate solved. It also means different catalogues can be used for PCC and astrometry. The tool is also available as the `pcc` command, so it can be embedded in image post-processing scripts.

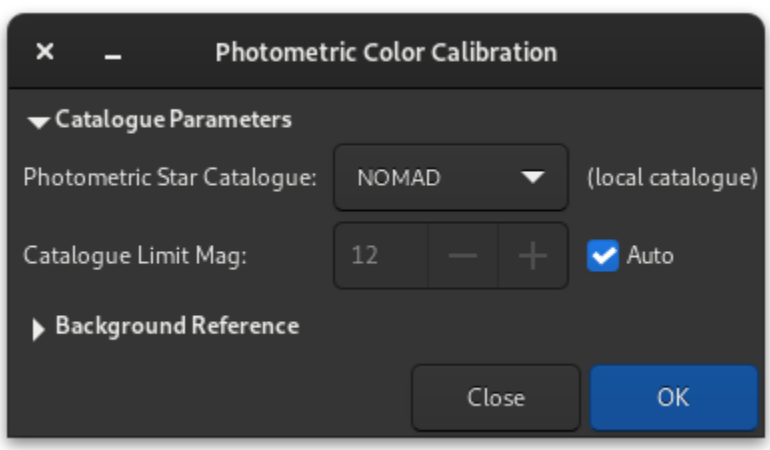



Fig. 12: Photometric Color Calibration dialog window.

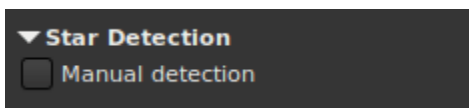
If the image was previously plate solved, turn on the *annotations* feature to check that catalogues align with the image. If the astrometric solution is not good enough, please redo a plate solving.

- The **Catalog Settings** section allows you to choose which photometric catalog should be used, NOMAD, APASS or Gaia DR3, as well as the limiting magnitude.

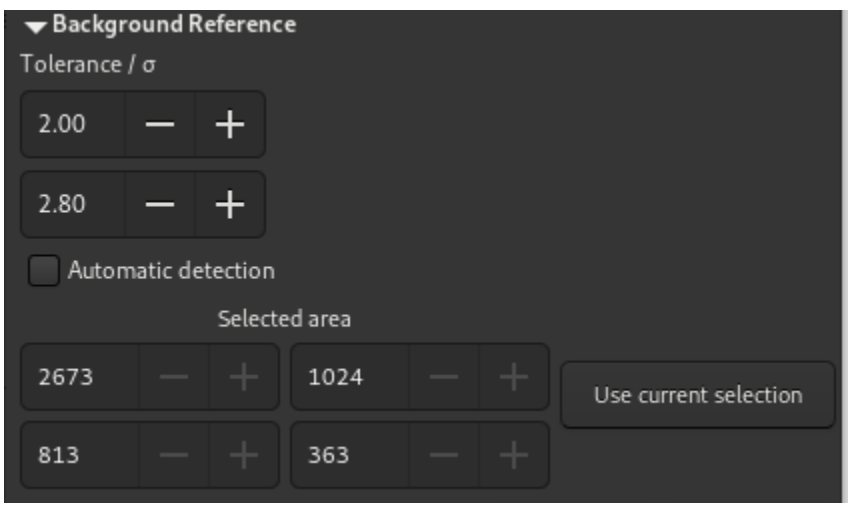
> **Tip**
>
> The NOMAD and GAIA catalogs can be *locally installed*, while the APASS catalog requires internet access to obtain its contents.
>
> The local Gaia catalog used for PCC is the astrometric extract, which is a much smaller download than the whole SPCC catalog. If this is available it will be shown as the default.

> **Note**
>
> Whereas the NOMAD and APASS catalogs use the catalogued B-V index and estimate the effective temperature from this value, Gaia provides an accurately modelled effective temperature therefore when this catalog is used the Teff field is used directly, giving more accurate results.

- The **Star Detection** section allows you to manually select which stars will be used for the photometry analysis. It's better to have hundreds of them at least, so individual picking would not be ideal.




- If desired, the **Background Reference** can be manually selected as described in *Manual Color Calibration*. This can be useful in the case of nebula images where the background sky parts are small.




When enough stars are found and the astrometric solution is correct, the PCC will print this kind of text in the Console tab:

```
Applying aperture photometry to 433 stars.
70 stars excluded from the calculation
Distribution of errors: 1146 no error, 18 not in area, 48 inner radius too small, 4␣
↪pixel out of range
Found a solution for color calibration using 363 stars. Factors:
K0: 0.843       (deviation: 0.140)
K1: 1.000       (deviation: 0.050)
K2: 0.743       (deviation: 0.130)
```

(continues on next page)

(continued from previous page)

```
The photometric color correction seems to have found an imprecise solution, consider␣
↪correcting the image gradient first
```

We can understand that 433 stars were selected from the catalogue and the image for photometric analysis, but somehow, only 363 we actually used, 70 being excluded. The line *Distribution of errors* explains for what reason they were excluded: 18 were not found in the expected position, 48 were too big and 4 probably saturated. It is very common to have many stars rejected because they don't meet the strict requirements for a valid photometric analysis.

We can also see that the PCC found three coefficients to apply to the color channels to correct the white balance. The *deviation* here, which is the average absolute deviation of the color correction for each of the star of the photometric set, is moderately high. On well calibrated images without gradient, with correct filters and without a color nebula covering the whole image, devation would get closer to 0.

**Siril command line**

```
pcc [-limitmag=[+-]] [-catalog=] [-bgtol=lower,upper]
```

Run the Photometric Color Correction on the loaded plate-solved image.

The limit magnitude of stars is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.

The star catalog used is NOMAD by default, it can be changed by providing **-catalog=apass**, **-catalog=localgaia** or **-catalog=gaia**. If installed locally, the remote NOMAD (the complete version) can be forced by providing **-catalog=nomad**

Background reference outlier tolerance can be specified in sigma units using **-bgtol=lower,upper**: these default to -2.8 and +2.0

### 10.2.3 Spectrophotometric Color Calibration

**Warning**

The calibration of the colors by photometry **must** imperatively be carried out on a linear image whose histogram was not yet stretched. Otherwise, the photometric measurements will be wrong and the colors obtained without guarantee of being correct.

Spectrophotometric Color Calibration (`Ctrl` + `Shift` + `C`) is the newest method of color calibration available in Siril. This method uses the extensive spectral data available in the Gaia DR3 catalogue [GaiaDR3]. This can be accessed either through direct querying of the online catalogue or by downloading a local extract and querying the local catalogue.

**Warning**

Note that when the Gaia archive is offline for maintenance or due to a fault, Siril's SPCC functionality will not be available via the remote catalog. Fortunately the archive is normally very reliable, however a status indicator is built into the SPCC dialog. The archive status is checked when the dialog starts up and can be re-checked by clicking the status button.

The offline Gaia SPCC extract will still work fine if the Gaia archive is offline.

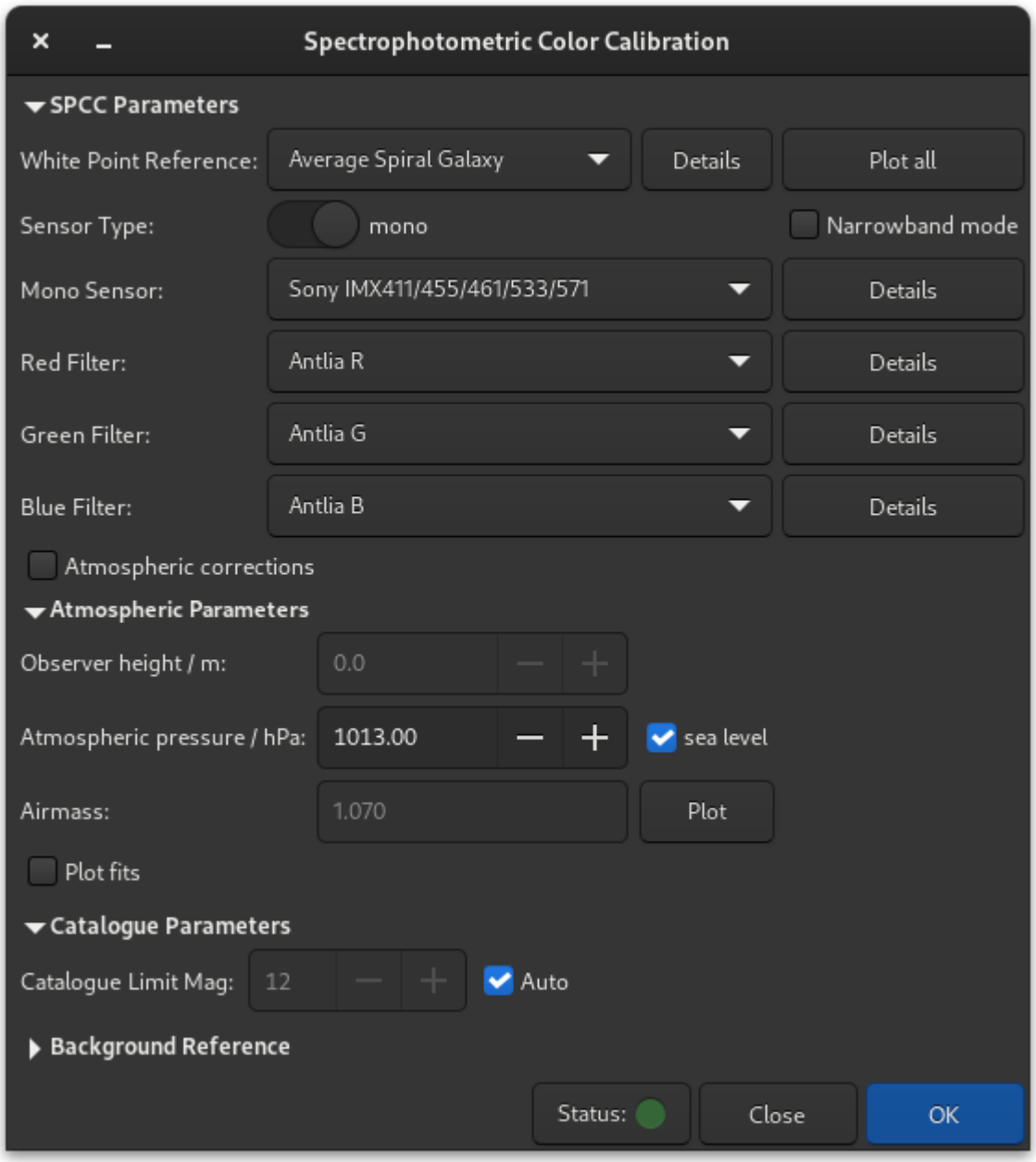



Fig. 13: SpectroPhotometric Color Calibration dialog window.

**Tip**

**What's the difference between SPCC and PCC? When should I use one rather than the other?** SPCC is a more accurate version of PCC and renders the latter obsolete. SPCC takes your setups sensor and filters into account. As a result, the color produced is closer to "reality". The example in the picture below illustrates the difference in results.

Fig. 14: Comparison between PCC (left) and SPCC (right): click to enlarge. (Courtesy of Ian Cass)

### Local SPCC Catalog

From 1.4.0 an offline SPCC catalog is available using Gaia DR3 data. Note that the catalog is chunked into 48 files covering each level 1 HEALpixel.

**Theory**

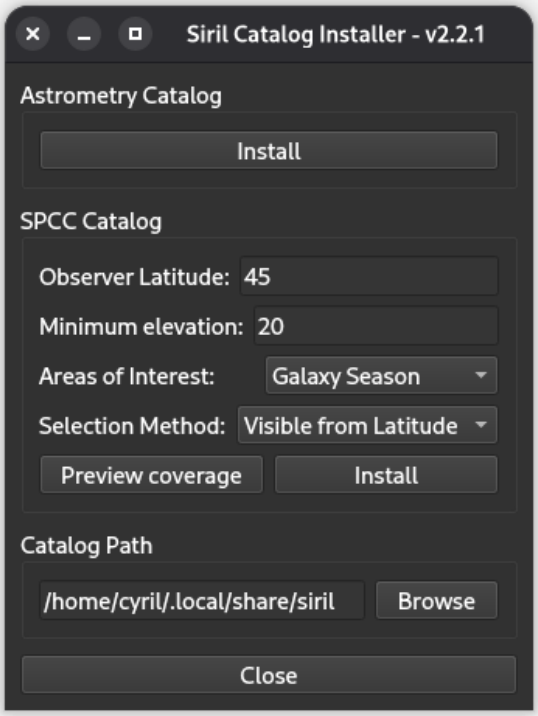
HEALpix (Hierarchical Equal Area isoLatitude Pixelisation) is an algorithm for pixelising a sphere based on subdivision of a distorted rhombic dodecahedron. Mathematical details can be found on Wikipedia [Wiki_HEALPIX]. Gaia sources use a Level 12 NESTED HEALpix scheme and the HEALpixel number is encoded into the source_id. The specification of the Gaia DR3 catalogue extracts and their file format is documented here (PDF).

The nested nature of the scheme means that HEALpixels that are close together in the sky have numbers that are close together. The hierarchical property also means that it is possible to index sources in HEALpixels at a deep HEALpixel level and divide the catalog into chunks at a shallower level while still supporting a highly efficient catalog search algorithm.

It is possible to download the entire catalog or only the chunks you need. The folder location to store the catalog files is set in *Preferences->Astrometry*.

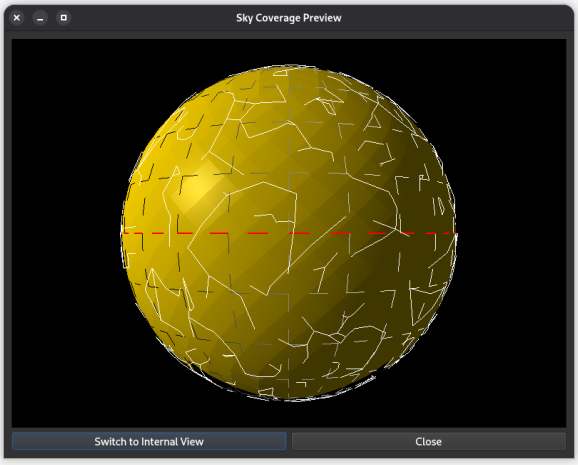
**Sirilpy script**

The easiest way to install the catalog is to use the built-in Python script `Siril_Catalog_Installer.py` in the *Scripts->Python Scripts* menu.




This provides an interface that allows you to install either the whole catalog, or only the chunks that are visible from your observing latitude above a certain elevation, or only sets of chunks corresponding to certain themes (*Milky Way*, *Summer Triangle*, *Galaxy Season* etc.) Select the latitiude / elevation or area of interest if desired, and then select the selection method (*All*, *Visible from Latitude* or *Area of Interest*).

You can preview the coverage using the *Preview coverage* button.




Finally, clicking *Install* will download, verify, uncompress and install the selected chunks and also set the catalog path in Siril's preferences. A default catalog path is suggested in the text entry widget, but can be changed to a different location if you prefer.

If you wish to install the offline SPCC catalog files manually, they can be downloaded from Zenodo. Either individual level 1 HEALpixels can be downloaded or the entire catalog can be downloaded as an archive.

> **Tip**
>
> When you download "All Files" from the Zenodo record, the download is a zip archive that you will need to extract, however the zip archive is just a convenient way of bundling all of the individual files; the data files inside the zip archive are themselves compressed with bzip2 compression, and you will need to decompress the individual .bz2 files before Siril can use them. Support for this compression format is available by default in Linux and MacOS, and is provided in Windows by various archive programs including 7-Zip and Pea-Zip, which are both Free and Open Source software.

All compressed files have accompanying sha256sums and there is a file containing all the sha256sums of the uncompressed files as well, for additional validation. The Zenodo record also provides a DOI reference that can be used to cite the dataset if you use it in academic work.

Siril uses an optimized extract of the Gaia DR3 xp_sampled datalink product. As with the astrometric extract, the offline catalogue is capped at the 127 brightest sources per level 8 HEALpixel. The catalogue contains fewer sources than the astrometric extract as xp_sampled spectra are typically only provided for sources brighter than magnitude 17.6 and therefore more HEALpixels in emptier parts of the sky have fewer than 127 sources compared with the astrometric extract (i.e. these HEALpixels contain all the available Gaia DR3 sources with xp_sampled data), but this approach still avoids overpopulation of the catalogue in extremely crowded parts of the sky while providing the best SNR. In those HEALpixels with fewer than 127 xp_sampled sources, the local catalog is as comprehensive as using the online Gaia archive directly.

The xp_sampled is converted from float32 to float16 data with an additional byte setting the exponent to be applied to the xp_sampled data for the source to overcome limitations on exponents expressible with float16. This is entirely justifiable given the error bars on the xp_sampled data and makes no practical difference to the accuracy of the results. It means that we can provide a highly effective, purpose-optimized local SPCC catalogue in under 21GB of data.

### How it Works

SPCC requires knowledge of your sensor and the RGB filters you use. These are provided through an *online repository* which Siril will sync, either automatically at startup or manually when required. Sensor and filter information is updated via the same synchronization method as used for the online scripts repository. (This means that as data on new filters or sensors becomes available it can be added to the repository without requiring an update to the application.)

In the GUI you select your sensors and filters from the widgets in the SPCC dialog. Don't worry if there isn't an exact match for your equipment, just pick the closest option, or the appropriate default option. You also need to select a white reference. The default reference is the Average Spiral Galaxy reference which is suitable for a wide range of astrophotographic scenes, however there is an extensive range of galaxy and star types to choose from. The Sun's spectral type is G2(v) so if you want to balance your image using sunlight as a white reference, you would pick **Star, type G2(v)** from the list.

SPCC then uses the stellar spectra in Gaia DR3 and knowledge of your imaging sensor and filters to compute for each star in the catalogue that matches a star detected in the image by Siril the expected flux in each color channel. It then compares this with the actual flux measured in each channel using Siril's photometric capabilities.

Given the sensor and filter knowledge, SPCC computes the expected flux in each channel for the specified white reference. A robust linear fit is obtained to give the best fit of catalogue to image R/G and B/G flux ratios for each star and for the white reference. This fit is used to derive correction coefficients which are applied multiplicatively to each channel, resulting in spectrophotometrically accurate color channels.

Your image must be plate solved for SPCC to work: if it is not already, this should be done with the dedicated tool. It is important to make sure that the plate solving information is correct, as some software is known to add inaccurate WCS data to images.

### Graphical Interface

- **Selection of Sensor** In order to select your sensor, ensure that the mono / OSC toggle button is set correctly. You will then see the appropriate dropdown to choose from the available sensors.
- **Selection of Filters** SPCC can operate in two modes.
  - The default mode is broadband operation. In this mode, the *Narrowband mode* check box should be unchecked. You can choose either red, green and blue filters (for composited images made with a mono sensor) or OSC filters, for example light pollution filters, for images made with an OSC sensor.

    > **Warning**
    >
    > If you select a DSLR OSC sensor (e.g. a Canon EOS 600D) an additional widget will become visible to select a DSLR Low Pass Filter. This allows you to tailor whether your camera has been astro-modded or not. You **must** select an option here or the process will complain that you haven't set all the necessary filters!
    >
    > Options exist for Canon and Nikon OEM low-pass filters as well as the popular Baader BCF astro-mod filter that lets Ha and Sii through but stil blocks longer IR wavelengths and "Full spectrum" which is modelled as a perfect clear filter.
    >
    > If you have an unmodified camera of a different model or brand, select any of the Canon or Nikon low-pass filters: the effect is very minor as these wavelengths are right at the edge of human visual perception anyway.

  - By checking the *Narrowband mode* check box, you enable narrowband mode. This is intended either for images composited from narrowband filters used with a mono sensor or for images made using an OSC sensor with a dual, tri-band or quad band narrowband filter. In this mode the available controls change, and for each color channel you enter the nominal wavelength and bandwidth of the filter passband. For ultra-narrowband mono filters the passband may be as little as 3nm; for a quadband OSC filter like the Altair QuadBand V2 the passbands may be as much as 35nm. Note that for a HOO composition where two channels are set to the same data, the nominal wavelength and bandwidth should be set equal in the SPCC interface too.

    > **Tip**
    >
    > Some manufacturers specify a center wavelength and FWHM. It is fine to use the FWHM as the bandwidth: these filters have very sharp cutoffs.

    > **Warning**
    >
    > Don't expect to retrieve the Hubble palette for SHO imaging using the wavelengths of the SII, H$\alpha$ and OIII filters respectively. The result will be an image with a huge green cast. This is easily explained by the fact that the SII emission line is much fainter than that of hydrogen, and the SPCC gives a representation of real intensities. But this is not the case in the Hubble palette. In fact, *manual color calibration* will give better results.

- **Selection of DSLR Low Pass Filter (LPF)** DSLRs contain a low-pass filter (sometimes also called a 'hot mirror'. These reduce transmittance at wavelengths of interest to astronomers (Ha at 656nm and S-II at 674nm). If the selected OSC is a DSLR, a dropdown will be provided from which you can the appropriate LPF profile. Options exist for stock LPFs as well as astro-modified LPFs and an ideal **Full spectrum** filter model for if the LPF has been removed altogether.

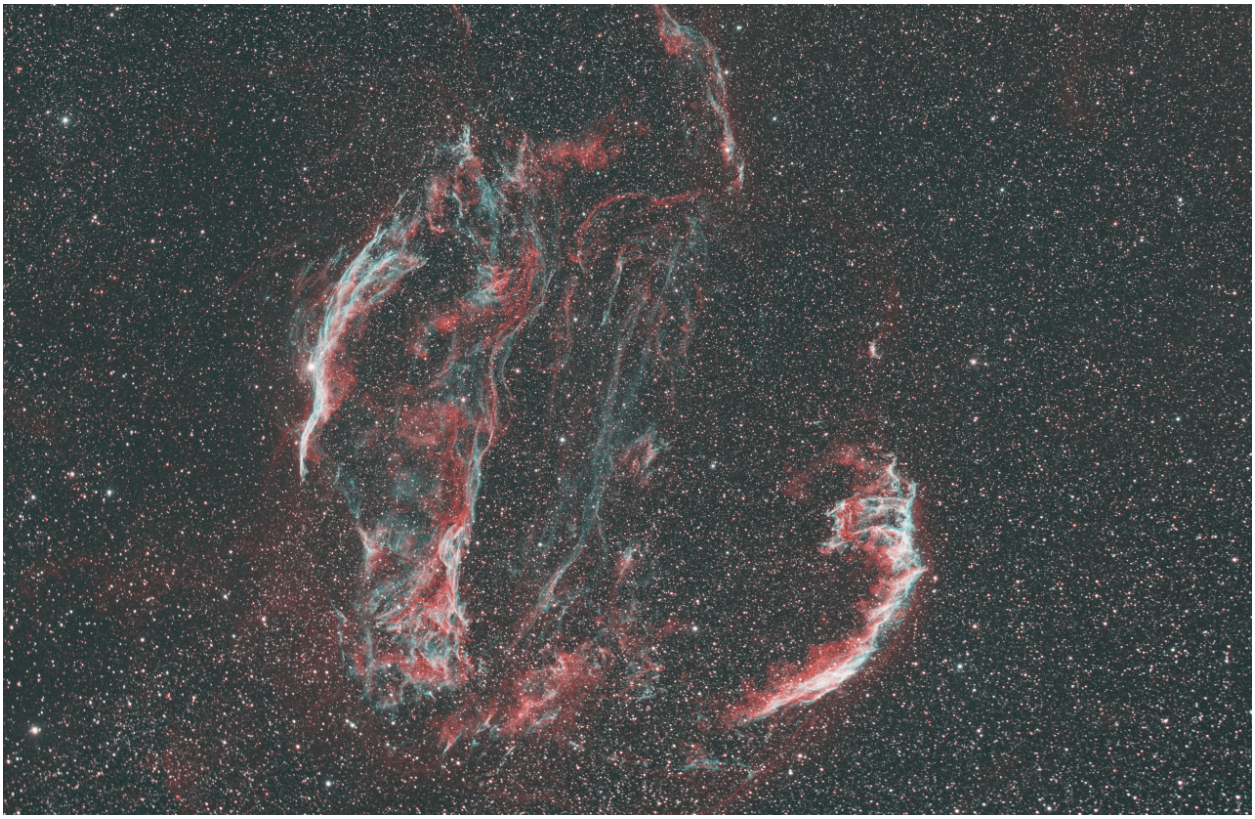


Fig. 15: Calibrated HOO image. Image by Cyril Richard.

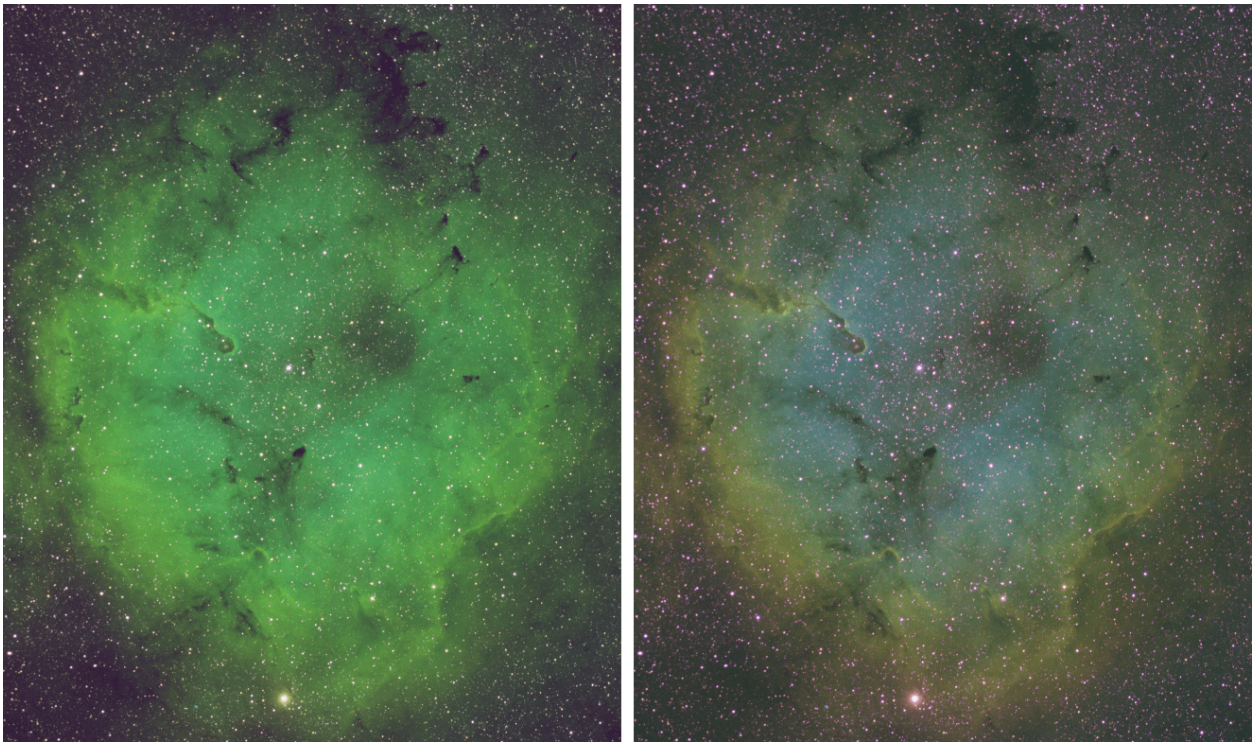


Fig. 16: SHO image calibrated by SPCC compared to the same, manually calibrated one. The entire nebula was taken as a white reference during manual calibration. Image by Cyril Richard.

- **Selection of White Reference** SPCC requires an absolute white reference spectrum. The default is **Average Spiral Galaxy** and the source spectra used to create this white reference are taken from the SWIRE templates [SWIRE] in a manner consistent with other astrophotography software providing the same white reference. A wide range of other white references is available, covering the full range of galaxy and star classifications [Stellar]. If you wish to use sunlight as your white reference, you would choose the white reference **Star, type G2(v)** as the Sun is a type G2(v) star.

> **Tip**
>
> **Summary of Stellar Spectral Classifications** Stellar classifications have two parts, a Morgan-Keenan type and a Luminosity index.
>
> The first part of the spectral classification (G2 in the case of the Sun) takes one of the following letters: O, B, A, F, G, K, M. O represents extremely hot blue stars, while M represents cool red stars. The sun is roughly in the middle of the spectrum. The number represents intermediate cases, for example a B5 star is halfway between type B and type A.
>
> The second part of the spectral classification is the luminosity index ranging from i to v. Stars with luminosity index i are supergiants, whereas stars with luminosity index v are dwarfs. Main sequence stars such as the sun have a luminosity index of iv.

- **Atmospheric Correction** Siril's SPCC supports atmospheric correction. When imaging from Earth we image through the atmosphere. This does not have perfect transmittance and therefore acts as another, non-optional, filter in the imaging chain between the sensor and the astronomical object. Whether or not to correct for this is an artistic choice each astrophotographer must make, but the option is provided.

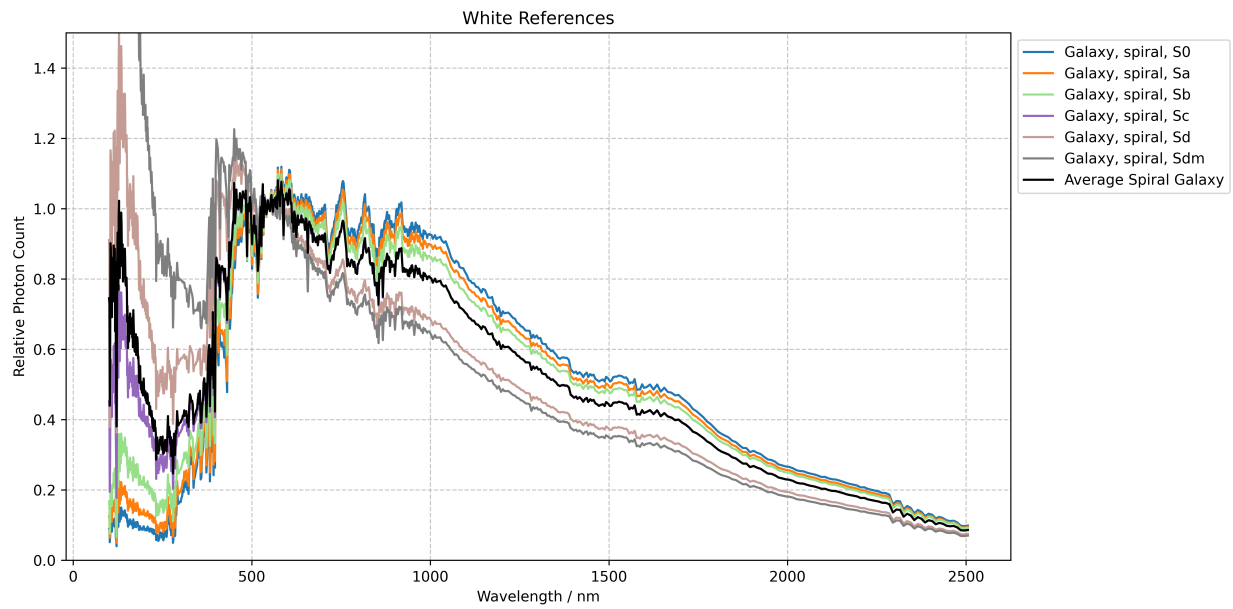



Fig. 17: Graphs showing white reference data from spiral galaxies. At around 350 nm, the Average Spiral Galaxy data become identical to the Sc galaxies, which are also a good representation of the white reference.

**Theory**

Atmospheric extinction arises from several sources. The most important are:

- Rayleigh scattering. This is the elastic scattering of light by particles that are small compared with the wavelength of light. The strong wavelength dependence of the Rayleigh scattering ($\approx^4$) means that shorter (blue) wavelengths are scattered more strongly than longer (red) wavelengths.
- Aerosol scattering. This is scattering of light by particles that are larger than the wavelength of light. This is quite variable but (in the absence of significant short term dust or smoke effects) relatively spectrally flat and less significant than Rayleigh scattering.
- Molecular absorption lines.

Siril models only Rayleigh scattering. This is the most important contribution in most atmospheric conditions, and is highly predictable making it easy to model without requiring the user to provide complex data.

The formula for the Rayleigh transmittance of the atmosphere as a function of wavelength $\lambda$ nm, observer height $H$ m and pressure $p$ hPa is:

$$\tau_R(\lambda, H, p) = \left(\frac{p}{1013.25}\right)\left(0.00864 + 6.5 \times 10^{-6} \cdot H\right)\lambda^{-(3.916+0.074\lambda+\frac{0.050}{\lambda})}.$$

Under normal circumstances aerosol scattering has a roughly flat response in the visible region. This changes in specific conditions, for example when there is high atmospheric smoke particle concentration after wildfires or, in parts of Europe, when Saharan dust is carried into the atmosphere. However these effects are very difficult to model accurately as they depend on the concentration of sand or smoke particles in the atmosphere at the time. Siril therefore does not model this effect.

The main molecular absorption lines in the visible spectrum are the Chappuis stratospheric ozone bands and the Fraunhofer B molecular oxygen absorption line. However the Fraunhofer B line is very narrow and does not have a significant effect on overall calibration. The Chappuis bands are broad but with a low peak

Fig. 18: NGC 4414 is a great example of a Sc-type galaxy, the type closest of the average spiral galaxy used as white reference by default. Image Credit: NASA, ESA, W. Freedman (U. Chicago) et al, & the Hubble Heritage Team (AURA/STScI), SDSS; Processing: Judy Schmidt.

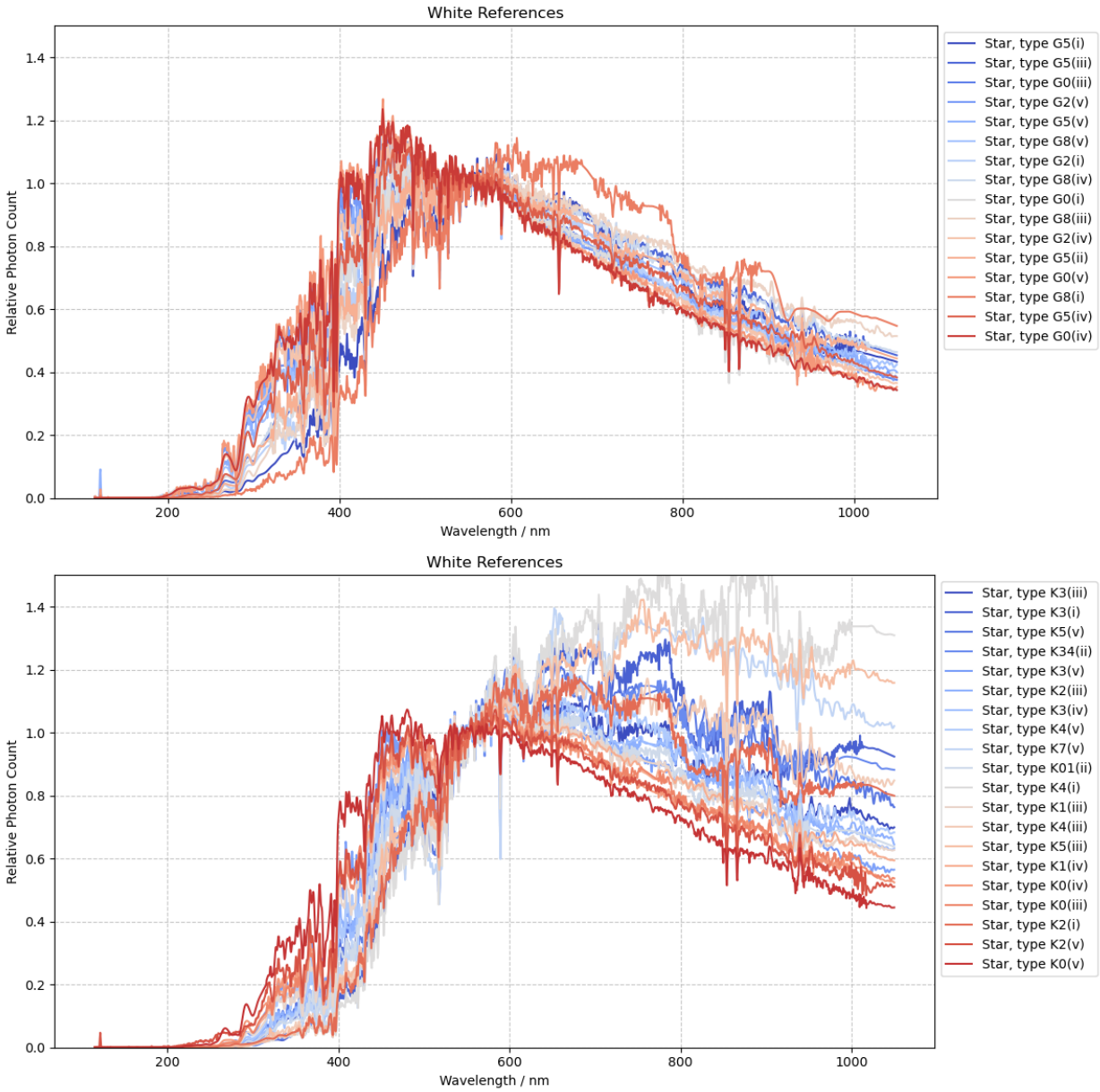



Fig. 19: Graphs showing white reference data for a set of two different star classes, G and K.

Fig. 20: Difference in color calibration depending on the choice of white reference. On the left, an M-type star, on the right the average spiral galaxy. Please note that the data are linear, and only an autostretch has been applied to the visualization.

absorption, with a much smaller overall impact than Rayleigh scattering. Molecular absorption bands are not currently modelled in Siril.

When selecting the *Atmospheric correction* check box, the following options become available:

- Observer Height. This allows setting of the observer height, which is used in the Rayleigh extinction calculation. Set this to the altitude of your observatory above sea level. Some capture software sets the FITS header `SITEELEV` card: if this is present, the height from this card will be used, otherwise the value is editable and defaults to 10 m.
- Atmospheric pressure. This allows setting atmospheric pressure at the time of observation. For convenience it can be specified as sea level pressure (as provided by weather forecasts) or as local pressure (as measured by a barometer at the observatory). In case you are unsure, the default is standard atmospheric pressure at sea level (1013.25 hPa).

**Theory**

If the pressure is provided as a sea-level pressure measurement, the local pressure at the observer's height is calculated according to the barometric formula:

$$P(h) = P_0 \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}},$$

where:

* $L = 0.0065$ K/m (Temperature lapse rate),
* $T_0 = 288.15$ K (Sea level standard temperature),
* $g = 9.80665$ m/s² (Acceleration due to gravity),
* $M = 0.0289644$ kg/mol (Molar mass of Earth's air),

* $R = 8.3144598\ \mathrm{J}/(\mathrm{molK})$ (Universal gas constant).

– Airmass. This is not an editable parameter but shows the airmass that will be used in the calculations. It is obtained, in order of preference, from the `AIRMASS` FITS header card; by calculation using the `CENTALT` FITS header card; or as a last resort by using the average zenith angle of all parts of the more than 30° above the horizon. The tooltip shows which source the used figure is based on.

**Theory**

If the `AIRMASS` header is unavailable the calculation used to derive airmass from zenith angle is calculated in accordance with [Young1994]:

$X(z) = \frac{1.002432\cos^2 z + 0.148386\cos z + 0.0096467}{\cos^3 z + 0.149864\cos^2 z + 0.0102963\cos z + 0.000303978}$.

The interface allows you to view details of the selected sensor, filter and white reference using the *Details* button next to each combo box. From the details information box that this brings up you also have the option to plot the Quantum Efficiency (for sensors) or transmittance (for filters) or relative photon count (for white references) against wavelength. A *Plot All* button is also available in the main SPCC dialog which allows you to see the responses of all your filters and your sensor and the white reference spectrum all plotted together.

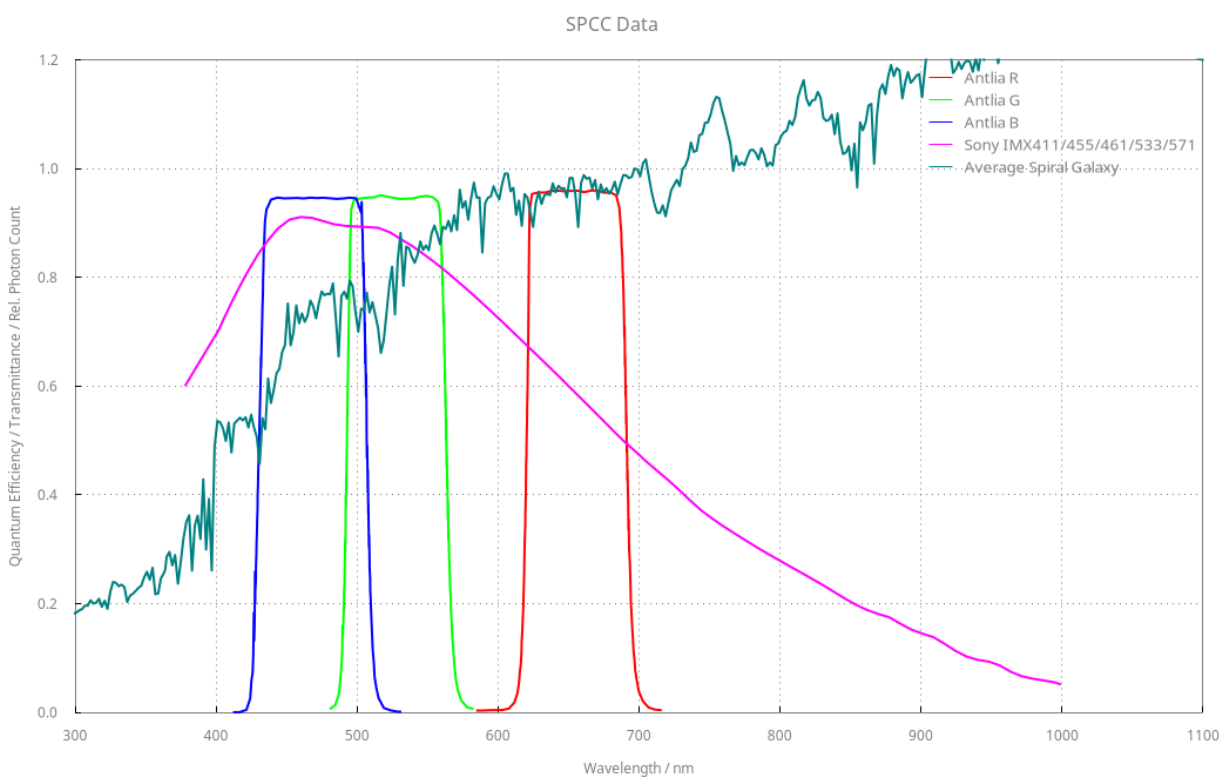



Fig. 21: Plotting all the responses of all your filters and your sensor and the white reference spectrum all plotted together

When you are happy, click *Apply* and SPCC will run. It will cache catalogue data but the first time you apply it to an image it will take a few seconds to perform the online catalogue searches and retrieve the source and spectral data. SPCC will then be applied to the image. Additional plots showing the linear fit of the catalogue Red / Green and Blue / Green to image Red / Green and Blue / Green ratios.

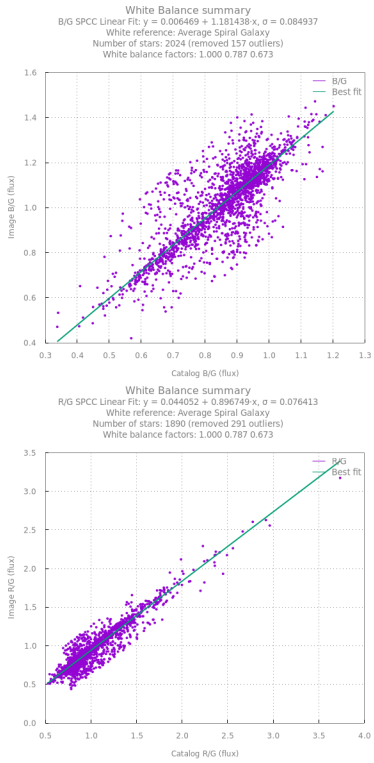



Fig. 22: By default, Siril outputs graphs showing the fits used in the process. In this example the magnitude was limited to 17.

**Tip**

**How do I process L-RGB images?** We recommend processing only RGB with SPCC. The L layer must be added at a later stage, when the histograms have been stretched.

**Tip**

For images taken with an OSC sensor, we recommend using *Bayer Drizzle* to recover image colors. This ensures more accurate colors as shown in the following image.

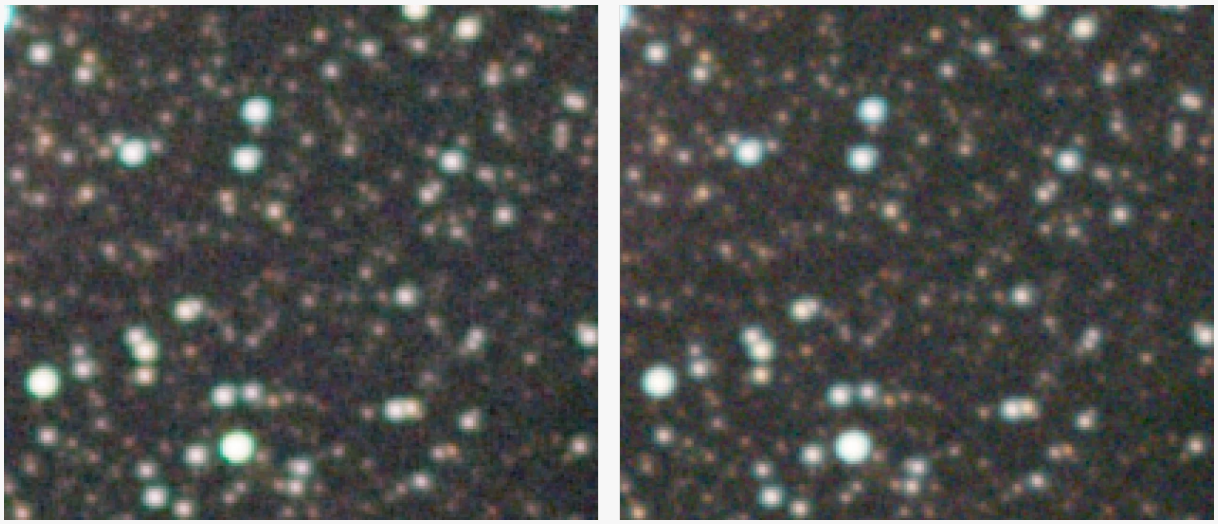


Fig. 23: SPCC applied identically to the same image. On the left, conventional demosaicing using the VNG algorithm; on the right, the Bayer Drizzle technique. A dominant green hue is clearly visible on the conventionally demosaiced image. **Note** that the VNG algorithm was chosen for this example because the effects explained here are more pronounced. However, in Siril, the default demosaicing algorithm is RCD. Click to enlarge image.

### SPCC filter and sensor database

#### Converting the Data

The format used for the database is JSON (a lightweight data-interchange format derived from JavaScript object notation). We recommend starting with an existing file from the database that suits your needs and saving it under the name of your sensor or filter. You can then simply replace the values in the file with the data you have obtained.

- In the `wavelength` array, enter your wavelength measurements. Make sure to properly set the `units` field to one of the following values: `angstroms`, `nm`, `micrometres`, or `m`.
- In the `values` array, enter either:
  - Transmittance values for filters
  - Quantum efficiency values for sensors

  Set the `range` field according to your data scale (e.g., `"range": 100` if your values are percentages, `"range": 1` if they are normalized to 1).

#### How to Contribute

The SPCC database is designed to store JSON files of OSC/monochrome sensors and filters available in the market. Its primary objective is to gather extensive data, fostering collaboration within the community.

We greatly value community contributions and encourage active participation. We are in need of data spanning ideally from 300nm to 1100nm. Software tools can be employed to extract curves/charts found online, and contacting manufacturers directly for data is also an option.

Read this page and help us by contributing.

**Important**

**We do not include narrowband filters**. These highly specific filters are synthesized in Siril, ensuring precision. This also applies to duonarrowband filters.

### JSON File Format Reference

Here is the template for the JSON files used in the SPCC database:

```
[
  {
    "model": "sensor model / filter set",
    "name": "sensor / filter name",
    "type": "MONO_SENSOR | OSC_SENSOR | MONO_FILTER | OSC_FILTER | OSC_LPF | WB_REF",
    "dataQualityMarker": 1 - 5,
    "dataSource": "Describe where the data came from",
    "manufacturer": "Manufacturer name",
    "version": 1,
    "channel": "RED | GREEN | BLUE | LUM",
    "wavelength": [Comma separated array of wavelengths],
    "values": [Comma separated array of values]
  }
]
```

### Important Notes

- Definition of the `dataQualityMarker` field:
  1. Data of unknown provenance. **Not accepted** for the siril-spcc-database repository.
  2. Data scanned from OEM or other reputable plots in image format.
  3. Lower resolution tabulated data provided by the OEM, or academic data relating to ideal standard filter transmittance (e.g. generic standard photometric filters).
  4. High resolution (no more than 2nm spacing) tabulated data provided by the OEM.
  5. Data specific to your own filter which you have personally calibrated using appropriate equipment. This is the highest possible quality marker and will never be given to .json files in the repository which can only ever be generic to an equipment model, not specific to your individual equipment item. Note that the actual quality of this data is entirely dependent on the quality of your calibration equipment - the old adage "garbage in, garbage out" applies.
- The `model` name requirements:
  - Must be identical for all related JSON objects in a set
  - Examples:
    - RGB filter set: `"model": "Chroma RGB"`
    - OSC sensor: `"model": "ZWO ASI2600MM"`
- The `channel` field:
  - Required only for `"type": "OSC_SENSOR"` or `"type": "MONO_FILTER"`
  - For OSC sensors, include one JSON object per channel (`RED`, `GREEN`, `BLUE`)
  - Preferred channel order: `RED`, `GREEN`, `BLUE`

- The `wavelength` array requirements:
  - Minimum coverage: 380nm to 700nm
  - Maximum useful range: 336nm to 1020nm (Gaia DR3 spectra limits)
  - Values must be monotonically increasing
  - No duplicate values allowed
  - Must use specified units (`angstroms`, `nm`, `micrometres`, `m`)

> **Note**
>
> If your sensor data only extends down to 400nm (which is common with some manufacturers), it is acceptable to extrapolate a single point at 380nm. The sensor response below 400nm typically follows a predictable pattern across different sensors. Adding this extrapolated point at 380nm is preferable to letting the curve end at 400nm, which would effectively treat all response below 400nm as zero. The impact of this extrapolation is minimal since the CIE 1931 response is very low in this wavelength range.

- The `values` array requirements:
  - For filters: contains transmittance values
  - For sensors: contains quantum efficiency values
  - Set appropriate `range` value (e.g., 100 for percentages)
  - Siril scales all values to 0.0-1.0 range internally

### Saved Preferences

As most users are likely to do most of their imaging with one setup, or maybe two, it would be tedious to reselect the sensor and filters each time. The user choices are therefore automatically remembered when set and restored next time the tool is used, even if Siril is closed and restarted in between. This works using the preferences system but there is no need to use the preferences dialog to remember the set sensor and filters, it is done automatically.

The chosen white reference is not remembered: the default Average Spiral Galaxy is a suitable choice for most astronomical scenes, and alternative white references would normally be set for a specific image to draw out a particular aspect of the color.

> **Siril command line**

```
spcc [-limitmag=[+-]] [ { -monosensor= [ -rfilter= ] [-gfilter=] [-bfilter=] | -
↪oscsensor= [-oscfilter=] [-osclpf=] } ] [-whiteref=] [ -narrowband [-rwl=] [-gwl=]␣
↪[-bwl=] [-rbw=] [-gbw=] [-bbw=] ] [-bgtol=lower,upper] [ -atmos [-obsheight=] { [-
↪pressure=] | [-slp=] } ]
```

Run the Spectrophotometric Color Correction on the loaded platesolved image.

The limit magnitude of stars is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.
The star catalog used for SPCC is always Gaia DR3: by default the local Gaia DR3 xp_sampled catalog will be used if available but this can be overridden with **-catalog={gaia | localgaia}**.

The names of sensors and filters can be specified using the following options: **-monosensor=**, **-rfilter=**, **-gfilter=**, **-bfilter=** or **-oscsensor=**, **-oscfilter=**, **-osclpf=**; the name of the white reference can be specified using the **-whiteref=** option. In all cases the name must be provided exactly as it is in the combo boxes in the SPCC tool. Note that sensor, filter and white reference names may contain spaces: in this case when using them as arguments to the **spcc** command, the entire argument must be enclosed in quotation marks, for example "-whiteref=Average Spiral Galaxy".

Narrowband mode can be selected using the argument **-narrowband**, in which case the previous filter arguments are ignored and NB filter wavelengths and bandwidths can be provided using **-rwl=**, **-rbw=**, **-gwl=**, **-gbw=**, **-bwl=** and **-bbw=**.

If one of the spectral data argument is omitted, the previously used value will be used.

Background reference outlier tolerance can be specified in sigma units using **-bgtol=lower,upper**: these default to -2.8 and +2.0.

Atmospheric correction can be applied by passing **-atmos**. In this case the following optional arguments apply: **-obsheight=** specifies the observer's height above sea level in metres (default 10), **-pressure=** specifies local atmospheric pressure at the observing site in hPa, or **-slp=** specifies sea-level atmospheric pressure in hPa (default pressure is 1013.25 hPa at sea level)

```
spcc_list { oscsensor | monosensor | redfilter | greenfilter | bluefilter | oscfilter␣
↪| osclpf | whiteref }
```

Print a list of SPCC names available for use to define sensors, filters or white references using the **spcc** command. This command requires an argument to set which list is printed: the options are **oscsensor**, **monosensor**, **redfilter**, **greenfilter**, **bluefilter**, **oscfilter**, **osclpf** or **whiteref**.

Note that sensor, filter and white reference names may contain spaces: in this case when using them as arguments to the **spcc** command, the entire argument must be enclosed in quotation marks, for example "-whiteref=Average Spiral Galaxy"

Links: *spcc*

### References

## 10.3 Other Color Tools

### 10.3.1 Color Saturation

This tool is used to increase the color saturation of the image. It is possible to choose between a specific hue or the global hue to enhance. The strength of the saturation is adjusted with the slider *Amount*.

The *Background factor* slider sets the factor multiplied by the background value. Lower is the value, stronger is the saturation effect. While a high value will preserve the background.

**Siril command line**

```
satu amount [background_factor [hue_range_index]]
```

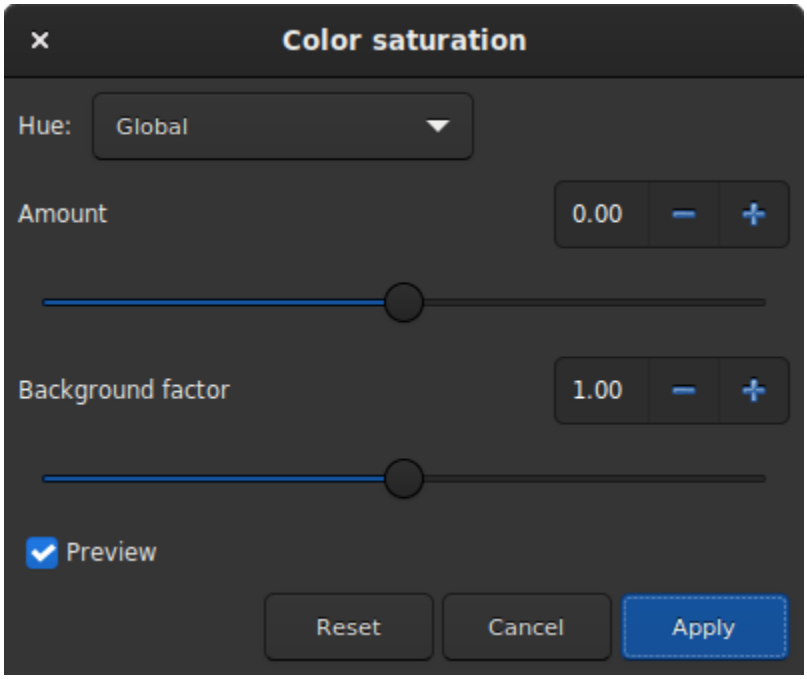



Fig. 24: Color Saturation dialog window.

Enhances the color saturation of the loaded image. Try iteratively to obtain best results.

**amount** can be a positive number to increase color saturation, negative to decrease it, 0 would do nothing, 1 would increase it by 100%

**background_factor** is a factor to (median + sigma) used to set a threshold for which only pixels above it would be modified. This allows background noise to not be color saturated, if chosen carefully. Defaults to 1. Setting 0 disables the threshold.

**hue_range_index** can be [0, 6], meaning: 0 for pink to orange, 1 for orange to yellow, 2 for yellow to cyan, 3 for cyan, 4 for cyan to magenta, 5 for magenta to pink, 6 for all (default)

### 10.3.2 Remove Green Noise

Because green is not naturally present in deep sky images (except for comets and some planetary nebulae), if the image has already been calibrated, its colors are well balanced and the image is free of any gradient, we can assume that if the image contains green, it belongs to the noise. It is then interesting to find a method to remove this dominant green. This is exactly what the Remove Green Noise tool proposes, which is derived from the Subtractive Color Noise Reduction tool, but for green only.

**Warning**

This tool is not intended for direct use on a typical green image from a stack where the background sky level has not been equalized. Its use in such conditions would destroy the image's chrominance.

**Warning**

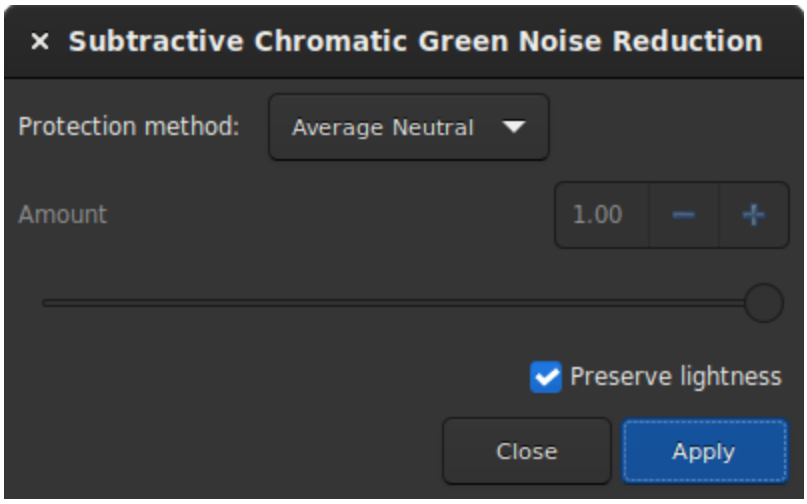



Fig. 25: Remove Green Noise dialog window.

This tool is designed to be used on non-linear images, as it operates in a non-linear color space. Make sure the histogram has been stretched before using this tool to avoid incorrect processing results.

This tool has 3 settings. The protection method, the amount (called $a$ in the following section), and a *Preserve lightness* button. The following methods present the different existing ways to remove the green pixels by replacing them with a mix of Red and Blue. The amount is only available for methods with mask protection. The choice of its value must be done with caution in order to minimize the rise of the magenta cast in the sky background. Do not hesitate to use the *Undo* and *Redo* buttons in order to fine-tune the value.

### Protection method

#### Maximum Mask Protection

$$m = \max(R, B)$$
$$G' = G \times (1^{-}a) \times (1^{-}m) + m \times G$$

#### Additive Mask Protection

$$m = \min(1, R + B)$$
$$G' = G \times (1^{-}a) \times (1^{-}m) + m \times G$$

#### Average Neutral Protection (default method)

$$m = 0.5 \times (R + B)$$
$$G' = \min(G, m)$$

#### Maximum Neutral Protection

$$m = \max(R, B)$$
$$G' = \min(G, m)$$

Finally, the *Preserve lightness* button preserves the original CIE L* component in the processed image, in order to only process chromatic component, it is highly recommended to let this option checked.

**Siril command line**

```
rmgreen [-nopreserve] [type] [amount]
```

Applies a chromatic noise reduction filter. It removes green tint in the current image. This filter is based on PixInsight's SCNR and it is also the same filter used by HLVG plugin in Photoshop.
Lightness is preserved by default but this can be disabled with the **-nopreserve** switch.

**Type** can take values 0 for average neutral, 1 for maximum neutral, 2 for maximum mask, 3 for additive mask, defaulting to 0. The last two can take an **amount** argument, a value between 0 and 1, defaulting to 1

### 10.3.3 Negative Transform

Negative transformation refers to subtracting pixel values from $(L-1)$, where $L$ is the maximum possible value of the pixel, and replacing it with the result.

The *Negative transformation* tool is different from the negative view in the toolbar. Indeed, the transformation is not only visual, but actually applied to the pixel values. If you save the image, it will be saved as a negative.

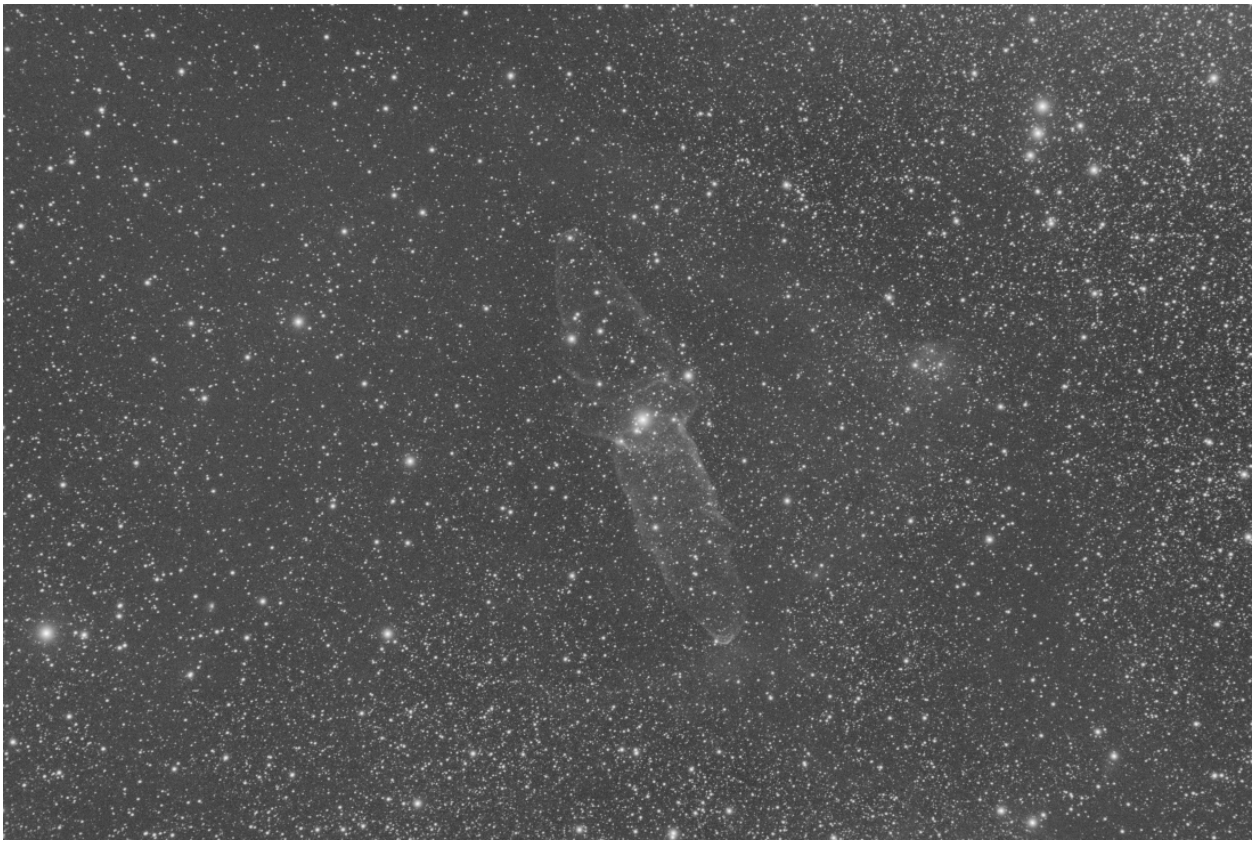


Fig. 26: Original image with weak signal (Image Cyril Richard).

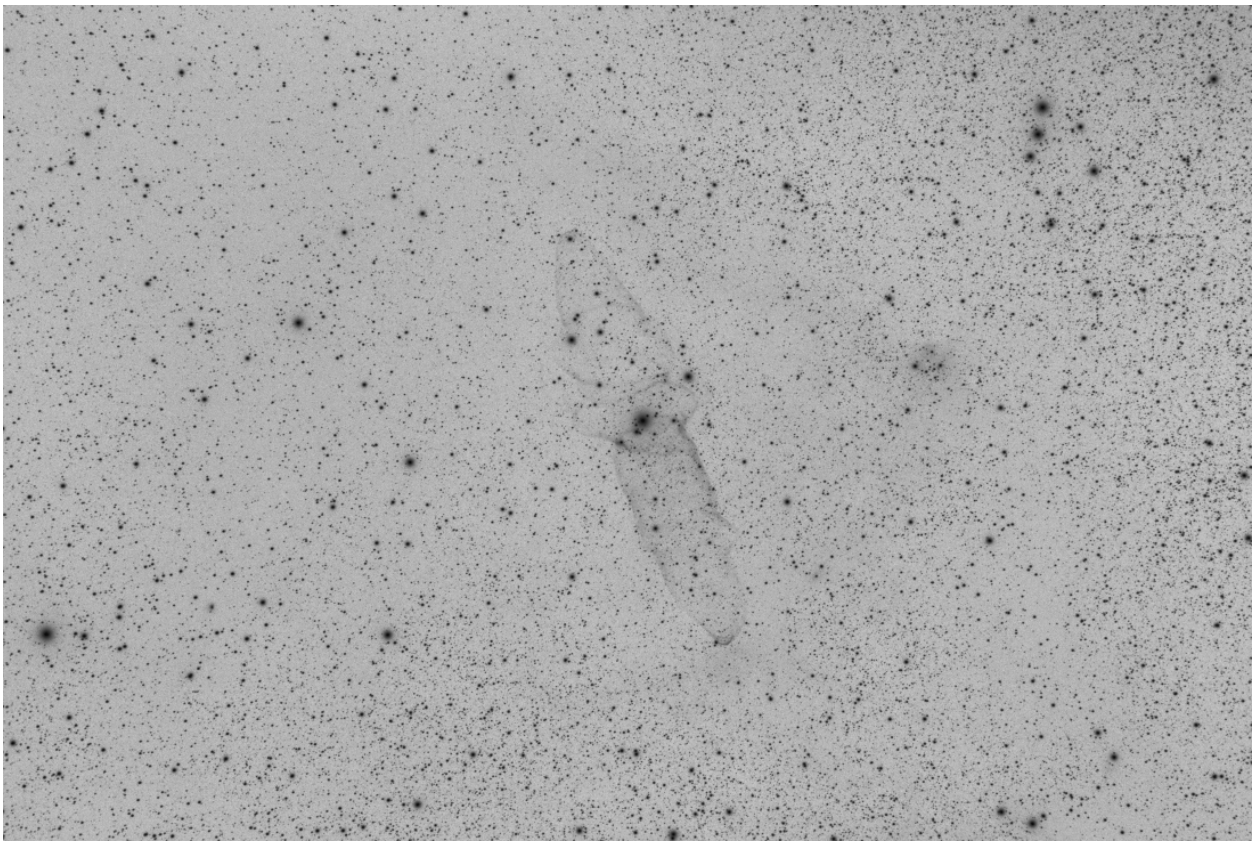


Fig. 27: Negative image where the signal is more visible (Image Cyril Richard).

**Tip**

A common use of the negative transformation tool is to remove the magenta cast from SHO images. In this case one need to apply *Negative transformation*, then *Remove Green Noise*, then *Negative transformation* again.

**Siril command line**

```
neg
```

Changes pixel values of the currently loaded image to a negative view, like 1-value for 32 bits, 65535-value for 16 bits. This does not change the display mode

## 10.4 Filters

This section presents all the filters present in Siril. Filters are tools that will modify the pixels of the image according to the needs.

### 10.4.1 À Trous Wavelets Transform

A wavelet is a function at the base of the wavelet decomposition, a decomposition similar to the short term Fourier transform, used in signal processing. It corresponds to the intuitive idea of a function corresponding to a small oscillation, hence its name.

There are many types of wavelet functions that have their own names, as shown in the figure below.

The À Trou Wavelet Transform used in Siril performs decomposition of an image into a series of scale layers, also known as wavelet layers.

These layers can be extracted with the *Wavelet Layers* extraction tool, however here, they are used without being visually accessible. In general, this algorithm is widely used at the end of a planetary image stack. Because the noise is exclusively contained in one of the wavelet layers, it is possible to bring out the details of the image by containing the noise amount.

The first thing to do is to click on the *Execute* button in order to calculate the wavelet layers using the parameters defined above, such as:

- **Type**: There are two types of algorithms possible: Linear and BSpline. The latter will usually be chosen, even if it is a bit slower.
- **Nb of layers**: Number of wavelet layers that will be used. 6 is the maximum number of layers that can be defined. To work on a larger number of layers it is possible to use the command line explained below.

Then, each layer has a slider that allows to modify the contrast of this layer. If less than 6 layers have been created, then only the corresponding sliders will be active. A value greater than 1 improves the details while a smaller value tends to reduce them.

This is a liveview tool. The changes are displayed in real time and you have to click on *Apply* to validate them. Clicking on *Reset* resets all the sliders to 1, and thus cancels any transformation in progress.

**Siril command line**

```
wavelet nbr_layers type
```

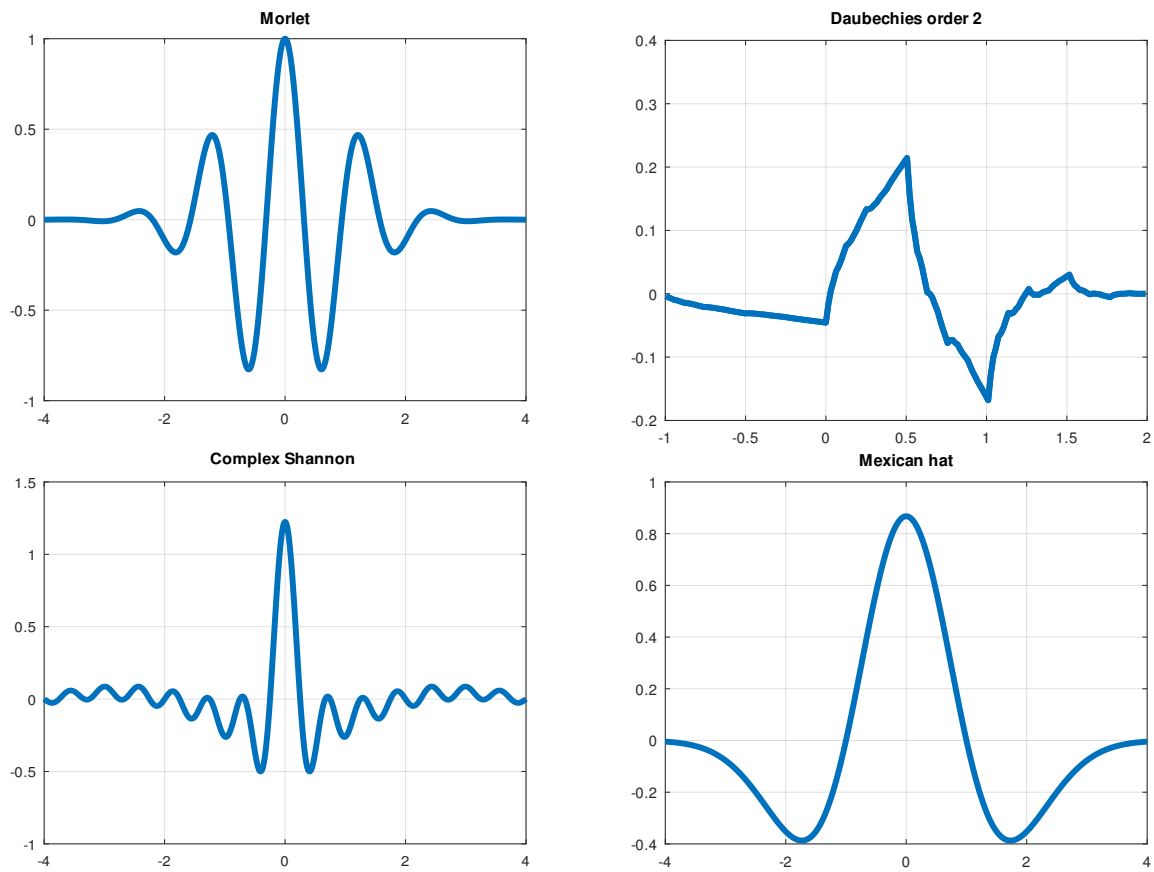



Fig. 28: An example of four different types of wavelets.

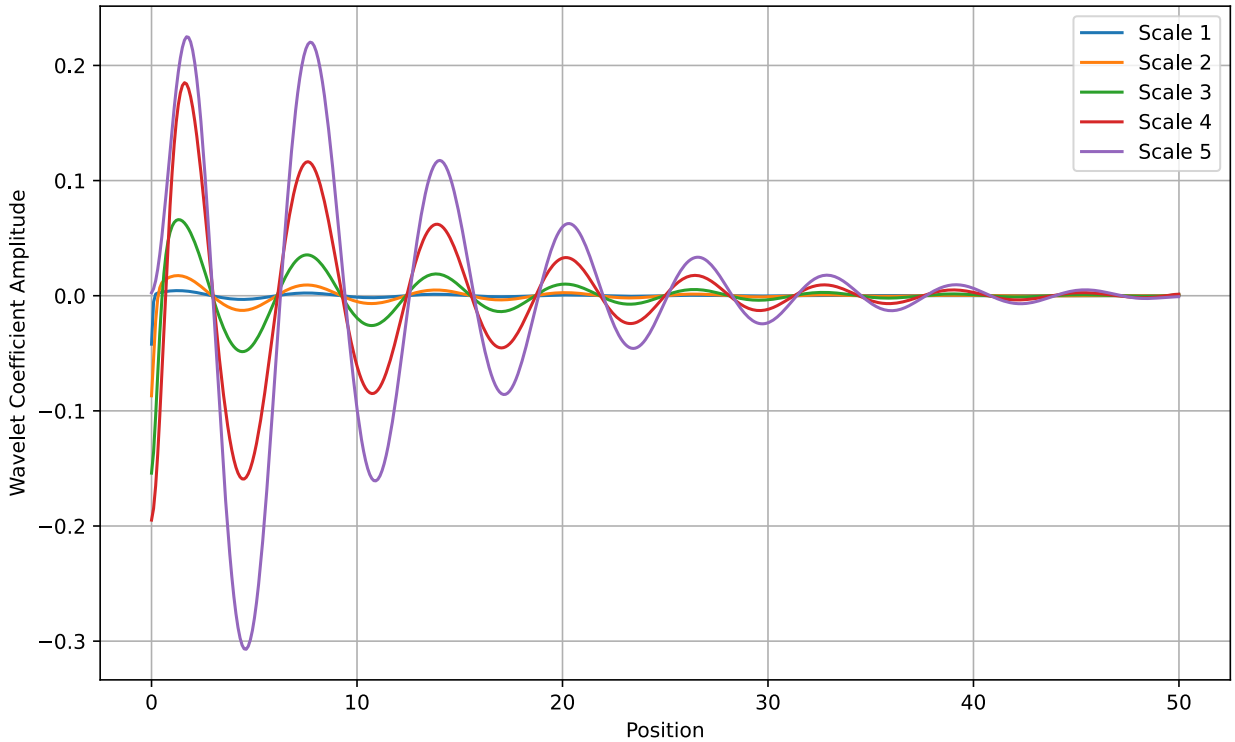



Fig. 29: À Trous Wavelets Transform representation with bspline interpolation.

Computes the wavelet transform of the loaded image on (**nbr_layers**=1...n) layer(s) using linear (**type**=1) or bspline (**type**=2) version of the 'à trous' algorithm. The result is stored in a file as a structure containing the layers, ready for weighted reconstruction with WRECONS.

See also EXTRACT

Links: *wrecons*, *extract*

**Siril command line**

```
wrecons c1 c2 c3 ...
```

Reconstructs to current image from the layers previously computed with wavelets and weighted with coefficients **c1**, **c2**, ..., **cn** according to the number of layers used for wavelet transform, after the use of WAVELET

Links: *wavelet*

The example given in the image above would be written in the command line as follow:

```
wavelet 6 2
wrecons 31 5 1 1 1 1
```

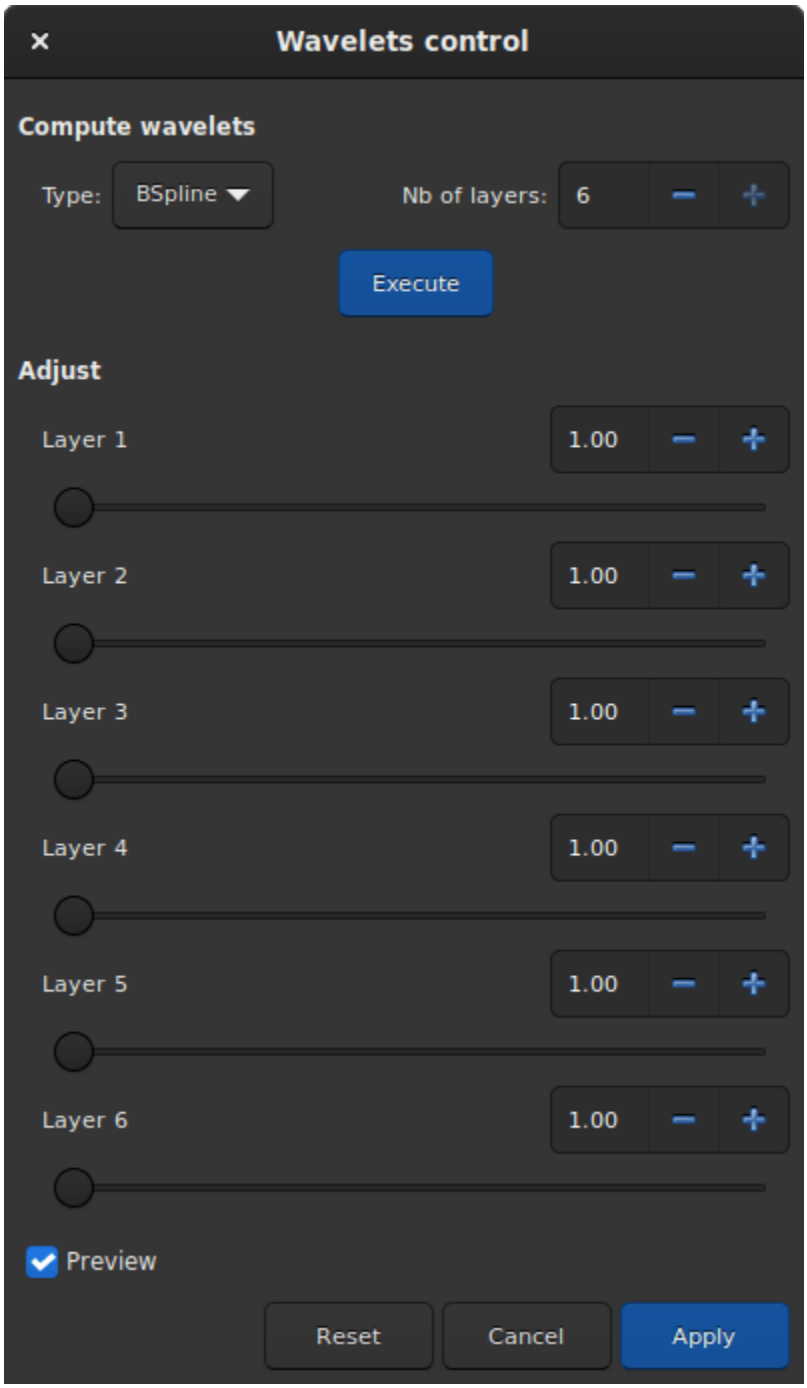



Fig. 30: Wavelet tool dialog box.

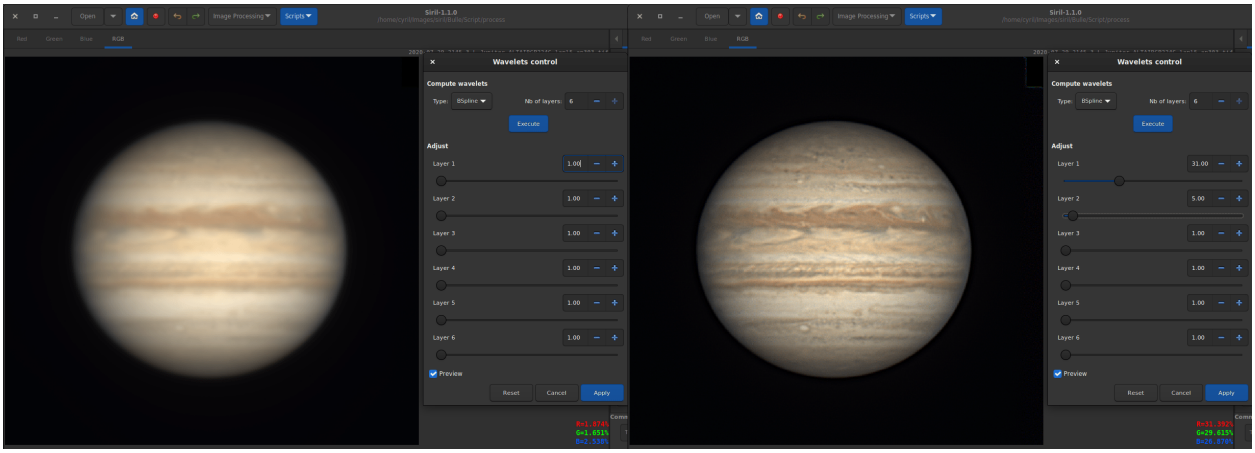



Fig. 31: Wavelets applied on a Jupiter image (courtesy of J.-L. Dauvergne). The image on the left is the raw image of the stacking output, while the image on the right is the same image on which wavelets are applied.

### 10.4.2 Banding Reduction

In some cases, images may suffer from a banding defect. This is usually caused by the sensor, or appears in short exposures due to electronic noise; calibration by darks, bias and flats do not improve the images.

The banding reduction window dialog has some parameters to optimize the processing:

- **Amount** defines the strength of the correction. The higher the value the stronger the correction, but a value of 1 should be good in most cases as it applies the same correction as ideally computed.
- **Protect from Highlights** will ignore bright pixels when the option is checked, this can be useful for images with large bright objects.
- **Sigma Factor** will adjust the highlight protection. Higher value will ignore only pixels that are brighter.
- **Vertical banding** removes vertical banding instead of the default horizontal banding. This should rarely be needed, but some cameras also have this problem.

Applying the following filter to the original image, with parameter values as shown in the illustration, gives a nice result free of banding.

This filter can also be applied to a sequence. Adjust the filter for the loaded image (with a sequence already loaded), then check the *Apply to sequence* button and define the output prefix of the new sequence (`unband_` by default).

**Siril command line**

```
fixbanding amount sigma [-vertical]
```

Tries to remove the horizontal or vertical banding in the loaded image.

**amount** defines the amount of correction, between 0 and 4.

**sigma** defines the highlight protection level of the algorithm, higher sigma gives higher protection, between 0 and 5. Values of 1 and 1 are often good enough.

**-vertical** option enables to perform vertical banding removal, horizontal is the default

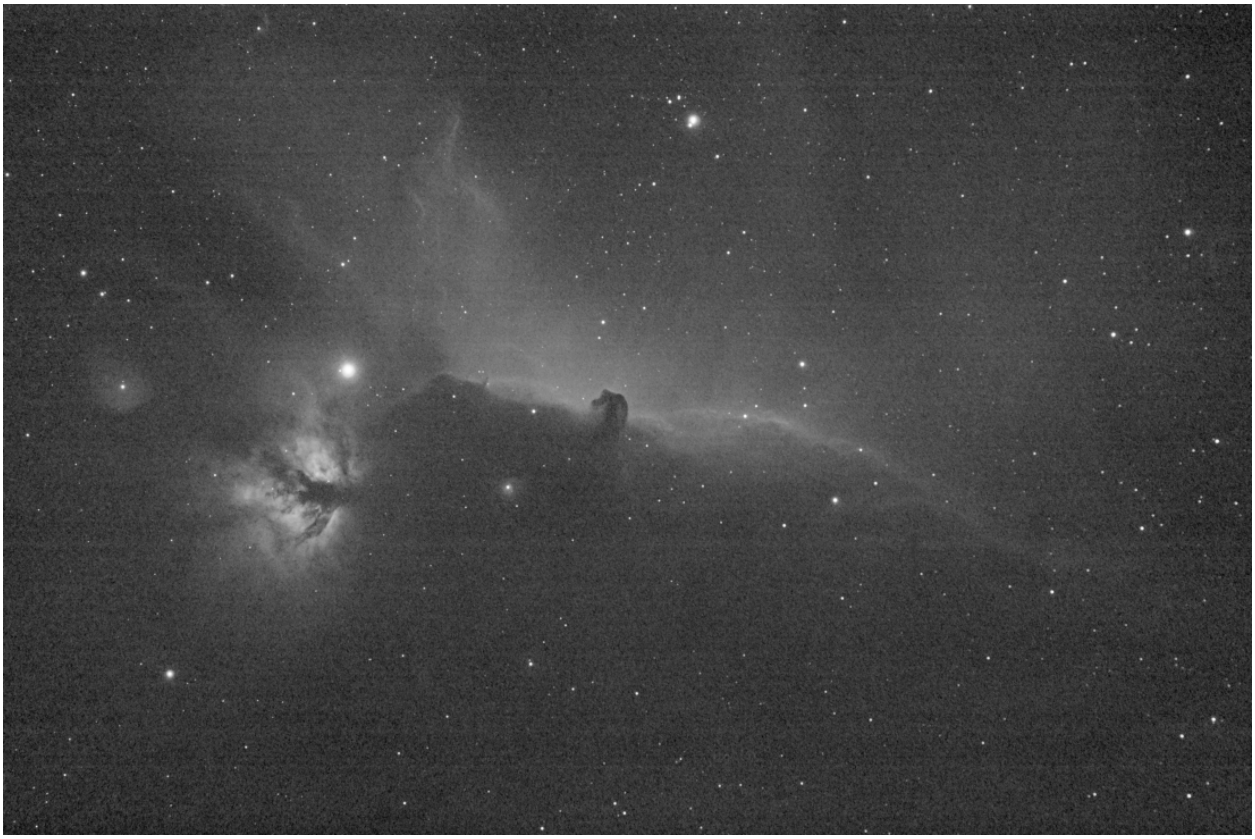


Fig. 32: Original image with visible horizontal banding.

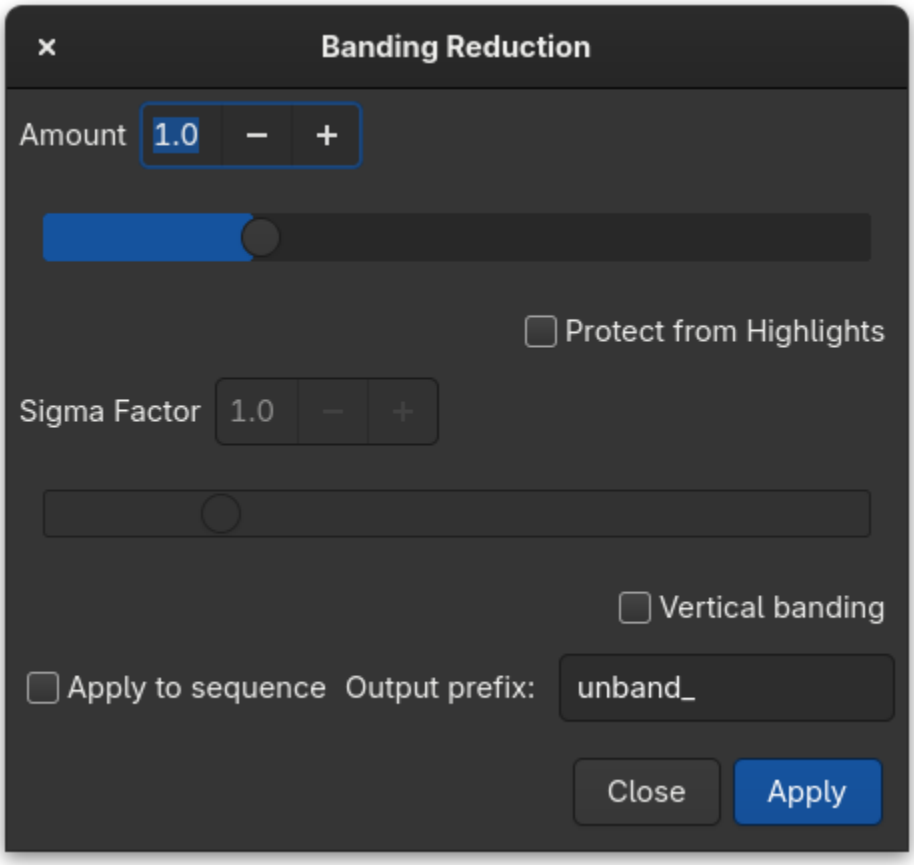



Fig. 33: Banding Reduction dialog box.

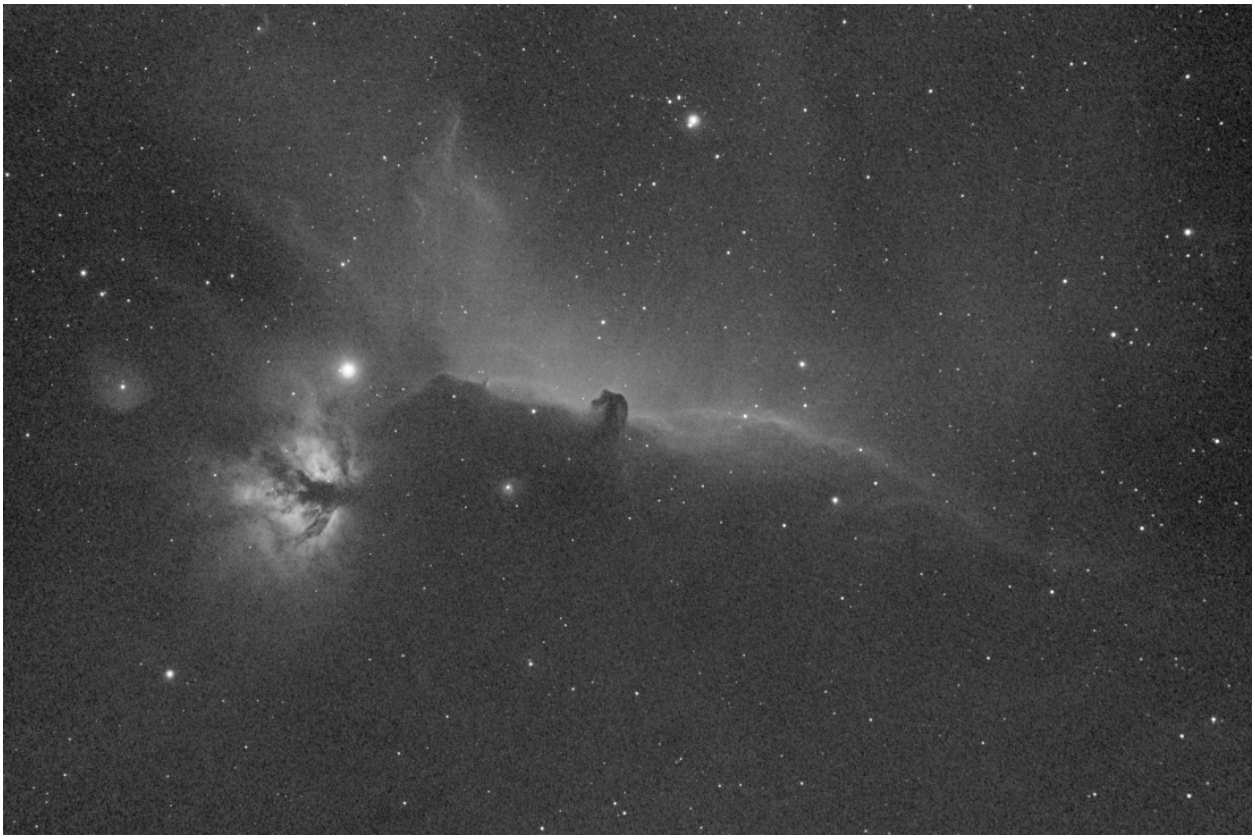


Fig. 34: Result after the filter has been run. No more bandings are visible.

**Siril command line**

```
seqfixbanding sequencename amount sigma [-prefix=] [-vertical]
```

Same command as FIXBANDING but for the sequence **sequencename**.

The output sequence name starts with the prefix "unband_" unless otherwise specified with **-prefix=** option

Links: *fixbanding*

### 10.4.3 Contrast-Limited Adaptive Histogram Equalization (CLAHE)

The CLAHE method is used to improve the contrast of images. It differs from ordinary histogram equalization in that the adaptive method calculates multiple histograms, each corresponding to a separate section of the image, and uses them to redistribute the brightness values of the image. It can therefore improve local contrast and enhance edge definition in each region of an image.

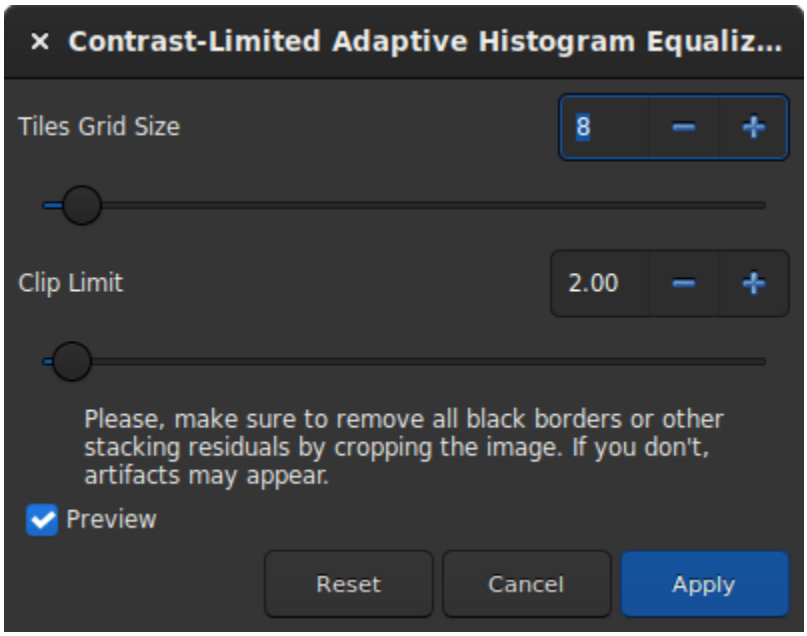



Fig. 35: Dialog box of the Contrast-Limited Adaptive Histogram Equalization filter.

**Tip**

This filter is a liveview filter. In other words, every change in the settings is automatically visible on the screen, but this can be disabled by unchecking the *Preview* button.

- The size of the tiles, in which the histograms are calculated, can be defined via a slider. By default it is set to 8.
- The Clip Limit is the option that prevents to overamplify noise in relatively homogeneous regions of an image. Then, the clipped part of the histogram that exceeds the clipping limit is redistributed equally among all the bins of the histogram.

**Tip**

This filter works better on non-linear data. It is recommended to stretch the image before.

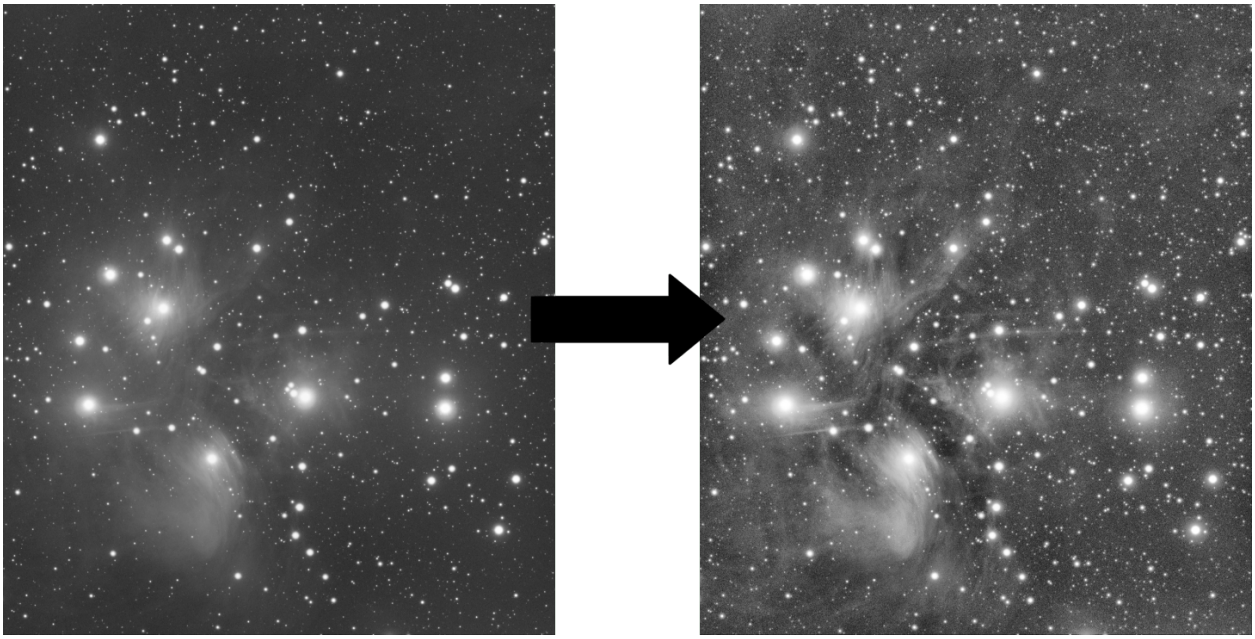


Fig. 36: An example of CLAHE filter applied to a non-linear data with `Tiles Grid Size=21` and `Clip Limit=4.20`.

**Siril command line**

```
clahe cliplimit tileSize
```

Equalizes the histogram of an image using Contrast Limited Adaptive Histogram Equalization.

**cliplimit** sets the threshold for contrast limiting.
**tilesize** sets the size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles

### 10.4.4 Cosmetic Correction

In Siril, the cosmetic correction is the step that gets rid of hot and cold pixels in the image. It is usually done during pre-processing using the master-dark. This is because the latter usually contains a good map of the defective pixels and it is easier to find them on it. However, when you don't have a master-dark, Siril offers an alternative with an automatic detection algorithm of these pixels in a light image.

The dialog window contains several parameters necessary for the proper functioning of the tool. However, using the default settings usually gives good results.

- **Cold Sigma**: How many times (in average deviation units) a pixel value must differ from the value of surrounding neighbors to be considered as a cold pixel.

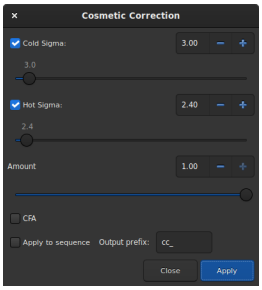



Fig. 37: Dialog window of Correction Cosmetic tool.

- **Hot Sigma**: How many times (in average deviation units) a pixel value must differ from the value of surrounding neighbors to be considered as a hot pixel.
- **Amount**: This is a modulation parameters where 0 means no correction and 1, 100% corrected.
- **CFA**: This option needs to be checked for CFA images with Bayer pattern. It does not work for X-Trans sensor.

This operation can be applied to sequences. Open a sequence and prepare the settings you want to use, then check the *Apply to sequence* button and define the output prefix of the new sequence (`cc_` by default).

**Theory**

### Hot pixels detection

Let's call $m_{5\times5}$ the median of the 5 nearest neighbors. If the pixel value is greater than

$$m_{5\times5} + \max(\text{avgDev}, \sigma_{\text{high}} \times \text{avgDev}),$$

with avgDev, the *Average Deviation* of the whole image.

Then the pixel is replaced by the average of the $3 \times 3$ pixels: $a_{3\times3}$, but only if

$$a_{3\times3} < m_{5\times5} + \frac{\text{avgDev}}{2}.$$

### Cold pixels detection

If the pixel value is less than

$$m_{5\times5} - (\sigma_{\text{low}} \times \text{avgDev}),$$

then the pixel is replaced by $m_{5\times5}$.

Fig. 38: Animation showing cosmetic correction.

**Siril command line**

```
find_cosme cold_sigma hot_sigma
```

Applies an automatic detection and replacement of cold and hot pixels in the loaded image, with the thresholds passed in arguments in sigma units

**Siril command line**

```
find_cosme_cfa cold_sigma hot_sigma
```

Same command as FIND_COSME but for CFA images

Links: *find_cosme*

### 10.4.5 Deconvolution

Deconvolution is a mathematical tool to compensate for blurring or distorting effects in an image. The true scene is not what is recorded on your sensor - you record an estimate of the true scene convolved by a PSF (in mathematical terms, the "blurring PSF" that represents atmospheric distortion, physical properties of your telescope, motion blur and so on, degrading your estimate). Deconvolution can, to an extent, reverse this image degradation. However, it is important to say up front that deconvolution is what mathematicians call an ill-posed problem (like most inverse problems). Ill-posed means that either a solution may not exist, if a solution does exist it may not be unique, and it may not have continuous dependence on the data. Essentially it means deconvolution is, even theoretically, really hard and there are no guarantees it will work.

All this is made even harder when we don't know exactly what the PSF is that we're trying to remove. In astronomy we can in theory get an idea of the PSF by the effect of blurring on the point sources (stars) that we are imaging. However sometimes the true PSF isn't constant across the image, sometimes other factors such as star saturation prevent the star PSF being an entirely accurate estimate of the PSF, and sometimes (for example lunar imaging) there are no stars.

Siril aims to provide a flexible approach to deconvolution. There are several options for defining or estimating the PSF, and several deconvolution algorithms to choose from for the final stage of deconvolution once the PSF is defined.

Fig. 39: Example of deconvolution on star field.

Deconvolution is accessed through the *Image Processing* menu or using Siril commands.

#### Overview of Usage

- To generate a deconvolution PSF, select the required PSF generation method and press *Generate PSF*. This can be performed separately from the actual deconvolution so that the user can see the effect of changing the PSF parameters.
- Siril only generates monochrome PSFs as this is the most common use case and simplifies the user interface. However, three monochrome PSFs can be saved and composited to produce a 3-channel PSF which can be loaded and used to deconvolve 3-channel images.
- To apply deconvolution to a single image, select the required PSF generation method and press *Apply*. If a blind PSF estimation method has previously been run, the method will automatically be set to *Previous PSF*, in order to avoid unnecessarily recalculating the PSF.
- To apply deconvolution to a sequence, proceed as above but ensure that you activate the *Apply to Sequence* checkbox. You may also specify a custom prefix to give to the output sequence: if no other prefix is provided, the default (`dec_`) will be used.

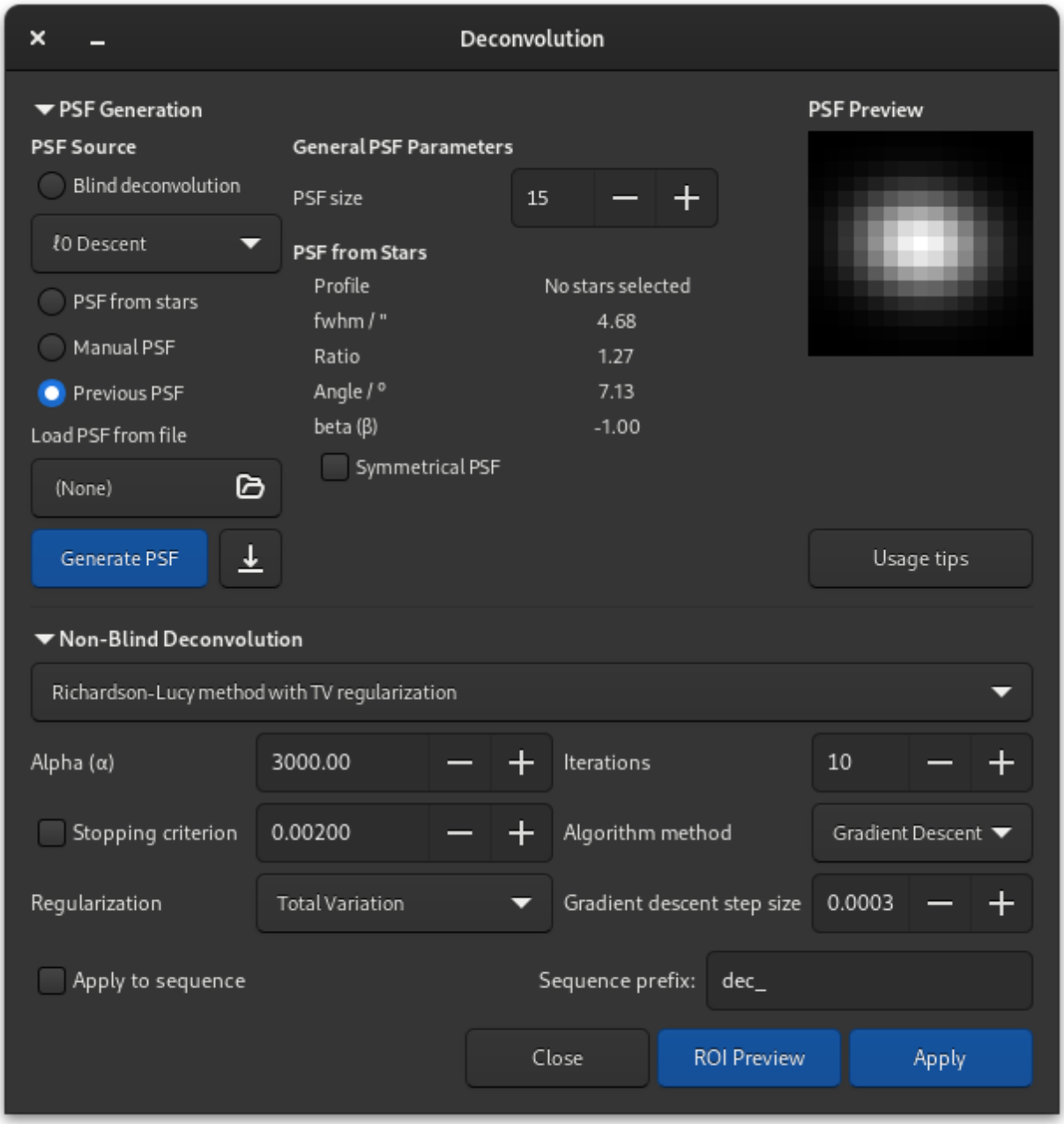



Fig. 40: Dialog box of the deconvolution tool.

- When deconvolving a sequence, the PSF will be calculated only for the first image. The same PSF will be re-used for all images in the sequence.

### Overview of Blur Kernel Definition Methods

- $_0$ **Descent**: This is the default PSF estimation method based on work by Anger, Delbracio and Facciolo. The parameters do not generally require adjustment, except that for particularly large PSFs you may wish to try the multiscale PSF estimation model. Multiscale is off by default as during development it was noted to have a tendency to produce rather unnatural results with the more common small to medium PSF sizes.
- **Spectral Irregularities** [Goldstein2012]: This PSF estimation method is offered as an alternative. In general it does not perform as well as the $_0$ descent method, however it may be useful if you discover an image where the default method does not give good results. For this method the latent sharp image needs not to contain any edges as long as the spectral decay model is respected. On the other hand, the $_0$ descent assumes a similar model (since edges have the same spectral decay), but requires to have sparse gradients and be contrasted, thus edges to be in-phase, so theoretically this model may work better on low contrast starless images. Some experimentation is likely required to find the algorithm that best fits your data.
- **PSF From Stars**: This method models a PSF from the average PSF of the selected stars. It is important to be selective about the stars you choose: they must not be saturated as that would give a gross distortion of the PSF estimate, but they must also not be so faint that Siril's star analysis functions provide inaccurate measurements of the stars. The stars chosen should be reasonably bright, fairly central to the image and in an area of the image with a fairly constant background. Once stars are selected, you can pick either a Gaussian or Moffat star profile model and when executing the deconvolution the PSF will be synthesized from the average parameters of the selected stars. If no stars are selected, Siril will attempt to autodetect stars with a peak amplitude between 0.07 and 0.7, with a Moffat profile. This range avoids saturated stars as well as those that are too faint to give an accurate solution, and generally provides good results.
- **Manual PSF**: This method allows you to define a PSF manually. Gaussian, Moffat or disc PSF models can be defined. Note that the FWHM is specified in pixels, not arc seconds. The Gaussian and Moffat models are suitable for deconvolving the shapes of stars resulting from atmospheric distortion; the disc PSF model is suitable for deconvolving the effect of being slightly out of focus.
- **Load PSF from file**: This method allows you to load a PSF from any image format supported by Siril. The provided PSF must be square (it will be rejected if not square) and should be odd (it will be cropped by one pixel in each direction if not odd, but this will give a slightly off-centre PSF and is not optimal compared with providing an odd PSF in the first place). Either monochrome or 3-channel PSFs may be loaded. If a 3-channel PSF is loaded in conjunction with a monochrome image, the evenly-weighted luminance values of the PSF will be used. If a 3-channel PSF is loaded together with a 3-channel image then each channel of the image will be deconvolved using the corresponding channel of the PSF. If a monochrome PSF is loaded together with a 3-channel image then the image will be converted to the LAB colour space and the L (Luminance) channel will be deconvolved using the monochrome PSF for computational efficiency, and the deconvolved L will be recombined with the A and B channels and converted back to RGB.
- **Previous PSF**: This method allows reuse of the previously estimated PSF. It is mostly of use with the blind PSF estimation methods: if you are content with the estimated PSF but wish to make a number of test runs using different parameters for the final stage of deconvolution, you can reuse the previous PSF and save some computation time.
- Once estimated, PSFs may be saved if desired. If Siril is compiled with libtiff support then the PSF will be saved in 32-bit TIFF format, with the same filename as the current image but date-and-time-stamped and suffixed with `_PSF`. If Siril has been built without libtiff support, the PSF will be saved as a FITS file. While this is Siril's primary format for astronomical image files, TIFF is preferred for PSFs: the disadvantage of using the FITS format for PSFs is potential reduced compatibility with image editors that you may wish to use to edit or examine the saved file.

**Tip**

If the blind generation of a deconvolution PSF can be done on linear and non-linear data, the use of a PSF from star PSF can only be done on linear images. Otherwise the PSF values would not be valid.

**Tip**

If a ROI is set, blind PSF estimation methods will calculate the PSF from the ROI rather than the whole image. If you do wish to calculate a PSF from the whole image you must clear the ROI before estimating the PSF, then set the ROI to apply a preview. However, you also have the option of calculating the PSF only from a selected part of the image. This may be desirable if you have optical aberrations close to the edge of the image and wish to estimate the PSF from the central area only.

### Overview of Non-Blind Deconvolution

- **Richardson-Lucy Deconvolution** [Lucy1974]: This is the default non-blind deconvolution algorithm. It is an iterative method, famous for its use in correcting image distortions in the early operating period of the Hubble Space Telescope, and in Siril is regularized using either the Total Variation method, which aims to penalize the algorithm for amplifying noise, or the Frobenius norm of the local Hessian matrix. This regularization is based on second derivatives. As well as regularization an early stopping parameter is provided, which allows the algorithm to be halted early once its rate of convergence falls below a certain level. Increasing the value of the early stopping parameter can reduce ringing around stars and sharp edges. Two formulations of the Richardson-Lucy algorithm are provided: the multiplicative formulation and the gradient descent formulation. The latter can allow better control, as the gradient descent step size can be altered (the downside of this is that by using more small steps, more iterations are required to reach the same level of convergence). The bigger advantage of the gradient descent method is that it allows more regularization to be used - this can be problematic in the multiplicative Richardson-Lucy algorithm as the regularization term appears in the denominator and small values here (strong regularization) can cause instability. Siril will use naive convolution for small kernel sizes and FFT-based convolution for larger kernel sizes where FFTs provide a more efficient algorithm. (This is automatic and requires no user intervention.)
- **Wiener Filtering Method**: This method is a non-iterative deconvolution method. It models an assumed Gaussian noise profile, i.e. noise modelled by a constant profile. The constant alpha is used to set the regularization strength in relation to the noise level. As with the other algorithms, a smaller value of alpha provides more regularization. This algorithm can be good for lunar images where the noise regime is Gaussian not Poisson, but usually works badly on deep space imagery where the noise still tends to have a Poisson character.
- **Split Bregman Method**: This method is used internally within the blur PSF estimation processes, and is also offered as a final stage deconvolution algorithm. It is a commonly used algorithm in solving convex optimization problems. This algorithm is also regularized using a total variation cost function. It does not perform as well as Richardson-Lucy on starscapes but may be considered for starless images or lunar surface images.

**Tip**

Choice of deconvolution method is very important to obtaining good results. Generally for DSO images it is important to use a Richardson-Lucy method: both the Split Bregman and Wiener methods give poor results around stars because of the extreme dynamic range. For linear images it is usually best to use the gradient descent Richardson-Lucy methods, and if ringing occurs around bright stars then reduce the step size. This approach reduces the impact of each iteration therefore more iterations are required, but it does mean that you can achieve finer control taking deconvolution just up to the point where artefacts start to form and then backing off very slightly. For stretched images the multiplicative Richardson-Lucy algorithms may be used.

> **Tip**
>
> For stacked lunar and planetary images the Split Bregman or Wiener methods can be more appropriate. These methods do not generally require iteration in the way that Richardson-Lucy does, and they may be better suited to the noise characteristics of stacked, high signal-to-noise ration images. (The Richardson-Lucy algorithm is based on an assumption of Poisson noise, which is usually true for DSO imaging, whereas the Wiener method implemented here assumes a Gaussian noise distribution which may fit stacked planetary / lunar images better).

### Parameters and Settings

### General Settings

- PSF size. The input PSF size should be chosen sufficiently large to assure that the PSF is included in the specified domain. However, setting it too large can result in a poorer and more time-consuming result from the blind PSF estimation methods.
- Lambda ($\lambda$). Regularization parameter for PSF estimation. Try decreasing this value for noisy images.

### 0 descent PSF estimation settings

- Multiscale. This setting enables multiscale PSF estimation. This may help to stabilize the PSF estimaate when specifying a large PSF size, but some PSFs generated with this option can give rise to unnatural looking results and it is therefore off by default.
- Expert settings. These should not normally require adjustment, but are made available for the curious.
  - Gamma sets the regularization strength used when carrying out the sharp image prediction step. For a given gamma, as the noise increases the estimation also gets noisier. If gamma is increased, the estimation is less affected by noise but tends to be smoother. The default value of 20 was determined experimentally in [Anger2019].
  - Iterations sets the number of iterations used in the PSF estimate procedure. The authors of the algorithm report that there is minimal benefit in increasing this to 3 and no benefit at all in increasing it beyond 3.
  - Lambda ratio and lambda minimum sets the parameters for refining the sharp image prediction through successive values of the sharp image predictor regularization parameter at each iteration of the method.
  - Scale factor, upscale blur and downscale blur are only used when multiscale estimation is active. These set the default scale factor between each scale level and the amount of blurring to use when rescaling between each scale.
  - Kernel threshold. Values below this level in the PSF estimate are set to zero.

### Spectral Irregularity PSF estimation settings

- Compensation factor controls the strength of a filter used to avoid excessive sharpness in the estimated PSF. For images with intrinsic blur, a value close to unity should be used. For intrinsically sharp images, low values can result in artefacts and the value should be increased to a large number, effectively disabling the filter.
- Expert settings. These should not normally require adjustment, but are made available for the curious.
  - Inner loop iterations sets the number of iterations performed in the inner loop of the spectral irregularity method. The algorithm converges quickly and it may be possible to reduce this to approximately 100 without much degradation of the result.
  - Samples per outer loop. This controls how many random phases should be sampled. Because the phase retrieval starts with random values for each sample, it is important to draw enough samples to avoid converging to a local minimum. The PSF stabilizes quickly for low noise images, but if looking for improved

results from this method, this is the first of the expert settings to try adjusting especially with images with higher noise levels.

- Outer loop iterations. [Anger2018], suggests that 2 iterations can be enough to produce a plausible PSF estimate, and there is negligible value in increasing this above 3.

### PSF from Stars

- This PSF generation method has no adjustable parameters. It generates a PSF based on the average parameters of the selected stars, using the *findstar* command or the *Dynamic PSF* dialog. The average parameters are shown in the deconvolution dialog when this PSF generation method is selected. It is preferable for the user to actively select the stars they wish to use for this method, to obtain the most accurate PSF. Ideally around 10 fairly bright but not saturated stars should be selected from the central region of the image (to exclude stars that may suffer from coma or other aberrations). However, if the user has not selected any stars, Siril will attempt to autodetect suitable stars by running its detection routine with filters set to keep only stars with peak amplitudes between 0.07 and 0.7. This range avoids both saturated stars and those that are too faint to give an accurate solution. It will work well in most cases but may still be affected by off-centre aberrations.
- If you select the *Symmetrical PSF* checkbox, the generated PSF will be circular. This will match the average FWHM and beta of the selected stars but will not match any elongation.

### Manual PSF

This PSF generation method allows specification of a custom parametric PSF.

- Profile type allows choice of PSF profile. Gaussian, Moffat, disc and Airy disc PSFs are supported.
  - Gaussian and Moffat PSFs are used for matching star parameters measured from the image. They should provide a good estimate of the total blur function being applied to the image, as stars are point sources.

Fig. 41: An example of Moffat PSF with fwhm=5", Angle=45°, Ratio=1.20, $\beta = 4.5$ and a PSF size of 15.

  - Disk PSFs are used to deconvolve images that are out of focus.

Fig. 42: An example of Disk profile with fwhm=5" and a PSF size of 15.

  - Airy disc PSFs are used to deconvolve the diffraction that arises as a physical consequence of diffraction by the aperture of your telescope.

Fig. 43: An example of an Airy-Disk PSF with Diameter=250mm, Focal Length=4500mm, Wavelength=525nm, Pixel Size=2.9µm, Central Obstruction=40% and a PSF size of 41.

- FWHM specifies the full width at half maximum of the chosen profile (for disc PSFs it simply specifies the radius).
- Beta ($\beta$) specifies the beta parameter used in the Moffat PSF profile. It is ignored for other PSF profiles.

- For Airy disc PSFs a number of parameters of your telescope and sensor are required:
  - Aperture
  - Focal length
  - Sensor pixel size
  - Central wavelength being imaged. Siril will try to extract this data from your image metadata where available, but if some parameters are missing or look unreasonable Siril will highlight them and print a warning in the log recommending you check them. The ratio of the central obstruction is also required to generate an accurate Airy disc. This is expressed as a percentage, i.e. the total area of the central obstruction divided by the total area of the aperture x 100. For refractors this is zero; for other telescopes it varies: it may be around 20% for a Newtonian reflector or as much as 40-50% for some Corrected Dall-Kirkham telescopes. You will need to measure your instrument or consult the manufacturer's specifications.

### Richardson-Lucy Deconvolution

The parameters used to configure Richardson-Lucy deconvolution in Siril are as follows:

- **alpha** sets the regularization strength. A smaller value of alpha gives stronger regularization and a smoother result; a larger value reduces the regularization strength and preserves more image detail, but may result in the amplification of noise.
- **Iterations** specifies the maximum number of iterations to use. In the absence of noise, a large number of iterations will cause deconvolution to converge the estimate closer to the true image, however an excessively large number of iterations will also magnify noise and cause ringing artefacts around stars. The default is 1 iteration: a higher number can be set to compute multiple iterations automatically, or you can keep pressing *Apply* to apply one iteration at a time until you are happy with the result. (Or go one further, decide you're no longer happy and use *Undo*.)
- **Stopping criterion** sets a convergence criterion based on successive estimate differences. This will stop the algorithm once convergence is within the specified limit. This is an important parameter - if you are getting rings around stars in your fnal image, try increasing the value of the stopping criterion. This may be disabled altogether using the check button.
- **Algorithm method** specifies whether to use the multiplicative implementation or the gradient descent implementation.
- **Step size** specifies the step size to use for the gradient descent implementation. Do not set it too large or the algorithm will not converge. This parameter has no effect if the multiplicative implementation is selected.

> **Tip**
>
> For linear images, try using the gradient descent methods provide the control necessary to prevent ringing around stars. For deconvolving stretched images, however, this can be unnecessarily slow, so using the multiplicative methods can often save time without compromising image quality.

### Split Bregman Deconvolution

The parameters used to configure Split Bregman deconvolution in Siril are as follows:

- **alpha** sets the regularization strength. A smaller value of alpha gives stronger regularization and a smoother result; a larger value reduces the regularization strength and preserves more image detail, but may result in the amplification of noise.
- **Iterations** specifies the maximum number of iterations to use. The Split Bregman method does not require multiple iterations in the form implemented here, but may be iterated if desired. This generally makes only a small difference and therefore defaults to 1.

### Wiener Deconvolution

Wiener deconvolution in Siril only requires one parameter:

- **alpha** sets the regularization strength. A smaller value of alpha gives stronger regularization and a smoother result; a larger value reduces the regularization strength and preserves more image detail, but may result in the amplification of noise.

### FFTW Performance Settings

The PSF estimation and deconvolution algorithms make extensive use of fast Fourier transforms using the FFTW library. This offers a number of tuning options, which can be adjusted in the performance tab of the main Siril *preferences* dialog.

### Note on Image Row Order

Different types of image processed by Siril can have their pixel data arranged in different orders. SER video files always store data top down, whereas FITS files may store data either bottom up or top down. Bottom up is the original recommendation, however increasingly FITS are sourced from CMOS cameras which tend to follow a top down pixel order.

When an image is deconvolved with a PSF created from the same image (or with it open) this causes no problem. However there is potential for problems to arise if a PSF is generated with an image with one row order and used to deconvolve an image or sequence with the opposite row order. This is a niche use case, but handling it consistently results in behaviour which at first sight can be surprising: it is therefore explained below.

Siril handles the issue by tracking the row order of the image the PSF was created with. PSFs are always saved using a bottom up row order (automatically flipping them if they were created with a top down image), and when loaded the row order is matched to the row order of the currently open image. If an image of the opposite row order is opened, the row order of the PSF will be changed to match. This means that if, for example, you take some bottom up FITS images, use one of them to generate a PSF, and then convert them to a top down SER sequence, the PSF will be converted to the correct orientation to match the SER sequence. If a PSF is being previewed at the time an image with the opposite row order is opened the preview will not update immediately: the row order change will be detected automatically and the PSF flipped at the time when it is applied to the image.

### Rogues' Gallery

This section shows some examples of where deconvolution has gone wrong, together with explanations of why.

### Deconvolution: Usage Tips

You've arrived here from the hints button in the deconvolution tool in Siril. No worries: deconvolution is a tricky technique. Even theoretically, it's really hard: there are no guarantees the maths will always converge to a unique solution that improves your image. That said, here are some tips to help you get the most out of Siril's deconvolution algorithms.

### What PSF to use?

Using an accurate PSF is fundamental to achieving good results from deconvolution. The two simplest ways to generate a PSF are to use a blind PSF estimate, or to model your PSF on stars in your image.

### PSF from Stars

Siril can detect and model stars in your image. See the Dynamic PSF help page for details. To get a good model for your PSF, try selecting the Moffat star profile in Dynamic PSF. Stars are point sources so the spread function of an average star is a good model for the blurring effects that we are trying to remove by deconvolution.

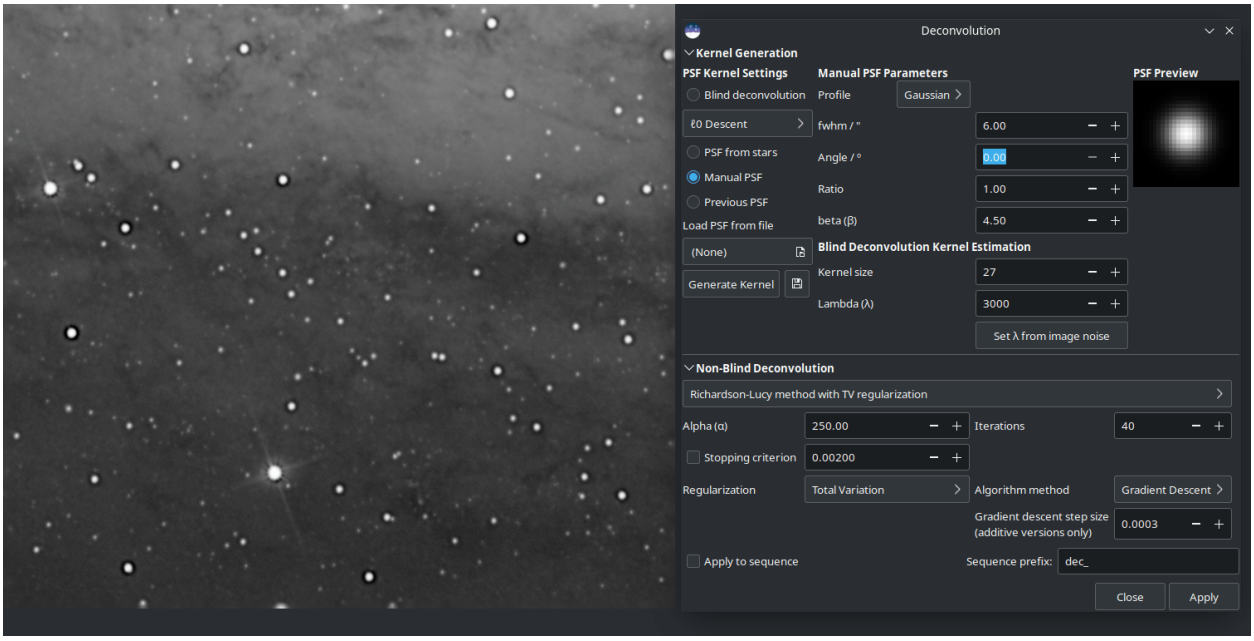



Fig. 44: The manually specified PSF was too big, resulting in large dark rings around stars.

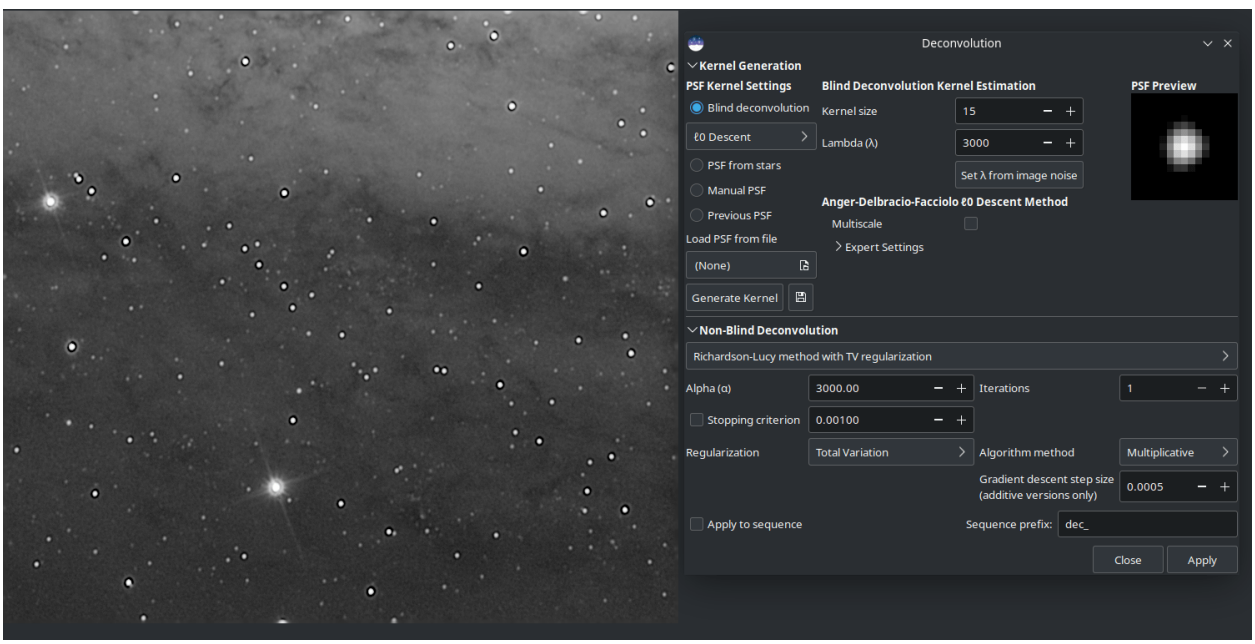



Fig. 45: Too many iterations have been applied. (I applied them one at a time to exaggerate the result, which is why the iterations parameter still says 1.)

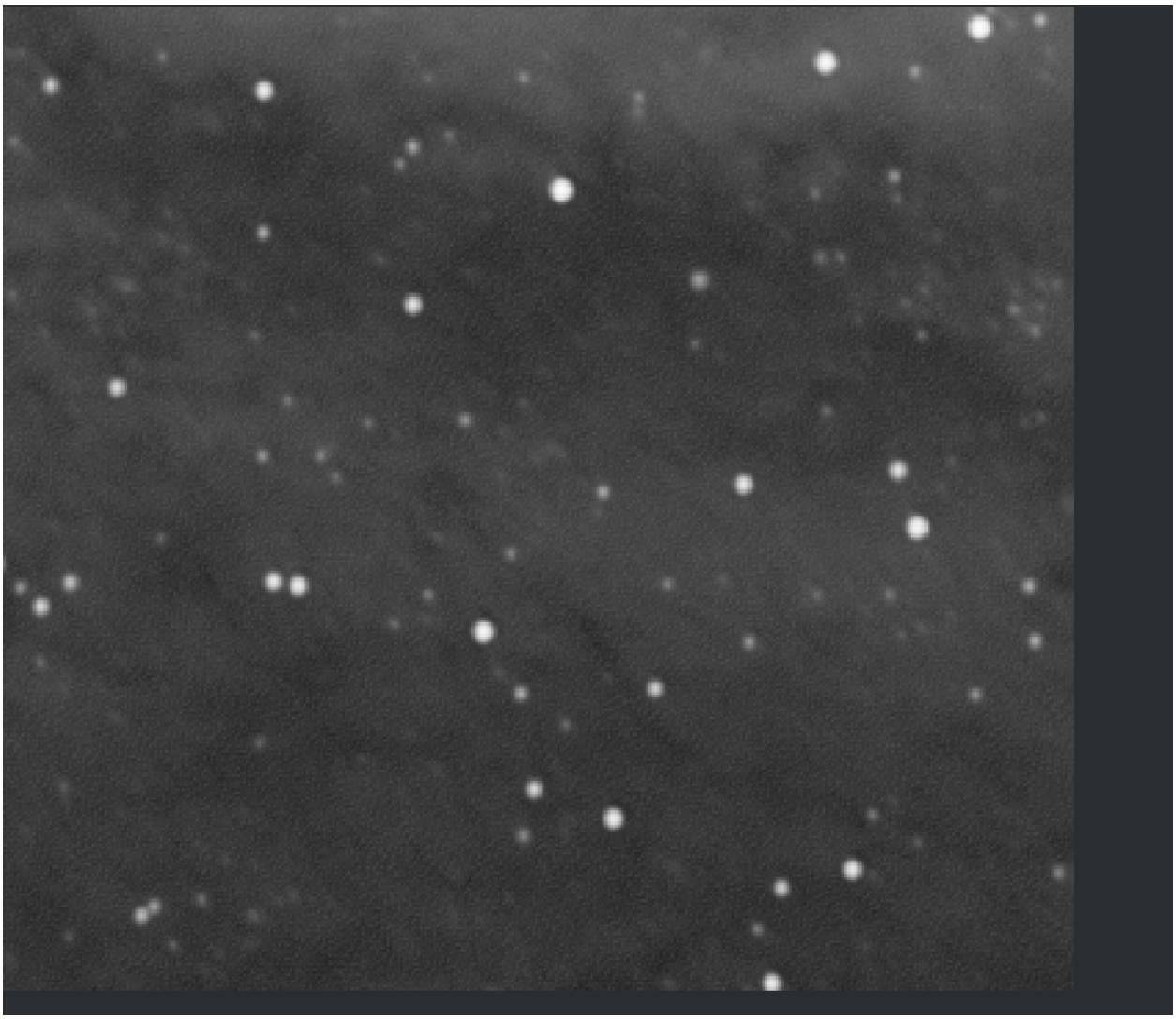


Fig. 46: Close-up showing the effect of trying to apply too much regularization (alpha = 30) using the multiplicative version of Richardson-Lucy. For strong regularization and / or better control over each iteration, the gradient descent formulation is recommended.

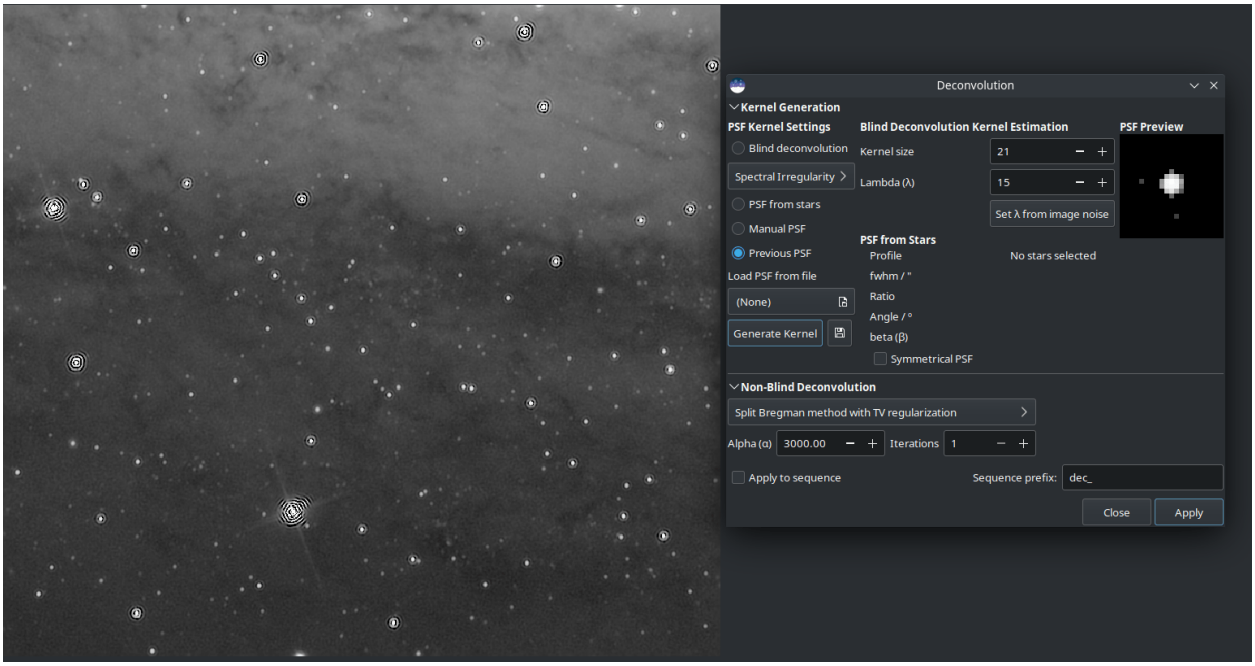



Fig. 47: Typical example of attempting to deconvolve an unstretched starfield using Split Bregman (in this case) or Wiener filter deconvolution. Those are better for planetary / lunar / solar images; for starscapes Richardson-Lucy is always recommended.

> **Tip**
>
> Once you have detected stars, sort them by peak amplitude (parameter "A"). Select and delete any with amplitude greater than 0.7 or less than 0.1, and if your image contains background galaxies check that no false positives remain. Stars in this brightness range are not saturated and not too faint to give an accurate PSF model.

> **Tip**
>
> If the blind generation of a deconvolution PSF can be done on linear and non-linear data, the use of a PSF from star PSF can only be done on linear images. Otherwise the PSF values would not be valid.

### Blind PSF Estimate

These methods can automatically estimate a PSF based on the image itself. If you have no better prior knowledge of the PSF such as stars in the image (for example, lunar imagery that contains no stars) then this may be your best option. In most cases it is recommended to use the default $_0$ method: it is faster and usually gives better results.

> **Tip**
>
> However you are generating your PSF, check the preview to make sure that it does not look cropped. If it does, increase the PSF size until no significant parts of the PSF are cropped.

### Other PSF Generation Methods

Other PSF generation methods worthy of mention are the manual disc profile and the Airy disc. The disc profile can be used to improve images where the focus is slightly off. Try to match the size of the disc to the size of the out-of-focus blur. The Airy disc can be used to fix the slight blurring caused by the diffraction of the telescope tube itself.

> **Tip**
>
> If you have exceptional seeing (little to no atmospheric blur) deconvolving the image using an Airy disc may be all that you need.

### Deconvolving the Image

Once you've generated a PSF you're happy with, you're ready to deconvolve your image. It is important to use the right settings to get good results.

> **Tip**
>
> Deconvolution is quite slow for large images. To make it quicker to find the best parameters, save your work at this point and use the ROI feature.

### Images with Stars

Images containing stars, especially linear (unstretched) data, should always be deconvolved using the Richardson-Lucy methods. Ignore Split Bregman and Wiener: those algorithms are suited to solar system images.

Deep space images pose 2 challenges with deconvolution: ringing around bright stars, and noise amplification in the background.

To deal with rings around stars, try using the Gradient Descent method and increase the number of iterations gradually until you start to see signs of dark rings forming around stars, then reduce the iterations just a little.

The above animation shows the effect of reducing the numebr of iterations of the multiplicative formulation of Richardson-Lucy: it also demonstrates the finer control that can be achieved by using the gradient descent method, at the cost of more iterations.

To deal with amplification of background noise, you can try applying a little noise reduction before deconvolution. In the Noise Reduction dialog, choose the Anscombe VST secondary denoising algorithm and leave the modulation quite low, try around 50-60%. You just want to take the edge off the noise to allow you to push the number of iterations a little further, not generate a completely smooth image.

### Lunar Images

Typically you may wish to sharpen a lunar image after stacking. Stacked lunar images can be sharpened very nicely using the Split Bregman or Wiener methods. My usual choice is Split Bregman. Try leaving the value of $\alpha$ at the default, and deconvolving the image using a blind estimated $_0$ PSF. An example of this is shown below using a freshly stacked lunar image (i.e. no wavelet processing has been done to it). Despite the limitations of the GIF animation format the sharpening can clearly be seen; it is also clear that the results from Split Bregman and Wiener are very similar.

### Stacked Planetary Images

A typical planetary workflow involves stacking the planetary SER video in a specialist tool such as Autostakkert! or Astrosurface, and then sharpening the resulting image using wavelets and deconvolution. A combination of Siril's A trous Wavelets tool and the Deconvolution tool gives excellent results as shown here. This image of Jupiter was initially sharpened using wavelets with the first layer control set to 75, the second set to 10 and the others all left at the default. A colour PSF was then built from 3 Airy discs calculated for the telescope and sensor used (a 6" Newtonian with a 3x Barlow lens and an ASI462MC sensor with 2.9 micron pixels) composited using the RGB composition tool. This was used to deconvolve the image with 6 iterations of Richardson-Lucy (here I used the multiplicative version). At each step the image becomes sharper.

Raw stack, still blurry.

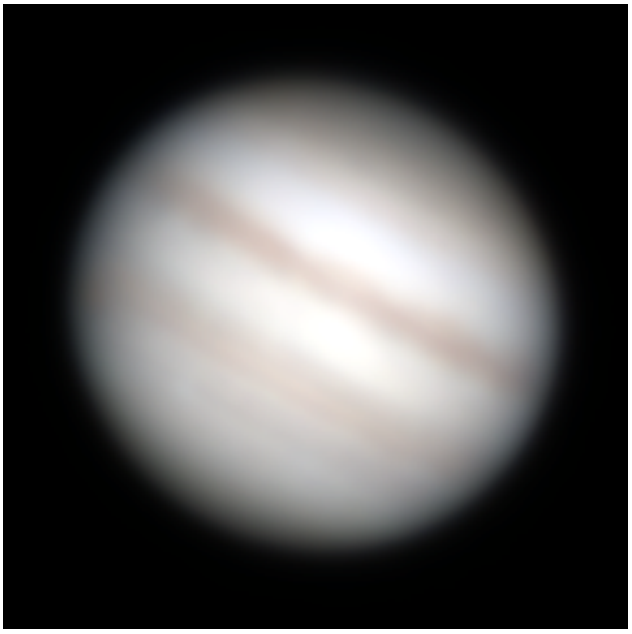
Processed with Siril wavelet decomposition, wavelet layer 1 strength 75, wavelet layer 2 strength 10.

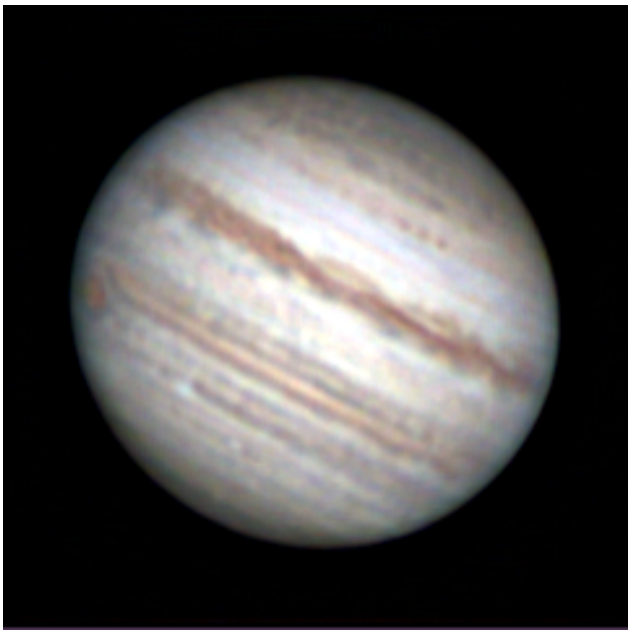
Processed with Siril wavelets as above, and then with 6 iterations of multiplicative Richardson-Lucy deconvolution.

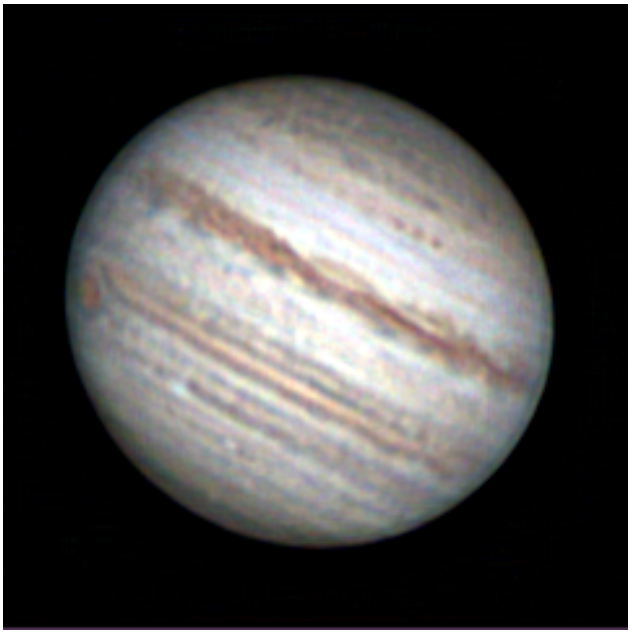
### Unstacked Planetary Sequences

> **Tip**
>
> Warning: this method is extremely slow as it requires individual processing of typically 30,000 (or more) images in a planetary sequence!

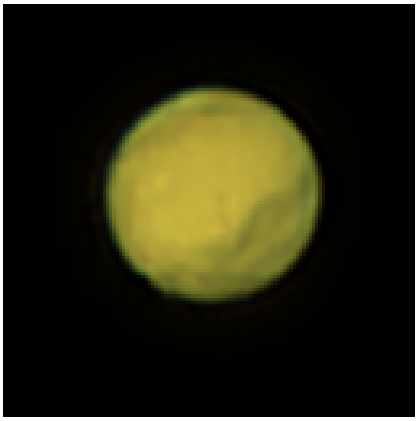
Some users have suggested mitigating telescope diffraction pre-stacking by deconvolving your sequence using an Airy disc PSF. To do this with a typical one-shot colour planetary camera, the sequence has to be set to debayer on load. You







can take this one step further if you wish by generating three separate Airy discs for red, green and blue wavelengths (typically 600nnm, 530nm and 450nm respectively). Siril cannot directly generate a colour PSF (the deconvolution UI is busy enough!) but if you save each of the red, green and blue Airy discs separately you can combine them into a colour PSF using the RGB composition tool. Save this, and if a colour or sequence is loaded the PSF will load in colour and will deconvolve each colour channel using the appropriate PSF.

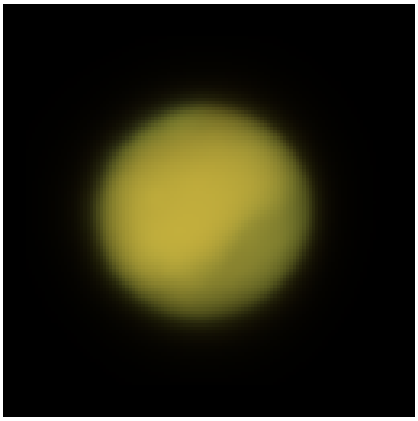


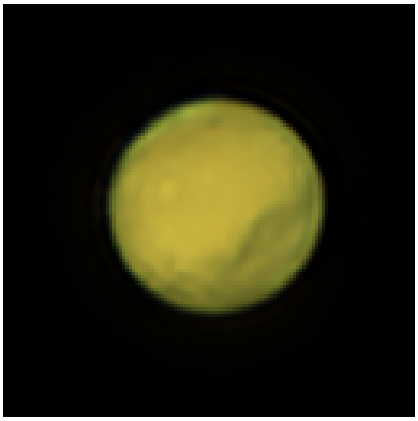
Stacked and sharpened without individually deconvolving frames.

Raw stack: best 30% of 91k frames individually deconvolved using Siril.

Result of sharpening the individually deconvolved stack.

In the image above a slight improvement to the shape of the edge is evident in the version that was frame-by-frame deconvolved with an Airy disc PSF using Siril's Richardson-Lucy method prior to stacking, but care must be taken to avoid loss of details. This process is very slow: my development machine took 4.5 hours to deconvolve each of the 91k frames in this sequence, and the improvement may be minor if any.





### Commands

**Siril command line**

```
makepsf clear
makepsf load filename
makepsf save [filename]
makepsf blind [-l0] [-si] [-multiscale] [-lambda=] [-comp=] [-ks=] [-savepsf=]
makepsf stars [-sym] [-ks=] [-savepsf=]
makepsf manual { -gaussian | -moffat | -disc | -airy } [-fwhm=] [-angle=] [-ratio=] [-
↪beta=] [-dia=] [-fl=] [-wl=] [-pixelsize=] [-obstruct=] [-ks=] [-savepsf=]
```

Generates a PSF for use with deconvolution, any of the three methods exposed by RL, SB or WIENER commands. One of the following must be given as the first argument: **clear** (clears the existing PSF), **load** (loads a PSF from a file), **save** (saves the current PSF), **blind** (blind estimate of tke PSF), **stars** (generates a PSF based on measured stars from the image) or **manual** (generates a PSF manually based on a function and parameters).

No additional arguments are required when using the **clear** argument.

To load a previously saved PSF the **load** argument requires the PSF *filename* as a second argument. This may be in any format that Siril has been compiled with support for, but it must be square and should ideally be odd.

To save a previously generated PSF the argument **save** is used. Optionally, a filename may be provided (this must have one of the extensions ".fit", ".fits", ".fts" or ".tif") but if none is provided the PSF will be named based on the name of the open file or sequence.

For **blind**, the following optional arguments may be provided: **-l0** uses the l0 descent method, **-si** uses the spectral irregularity method, **-multiscale** configures the l0 method to do a multi-scale PSF estimate, **-lambda=** provides the regularization constant.

For PSF from detected **stars** the only optional parameter is **-sym**, which configures the PSF to be symmetric.

For a **manual** PSF, one of **-gaussian**, **-moffat**, **-disc** or **-airy** can be provided to specify the PSF function, Gaussian by default. For Gaussian or Moffat PSFs the optional arguments **-fwhm=**, **-angle=** and **-ratio=** may be provided. For Moffat PSFs the optional argument **-beta=** may also be provided. If these values are omitted, they default to the same values as in the deconvolution dialog. For disc PSFs only the argument **-fwhm=** is required, which for this function is used to set the *diameter* of the PSF. For Airy PSFs the following arguments may be provided: **-dia=** (sets the telescope diameter), **-fl=** (sets the telescope focal length), **-wl=** (sets the wavelength to calculate the Airy diffraction pattern for), **-pixelsize=** (sets the sensor pixel size), **-obstruct=** (sets the central obstruction as a percentage of the overall aperture area). If these parameters are not provided, wavelength will default to 525nm and central obstruction will default to 0%. Siril will attempt to read the others from the open image, but some imaging software may not provide all of them in which case you will get bad results, and note the metadata may not be populated for SER format videos. You will learn from experience which are safe to omit for your particular imaging setup.

For any of the above PSF generation options the optional argument **-ks=** may be provided to set the PSF dimension, and the optional argument **-savepsf=filename** may be used to save the generated PSF: a filename must be provided and the same filename extension requirements apply as for **makepsf save filename**

Links: *psf*, *rl*, *sb*, *wiener*

**Siril command line**

```
rl [-loadpsf=] [-alpha=] [-iters=] [-stop=] [-gdstep=] [-tv] [-fh] [-mul]
```

Restores an image using the Richardson-Lucy method.

Optionally, a PSF may be loaded using the argument **-loadpsf=filename** (created with MAKEPSF).

The number of iterations is provide by **-iters** (the default is 10).

The type of regularization can be set with **-tv** for Total Variation, or **-fh** for the Frobenius norm of the Hessian matrix (the default is none) and **-alpha=** provides the regularization strength (lower value = more regularization, default = 3000).

By default the gradient descent method is used with a default step size of 0.0005, however the multiplicative method may be specified with **-mul**.

The stopping criterion may be activated by specifying a stopping limit with **-stop=**

Links: *psf*, *makepsf*

**Siril command line**

```
sb [-loadpsf=] [-alpha=] [-iters=]
```

Restores an image using the Split Bregman method.

Optionally, a PSF may be loaded using the argument **-loadpsf=filename**.

The number of iterations is provide by **-iters** (the default is 1).

The regularization factor **-alpha=** provides the regularization strength (lower value = more regularization, default = 3000)

Links: *psf*

**Siril command line**

```
wiener [-loadpsf=] [-alpha=]
```

Restores an image using the Wiener deconvolution method.

Optionally, a PSF created by MAKEPSF may be loaded using the argument **-loadpsf=filename**.

The parameter **-alpha=** provides the Gaussian noise modelled regularization factor

Links: *psf*, *makepsf*

#### References

### 10.4.6 Edge Preserving Filters

Siril offers two edge preserving filters: OpenCV's implementation of the Bilateral Filter, and a Guided Filter. Both of these filters can be used to reduce noise, and both preserve sharp edges and strong gradients in the image. The bilateral filter acts on a single image whereas the guided filter filters the image using a guide image to identify structures. In the simplest use, the guided filter acts in a self-guided manner by using the input image as the guide image.

The layout of the window dialog is quite simple and few settings are available.

- **Filter type**: Choose between the bilateral filter and the guided filter.
- **Diameter**: This defines the size of the filter kernel that will be used. When using the bilateral filter, setting a diameter of 0 will cause the filter kernel size to be set automatically based on the spatial sigma value. When using the guided filter this value must be set: a diameter of 0 will result in no filtering being done.
- **Sigma (spatial)**: This defines the spatial extent of the filter kernel. A larger value results in smoothing of noise

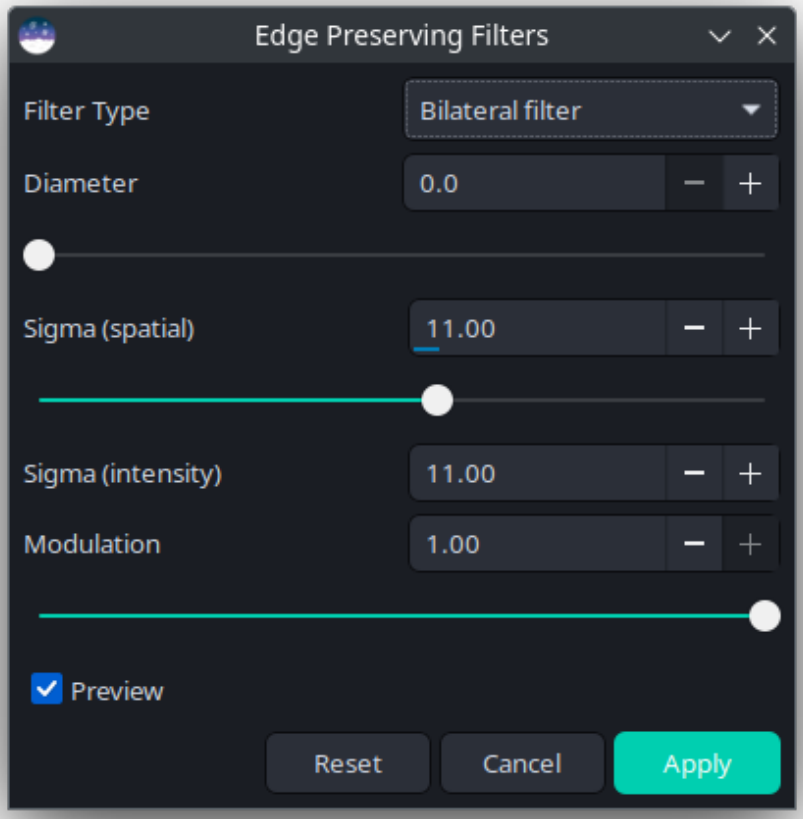



Fig. 48: Median dialog window.

over a greater image area, but is slower to compute. A smaller value results in more local smoothing and is faster to compute. Defaults are not set for the sigma values as the appropriate value may depend significantly on image characteristics, but it can be good to start with both sigma (spatial) and sigma (intensity) set to about 11.

- **Sigma (intensity)**: This defines the range of intensity variation to which the filter responds. A high value results in stronger filtering of noise but may result in loss of genuine detail; a low value reduces the filtering of noise, but will avoid loss of details with gentler gradients.
- **Guide image**: This allows selection of an image to use as a guide image when performing a guided filter. If the "self guided" check box is checked, the filter acts in self-guided mode using the input image as the guide image.
- **Modulation**: In Siril, modulation is a parameter between 0 and 1 mixing the original and processed images. A value of 1 keeps only the processed image, a value of 0 does not apply any edge-preserving filter at all.

Fig. 49: Showing the effect of changing the bilateral filter parameters. Note that the exact values required will depend on the noise characteristics of your data.

**Tip**

The parameters to the two different filter types do not behave identically, so when changing between bilateral and guided filters with sigma parameters set, you should expect to see some change in the preview result. The code applies a bit of compensation to the parameters provided to minimize the difference in behaviour of the two filters for the same input parameters but this is not exact (nor is it intended to be).

**Siril command line**

```
epf [-guided] [-d=] [-si=] [-ss=] [-mod=] [-guideimage=]
```

Applies an edge preserving filter. By default a bilateral filter is applied; a guided filter can be specified using the argument **-guided**. The filter diameter defaults to 3 and can be set using **-d=**. Be careful with values of d greater than 20 as the algorithm can be computationally expensive.

The intensity filtering sigma value can be set using **-si=** and the spatial sigma value can be set using **-ss=**. Sigma values represent the difference in pixel values over which the filter acts strongly: for 32-bit images the value should be between 0 and 1.0, whereas for 16-bit images it should be between 0 and 65535. The defaults if not specified are for both to be set to 11. If **-d=0** is set then the filter diameter will be set automatically based on the value of **-ss**. *Note that when applying a guided filter, only* **-sc** *applies.*

When specifying a guided filter, a guide image may be set using **-guideimage=**. The default if no guide image is specified is to perform a self-guided filter. *Note: the guide image must have the same dimensions as the image to be filtered!*

The strength of the filter can be modulated using the **-mod=** argument. If mod = 1.0 the full effect of the filter will be applied; for mod less than 1.0 a proportion of the original image will be mixed with the result, and for mod = 0.0 no filtering will be applied

### 10.4.7 Fourier Transform

A Fourier transform (FT) is a mathematical transform that decomposes functions into frequency components, which are represented by the output of the transform as a function of frequency. This transformation is widely used in imaging because it allows to see signals at regular frequencies.

**Theory**

#### Fourier transform

The Fourier transform is an analysis process, decomposing a complex-valued function $f(x)$ into its constituent frequencies and their amplitudes:

$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x}\, dx.$

#### Inverse transform

The inverse process is synthesis, which recreates $f(x)$ from its transform:

$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x}\, d\xi.$

Siril allows to transform an image in the frequency space thanks to a Fast Fourier Transform algorithm. The result is in the form of two images. The first one, automatically loaded, contains the magnitude (or modulus) of the transform, the second one contains the phase. The location of the two images must be entered in the **Direct Transform** tab (see illustration below) of the dialog. It is then possible to modify the modulus image by removing frequency peaks corresponding to unwanted signals. It is important not to forget to save the changes.

The *Centered* option, when checked, centers the origin of the Direct Fourier Transform. If not, the origin is at the top-left corner.

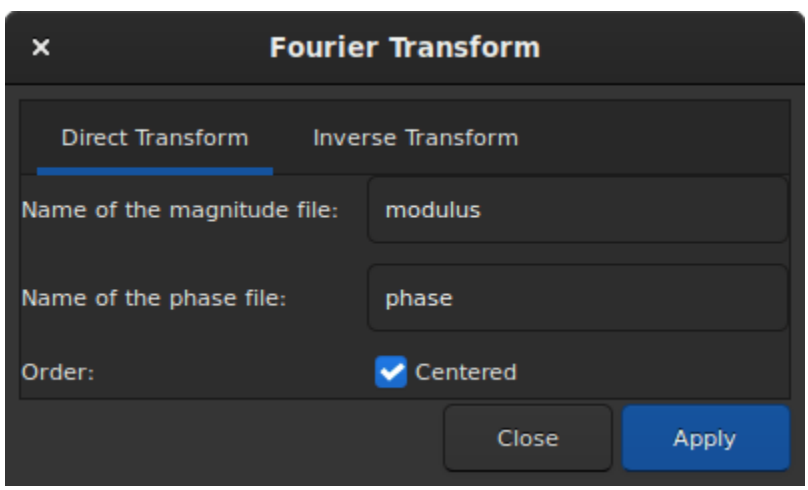



Fig. 50: Direct Transform tab.

To reconstruct the image, click on the **Inverse Transform** tab and enter the filepath of the modulus and phase images.

**Siril command line**

```
fftd modulus phase
```

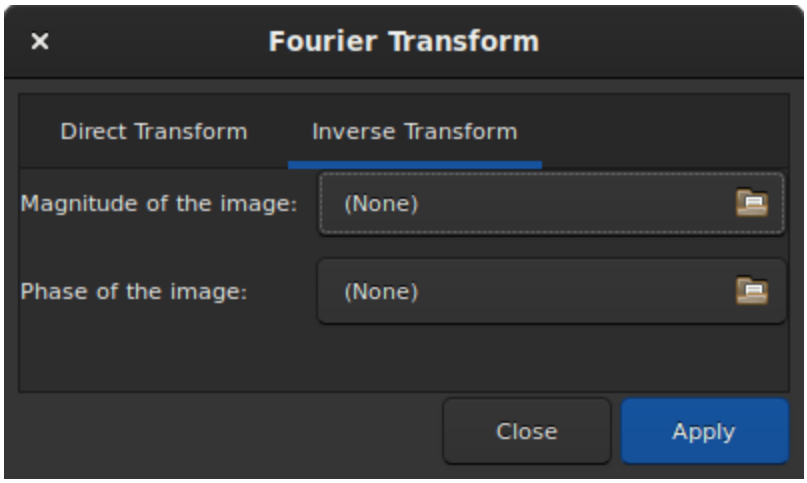



Fig. 51: Inverse Transform tab.

Applies a Fast Fourier Transform to the loaded image. **modulus** and **phase** given in argument are the names of the saved in FITS files

**Siril command line**

```
ffti modulus phase
```

Retrieves corrected image applying an inverse transformation. The **modulus** and **phase** arguments are the input file names, the result will be the new loaded image

### 10.4.8 Median Filter

The median represents the middle data point that half of data is smaller and half of data larger than this point. This is a robust estimator to remove outliers from a data set. Consequently, this tool can be useful as a naive denoiser, effective against *impulse noise*.

The layout of the window dialog is quite simple and few settings are available.

- **Kernel size**: From $3 \times 3$ to $15 \times 15$, this defines the size of a squared kernel that is used to apply the filter. The larger the kernel, the more blurred the result will be.
- **Iterations**: This defines the number of passes of the kernel.
- **Modulation**: In Siril, modulation is a parameter between 0 and 1 mixing the original and processed images. A value of 1 keeps only the processed image, a value of 0 does not apply any median filter at all.

**Siril command line**

```
fmedian ksize modulation
```

Performs a median filter of size **ksize** x **ksize** (**ksize** MUST be odd) to the loaded image with a modulation

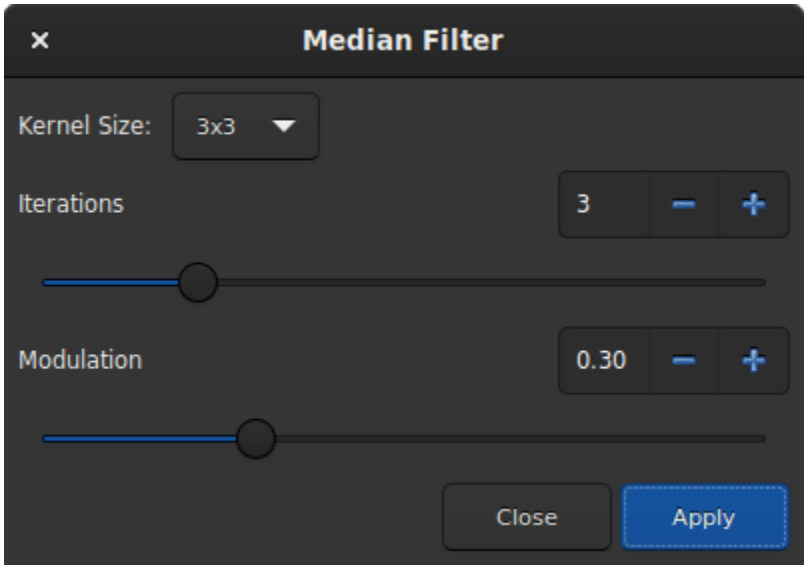



Fig. 52: Median dialog window.

parameter **modulation**.

The output pixel is computed as : out=mod x m + (1  mod) x in, where m is the median-filtered pixel value. A modulation's value of 1 will apply no modulation

### 10.4.9 Noise Reduction

#### Image Noise

Images suffer from various types of noise:

1. Impulse noise
   - This type of noise (sometimes called "salt and pepper noise") typically arises from hot or cold pixels. It is usually dealt with by using sigma rejection stacking, but sometimes you may need to deal with it if processing a single unstacked image.
2. Additive White Gaussian Noise
   - This type of noise is typical of well-illuminated photographs: it arises from the thermal and electronic fluctuations of the acquisition device, and the noise level is independent of the signal. AWGN can be reduced at capture time by using cooled cameras, and it is reduced in stacking because stacking $n$ images increases the correlated signal by a factor of $n$ whereas the uncorrelated noise only increases by a factor of $\sqrt{n}$. It is also the type of noise that most classical denoising algorithms are designed to remove.
3. Poisson Noise
   - When dealing with photon-starved images, the character of the noise ceases to be primarily Gaussian and the probabilistic nature of photon counting becomes significant or even dominant. This is modelled by a Poisson distribution and this type of noise is signal dependent.

### Noise Reduction in Siril

Siril provides well-studied state-of-the-art classical denoising algorithms. The criteria for choosing algorithms were:

- The algorithm should be analysed in peer reviewed academic journals, with a description of the algorithm and an objective quantitative comparison of its performance.
- The authors should have made available a F/OSS implementation. This is important to avoid IP issues and, where the reference implementations have been used directly, to ensure licence compatibility.
- The algorithms should perform at a reasonable speed.
- Finally, the implementation of the algorithm must be capable of processing 32 bit floating point pixel data.

Neural network denoising technology was investigated, but discounted at the current time on the grounds of development complexity. The denoising performance of neural networks can typically beat classical approaches by up to a dB peak signal-to-noise ratio, but performance is highly dependent on the neural network being trained on data representative of the real live data.

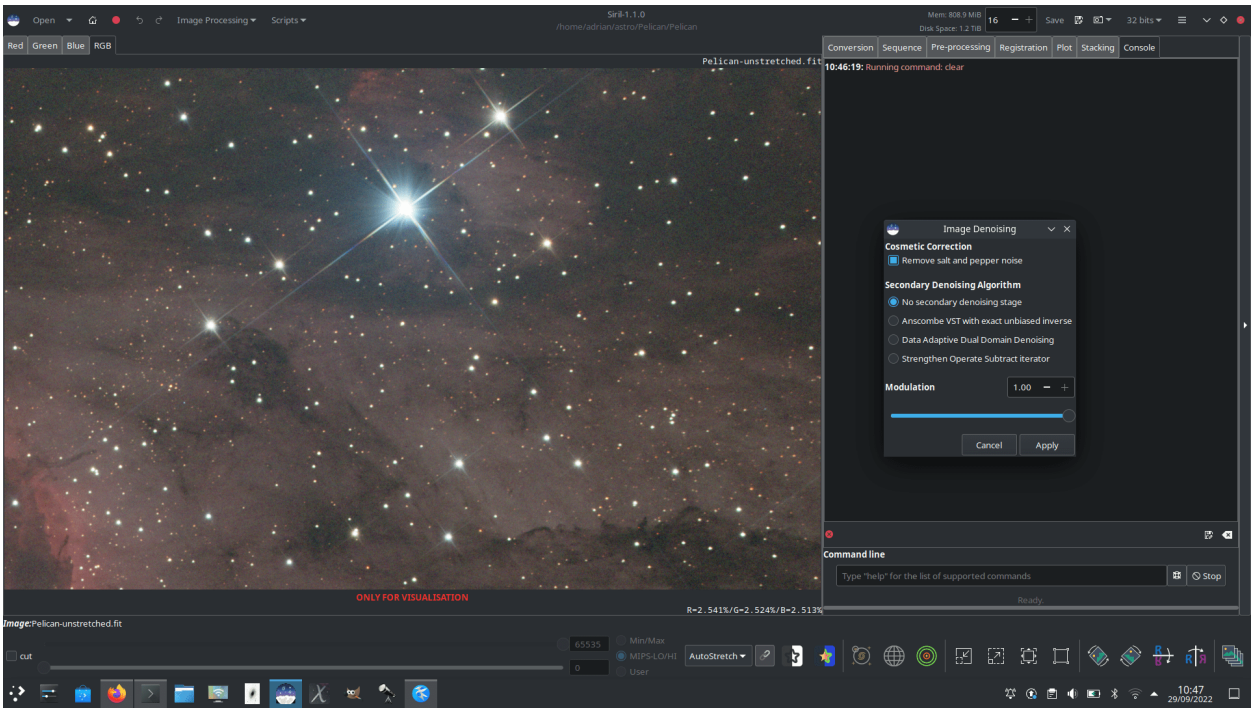



Fig. 53: Noise Reduction dialog

### Algorithms: Impulse Noise

Siril primarily removes impulse noise through sigma rejection stacking. If you use this stacking method, you shouldn't have any issues with impulse noise. However if you are working on a single exposure you may well find impulse noise in your image. This should be dealt with using Siril's **Cosmetic Correction** function before any other noise reduction is used, as the presence of impulse noise can skew AWGN denoising algorithms and create artefacts. It works in a similar way to sigma rejection, but on neighbouring pixels. Any pixel whose intensity is more than $n$ standard deviations away from its neighbours will be rejected and replaced by a value based on the median of the neighbours. In the Denoising tool Cosmetic Correction is active by default and will take place before any additional denoising steps. (Even if impulse noise removal has already been carried out, leaving the setting on does no harm.) Alternatively, Cosmetic correction can be applied manually using the *Cosmetic Correction* tool in the *Image Processing* menu.

### Algorithms: Additive White Gaussian Noise

The main AWGN noise reduction algorithm used in Siril is Non-Local Bayesian (NL-Bayes) denoising [Lebrun2013].

- Non-local denoising algorithms represented a major improvement over previous pixel-centred linear filters. NL-Bayes is an improved version of the earlier non-local denoising algorithms and offers one of the best classical AWGN denoising algorithms. It is marginally better than the modern "benchmark" algorithm Block Matching and 3D tranform (BM3D) noise reduction and much faster to execute.
- The key parameter required to optimise the performance of AWGN algorithms is sigma, the standard deviation of the noise. Siril measures the noise level directly from the image data and passes this to the NL-Bayes algorithm, therefore in the Siril denoising tool there are no configurable inputs to NL-Bayes.

Siril complements NL-Bayes with a number of other noise reduction algorithms:

- Data-Adaptive Dual Domain Denoising (DA3D) [Pierazzo2017]
  - This takes the output of NL-Bayes and uses it as a guide image. This guide image is used to reprocess the original image by performing frequency domain shrinkage on shape and data-adaptive patches. It slightly improves the performance of NL-Bayes at some additional computational cost. The shape and data-adaptive patches are dynamically selected, thus concentrating the computations on the areas with greatest image detail. It can also help to reduce staircase artefacts present in the guide image.
  - In the Siril denoising tool, DA3D is a simple toggle with no optional settings.
- Strengthen, Operate, Subtract iteration (SOS) [Romano2015]
  - SOS works by iterating the primary denoising algorithm several times. At each iteration the image is "strengthened" by mixing in a proportion of the original noisy image. The NL-Bayes algorithm runs on this strengthened image, after which the previous estimation is subtracted.
  - The image x at an iteration $k+1$ is given by $x_{k+1} = f(y + x_k) - x_k$ where $y$ is the noisy input image.
  - In the Siril denoising tool, SOS is a toggle with two parameters: the number of `iterations` can be set, and the proportion of the noisy image mixed in at each iteration (`rho`) can be set. Avoid setting `rho` too high as it can result in issues with SOS converging: the default values (`3 iterations` and `rho = 0.2`) are usually fine.

### Algorithms: Poisson and Poisson-Gaussian Noise

- Anscombe Variance-Stabilising Transform [Mäkitalo2011], [Mäkitalo2012]
  - Variance stabilising transforms are used for images with Poisson or Poisson-Gaussian noise to minimise the signal dependence of the noise and make it look more like AWGN, which NL-Bayes is good at removing, and then an inverse transform is applied on completion. The transform chosen for use in Siril is the Anscombe transform $A : x \rightarrow 2 \times \sqrt{\left(x + \frac{3}{8}\right)}$
  - As the transform is non-linear, use of the direct algebraic inverse results would bias the output. Siril therefore uses a closed-form approximation to the exact unbiased inverse, which is quick to calculate and produces a substantial improvement over other forms of inverse such as the asymptotic inverse.
  - In the Siril denoising tool, the Anscombe VST is a simple toggle with no optional settings.

Note that only one of the above mentioned complementary denoising algorithms can be chosen.

The animation below shows what is possible using variance stabilisation with a photon-starved image, in this case a single 5 minute red filter sub of the Pelican nebula, shown with the *AutoStretch* screen transfer function. Note the lack of blurring, bloating or loss of detail around stars and the sharp edge of the nebula in the bottom left part of the image compared with what might be obtained through more basic noise reduction schemes. Once stretched more sympathetically and combined with other channels this would greatly improve the quality achievable from very limited data (though more data is always the better solution!)

Fig. 54: Denoising a photon-starved image

### Modulation

In Siril, modulation is a parameter between 0 and 1 mixing the original and denoised images. A value of 1 keeps only the denoised image, a value of 0 does not apply any denoising at all. Modulation obviously reduces denoising performance, but in some instances if denoising has left flat areas of the image looking a little too smooth, you can use some modulation to restore the appearance of microtexture in these regions.

### When to run Noise Reduction

The noise reduction algorithms are designed to remove AWGN and should therefore perform best on unstretched images: if white noise has a non-linear stretch applied, its characteristics change and it is no longer white. Performing noise reduction on stretched images can still be done and will result in an improvement, but potentially will not be as effective as if applied at the linear stage.

### Noise Reduction Interface

The Siril Noise Reduction tool can be accessed in two ways: via the GUI, or via the command interface. The GUI is shown below. Note: the SOS advanced options are hidden if SOS is not selected.

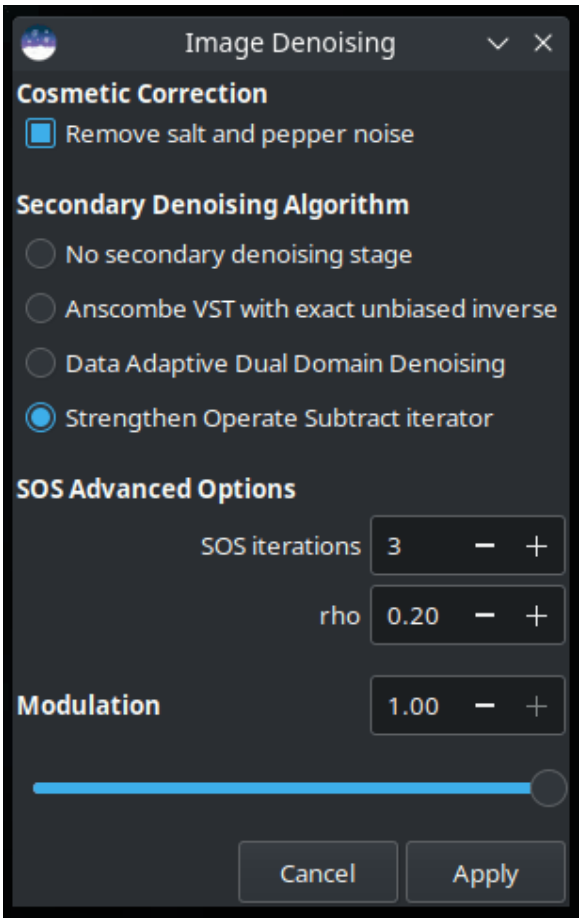



Fig. 55: Siril Noise Reduction GUI

Noise reduction can also be applied using Siril commands, either in the console or in scripts. The format is:

**Siril command line**

```
denoise [-nocosmetic] [-mod=m] [ -vst | -da3d | -sos=n [-rho=r] ] [-indep]
```

Denoises the image using the non-local Bayesian algorithm described by Lebrun, Buades and Morel.

It is strongly recommended to apply cosmetic correction to remove salt and pepper noise before running denoise, and by default this command will apply cosmetic correction automatically. However, if this has already been carried out earlier in the workflow it may be disabled here using the optional command **-nocosmetic**.

An optional argument **-mod=m** may be given, where 0 <= m <= 1. The output pixel is computed as : *out=m x d + (1 m) x in*, where *d* is the denoised pixel value. A modulation value of 1 will apply no modulation. If the parameter is omitted, it defaults to 1.

The optional argument **-vst** can be used to apply the generalised Anscombe variance stabilising transform prior to NL-Bayes. This is useful with photon-starved images such as single subs, where the noise follows a Poisson or Poisson-Gaussian distribution rather than being primarily Gaussian. It cannot be used in conjunction with DA3D or SOS, and for denoising stacked images it is usually not beneficial.

The optional argument **-da3d** can be used to enable Data-Adaptive Dual Domain Denoising (DA3D) as a final stage denoising algorithm. This uses the output of BM3D as a guide image to refine the denoising. It improves detail and reduces staircasing artefacts.

The optional argument **-sos=n** can be used to enable Strengthen-Operate-Subtract (SOS) iterative denoise boosting, with the number of iterations specified by n. In particular, this booster may produce better results if the un-boosted NL-Bayes algorithm produces artefacts in background areas. If both -da3d and -sos=n are specified, the last to be specified will apply.

The optional argument **-rho=r** may be specified, where 0 < r < 1. This is used by the SOS booster to determine the amount of noisy image added in to the intermediate result between each iteration. If -sos=n is not specified then the parameter is ignored.

The default is not to apply DA3D or SOS, as the improvement in denoising is usually relatively small and these techniques requires additional processing time.

In very rare cases, blocky coloured artefacts may be found in the output when denoising colour images. The optional argument **-indep** can be used to prevent this by denoising each channel separately. This is slower but will eliminate artefacts

### Comparison

The images below provide a simplistic comparison of the different algorithms. Note that only one image is used: in practice, different algorithms will be better suited for use on different images. All the images can be clicked on to view at 100% zoom.

### Original noisy image

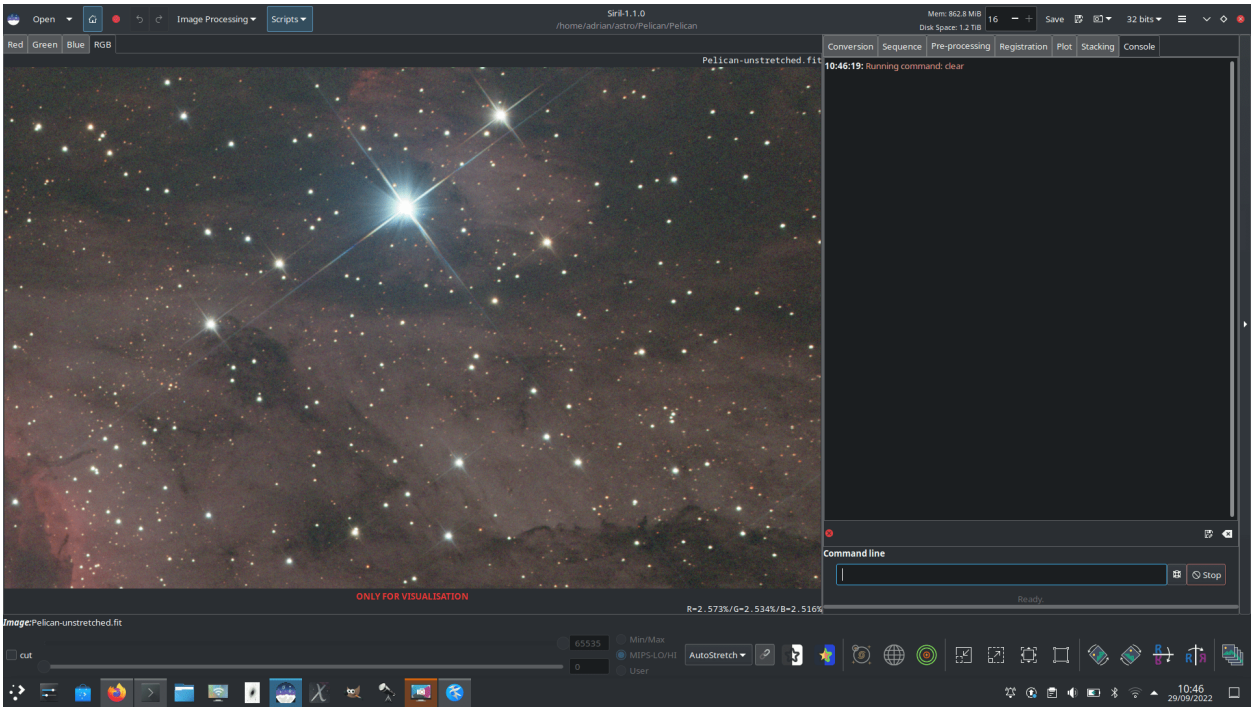



Fig. 56: Noisy image

### Denoised with NL-Bayes only

### Denoised with NL-Bayes only, with 75% modulation to restore some microtexture

### Denoised with NL-Bayes using the Anscombe transform

### Denoised with DA3D using a NL-Bayes guide image

### Denoised with NL-Bayes and SOS

### Limitations

The main limitation is that the algorithms work best when the noise is Gaussian in character (or can be made approximately Gaussian using the VST). There are some reasons why this might not be true:

- If the image has already been heavily processed, for example with deconvolution or wavelet sharpening, the character of the noise will not generally be Gaussian any more. If both noise reduction and deconvolution form part of your workflow, noise reduction should be done first.
- OSC images may denoise less well than mono or composited colour images. A small reduction of luminance AWGN is achieved but as a result of the deBayering process the character of the noise is changed so that it is no longer well modelled as AWGN, and is not removed very effectively. Additionally, for both OSC and composited mono colour images, chrominance noise tends not to be well modelled as AWGN and requires different treatment. At present chrominance noise is best tackled in general purpose image manipulation software such as The GIMP.

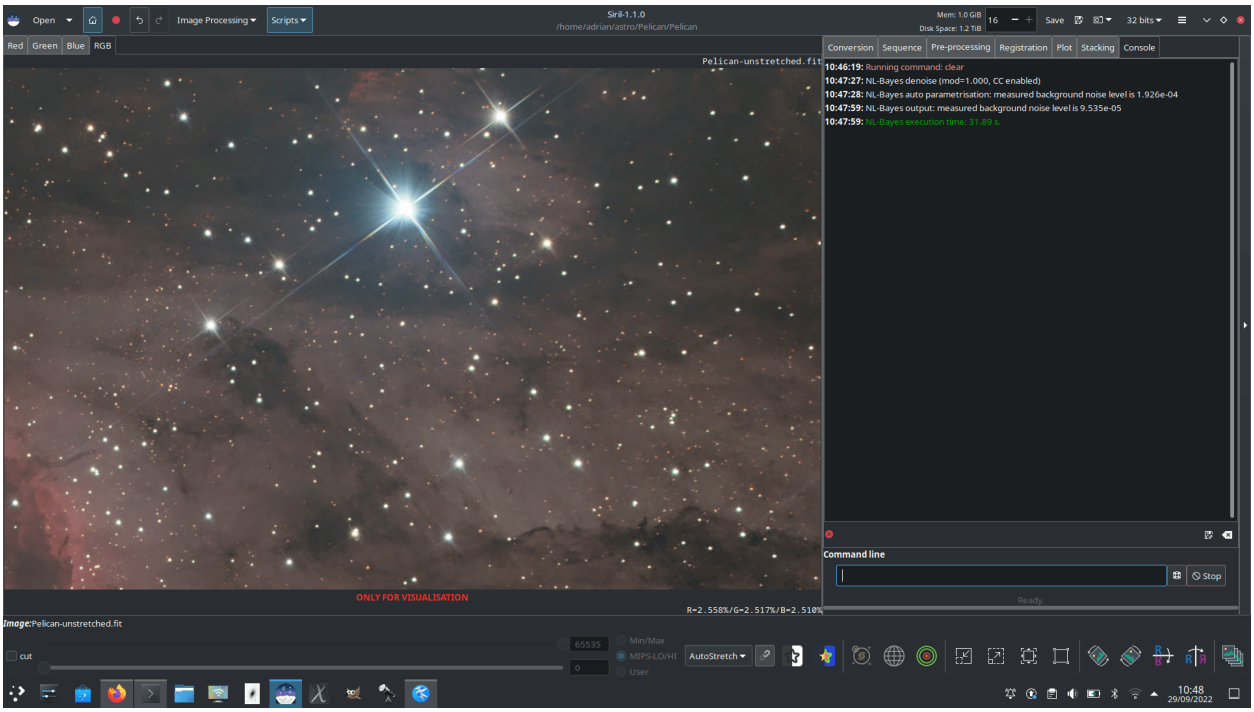



Fig. 57: Denoised with NL-Bayes only

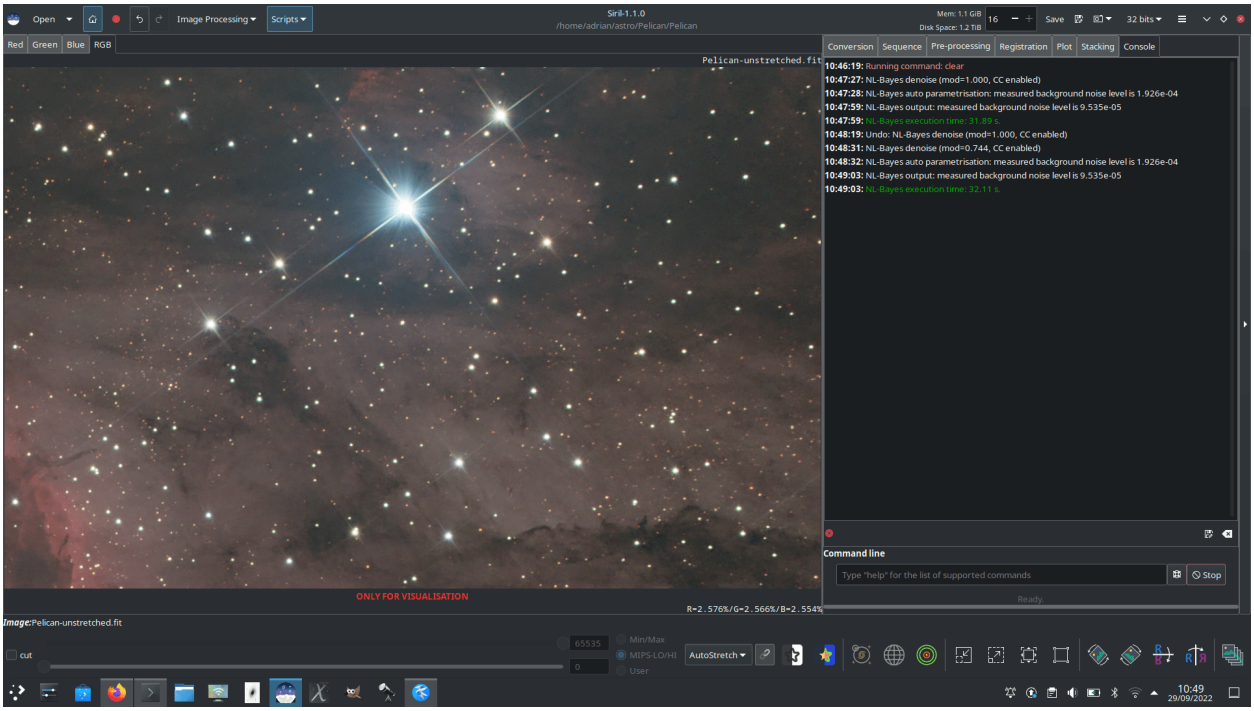



Fig. 58: Use of modulation

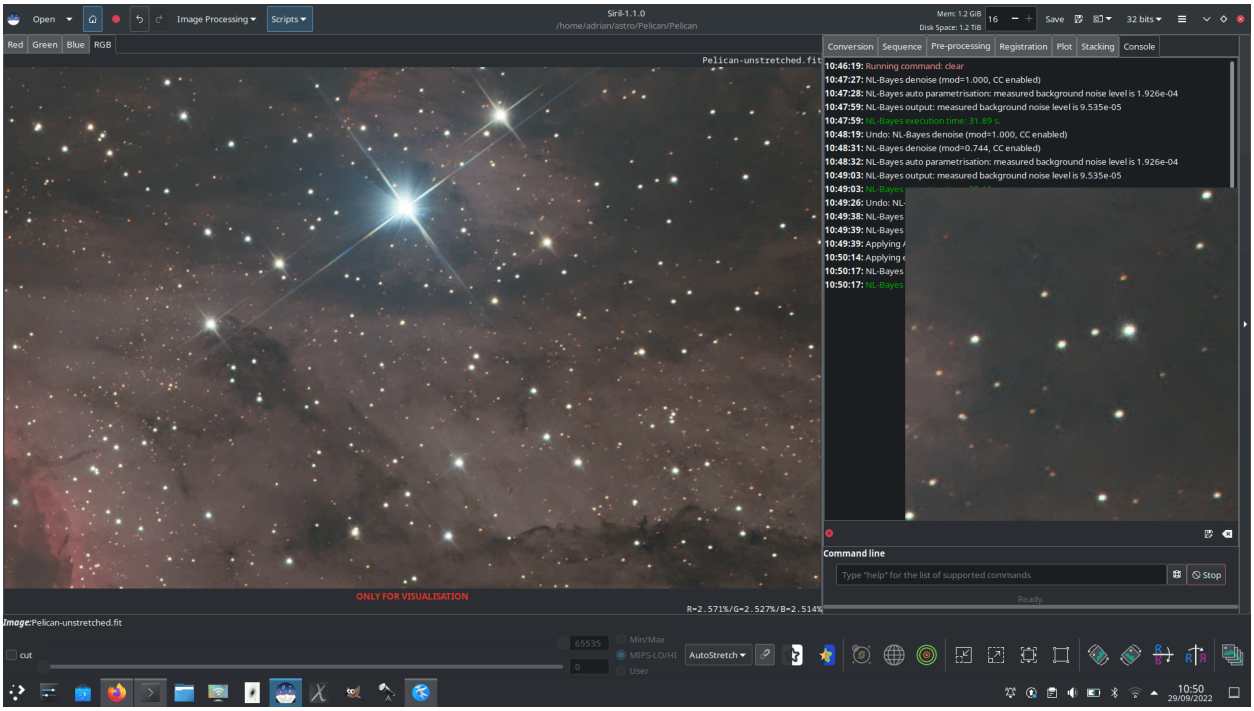



Fig. 59: Denoised with NL-Bayes, variance stabilised with Anscombe transform. A 200% uninterpolated zoom is shown to the right.

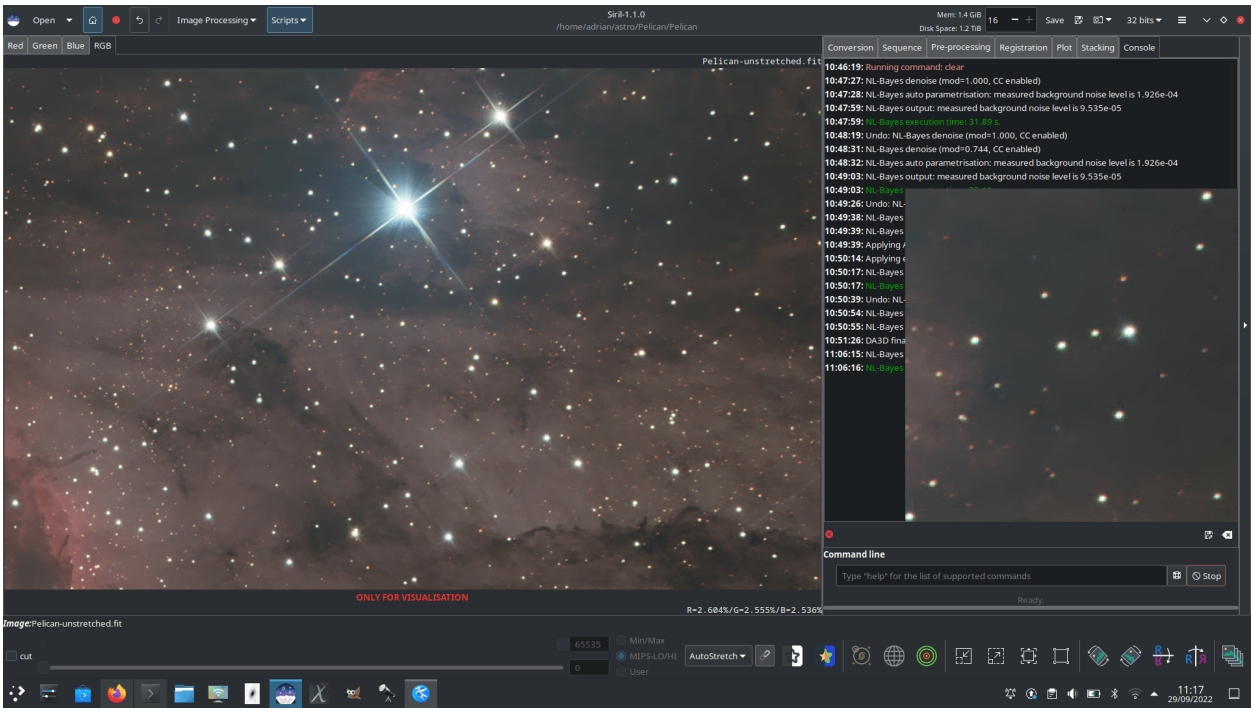



Fig. 60: Denoised with DA3D, guide image prepared using NL-Bayes. A 200% uninterpolated zoom is shown to the right.

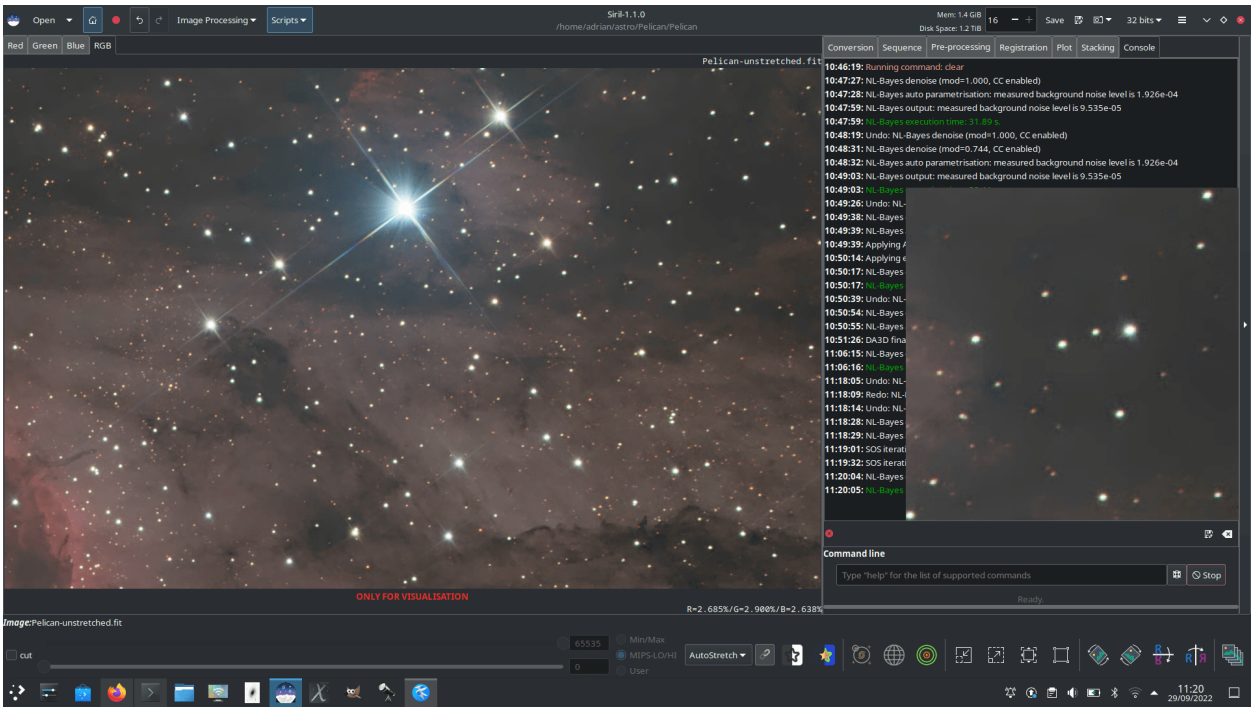



Fig. 61: Denoised with NL-Bayes and SOS iterations. A 200% uninterpolated zoom is shown to the right.

### References

## 10.4.10 Rotational Gradient (Larson Sekanina Filter)

The rotational gradient, also called Larson Sekanina filter, is a filter that allows to remove circular structures from an image, to better highlight other details. This technique is particularly effective to show the jets coming out of the nucleus of a comet.

The principle is quite simple: this image processing consists in subtracting two copies of the image from each other, one of the two copies having been previously rotated with respect to a point defined in the image.

- If there are circular structures around this point they are not modified by rotation and will disappear after rotation.
- If there are non-circular structures, like jets in the coma, they will be shifted in relation to each other between the two copies and the subtraction will amplify the contrast of this structure in the result.
- If the comet moves in the image, it is possible to add a radial shift.

**Theory**

Starting from an input image, the filter generates two images, each with a radial shift ($dr$ in pixels) and a rotational shift ($d\alpha$ in degrees) relative to the point $(x_c, y_c)$. These shifts have the same magnitude but opposite signs between the two images. The two images are then combined to produce the final image. In polar coordinates $(r, a)$ with respect to the point $(x, y)$ we have:

$$B'(\alpha, r, d\alpha, dr) = 2 \cdot B(\alpha, r) - B(\alpha - d\alpha, r - dr) - B(\alpha + d\alpha, r - dr)$$

with $B$: the starting image, $B'$: the resulting image, $dr$: the radial shift and $d\alpha$: the rotational shift

In the example below, concerning the comet C/2022 E3, the alignment was made on the comet and the stars show

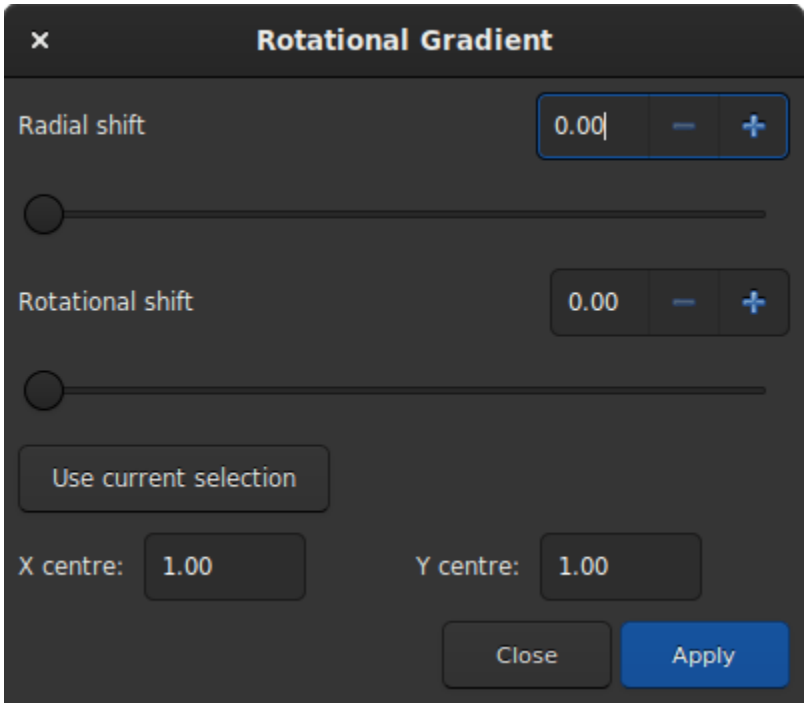



Fig. 62: Dialog box of the Rotational Gradient filter.

important trails. The coma is very circular and it is difficult to see details about its activity. Therefore, it is not necessary to define a radial shift. For the rotation, an angle of 15° was chosen (this choice was made after several attempts and using the undo button to go back). To choose the coordinates of the center of rotation, just make a selection around the cometary nucleus and click on *Use current selection*. This action will copy the coordinates of the centroid to the desired location.

A simple click on *Apply* will apply the filter. In our example, a jet becomes visible.

**Siril command line**

```
rgradient xc yc dR dalpha
```

Creates two images, with a radial shift (**dR** in pixels) and a rotational shift (**dalpha** in degrees) with respect to the point (**xc**, **yc**).

Between these two images, the shifts have the same amplitude, but an opposite sign. The two images are then added to create the final image. This process is also called Larson Sekanina filter

### 10.4.11 Unpurple Filter

This is a cosmetic filter designed to reduce the effect of purple fringing around stars that may be visible when using an achromat refractor especially when broadband imaging. It operates on stretched colour images.

Fig. 63: Image of a comet (C/2022 E3 (ZTF)) showing a coma and a tail (courtesy of Stéphane Garro).

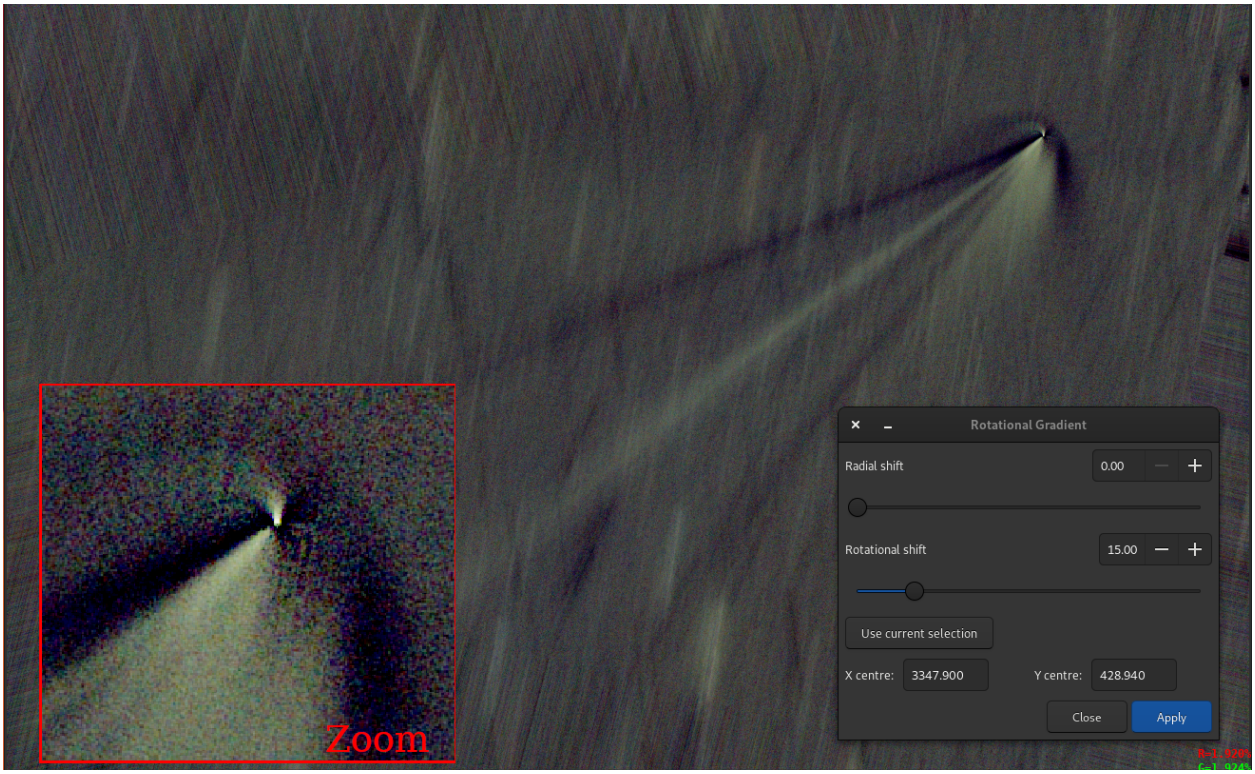



Fig. 64: After applying the filter, we notice a jet emerging from the core.

> **Tip**
>
> This filter is designed to work on non-linear images. It is therefore imperative to use it once the histogram has been stretched.

The filter offers two modes of operation where its effects can be global or limited to the stars only.

It reduces purple fringing by applying a luminance mask to the blue channel based on the green channel, and by allowing you to reduce the blue channel intensity. When in starmask mode, it limits the effects to areas around identified stars.

### Global

This mode is useful when working with a starmask or with images containing just stars, e.g. a globular cluster. This is not so useful for nebula or galaxy images where the colour might be affected.

To use, you should reduce the *Blue Adjustment* slider until the effects are optimal. This is generally around 0.15.

You may also increase the *Threshold* slider to restrict the effects to those pixels where the luminance is above this value. In this mode, the optimal value will be quite low.

### Starmask

This mode is useful for combined images where you want to restrict the effects of this filter to just the stars.

To use, you should tick *Use Starmask*. If you've not already defined a starmask using the Dynamic PSF function, there will be a delay of a few seconds whilst the starmask is generated. You may now temporarily reduce the *Blue Adjustment* to zero. Your stars should now be yellow. If you still see purple fringing, you should increase the Threshold slider until the fringing is entirely covered by yellow. Now you can increase the *Blue Adjustment* slider until your stars are not yellow but your purple fringing is reduced or eliminated, typically around 0.12-0.15.

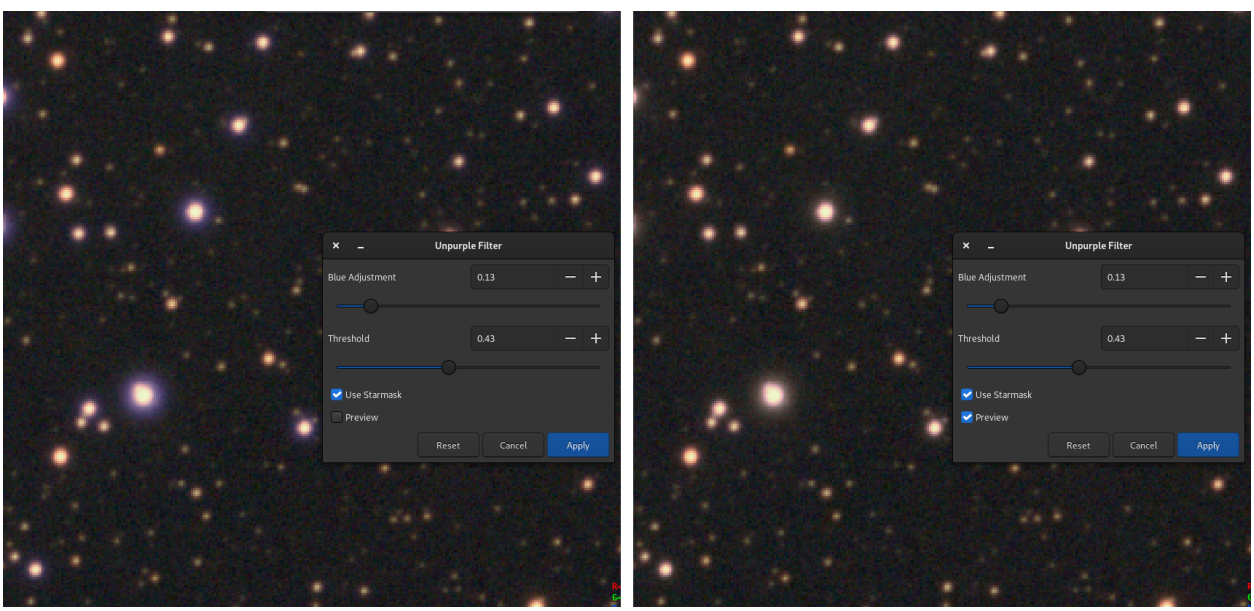



Fig. 65: Comparison before/after application of the unpurple filter. We can see very clearly that the stars have a better color, without the purple ring.

**Siril command line**

```
unpurple [-starmask] [-blue=value] [-thresh=value]
```

Applies a cosmetic filter to reduce effects of purple fringing on stars.

If the **-starmask** parameter is given, a star mask will be used to identify areas of the image to affect. If a Dynamic PSF has already been run, this will be used for the starmask, otherwise one will be created automatically. The **-mod=** parameter should be given a value somewhere around 0.14 to reduce the amount of purple. The **-thresh=** will specify the size modifier for each star in the starmask and should be large enough to cause the stars to be entirely processed without remaining purple fringing. The value should between 0 and 1, typically around 0.5.

If the **-starmask** parameter is not given, the purple reduction will be applied across the entire image for any purple pixels with a luminance value higher than the given **-thresh=**. In this case, the **-thresh=** value should be reasonably low. This mode is useful for starmasks or other images without nebula or galaxy

Links: *psf*

## 10.5 Star Processing

Stars are an integral part of deep sky images and play a crucial role in bringing out the beauty and detail of celestial objects. They often appear as brilliant dots of light, showcasing their brightness and colors, making deep sky images truly captivating. However, due to the limitations of observing conditions, the stars in these images may appear larger and over-exposed. To combat this, astronomers use advanced image processing techniques to separately process the stars and control their size and brightness in the final image.

This part of the documentation is then dedicated to everything related to the stars processing.

### 10.5.1 StarNet Star Removal

**Warning**

The Starnet interface built into Siril itself is now deprecated. StarNet 2.5 has been released and is supported by means of a Python script which is available in the scripts repository. The legacy C interface will **not** support Starnet 2.5 but will remain available in Siril 1.4.x for users of older versions of Starnet.

**In Siril 1.6 the legacy C interface will be removed** so all users will need to upgrade to Starnet 2.5 and use the python script. The script does provide CLI args so where scripts use the Siril `starnet` command they will need to adapt to using `pyscript StarNet.py` instead.

StarNet is a software developed by Nikita Misiura. Its first version was released under a free and opensource license. Unfortunately, version 2 became proprietary and the sources are since closed. The version 2 is available free of charge from there. Make sure you download the **Command Line Tool** version. Siril can interface with any version of the StarNet CLI tool, including the new experimental Torch-based version that has initially been released for M1- and M2-based Apple Macs.

**Warning**

If you are wondering **why StarNet doesn't launch**, please run it outside Siril first. It's not Siril's fault if it's not supported by your computer or badly installed for some reason. If your processor does not support the vectorization

instructions required by StarNet, there is no way to bypass that. The error message will be obtained by executing StarNet alone.

**Tip**

On **MacOS**, for Siril to detect and use StarNet correctly, it is necessary to fix some permissions and security issues first. Start by opening the Terminal application from the Utilities folder within Applications. In Terminal, you need to change your working directory from your home directory to the StarNetCLI installation directory. To do this type in `cd` followed by a `space` and then drag the StarNetCLI folder into the terminal window to copy its path. Press `enter`. Then type in the following four commands, pressing `enter` after each one:

```
xattr -r -d com.apple.quarantine libtensorflow_framework.2.dylib
xattr -r -d com.apple.quarantine starnet++
chmod +x starnet++
chmod +x run_starnet.sh
```

Then, at the first use with Siril, the execution of StarNet may fail with a warning about libtensorflow. Cancel out of this warning. Open System Preferences and under Privacy and Security click the *Allow anyways* button for libtensorflow. After this, StarNet should execute properly in Siril.

**Tip**

On **MacOS**, again, there is a new Starnet executable optimized for the Apple Silicon chip that has been released on the site: https://www.starnetastro.com/experimental/. This new version is much faster than previous version because it uses the new MPS accelerated PyTorch (https://developer.apple.com/metal/pytorch/). Also, this version contains signed binaries, follow the installation instructions in the README.txt

However, it is still possible for Siril to run external binaries and this is what we decided to implement starting with Siril 1.2.0. For the settings, please refer to the preference *page*. It explains how to tell Siril where StarNet is located.

**Warning**

This is the location of the command line version of StarNet that need to be given, not the GUI one.

Note that StarNet requires its input in the form of TIFF images, therefore if Siril is compiled without libtiff support then the StarNet integration will not be available.

The primary purpose of StarNet is to remove all the stars from the images in order to apply a different process between the stars and the rest of the image. This usually helps to control star bloat during the different stretches, but it is also very useful for creating images of comets where the comet tracking rate can be significantly different to the distant stars.

The tool is very easy to use and only five options are available:

- **Pre-stretch linear image**: If selected, an optimised Midtone Transfer Function (MTF) stretch is applied to the image before running StarNet, and the inverse stretch is applied on completion. This is necessary for using StarNet at the linear stage of processing.
- **Recompose stars on completion**: If selected, on completion of the star removal process the star recomposition tool will open, providing an interface for independently stretching and blending the background and the stars if star reduction, rather than total removal, is desired. This option has no effect when processing a sequence.

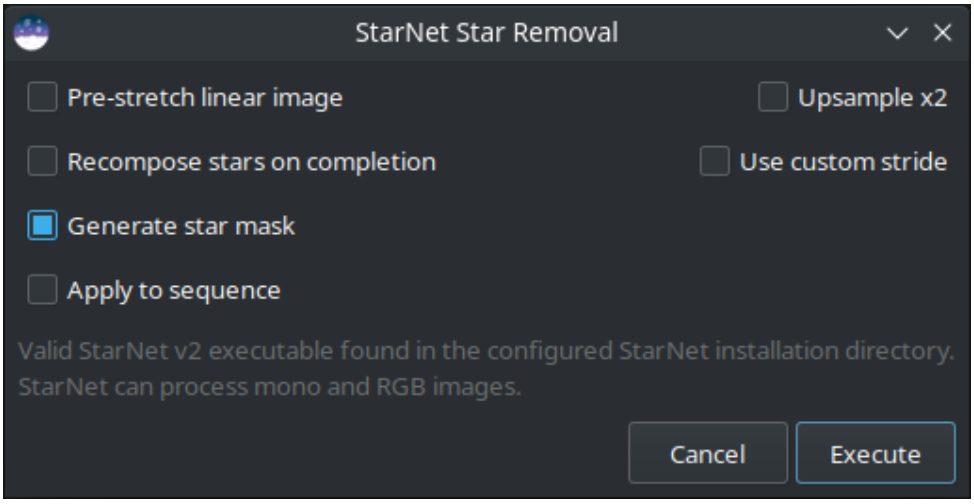



Fig. 66: StarNet dialog box.

- **Generate star mask**: This will generate a star mask and saved it in the working directory. The star mask is calculated as the difference between the original image and the starless image. The default is to produce a star mask.
- **Upsample x2**: This option will up-sample the image by a factor of 2 before running StarNet. This improves performance on very tight stars but quadruples processing time and may impair performance on very large stars. The image is rescaled to the original size on completion.
- **Use custom stride**: A custom value may be entered for the StarNet stride parameter. The default value is 256 and the StarNet developer recommends not to change this.

The StarNet process can easily be applied to a sequence. The togglebutton *Apply to sequence* selects whether the process will apply to a single image or to a sequence. Where the process is applied to a sequence, a new sequence will be created containing the starless images and, if star mask generation is selected, a second sequence will be created containing the corresponding star mask images.

More information about StarNet can be found on the original website.

A click on *Execute* will run the process. It can be slow, depending of your machine performance. However, Siril shows a progressbar to follow the processing. As with other Siril processes, if processing a sequence the progressbar will only update after completion of each image in the sequence, and will show the overall progress through the sequence.

### Commands

**Siril command line**

```
starnet [-stretch] [-upscale] [-stride=value] [-nostarmask]
```

Calls StarNet to remove stars from the loaded image.

**Prerequisite:** StarNet is an external program, with no affiliation with Siril, and must be installed correctly prior the first use of this command, with the path to its CLI version installation correctly set in Preferences / Miscellaneous.

The starless image is loaded on completion, and a star mask image is created in the working directory unless the optional parameter **-nostarmask** is provided.

Optionally, parameters may be passed to the command:
- The option **-stretch** is for use with linear images and will apply a pre-stretch before running StarNet and the inverse stretch to the generated starless and starmask images.
- To improve star removal on images with very tight stars, the parameter **-upscale** may be provided. This will upsample the image by a factor of 2 prior to StarNet processing and rescale it to the original size afterwards, at the expense of more processing time.
- The optional parameter **-stride=value** may be provided, however the author of StarNet *strongly* recommends that the default stride of 256 be used

**Siril command line**

```
seqstarnet sequencename [-stretch] [-upscale] [-stride=value] [-nostarmask]
```

This command calls Starnet++ to remove stars from the sequence **sequencename**. See STARNET

Links: *starnet*

### 10.5.2 Star Recomposition

Star Recomposition is a GUI tool to aid in combining starless and star mask images. It doesn't provide any unique image manipulation that can't be done in other ways, for example using PixelMath and the Generalized Hyperbolic Stretch tool, but it does provide a real-time preview of the combination of two separate images with different stretches applied to each.

There is no command-line equivalent for this tool as it is purely graphical in nature, however starless and star mask images could be combined using the PixelMath *pm* command and GHT-related commands (*ght*, *invght*, *modasinh*, *invmodasinh* and *linstretch*).

The tool is found in the Image Processing menu, in the Star Processing sub-menu.

The dialog is divided into two columns, one for each of the input images.

Each input image is loaded using the respective file chooser. Each column has a stretch histogram preview, which may be minimized to aid use on small displays, a set of GHS stretch controls, and Reset and Apply buttons.

The histogram mode can be changed between linear and logarithmic using the toggle at the bottom of the dialog. This dialog obeys the Siril-wide preference for linear or logarithmic histograms that can be set in the Preferences window.

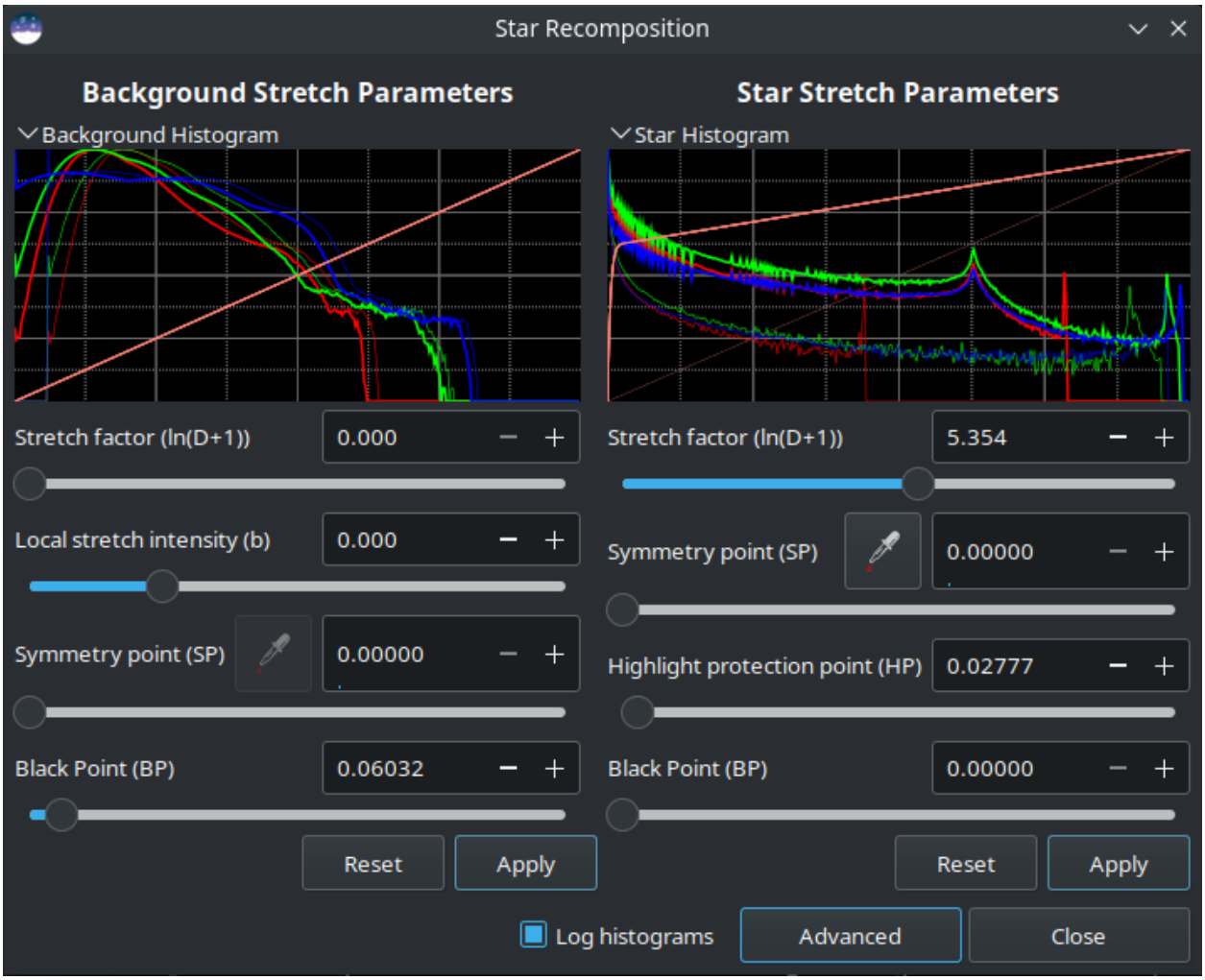



Fig. 67: Star Recomposition dialog box.

> **Tip**
>
> Star Recomposition does not obey the FITS 16-bit preference as there could be an adverse effect on performance in some cases (the calculations must be done using floating point maths anyway). So if you really want a 16-bit result to be created from 32-bit input images then you will need to use the *precision switch*.

### Simple Mode

The dialog has two views, which determines what controls are shown. It opens in Simple mode, which shows only the most useful controls for a typical starless / starmask combination.

- The stretch type for the starless image is set to Generalized Hyperbolic stretch and the Stretch Factor, Local stretch intensity, Symmetry Point and Black Point controls are shown. As well as using the SP control, the Symmetry Point can be set using the eyedropper tool to select the average pixel value of a selection from the image.

  *Note that the eyedropper tool is disabled when there is an unapplied BP shift: because of the process of applying the hyperbolic stretch and then the BP shift, the behaviour of the tool becomes non-intuitive when a non-zero BP parameter is set. To resolve this, simply apply the BP shift and the eyedropper will become available again for your next hyperbolic stretch.*

- The stretch type for the star mask image is set to Modified Arcsinh stretch and the Stretch Factor and Highlight Protection controls are shown.
- The human-weighted luminance color model is used for both sets of stretches: this does a better job of preserving colors in the unstretched image.

Details of all the stretch controls, both those shown in Simple mode as well as those shown in Advanced mode, can be found on the Generalized Hyperbolic Stretch documentation page.

The BP control works in a slightly different way to the BP control in the standalone GHS linear (black point adjust) stretch. In this tool the Black Point adjustment is applied *after* the hyperbolic stretch, whereas in the standalone tool it is a separate stretch applied by itself. When trying to optimize the combination of independent stretches to the two input images, this was found to be the most workable approach. It does mean that the amount of black point shift required in this tool is different to the amount required in the GHS tool, and that the Black Point cannot be set by clicking on the histogram.

Each stretch is independent. The stretch settings for the starless side can be applied using the left-hand Apply button: this stretches the starless image according to the current stretch settings and then resets the stretch settings so that further stretches can be applied in an iterative manner. Similarly, the stretch settings for the star mask can be applied using the right-hand Apply button. Either set of stretch settings can be reset to the defaults using its respective Reset button.

The dialog can be toggled between Simple and Advanced mode using the button at the bottom.

### Advanced Mode

In Advanced mode the full range of GHS stretch controls are available including Stretch Type, Colour stretch model and Shadow protection point for both input images. This allows greater customization of the two stretches if required. If the user interface is put back to Simple mode, any changes made using the advanced controls remain in effect, only the controls are hidden.

> **Note**
>
> It is not possible to stretch the saturation channels in this tool. The tool is already quite memory-hungry and CPU intensive: doubling the memory requirement by adding a HSL copy of each working image is considered excessive. Saturation can easily be stretched separately after the combination is complete.

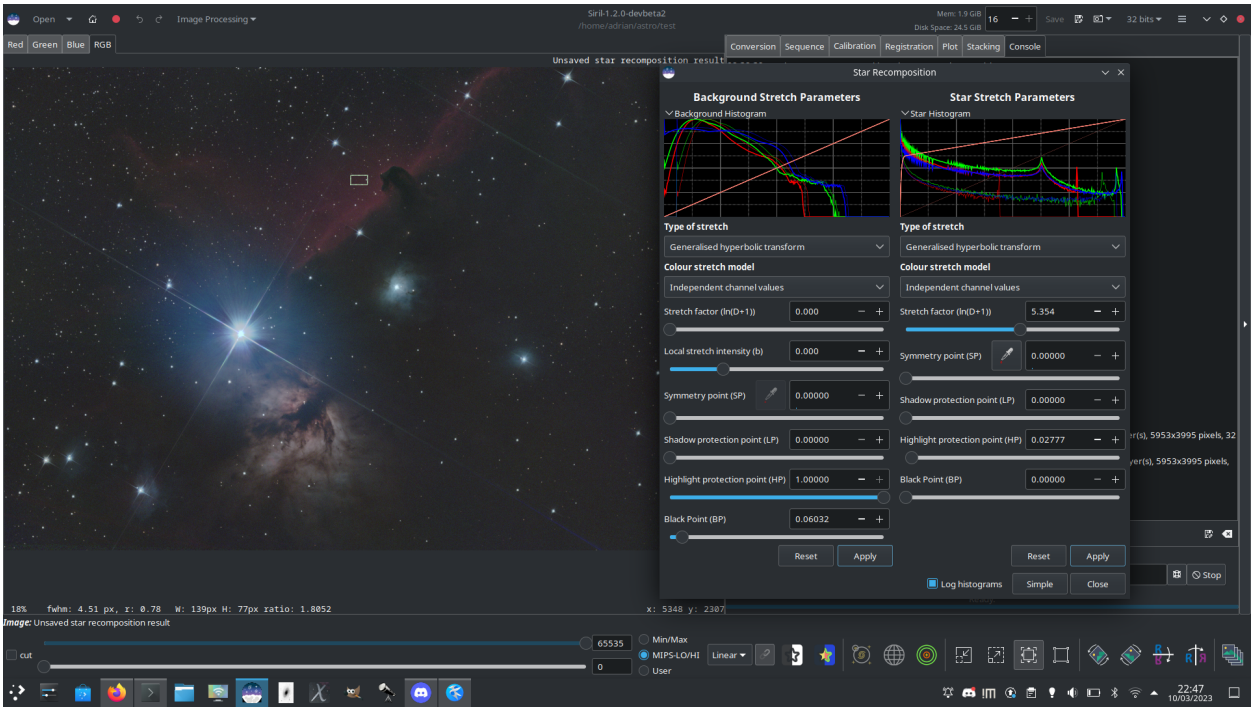



Fig. 68: Using Star Recomposition combine a starless image and star mask of the Alnitak region

### 10.5.3 Desaturate Stars

When a star finder is applied to an image (whose data are always linear), ellipses are displayed around the stars. When an ellipse is magenta, it means that the star is saturated.

A saturated star is a star whose brightest pixels have no more information and are clipped to the maximum value. In general we try to not to saturate the stars, even if this is not possible for the brightest. If despite the precautions there are still saturated stars, Siril has an algorithm that will reconstruct the profile of the star taking into account the results of the adjustment made during the findstar.

First, you need to perform a star detection, either with the *findstar* command or the button of the *Dynamic PSF* Window. Then, the desaturate tool is found in *Star Processing* → *Desaturate Stars*.

> **Tip**
>
> We recommend using a Moffat profile in the *Dynamic PSF* window to get better parameters.

> **Warning**
>
> It is important to run this tool on linear images, otherwise the stars will not have a Gaussian/Moffat profile and the calculations will be invalid.

After clicking on the tool, Siril switches to the console and displays the results of the current process:

```
22:26:17: Star synthesis (desaturating clipped star profiles): processing...
```

(continues on next page)

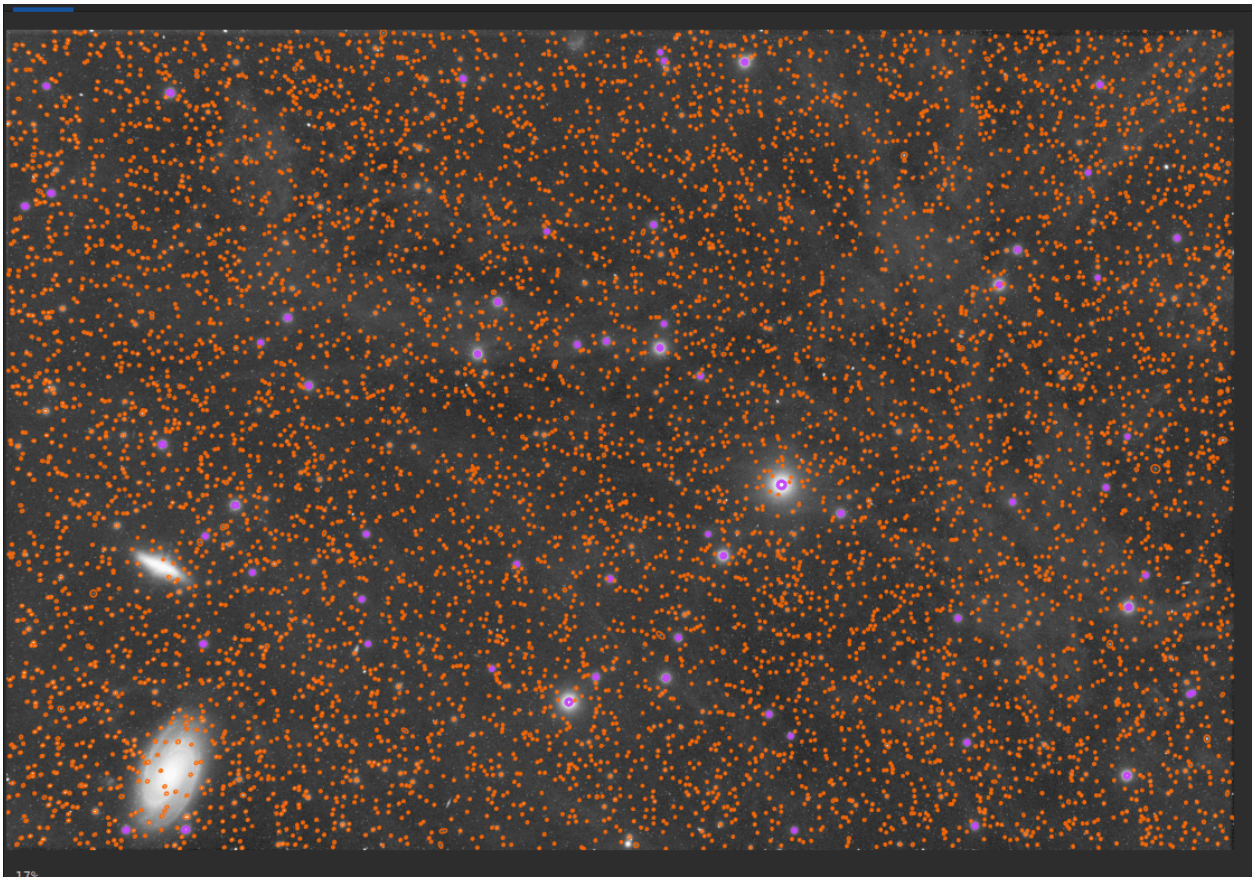


Fig. 69: A star detection shows all stars found by Siril. Magenta ellipses are for saturated stars. The image is displayed in autostretch view: data are still linear.

(continued from previous page)

```
22:26:17: Findstar: processing for channel 0...
22:26:21: Star synthesis: desaturating stars in channel 0...
22:26:21: Star synthesis: 70 stars desaturated
22:26:21: Remapping output to floating point range 0.0 to 1.0
22:26:21: Execution time: 4.09 s
```

It is necessary to run a star detection again to see the changes.

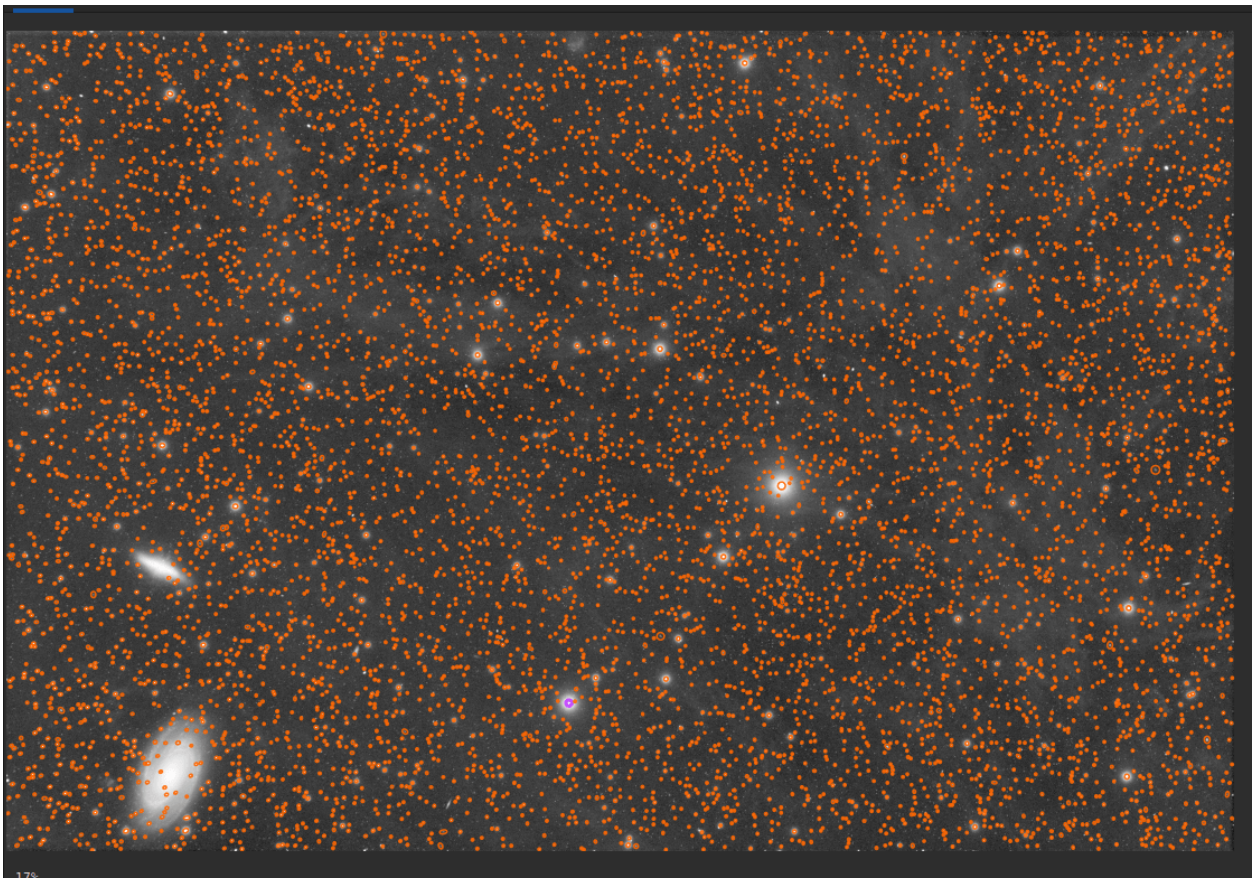


Fig. 70: After a desaturate processing, no more magenta ellipses are visible. All stars have been reconstructed. The image is displayed in autostretch view: data are still linear.

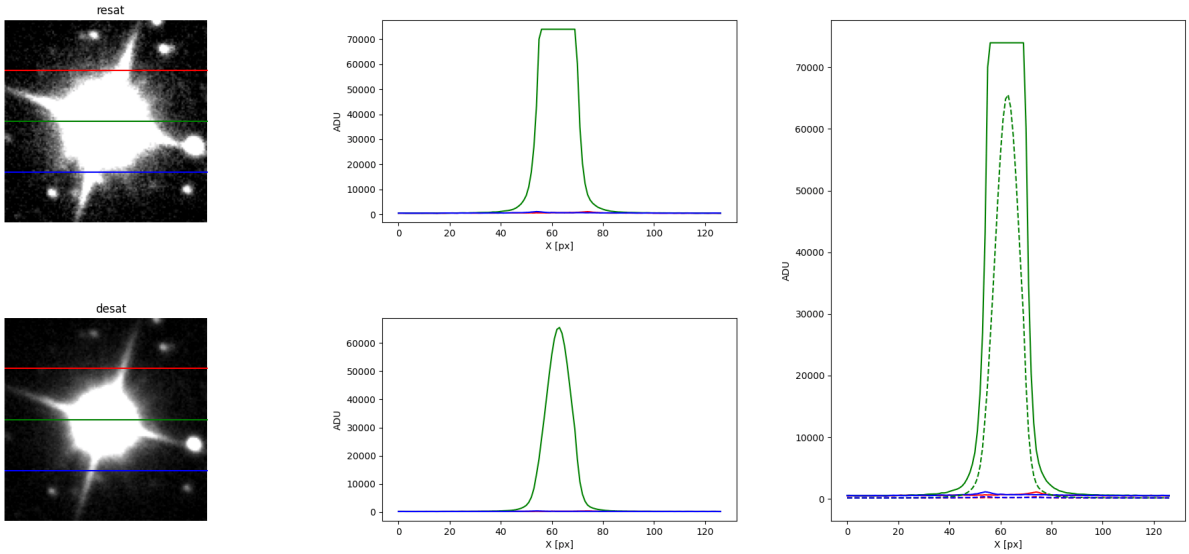



Fig. 71: Comparison for a star before and after the application of the desaturation tool.

**Siril command line**

```
unclipstars
```

Re-profiles clipped stars of the loaded image to desaturate them, scaling the output so that all pixel values are <= 1.0

### 10.5.4 Full Resynthesis

The Full Resynthesis tool aims to help to fix badly distorted stars using Siril's star fitting functions. It can be helpful for rescuing images that suffer from bad coma or other distortions. If Siril can detect the stars it can fix them.

The tool is located in the Image Processing menu, in the Star Processing sub-menu.

The output of the tool is a synthetic star mask. In order to make use of this, it must be recombined with a starless version of the original image. This can be prepared using the *starnet* command or Starnet GUI tool, or using third party star removal software.

This tool has no options, you simply click on the menu item to use it, or use the command *synthstar*.

If no stars have been detected in the image, the tool will automatically detect stars using the current star modelling parameters accessible via the Dynamic PSF tool or using the *setfindstar* command.

If stars have been modelled using the Dynamic PSF tool or the *findstar* command, the detected stars will be resynthesized using their individual modelled luminosity profiles. A shortcut to the Dynamic PSF tool is provided by means of the configuration button in the GUI menu next to the Full Resynthesis tool.

It is recommended to carry out star detection manually first, as it allows verification of the results: if any galaxies have been incorrectly detected as stars, they can be removed from the list of stars before running resynthesis.

Once the synthetic star mask has been created it can be combined with the starless image using the Star Recombination tool.

#### Commands



**Siril command line**

```
synthstar
```

Links: *psf*

## 10.6 Geometry

### 10.6.1 Rotate

#### Rotate 90 degrees

It is possible to rotate the image 90 degrees clockwise and counterclockwise with the dedicated menu. Here the rotation is done without interpolation of the pixels and it is therefore the preferred method if you want to rotate the image by a multiple of 90 degrees. This feature is also reachable through the icons and in the toolbar.

#### Rotate&Crop

For a rotation of another angle you have to use the Rotate&Crop tool. It allows a precise rotation and cropping that can be easily controlled.

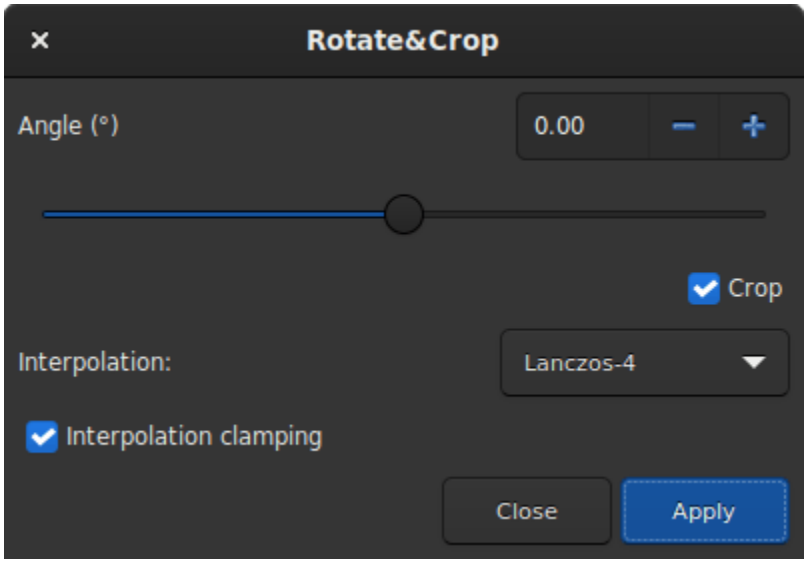



Fig. 72: Rotate&Crop dialog box showing all settings.

Five interpolation algorithms are available:

- Nearest Neighbor
- Bilinear
- Bicubic
- Pixel Area Relation
- Lanczos-4 (Default)

Lanczos-4 is the one that gives the best results. However, if you see artifacts, especially stars surrounded by black pixels, then you may want to try others. However, the button *Interpolation clamping* applies a clamping factor to Bicubic and Lanczos-4 interpolation in order to prevent ringing artifacts.

If you don't want the image to be cropped after rotation, then you should uncheck the *crop* button. However, the missing areas of the picture will be filled with black pixel.

The interest of this tool is that the rotation of the image is represented by a red frame, as illutrated in the figure below. In addition, if a selection is active, it is possible to change its size and see in real time the framing evolve.

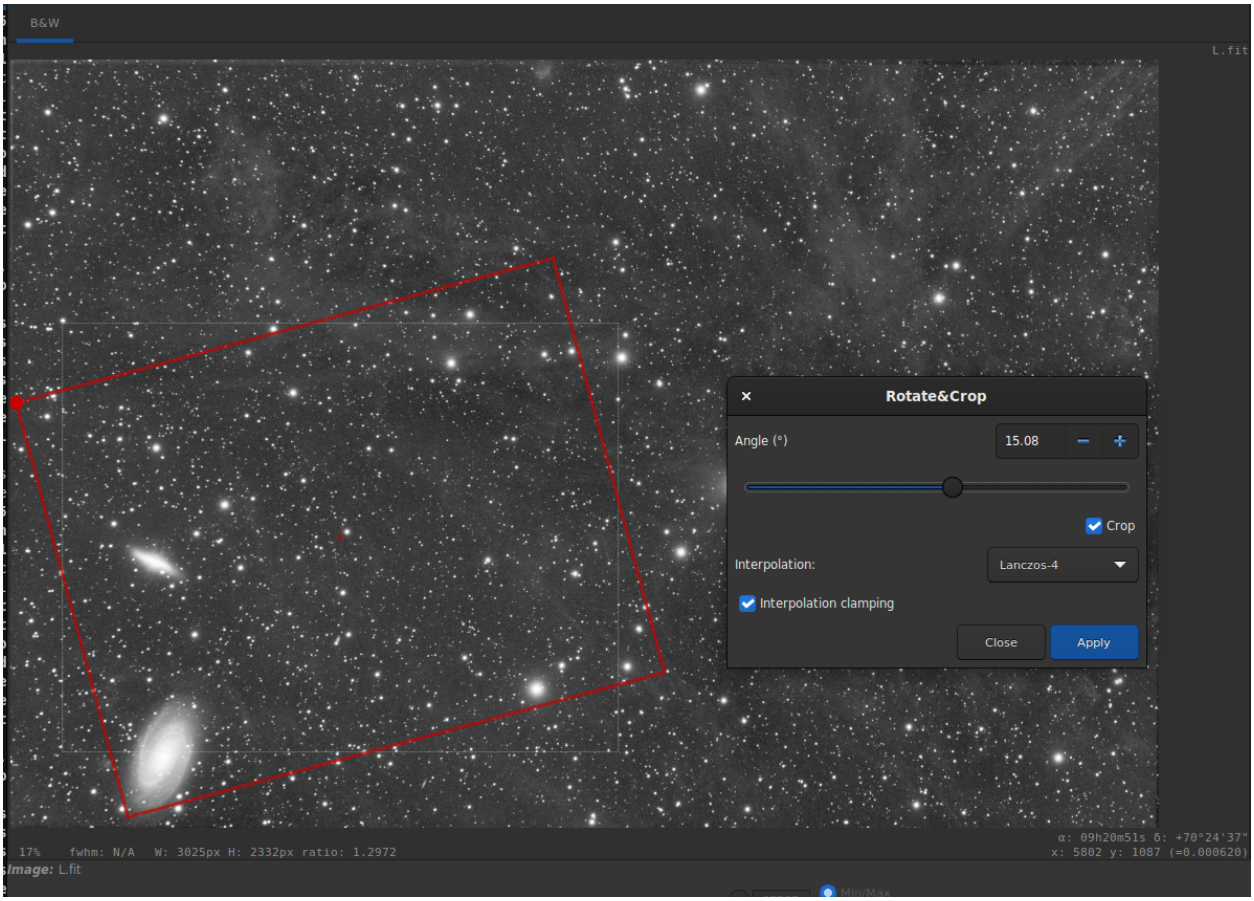



Fig. 73: Rotate&Crop dialog box with a active selection. Click to enlarge the figure and see the details better.

**Siril command line**

```
rotatePi
```

Rotates the loaded image of an angle of 180° around its center. This is equivalent to the command "ROTATE 180" or "ROTATE -180"

Links: *rotate*

**Siril command line**

```
rotate degree [-nocrop] [-interp=] [-noclamp]
```

Rotates the loaded image by an angle of **degree** value. The option **-nocrop** can be added to avoid cropping to the image size (black borders will be added).

Note: if a selection is active, i.e. by using a command `boxselect` before `rotate`, the resulting image will be a rotated crop. In this particular case, the option **-nocrop** will be ignored if passed.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.
Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument

### 10.6.2 Mirror

It is also possible to apply a mirror transformation to the image. Either along the x axis or along the y axis. This transformation is also accessible via the buttons and of the toolbar.

**Siril command line**

```
mirrorx [-bottomup]
```

Flips the loaded image about the horizontal axis. Option **-bottomup** will only flip it if it's not already bottom-up

**Siril command line**

```
mirrory
```

Flips the image about the vertical axis

### 10.6.3 Binning

The binning is a special transformation for resampling image. It computes the sum or mean of the pixels 2x2, 3x3, ... (depending of the binning factor) of the in-memory image (like the analogic binning of CCD camera).

**Siril command line**

```
binxy coefficient [-sum]
```

Computes the numerical binning of the in-memory image (sum of the pixels 2x2, 3x3..., like the analogic binning of CCD camera). If the optional argument **-sum** is passed, then the sum of pixels is computed, while it is the average when no optional argument is provided

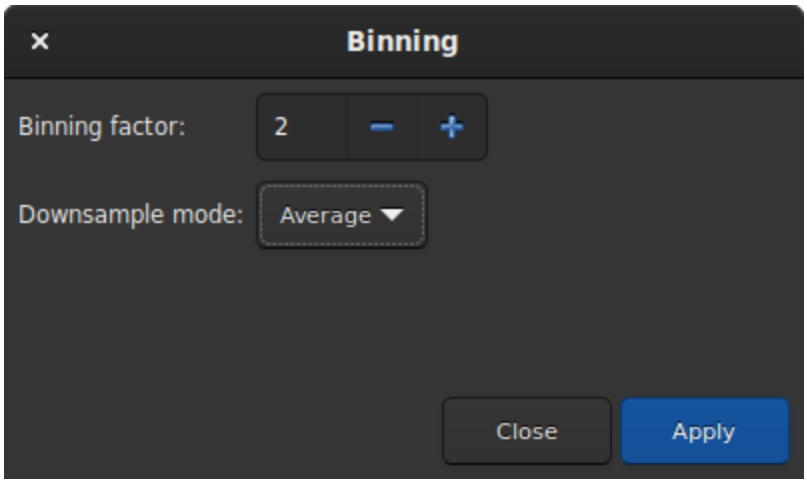



Fig. 74: Binning dialog box

### 10.6.4 Resample

The resample tool allows to resize the image at the cost of an interpolation chosen from the following list:

- Nearest Neighbor
- Bilinear
- Bicubic
- Pixel Area Relation
- Lanczos-4 (Default)

Lanczos-4 is the one that gives the best results. However, if you see artifacts, especially stars surrounded by black pixels, then you may want to try others. However, the button *Interpolation clamping* applies a clamping factor to Bicubic and Lanczos-4 interpolation in order to prevent ringing artifacts.

If you want to change the image ratio, then you should uncheck the *Preserve Aspect Ratio* button.

**Siril command line**

```
resample { factor | -width= | -height= | -maxdim= } [-interp=] [-noclamp]
```

Resamples the loaded image, either with a factor **factor** or for the target width or height provided by either of **-width=**, **-height=** or **-maxdim=**. This is generally used to resize images: a factor of 0.5 divides size by 2. The **-maxdim** argument can be used to resize the longest dimension of the image to a set size, which can be useful for optimizing images for certain websites, e.g. social media websites.

In the graphical user interface, we can see that several interpolation algorithms are proposed.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument

```
seqresample sequencename { -scale= | -width= | -height= } [-interp=] [-prefix=]
```

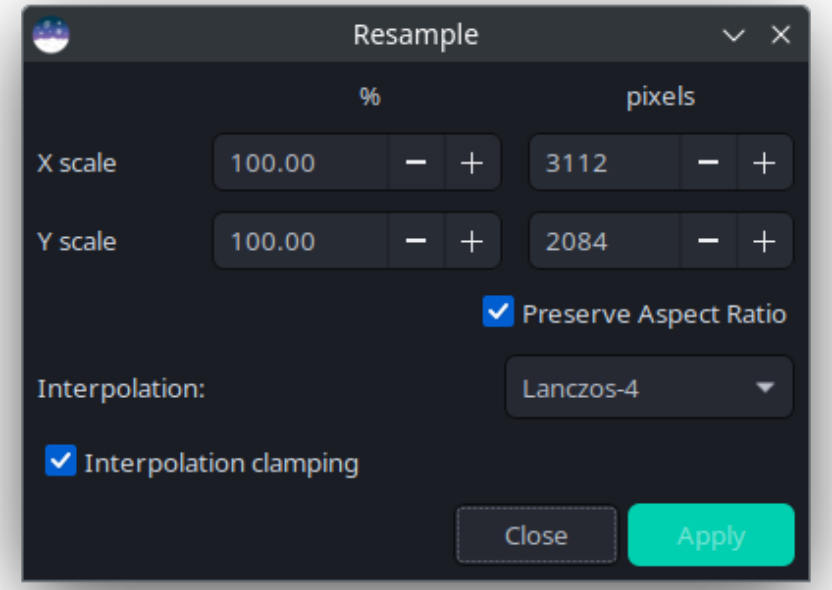



Fig. 75: Resample dialog box

Scales the sequence given in argument **sequencename**. Only selected images in the sequence are processed.

The scale factor is specified either by the **-scale=** argument or by setting the output width, height or maximum dimension using the **-width=**, **-height=** or **-maxdim=** options.

An interpolation method may be specified using the **-interp=** argument followed by one of the methods in the list **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}.. Clamping is applied for cubic and lanczos interpolation.

The output sequence name starts with the prefix "scaled_" unless otherwise specified with **-prefix=** option

## 10.7 Background Extraction

The sky background often has an unwanted gradient caused by light pollution, the moon, or simply the orientation of the camera relative to the ground. This function samples the background at many places of the image and looks for a trend in the variations and removes it following a smoothed function to avoid removing nebulae with it.

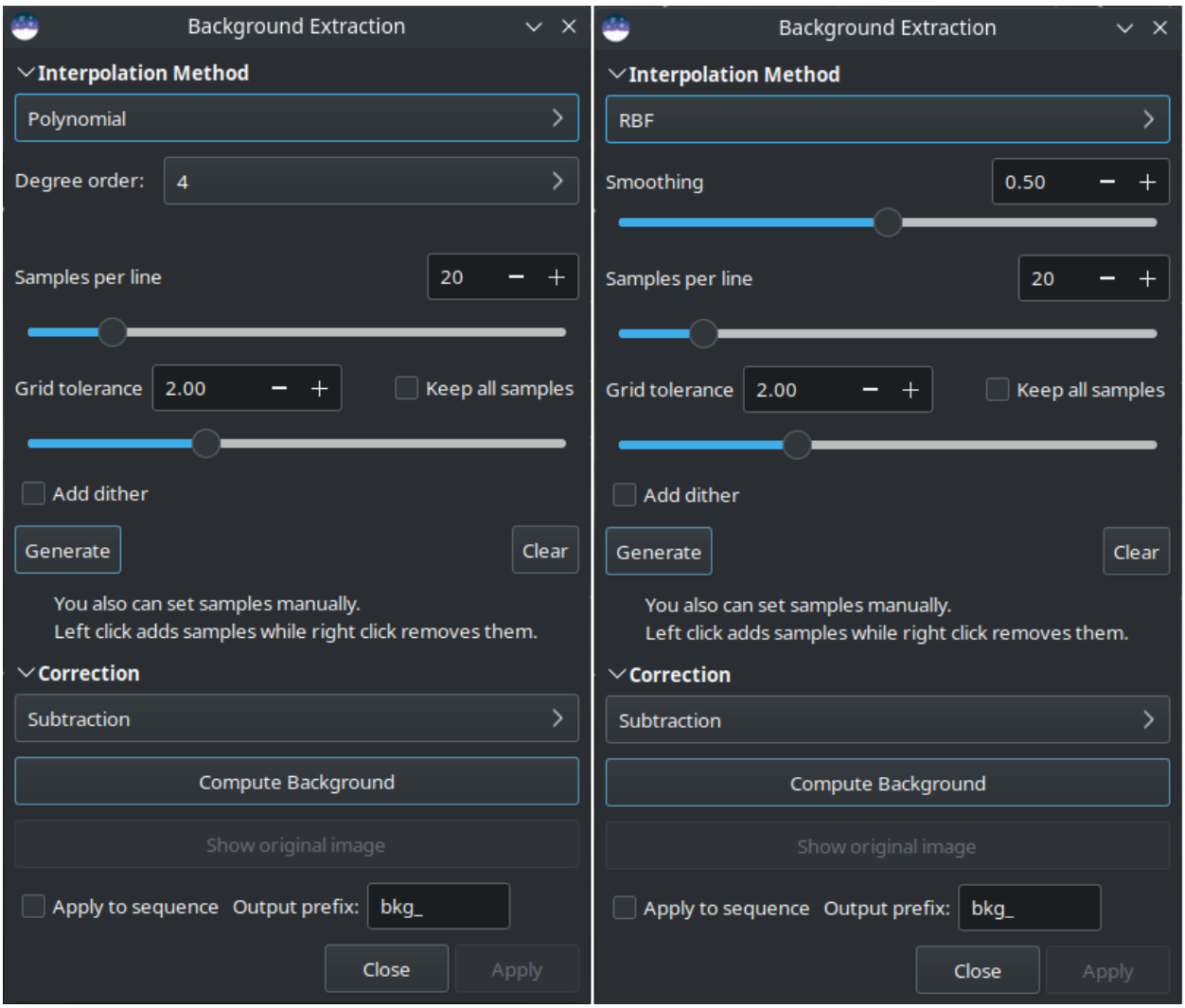



Fig. 76: Background extraction dialog box. On the left is the polynomial version, on the right RBF.

Samples can be automatically placed by providing a density (*Samples per line*) and clicking on *Generate*. If areas of the image are brighter than the median by some factor *Grid tolerance* times sigma, then no sample will be placed there.

**Tip**

If you have very strong gradients, for example when imaging in high Bortle urban skies, even the maximum grid tolerance may be insufficient. In this case you can check the *Keep all samples* checkbox and the full sample grid will be populated. You will then need to remove samples from actual astronomical objects manually.

After generation, samples can also be added manually (left click) or removed manually (right click).

There are two algorithms to remove the gradient:

### 10.7.1 RBF

This is the most modern method. It uses the radial basis function to synthesize a sky background to remove the gradient with great flexibility. It requires a single parameter which is present in the form of a slider: *Smoothing*. With this value you can determine how soft or hard the transition between the sample points is calculated. A high smoothing factor makes sense for large and uniform gradients, and a correspondingly lower value for small, local gradations.

**Tip**

Start with the basic setting (50%) and gradually tweak for optimal results.

**Theory**

Radial basis functions are functions of the form $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$, whereby in our case we use the Euclidean norm $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2}$. The function $f$, which describes the background model, can now be expressed as a linear combination

$$f(\mathbf{x}) = \sum_i w_i \, \phi(\|\mathbf{x} - \mathbf{x_i}\|) + o$$

where $w_i$ corresponds to the weights for the different sample points and $o$ corresponds to a constant offset.

The requirement that the function $f$ should pass through the sample points results in the condition

$$\begin{pmatrix} \phi(\mathbf{x}_1 - \mathbf{x}_1) & \phi(\mathbf{x}_1 - \mathbf{x}_2) & \dots & \phi(\mathbf{x}_1 - \mathbf{x}_N) & 1 \\ \phi(\mathbf{x}_2 - \mathbf{x}_1) & \phi(\mathbf{x}_2 - \mathbf{x}_2) & \dots & \phi(\mathbf{x}_2 - \mathbf{x}_N) & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ \phi(\mathbf{x}_N - \mathbf{x}_1) & \phi(\mathbf{x}_N - \mathbf{x}_2) & \dots & \phi(\mathbf{x}_N - \mathbf{x}_N) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ o \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \\ 0 \end{pmatrix},$$

which can only be fulfilled if the matrix on the left-hand side is invertible. With the right choice of function $\phi$ this can always be guaranteed [Wright2003].

In addition, the summand $s\,I$ is added to the matrix on the left-hand side, where $s$ is a smoothing parameter and $I$ is the unit matrix. The summand causes a regularization, which results in a smoother result the larger the parameter $s$ is. This parameter can be changed with the *Smoothing* parameter of the dialog box.

For the radial basis function, we use the thin-plate spline $\phi(|\mathbf{x}|) = |\mathbf{x}|^2 \log(|\mathbf{x}|)$.

### 10.7.2 Polynomial

This is the original and simplest algorithm developed in Siril. Only one parameter is used in polynomial computation: the *Degree order*. The higher the degree, the more flexible the correction, but a too high degree can give strange results like overcorrection.

> **Tip**
>
> A degree 1 correction can be very useful for when you want to remove the gradient on the subs.

> **Warning**
>
> Background removal can be carried out on CFA images, but only if they have Bayer patterns. (It is not supported for X-TRANS patterns.) For Bayer patterned images, the image is treated as four spatially interleaved images, each corresponding to a CFA subchannel. Each subchannel is independently processed to remove its gradient and then the subchannels are recombined into the original interleaved pattern.
>
> The intended use for this is to remove linear gradients from sequence frames prior to using Drizzle on a Bayer patterned sequence, and in that case it is strongly recommended to use linear (polynomial order 1) gradient removal as with other pre-stacking gradient removal.

### 10.7.3 General settings

- **Add dither**: Hit this option when color banding is produced after background extraction. Dither is an intentionally applied form of noise used to randomize quantization error, preventing large-scale patterns such as color banding in images.
- **Correction**:
  - **Subtraction**: it is mainly used to correct additive effects, such as gradients caused by light pollution or by the Moon.
  - **Division**: it is mainly used to correct multiplicative phenomena, such as vignetting or differential atmospheric absorption for example. However, this kind of operation should be done by master-flat correction.
- **Compute Background**: This will compute the synthetic background and will apply the selected correction. The model is always computed from the original image kept in memory allowing the user to work iteratively.
- **Show original image**: Keep pressing this button to see the original image.

The background gradient of pre-processed image can be complex because the gradient may have rotated with the acquisition session. It can be difficult to completely remove it, because it's difficult to represent it with a polynomial function. If this is the case, you may consider removing the gradient in the subexposures: in a single image, the background gradient is much simpler and generally follows a simple linear (degree 1) function.

> **Tip**
>
> Sometimes unsightly color banding appears after background extraction. When this happens, there are two things to check. Firstly, if the image is in 16-bit, we strongly advise you to always use the 32-bit format. If, despite everything, you still observe such artifacts, the **add dither** option, explained above, is the solution to your problem.

Fig. 77: When such banding occurs after gradient extraction. It can be solved with the **add dither** option (Courtesy of Nathan B.).

**Tip**

Good results with the RBF algorithm generally require fewer samples than with the polynomial algorithm.

**See also**

For more explanations, see the corresponding tutorial here.

**Siril command line**

```
subsky { -rbf | degree } [-dither] [-samples=20] [-tolerance=1.0] [-smooth=0.5] [-
↪existing]
```

Computes a synthetic background gradient using either the polynomial function model of **degree** degrees or the RBF model (if **-rbf** is provided instead) and subtracts it from the image.

The number of samples per horizontal line and the tolerance to exclude brighter areas can be adjusted with the optional arguments. Tolerance is in MAD units: median + tolerance * mad.

Dithering, required for low dynamic gradients, can be enabled with **-dither**.
For RBF, the additional smoothing parameter is also available. To use pre-existing background samples (e.g. if you have set background samples using a Python script) the **-existing** argument must be used

**Tip**

The -existing command argument forces use of existing background samples. This option is primarily for use in conjunction with the Python module where `SirilInterface.set_bgsamples()` can be used to set custom background samples based on user-defined algorithms. If it is not provided, *subsky* will automatically regenerate background samples. Note that the -existing option is not available with the seqsubsky command, because sequence frames are not necessarily registered at the time background subtraction is carried out, so the samples for one frame do not necessarily apply to another.

**Siril command line**

```
seqsubsky sequencename { -rbf | degree } [-nodither] [-samples=20] [-tolerance=1.0] [-
↪smooth=0.5] [-prefix=]
```

Same command as SUBSKY but for the sequence **sequencename**.
Dithering, required for low dynamic gradients, can be disabled with **-nodither**. Note that the **-existing** option is not available for sequence background removal, as the frames of a sequence are not necessarily always aligned.

The output sequence name starts with the prefix "bkg_" unless otherwise specified with **-prefix=** option. Only selected images in the sequence are processed

Links: *subsky*

## 10.8 GraXpert Interface

**Tip**

The first GraXpert interface was written in C and simply called the GraXpert executable. However for various reasons this approach proved unreliable for a significant minority of users and with Siril 1.4.0-beta2 an updated interface has been released as a Python script. This directly interfaces with the GraXpert model files and provides much better performance and integration, and since it builds on the way the Python interface uses the native PyPI modules library to install the correct versions of dependecies for each user's machine it is expected to be more reliable and better able to make use of different types of GPU.

Owing to the way GraXpert distributes its AI models from an Amazon S3 instance, a copy of the GraXpert program needs to be available in order to download model files. The model files you can download are dependent on the version of GraXpert installed therefore it is recommended to install 3.1.0-RC2 which provides access to the stellar and background object deconvolution models as well as the denoising and background extraction models that have been available since version 3.0.0.

The old GraXpert interface was marked as deprecated in 1.4.0-beta2 and has since been removed from the code base. This documentation covers the new Python-based interface.

Siril provides an interface to the AI models provided by the third party GraXpert tool. This is a third party piece of Free and Open Source software that provides sophisticated background removal and denoising routines. There is a lengthy history of collaboration between Siril and GraXpert, and the RBF algorithm used in Siril's core background removal function was contributed by the GraXpert developers several years ago.

In order to be used, the path to the working GraXpert installation must be set in *Preferences->Miscellaneous*.

### 10.8.1 Installation and Configuration

The GraXpert-AI.py script can be installed in the usual way, by going to the *Scripts* menu -> *Get Scripts* and selecting the check box next to the script in the list.

To configure the script you need to tell Siril where your GraXpert installation is. To do this, go to *Preferences -> Miscellaneous* and select the GraXpert executable using the appropriate file chooser widget.

### 10.8.2 General Aspects of the GUI

The GUI is shown below:

The title bar changes to show the currently selected operation, and at the top of the window are the script title and version number. A dropdown allows you to select the operation you wish to perform. How this dropdown is populated depends on the version of GraXpert you have available. It will always feature Denoising and Background Extraction, even if no locla models are available (see the section on Model Manager for information on how to download models). If you have locally available deconvolution models, or if you have set the location of a GraXpert version that supports deconvolution, you will also see Deconvolution (Stars) and Deconvolution (Objects), however if these models are not available locally and you don't have a version of GraXpert that is capable of downloading them they will not be shown.

Next follows an area for operation-specific parameters. If you select Denoising you will find a widget to control strength, if you select Background Extraction you will find widgets for smoothing and the type of background extraction to perform, and so on. These are described in more detail below.

The Advanced setting provides a means of controlling the inferencing batch size and whether or not to use GPU acceleration. It is generally a good idea to leave the batch size at the default: it sets how many chunks of the image can be processed in parallel, and increasing it may result in out-of-memory errors.

**Tip**

The GraXpert models use ONNX to perform inferencing. This is a hardware- agnostic Machine Learning framework that provides various runtimes targeting different types of acceleration hardware. There is a specific NVidia runtime, a DirectML runtime that provides generic Windows GPU acceleration, a MacOS runtime and Linux runtimes for NVidia, AMD and Intel GPUs. The script will automatically try to install the most suitable runtime for your hardware, however in the event that it does not work there is always a CPU runtime to revert to - simply disable the GPU acceleration checkbox and please submit a bug report as this is new functionality and we are especially keen to hear of any issues relating to the ONNX runtime selection.

Finally, the UI provides *Apply* and *Close* buttons that perform the expected functions and a *Model Manager* button that opens a tool to discover and download versions of the AI models.

When opening the GUI, it will detect whether a single image or a sequence is loaded. If a single image is loaded it will process it and update the result in Siril, whereas if a sequence is loaded it will process all the selected frames of the sequence and create a new sequence with a prefix dependent on the operation being performed.

**Warning**

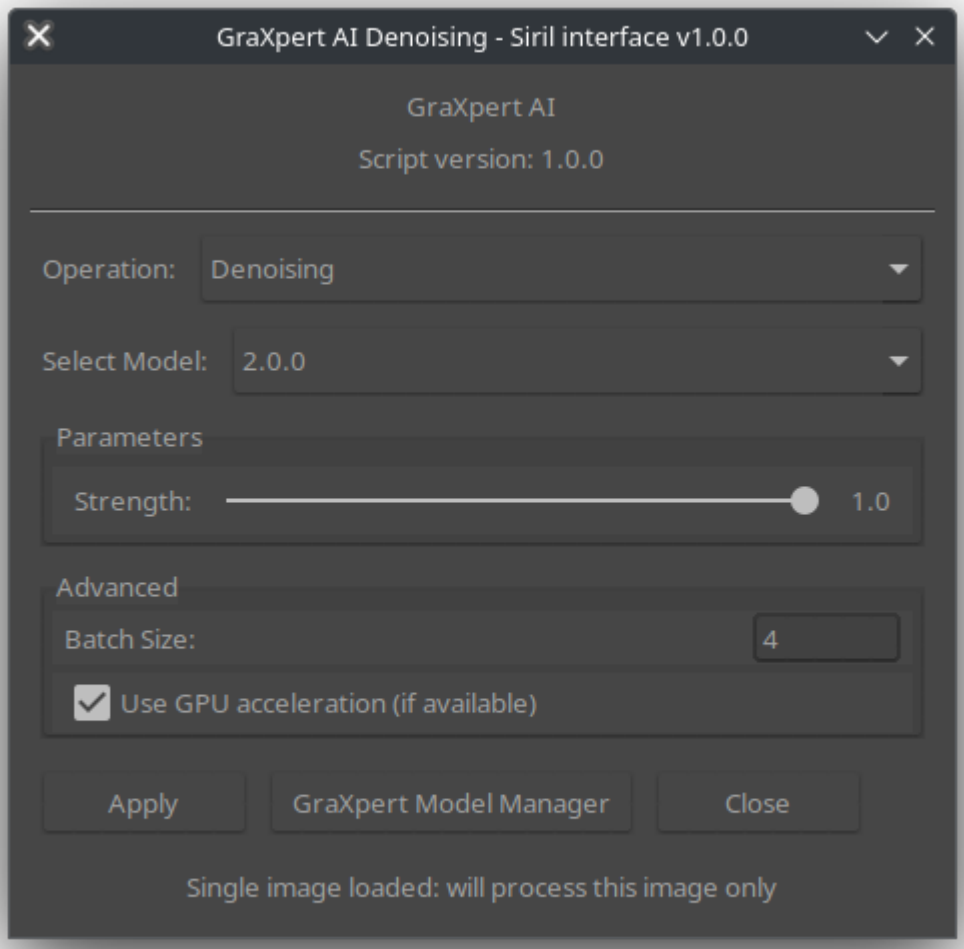



Fig. 78: GraXpert script interface, showing denoising options.

In Siril 1.4, Python scripting cannot subscribe to Siril event notifications so it is not possible for the script to detect if the image has been closed or if a change between a single image and a sequence occurs after opening the tool. Therefore if you do this, you should close and reopen the script GUI.

### 10.8.3 Background Extraction

The GraXpert background extraction AI model offers two parameters:

- Smoothing - this determines the smoothness of the background model that is generated.
- Correction type - this determines the type of correction that is applied: you can choose from subtraction or division.

Note that the GraXpert script does not provide an interface to the classical background removal algorithms provided by GraXpert. RBF is already natively available in Siril, and Spline and Kriging do not generally provide better performance than RBF.

### 10.8.4 Deconvolution

From version 3.1 onwards, GraXpert supports deconvolution. With the models downloaded, Siril supports both modes provided by GraXpert:

- Object deconvolution
- Stellar deconvolution

These modes can be selected using the *Operation* dropdown.

Both deconvolution modes have two parameters, *Strength*, which takes values between 0.0 (no deconvolution) and 1.0 (maximum deconvolution) and *Image FWHM (in pixels)*, which also takes values between 0.0 (prioritises fine structure retention) and 10.0 (prioritise deconvolution ability). If you set it to zero you can see a very (too) small effect on processed image, if so set it higher.

### 10.8.5 Denoising

To use the GraXpert AI denoising model, select *Denoising* in the Operations dropdown. This operation has only one parameter, *Strength*, which takes values between 0.0 (no denoising) and 1.0 (maximum denoising).

Denoising can take a little time, especially for large images and / or if using a CPU for inferencing. However once the denoising is done the denoised image is cached, so experimenting with different strengths is extremly quick.

### 10.8.6 Model Manager

The script provides a model manager to discover and download model versions. Whereas inferencing is done directly in the script, model download requires use of the actual GraXpert program because the models are stored in an Amazon S3 instance and GraXpert stores its credentials internally. The model manager will show radio buttons for each model that the installed version of GraXpert is capable of downloading. For v3.0.x this means denoising and background extraction; for v3.1.0-rcx this also adds stellar and object deconvolution.

The list of remotely available models is shown. If you want to discover models for a different AI operation, change the radio button to the operation you want a model for and click *Refresh*. A model can be downloaded by selecting it in the list and clicking *Download selected model*. Note that the download may take some time in the case of large models or slow network connections and GraXpert provides little feedback, so be patient. The models are stored in the GraXpert user data directory.

**Warning**

Do not move the GraXpert models from the GraXpert data directory structure where they are saved by the model manager. This is where the script expects to find them and it will ignore models put in any other location.

Once a model is downloaded it will automatically be populated in the model version dropdown in the main UI when the relevant AI operation is selected.

### 10.8.7 Scripting

The GraXpert-AI.py script provides a command-line interface for use in scripting. This uses the *pyscript* command as shown below.

**Siril command line**

```
pyscript [-async] scriptname.py [script_argv]
```

Executes a Siril python script

An optional argument **-async** may be provided. If it is provided it **must** be the first argument. This allows the script launched by pyscript to run in the background without blocking a script that calls it. This is expected to be useful for launcher-type scripts. The default is for the script to block until completion if pyscript is run as part of a script, but not to block if run from the GUI command entry (so the -async argument makes no difference when running like that).

The script name must be provided next (or first, if **-async** is not provided). If it is not found in the current working directory, the user-defined script paths specified in Preferences and the local siril-scripts repository will be searched. All subsequent arguments will be treated as script arguments and passed to the script as its argument vector. Note that the specific script must incorporate support for reading input from the argument vector

#### Operation Parameters

The following parameters are used to specify the operation to perform:

- **-denoise** specifies denoising
- **-deconv_obj** specifies object deconvolution
- **-deconv_stellar** specifies stellar deconvolution
- **-bge** specifies background extraction

#### General Parameters

The following parameters are recognised in all modes:

- **-h / --help** prints help to the Siril console and exits
- **-model** specifies the model version to use (e.g. -model="3.0.2") If this argument is omitted it defaults to the latest available version.
- **-batch** specifies the AI batch size
- **-gpu** specifies to use the GPU (this is the default)

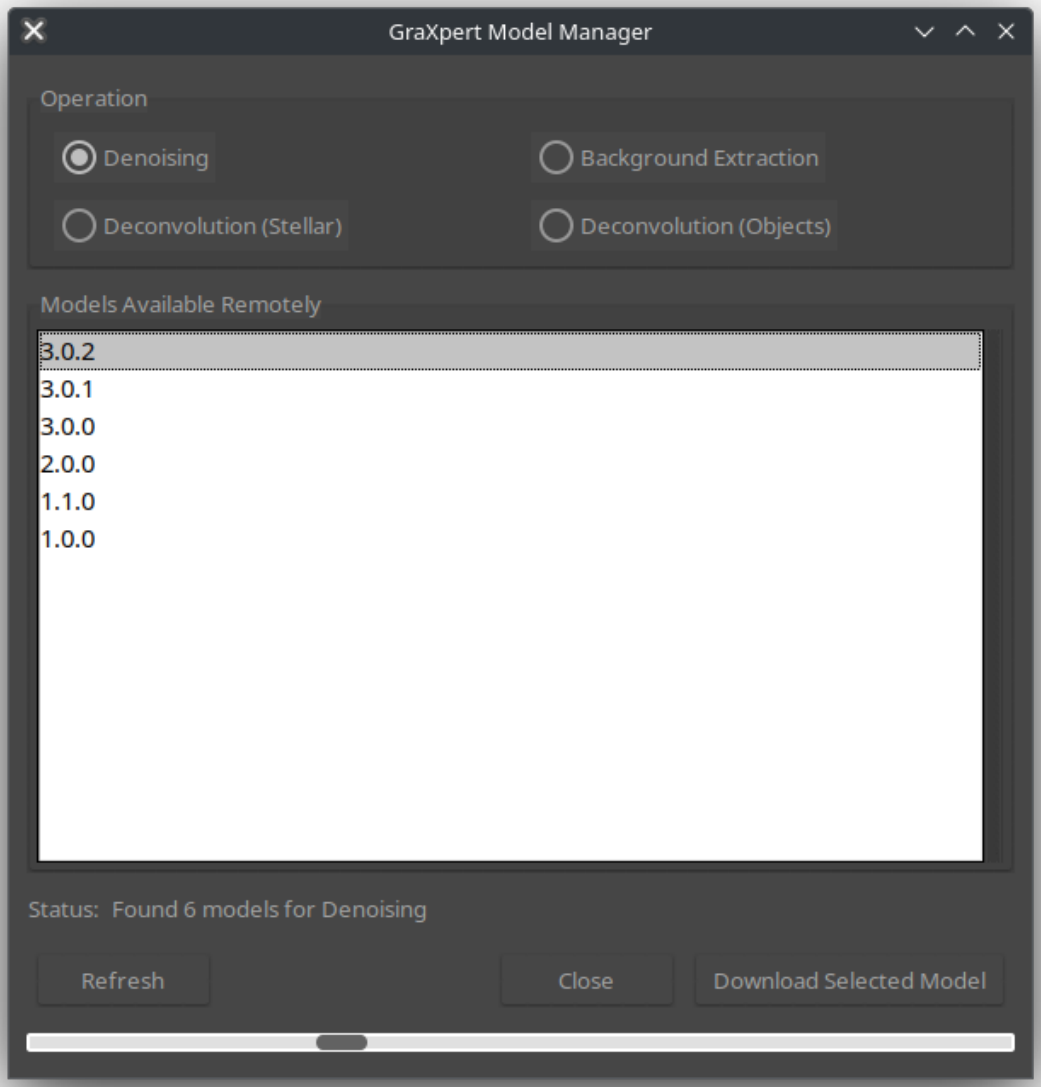



Fig. 79: GraXpert script interface, showing the model manager.

- **-nogpu** speficies to use CPU inferencing only
- **-listmodels** causes the script to list the available model versions for the specified operation and exit

> **Tip**
>
> Note that this script does not provide a -seq parameter as it expects to work on a sequence already loaded in Siril. When scripting, this can be achieved using the *load_seq* command first to load the desired sequence.

#### Background Extraction Parameters

- **-smoothing=** This option sets the smoothing parameter, between 0.0-1.0
- **-correction=** This option sets the correction type and may be either "subtraction" or "division"
- **-keep-bg** This option specifies that the modelled background should be saved as well. For a filename such as myfile.fit, the background will be saved as myfile_bg.fit

#### Denoising Parameter

- **-strength=** This option sets the strength parameter, between 0.0-1.0

#### Deconvolution Parameters

The same parameters are available for both stellar and object deconvolution:

- **-strength** This option sets the deconvolution strength, between 0.0-1.0
- **-psfsize** This option sets the PSF size, as described above

#### Examples

*pyscript GraXpert_AI.py -denoise -model='3.0.1'*

Or, if calling it from another Python script:

*siril.cmd("pyscript", "GraXpert-AI.py", "-denoise", "-model='3.0.1'")*

## 10.9 Channel Extraction

### 10.9.1 Split Channels

This function creates three monochrome images from a 3-channel color image, depending on the configured color space. For RGB, it's simply splitting the file in three. For the others, it involves computation of the equivalent color space, either HSL (hue-saturation-lightness), HSV (hue-saturation-value) or CIELAB.

> **Tip**
>
> If no name is given to a channel, then the channel is not extracted.

> **Siril command line**

```
split file1 file2 file3 [-hsl | -hsv | -lab]
```

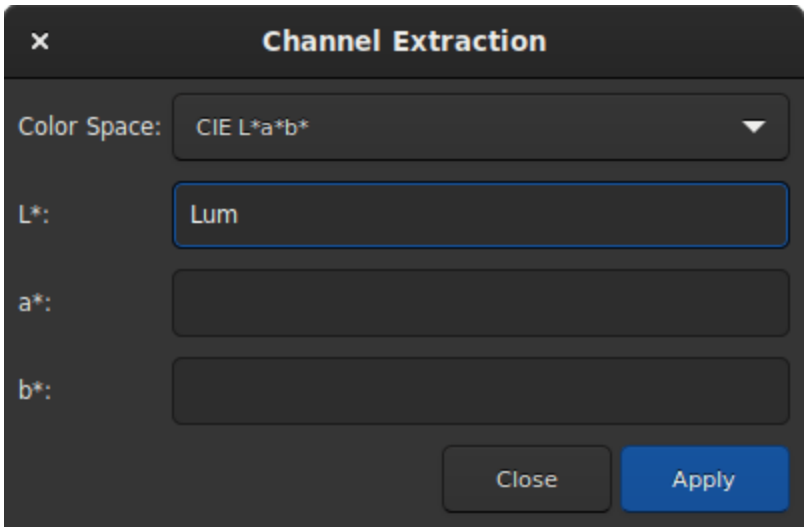



Fig. 80: Split channels dialog box.

Splits the loaded color image into three distinct files (one for each color) and saves them in **file1**.fit, **file2**.fit and **file3**.fit files. A last argument can optionally be supplied, **-hsl**, **-hsv** or **lab** to perform an HSL, HSV or CieLAB extraction. If no option are provided, the extraction is of RGB type, meaning no conversion is done

### 10.9.2 Split CFA Channels

CFA means color filter array. This term is often used to describe one-channel image content of a color image, with each pixel corresponding to values acquired behind an on-sensor filter. This is to oppose to debayer images (or debayered or demosaiced).

Opening a CFA image in Siril is required for pre-processing, like removing the dark signal before interpolating the image into 3-channel color. We can also use the color filter information to extract images like this:

- **Split CFA Channels**: four images are created from the CFA image, each representing one filter of the Bayer matrix, so in general R.fit, G1.fit, G2.fit and B.fit. It is useful if the goal is to process separately the different colors of the image.
- **Extract Ha**: using an H-alpha filter with a color camera image (OSC: on-sensor color, or one-shot color camera) means that only the pixels with red filters will be useful, so in general only a quarter of them. This function creates a new image that contains only the pixels associated with the red filter documented in the Bayer matrix of the image.
- **Extract Ha/OIII**: for OSC cameras, filters that let through photons from H-alpha and O-III wavelength have appeared. This extraction creates two images: an image from the red pixels like the Extract Ha, and an image combining the green and blue pixels into one for O-III. Both images are half the definition of the input image.

**Note**

There is a frequently asked question about why Ha and OIII images are different sizes and how they are split out. This note attempts to explain an answer to that FAQ.

In a colour image sensor the pixels are covered in a very fine filter matrix called a Color Filter Array (CFA) or Bayer matrix. The arrangement of filtered pixels is one of a number of patterns: RGGB, GBGR etc.

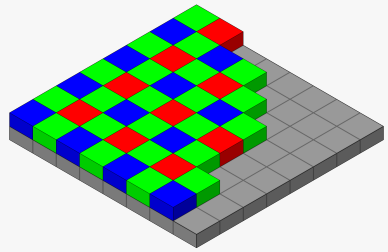


Fig. 81: Original image by Cburnett, licensed as CC BY-SA 3.0.

Of these pixels, only the R pixels are sensitive to Ha. So first we split out all the red pixels into a Ha image. As only 1 in 4 of the CFA elements are red, the image dimensions of the Ha image are half that of the original sub.

The remaining pixels, G and B, are all sensitive to OIII. The sensitivity of the G filtered pixels to OIII is different to the sensitivity of B filtered pixels to OIII, however they are imaging the same scene and evenly distributed so the average intensity must be the same.

$$\mathrm{G_i} = \mathrm{G_{io}} \times \frac{3 \times \overline{\mathrm{G_o}}}{2 \times \overline{\mathrm{G_o}} + \overline{\mathrm{B_o}}}$$
$$\mathrm{B_i} = \mathrm{B_{io}} \times \frac{3 \times \overline{\mathrm{B_o}}}{2 \times \overline{\mathrm{G_o}} + \overline{\mathrm{B_o}}}$$

Where $\mathrm{B_i}$ is the $i^{\mathrm{th}}$ blue pixel, $\mathrm{B_{io}}$ is the $i^{\mathrm{th}}$ original blue pixel and $\overline{\mathrm{B_o}}$ is the average of all the original blue pixels (and similarly for the green pixels).

So far we have an equalised set of G and B pixels with gaps where the R pixels have been removed. So finally we use bilinear interpolation to estimate the R pixel values and end up with a full size OIII image.

**Note**

The Ha/OIII resampling option is how to handle the output of Extract Ha/OIII. No resampling produces full resolution OIII image and a half resolution Ha image; upsample Ha upsamples the Ha image by a factor of 2 to match the OIII image; downsample OIII downsamples the OIII image by a factor of 2 to match the Ha image.

You may wish to use drizzling to upscale the Ha data instead of upscaling. As drizzling is a stacking method, in this case you must use *seqextract_HaOiii* to extract the Ha and OIII from each frame of the sequence, and then stack the OIII images in the usual way and the Ha images with a 2x drizzle.

- **Extract Green**: for photometry, it's often useful to only process the green part of the CFA image, because it is more sensitive and has two pixels to average, reducing noise even more. Of course, the created image also sees its definition halved by two.

**Note**

These functions only work if the Bayer matrix has been properly documented by the acquisition software and if the image format supports it, so in general FITS or SER.

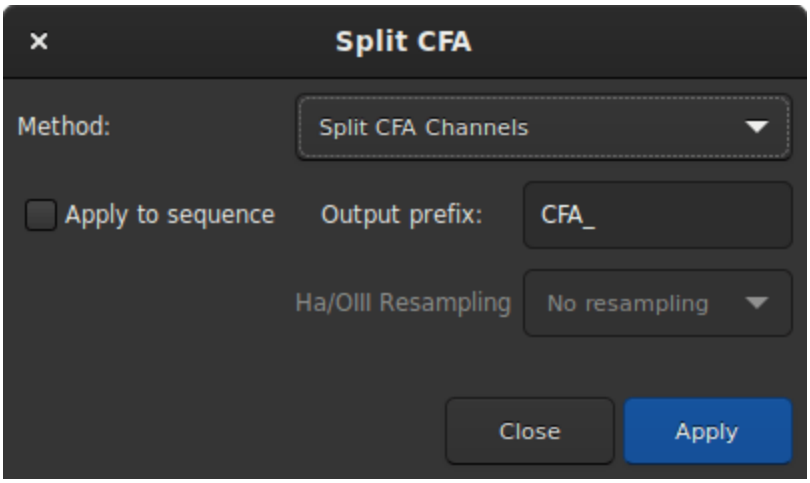



Fig. 82: Split CFA channels dialog box.

> **Warning**
>
> This does not work with other filter matrices than the Bayer matrices, like the Fujifilm X-TRANS.

### 10.9.3 Wavelet Layers

This tool extracts the different planes of the image by applying the wavelet process. Each plane is saved in an image and the set of images can be read as a sequence. You can choose up to 9 layers for the wavelet calculation and the type of the algorithm is either Linear or BSpline. The latter is usually the preferred one.

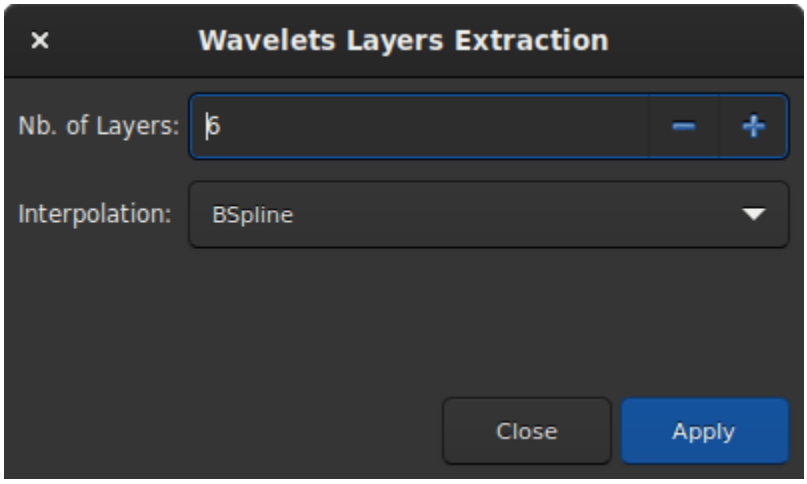



Fig. 83: Wavelet Layers extraction dialog box.

The decomposition is done through a number of detail layers defined at increasing characteristic scales and a final residual layer, which contains the remaining unresolved structures.

Fig. 84: Original image of M45 (courtesy of V. Cohas).

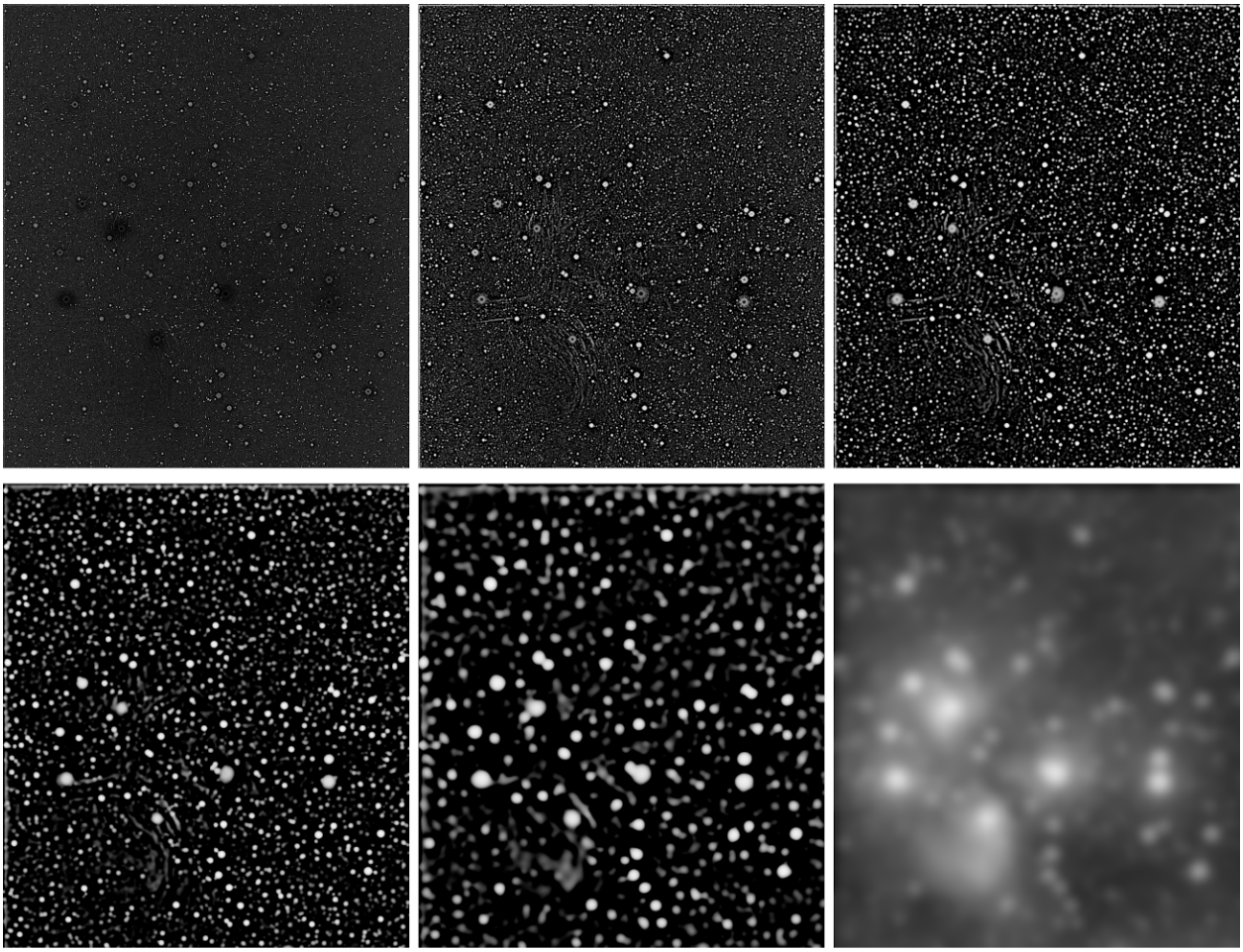


Fig. 85: 6 extracted planes.

## 10.10 Linear Match

Linear matching is the process of finding a linear function that matches best (in the sense of Least Squares) the intensity of pixels from one image to those of a reference image. This is a quick and easy way to balance the histograms of different images.

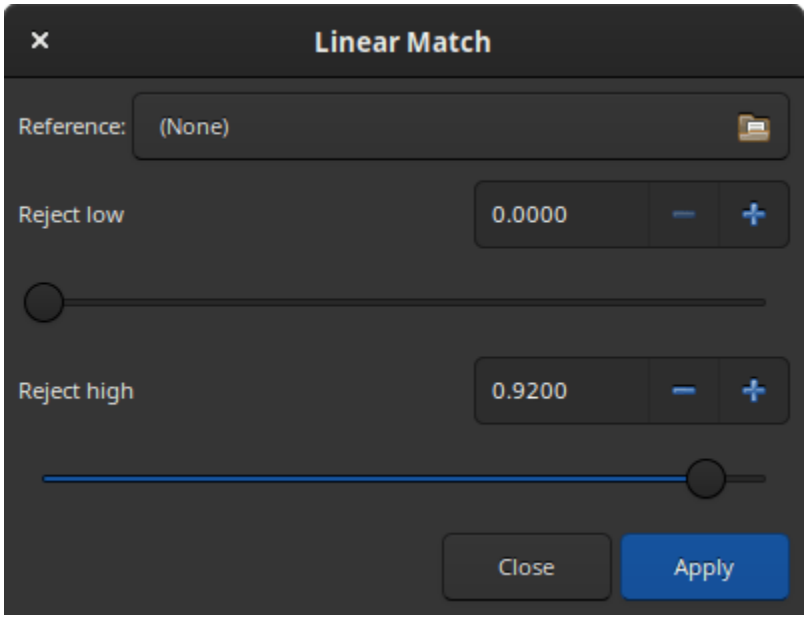



Fig. 86: Linear match dialog

The *Reference* allows you to pick the reference image.

The *Reject low* and *Reject high* sliders allows to exclude pixels values in the left and right tails of the intensities distributions. They are defined as quantiles, in the range [0, 1]. For instance, default for high is 0.92, meaning that the 8% brightest pixels will be excluded from the fitting to find the linear match coefficients.

> **Warning**
>
> The image and reference must be aligned prior to applying a linear match. Otherwise, there is no reason to assume that their pixels intensities are correlated.

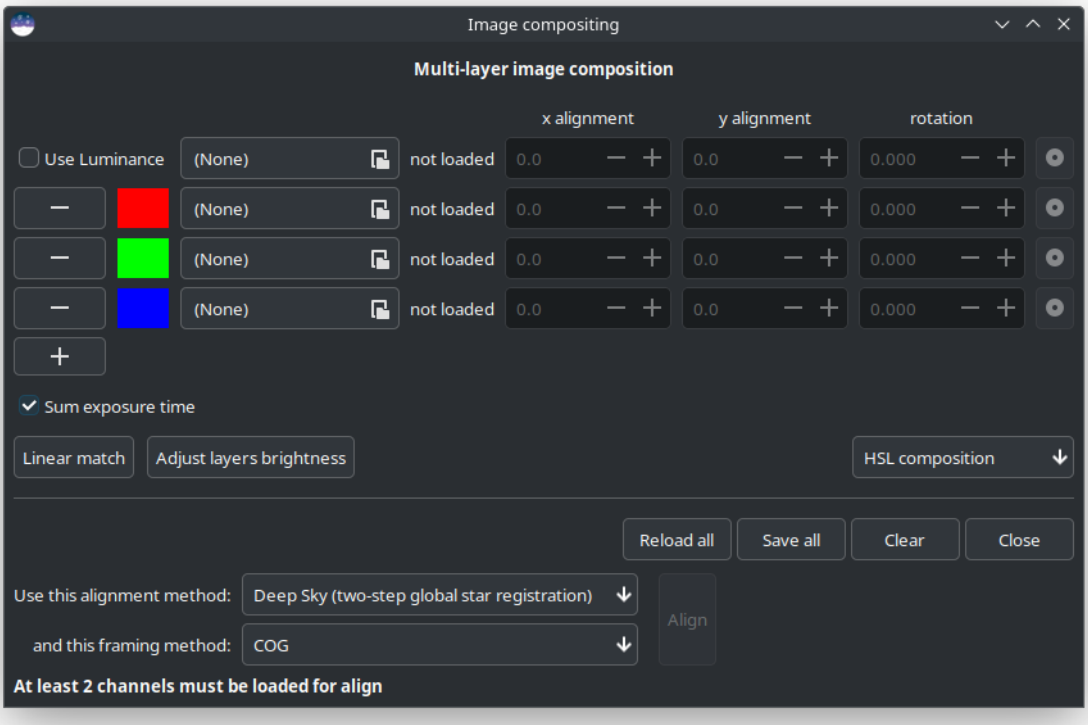
**Siril command line**

```
linear_match reference low high
```

Computes and applies a linear function between a **reference** image and the loaded image.

The algorithm will ignore all reference pixels whose values are outside of the [**low**, **high**] range

## 10.11 RGB compositing




The RGB composition tool allows you to assemble up to 8 monochrome images to form a single color image. The tool has been updated to provide access to all the best registration methods available in Siril, plus a manual registration method unique to this tool.

The operation of this tool is quite simple, just load the images and assign them a color. The first field, optional, is reserved for the luminance layer. Once a luminance layer is loaded you can integrate it or not in the composition thanks to the *Use Luminance* button. Each color can be customized by clicking on it and choosing a new one. When more than 3 images (or 4 if there is luminance) are loaded, or when the assigned colors are color mixtures rather than pure R, G and B, it may be necessary to adjust the brightness of each channel. The *Adjust layers brightness* button performs this operation automatically.

When compositing different channels, especially from narrowband filters, the histograms may be severely mismatched. A basic initial correction can be made by applying a linear match using the *Linear match* button. This works in the

same way as the *Image Processing* menu entry, but is tailored to this particular tool. The top layer loaded (usually Luminance or Red) is taken as the reference channel, and all other channels are automatically linearly matched to it. This is particularly important if assigning non-pure colours to some layers, as each layer will potentially be blended into each RGB channel in the result, and it will no longer be possible to separately adjust the histograms of the input layers by adjusting the R, G and B histograms of the composition.

It is possible enable or not the *Sum exposure time*. This option gives the user the option of summing or not the exposures of individual images, storing the result in the FITS header.

> **Note**
>
> For binning and image dimensions, the first loaded image determines the size of the output image. If you have images of different sizes, you should always load the largest first. If your images are different just because of binning, so with the same field of view, the composition tool will upscale the smaller images when they are loaded to match the size of the first loaded image. It is useful for the common L-RGB taken with the colour filters in bin 2. This also means that if two images have not been taken with the same sensor, it is unlikely they will have the same field of view and pixel sampling after image resampling. Whether or not this tool can align such images will depend on the scale mismatch and the alignment method used.

Three color spaces are available for rendering the composition:

- HSL (for hue, saturation, lightness)
- HSV (for hue, saturation, value; also known as HSB, for hue, saturation, brightness)
- CIE L*a*b*

and are left to the choice of the user.

Once the composition is finished, it is possible to do a linear match of the channels using the *Linear match* button.

If the images are not aligned with each other, then it is possible to align them. The following algorithms are possible:

- **Deep Sky (two step global star registration)**: no selection is required for this method. This generates a very accurate 8 degrees-of-freedom homography mapping between images.
- **Planetary (DFT pattern alignment)**: you have to draw a selection around the object you want to align. A high enough contrast is required for the algorithm to work properly. This method only registers image shifts, it cannot compensate for rotation. This is fine for planetary images but for deep sky images it is recommended to use the global star registration method.
- **KOMBAT**: as with the DFT pattern matching, you have to draw a selection around the object you want to align. This method also only registers image shifts. It is much faster than the DFT method, but can be less successful when the image patterns in the different channels are significantly different (for example, two planetary channels filtered with IR and UV filters may not share enough pattern similarity for KOMBAT to match them).
- **Manual alignment**: this method is unique to the RGB composition tool. When selected, the spin buttons change function from displaying information about alignment shifts to allowing you to enter desired shifts in x and y as well as rotation. Rotation defaults to centering on the image center, but a custom rotation center may be chosen for each layer using the centermark button at the right of each set of layer controls. This method may be of use if you have imagery that is not suited to alignment using any of the other methods, or if you wish to fine tune results.

You can choose the framing method that will be applied to your aligned layers. The following methods are possible:

- **First channel** - this method frames all other frames to the first channel (usually Luminance or Red). This makes particular sense for LRGB compositions.
- **MIN** - this method applies the minimum framing: the largest rectangle including only regions that are covered by all the input images will be framed.

- **COG** - this method selects an optimum framing for the composition based on the centre of gravity of the aligned image pixels.

> **Note**
>
> Framing methods do not apply to DFT shift alignment. This method always frames based on the first layer. If this alignment method is selected, the chosen framing method will be ignored.

> **Tip**
>
> On alignment the RGB composition tool aligns the loaded images in memory. That means that if you decide you aren't happy with the alignment and want to try a different method or a different framing, you have to reload all the images to get back to their original state before alignment. The *Reload all* button is provided for this purpose.

If you prefer to composite your images in a different way, for example using Pixel Math, you can still use the RGB composition tool as a quick means of aligning the images. After alignment, pressing *Save all* will save all the aligned images with their original filenames prefixed with *comp_*. These can then be brought into whatever tool you like and, unlike if you create a sequence with your stacks and align it in the normal way, it is obvious from the filename which image is which.

This tool is by nature graphical and is not exactly replicated in a Siril command, however there are commands that can be used to achieve part of the process:

- *Registration* can align a sequence containing your stacked images,
- The *rgbcomp* command can compose a RGB image from 3 separate R, G and B images or 4 L, R, G, B images, however the images must be pre-aligned and it cannot assign mixed colors to the input images.
- *Pixelmath* can also be used in scripts to combine channels.
- There is also a built-in RGB Composition script that can partly automate the process.

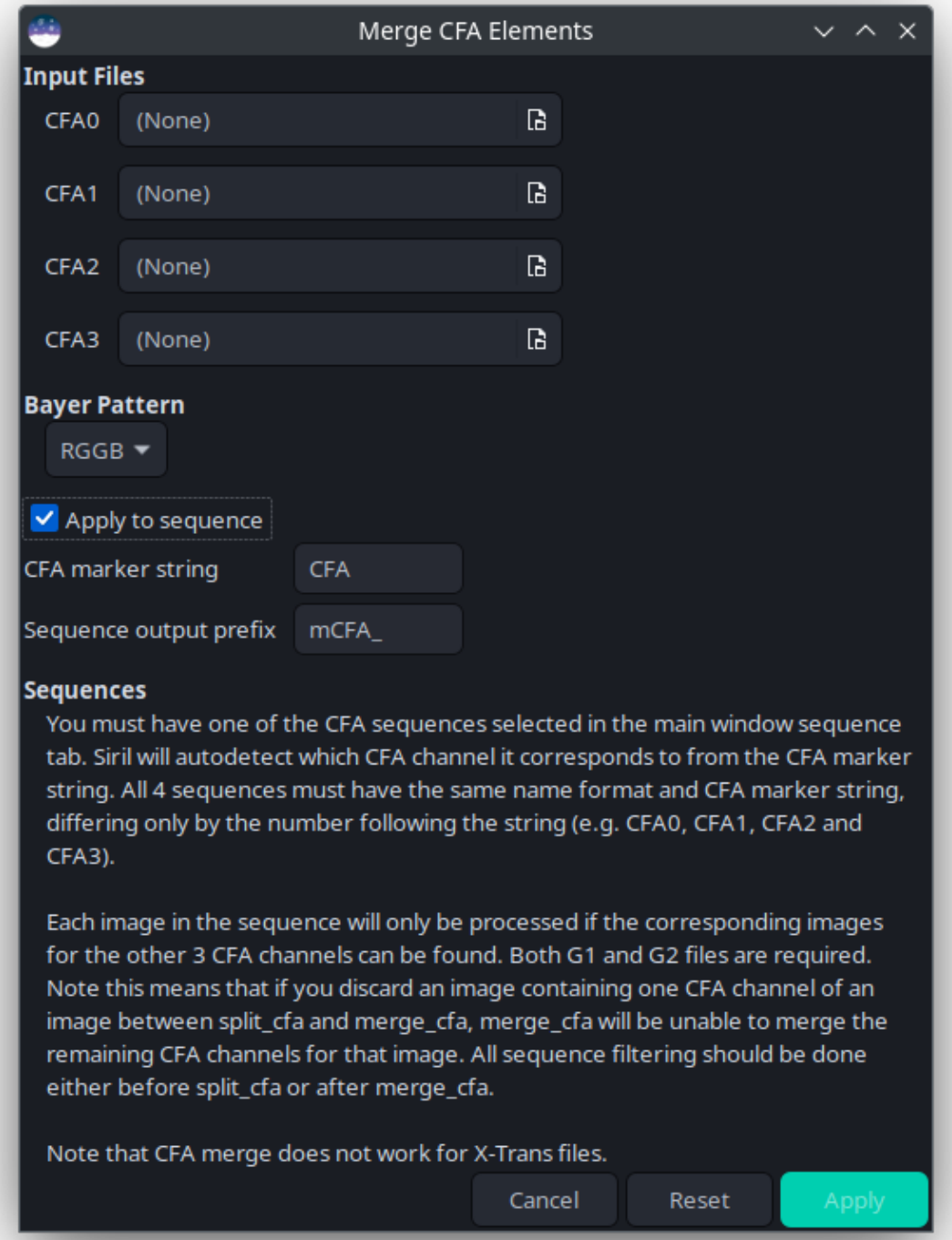
> **Siril command line**

```
rgbcomp red green blue [-out=result_filename] [-nosum]
rgbcomp -lum=image { rgb_image | red green blue } [-out=result_filename] [-nosum]
```

Creates an RGB composition using three independent images, or an LRGB composition using the optional luminance image and three monochrome images or a color image. Result image is called composed_rgb.fit or composed_lrgb.fit unless another name is provided in the optional argument. Another optional argument, **-nosum** tells Siril not to sum exposure times. This impacts FITS keywords such as LIVETIME and STACKCNT

## 10.12 Merge CFA Channels

The purpose of this tool is to combine multiple monochrome images that have been previously extracted from a CFA sensor (with the *Extraction* → *Split CFA channels...* menu for example). The tool merges the separate red, green (x2), and blue channel images into a single composite image called CFA image.


Merge CFA Elements
Input Files
CFA0
(None)
CFA1
(None)
CFA2
(None)
CFA3
(None)
Bayer Pattern
RGGB
Apply to sequence
CFA marker string
CFA
Sequence output prefix
mCFA_
Sequences
You must have one of the CFA sequences selected in the main window sequence tab. Siril will autodetect which CFA channel it corresponds to from the CFA marker string. All 4 sequences must have the same name format and CFA marker string, differing only by the number following the string (e.g. CFA0, CFA1, CFA2 and CFA3).
Each image in the sequence will only be processed if the corresponding images for the other 3 CFA channels can be found. Both G1 and G2 files are required. Note this means that if you discard an image containing one CFA channel of an image between split_cfa and merge_cfa, merge_cfa will be unable to merge the remaining CFA channels for that image. All sequence filtering should be done either before split_cfa or after merge_cfa.
Note that CFA merge does not work for X-Trans files.
Cancel
Reset
Apply

**Warning**

This tool is dedicated to images from a Bayer matrix and therefore it cannot work with images from X-Trans files from Fuji cameras.

The dialog is split in three different parts:

- **Input files**: Select the images containing the CFA0, CFA1, CFA2 and CFA3 Bayer subpatterns. If these have been produced using Siril's "Split CFA" function they will have the CFA prefix.
- **Bayer Pattern**: Sets the Bayer pattern header to be applied to the result. This must match the Bayer pattern of the image that the original Bayer subchannels were split from.
- The sequence part, at the bottom, allows to process whole sequences by reconstituting a CFA image sequence. Clicking on the *Apply to sequence* button displays a help text to proceed correctly. This text is reported in the next tooltip. There are two available options:
  - **Sequence input marker**: Identifier prefix used to denote CFA number in the separate CFA channel images. This should be set to whatever sequence prefix was used when the split_cfa process was run (default: `CFA_`).
  - **Sequence output prefix**: Prefix of the image names resulting from the merge CFA process. By default it is `mCFA_`.

**Tip**

You may have any of the CFA0, CFA1, CFA2 or CFA3 sequences selected in the main window sequence tab: Siril will determine which one it is from the prefix and the number.

Your separate sub-CFA sequences should have been processed in exactly the same way; they **must** have the same sequence name format and CFA marker string, differing only but the number 0-3 following the CFA marker string (e.g. CFA0_bg_pp_lights, CFA1_bg_pp_lights, CFA2_bg_pp_lights and CFA3_bg_pp_lights).

Each image in the sequence will only be processed if the corresponding images for the other 3 CFA channels can be found. Both Green subchannel images are required. Note this means that if you discard an image containing one CFA channel of an image between split_cfa and merge_cfa, merge_cfa will be unable to merge the remaining CFA channels for that image. All sequence filtering should be done either before split_cfa or after merge_cfa.

**Siril command line**

```
merge_cfa file_CFA0 file_CFA1 file_CFA2 file_CFA3 bayerpattern
```

Builds a Bayer masked color image from 4 separate images containing the data from Bayer subchannels CFA0, CFA1, CFA2 and CFA3. (The corresponding command to split the CFA pattern into subchannels is **split_cfa**.) This function can be used as part of a workflow applying some processing to the individual Bayer subchannels prior to demosaicing. The fifth parameter **bayerpattern** specifies the Bayer matrix pattern to recreate: **bayerpattern** should be one of 'RGGB', 'BGGR', 'GRBG' or 'GBRG'

**Siril command line**

```
seqmerge_cfa sequencename0 sequencename1 sequencename2 sequencename3 bayerpattern [-
↪prefixout=]
```

Merges 4 sequences of images to recombine the Bayer pattern. The sequences are specified in the arguments **sequencename0**, **sequencename1**, **sequencename2** and **sequencename3**.

The Bayer pattern to be reconstructed must be provided as the second argument as one of RGGB, BGGR, GBRG or GRBG (the order of the Bayer channels must match the order of the specified sequences).

Note: all 4 input sequences **must** be present and have the same dimensions, bit depth and number of images.

The output sequence name starts with the prefix "mCFA_" and a number unless otherwise specified with **-prefixout=** option

## 10.13 Pixel Math

One of the most powerful tools in Siril is the Pixel Math. It allows you to manipulate the pixels of the images using mathematical functions. From simple addition or subtraction, to more advanced functions, like MTF, Pixel Math is a perfect tool for astronomical image processing.

This page aims to describe the tool entirely, to see detailed examples, please refer to the excellent tutorial on the site.

The window is divided into 5 parts.

1. The first one, including 3 text zones receiving the mathematical formulas. Only the first one is used if you want to produce a monochrome image. Uncheck the *Use single RGB/K expression* button to produce RGB output.
2. The second is the variables area with the selection of *Functions* and *Operators*. Each variable is an image that must be loaded beforehand with the + button. You can click on the desired function and/or operator to make it appear in the formula entry to make it appear in the formula entry.
3. The third, the **parameters** field, allows the user to define parameters that are separated by `,`. For example, if you set parameters with the expression `factor=0.8, K=0.2`, all the occurences of `factor` and `K` in the formula above will be replaced by 0.8 and 0.2 respectively. `Ha * factor + OIII * K` would therefore evaluate to `Ha * 0.8 + OIII * 0.2`.
4. The **output** field is reserved for scaling the image within a given range and to enable or not the *Sum exposure time*. This last option gives the user the option of summing or not the exposures of individual images, storing the result in the FITS header. One need to expand the frame before using it.

   Pixel Math rescale box
5. Finally, the **presets** area allows the user to reuse previously saved formulas with the button to the right of the formula areas. One need to expand the frame before using it. Double-click on the formula to copy it to the right entry.

### 10.13.1 Usage

#### Name of variables

By default it is possible to load 10 images simultaneously. Each image is given a variable name starting with I followed by a number from 1 to 10. However, if the loaded image contains the keyword `FILTER`, then the value of the latter

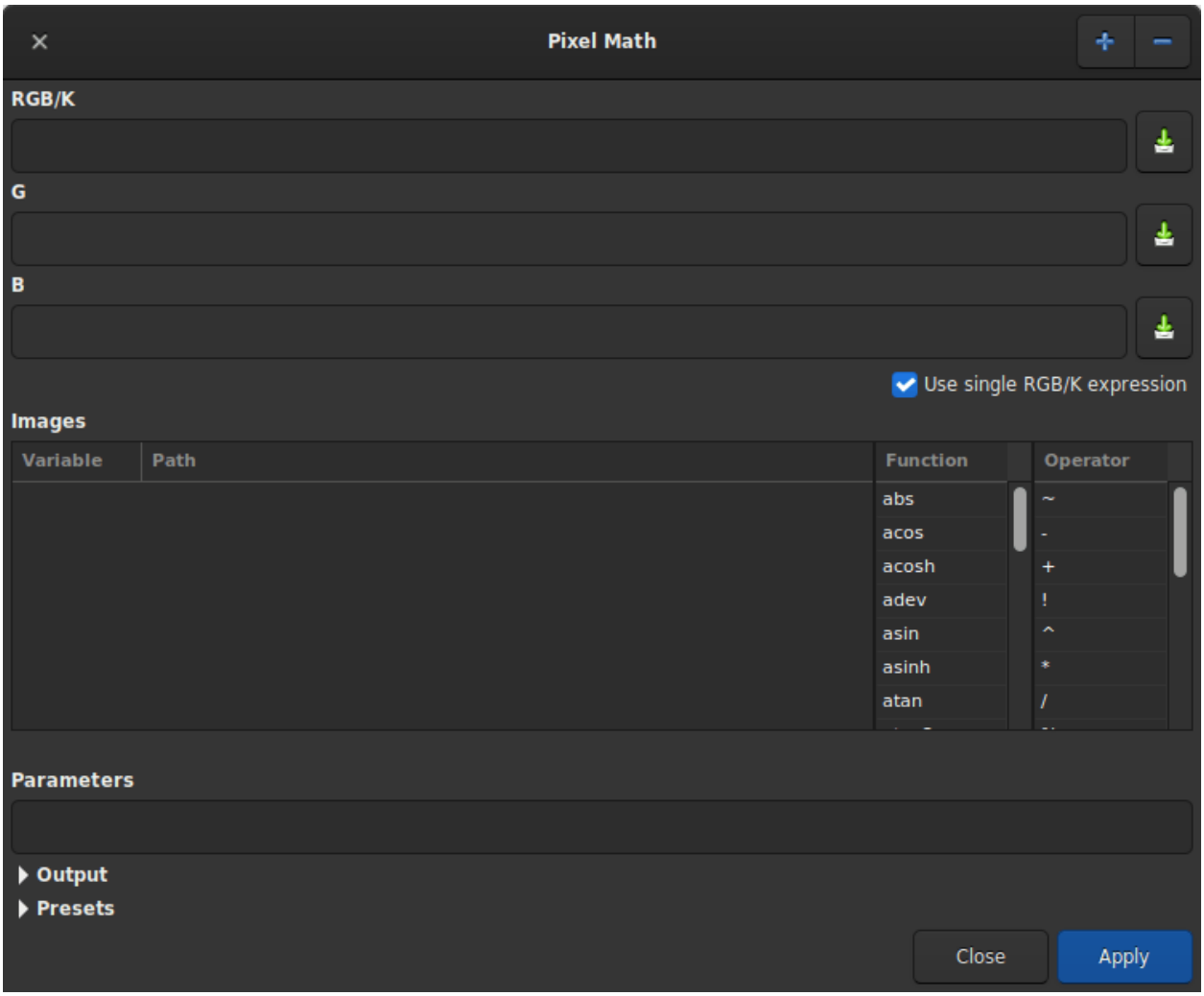



Fig. 87: Pixel Math dialog box as shown at opening

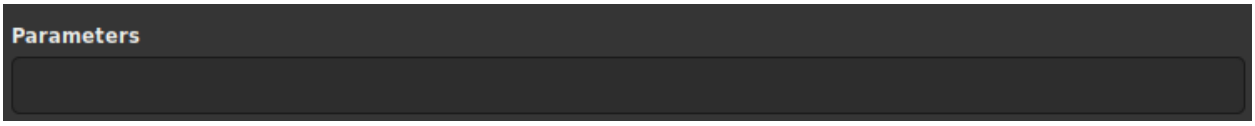



Fig. 88: Pixel Math parameters box

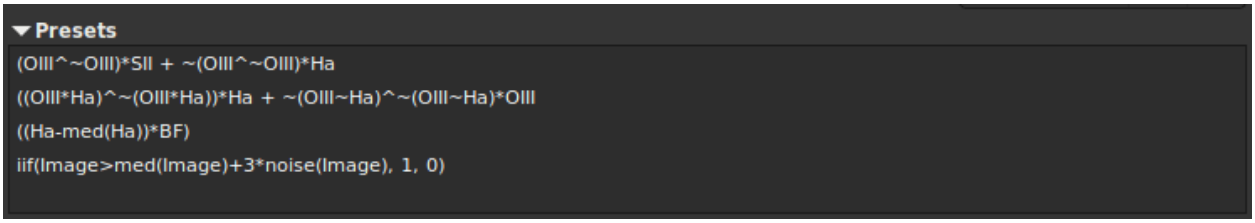



Fig. 89: Pixel Math presets

becomes the default variable name. Of course it is always possible to change it by double clicking on it.

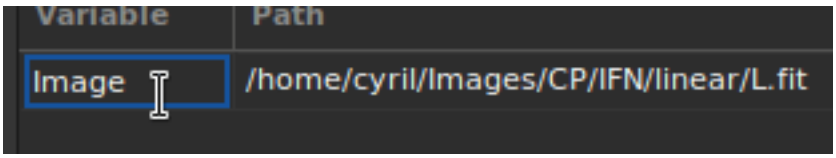



Fig. 90: It is possible to change the name of the variable.

`$T` is a reference to the current image, meaning the image on which the PixelMath operations will be applied.

**Tip**

It is possible to use the currently loaded image by using the `$T` token. Note, however, that unlike other programs, the expression `$T[i]`, with `i=0,1,2`, is not recognized.

### Examples

Let's take a monochrome image of galaxies This is a linear data seen through the autostretch view.

The following expression:

```
iif(Image>med(Image)+3*noise(Image), 1, 0)
```

will produce a star mask.

**Siril command line**

```
pm "expression" [-rescale [low] [high]] [-nosum]
```

This command evaluates the expression given in argument as in PixelMath tool. The full expression must be between double quotes and variables (that are image names, without extension, located in the working directory in that case) must be surrounded by the token $, e.g. "$image1$ * 0.5 + $image2$ * 0.5". A maximum of 10 images can be used in the expression.

Image can be rescaled with the option **-rescale** followed by **low** and **high** values in the range [0, 1]. If no low and high values are provided, default values are set to 0 and 1. Another optional argument, **-nosum** tells Siril not to sum exposure times. This impacts FITS keywords such as LIVETIME and STACKCNT

Fig. 91: Original image.

Fig. 92: After the formula above.

### 10.13.2 Functions

There are two types of functions. Those that apply directly to the pixels and those that apply to the entire image (such as the statistics functions).

Table 1: Pixel oriented functions

| Function | Use case | Definition |
|---|---|---|
| abs | abs ( x ) | Absolute value of x. |
| acos | acos ( x ) | Arc cosine of x. |
| acosh | acosh ( x ) | Hyperbolic arc cosine of x. |
| asin | asin ( x ) | Arc sine of x. |
| asinh | asinh ( x ) | Hyperbolic arc sine of x. |
| atan | atan ( x ) | Arc tangent of x. |
| atan2 | atan2 ( y, x ) | Arc tangent of y/x. |
| atanh | atanh ( x ) | Hyperbolic arc tangent of x. |
| ceil | ceil ( x ) | Round x upwards to the nearest integer. |
| cos | cos ( x ) | Cosine of x. |
| cosh | cosh ( x ) | Hyperbolic cosine of x. |
| e | e | The constant e=2.718282... |
| exp | exp ( x ) | Exponential function. |
| fac | fac( x ) | Factorial function. |
| iif | iif( cond, expr_true, expr_false ) | Conditional function (or inline if function).<br>Returns *expr_true* if *cond* evaluates to nonzero.<br>Returns *expr_false* if *cond* evaluates to zero. |
| floor | floor ( x ) | Highest integer less than or equal to x. |
| ln | ln ( x ) | Natural logarithm of x. |
| log | log ( x ) | Base-10 logarithm of x. |
| log10 | log10 ( x ) | Base-10 logarithm of x. |
| log2 | log2 ( x ) | Base-2 logarithm of x. |
| max | max ( x, y ) | Maximum function. |
| min | min ( x, y ) | Minimum function. |
| mtf | mtf ( m, x ) | Midtones Transfer Function (MTF) of x for a midtones balance parameter m in the [0, 1] range. |
| ncr | ncr ( x, y ) | Combinations function. |
| npr | npr ( x, y ) | Permutations function. |
| pi | pi | The constant =3.141592... |
| pow | pow ( x, y ) | Exponentiation function. |
| sign | sign ( x ) | Sign of x:<br>$+1$ if $x > 0$<br>$1$ if $x < 0$<br>$0$ if $x = 0$. |
| sin | sin ( x ) | Sine of x. |
| sinh | sinh ( x ) | Hyperbolic sine of x. |
| sqrt | sqrt ( x ) | Square root of x. |
| tan | tan ( x ) | Tangent of x. |
| tanh | tanh ( x ) | Hyperbolic tangent of x. |
| trunc | trunc ( x ) | Truncated integer part of x. |

Table 2: Statistics functions

| Function | Use case | Definition |
|---|---|---|
| adev | adev ( Image ) | Average absolute deviation of the image. |
| bwmv | bwmv ( Image ) | Biweight midvariance of the image. |
| height | height ( Image ) | Height in pixels of the specified image. |
| mad | mad ( Image ) | Median absolute deviation of the image. The use of mdev is also possible. |
| max | max ( Image ) | Pixel maximum of the image. |
| mean | mean ( Image ) | Mean of the image. |
| med | med ( Image ) | Median of the image. The use of median is also possible. |
| min | min ( Image ) | Pixel minimum of the image. |
| noise | noise ( Image ) | Estimation of Gaussian noise in the image. |
| sdev | sdev ( Image ) | Standard deviation of the image. |
| width | width ( Image ) | Width in pixels of the specified image. |

### 10.13.3 Operators

Table 3: Operators

| Operator | Use case | Definition |
|---|---|---|
| ~ | ~x | Pixel Inversion operator. |
| - | -x | Unary Minus operator (sign change). |
| + | +x | Unary Plus operator. |
| ! | !x | Logical NOT operator. |
| ^ | x ^ y | Exponentiation operator. |
| * | x * y | Multiplication operator. |
| / | x / y | Division operator. |
| % | x % y | Modulus operator. |
| + | x + y | Addition operator. |
| - | x - y | Subtraction operator. |
| < | x < y | Less Than relational operator. |
| <= | x <= y | Less Than Or Equal relational operator. |
| > | x > y | Greater Than relational operator. |
| >= | x >= y | Greater Than Or Equal relational operator. |
| == | x == y | Equal To relational operator. |
| != | x != y | Not Equal To relational operator. |
| && | x && y | Logical AND operator. |
| \|\| | x \|\| y | Logical OR operator. |

CHAPTER

ELEVEN

# ASTROMETRY

Astrometry is the science dealing with the positions and motions of celestial objects. Astrometry is essential in modern astrophotography where capture software like N.I.N.A, Ekos, APT or others, plate solve the images in order to obtain an astrometric solution, meaning they will precisely know the position of the frame with regards to the sky. Astrometry can also be used at processing stage, like in photometric color calibration tool for example.

## 11.1 Platesolving

The platesolving is a major step in astronomical image processing. It allows images to be associated with celestial coordinates, giving the ability to know what object is in the observed field of view. Many of Siril's tools, such as the Spectrophotometric or Photometric Color Calibration (SPCC or PCC) tools, need to know the coordinates of the image with sufficient accuracy in order to work.

Astrometry in Siril can be performed in two different ways:

- Using the dedicated tool accessible via the menu *Tools* → *Astrometry* → *Image Plate Solver*, or using the shortcut `Ctrl` + `Shift` + `A`.
- Using the `platesolve` command, introduced in Siril 1.2.

**Siril command line**

```
platesolve [-force] [image_center_coords] [-focal=] [-pixelsize=]
platesolve ... [-noflip] [-downscale] [-order=] [-radius=] [-disto=]
platesolve ... [-limitmag=[+-]] [-catalog=] [-nocrop]
platesolve ... [-localasnet [-blindpos] [-blindres]]
```

Plate solve the loaded image.

If the image has already been plate solved nothing will be done, unless the **-force** argument is passed to force a new solve. If WCS or other image metadata is erroneous or missing, arguments must be passed:

the approximate image center coordinates can be provided in decimal degrees or degree/hour minute second values (J2000 with colon separators), with right ascension and declination values separated by a comma or a space (not mandatory for astrometry.net).

focal length and pixel size can be passed with **-focal=** (in mm) and **-pixelsize=** (in microns), overriding values from image and settings. See also options to solve blindly with local Astrometry.net

Unless **-noflip** is specified, if the image is detected as being upside-down, it will be flipped.

For faster star detection in big images, downsampling the image is possible with **-downscale**.

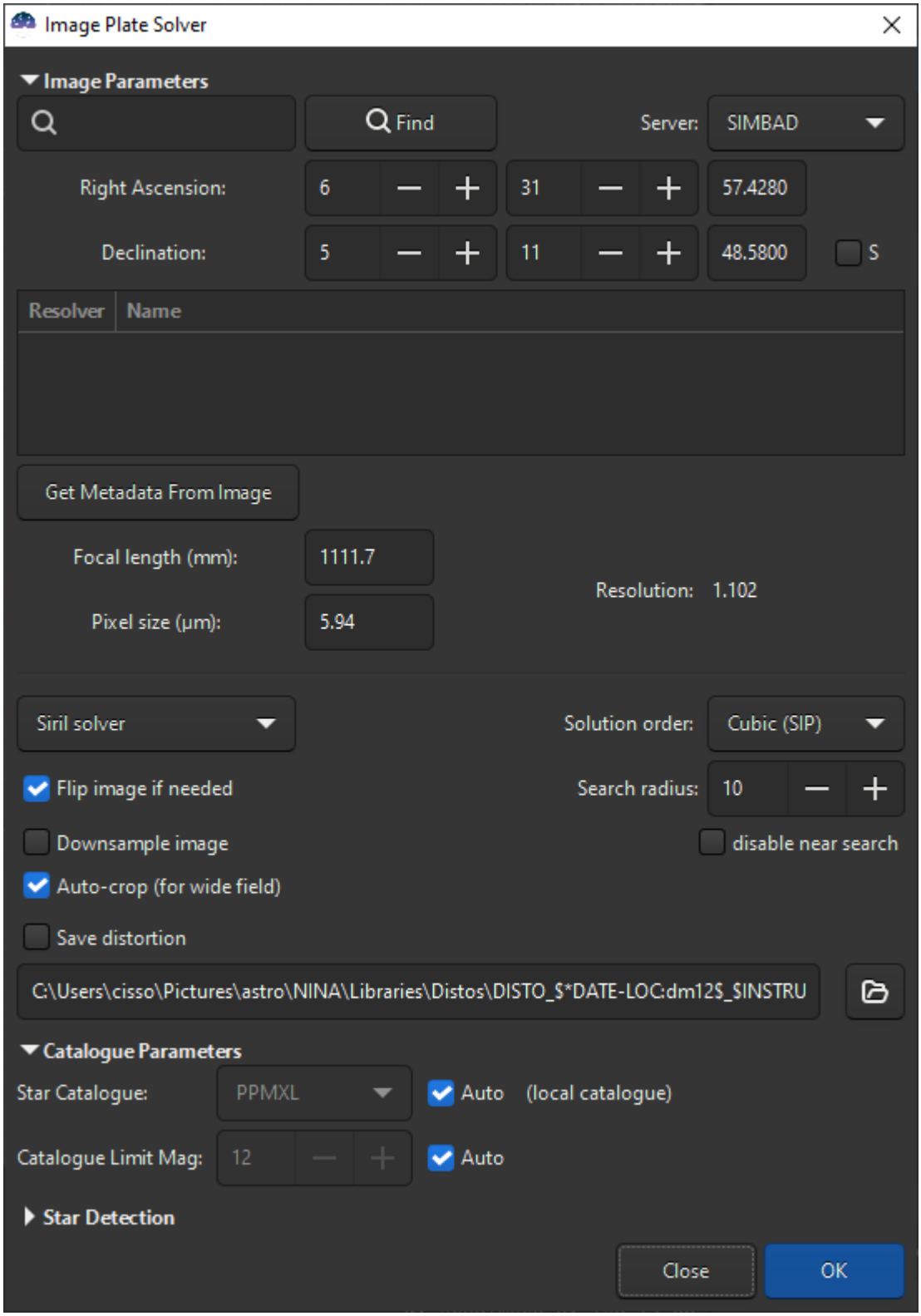



Fig. 1: Platesolving dialog

The solve can account for distortions using SIP convention with polynomials up to order 5. Default value is taken form the astrometry preferences. This can be changed with the option **-order=** giving a value between 1 and 5.

When using Siril solver local catalogues or with local Astrometry.net, if the initial solve is not successful, the solver will search for a solution within a cone of radius specified with **-radius=** option. If no value is passed, the search radius is taken from the astrometry preferences. Siril near search can be disabled by passing a value of 0. (cannot be disabled for Astrometry.net).

You can save the current solution as a distortion file with the option **-disto=**.

Images can be either plate solved by Siril using a star catalog and the global registration algorithm or by astrometry.net's local solve-field command (enabled with **-localasnet**).

**Siril platesolver options:**

The limit magnitude of stars used for plate solving is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.

The choice of the star catalog is automatic unless the **-catalog=** option is passed: if local catalogs are installed, they are used, otherwise the choice is based on the field of view and limit magnitude. If the option is passed, it forces the use of the catalog given in argument, with possible values: tycho2, nomad, localgaia, gaia, ppmxl, brightstars, apass.

If the computed field of view is larger than 5 degrees, star detection will be bounded to a cropped area around the center of the image unless **-nocrop** option is passed.

**Astrometry.net solver options:**

Passing options **-blindpos** and/or **-blindres** enables to solve blindly for position and for resolution respectively. You can use these when solving an image with a completely unknown location and sampling

Since version 1.2, plate solving can be done by two different algorithms. The first was the only one in Siril until this version, it is based on the global registration star matching algorithm, trying to register images onto a virtual image of a catalog with the same field of view. The second is new, it is using an external program called `solve-field` from the Astrometry.net suite, installed locally. For Windows platforms, the simplest way to get it is to use ansvr.

Since version 1.3, Siril internal solver will also look within a cone around the initial target coordinates if it did not find a match. This is only available when using *local star catalogues*. The log will show this information when that happens:

```
Initial solve failed
Attempting a near solve with a radius of 10.0 degrees
```

Astrometric solutions require a few parameters to be found, like image sampling. The window of the tool helps gathering those parameters, we will now see how to fill them correctly.

### 11.1.1 Image parameters

#### Target coordinates

Finding an astrometric solution is easier and faster when we roughly know were we are looking. Siril's plate solver, as it's comparing a catalog with the image, needs to know approximately the position of the center of the image to get the catalog excerpt. Astrometry.net has all the catalogs it needs locally, so it can browse through all of it to find a solution, but it is of course much faster to tell it where to start.

Acquisition software often also control the telescope nowadays and should know the approximate coordinates where the image was taken. In that case, using a FITS format, these coordinates will be provided in the image metadata, the

FITS header. This is not always the case, and clearly not the case when RAW DSLR images are created instead of FITS.

When opening the plate solver window, the current image's metadata is loaded and displayed in the window. If no coordinates appear at the top, or if RA and Dec remain zero, some user input is needed. If you don't know at all what the image is, use a blind solve with astrometry.net. Otherwise, provide equatorial J2000 coordinates corresponding to as close as the center of the image as possible, either by filling the fields if you already know the coordinates, or by making a query with an object name (not yet possible from the command).

The text field at the top left of the window is the search field, its goal being to convert an object name to its coordinates. Pressing `ENTER` or clicking the *Find* button will first search the object in the local annotation catalogues. If not found, a Web request will be made to obtain its coordinates. Several results may be found for the entered name, they will be displayed in the list below. Selecting one updates the coordinates at the top, the first is selected by default.

It is also possible to choose the server on which you want to execute the query, it does not change the results much, but sometimes one of them can be online, so others would act as a backup, between CDS, VizieR and SIMBAD (default).

**Note**

If the object is not found, please try with the full name or with the name from a catalogue. The annotation catalogues contain a few common names, the online services too, but not all, and they don't find partial answers. For example, for the Bubble Nebula, please enter `NGC 7635` or `bubble nebula`, not `bubble`.

The coordinate fields are filled in automatically, but it is possible to define your own. Don't forget to check the *S* box if the object you are looking for is located in the southern hemisphere of the sky (negative declinations).

### Image sampling

Image sampling is the most important parameter for plate solving. Given in arcseconds per pixel in our case, it represents how much zoomed on the sky the image is, so how wide a field to search for.

It is derived from two parameters: focal length and pixel size. They are often available in the image metadata as well. When not available from the image, the values stored in the settings are used. The values of the images and of the preferences can be set using the *Information* dialog. In any case, check the displayed value before plate solving and correct if needed. If an astrometric solution is found, the default focal length and pixel size will be overwritten. This behavior can be disabled in the settings.

**Warning**

If binning was used, it should be specified in the FITS header, but this can take two forms: the pixel size can remain the same and the binning multiplier should be used to compute the sampling, or the pixel size is already multiplied by the acquisition software. Depending on the case you are facing, either of the forms can be chosen from the preferences or from the *Information* window.

Pixel size is given in the specification of astronomical cameras, and can generally be found on the Web for DSLR or other cameras. The number of sensors is limited and most of them are known.

Focal length depends on the main instrument, but also on backfocus and correcting or zooming lenses used. Give a value as close as what you believe the effective focal to be, if an astrometric solution is found, the computed focal length will be given in the results and you will be able to reuse that in your acquisition software and for future uses of the tool.

When either of the fields is updated, the sampling is recomputed and displayed in the window (called 'resolution' here). Make sure the value is as close to reality as possible.

> **Tip**
>
> Data written in orange color in the GUI indicates values which could not be retrieved from the image header. It does not mean they are wrong (they could have been loaded from the preferences values and be valid) or mandatory (you could blind-solve), this color is just here to differentiate from values read from the header.

### 11.1.2 Solver parameters

Since Siril 1.2, Siril can use two different solvers, its internal solver and *Astrometry.net local installation*. The interface differs depending on whether one or the other is selected in the dedicated drop-down list.

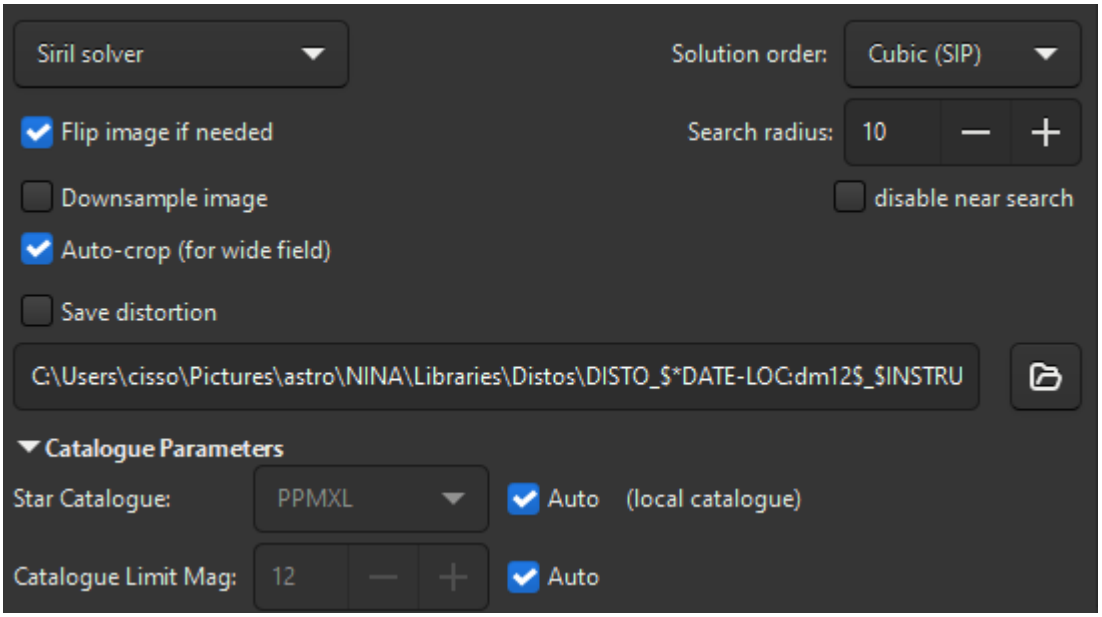



Fig. 2: Siril internal solver options

#### Distortions

The option *Solution order*, selected via the drop-down list, specifies the order of the platesolving solution. When selecting `Linear`, the solver will try to fit a solution assuming there are no distortions in the image (i.e. the field is optically flat). However, this assumption may not be true in presence of optical aberrations (wrong backfocus, no field-flattener etc...). Since version 1.3, the platesolver can try to fit up to fifth-order polynomial distortions, following the SIP convention. By default, the platesolver uses `Cubic (SIP)` distortions, which should fit most use cases. This default can be changed in the *Preferences*. This option is available with both solvers.

You can also choose to save distortions to a wcs file by ticking the *Save distortion* box. Use the filechooser below to specify its path. If you have defined a master distortion path in the *preferences*, its path will be shown by default. This file can then be used to undistort images during *Global Registration* and *2pass Registration*

#### Search radius

When using *local star catalogues* are installed or when using *Astrometry.net*, a cone around the target position will be searched. The size of this cone in degrees can be changed using the *Search radius* control, which default value can be changed in the *Preferences*. For Siril solver, this feature can be disabled by ticking the *disable near search* box.

### Astrometry.net blind solve

When Astrometry.net solver is selected, two additional options are available:

1. *Ignore position (blind solve)* enables to ignore the position specified in the Target coordinates control.
2. *Ignore sampling (blind solve)* enables to ignore the resolution computed from the pixel size and focal length.

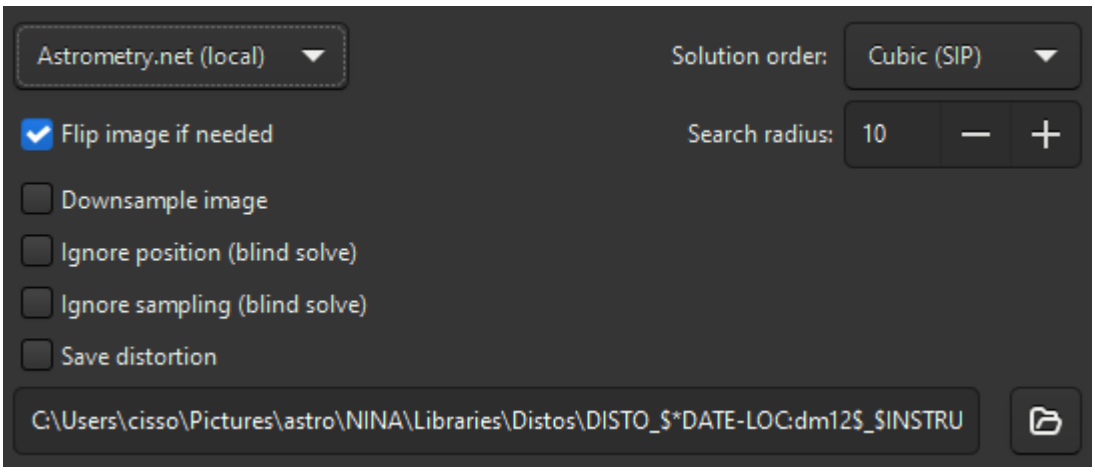



Fig. 3: Astrometry.net solver options

Use these two options together if the location and resolution of the image are completely unknown.

> **Warning**
>
> There is no magic happening here. In order to solve for any field of view, you will need to have the necessary *indexes* installed that cover the actual field of view of the image being solved.

### Other parameters

Finally, there are three toggle buttons at the bottom of the frame:

1. The option *Downsample image* downsamples the input image to speed-up star detection in it. The downside is that it may not find enough stars or give a less accurate astrometric solution. The size of the output image remains unchanged.
2. If the image is detected as being upside-down by the astrometric solution, with the option *Flip image if needed* enabled, it will be flipped at the end. This can be useful depending on the capture software, if the image has not the right orientation when it is displayed in Siril (see more *explanations*).
3. When the option *Auto-crop (for wide field)* is applied, it performs a platesolve only in the center of the image. This is only done with wide field images (larger than 5 degrees) where distortions away from the center are important enough to fool the tool. Ignored for astrometry.net solves.

## 11.1.3 Catalogue parameters

This section is relevant for Siril internal solver only. Several online catalogues can be used and also two catalogues that can be installed locally for faster and more reliable operation.

### Using online catalogues

By default, this section is insensitive because everything is set to automatic. By unchecking the auto box, however, it is possible to choose the online catalog used for the platesolve, which may depend on the resolution of the image. The choice is done between:

- TYCHO2, a catalogue containing positions, proper motions, and two-color photometric data for 2,539,913 of the brightest stars in the Milky Way.
- NOMAD, a merge of data from the Hipparcos, Tycho-2, UCAC2, Yellow-Blue 6, and USNO-B catalogs for astrometry and optical photometry, supplemented by 2MASS near-infrared. The almost 100 GB dataset contains astrometric and photometric data for about 1.1 billion stars.
- Gaia DR3, released on 13 June 2022. The five-parameter astrometric solution, positions on the sky (, ), parallaxes, and proper motions, are given for around 1.46 billion sources, with a limiting magnitude of G = 21. This represents the state of the art in accurate astrometry data and provides spectral data for a wide range of sources which is used in SPCC.
- PPMXL, a catalog of positions, proper motions, 2MASS- and optical photometry of 900 million stars and galaxies.
- Bright Stars, a star catalogue that lists all stars of stellar magnitude 6.5 or brighter, which is roughly every star visible to the naked eye from Earth. The catalog contains 9,110 objects.
- APASS, a star catalogue that lists all stars of stellar magnitude 6.5 or brighter, which is roughly every star visible to the naked eye from Earth. The catalog contains 9,110 objects.

> **Note**
>
> An internet connection is required to use these online catalogs.

All these catalogs are made available through VizieR catalogue access tool, CDS, Strasbourg, France (DOI:10.26093/cds/vizier). The original description of the VizieR service was published in 2000, A&AS 143, 23.

The *Catalogue Limit Mag* is an option that allows you to limit the magnitude of the stars retrieved in the catalog. The automatic value is calculated from the image resolution.

### Using the local KStars catalogues (NOMAD)

With version 1.1, starting in June of 2022, it was possible to rely on a locally installed star catalogue, for disconnected or more resilient operation. The star catalogue we found to be the most adapted to our needs is the one from KStars. It is in fact composed of four catalogues (documented here in KStars), two of them not being directly distributed in the base KStars installation files:

- **namedstars.dat**, the brightest stars, all of them have names
- **unnamedstars.dat**, also bright stars, but down to magnitude 8
- **deepstars.dat**, fainter stars extracted from The Tycho-2 Catalogue of the 2.5 Million Brightest Stars, down to magnitude 12.5
- **USNO-NOMAD-1e8.dat**, an extract of the huge NOMAD catalogue limited to B-V photometric information and star proper motion in a compact binary form, down to magnitude 18.

**This catalogue can be used for plate solving and for PCC.**

When comparing these catalogues with the online NOMAD, we can easily see that many stars are missing. If not enough are found for your narrow field, you should still use the remote queries. A nice thing to check when the catalogues are installed is highlighting which stars of the image will be used for the PCC, those available with photometric information in the catalogues, using the *conesearch* command.

```
conesearch -phot
```

### Using the local Gaia DR3 Catalogues

In version 1.4 support for a local extract of the Gaia DR3 catalogue has been added. This *astrometric* Gaia extract differs from the other catalogues in that it is focused on astrometry as applied to plate solving, so instead of defining a limiting magnitude the catalogue has been designed to provide an even coverage of stars in each level 8 HEALPixel (you can read more about HEALPixels here). This avoids including excessive numbers of stars in densely populated parts of the sky and ensures that sufficient fainter stars are included in emptier HEALPixels to support accurate plate solving.

The default catalogue will be selected at startup based on two criteria:

- If a local catalogue is available it is preferred to the remote equivalent
- If a Gaia catalogue (local or remote) is available it is preferred to NOMAD

**Tip**

Although the offline Gaia DR3 extract can be used with the *conesearch* command (by specifying the catalogue name *localgaia*) it is not really designed for this purpose. In busy regions of the sky the limiting magnitude will be relatively bright and therefore many stars visible in your images will not be contained in the catalogue whatever limiting magnitude you set. For annotation cone searches in such regions you will be better served by specifying the catalogue name *gaia* instead, which will conduct a remote cone search using the Vizier mirror of the Gaia DR3 archive. This is both fast (for a remote query) and comprehensive.

**This catalogue can be used only for plate solving.** A *photometric* Gaia extract also exists, and can be used for the SPCC, it is documented on the *SPCC page*.

### Download KStars NOMAD

The first two files are available in KStars source, the Tycho-2 catalogue from a debian package and the NOMAD catalogue from KStars files too, as documented in this small article for KStars installation. It is has multiple worldwide mirrors as indicated in the articles.

To make things easier to Siril users, and possibly to KStars users too, we redistribute the four files in a single place, and in a more compressed format. With the LZMA algorithm (used by xz or 7zip), the file size is 1.0GB instead of the 1.4GB with the original gzip file. So those *.xz* files are compressed, make sure you uncompress them with a suitable software before copying them.

Direct download links are available here (right click on each file name on the left and save the links).

In case the link is unavailable and to make it faster from anywhere, it is also distributed with bittorrent, using this torrent file or the following magnet link.

### Download Astrometric Gaia DR3

The Gaia DR3 local astrometric extract is available from Zenodo It comes with a sha256 checksum for the compressed archive.

Full details, plus a citation reference and a checksum for the uncompressed archive for extra paranoia, can be found at the Zenodo record. The specification of the Gaia DR3 catalogue extracts and their file format is documented here (PDF).

> **Tip**
>
> For convenience, an offline version of the Gaia DR3 astrometric catalogue can be installed using the Catalogue Installer script in the *Scripts* menu. This catalogue takes up 1.5GB once installed compared with 2.1GB for the KStars NOMAD catalogue and provides higher precision astrometry and even density of stars in all regions, for more reliable and accurate plate solving. **It is strongly recommended to install and use this catalogue for all astrometry purposes within Siril**.

### Installation in Siril

The files can be put anywhere and their paths given to Siril in the settings, but there is a default location for the four files: `~/.local/share/kstars/` on Linux. They can be linked there to avoid unnecessary copies. Settings can be changed from the command line now, using the *set* command.

When available and readable, Siril will not use the Web service to retrieve astrometric or photometric data. See the messages in the log tab or on the console to verify that the catalogue files are used as expected.

From Siril 1.4.0 onwards, Siril will first look in *local annotation catalogs* to find the coordinates of an object passed in the platesolving dialog, to locate the center of the image. This means that, provided you have the local star catalogues installed, you can solve your images without Internet connection. Of course, this should only be needed if the acquisition software did not record the target coordinates in the FITS header, or when using SER file format which cannot hold this information.

### Usage

With the addition of the new link between Siril's plate solver and the local catalogue and the new link between Siril's PCC and the local catalogue, a new command `conesearch` was created (from Siril 1.4.0) to display catalog objects in a plate solved image. To display stars that contain photometric information (the B-V index) and can be used for calibration, you can for instance use the following:

```
conesearch -phot
```

This is a good way to verify that the plate solving and the image are aligned, in addition to the object annotation feature (see *annotations*).

### Technical information

For photometry, Siril only uses the B-V index (or, for the Gaia catalogue, the effective temperature field Teff) which gives information about star colour. The three image channels are then scaled to give the best colour representation to all stars in the image.

For more information about the KStar binary file type, see this page and this discussion on kstars-devel and some development notes in Siril here and here.

Sha1 sums for the 4 catalogue files:

```
4642698f4b7b5ea3bd3e9edc7c4df2e6ce9c9f7d  namedstars.dat
53a336a41f0f3949120e9662a465b60160c9d0f7  unnamedstars.dat
d32b78fd1a3f977fa853d829fc44ee0014c2ab53  deepstars.dat
12e663e04cae9e43fc4de62d6eb2c69905ea513f  USNO-NOMAD-1e8.dat
```

Licenses for the 4 catalogue files.

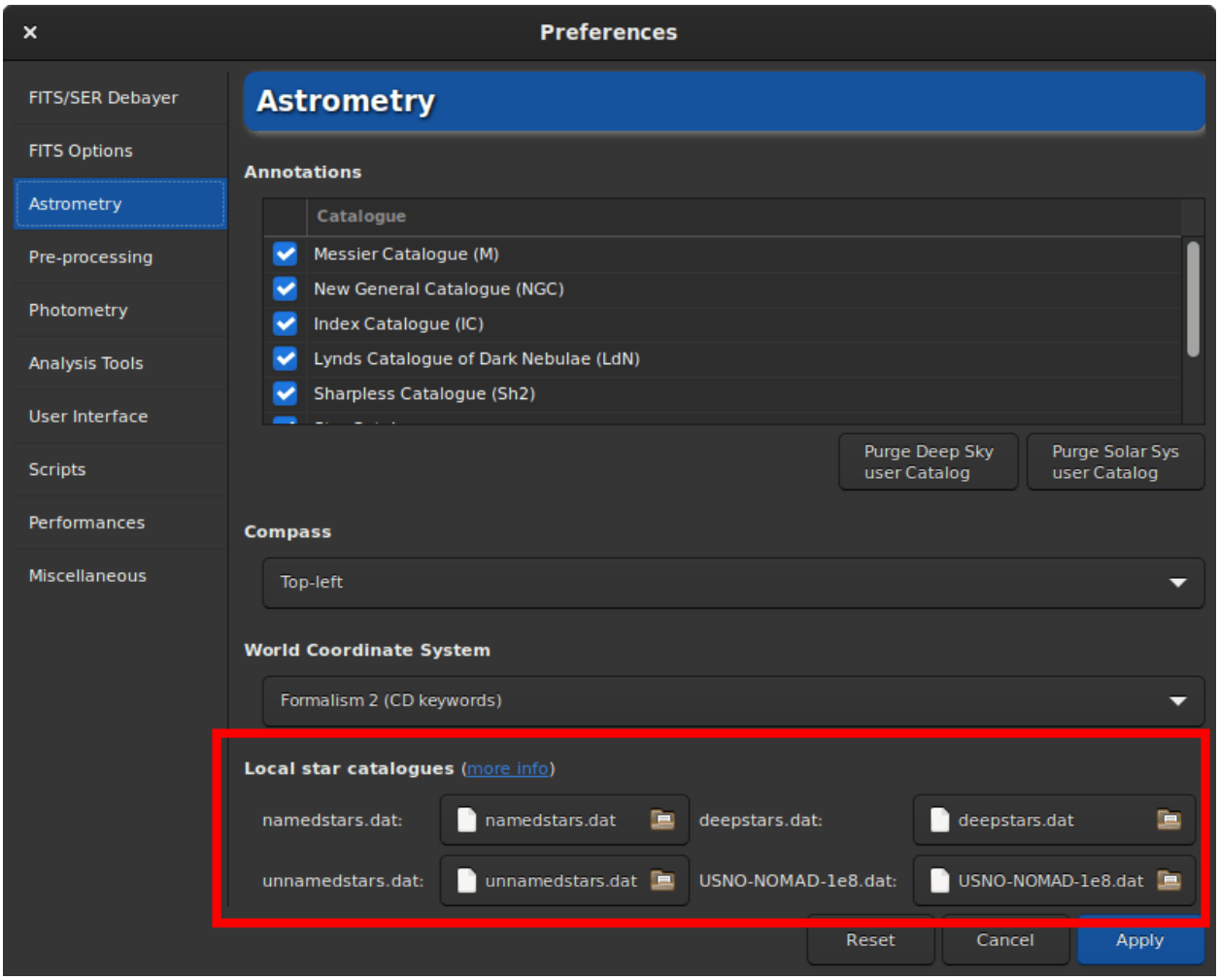



Fig. 4: Preferences with local catalogues

### 11.1.4 Using the local astrometry.net solver

#### Installation

Since version 1.2, `solve-field`, the solver from the astrometry.net suite, can be used by Siril to plate solve images or sequence of images.

For Windows platforms, the simplest way to get it is to use ansvr. If you did not modify the default installation directory, that is *%LOCALAPPDATA%\cygwin_ansvr*, Siril will search for it without additional setup. If you have cygwin and have build astrometry.net from the sources, you must specify the location of cygwin root in the *Preferences*.

For MacOS, please follow these instructions. Install with homebrew and add it to the `PATH`. Also make sure that the program works for the test images, as indicated in the instructions, and outside of Siril.

For non-Windows OS, the executable is expected to be found in the `PATH`.

The use of this tool makes it possible to *blindly* solve images, without a priori knowledge of the area of the sky they contain or its resolution. It's also a good alternative to Siril's plate solver in case it fails, because it's a dedicated and proven tool that also can take field distortion into account.

Default settings should be fine, but can be modified if you really want to, using the *set* command (default values specified between parens) or in the *Astrometry* tab of preferences. How wide the range of allowed scales is (15%), how big the radius of the search from initial coordinates is (10 degrees), the polynomial order for field distortion (0, disabled), removing or not the temporary files (yes), using the result as new default focal length and pixel sizes (yes).

#### Index files

Astrometry.net needs index files to run. We strongly recommend you use the latest index files available from their website, i.e. the 4100 and 5200 series. The field of view of each series is described in their github page. (the official documentation does not yet include this table).

On Unix-based system, you can just follow along the instructions in the documentation.

On Windows, if you are running ansvr, those recent index files will not be made available by the Index Downloader. You can still download them separately and store them where the other index files are kept (would recommend to remove the old files, although it may mess up the Index Downloader).

#### How it works

Just like the internal solver, Siril will proceed with extracting the stars from your images (so as to benefit from internal parallelism) and submit this list of stars to astrometry.net `solve-field`. If you then want astrometry.net to crawl the index in parallel, you will need to specify it through the `astrometry.cfg` file.

### 11.1.5 Solving sequences

When a sequence is loaded, an additional GUI element is present at the bottom of the dialog.

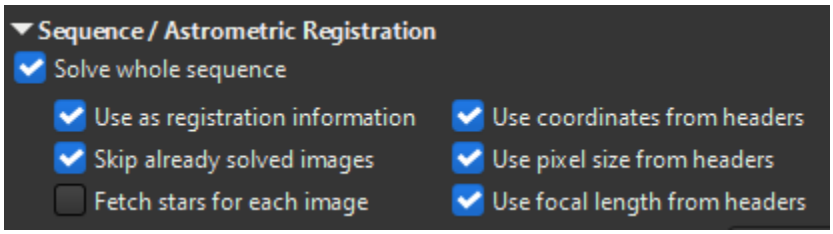



Fig. 5: Sequence astrometry options

You can specify to solve the whole sequence. The images already solved will be solved again unless the *Skip already solved images* box is ticked. To use *Astrometric registration*, you will need to process the whole sequence so that

useful information is stored with the sequence (FWHM, number of stars, background level...), so leave the box *Use as registration information* ticked.

When using Siril solver with local catalogues or when using Astrometry.net, the information contained in the header (if present) will be used to update the image target center and resolution for each image. However, when using Siril solver with online catalogues, a single star catalogue will be downloaded by default to avoid too much network traffic and server requests. If the images do not have too much drift and the same resolution, this is normally sufficient. However, if the images do not have enough overlap or different sampling, you can force downloading one star catalogue per image by ticking the *Fetch stars for each image* box.

Finally, the 3 boxes on the right will enable to control if the platesolver should read target coordinates, pixel size and focal from each image header or the values specified in the dialog.

At the end of the sequence solving, the log will report how many images were solved and if any were skipped.

```
Sequence processing succeeded.
Execution time: 676.35 ms
3 images successfully platesolved out of 3 included
(2 were already solved and skipped)
```

Solving sequences is also available via the command *seqplatesolve*.

> **Note**
>
> When solving FITS sequences or a *FITSEQ* file, the images are directly saved, without creating a new sequence. For FITS sequences, if the sequence was created using symbolic links, the original files are not updated. Instead, the name of the symbolic link is used to create a new FITS file, leaving the original untouched. When solving a *SER* sequence, a new sequence with the prefix `ps_` is created as SER cannot store WCS data.

### 11.1.6 Star detection

By default, the star detection uses the *findstar* algorithm with the current settings. It works very well to find many stars, but in some occasions we would like to detect the stars manually, or simply view which are used. A first step would be to open the *PSF* window and launch star detection, then adjust the settings (see the related documentation *documentation*).

Another approach would be to select the stars one by one by surrounding them with a selection then via a right click, choose *Pick a Star*. The more stars selected, the more likely the algorithm is to succeed.

Then in the astrometry window, expand the star detection section and activate the *Manual detection*. Instead of running *findstar*, it will use the current list of stars.

> **Note**
>
> Detection always uses the green layer for RGB images. For undebayered CFA images, the green layer is extracted on-the-fly and used instead.

### 11.1.7 Understanding the results

When an astrometric solution is found, we can see in the Console tab this kind of messages:

```
Up is -5.26 deg CounterClockWise wrt. N
Resolution:      3.051 arcsec/px
Focal length:  254.21 mm
```

(continues on next page)

(continued from previous page)

```
Pixel size:      3.76 µm
Field of view:    04d 51m 58.27s x 03d 01m 1.21s
Image center: alpha: 21h02m02s, delta: +68°10'48"
Was 119.64 arcmin from initial value
Saved focal length 254.21 and pixel size 3.76 as default values
Flipping image and updating astrometry data.
```

The astrometric solution gives us the J2000 equatorial coordinates of the image center, the projected horizontal and vertical dimension of the image on the sky, the focal length that could give this field for the given pixel size and consequently the actual image sampling, the angle the image makes with the north axis, the field of view and image center. It also tells what was the distance with the initial specified center.

If it fails, check that start coordinates and pixel size are correct and try changing the magnitude, this will change the amount of stars downloaded from the catalogs, and maybe more stars will be identified. If Siril's plate solve won't find a solution, it is still possible to use an external tool to do it, the solution will be written in the *FITS header* either way.

### 11.1.8 Visualizing distortions

To check the validity of the solution, you can use the *conesearch* command. This should display stars positions found from catalogues and inspect if they match with the actual stars in the image.

The two images below shows annotations in the top right corner of an image with significant distortion. The top one is the linear solution while the bottom one has been solved accounting for cubic polynomials.

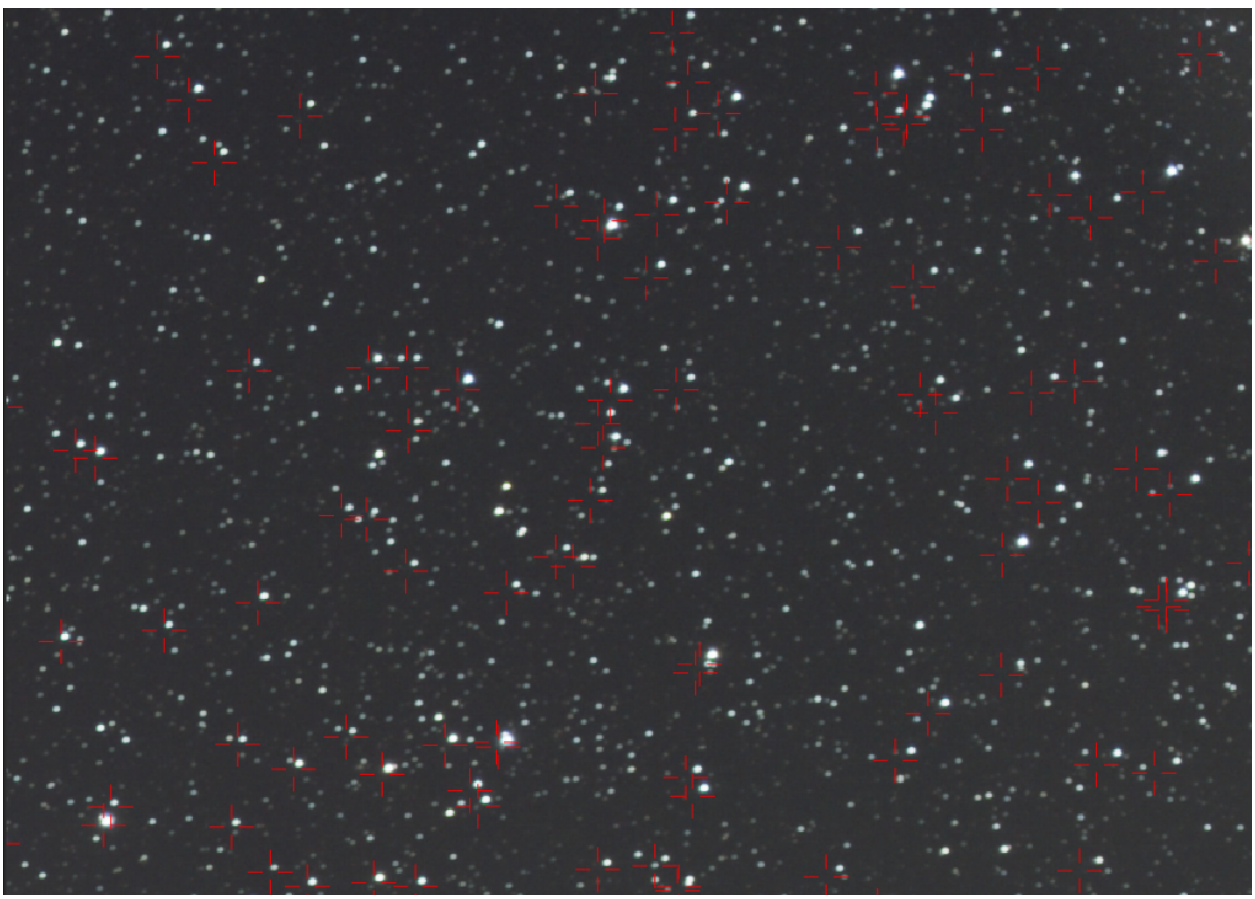


Fig. 6: Stars annotations with linear solution

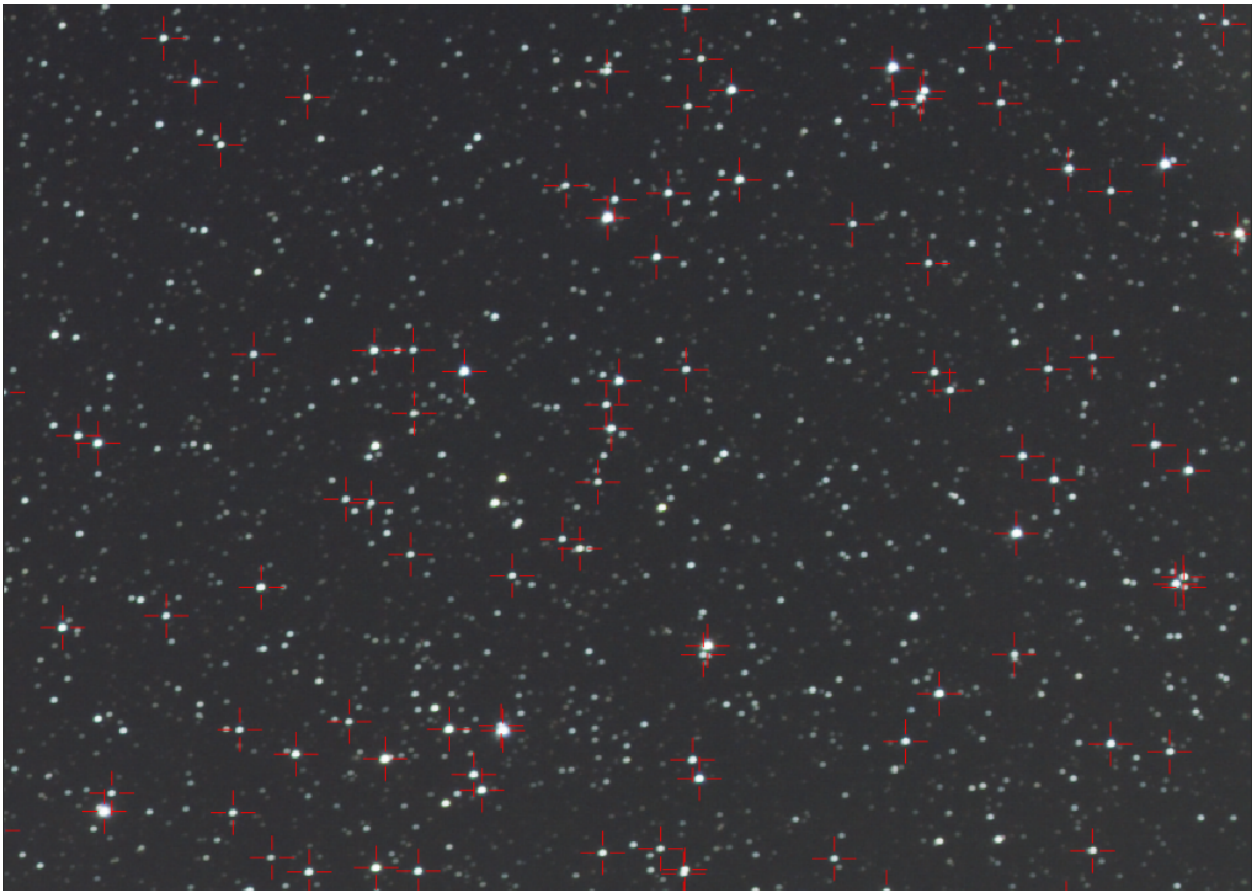


Fig. 7: Stars annotations with cubic solution

If platesolving with distortions is sucessfull, the *disto* command or the *Tools* → *Image Analysis* → *Show Distortions* menu can display a representation of the corrections as an overlay on the image.

**Siril command line**

```
disto [clear]
```

Shows distortion field on a plate-solved image which solution includes distortion terms

Pass option **clear** to disable

An example is shown below.

Fig. 8: Image with distortion overlay

**Sirilpy script**

If you want to visualize the distortion field in 3D, you can use the Python script `Distortion3D.py` in the *Scripts->Python Scripts* menu. You will need to enable getting scripts from the *scripts repository*.

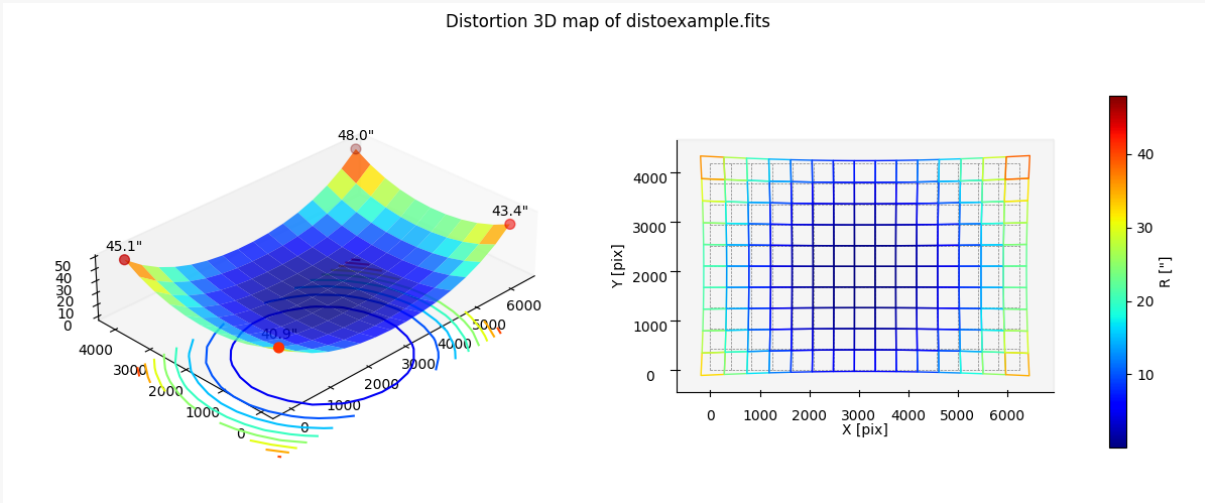



Fig. 9: 3D distortion map

## 11.2 Annotations

Annotations are glyphs displayed on top of images to depict the presence of known sky objects, like galaxies, bright stars and so on. They come from catalogues but can only be displayed on images for which we know which part of the sky they represent, images that have been **plate solved** and contain the world coordinate system (WCS) information in their header, so only *FITS* or *Astro-TIFF* files.

Plate solving, can be done within Siril in the menu *Tools* → *Astrometry* → *Image Plate Solver...* entry, or using external tools like astrometry.net or ASTAP.

When a plate solved image is loaded in Siril, you can see the sky coordinates for the pixel under the mouse pointer displayed at the bottom right corner and the buttons related to annotations become available. The first button toggles on or off object annotations, the second the celestial grid and the compass.

### 11.2.1 Offline annotation catalogues

Siril comes with a predefined list of catalogues for annotations:

- Messier catalogue (M)
- New General Catalogue (NGC)
- Index Catalogue (IC)
- Lynds Catalogue of Dark Nebulae (LdN)
- Sharpless Catalogue (Sh2)
- Star Catalogue (3661 of the brightest stars)
- IAU constellations lines
- IAU constellations names (positions collected from this page)

In addition, 2 *user defined catalogues* can be used:

- User DeepSky Objects Catalogue (DSO)
- User Solar System Objects Catalogue (SSO)

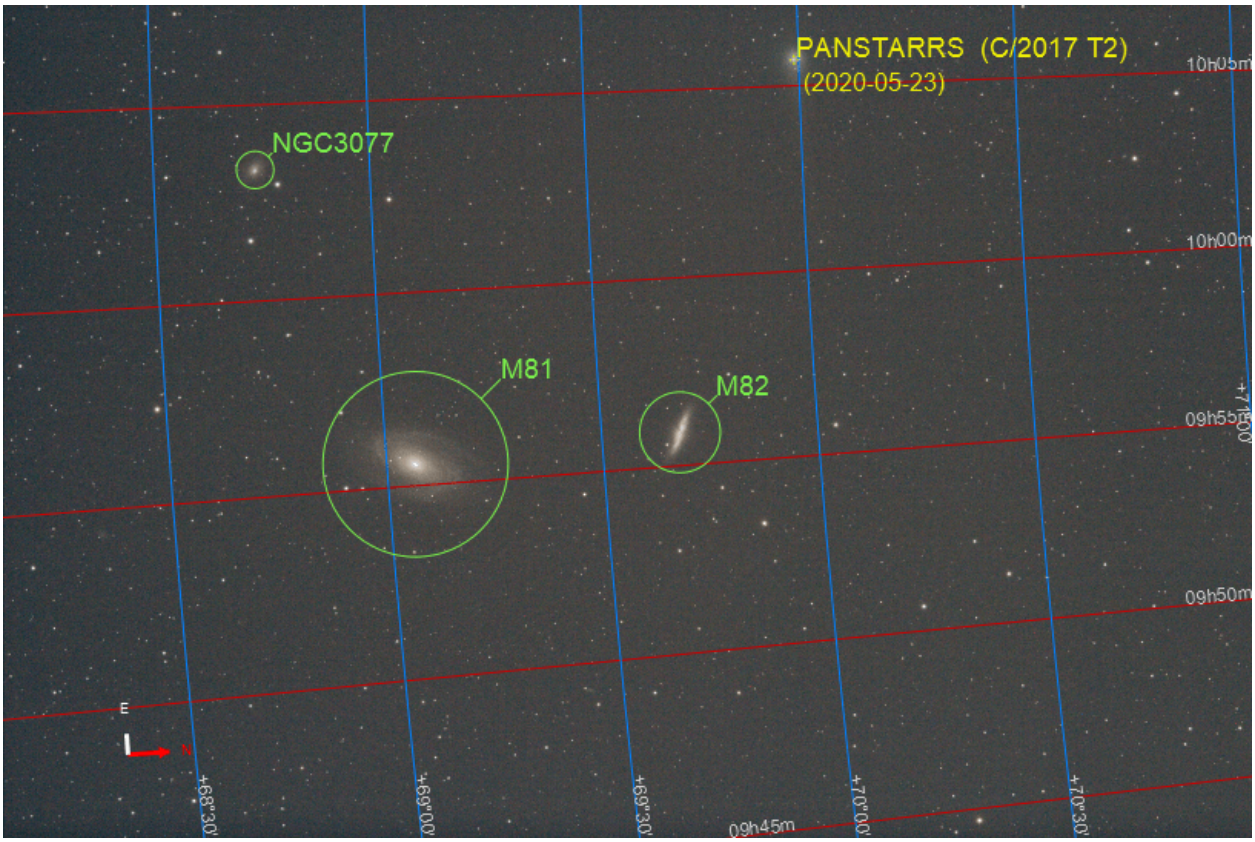



Fig. 10: View of full annotated image

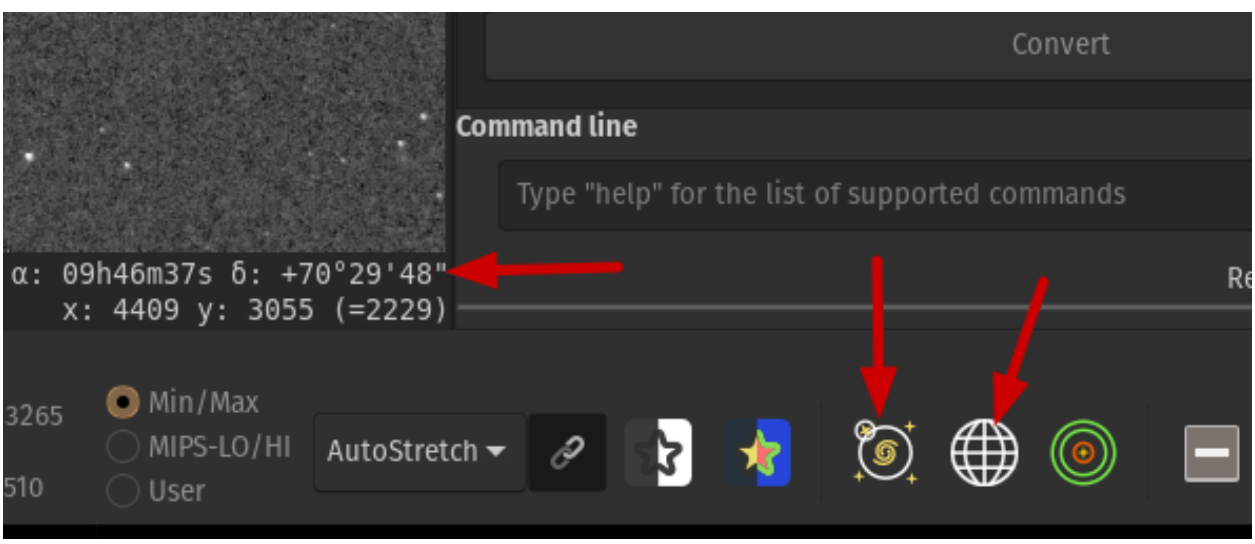



Fig. 11: Buttons for annotations

They are populated with the commands described in the section about *searching for a known object*.

They are stored in the user settings directory. Their location depends on the operating system:

- for *Unix-based* OS they will be in `~/.config/siril/catalogue`
- on *Windows* they are in `%LOCALAPPDATA%\siril\catalogue`.

All of the above catalogues can be enabled/disabled for display in the *Preferences menu* → *Astrometry tab*.

The two *user defined catalogues* can also be purged (*i.e.* deleted) via the appropriate buttons.

A slider on the right side, allows you to easily navigate across the catalogue list.

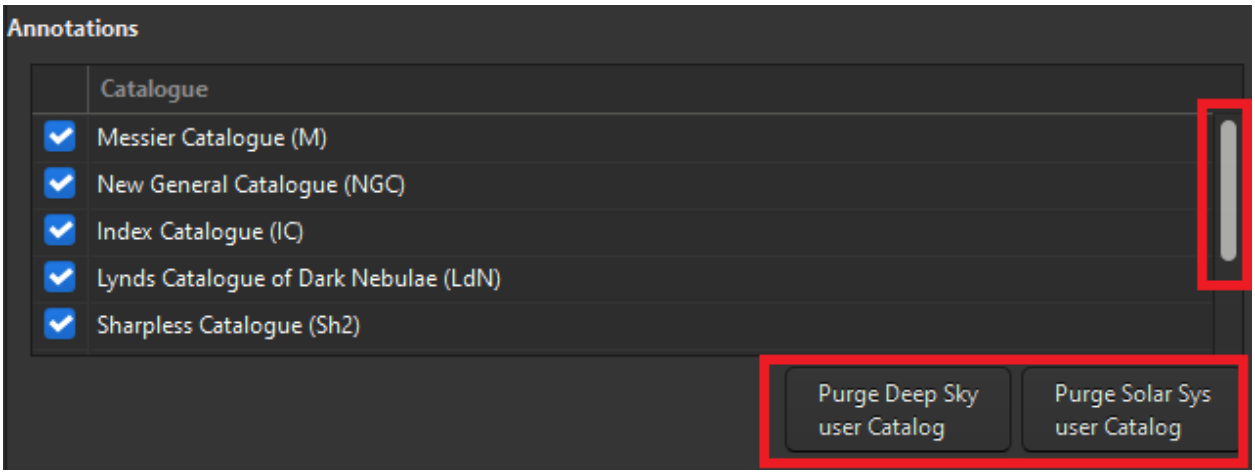



Fig. 12: Catalogue management in Preferences/Astrometry

There's also a simpler way of accessing this list, as well as the button for displaying/hiding catalog annotations. Right-click on the annotations button in the toolbar and the list will be displayed in a popup window.

These annotation catalogues are used primarily for display purposes. Starting from Siril 1.3, they are also used to locate the center of the image for astrometry tool. If the object is found locally, the resolver will be shown as Local. If not, it will fall back using an online resolver.

### 11.2.2 Online annotation catalogues

You may want to query other databases than the ones already shipped with Siril, described in the *offline annotation catalogues* section. This works, again, for plated-solved images only.

Starting from Siril 1.3, this is possible with the command *conesearch*. This new command replaces and expands capabilities previously provided by `nomad` and `solsys` from Siril 1.2.

**Siril command line**

```
conesearch [limit_magnitude] [-cat=] [-phot] [-obscode=] [-tag={on|off}] [-log=
↪{on|off}] [-trix=] [-out=]
```

Displays stars from the local catalog by default for the loaded plate solved image, down to the provided **limit_magnitude** (13 by default for most catalogues, except 14.5 for aavso_chart, 20 for solsys, and omitted for pgc).
An alternate online catalog can be specified with **-cat=**, taking values

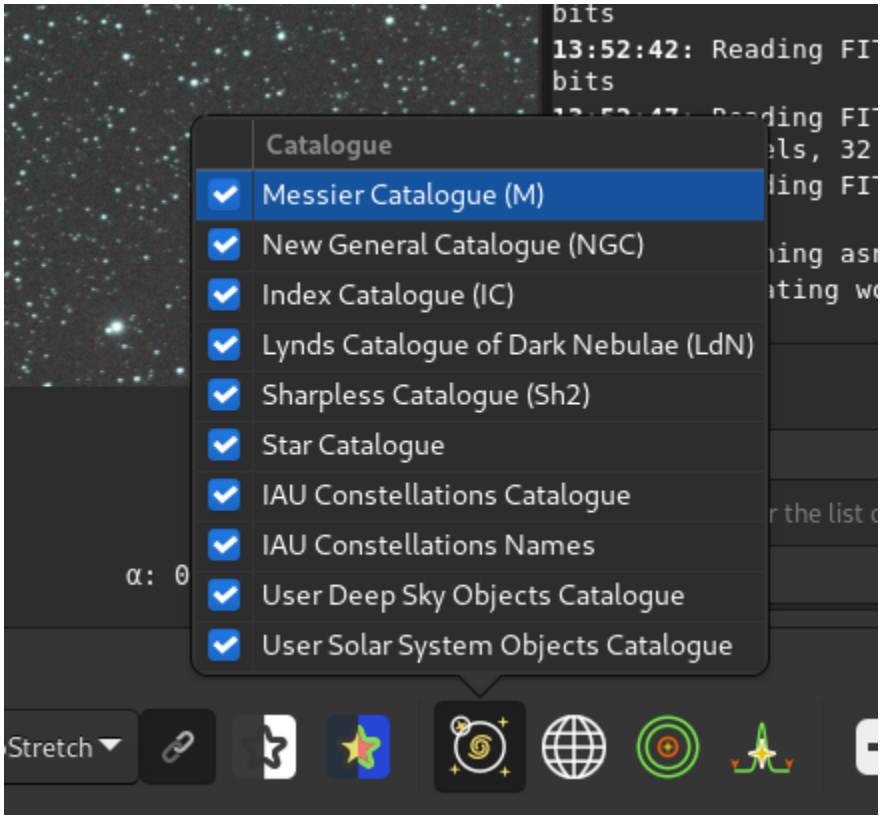



Fig. 13: Catalogue management in the pop-up window triggered by right-clicking on the annotation button.

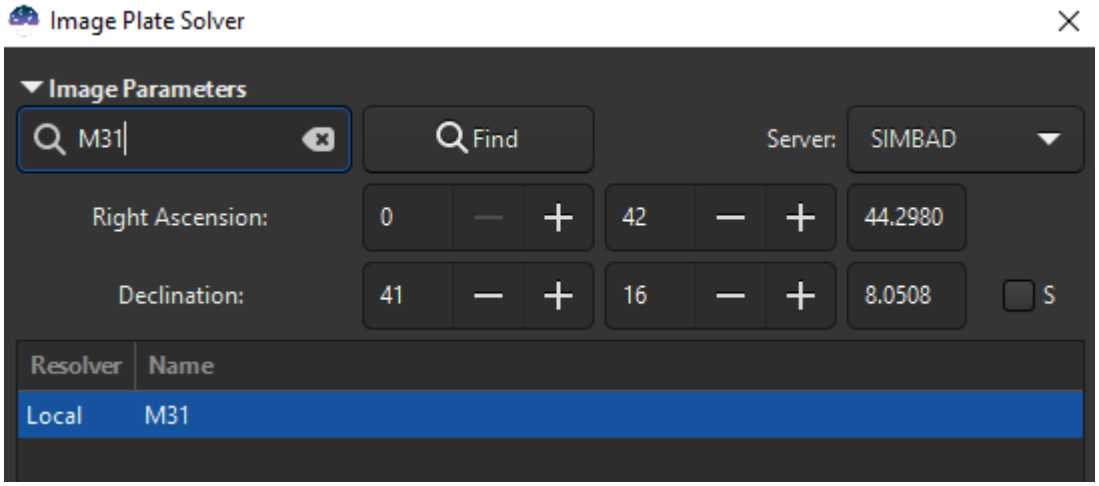



Fig. 14: Object resolved from local annotations catalogues

- for stars: tycho2, nomad, gaia, localgaia, ppmxl, bsc, apass, gcvs, vsx, simbad, aavso_chart
- for exoplanets: exo
- for deep-sky: pgc
- for solar system objects: solsys (closest IAU observatory code can be passed with the argument **-obscode=** for better position accuracy)

For stars catalogues containing photometric data, stars with no B-V information will be kept; they can be excluded by passing **-phot**
The argument **-trix=** can be passed instead of a catalogue followed by a number between 0 and 511 to plot stars contained in local catalogues trixel of level 3 (for dev usage mainly)

Some catalogs (bsc, gcvs, pgc, exo, aavso_chart, varisum and solsys) will also display, by default, names alongside markers in the display (GUI only) and list them in the log. For others with larger number of objects, namely vsx and simbad, the information can also be shown but, as it may clutter the display, it is not activated by default. This behavior can be toggled on/off with the options **-tag=on|off** to display names alongside markers and **-log=on|off** to list the objects in the console log

The list of items that are present in the image can optionally saved to a csv file by passing the argument **-out=**

This command is accessible from the graphical interface via the *Tools* → *Astrometry* → *Annotate...*.

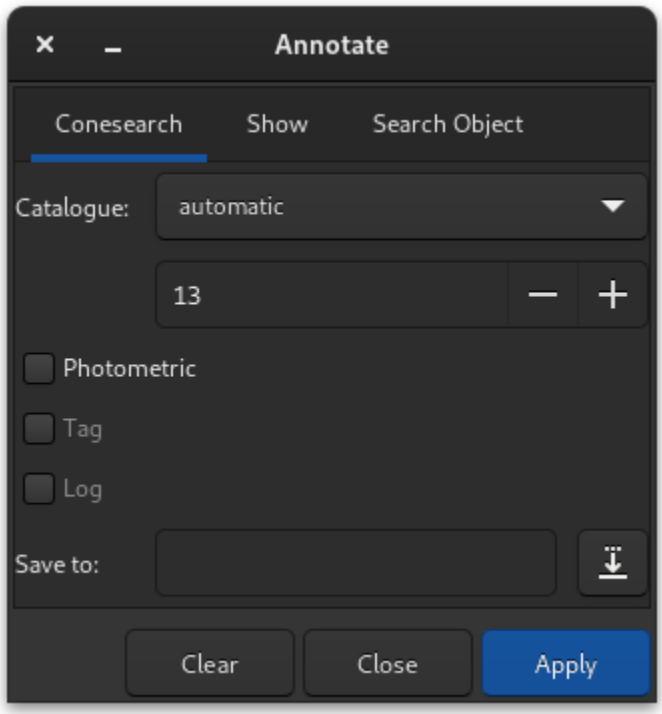



Fig. 15: Graphical version of the *conesearch* command

The table below lists all the catalogues that are available, alongside links to the original data.

Table 1: Table of available catalogues

| Catalogue | *-cat=* | Reference |
|---|---|---|
| Local | *Omitted* | using *local star catalogues* |
| Tycho2 | tycho2 | https://cdsarc.cds.unistra.fr/viz-bin/cat/I/297 |
| Nomad | nomad | https://cdsarc.cds.unistra.fr/viz-bin/cat/I/297 |
| GAIA DR3 | gaia | https://cdsarc.cds.unistra.fr/viz-bin/cat/I/355 |
| PPMXL | ppmxl | https://cdsarc.cds.unistra.fr/viz-bin/cat/I/317 |
| Bright Stars Catalogues | bsc | https://cdsarc.cds.unistra.fr/viz-bin/cat/V/50 |
| APASS DR9 | apass | https://cdsarc.cds.unistra.fr/viz-bin/cat/II/336 |
| GCVS | gcvs | https://cdsarc.cds.unistra.fr/viz-bin/cat/B/gcvs |
| VSX | vsx | https://cdsarc.cds.unistra.fr/viz-bin/cat/B/vsx |
| SIMBAD | simbad | https://simbad.cds.unistra.fr/simbad/sim-tap/ |
| PGC/HYPERLEDA | pgc | https://cdsarc.cds.unistra.fr/viz-bin/cat/VII/237 |
| Exoplanet archive | exo | https://exoplanetarchive.ipac.caltech.edu |
| AAVSO VSP | aavso_chart | https://app.aavso.org/vsp/ |
| IMCCE Solar system skybot | solsys | https://vo.imcce.fr/webservices/skybot/?conesearch |

The following queries are brought to you thanks to:

- Vizier (through CDS Vizier TAP query service)
- Simbad (through CDS Simbad TAP query service)
- IMCCE (through its skybot conesearch service)
- NASA exoplanet archive (though its TAP query service)

For solar system object queries, you may pass an additional parameter **-obscode=**, the 3-symbol code for an IAU observatory close by to your observing location. This will improve annotations accuracy. Please note that results may still slightly differ from those obtained by making a *direct ephemerides query for a specific object*, which uses the exact observing location (if present in the FITS header).

**Tip**

A preferred observatory code may be set in the *Astrometry* tab in the *Preferences* dialog. If set, this will be used for the *Solar System Objects* GUI tool and will also be used in the *conesearch* command unless the **-obscode=** argument is provided.

These additional annotations will be displayed in **RED**, to differentiate them from offline annotations, shown in **GREEN**. These annotations will be erased as soon as the `Show Objects names` button is toggled.

### 11.2.3 Extra catalogues

You may want to display your own user catalogues. This can be done with the command *show*. This command can also be used to display, for instance, csv files created with the feature to find *comparison stars*.

**Siril command line**

```
show [-clear] [{ -list=file.csv | [name] RA Dec }] [-nolog] [-notag]
```

Shows a point on the loaded plate solved image using the temporary user annotations catalogue, based on its

equatorial coordinates. The **-clear** option clears this catalogue first and can be used alone.
Several points can be passed using a CSV file with the option **-list=** containing at least ra and dec columns. If the passed file also contains a column name, names will be used as tags in the image and listed in the Console, unless toggled off with the options **-notag** and **-nolog**.

This is only available from the GUI of Siril

These catalogues may be any *csv* (comma-separated) file, respecting the following rules:

- comments lines if any, should start with a # sign
- a line should be present at the top with the column names, comma separated
- at least *ra* and *dec* columns should be provided, in decimal degrees.
- the columns can be written in no particular order
- other columns can be passed:
  - *name* (str)
  - *diameter* (double), the object diameter in arcmin
  - *mag* (double), the object magnitude
  - *type* (str), which will be appended between *()* after the name in the Console

Other columns than those listed above may be passed but they will not be used.

This command is accessible from the graphical interface via the *Tools* → *Astrometry* → *Annotate...*. You can either load a `*.csv` file containing several lines of comma-separated ra, dec coordinates in *Import CSV*, or a single point in the *Single Coordinates* tab.

In the case of a single point, you can save the result in the *Deep Sky user catalogue* by clicking on the save button as shown in the illustration bellow:

**Tip**

The save button will only work if the field *Display Name* is filled.

**Tip**

You can select a star or object and obtain its coordinates using the *Get coordinates from selection* button.

**List of known user catalogues:**

Sometimes, users create their own catalogues, we can try to link them here to help everybody.

- Variable stars, extracted from GCVS 5.1, discussed here in French, (`file link`).

**Warning**

Contrarily to the instructions discussed in the linked topic, it is not recommended to replace the user-DSO catalogue with such files. The usage is discouraged as some of them could be particularly big and would slow down tremendously every annotation redraw.

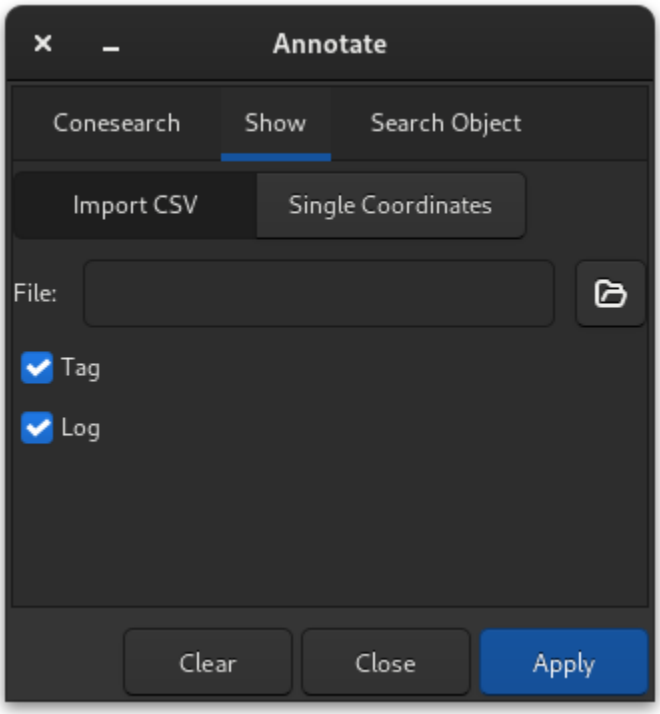



Fig. 16: Graphical version of the *show* command

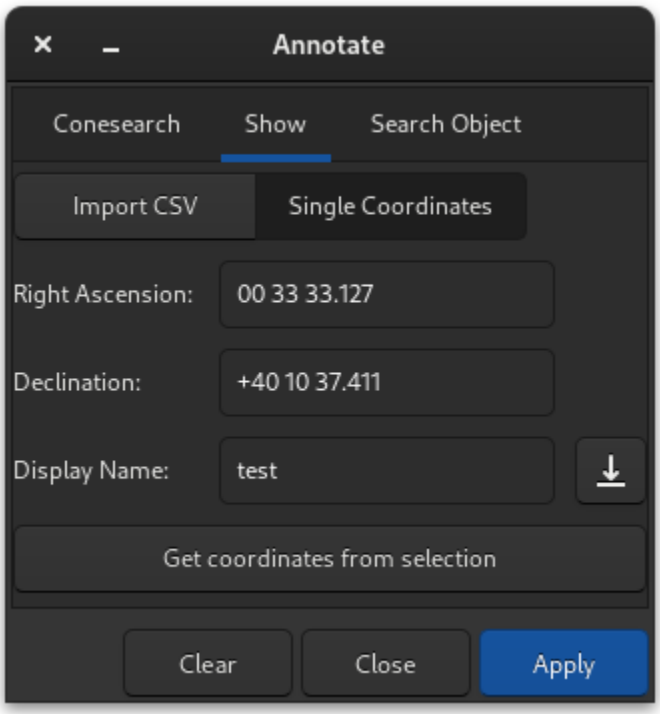



Fig. 17: It is possible to show only one single point and to save it in the *Deep Sky user catalogue*.

### 11.2.4 Search for a known object

If you know a specific object is somewhere in the image (if not, see the *search for an unknown object* section), it is possible to add it to annotations.

#### Deep-sky objects

Load a plate resolved image and type `Ctrl+Shift+/` or *Tools* → *Astrometry* → *Annotate...* then go to the *Search Object* tab of the dialog box.

A search dialog box is available, as illustrated below, in which object names can be entered.

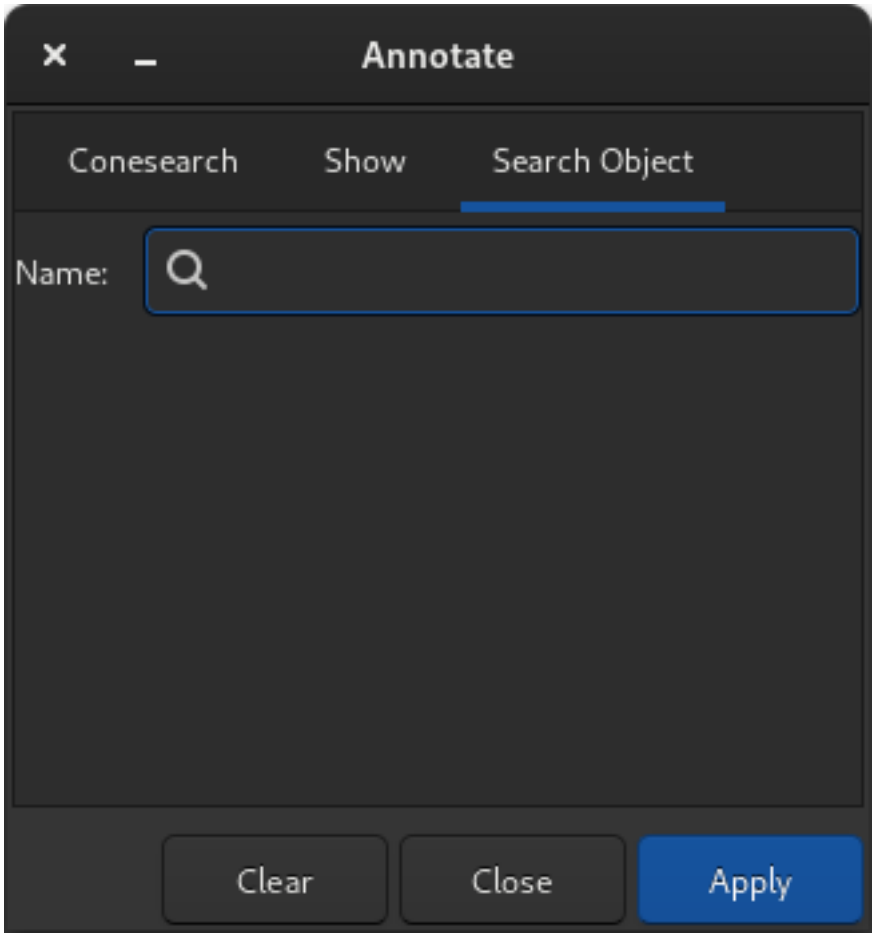



Fig. 18: Search Object dialog

Pressing `Enter` or *Apply* will first search for this name the existing annotation catalogues in case it already exists under another name. If not it will send an online request to SIMBAD to get the coordinates of an object with such a name. If found, and not already in any catalogue, the object will be added to the *Deep Sky user catalogue*.

The items of this catalogue are displayed in **ORANGE** while the objects from the predefined catalogues are displayed in **GREEN**.

Examples of valid input (not case sensitive):

- `HD 86574` or `HD86574` are both valid for this star

Fig. 19: Deep sky objects from user and predefined catalogues

### Solar-system objects

From Siril version 1.2, objects from the solar system can also be searched for, using the Miriade ephemcc service. This is done in the same manner as for Deep Sky Objects, but prefixing the name of the object to be searched by some keyword representing the type of object: `a:` for asteroids, `c:` for comets, `p:` for planets, `dp:` for dwarf planets and `s:` for natural satellites. If you query an image taken from a close enough date and time (same night) than another image already annotated with SSOs, their cached positions will be used and corrected by each object velocity as returned by the ephemerids. The items of this catalogue are displayed in **YELLOW**.

Examples of valid inputs (not case sensitive):

- `c:67p` or `c:C/2017 T2` are valid forms for comets
- `a:1` and `a:ceres` are both valid for *(1) Ceres*
- `a:2000 BY4` is valid for *103516 2000 BY4*
- `p:4` or `p:mars` are both valid for Mars
- `dp:Pluto` is valid for Pluto
- `s:Moon` or `s:Io` is valid for natural satellites.

**Warning**

Images that do not have a `DATE-OBS` header key cannot be annotated for SSOs. Images that do not have observer location information (SITELAT, SITELONG and SITEELEV header keys) will still be annotated, but assuming a geocentric observer position, i.e. as if observing from center of the Earth. Depending on objects distance wrt. Earth, this may result in positions being slightly offset from their real positions.

### Command *catsearch*

The same feature is accessible through the command *catsearch*:

**Siril command line**

```
catsearch name
```

Searches an object by **name** and adds it to the user annotation catalog. The object is first searched in the annotation catalogs, if not found a request is made to SIMBAD.

The object can be a solar system object, in which case a prefix, 'a:' for asteroid, 'p:' for planet, 'c:' for comet, 'dp:' for dwarf planet or 's:' for natural satellite, is required before the object name. The search is done for the date, time and observing location found in the image header, using the IMCCE Miriade service

## 11.2.5 Search for an unknown object

Especially useful for photometry works, it is possible to identify a star or other objects in the image by drawing a selection around them, right clicking to bring up the context menu, and selecting the *PSF* entry.

This will open the PSF window, and if it is a star it will display the Gaussian fit parameters, but it will also display a Web link at the bottom left of the window. Opening it will bring you to the SIMBAD page for the coordinates of the object and in many cases will give you the name of the object.

SIMBAD does not have all known objects, but the coordinates from the page can still be used as a starting point to look for the object in other online catalogues, for example Gaia DR3 (VizieR).

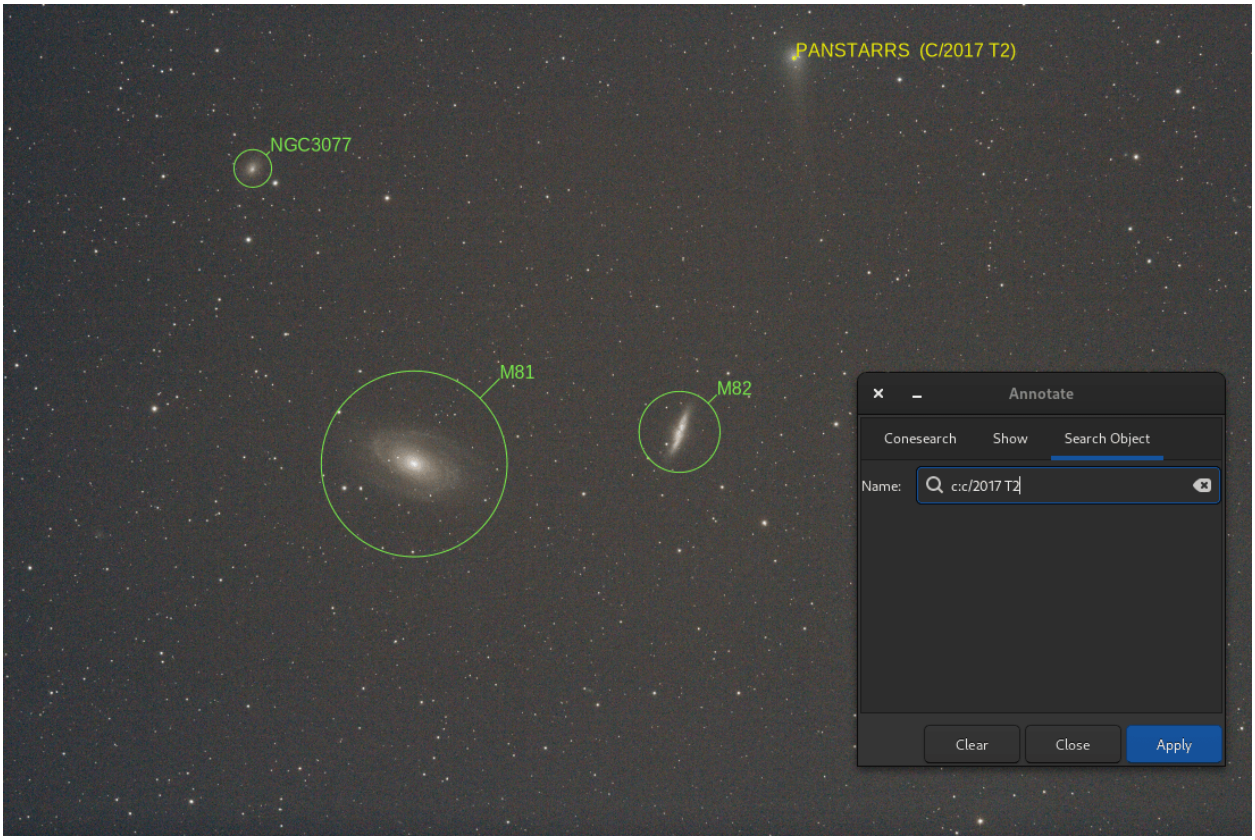



Fig. 20: Result of a Search Solar System process

CHAPTER

TWELVE

# DYNAMIC PSF

This section describes the two essential steps which are undertaken to detect stars in individual frames. The detection on a single image can be run or fine-tuned using *Tools* → *Image Analysis* → *Dynamic PSF* or with shortcut `Ctrl` + `F6`.

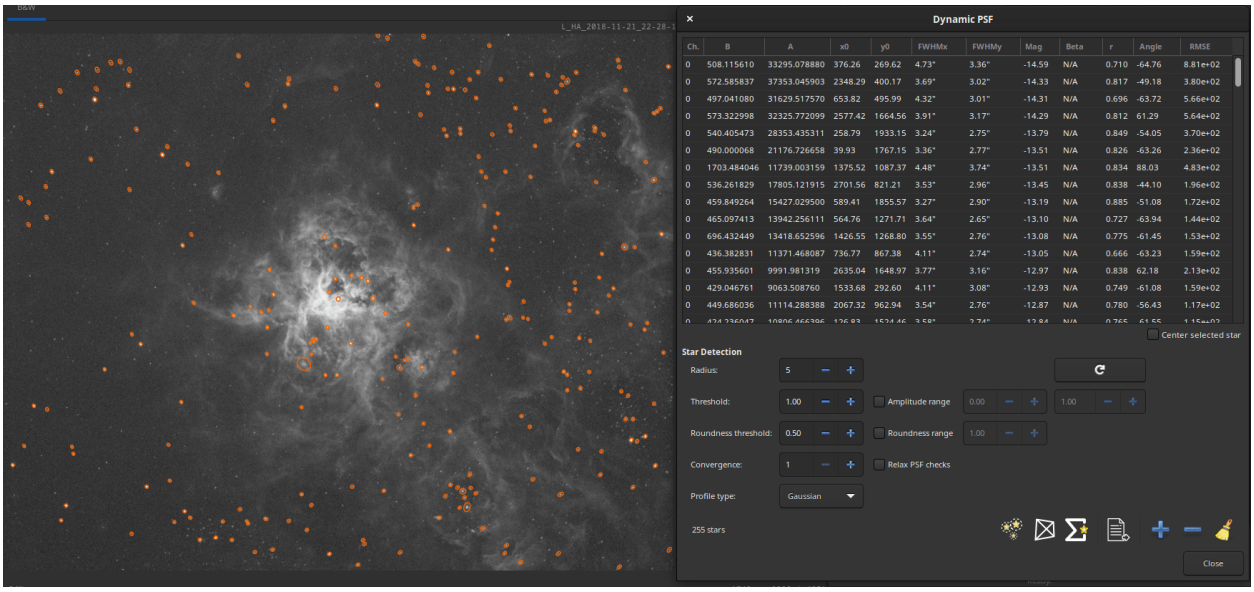



Fig. 1: Dynamic PSF running on a deep sky image.

The process is as follows:

- make an initial detection of potential star candidates
- fit a PSF model on each candidate. Run sanity checks based on model fitting parameters to make sure this is a star and reject non-star candidates.

At the end of this process, the result is a list of stars, with positions in the image wrt. top left corner and measured quantities of all the stars in the list.

## 12.1 Initial star candidates

While it may seem obvious when looking at an image where the stars are, it is a bit more difficult to translate the process into mathemical operations and criteria. This section briefly describes the underlying algorithm. It is inspired from DAOPHOT software manual [Stetson1987], with simplifications brought to boost performance. The original algorithm was aiming at being extensive in detecting all possible stars, serving the purpose of building star catalogues, while Siril needs to detect stars primarily as features for registration . It also needs to be resilient to the wide variety of images

that are submitted - most of us do not have a professional-grade astronomical gear in the backyard - and we have had to make some choices regarding prior knowledge of the imaging condition (sampling, seeing etc).

Over the years, our implementation has evolved to produce what it is today. It aims at not missing very bright stars, which are important for registration, and reject as much as possible outliers, while remaining reasonnably fast at doing so.

It can be decomposed in the following steps:

- compute the statistics of the image to capture both the background level, as the median of the image, and its noise. This assumes that the image is relatively flat. As a consequence, detection will tend to be less efficient in the corners if heavy vignetting is present after Calibration.
- Compute as well a dynamic range, defined as the maximum of the image minus its background. This will later on be useful to detect saturated stars.
- smoothen the image with a gaussian kernel. Ideal smoothing would be to use the kernel that has the same FWHM as the image. Instead, we have chosen an arbitrary size which produced satisfactory results over a very wide range of conditions. This enables to be "blind" to the imaging parameters.
- on the smoothened version of the image, detect local maxima over a level defined as the background level plus X times the noise (X can be varied using the `threshold` value in the GUI). Make sure this is a maximum over a certain box size (defined by the `radius` parameter).
- run sanity check to make sure the maximum and its neighbors are well above the surrounding pixels (to reject patches in the bright parts of nebula for instance).
- run sanity check to guess if the core around the maxima is saturated, i.e. consistently close to the upper bound of the dynamic range. If true, run an edge-walking algorithm to detect the limit of the saturated part.
- Use first and second derivatives along an horizontal and vertical line passing through the center to guess star local background, amplitude and width in all directions (top, down, left and right).
- If the parameters are symmetrical enough in all directions (up to the `roundness` parameter), confirm the star as a potential candidate.

Once the list of potential candidates has been assembled, they are sorted by decreasing amplitudes and fed to the PSF fitting algorithm described in the *minimization* section.

## 12.2 Models

Two models are used in the Dynamic PSF window. In general, Moffat is much better suited to fit objects such as stars.

**Theory**

1. An elliptical Gaussian fitting function defined as

$$G(x,y) = B + Ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)}. \tag{12.1}$$

2. An elliptical Moffat PSF fitting function defined as

$$M(x,y) = B + A\left(1 + \frac{(x-x_0)^2}{\sigma_x^2} + \frac{(y-y_0)^2}{\sigma_y^2}\right)^{-\beta}, \tag{12.2}$$

where:

- $B$ is the average local background.

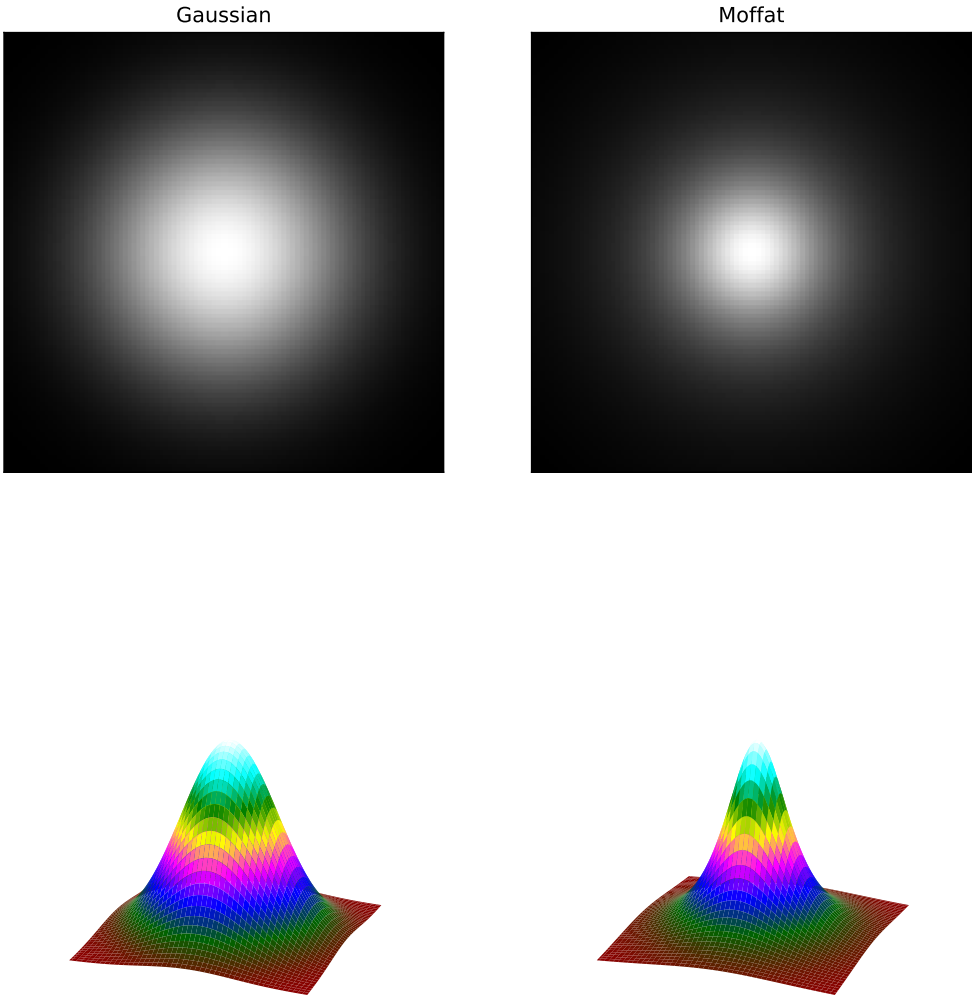



Fig. 2: Displays of two circular PSFs according to a Gaussian profile and a Moffat profile. Both models use the same parameters and the Moffat profile uses uses $\beta = 1.4$.

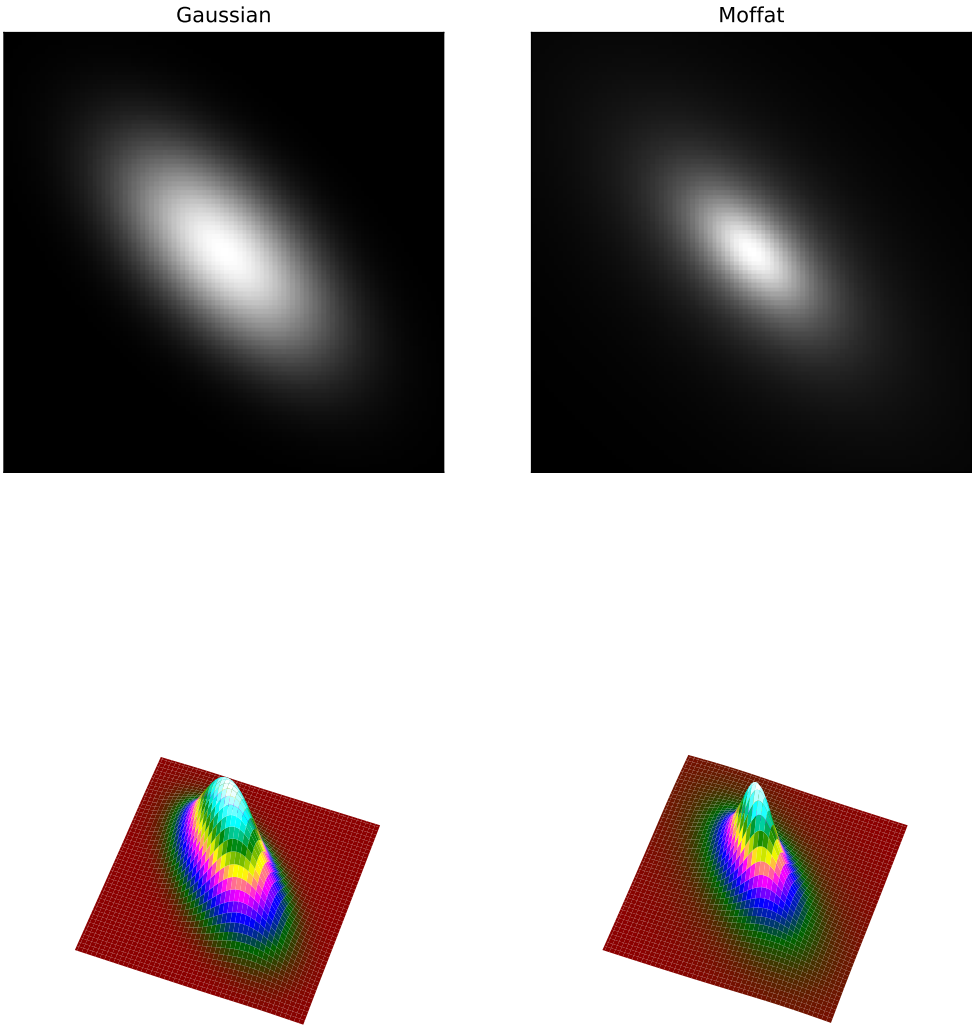



Fig. 3: Rotated Gaussian and Moffat function have $\sigma_x = 2\sigma_y$, $\theta = 45$. For Moffat, $\beta = 1.4$.

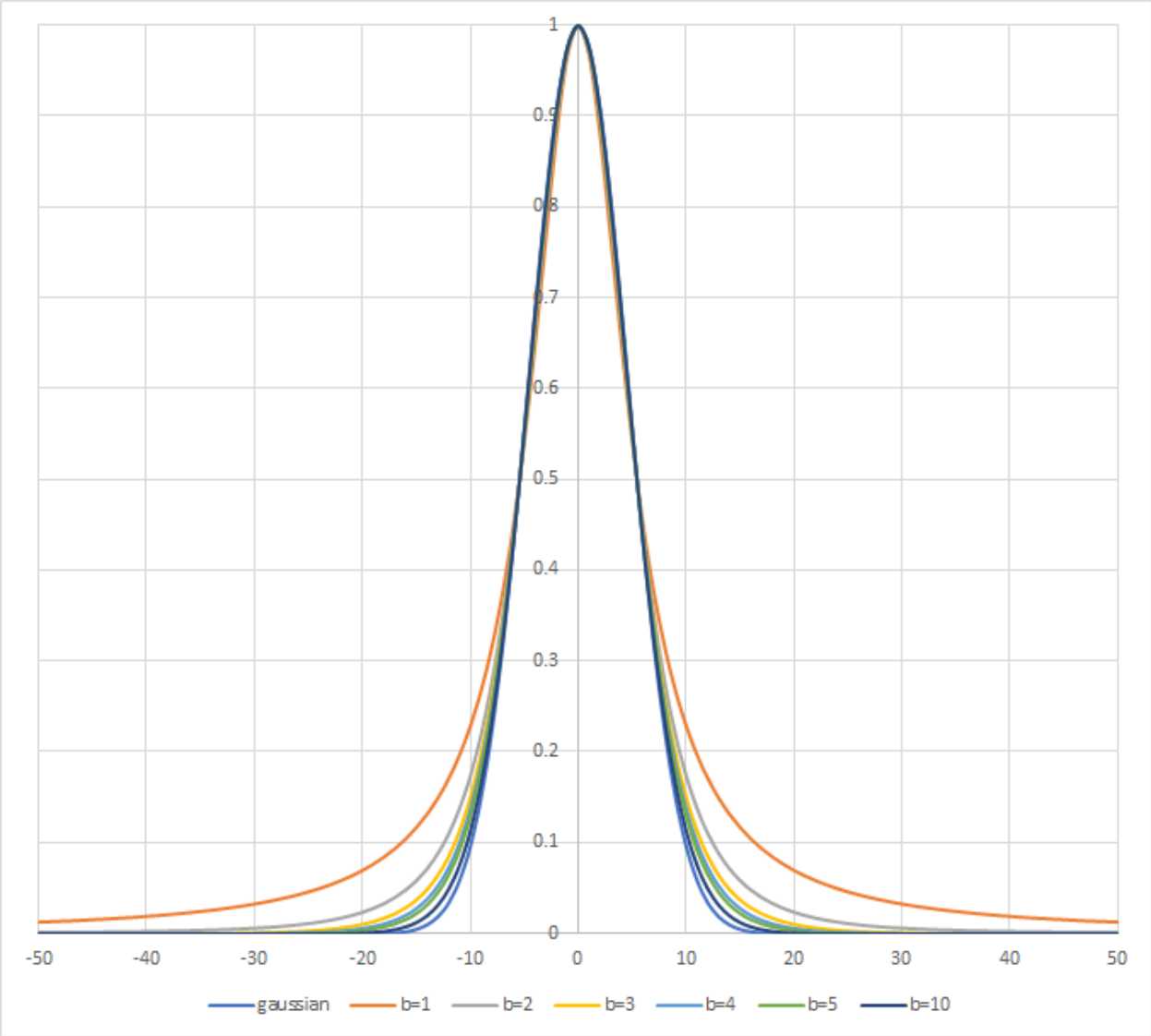



Fig. 4: Star Profile with Gaussian and Moffat models. Several $\beta$ values are tried. $\beta = 10$ gives a profile very close to the Gaussian one.

- $A$ is the amplitude, which is the maximum value of the fitted PSF.
- $x_0$, $y_0$ are the centroid coordinates in pixel units.
- $\sigma_x$, $\sigma_y$ are the standard deviation of the Gaussian distribution on the horizontal and vertical axes, measured in pixels.
- $\beta$ is the exponent from the Moffat formula that controls the overall shape of the fitting function. The upper bound of this parameters was set to 10. A higher value is meaningless and means that the Gaussian is good enough to fit the star.

Other parameters are derived from these fitted variables:

- $\text{FWHM}_x$ and $\text{FWHM}_y$: The Full Width Half Maximum on the X and Y axis in pixel units. These parameters are calculated as follow:
  - $\text{FWHM}_x = 2\sigma_x\sqrt{2\log 2}$.
  - $\text{FWHM}_y = 2\sigma_y\sqrt{2\log 2}$.
  - It is possible to obtain the FWHM parameters in arcseconds units. This requires you fill all fields corresponding to your camera and lens/telescope focal in the setting parameter window in the burger menu, then *Image Information* and *Information*. If standard FITS keywords `FOCALLEN`, `XPIXSZ`, `YPIXSZ`, `XBINNING` and `YBINNING` are read in the FITS HDU, the PSF will also compute the image scale in arcseconds per pixel.
- $r$: The roundness parameter. It is expressed as $\text{FWHM}_x/\text{FWHM}_y$, with $\text{FWHM}_x > \text{FWHM}_y$ the symmetry condition.
- Another parameters is also fitted in both Gaussian and Moffat models. This is the rotation angle $\theta$, defined in the range $[-90, +90]$. The addition of this parameter implies a coordinate change where the $x$ and $y$ variables expressed in (12.1) and (12.2) are replaced by $x'$ and $y'$:

$$\begin{aligned} x' &= +x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta. \end{aligned} \tag{12.3}$$

## 12.3 Minimization

Minimization is performed with a non-linear Levenberg-Marquardt algorithm thanks to the very robust GNU Scientific Library. This algorithm is used to find the minimum of a function that maps a set of parameters to a set of observed values. It is a combination of two optimization techniques: the gradient descent method and the inverse-Hessian method.

The Levenberg-Marquardt algorithm adjusts the trade-off between these two methods based on the curvature of the function being minimized. When the curvature is small, the algorithm uses the gradient descent method, and when the curvature is large, the algorithm uses the inverse-Hessian method.

Since version 1.2.0, the saturated part of the star is removed from the fitting process, enabling to capture much more accurately the non-saturated part. This is what enables to "reconstruct" the star profile when using the *Desaturate* menu option or *unclipstars* command.

## 12.4 Use

Dynamic PSF can be called from two different ways depending on what you want:

You may want to fit just one or a few stars. In this case, after drawing a selection around an unsaturated star (this is important for the accuracy of the result) you can either right click and choose the *Pick A Star* item, click the + button

in the Dynamic PSF dialog, or type `Ctrl` + `Space`. An ellipse is then drawn around the star. To open the dialog it is also possible to use the shortcut `Ctrl` + `F6`.

You may want to analyze as many stars as possible by clicking on the icon , or using the command line *findstar*. All detected stars are surrounded by an ellipse: orange if the star is ok, magenta if the star is saturated. It is also possible to show the average of the computed parameters as illustrated below, by clicking on the button.

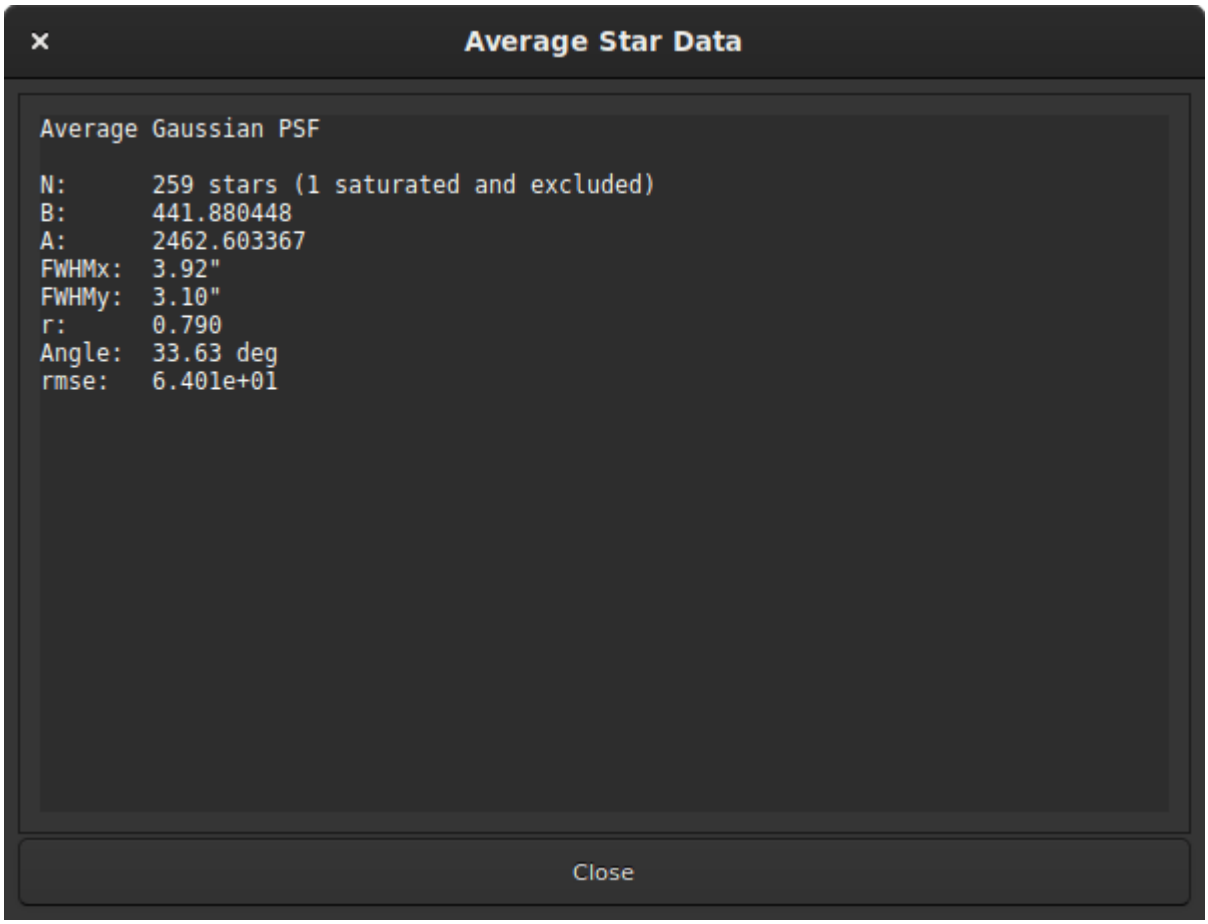



Fig. 5: Average of the fitted stars in the Gaussian model.

Star detection has a number of uses:

- Siril uses it internally for astrometric purposes when registering sequences of images. This is automatic and requires no user intervention.
- Because stars are so bright compared with dim features of interest such as nebulae or galaxies, it is very common that some stars in an image will be saturated, meaning their brightness profile is clipped. This can cause problems for some image processing functions, particularly deconvolution, and results in loss of colour information and slightly greater star bloat when applying stretches. Analysis of all the stars will show you which ones are saturated, and you can then use the *Desaturate* menu option or *unclipstars* command to fix the problem by synthesis of the clipped part of the profile.

**Siril command line**

```
unclipstars
```

Re-profiles clipped stars of the loaded image to desaturate them, scaling the output so that all pixel values are <= 1.0

- Ideally all stars in an image should be perfectly round, however problems such as coma, astigmatism and bad tracking, as well as issues such as incorrect back focus to field flatteners, can give rise to patterns of elliptical stars in an image. The ellipses produced by the Dynamic PSF tool can give a good visual illustration of such problems.
- Examination of the average star parameters, especially FWHM and the Moffat fitting function beta parameter, can provide information about the quality of seeing in an image.
- Detection of all stars is the first step in using the *Star Tools → Full Resynthesis tool*. This synthesizes corrected luminosity profiles for all detected stars, and can be used to create a synthetic star mask which can then be mixed with a starless image generated by Starnet++ to fix otherwise irredeemable stars in an image. In this case detection of stars using the Moffat profile may give a more realistic result and can also make it easier to filter out galaxies incorrectly detected as stars by using the Minimum beta setting.
- The **Center Selected Star** toggle button can be used to find a particular star in the list quickly and easily in the image, by centring it in the viewport. This is useful if you have detected all stars and wish to check specific solutions to ensure they really are a star and not a galaxy or a cosmic ray.
- Similarly to this, clicking on an orange or magenta star ellipse in the main viewport will highlight the selected star solution in the Dynamic PSF dialog. This could be useful if you wish to see the parameters of an individual star.
- Siril's deconvolution functions support using Dynamic PSF measurements to synthesize a deconvolution function that matches star parameters measured directly from the image.

## 12.5 Configuration

Dynamic PSF can be configured using the settings in the Dynamic PSF dialog:

- **Radius** sets the half-size of the search box. If you have trouble detecting particular stars you can try changing this but usually the default is fine.
- **Threshold** changes the threshold above the noise for star detection. If you increase this value, fewer faint stars will be detected. You may still wish to do this for very noisy images though. Decreasing this value may detect more faint stars, but will also make the algorithm more likely to misidentify random noise spikes as stars.
- **Roundness** threshold sets the allowable ellipticity for detections to be accepted as stars. Highly elliptical stars may occur due to imperfect tracking or aberrations, but sometimes stars that are too close from each others are also detected as a single very elongated star. To highlight all these problems, it is possible to enable a higher bound for the roundness. A maximum value of 1 is equivalent to disabling the range, leaving only the minimum value. This roundness range should be disabled for registration or astrometry.
- **Convergence** sets a criterion used by the solver. Increasing it will allow the solver more interations to converge and can potentially detect additional stars, but may increase the solver running time.
- Profile type chooses between solving **Gaussian** or **Moffat** profiles for the stars.
- **Minimum beta** sets a minimum permissible value of beta for a detection to be accepted as a star. Galaxies may sometimes be detected as Moffat profile stars but they have diffuse profiles and the value of beta is usually very low, less than around 1.5.

- **Relax PSF checks** allows for relaxation of several of the star candidate quality checks. This is likely to result in a significant increase in false positive star detections, often with wild parameters.
- A range of **minimum** and **maximum amplitude** can be set to limit the amplitude (parameter named `A` in reports) of detected stars. This is useful if only non-saturated stars need to be selected, for PSF fitting in deconvolution for example. Note that removing the saturated stars from detection can break registration and astrometry.

**Tip**

The settings defined in this window can be tested on the currently loaded image. However, you have to keep in mind that they will also be used for all images of the sequence, especially for the global alignment registration.

The *findstar* command will obey the same settings entered in the Dynamic PSF dialog, but it can also be configured using the *setfindstar* command.

**Siril command line**

```
findstar [-out=] [-layer=] [-maxstars=]
```

Detects stars in the currently loaded image, having a level greater than a threshold computed by Siril.
After that, a PSF is applied and Siril rejects all detected structures that don't fulfill a set of prescribed detection criteria, that can be tuned with command SETFINDSTAR.
Finally, an ellipse is drawn around detected stars.

Optional parameter **-out=** allows the results to be saved to the given path.
Option **-layer=** specifies the layer onto which the detection is performed (for color images only).
You can also limit the maximum number of stars detected by passing a value to option **-maxstars=**.

See also CLEARSTAR

Links: *psf*, *setfindstar*, *clearstar*

**Siril command line**

```
setfindstar [reset] [-radius=] [-sigma=] [-roundness=] [-focal=] [-pixelsize=] [-
↪convergence=] [ [-gaussian] | [-moffat] ] [-minbeta=] [-relax=on|off] [-minA=] [-
↪maxA=] [-maxR=]
```

Defines stars detection parameters for FINDSTAR and REGISTER commands.

Passing no parameter lists the current values.
Passing **reset** resets all values to defaults. You can then still pass values after this keyword.

Configurable values:

**-radius=** defines the radius of the initial search box and must be between 3 and 50.

**-sigma=** defines the threshold above noise and must be greater than or equal to 0.05.

**-roundness=** defines minimum star roundness and must between 0 and 0.95. **-maxR** allows an upper bound to roundness to be set, to visualize only the areas where stars are significantly elongated, do not change for registration.

**-minA** and **-maxA** define limits for the minimum and maximum amplitude of stars to keep, normalized between 0 and 1.

**-focal=** defines the focal length of the telescope.

**-pixelsize=** defines the pixel size of the sensor.

**-gaussian** and **-moffat** configure the solver model to be used (Gaussian is the default).

If Moffat is selected, **-minbeta=** defines the minimum value of beta for which candidate stars will be accepted and must be greater than or equal to 0.0 and less than 10.0.

**-convergence=** defines the number of iterations performed to fit PSF and should be set between 1 and 3 (more tolerant).

**-relax=** relaxes the checks that are done on star candidates to assess if they are stars or not, to allow objects not shaped like stars to still be accepted (off by default)

Links: *findstar*, *register*, *psf*

## 12.6 References

CHAPTER

THIRTEEN

# FITS HEADER

Keywords contained in the header of a FITS file can be displayed in Siril. To do this, simply click on *Tool* → *FITS Header*.

Since version 1.3.0, it is possible to modify the value of keywords supported by Siril, either with the *update_key* command, or via the GUI, in the appropriate window. This window is divided into two tabs. The first, the *Header Editor*, lets you edit the keywords of the loaded image. When a sequence is loaded, the changes are applied to every image of the sequence (see the note below). It may vary from what the file header actually looks like, and represents more the state it will be in after saving. Keyword values are updated in real time. The second tab displays the header in text format as written in the file and as it was represented for Siril versions below 1.3.0.

**Warning**

HISTORY keywords are not displayed in the editor. They are visible in the *Saved Header* tab.

Another way of displaying the header is to use the command line *dumpheader*. It shows the same header as displayed in the *Saved Header* tab.

**Warning**

As SER files contain very few keywords and are different from FITS files, this command is not applicable to this type of sequence.

Key name, key values and key comments can be modified, however, the key must be unprotected. For easy recognition, protected keys are displayed in salmon color.

Editing is very simple, just `double-click` on the cell to be modified. The first time selects the field, the second opens the edit mode. Pressing the `Enter` key validates the entry. To make the changes effective, remember to save the file.

**Note**

When a sequence is loaded, editing a key name, value or comment applies the change to the FITS header of **every image** of the sequence. A confirmation is asked the first time; editing then stays enabled until the window is closed. This is the graphical equivalent of the *sequpdate_key* command and is not available for SER sequences.

**Warning**

Please note that Siril does not check the validity of the value entered. It is up to the user to enter a valid value. An incorrect value may lead to undesirable behavior in Siril's keyword processing.

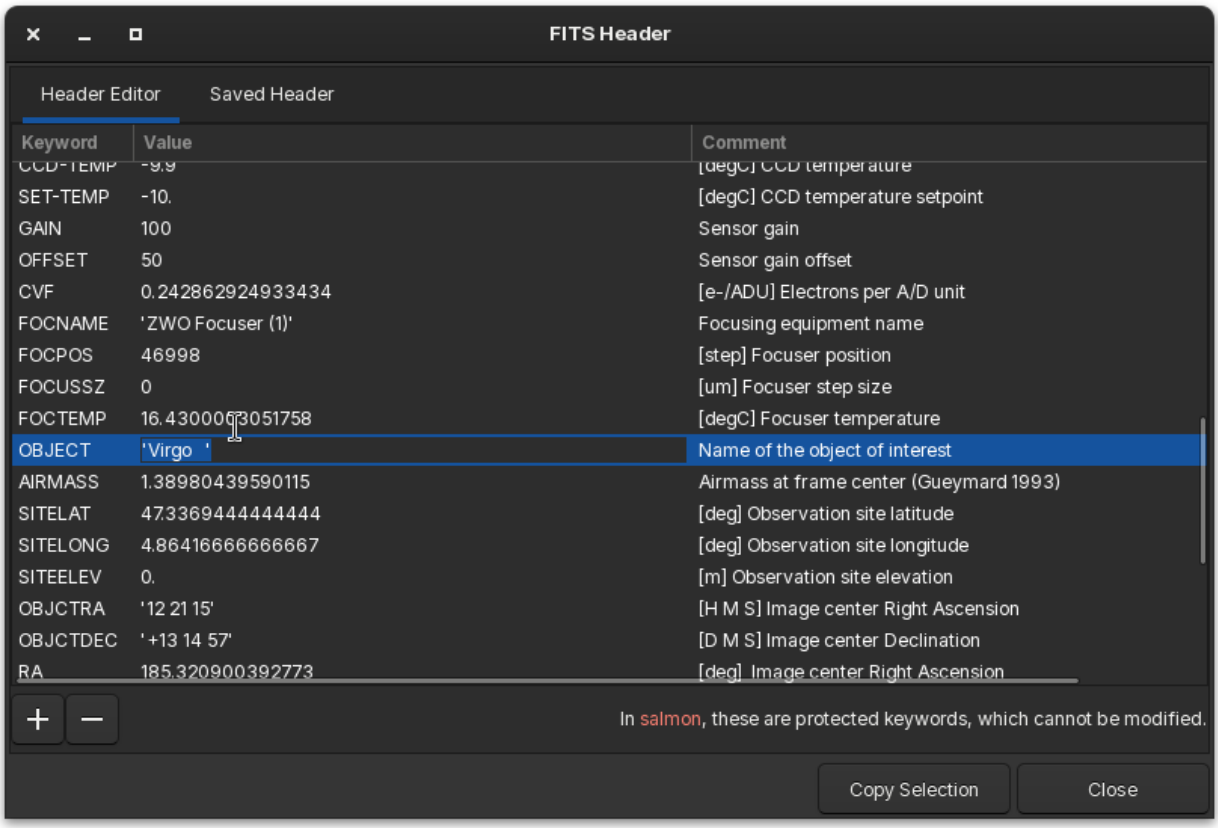



Fig. 1: FITS Header dialog when editing the value of a keyword.

The window contains an option in the form of a *Copy selection* button. It copies the selected lines to the clipboard in the original format of the FITS header, and works for both tabs.

Finally, you can add a new keyword using the + button at bottom left, and delete one using the - button:

1. To add a keyword, click on the + button. A new dialog box opens, as shown below. This window lets you add a new keyword, whose name will be limited to 8 characters (the `HIERARCH` convention is not used). You can leave the *Comment* field empty, if the other two are filled in. However, the user can only fill in the *Comment* field to add a single comment to the FITS header. Finally, if the keyword exists, then its value will be updated with the information provided.
2. To delete a keyword, select one or more lines and click on the - button (or with the keys `Del` or `Backspace`). The selected keywords will be deleted, unless they are protected.

> **Tip**
>
> When a sequence is loaded, it is only possible to select one line at a time. This means you can only delete one keyword at a time from a sequence. But for a single image, you can select several.

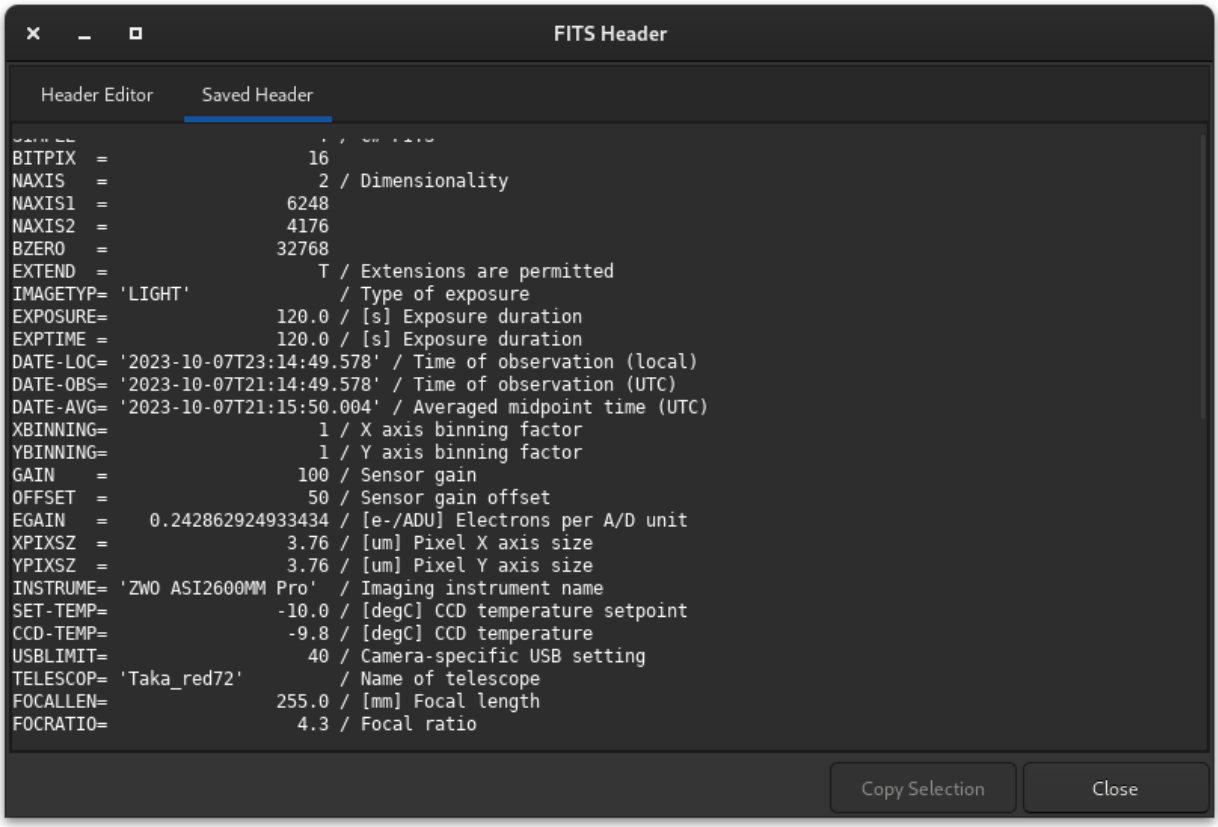



Fig. 2: Second tab of the FITS Header dialog.

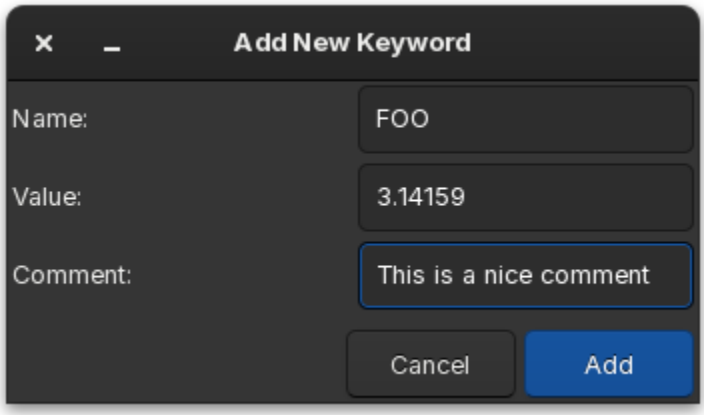



Fig. 3: The window for adding new keywords.

**Siril command line**

```
dumpheader
```

Dumps the FITS header of the loaded image in the console

**Siril command line**

```
update_key key value [keycomment]
update_key -delete key
update_key -modify key newkey
update_key -comment comment
```

Updates FITS keyword. Please note that the validity of **value** is not checked. This verification is the responsibility of the user. It is also possible to delete a key with the **-delete** option in front of the name of the key to be deleted, or to modify the key with the **-modify** option. The latter must be followed by the key to be modified and the new key name. Finally, the **-comment** option, followed by text, adds a comment to the FITS header. Please note that any text containing spaces must be enclosed in double quotation marks

**Siril command line**

```
sequpdate_key sequencename key value [keycomment]
sequpdate_key sequencename -delete key
sequpdate_key sequencename -modify key newkey
sequpdate_key sequencename -comment comment
```

Same command as UPDATE_KEY but for the sequence **sequencename**. However, this command won't work on SER sequence

Links: *update_key*

CHAPTER

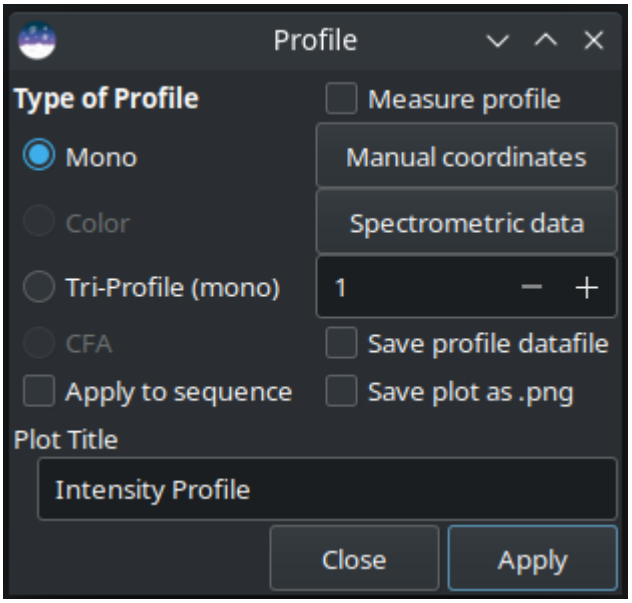
FOURTEEN

# INTENSITY PROFILING

Siril has an intensity profiling mode. The user selects a line between two points and Siril will generate a graph of the pixel values between them. This has several uses. It can be used to inspect the intensity profile of an individual star or a whole galaxy.

## 14.1 Basic Intensity Profile

To make a basic intensity profile of a star or other object, select the Profile button in the bottom toolbar. This puts Siril into profiling mode and opens a small dialog.




You can now click and drag on the main image display to set the start and finish points of the line you wish to profile. If you hold down the `Shift` key while dragging the line, it will snap to be either horizontal or vertical.

**Tip**

When the profile line is exactly horizontal or exactly vertical, exact pixel values can be used directly from the image. When the profile line is neither horizontal nor vertical, the points to be plotted do not fall exactly on a pixel and bilinearly interpolated pixel values are therefore used.

A custom title for your plot can be entered in the control at the bottom of the dialog.

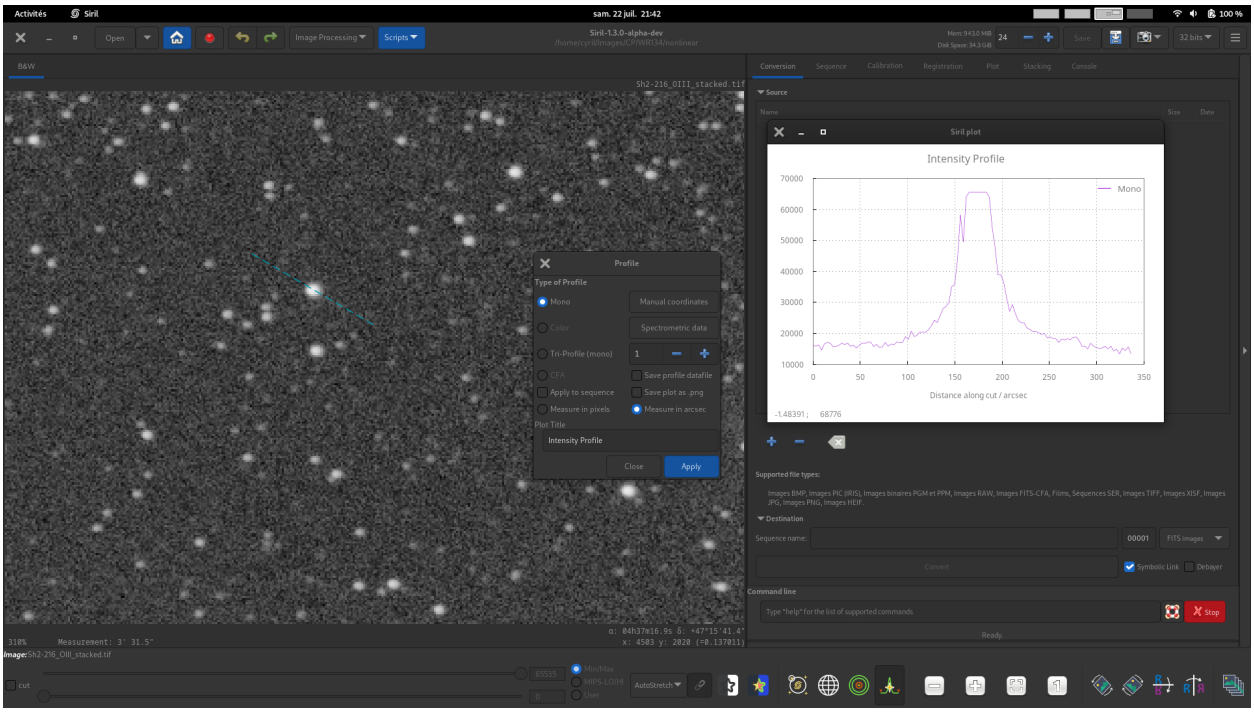
**Tip**

When processing a sequence, it is possible to have the custom title display the image number and total by adding () to the end of the title. For example entering **Solar Spectra** () as the title for a 5 image sequence will generate titles **Solar Spectra (1 / 5)**, **Solar Spectra (2 / 5)** etc. The brackets are ignored and removed if processing a single image.

## 14.2 Types of Profile

Use the radio buttons to select the type of profile you want. (Click on the example images below to see them full size.)

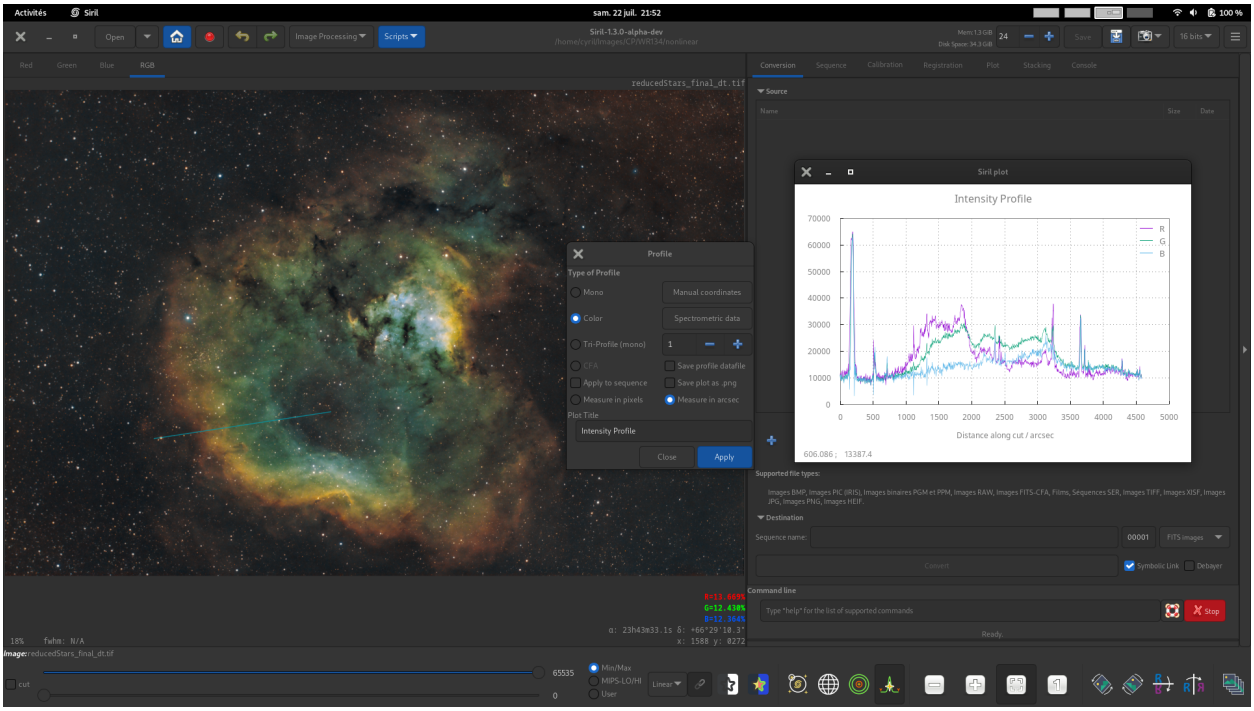
- **Mono profile**. For mono or color images, generate a luminance profile between two points. This mode can be used with spectrometric data.




**Tip**

If a color image is loaded but the mono profiling mode is selected, the profile will be made according to the viewport. The R, G and B viewports provide mono profiles of their respective channel and the RGB viewport provides a luminance profile weighting all 3 channels equally.

- **Color profile**. For color images, generate three profiles for the R, G and B pixel values between two points. This mode can be used with spectrometric data.




- **Tri-profile (mono)**. For mono or color images, generate three parallel equispaced luminance profiles between two points. The spacing between the 3 profiles can be set using the spin button.
- **CFA**. For images with a Bayer pattern only, generate four profiles for the four CFA subchannels between two points. This can be particuarly useful for inspecting the profile of Bayer patterned flats or other Bayer pattern images before they are debayered.

Click *Apply* to generate your profile.

## 14.3 Precise Coordinate Entry

In order to make it easy to input coordinates precisely and repeatably, a manual entry method is provided. Click the *Manual Coordinates* button and you can enter the X and Y coordinates of the start and end points of the profile line. If a profile line is already drawn but one point is not quite in the place you want it, you can use this popup dialog to fine tune the placement of the endpoints.

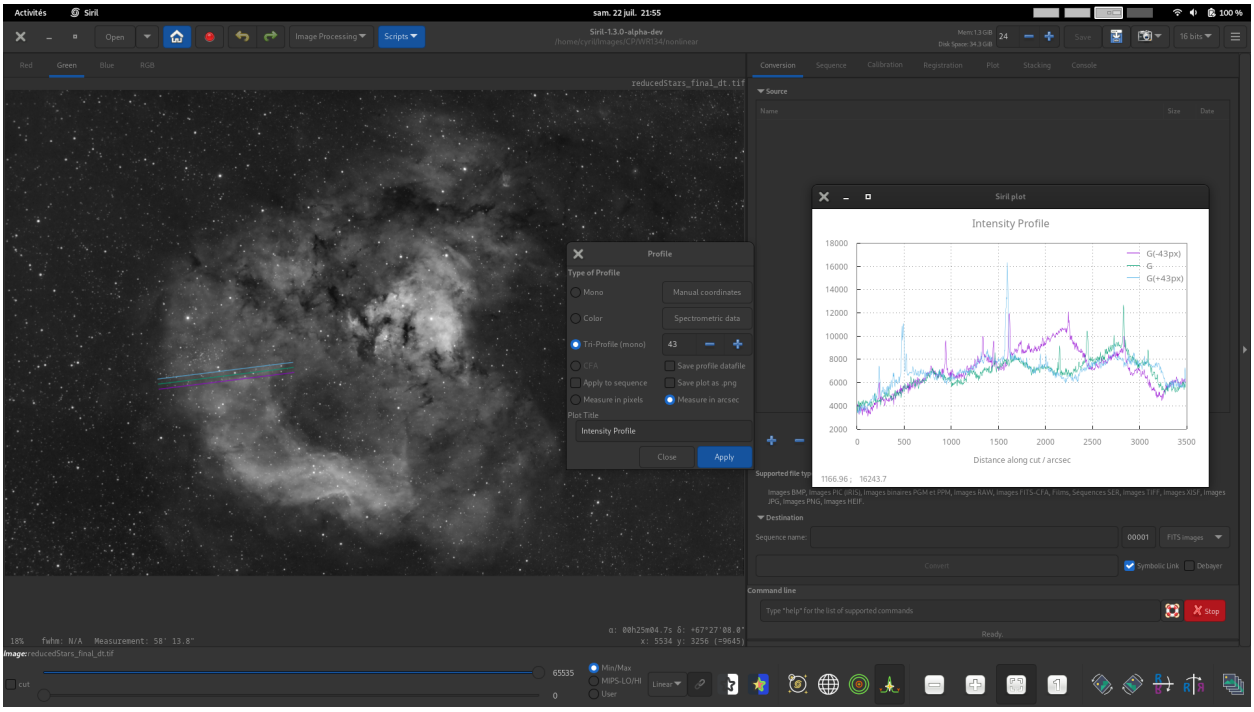
If you wish to set an endpoint exactly to the position of a star, make a rectangular selection around the star and click the relevant star button to the right of the dialog.

> **Note**
>
> When using the CFA mode, coordinates are given in the *input* image. However each CFA channel is half the width and half the height. The x axis in the CFA mode plot is measured in pixels in the CFA subchannel, i.e. it will span half the number of pixels that it does in the input image.

## 14.4 Measurement

The intensity profile line can be used as a measuring tool in two ways:

- Checking the *Measure profile* checkbox will measure all profile lines dragged with the mouse, similarly to the `Ctrl` + `Shift` + `Drag` quick measurement function.



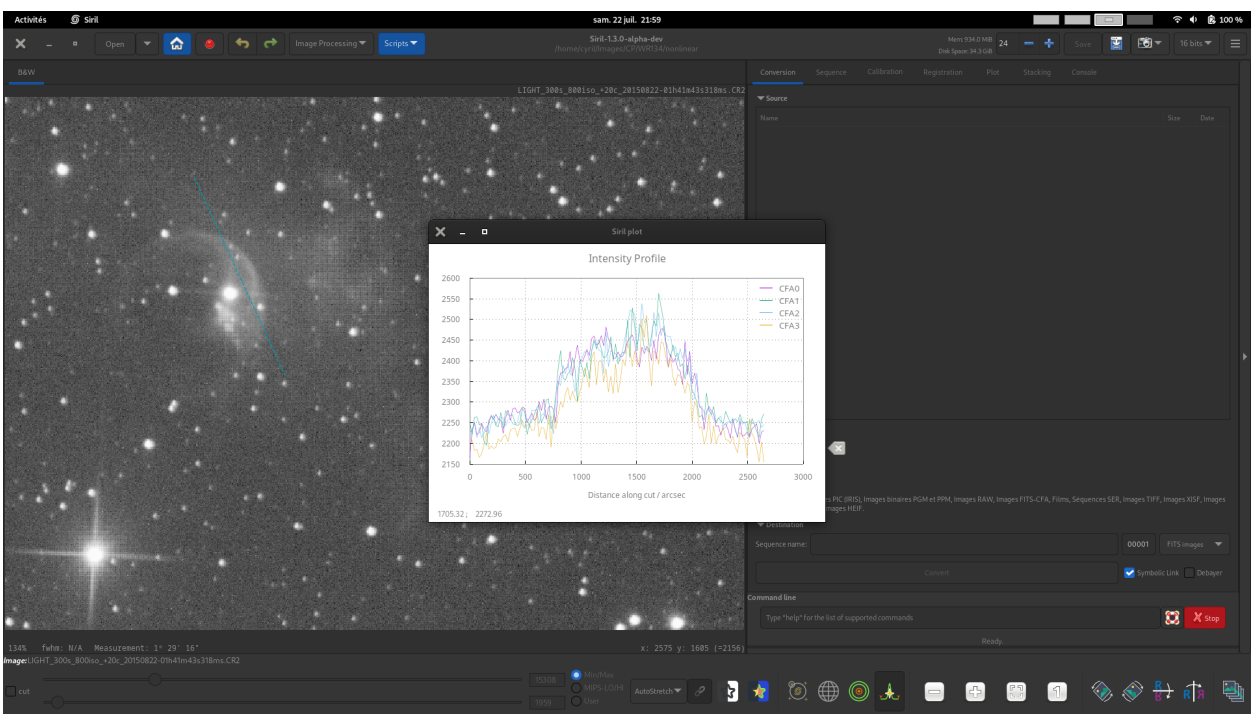




Fig. 1: This image demonstrates use of the Custom Title control to set a custom title for the plot.

- In the Coordinates dialog there is a *Measure* button. This provides the same measurement function but allows you to set the endpoints exactly, and then measure the profile line on demand. By selecting stars, minor planets or comet nuclei as end points as described above, measurements between two celestial bodies can be made very precisely (with sub-pixel precision).

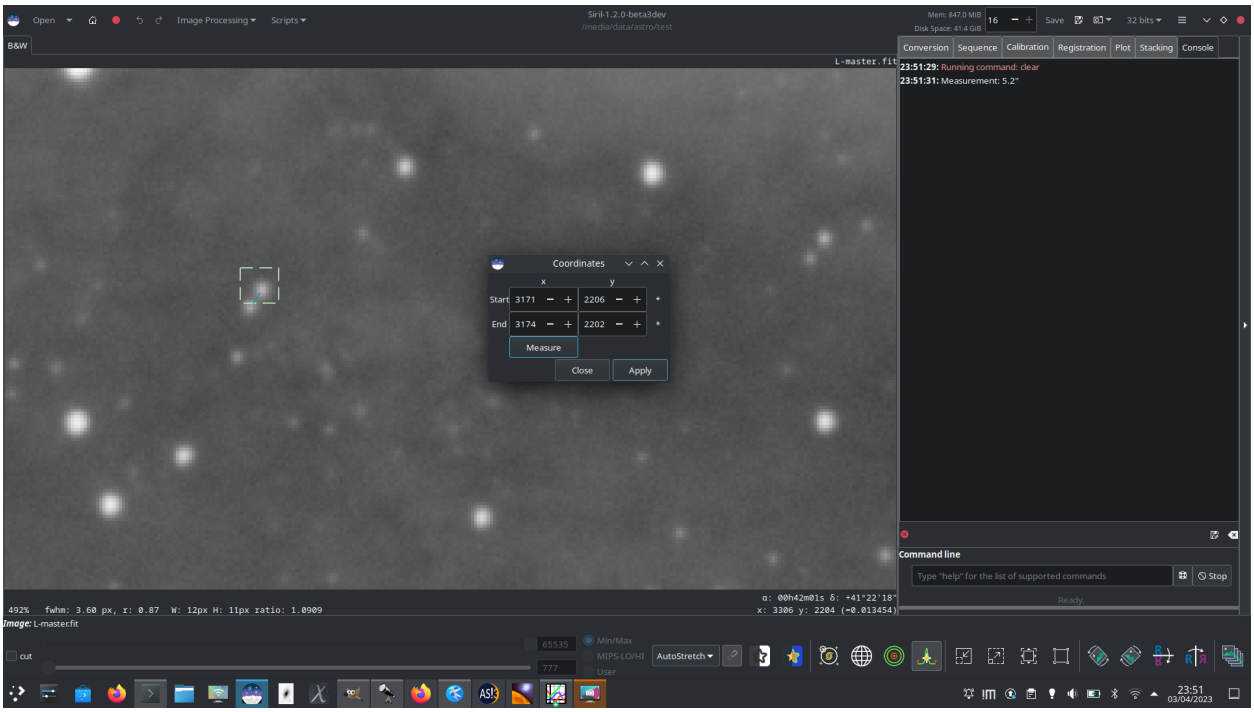



Fig. 2: Here, two close stars have been selected and set as the endpoints and the separation between them measured as 5.2 arcsec. This could be used to study close binaries or to triangulate the position of a minor planet.

> **Note**
>
> Siril's measurement function makes the small angle approximation for the angular separation $\theta$. The most significant error term is proportional to $\theta^3$ and is less than 1% for measurements up to 10°: it is therefore valid for most astrometric uses, but will become inaccurate for large measurements across ultra-wide field images. A warning will be written to the log for measurements over 10°.

## 14.5 Siril Plot Tool

The profiling feature uses Siril internal plotting tool to display the different profiles. With the `*.dat` files produced, you can still use any plotting tool of your liking to explore the underlying data.

A `right-click` anywhere in the plotting surface will pop-up a contextual menu to:

- Show/hide grids and legend
- Export current view to clipboard, `*.png` or `*.svg`
- Save underlying data to a `*.dat` file

Note that all exports account for the current zoom/pan while saving to dat will export unfiltered data.

The following GUI interactions are avalaible:

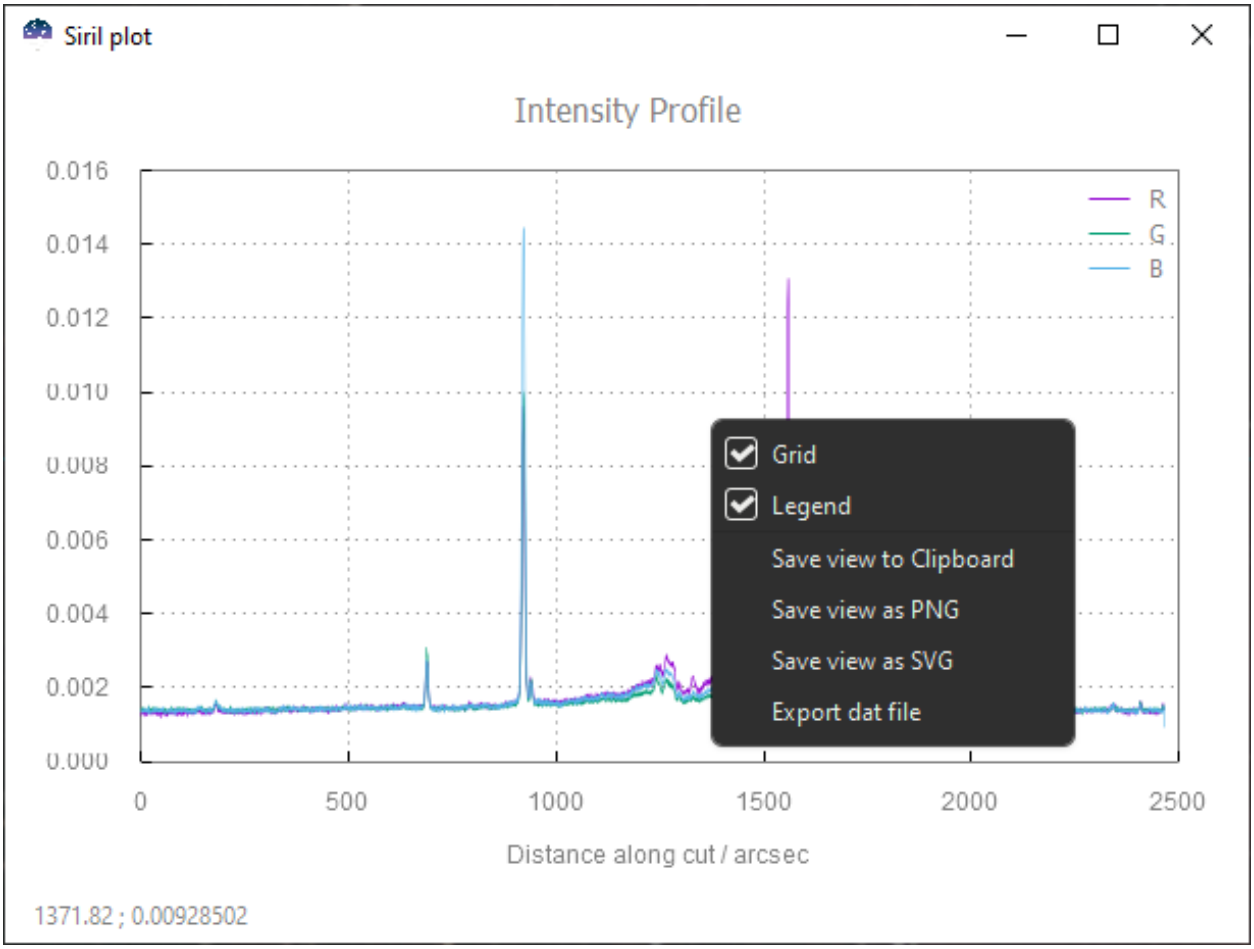



Fig. 3: Siril plot contextual menu

- `Click` + `Drag` to draw a selection. The zoom is set to the selected zone when the mouse is released.
- `Ctrl` + `Drag` to pan the current view.
- `Ctrl` + `Scroll` to zoom in/out.
- `Double-click` to reset to the default position/zoom.

## 14.6 Commands

**Siril command line**

```
profile -from=x,y -to=x,y [-tri] [-cfa] [-arcsec] { [-savedat] | [-filename=] } [-
↪layer=] [-width=] [-spacing=] ["-title=My Plot"]
```

Generates an intensity profile plot between 2 points in the image, also known as a *cut*. The arguments may be provided in any order. The arguments **-to=x,y** and **-from=x,y** are mandatory.

The argument **-layer={red | green | blue | lum | col}** specifies which channel (or luminance or colour) to plot if the image is color. It may also be used with the **-tri** option, which generates 3 parallel equispaced profiles each separated by **-spacing=** pixels, but note that for tri profiles the **col** option will be treated the same as **lum**.

The option **-cfa** selects CFA mode, which generates 4 profiles: 1 for each CFA channel in a Bayer patterned image. This option cannot be used with color images or mono images with no Bayer pattern, and cannot be used at the same time as the **-tri** option.

The option **-arcsec** causes the x axis to display distance in arcsec, if the necessary metadata is available. If not provided or if metadata is not available, distance will be shown in pixel units.

The argument **-savedat** will cause the data files to be saved: the filename will be written to the log. Alternatively the argument **-filename=** can be used to specify a filename to write the data file to. (The **-filename=** option implies **-savedat**.)

The argument **"-title=My Title"** sets a custom title "My Title"

**Siril command line**

```
seqprofile sequence -from=x,y -to=x,y [-tri] [-cfa] [-arcsec] [-savedat] [-layer=] [-
↪width=] [-spacing=] [ {-xaxis=wavelength | -xaxis=wavenumber } ] [{-wavenumber1= | -
↪wavelength1=} -wn1at=x,y {-wavenumber2= | -wavelength2=} -wn2at=x,y] ["-title=My␣
↪Plot"]
```

Generates an intensity profile plot between 2 points in each image in the sequence. After the mandatory first argument stating the sequence to process, the other arguments are the same as for the **profile** command. If processing a sequence and it is desired to have the current image number and total number of images displayed in the format "My Sequence (1 / 5)", the given title should end with () (e.g. "My Sequence ()" and the numbers will be populated automatically)

CHAPTER

FIFTEEN

# IMAGE INSPECTORS

Siril has several tools that can help you to analyze your image and tell you about the quality of the shot. In particular if your setup has or not optical defects.

## 15.1 Tilt

The first tool proposed by Siril is the tilt calculation. Sensor tilt occurs when the sensor is not orthogonal to the imaging plane: this requires an intervention on the optical system. You can execute this functionality in two different ways. Either via the GUI (in the menu *Tools* → *Image Analysis* → *Show Tilt*) or via the command line. The latter even offers an alternative that allows you to calculate the tilt over a whole sequence of images for greater accuracy. The following command:

**Siril command line**

```
tilt [clear]
```

Computes the sensor tilt as the FWHM difference between the best and worst corner truncated mean values. The **clear** option allows to clear the drawing

will output:

```
22:28:13: Running command: tilt
22:28:13: Findstar: processing for channel 0...
22:28:15: Stars: 7598, Truncated mean[FWHM]: 3.40, Sensor tilt[FWHM]: 0.31 (9%), Off-
↪axis aberration[FWHM]: 0.39
```

In the console are indicated:

- the number of stars used for the measurement
- the average FWHM on the sensor, free of outliers
- the tilt, expressed as the difference between the best and the worst FWHM on the 4 corners of the image with in parenthesis the percentage of tilt deviation (value greater than 10% indicates a tilt problem)
- the aberration, expressed by the difference in FWHM between the stars in the center and the stars on the edges of the sensor

If the number of stars detected is low (<200), the dynamic PSF detection parameters allow improvement by adjusting the threshold / radius. In fact, the greater the number of stars used in the calculation, the more reliable the result of the analysis.

**Warning**

For the result to make sense, it is preferable to run this command on a sub and not a stacking result. A pre-processed image (just demosaiced for color sensors) is therefore ideal. Moreover, the drawn quadrilateral has its proportions exaggerated, in order to be more visible on the screen. It can't correspond exactly to reality.

Fig. 1: Display of the tilt diagram

**Tip**

As well as using the *tilt -clear* command, the tilt diagram can be cleared using the *Remove* button in the Dynamic PSF dialog.

**Siril command line**

```
seqtilt sequencename
```

Same command as TILT but for the sequence **sequencename**. It generally gives better results

Links: *tilt*

## 15.2 Aberration Inspector

This tool creates a 3x3 mosaic of the image center, corners and edges. This makes it easy to compare the shape of the star in different sections of the image. You can access this feature in the menu *Tools* → *Image Analysis* → *Aberration Inspector*. You can change the settings of this tool, to modify the size of the panels and the window, in the *preferences*.

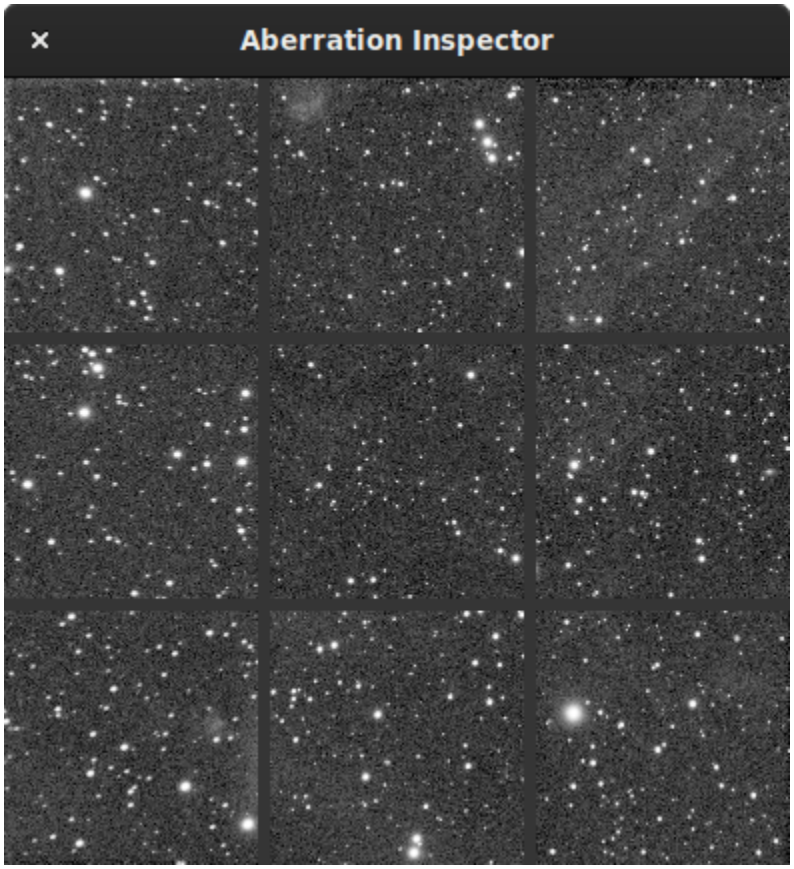



Fig. 2: Aberration inspector window showing aberration in the stars located in the edges because of the optical system.

It is also a very good indicator to know if the image contains a gradient: the differences in brightness becoming very visible.

**Siril command line**

```
inspector
```

Splits the loaded image in a nine-panel mosaic showing the image corners and the center for a closer inspection (GUI only)

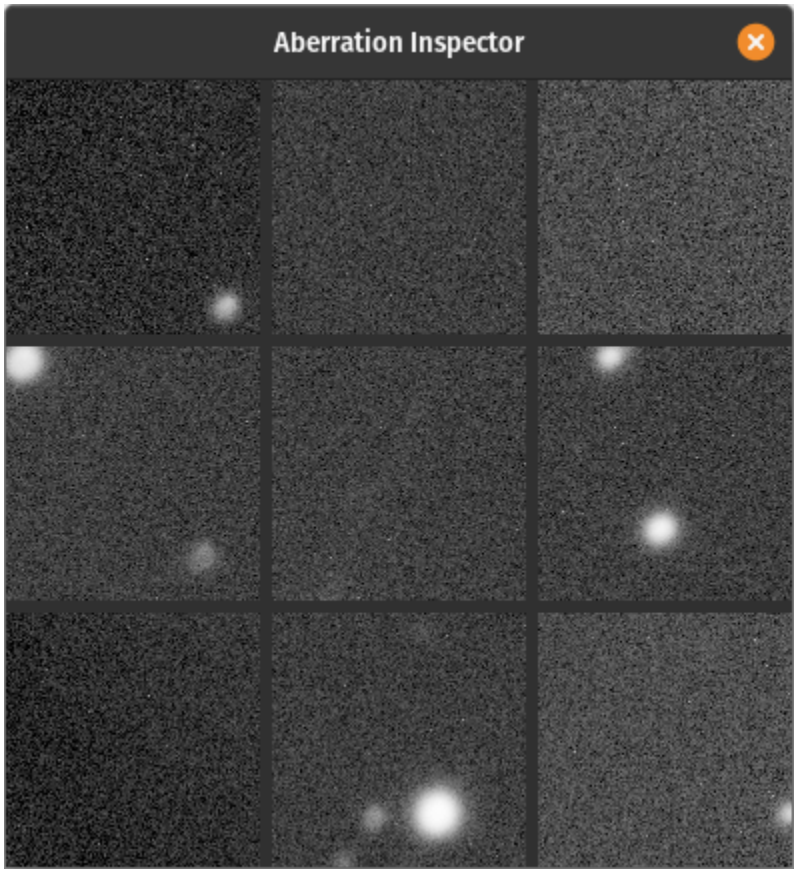



Fig. 3: Aberration inspector window showing differences in brightness.

CHAPTER

SIXTEEN

# PHOTOMETRY

This section introduces you to all the utilities related to photometry, first explaining the principles of photometry, then how it is used in Siril.

Siril is able to determine the magnitude of stars as well as its uncertainty. From there it is possible to study the variability of certain stars, exoplanets, or occultations. A light curve is also built at the end of the process.

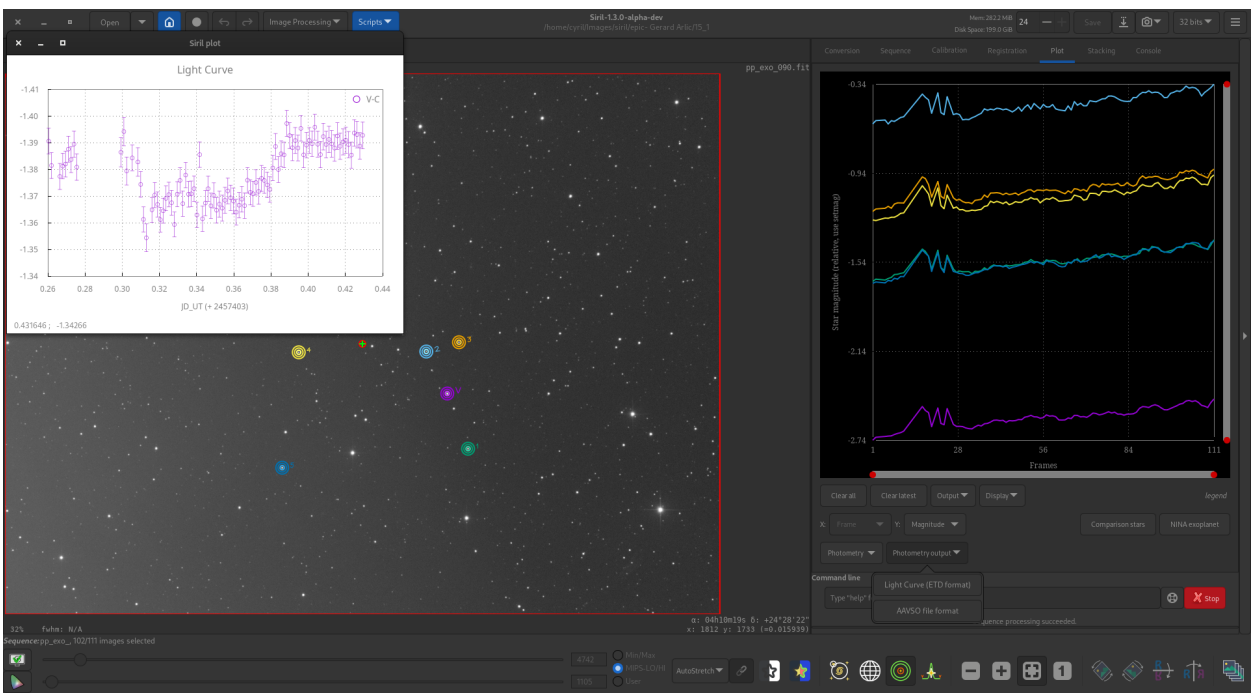



Fig. 1: Example of exoplanet photometry in Siril.

## 16.1 Principles

Photometry is the science of the measurement of light. It aims to measure the flux or intensity of light radiated by astronomical objects. In Siril, photometry can be used to analyze the light curve of variable stars, transits of exoplanets or occultations of stars, or to calibrate colors in RGB images.

Aperture photometry is the method used. Its basic principle is to sum-up the observed flux in a given radius from the center of an object, then subtract the total contribution of the sky background in the same region (calculated in the ring between the inner and outer radii, excluding the deviant pixels), leaving only the flux of the object to calculate an instrumental magnitude. This is illustrated in the following figure.

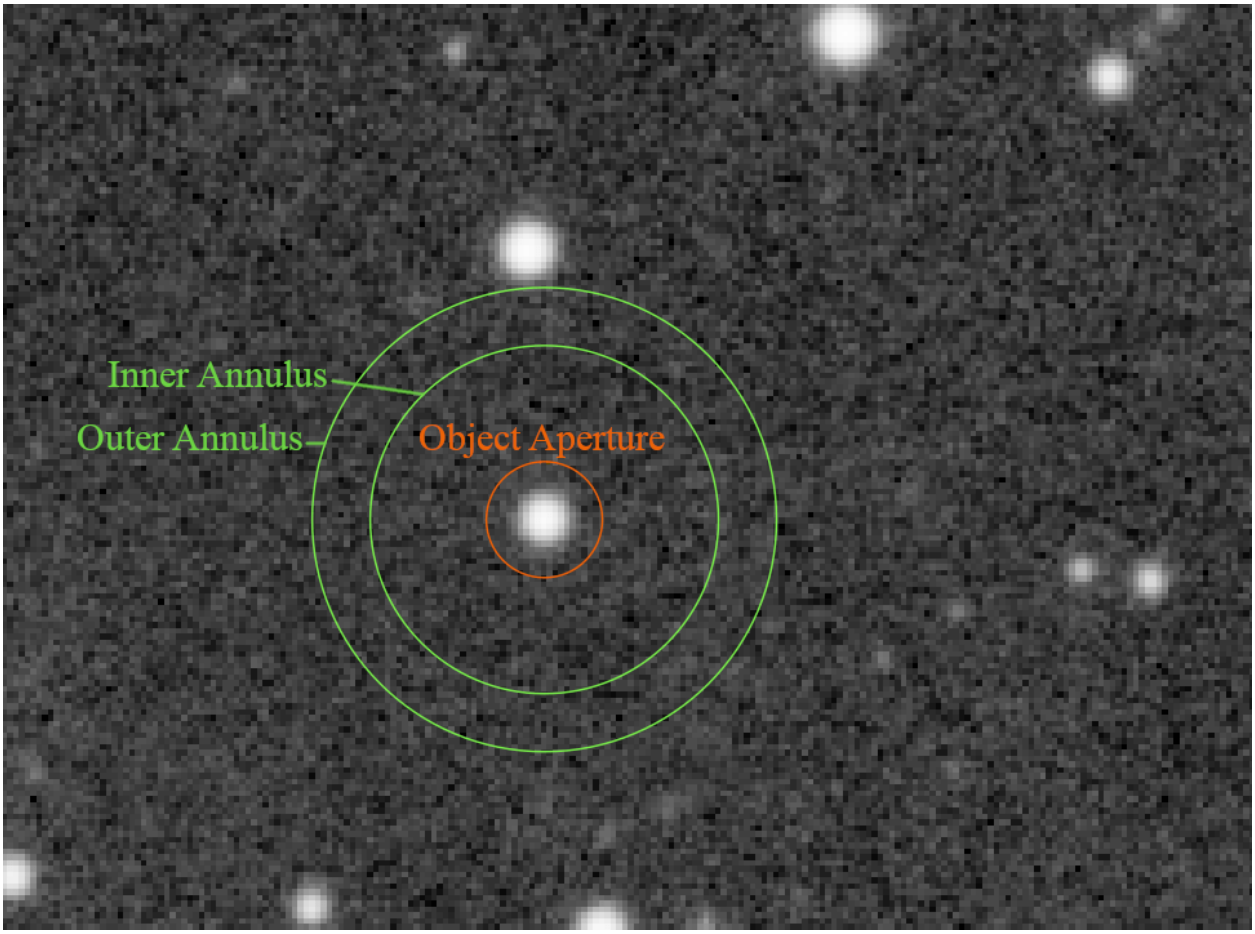



Fig. 2: Circles of the aperture photometry

The values of these settings can be changed in the *Photometry* section of preferences or using the `setphot` command. The *aperture* must contain all pixels of the object to measure, the *annulus* should by opposition not contain any of its pixels. By default, the *aperture* is adjusted for a target using twice the PSF's FWHM, but the annulus size is fixed. These values should be adjusted for a given sampling, and checked with care.

> **Note**
>
> The following text is a truncated and modified copy of the excellent MuniPack software documentation, from David Motl and released under the GNU Free Documentation License, whose sources are available here.

### Measuring magnitude of an object

The sum $S$ of pixels in a small area $A$ around an object is a sum of the object's net intensity $I$ plus background intensity $B \cdot A$:

$$S = I + B \cdot A \tag{16.1}$$

The values of $S$ and $B$ are derived from the source frame, the area $A$ is determined as the area of circle of radius $r$, where $r$ is the size of the aperture in pixels. It is then easy to compute the net intensity $I$ of an object in ADU:

$$I = S - B \cdot A \tag{16.2}$$

Supposing that the net intensity $I$ is proportional to the observed flux $F$, we can derive the apparent magnitude $m$ of the object, utilizing the Pogson's law:

$$m = -2.5 \log_{10} \left( \frac{I}{I_0} \right) \tag{16.3}$$

### Estimating the measurement error

Once we have derived the raw instrumental brightness of an object, we will try to estimate its standard error. First of all, we will recall a few general rules that apply to the standard error and its propagation. This is a general rule for error propagation through a function $f$ of uncertain value $X$:

$$\mathrm{Var}(f(X)) = \left( \frac{df}{dx} \right)^2 \mathrm{Var}(X) \tag{16.4}$$

Using this general rule, we derive two laws of error propagation. In the first case, the uncertain value $X$ is multiplied by a constant $a$ and shifted by a constant offset $b$. This law can also be used in the case where only a multiplication or only an offset occurs.

$$\mathrm{Var}(aX + b) = a^2 \, \mathrm{Var}(X) \tag{16.5}$$

The second law defines the error of a logarithm of uncertain value $X$:

$$\mathrm{Var}(\log(\pm bX)) = \frac{\mathrm{Var}(X)}{\bar{X}^2} \tag{16.6}$$

Please note, that the *log* function here is the natural logarithm, while the Pogson's formula (see above) incorporates the base-10 logarithm. The following equation helps us to deal with this difference:

$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)} \tag{16.7}$$

Putting these two equations together we get:

$$\mathrm{Var}(\log_{10}(\pm bX)) = \frac{\mathrm{Var}(X)}{\bar{X}^2 \, \log(10)^2} \tag{16.8}$$

If we have two uncorrelated uncertain variables $X$ and $Y$, the variance of their sum is the sum of their variances, this equation is known as Bienaymé formula.

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) \tag{16.9}$$

From this formula, we can also derive the standard error of a sample mean. If we have $N$ observations of random variable $X$ with sample-based estimate of the standard error of the population $s$, then the standard error of a sample mean estimate of the population mean is

$$SE_{\bar{X}} = \frac{s}{\sqrt{N}} \tag{16.10}$$

Armed with this knowledge, we can start thinking about the estimation of standard error of object brightness. We will consider the following three sources of uncertainty: (1) random noise inside the star aperture that includes the thermal noise of the detector, read-out noise of the signal amplifier and the analog-to-digital converter, (2) Poisson statistics of counting of discreet events (photons incident on a detector) that occur during a fixed period of time and (3) the error of estimation of mean sky level.

For the estimation of mean sky level, we have used the robust mean algorithm. It allows to estimate its sample variance $\sigma^2_{pxl}$. This is a pixel-based variance and because we have summed together $A$ pixels in the star aperture, the Bienaymé formula applies, the sum $S$ is a sum of $A$ uncorrelated random variables, each of which has variance $\sigma^2_{pxl}$. For the variance of the first source of error we get:

$$\sigma_1^2 = A\,\sigma^2_{pxl} \tag{16.11}$$

where $A$ is a number of pixels in the star aperture.

From Poisson statistics we can derive a variance that occur due to counting of discreet events, photons incident on a detector, that occur during a fixed period of time, the exposure. We will again need to use the gain $p$ of the detector to convert a intensity in ADU to a number of photons. If the measured net intensity of an object is $I$ we compute the mean number of photons $\lambda$ as

$$\lambda = I\,p \tag{16.12}$$

**Note**

The value of the gain $p$ of the detector can be changed in the *Photometry* section of Siril's preferences

Then, the variance of intensity due to Poisson statistics is equal to its mean value.

$$\sigma^2_{ph} = \mathrm{Var}(\mathrm{Pois}(\lambda)) = \lambda = I\,p \tag{16.13}$$

The variance is in photons, we have to convert it back to ADU to get the variance in units $ADU^2$.

$$\sigma_2^2 = \frac{\sigma^2_{ph}}{p^2} = \frac{I\,p}{p^2} = \frac{I}{p} \tag{16.14}$$

We have derived the sky level as a sample mean of pixel population in the sky annulus. Because each pixel in the annulus has variance $\sigma^2_{pxl}$, the variance of sample mean is

$$s^2_{sky} = \frac{\sigma^2_{pxl}}{n_{sky}} \tag{16.15}$$

where $n_{sky}$ is the number of pixels in sky annulus.

From equation (16.9) we compute the variance of object's intensity as

$$\sigma_{ADU}^2 = \sigma_1^2 + \sigma_2^2 + A^2\, s_{sky}^2 \tag{16.16}$$

Note, that in equation (16.2) the sky level is multiplied by $A$, so we have to multiply its variance by $A^2$ - see the equation (16.16). Now, we use the law of error propagation for the logarithm adopted to match the formula of the Pogson's law.

$$\sigma_{mag}^2 = \left(\frac{-2.5}{I\ \log(10)}\right)^2 \sigma_{ADU}^2 \tag{16.17}$$

Putting equations (16.17) and (16.16) together, we can derive the standard error of the object's brightness in magnitudes as

$$\sigma_{mag} = \frac{1.08574}{I}\sqrt{\sigma_{ADU}^2} \tag{16.18}$$

## 16.2 Quick photometry

### 16.2.1 Photometry areas explained

As Siril only performs aperture photometry, it's important to understand and set the correct parameters.

The stars are modeled according to your choice: Gaussian or Moffat as decribed *here* in the *Dynamic PSF* chapter.

So a particular star on a particular image has its own FWHM.

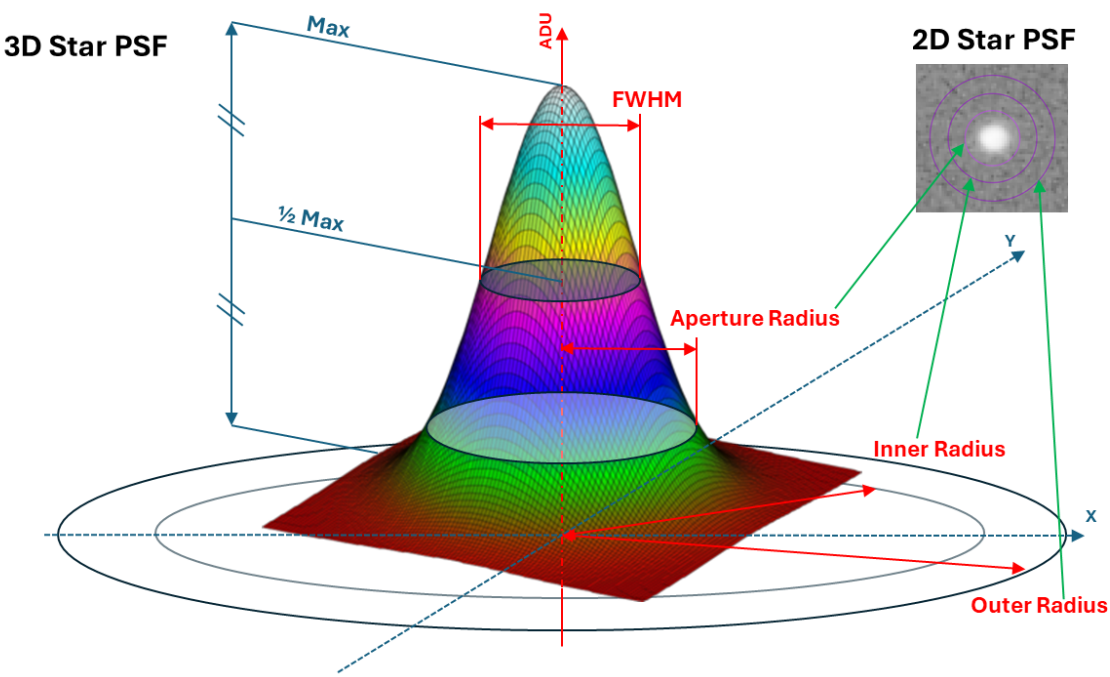



Fig. 3: PSF and photometry terms.

The aperture photometry process needs 3 radii:

1. The outer circle, defined by the **outer radius** (most often expressed in pixel).
2. The inner circle, defined by the **inner radius** (most often expressed in pixel). These 2 circles define the annulus which is used to measure the background (that is the sky level + additionnal noise).

3. The aperture circle, defined by the **aperture radius**. It can be expressed in pixel or as a ratio of the star FWHM (the automatic checkbox in the window). This area is used to measure the star signal added to a backgroud signal.

These 3 radii can be set and tweaked individually from the GUI *Preferences →Photometry* tab.

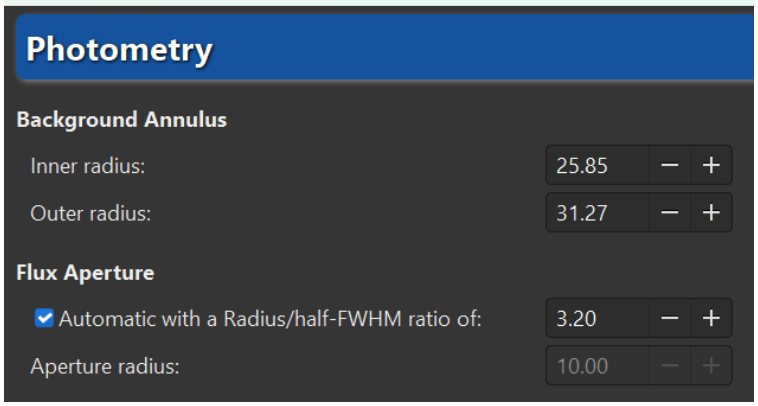



Fig. 4: Radii settings.

Or via the command line interface:

**Siril command line**

```
setphot [-inner=20] [-outer=30] [-aperture=10] [-dyn_ratio=4.0] [-gain=2.3] [-min_
↪val=0] [-max_val=60000]
```

Gets or sets photometry settings, mostly used by SEQPSF. If arguments are provided, they will update the settings. None are mandatory, any can be provided, default values are shown in the command's syntax. At the end of the command, the active configuration will be printed.

The Aperture size is dynamic unless it is forced. If so, the **aperture** value from the settings is used. If dynamic, the radius of the aperture is defined by the supplied dynamic ratio ("radius/half-FWHM").
Allowed values for the argument **-dyn_ratio** are in the range [1.0, 5.0]. A value outside this range will automatically set the aperture to the fixed value **-aperture**.

Gain is used only if not available from the FITS header

Links: *seqpsf*

**Tip**

These radii settings apply to all aperture photometric processes: **quick photometry** and **Light Curves**. Check them carfully.

### 16.2.2 Photometry on hand-picked objects of a single image

The **quick photometry** button is a button located in the toolbar and used to perform a photometry of the stars, this is generally the simpliest way to proceed.

**Tip**

If the star is in the middle of several stars and the tool fails to point to the right star, an alternative solution is to draw a selection around the star and then right-click and click on *PSF*. It may also be interesting to know that `Ctrl`-middle click (or `Cmd`-middle click on MacOS) draws a selection of a recommended size for PSF/photometry (based on the configured outer radius).

**Tip**

When photometry is performed on the RGB layer, the results are actually calculated on the green layer. To obtain photometry on the red or blue layers, you need to work on the corresponding channels.

**Siril command line**

```
psf [channel]
```

Performs a PSF (Point Spread Function) on the selected star and display the results. For headless operation, the selection can be given in pixels using BOXSELECT. If provided, the **channel** argument selects the image channel on which the star will be analyzed. It can be omitted for monochrome images or when run from the GUI with one of the channels active in the view

Links: *boxselect*

Click on this button to change the image selection mode, then click on a star. The photometry and the PSF (Point Spread Function) of the star are computed, giving plenty of details.

Two models are used for the calculation of the PSF, which can be selected by the user in the *Dynamic-PSF* window (`Ctrl` + `F6`).

The result of the photometry and the associated PSF are displayed in the form:

```
PSF fit Result (Gaussian, monochrome channel):

Centroid Coordinates:
            x0=5258.25px     09h25m34s J2000
            y0=2179.72px    +69°49'31" J2000
```

(continues on next page)

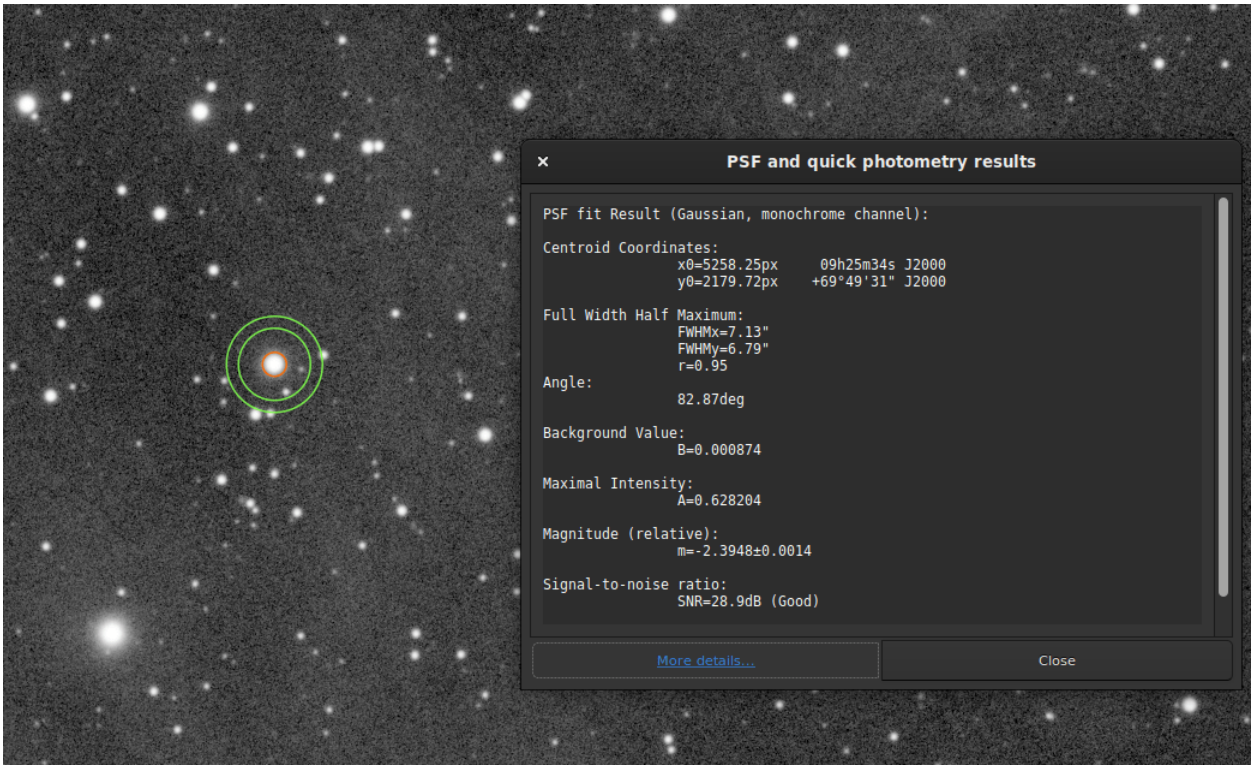



Fig. 5: Photometry results window.

(continued from previous page)

```
Full Width Half Maximum:
            FWHMx=7.13"
            FWHMy=6.79"
            r=0.95
Angle:
            82.87deg

Background Value:
            B=0.000874

Maximal Intensity:
            A=0.628204

Magnitude (relative):
            m=-2.3948±0.0014

Signal-to-noise ratio:
            SNR=28.9dB (Good)

RMSE:
            RMSE=1.890e-03
```

1. The fit was done with the **Gaussian** fitting function so no additional parameters are needed. However, if Moffat was used, the following output will be shown:

```
PSF fit Result (Moffat, beta=2.9, monochrome channel):
```

2. **Centroid Coordinates** gives the coordinates of the centroid in pixels. However, like in the example above, if astrometry was set on the image, Siril gives coordinates in the World Coordinate Systems (RA and Dec).
3. **Full Width Half Maximum** (FWHM) is returned in arcsec if the image scale is known (obtained from its header or from the GUI *Image information* → *Information*) and in pixels if not. The roundness $r$ is also computed as the ratio of $\frac{\text{FWHMy}}{\text{FWHMx}}$.
4. **Angle** is the rotation angle of the $X$ axis with respect to the centroid coordinates. It varies in the range $[-90, +90]$.
5. **Background Value** is the local background in the $[0, 1]$ range for 32-bits images and $[0, 65535]$ for 16-bits images. This is a fitted value, not the background computed in the aperture photometry annulus.
6. The **maximum Intensity** value is also a fitted value and represents the amplitude. It is the maximum value of the fitted function, located at the centroid coordinates.
7. The **magnitude**, given with its uncertainty, is the result of photometry. However, if for some reasons the calculation cannot be done (saturated pixels or black pixels), an uncertainty of **9.999** is given. In this case, the photometry is flagged as invalid but a magnitude value is still given, although it should be used with caution.
8. An estimator of the **signal-to-noise ratio** is shown in the results. Its value is calculated from the following formula and given in dB:

$$\text{SNR} = 10 \log_{10} \left( \frac{I}{N} \right) \tag{16.19}$$

   where $I$ is the net intensity, proportional to the observed flux $F$ and $N$ the total of uncertainties as expressed in (16.18).

   For easier understanding, it is associated with 6 levels of quality:

   1. Excellent (SNR > 40dB)
   2. Good (SNR > 25dB)
   3. Satisfactory (SNR > 15dB)
   4. Low (SNR > 10dB)
   5. Bad (SNR > 0dB)
   6. N/A

      This last notation is displayed only if the computation failed, for one reason or another.
9. Finally, **RMSE** gives an estimator of the fit quality. The lower the value, the better the result.

When the image is plate-solved, the button *More details* at the bottom of the window links to a page on the SIMBAD website with information about the selected star. However, it is possible that the page does not give any additional information if the star is not in the SIMBAD database.

### 16.2.3 Quick photometry on sequences

Quick photometry can also be performed on a sequence. This is generally intended to obtain a light curve as explained *here*. To proceed, you must **load a sequence**, make a selection around a star, then **right click** on the image.

> **Tip**
>
> Ideally, the sequence must be registered without interpolation so as not to alter the raw data. For example, use the **global registration** with the option **Save transformation in seq file only**.

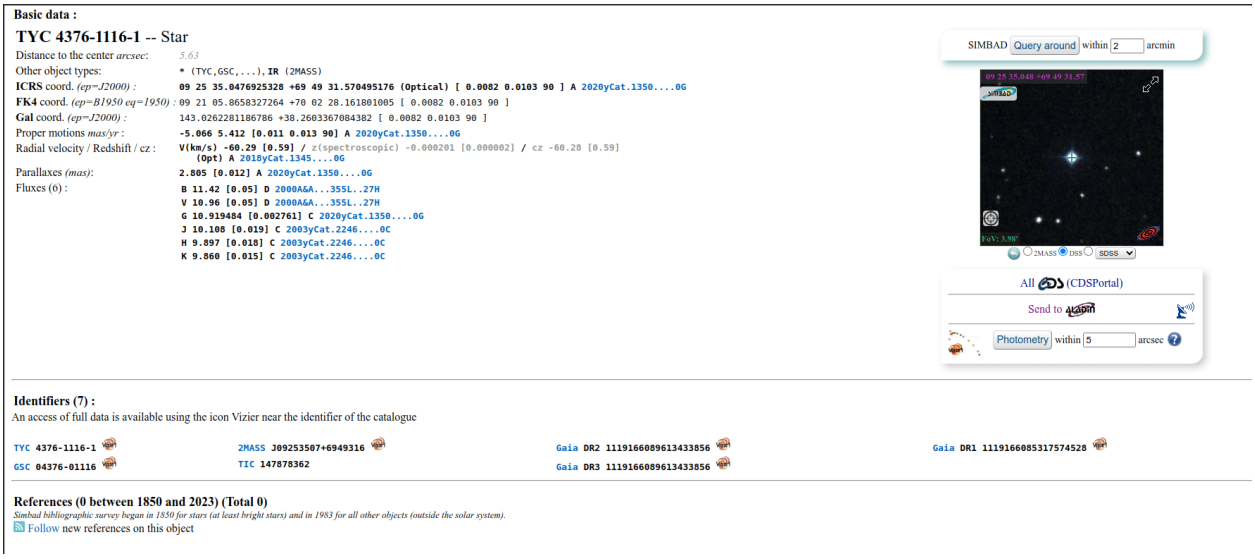



Fig. 6: More details about the analyzed star. Click on the picture to enlarge.

> **Note**
>
> Make sure the inner and outer radii for the background annulus are adapted to the star and sequence being analyzed. Some images may have much larger FWHM than the reference image, because of sky conditions or bad tracking. They can be changed in the *preferences* or with the `setphot` command.

At the end of the process, Siril automatically opens the plot tab showing computed curves. It is possible to click on several stars to reproduce the calculation, however the first star keeps the particular status of *variable*, and the others serve as *references*. This is important in the calculation of the light curve.

### 16.2.4 Computing true magnitudes

The calculated magnitude is only meaningful if it is compared to others in the linear image. Indeed, the value given does not correspond at all to the true visible magnitude of the star, it is uncalibrated, also called relative magnitude.

Siril provides tools that can be used to calculate an approximate apparent magnitude. This requires knowing the magnitude of another star visible on the image that will act as reference. It is currently possible to use only a single star as reference, hence the *approximate*. For a greater precision, use a star of similar color and magnitude as the star(s) you want to measure should be chosen, and its provided magnitude should be in adequation with the filter used to capture the image. Catalogs contain magnitudes computed using a photometric filters, which is generally not what amateur use to make nice pictures, this adds another approximation.

- Do a quick photometry on a known star, the given relative magnitude is `-2.428`. It is possible to find out the actual visible magnitude by clicking on the *More details* button as explained above. Let's say the value found is `11.68` (make sure you use a value corresponding to the spectral band of the image).
- Once done, keep the star selected, then enter the following command in Siril

  ```
  setmag 11.68
  ```

  That will output something like

  ```
  10:50:49: Relative magnitude: -2.428, True reduced magnitude: 11.680,␣
  ↪Offset: 14.108
  ```

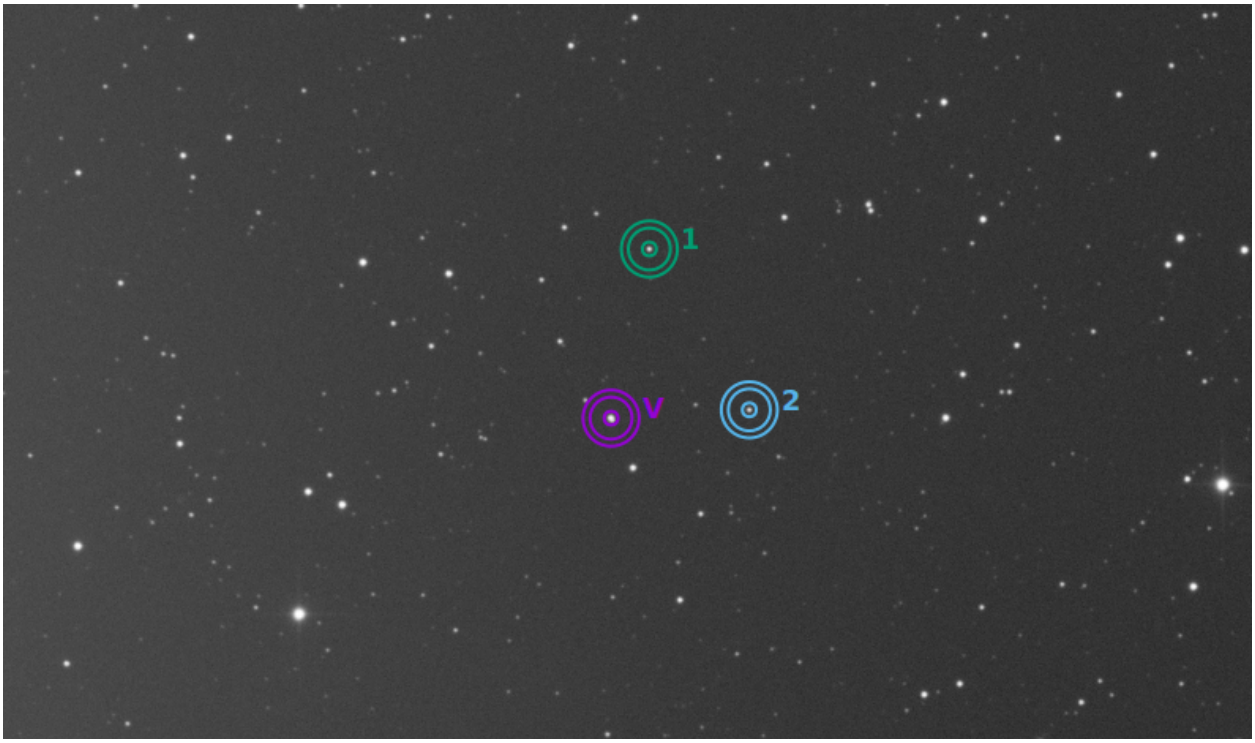



Fig. 7: In this example, 3 stars have been analyzed. The first one is used as variable. The others are references.

**Siril command line**

```
setmag magnitude
```

Calibrates the magnitudes by selecting a star and giving the known apparent magnitude.

All PSF computations will return the calibrated apparent magnitude afterwards, instead of an apparent magnitude relative to ADU values. Note that the provided value must match the magnitude for the observation filter to be meaningful.
To reset the magnitude constant see UNSETMAG

Links: *psf*, *unsetmag*

- Now, all calculated magnitudes must have values close to their true visual magnitude. However, this is especially true for stars whose magnitude is of the same order of magnitude as the star taken as reference.
- To unset the computed offset, just type

```
unsetmag
```

**Siril command line**

```
unsetmag
```

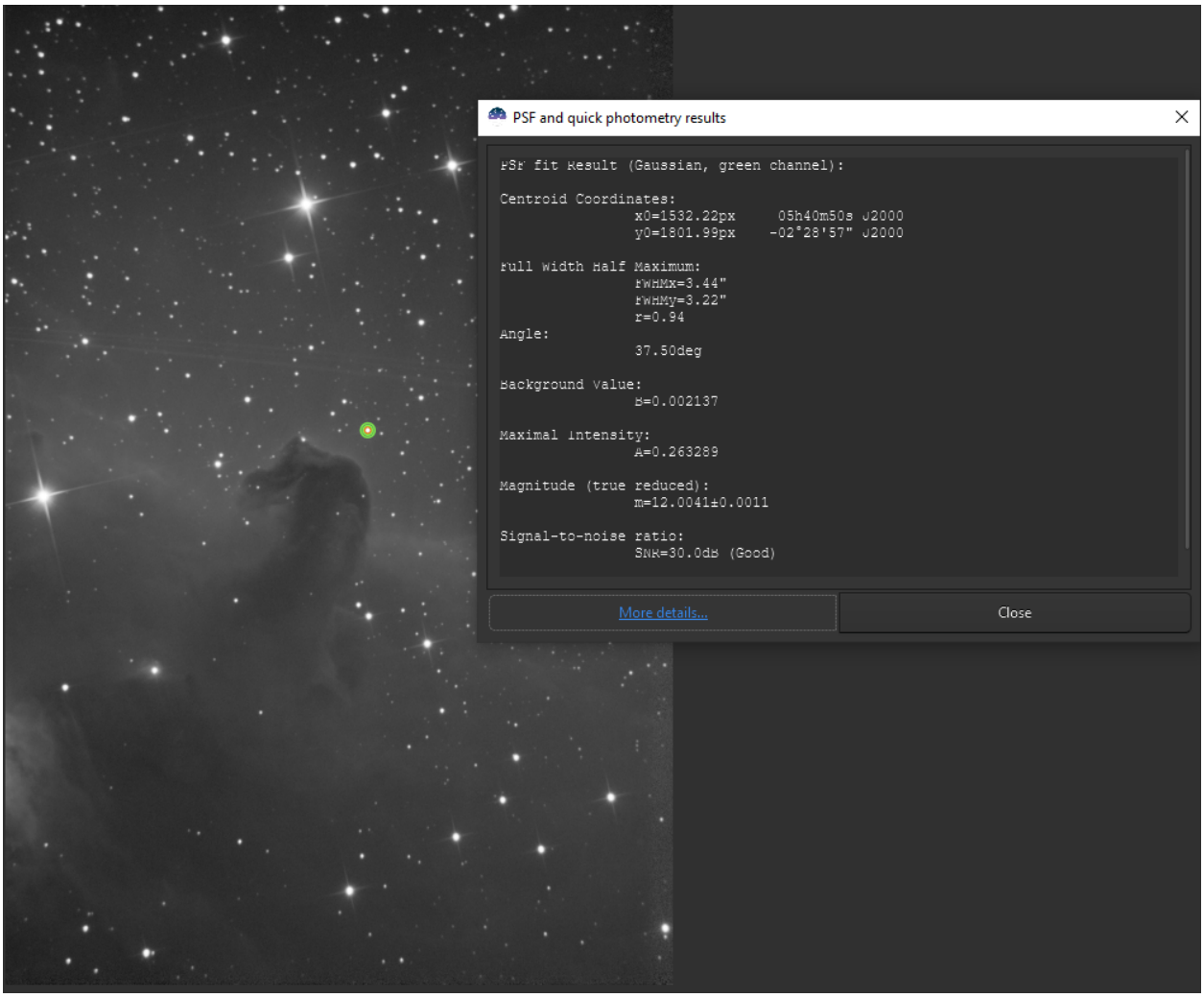



Fig. 8: Photometry results window with true magnitude set.

Resets the magnitude calibration to 0. See SETMAG

Links: *setmag*

**Tip**

The same commands exist for the sequences. They are `seqsetmag` and `sequnsetmag`. It is used in the same way when a sequence is loaded.

**Siril command line**

```
seqsetmag magnitude
```

Same as SETMAG command but for the loaded sequence.

This command is only valid after having run SEQPSF or its graphical counterpart (select the area around a star and launch the PSF analysis for the sequence, it will appear in the graphs).
This command has the same goal as SETMAG but recomputes the reference magnitude for each image of the sequence where the reference star has been found.
When running the command, the last star that has been analysed will be considered as the reference star. Displaying the magnitude plot before typing the command makes it easy to understand.
To reset the reference star and magnitude offset, see SEQUNSETMAG

Links: *setmag*, *seqpsf*, *psf*, *sequnsetmag*

**Siril command line**

```
sequnsetmag
```

Resets the magnitude calibration and reference star for the sequence. See SEQSETMAG

Links: *seqsetmag*

## 16.3 Light curves

In astronomy, a light curve is a graph of light intensity of a celestial object as a function of time, typically with the magnitude of light received on the y axis and with time on the x axis. Siril is able to generate such curves when analysing stars.

There are now two ways of selecting the variable and references (also called comparison) stars: manually, or using a list of stars obtained by the N.I.N.A. exoplanet plugin.

### 16.3.1 Manual star selection

Start by switching to the **Photometry Mode** (click on ).

Select (right click) the variable star followed by the comparison stars you wants, as explained *here*.

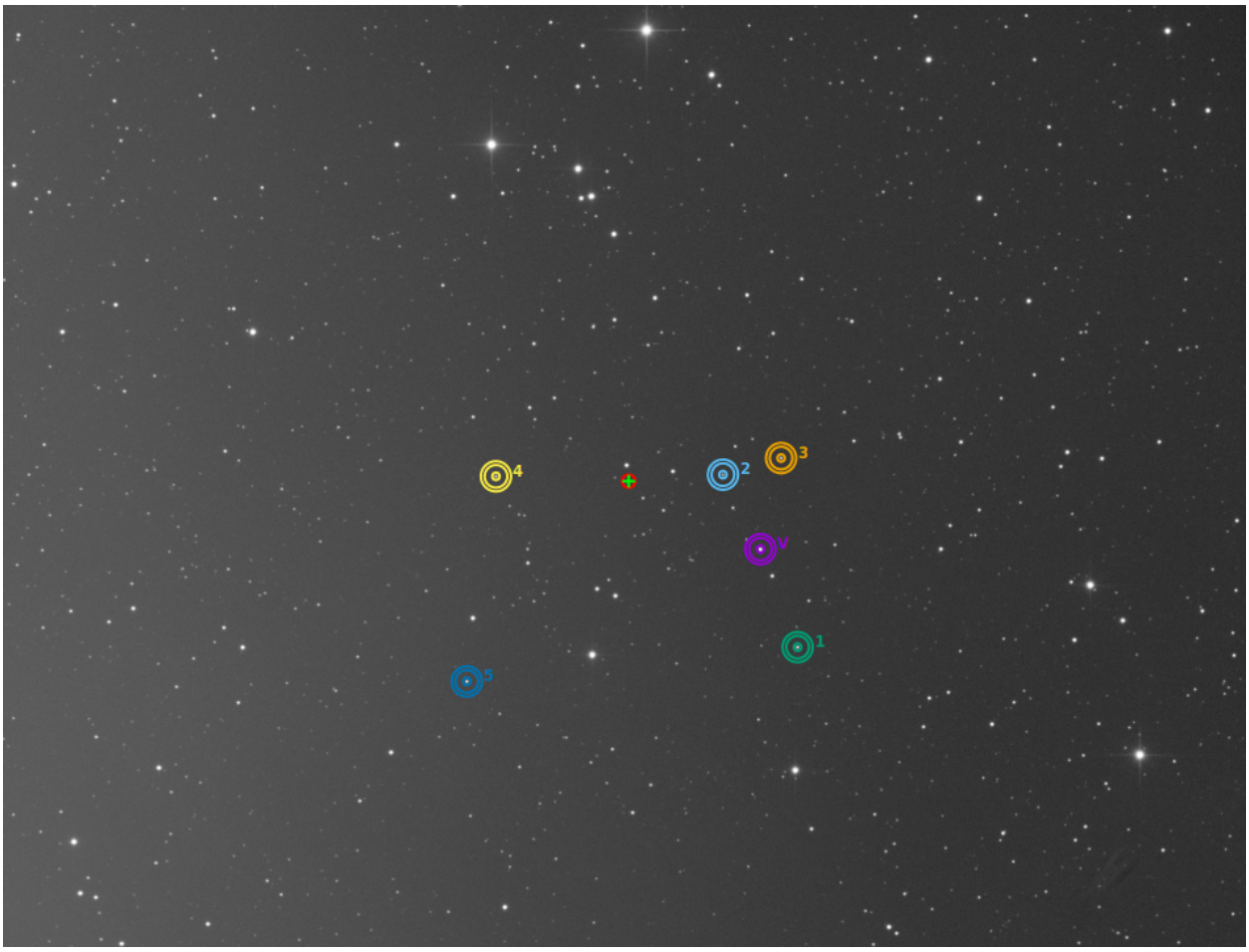



Fig. 9: One star is the variable (purple with a V) and the 5 others are used as references.

**Warning**

Make sure to not select variable stars for references. If the astrometry is done on your image, do not hesitate to use the *SIMBAD request* to know more about the stars.

One handy way to check your comparison star is to display the 3 variable catalogues related to variable stars: GCVS, AAVSO-VSX and GAIA-varisum.

You have to set your own parameter (limit magnitude), but as an example, here are the commands:

```
conesearch 18.0 -cat=gcvs
conesearch 18.0 -cat=aavso_chart
conesearch 18.0 -cat=varisum
```

> **Tip**
>
> It is preferable to choose references whose magnitude is close to that of the variable as explained later *here*.

Once done, Siril automatically loads the Plot tab as shown in the figure below. This shows FWHM curves expressed as a function of frame number.

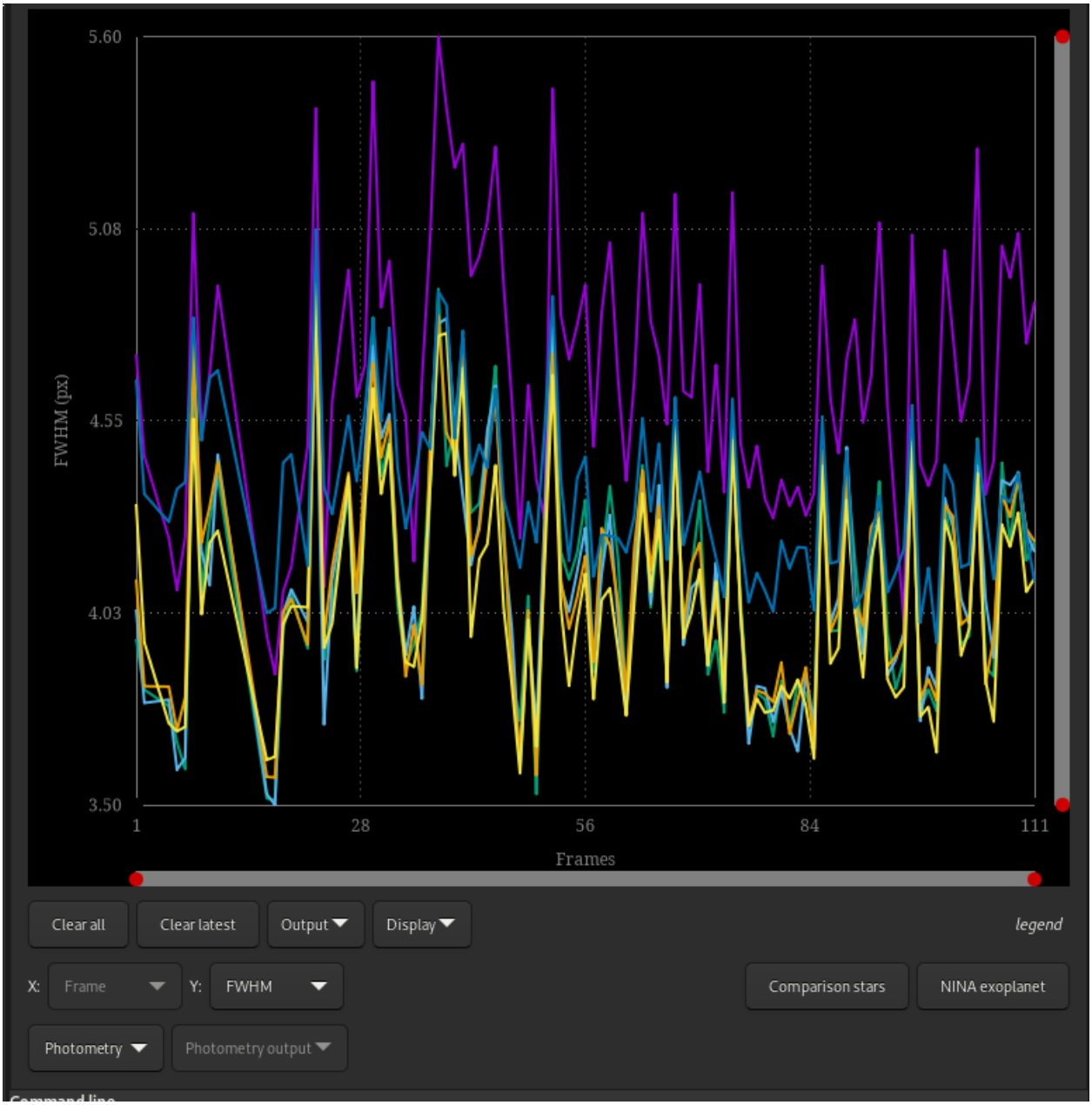



Fig. 10: The plot tab as showed right after the quick photometry on sequence.

What interests us in this part is to display the magnitude curves. Simply go to the drop-down menu and change **FHWM** to **Magnitude**. The magnitude curves of each analyzed star are then displayed. The submenus located in *Photometry* of the *Tools* menu are also sensitive.

### Saving the selected comparison stars

When satisfied with the comparison stars set, one can to save them for future use in an *Automated Light Curve* process or just keep a trace for further documentation.

In the *Tools* menu, select *Photometry* → *Create Comparison Stars File...*.

The default *Use the stars in the currently loaded image* is selected.

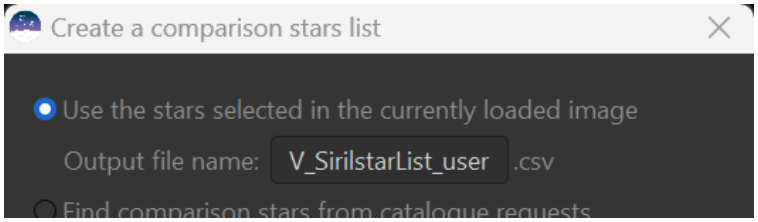



Fig. 11: Saving the hand selected stars in a .csv file.

The default output file name will be `V_SirilstarList_user.csv` but you can change it according to your convenience.

After you click *OK*, the file will be created and saved in your current working directory.

### Light Curve (ETD format)

The button *Light Curve (ETD format)*, located in the menu *Tools* → *Photometry* → *Create Output* is dedicated to exoplanet transit data, producing a file in the format managed by the ETD website.

Once the analysis is completed with a number of reference stars of at least 4 or 5 (the higher the number, the more accurate the result. However, a maximum number of 19 stars is allowed). Siril will ask for a file name to save the data in `csv` format, then the light curve will be displayed in a new window.

```
#JD_UT (+ 2457403)
# JD_UT V-C err
2457403.260428 -1.01054 0.00468598
2457403.261956 -1.00321 0.0046727
2457403.263461 -1.00132 0.00540513
2457403.264977 -0.994067 0.00505492
2457403.266481 -1.00397 0.00465623
2457403.267998 -1.00532 0.00467666
...
```

This is a 3-column file. The first 2 lines, starting with a "#", are comments. Then, the first column is the precise time of observation in Julian Date format. The second column is the magnitude of the variable ($V$), from which the sum of the instrumental magnitudes of the comparison stars has been subtracted. The third column represents magnitude uncertainties.

**Theory**

$V$ is given by the following formula:

$$V = (V_{\text{ins}} - C_{\text{ins}}), \tag{16.20}$$

with

$$C_{\text{ins}} = -2.5 \log_{10} \left( \frac{1}{N} \sum 10^{-0.4 C_{\text{ins},i}} \right). \tag{16.21}$$

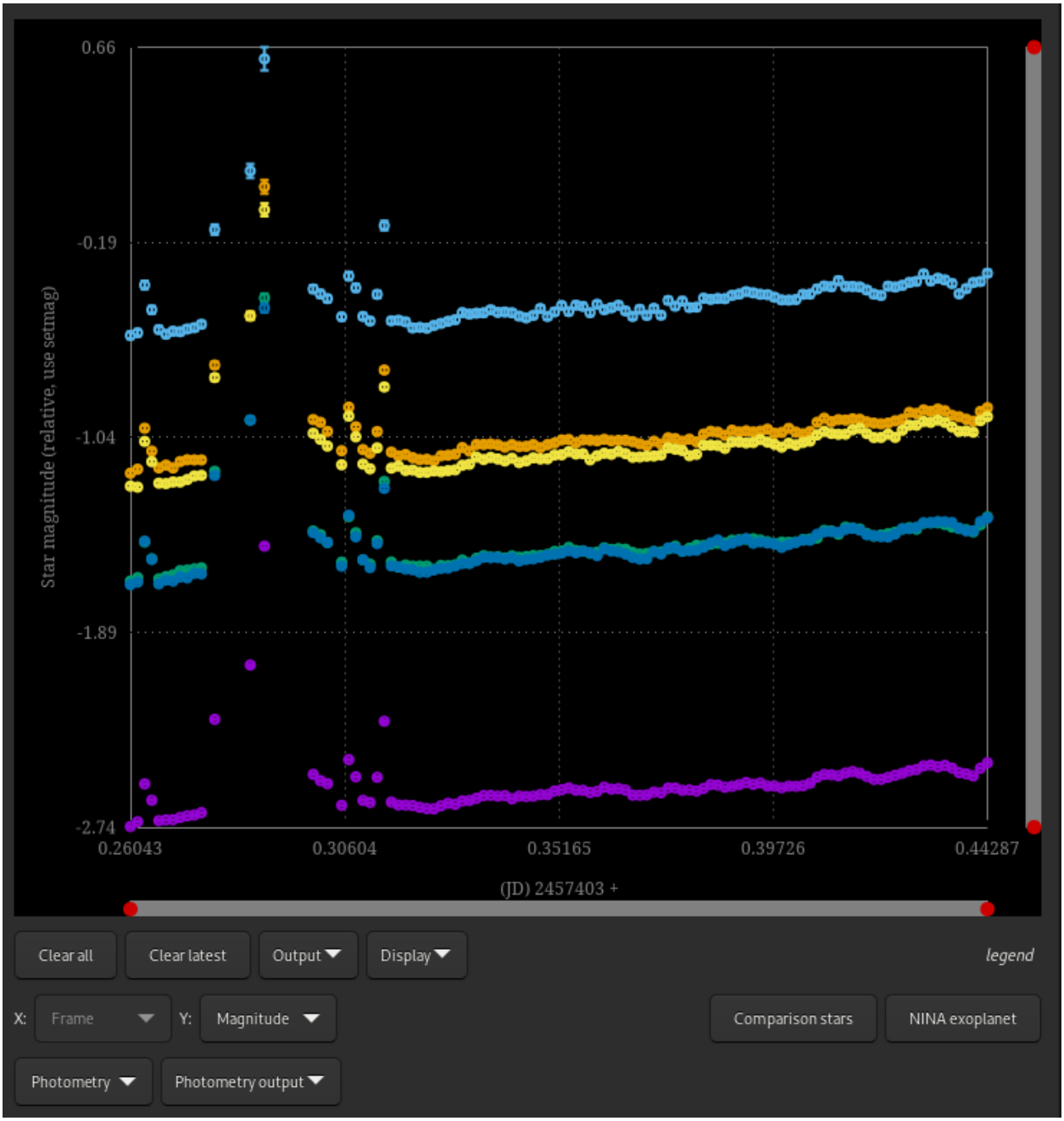



Fig. 12: Switching to magnitude view make the *Photometry output* button sensitive.

$N$ is the total number of comparison stars and $C_{\text{ins}}, i$ is the instrumental magnitude of i-th star in the ensemble.

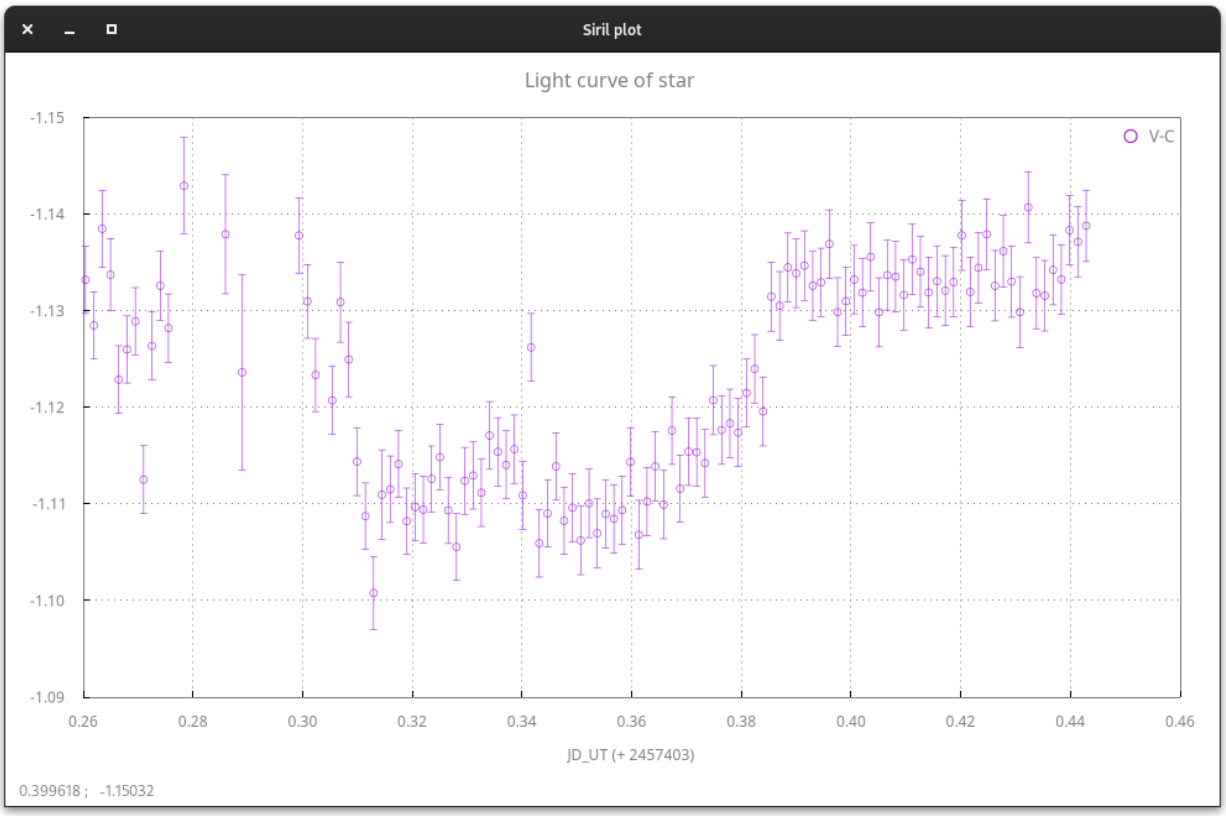



Fig. 13: Light curve of an exoplanet transit.

**Tip**

The light curve is drawn with Siril internal plotting tool. Refer to its *documentation* for more detailed instructions on its use.

### AAVSO extended file format

Located in the menu *Tools* → *Photometry* → *Create Output*, the button, *AAVSO file format*, is dedicated to the AAVSO extended file format for variable stars. This opens a new dialog, which once completed, exports a csv file that can be uploaded to the dedicated website.

**Theory**

In Siril, the AAVSO file format exported uses the standardized magnitude defined as:

$$V_{\text{std}} = (V_{\text{ins}} - C_{\text{ins}}) + C_{\text{std}} \tag{16.22}$$

where $V_{\text{ins}}$ and $C_{\text{ins}}$ are the instrumental magnitude of the variable and the comparison, respectively, and $C_{\text{std}}$ is the chart magnitude for the comparison. You can specify $C_{\text{std}}$ value in the AAVSO parameters dialog, before export.

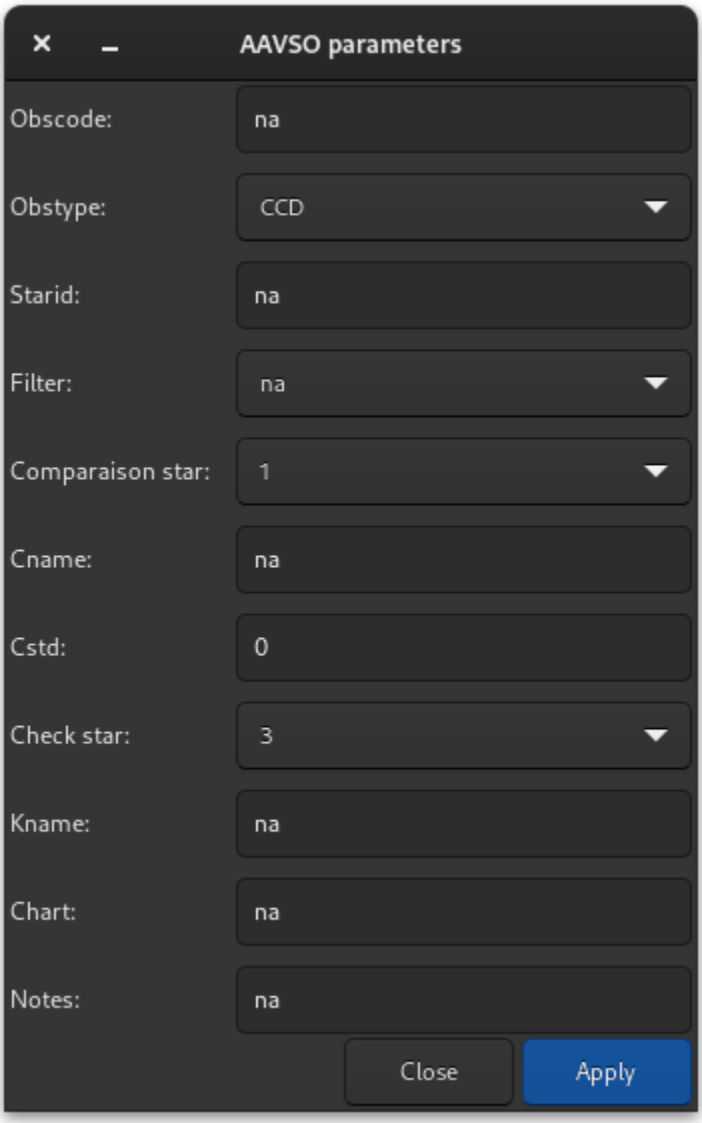



Fig. 14: AAVSO parameters dialog. This dialog is used to fill the AAVSO extended file. More information are found at the AAVSO website

An example of photometry is given below with the star Rx And. This star has great variability: its magnitude varies from 10.2 to 15.1 with a periodicity of about 13 days. After processing in Siril, taking a comparison star and a check star, we obtain the following AAVSO curve after data upload. The new data are on the far right (taken on November 24 and 25, 2023) and are shown in green.

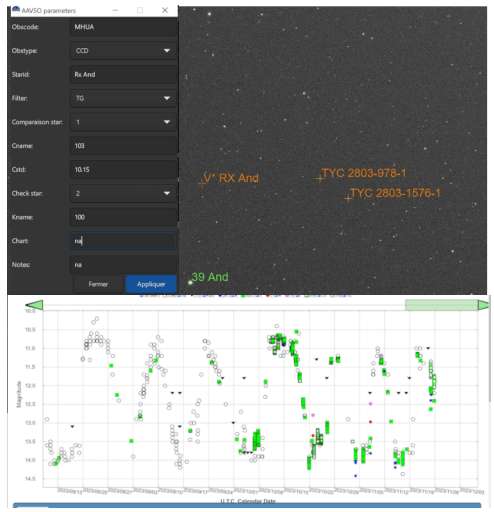



Fig. 15: A concrete example of variable star photometry (courtesy of H. Meunier).

## 16.3.2 Automatic star selection

### Automated Light Curve

In order to automate the process of exoplanet transit analysis, lists of reference stars, also called comparison stars, could be obtained from star catalogues, with the appropriate criteria: similar magnitude, similar color (to not change their relative magnitude with atmospheric extinction at different elevations), proximity.

The capture software N.I.N.A has an exoplanet plugin that will show such stars and allow the list to be saved in a CSV file, such as `csv file`:

```
Type,Name,HFR,xPos,yPos,AvgBright,MaxBright,Background,Ra,Dec
Target,HD 189733 b,2.6035068712769851,1992,1446,1640.3703703703704,39440,1917.
↪0601851851852,300.18333333333328,22.709722222222222
Var,SW Vul,2.8626145609282911,2972,276,26.14,2012,1905.445,300.02171,22.93517
Var,DQ Vul,2.372369130017419,3006,1040,28.180555555555557,2048,1906.9027777777778,300.
↪01254,22.78103
Var,HQ Vul,3.8351043206620834,157,1690,49.393939393939391,2104,1905.7454545454545,300.
↪55808,22.64067
...
Comp1,ATO J300.3222+22.7056,2.4268101078425852,1367,1465,352,4496,1913.9504132231405,300.
↪32229415181337,22.705681453738887
Comp1,HD 189657,2.5343988482845927,2527,2808,23.814814814814813,2012,1906.5061728395062,
↪300.08714683055996,22.4400393728
...
Comp2,000-BJP-946,2.2738807043120195,1832,750,29.962962962962962,2024,1910.0648148148148,
↪300.23741666666666,22.846999999999998
Comp2,000-BJP-942,2.0977710589704297,2760,1572,31.083333333333332,2096,1908.
↪6527777777778,300.025875,22.704777777777778
...
```

In the *Tools* menu, Siril can load this file using the *Automated Light Curve...* button. To use this, a few prerequisites must be met:

- the calibrated images sequence must be already loaded
- the reference image of the sequence must be plate solved, to make sure we identify the correct stars from their equatorial J2000 coordinates

From there, everything is automatic, showing the light curve for the selected comparison stars at the end of the process.

In this window, the used *aperture and background areas* are also reminded.

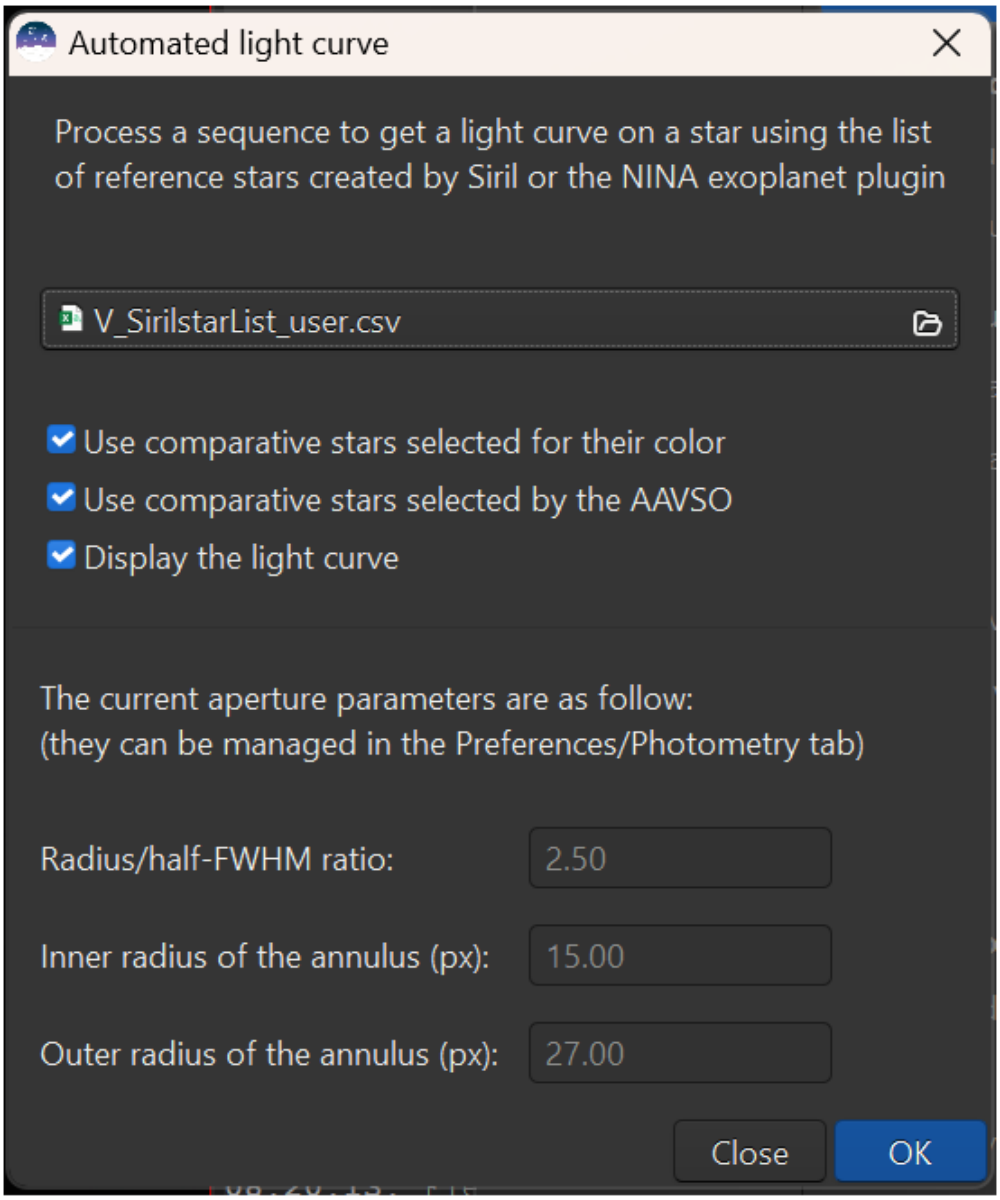



Fig. 16: The Automatic Photometry window.

The following video shows an automated processing of light curve with comparison star list from NINA:

### Light curve through Command

It is also possible to automate or create the light curve remotely using the `light_curve` command. As blind operation needs as much automation as possible, the configuration of the background annulus radii can be automated with the `-autoring` argument: it runs a star detection in the reference image and multiplies the mean FWHM with a configurable factor to obtain the inner and outer radii that should work with the sequence.

**Siril command line**

```
light_curve sequencename channel [-autoring] { -at=x,y | -wcs=ra,dec } { -refat=x,y |␣
↪-refwcs=ra,dec } ...
light_curve sequencename channel [-autoring] -ninastars=file
```

Analyses several stars with aperture photometry in a sequence of images and produces a light curve for one, calibrated by the others. The first coordinates, in pixels if **-at=** is used or in degrees if **-wcs=** is used, are for the star whose light will be plotted, the others for the comparison stars.
Alternatively, a list of target and reference stars can be passed in the format of the NINA exoplanet plugin star list, with the **-ninastars=** option. Siril will verify that all reference stars can be used before actually using them. A data file is created in the current directory named light_curve.dat, Siril plots the result to a PNG image if available
The ring radii for the annulus can either be configured in the settings or set to a factor of the reference image's FWHM if **-autoring** is passed. These autoring sizes are 4.2 time and 6.3 times the FWHM for the inner and outer radii, respectively.

See also the **setphot** command to set the same way the aperture radius size.

See also SEQPSF for operations on single star

Links: *seqpsf*

## 16.3.3 Generating a list of comparison stars

If you are not a NINA user, or you wish to build your own file and then use the *Automated light curve* feature , since Siril-1.3 you can generate a list of comparison stars using the new Siril tool.

### List through GUI

This feature can be accessed via the *Tools* menu, *Photometry* → *Create Comparison Stars File...*, once the current displayed image of the loaded sequence has been plate solved.

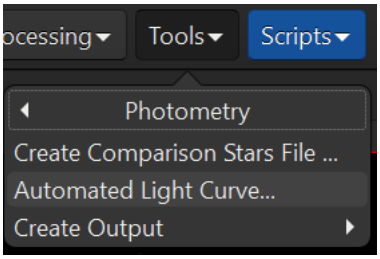



Fig. 17: The *Create Comparison Stars File...* feature in the *Tools* menu.

A prerequisite is to load a representative, plate solved, image of the sequence.

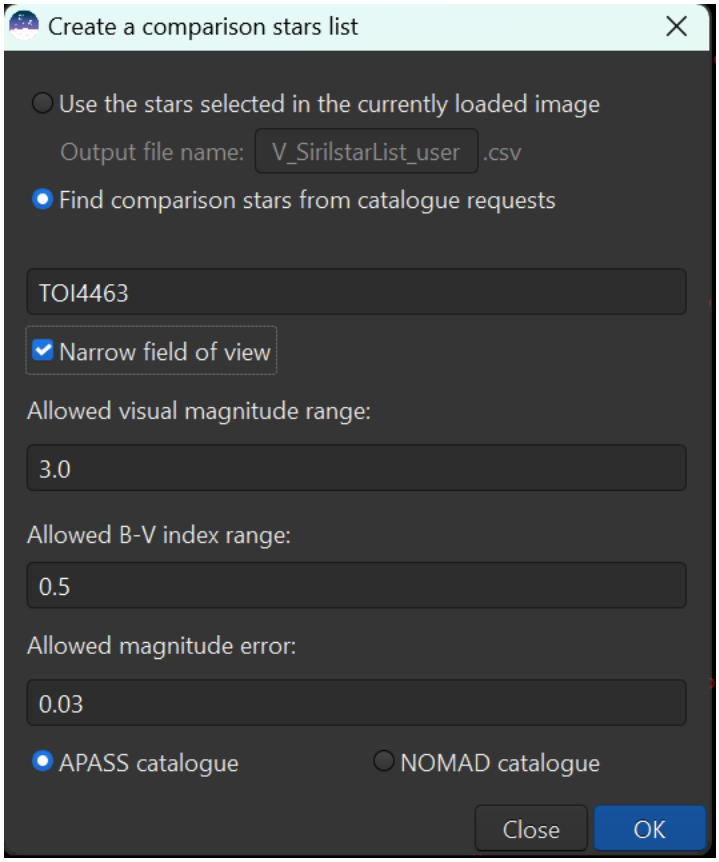
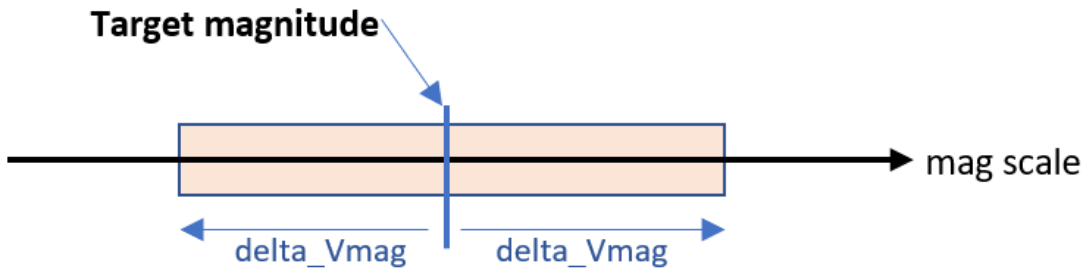
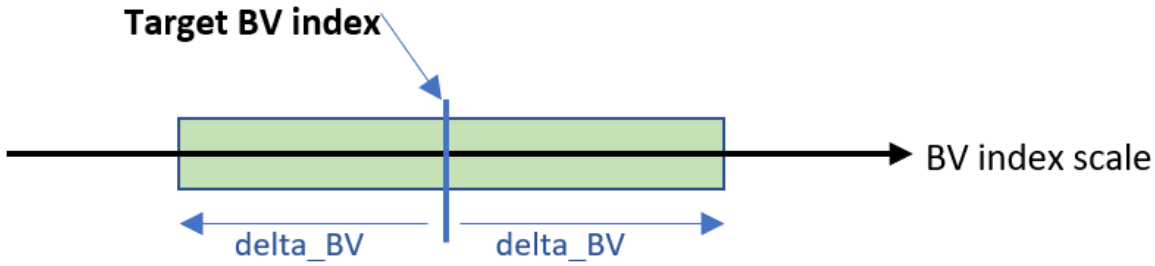



Fig. 18: Basic GUI for the Comparison Stars window.

- The **Find comparison stars from catalogue request** is mandatory in the use case.
- The first field must be the **variable star name** you plan to get the light curve of.
- The check button **Narrow field of view** enables to select a circle centered on the image.
- The second field is de **delta_Vmag** value. This is the maximum discrepancy from the target **V magnitude** (0.0 < delta_Vmag < 6.0). Default value is set to 3.0.
- The third field is the **delta_BV** value. This is the maximum discrepancy from the target **BV index** (0.0 < delta_BV < 0.7). Default value is set to 0.5.
- The fourth field is the **max_emag** value. This is the maximum allowed statistical error on Vamg (0.0 < max_emag < 0.1). Default value is set to 0.03.
- Then, you can choose the source of the comparison stars: from the APASS catalogue or NOMAD catalogue.




These two criteria help you to get comparison stars matching the target magnitude and color index.

The maximum number of comparison stars is internally set to 20.

After you click *OK*, a request is sent to the adequate server.

The answer is a list of stars, coming from the chosen catalogue, matching your criteria, displayed in red in your image and saved as a CSV file compatible with the NINA Exoplanet process.

**Note**: As the stars are red labelled, they belong to a temporary catalogue that shall be emptied after the *Annotation* button is toggled.

**Tip**

Additional information on choosing the comparison stars can be found in the DSLR Observing Manual - Version 1.4 page 67 from AAVSO.

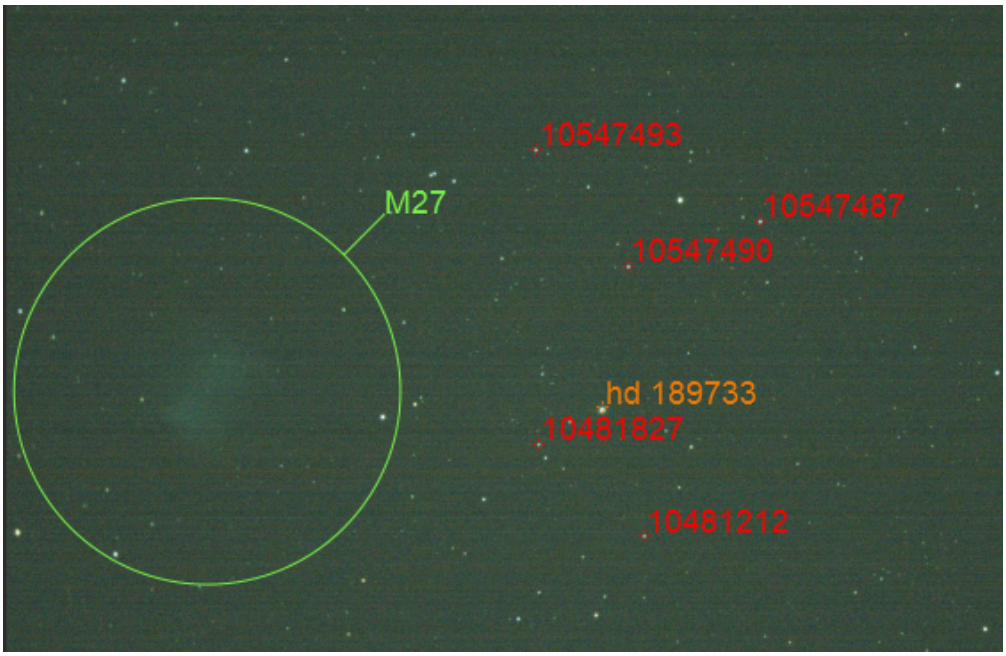



Fig. 19: Rendering example after an APASS request.

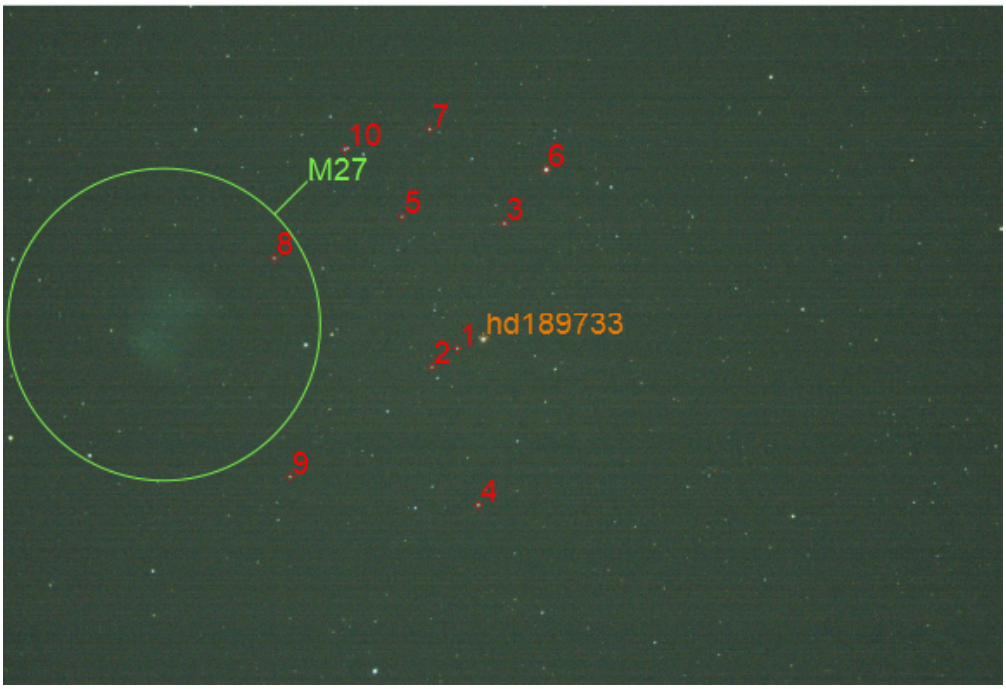



Fig. 20: Rendering example after a NOMAD request.

### List through command

A command also exists to perform the "find comparison stars" process. It allows you to refine your choice further more.

**Siril command line**

```
findcompstars star_name [-narrow|-wide] [-catalog={nomad|apass}] [-dvmag=3] [-dbv=0.
↪5] [-emag=0.03] [-out=nina_file.csv]
```

Automatically finds comparison stars in the field of the plate solved loaded image, for photometric analysis of a star's light curve according to
- the provided name of the star
- the field of view of the image, reduced to a diameter of its height if **-narrow** is passed, avoiding stars in the corners
- the chosen catalog (APASS by default), can be changed with **-catalog={NOMAD|APASS}**
- the difference in visual magnitude from the variable star, in the range [0, 6] with a default of 3, changed with **-dvmag=**
- the difference in color with the variable star, in the range [0.0, 0.7] of their B-V indices with a default of 0.5, changed with **-dbv=**
- the maximum allowed error on Vmag in the range [0.0, 0.1] with a default of 0.03, changed with **-emag=**.

The list can optionally be saved as a CSV file compatible with the NINA comparison stars list, specifying the file name with **-out=**. If the provided name is the special value **auto**, it is generated using the input parameters

See also LIGHT_CURVE

Links: *light_curve*

- `-narrow` limits the field of search to a circle inscribed and centred in your image. This avoids choosing stars in a potentially distorted part of the image.
- Without the `-narrow` option, comparison stars are searched across the entire picture. It can be useful in case of a poorly populated field of view.
- `[-catalog={nomad|apass}]` allows you to choose between the NOMAD catalogue and the APASS catalogue (by default).

**Tip**

The displayed names will be reduced to simple numbers, sorted by increasing distance wrt. image center These labels are only for information.

- `[-dvmag=3]`, `[-dbv=0.5]` and `[-max_emag=0.03]` stand respectively for **delta_Vmag**, **delta_BV** and **max_emag** criteria as explained previously.
- `[-out=nina_file.csv]` sets the name of the CSV file.

**Tip**

**APASS** stands for AAVSO Photometric All Sky Survey. Additional information can be found here.

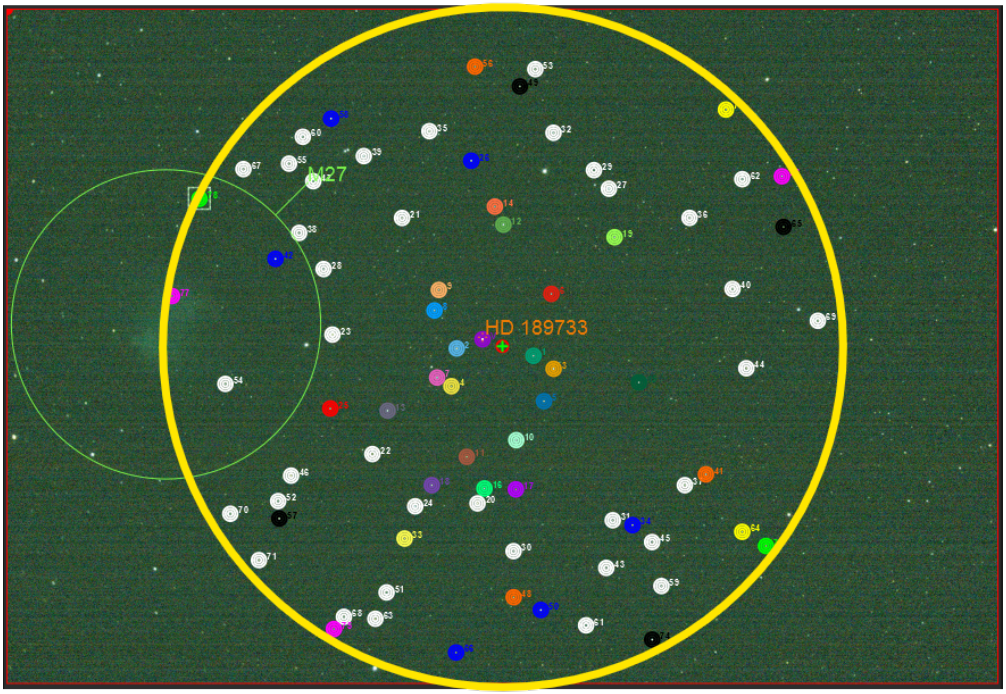



Fig. 21: View of the `-narrow` limited field of search. (the yellow circle is for illustration purposes only)

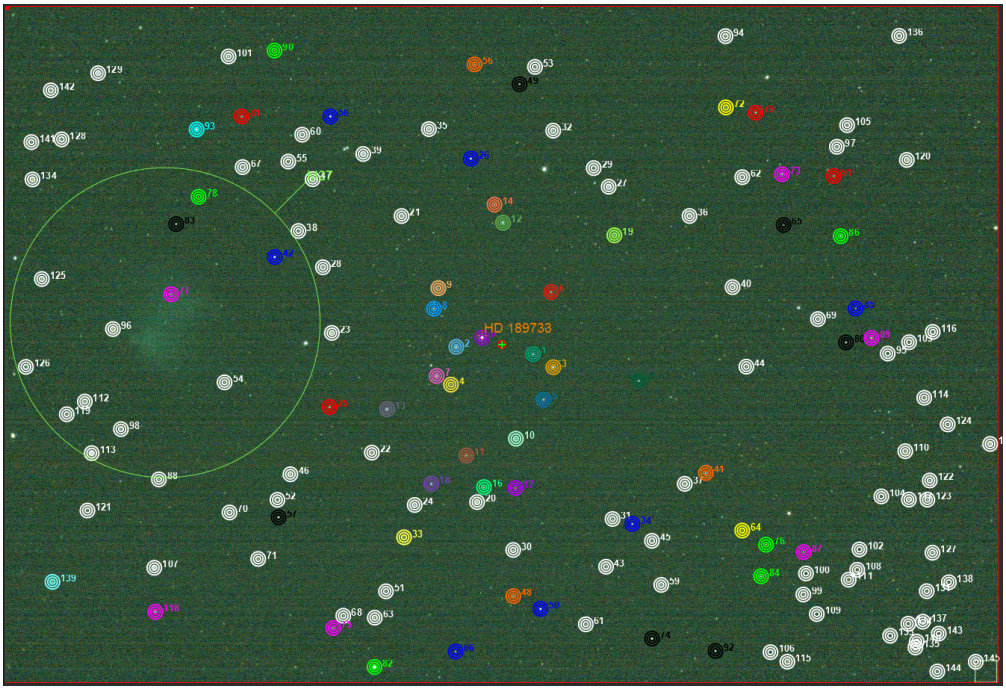



Fig. 22: View of the wider field of search.

The AAVSO Photometric All-Sky Survey (APASS DR9) provides calibrated magnitudes in the range 7.0 < Vmag < 17.0 for the entire sky, in the BVugriZsY bandpasses.

The photometry is accurate to about 0.02 mag, and the astrometry is accurate to about 0.15 arcsec (see Using APASS and 2GSS for studying variable stars).

### Discarding potential variable stars

Among the returned stars from either APASS or NOMAD, a few of them can be identified as variable stars. If one (or more) of these stars are selected as comparison stars, their flux variation can affect the final result for the variable star under study.

So the automatic comparison star list feature of Siril is able to find these outlier stars and discard them accordingly.

Three main catalogues can be used to detect variable stars:

- The General Catalogue of Variable Stars (GCVS). Provides more than 89000 variable stars.
- The AAVSO VSX, it lists all stars that have an entry in the AAVSO International Variable Star Index.
- The GAIA varisum GAIA varisum, a sub table of GAIA DR3 listing "potential variable objects".

These 3 catalogues can be used one by one or all at the same time during the discarding process. But you have to be aware that each selected catalogue results in an additional online request.

Basically, only the GAIA Varisum is selected. However, the user can still make his own choice.

The used catalogues are managed with a Siril variable, `photometry.discard_var_catalogues` according to the following table:

Table 1: Discard table to properly set `photometry.discard_var_catalogues`.

| Variable value | Binary | GCVS | AAVSO VSX | GAIA Varisum |
|---|---|---|---|---|
| 0 | 0000 | | | |
| 1 | 0001 | ✓ | | |
| 2 | 0010 | | ✓ | |
| 3 | 0011 | ✓ | ✓ | |
| 4 | 0100 | | | ✓ |
| 5 | 0101 | ✓ | | ✓ |
| 6 | 0110 | | ✓ | ✓ |
| 7 | 0111 | ✓ | ✓ | ✓ |

(It can be seen in the previous table that the variable can be written in binary format, with `bit#0` as GCVS, `bit#1` as VSX and `bit#2` as GAIA-Varisum).

One can read this variable with the command:

```
get photometry.discard_var_catalogues
```

One can assign another value to the variable with the command:

```
set photometry.discard_var_catalogues=4
```

As an example, with the command:

```
set photometry.discard_var_catalogues=7
```

You will have the 3 catalogues checked and the consol will display:

```
Contacting server
-> 5 variable stars found within the image from AAVSO Variable stars
Contacting server
-> 1 variable stars found within the image from GCVS
Contacting server
-> 8 variable stars found within the image from Gaia DR3 Variability
```

### Output file structure

While using the `findcompstars` command and setting `-out=auto` or the GUI *Comparison stars* button, one will get a file name with the following structure:

`[star_name]_SirilstarList_[delta_Vmag]_[delta_BV]_[max_emag]_[catalogue].csv`

Additional information is also supplied (as comments) in the file header:

```
# Sorted comparison stars for KELT-16 from APASS according to the following criteria
# Siril: 12 stars, dVmag 1.00, dBV 0.30, max e_mag 0.03
type,name,ra,dec,mag
Target,KELT-16,314.268494,31.661009,11.72
Comp1,1,314.256578,31.660406,12.307
Comp1,2,314.317434,31.754368,12.431
Comp1,3,314.305316,31.546814,11.682
Comp1,4,314.263680,31.541927,12.084
Comp1,5,314.465451,31.626255,12.084
Comp1,6,314.459388,31.596876,11.979
Comp1,7,314.452316,31.746360,11.504
Comp1,8,314.437181,31.827260,11.24
Comp1,9,314.314246,31.879192,12.313
Comp1,10,314.431752,31.839763,10.88
Comp1,11,314.006569,31.666214,11.896
Comp1,12,314.304606,31.925116,11.713
```

CHAPTER

SEVENTEEN

# PLOTTING FEATURE

Siril has a tab for displaying graphs of the data calculated during alignment or other calculations. This tab is very powerful and allows for easy sorting of the images as well as in-depth analysis of them. The shortcut to access this tab is `F5`.

## 17.1 Registration Plot

In order to improve manual sorting of your registered frames, plotting capabilities have been added in the Plot tab. After completing Registration of your sequence (or when loading a registered sequence), a dropdown list now enables to select parameters of interest for both plotting and sorting your data.

You can also choose to plot one of the parameter *vs* another. Items available in the dropdown list are:

- **FWHM**: This is the maximum width at half maximum, one of the most common criteria to judge the quality of a deep sky image.
- **Roundness**: The roundness $r$ is computed as the ratio of $\frac{\text{FWHMy}}{\text{FWHMx}}$.
- **wFWHM**: This is an improvement of a simple FWHM. The FWHM is weighted by the number of stars in the image. For the same FWHM measurement, an image with more stars will have a better wFWHM than an image with fewer stars. It allows to exclude much more spurious images by using the number of stars detected compared to the reference image.
- **Background**: Average sky background value.
- **# Stars**: This is the number of stars used for registration.
- **X Position**: X-shift with regards to the reference.
- **Y Position**: Y-shift with regards to the reference.
- **Quality**: This criterion is a number in the range [0, 1] that defines the quality of images that have been processed by any planetary registration algorithm.

Click on one of the data point to exclude the frame or to open it. The later option will load the picture and pop out the frame selector. The parameter chosen for Y values is reflected in the frame selector last column, which can then be used to sort, review and select or unselect subs from the sequence.

You can also mass select/unselect mutliple data points by drawing a selection onto the plot. The information at the top of the selection tells the number of points selected, as well as the boundary values of your selection. You can reshape the selection just as you would do with a drawn selection in the Image view. Once happy with your selection, a right-click will display a menu to keep or exclude the points, or to set the zoom to the selection.

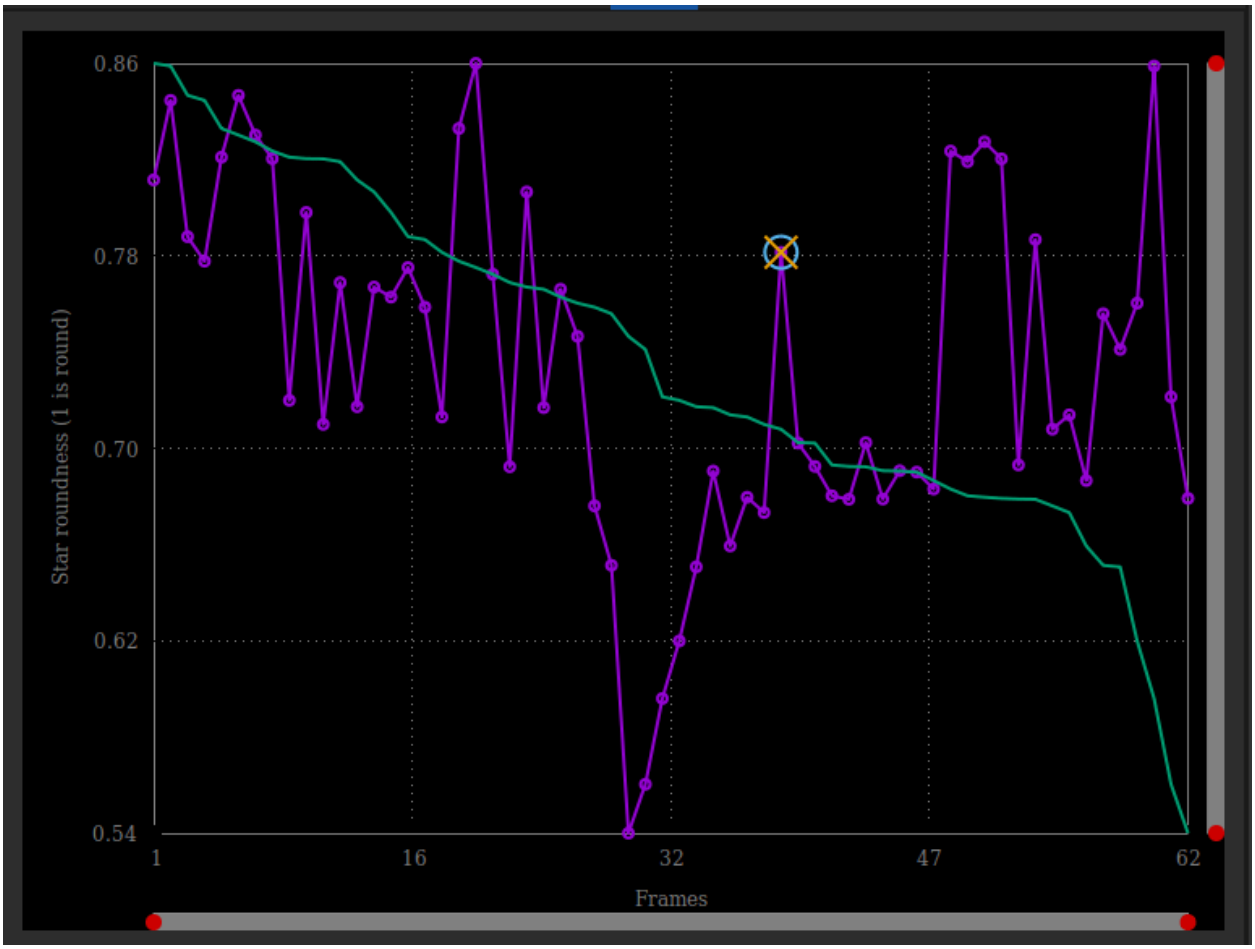



Fig. 1: Plot tab as shown after a global registration.

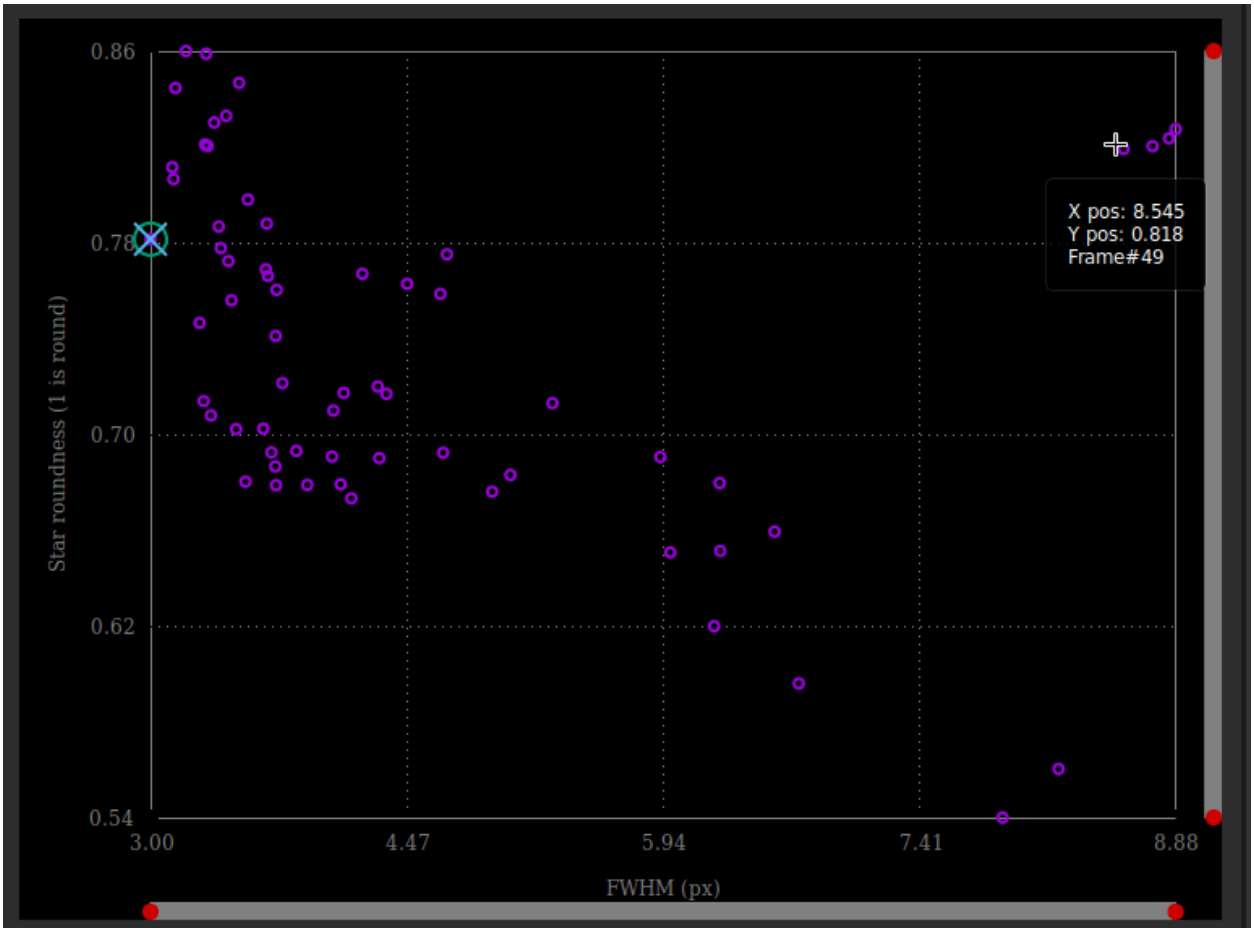



Fig. 2: The values of roundness vs FWHM are displayed as a scatter plot. Hover onto the different data points to show X and Y values, together with the corresponding frame number.

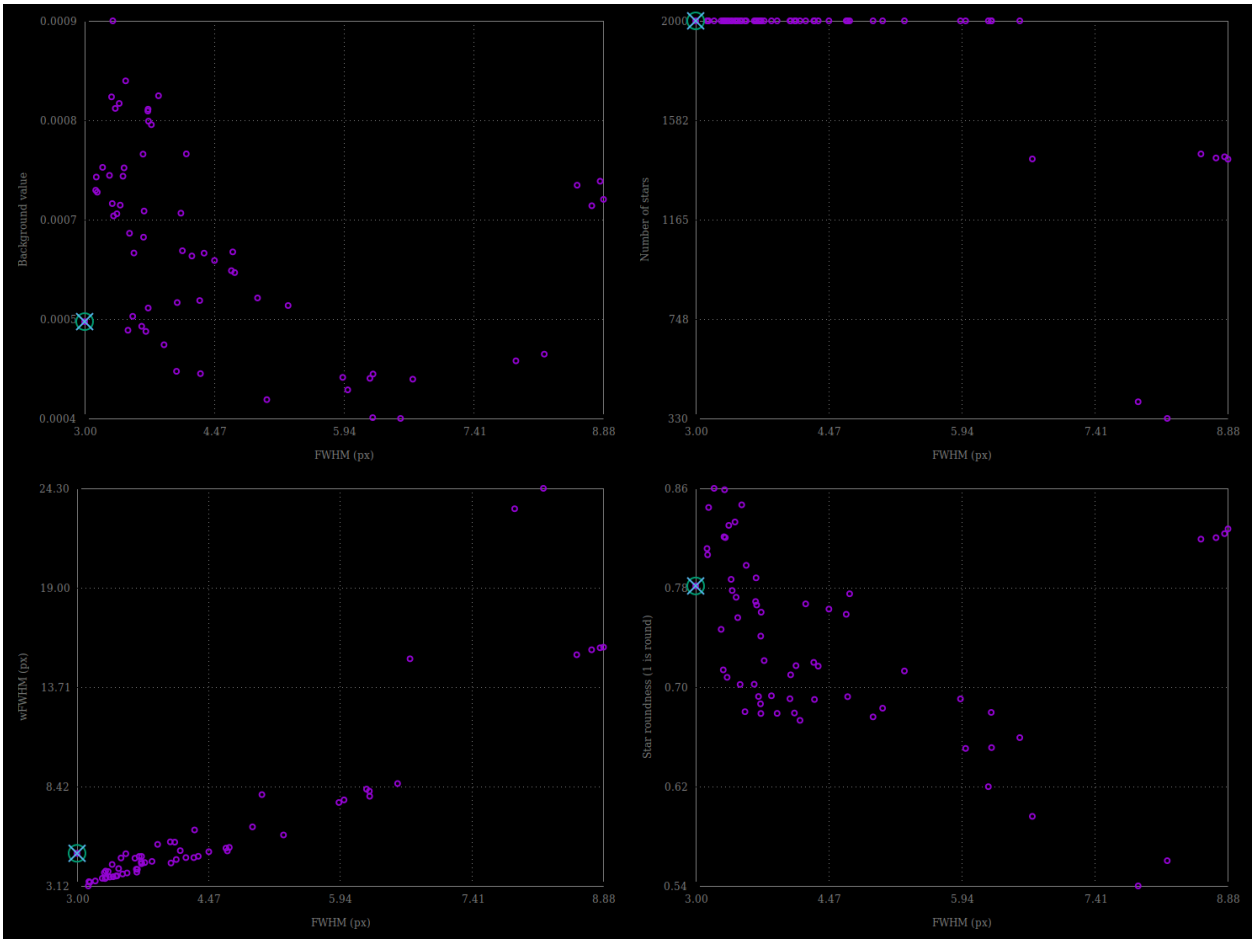



Fig. 3: Different possibility of graphics taken with the same set of images.

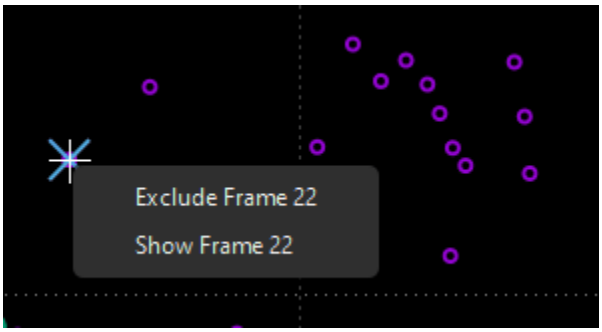



Fig. 4: Right-click on a data point to exclude or to load it in the Image preview

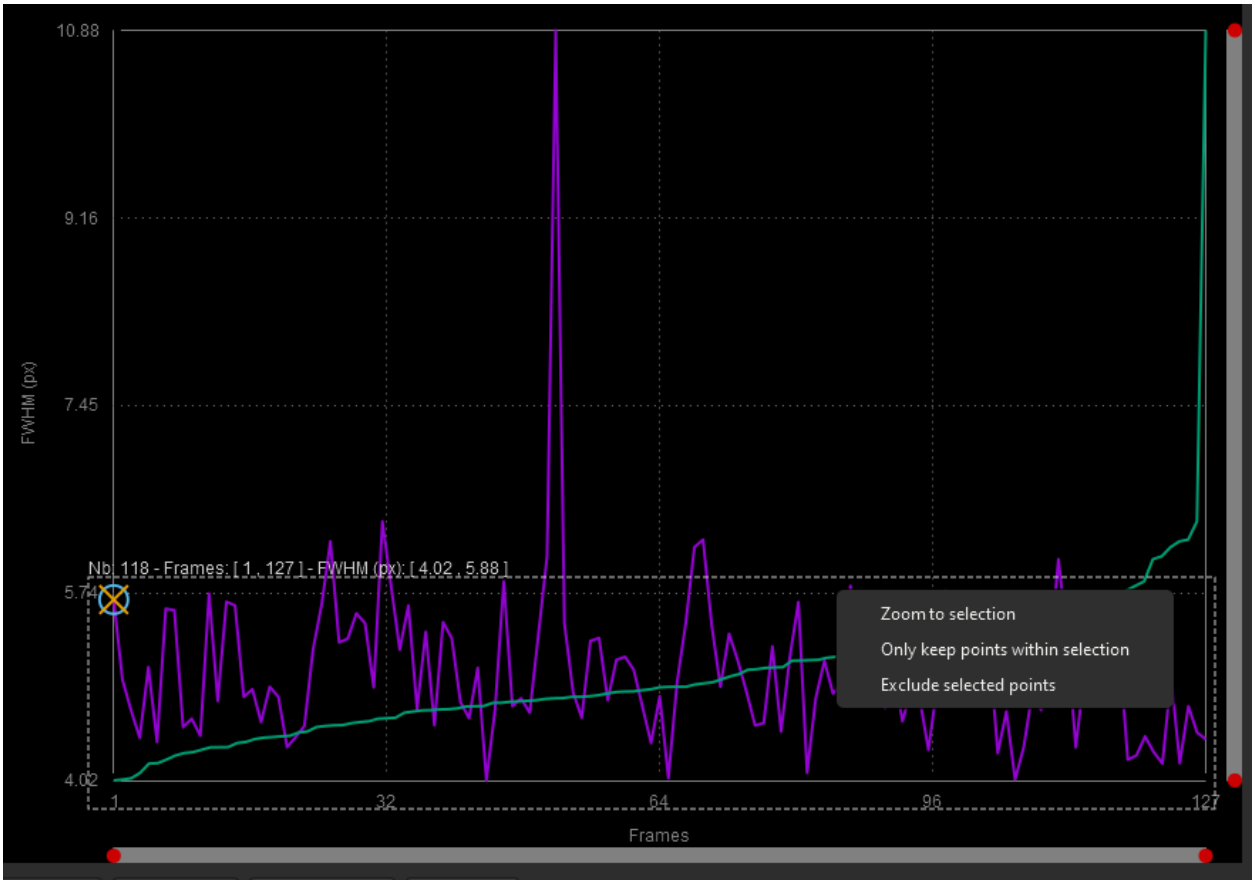



Fig. 5: Right-click on a drawn selection to mass select/unselect or to zoom

## 17.2 Photometry Plot

To complement sorting and filtering frames from the sequence, you can also perform a PSF on a star for the full sequence. The procedure is detailed in the *photometry* page. Then the photometry item in the first dropdown list becomes sensitive and is automatically selected. The other dropdown list contain the following items:

- **FWHM**: Maximum width at half maximum, as defined above.
- **Roundness**: The roundness $r$ is computed as the ratio of $\frac{\text{FWHMy}}{\text{FWHMx}}$.
- **Amplitude**: It is the maximum value of the fitted function, located at the centroid coordinates.
- **Magnitude**: Relative magnitude of the analyzed star.
- **Background**: Average of the local sky background value taken in the annulus.
- **X Position**: X-shift with regards to the reference.
- **Y Position**: Y-shift with regards to the reference.
- **SNR**: An estimator of the signal-to-noise ratio.

In photometry, unlike registration, it is not possible to change the X axis. And only the number of images can be used (or the Julian day).

- The *Clear plots* button combines the *Clear all* and *Clear latest* menus to clear either all photometric curves, or the last one drawn.

### 17.2.1 Tools menu

As of Siril 1.4, many options have been moved to the *Tools* menu, located to the right of the *Image processing* menu. This menu will be detailled in the section *Photometry* .

## 17.3 Shared options

Several output files can be produced by clicking on the *Output* button:

- *Show valid data in Siril plot* is an option that displays everything in the plot tab in a new instance of Siril plot.
- The *Export to CSV* button exports the displayed plot to a CSV file.

The *Display* button is used to change the axis units. It contains, depending on what is available, several buttons in a drop-down menu:

- If the sampling of the images is known, then the *arcsec* toggle button displays the FWHM in arcsec instead of pixel.
- In the case of photometry only (so not really a shared options), it is also possible to choose *Julian date* as the unit of the x-axis.

**Tip**

Hover the text "Hovering for the legend" will show the legend of the plot.

- Purple curve: actual plot as configured with X and Y selectors.
- Green curve: sorted values by order of decreasing quality.
- Circle marker: the value for the reference frame.
- Cross marker: the value for the currently loaded frame.

## 17.4 Plot interactions

Here is a summary of the possible interactions with the plot window:

- **Left-click** in a slider : puts the nearest red dot on it
- **Double-click** in a slider : resets this axis
- **Right-click + drag** in a slider : move the zoom on this axis
- **Left-click + drag** in the plot area : draws a selection
- **Left-click + drag** on a selection edge : resize the selection
- **Double-click** in the plot area : reset the zoom on the 2 axes
- **Right-click** when a selection is active : display the menu to : zoom on the selection/ keep only the points of the selection/ exclude the points of the selection
- **Left-click** when a selection is active : delete the selection

CHAPTER

EIGHTEEN

# STATISTICS

This is a documentation for Siril's statistics, given by the graphical user interface (GUI) from the menu *Tools* → *Image Analysis*) then selecting *Statistics...*, or using the stat command. Note that when using the GUI, it is possible to draw a selection in the loaded image and that when doing so, the statistics are computed on the pixels of region.

The option *Per CFA channel* allows you to calculate statistics for each R, G and B channel in CFA images, even if the image has not been demosaiced.

> **Tip**
>
> When a CFA image is loaded, the selection will be constrained to a minimum size of 2x2 for Bayer images and 3x3 for X-Trans images so that there is at least one pixel in each Bayer channel within the selection, to ensure that it is possible to generate statistics.

Many of these values are measures of statistical dispersion.

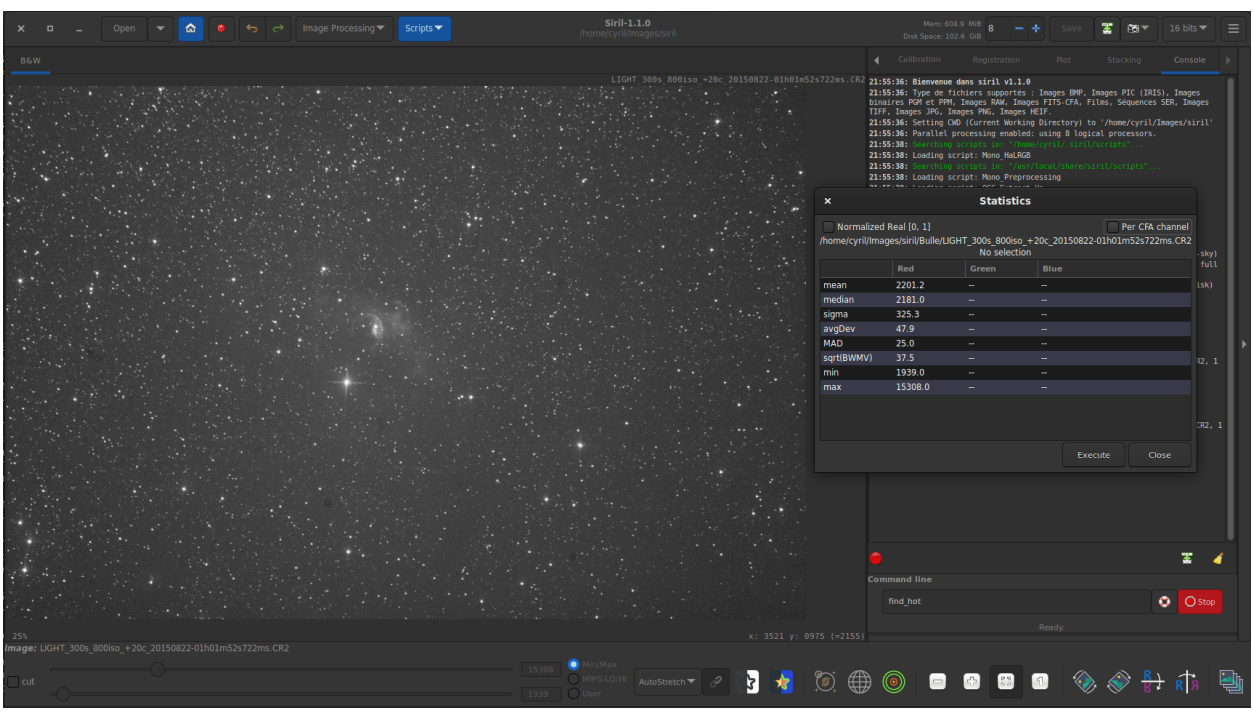



Fig. 1: One channel statistics for CFA image. The given values are not really relevant in this case.

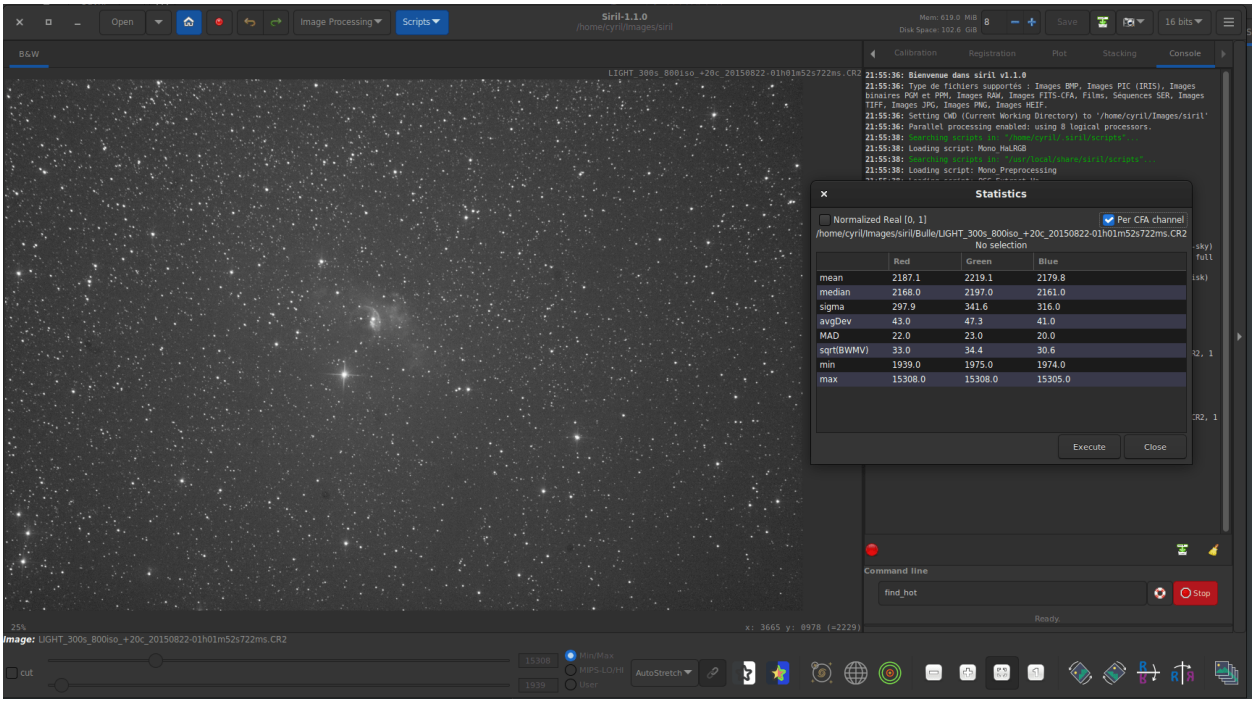



Fig. 2: Three channel statistics for CFA image.

**Siril command line**

```
stat [-cfa] [main]
```

Returns statistics of the current image, the basic list by default or the main list if **main** is passed. If a selection is made, statistics are computed within the selection. If **-cfa** is passed and the image is CFA, statistics are made on per-filter extractions

## 18.1 Estimators

### 18.1.1 Mean

This is the arithmetic mean, also known as average or arithmetic average. This is computed by doing the sum of the pixel values divided by the number of pixels in an image channel.

### 18.1.2 Median

The median is the value separating the higher half from the lower half of a dataset. Generally, it represents the value of the background of an astronomical image.

### 18.1.3 Sigma

Also known as the standard deviation, noted $\sigma$, this is a measure of dispersion of the image pixels based on squared differences from the average. The sigma value of a sub image containing only the background will represent the noise of the image.

### 18.1.4 Background noise

This estimator is available by the GUI from the menu *Tools → Image Analysis → Noise estimation*, and is also displayed at the end of stacking.

This is a measure of estimated noise in image background level, for pixels having a value low enough to be considered as background. It is an iterative process based on k.sigma (a factor of the standard deviation above the median), so there is no fixed threshold for the low enough.

**Siril command line**

```
bgnoise
```

Returns the background noise level of the loaded image

### 18.1.5 avgDev

The Average Deviation, also called AAD for average absolute deviation or mean absolute deviation. In order to understand what the average deviation is, one needs to understand what the term absolute deviation is. Absolute deviation is the distance between each value in the dataset and that dataset's mean (in this instance) or median (for MAD below). Taking all of these absolute deviations, finding the average, and the mean average deviation is computed. To simplify, if standard deviation is the squared deviation from the mean, this is the linear version of it.

### 18.1.6 MAD

The Median Absolute Deviation is a robust measure of how spread out a set of data is. The absolute deviation and standard deviation are also measures of dispersion, but they are more affected by extremely high or extremely low values. It is similar to the average deviation above, but relative to the median instead of the mean.

### 18.1.7 BWMV

The biweight midvariance is yet another tool to measure dispersion of a dataset, even more robust than others cited above to outliers. It discards the data points too far way from the median and computes a weighted variance, weights decreasing as the data points are further way from the median. The estimator of dispersion is the square root (marked as $\sqrt{BWMV}$) of this value.

### 18.1.8 Location and Scale

These parameters, often colloquially called scale and offset, are not displayed in the user interfaces but are computed internally by Siril. In order to align the histograms of the different images for normalization before stacking, one needs to compute where they are in terms of level and how wide they are in terms of spread. A valid estimator of location could be taken as the median while the MAD or the $\sqrt{BWMV}$ could be used for scale. However, in order to give more robustness to the measures, the pixels more than $6 \times$ MAD away from the median are discarded. On this clipped dataset, the median and $\sqrt{BWMV}$ are re-computed and used as location and scale estimators respectively. They are computed relative to the reference image of a sequence in Siril.

CHAPTER

NINETEEN

# SIRIL FOR SCIENTISTS

For years Siril has been a great tool for amateur astronomers. With its capability to produce light curves, query online catalogues, run operations on image sequences, and many other recent additions, it has also become a great tool for scientific analysis. This page is a quick reference for scientists who want to learn quickly about Siril, in form of a list of Q&A.

## 19.1 How can Siril help me with my research?

Siril's default file format is FITS files, containing either a single image or several (see the FITS cubes question below). Siril has always been developed with data accuracy in mind. Displaying an image can be done in various ways without destroying input data.

Siril has many tools that can help scientific observations: star detection, elaborate plate solving, registration of an image sequence using stars or using astrometric solutions, aperture photometry with magnitude calibration using catalogues, automated light curve creation, object lookup from online catalogues (stars, galaxies, solar system objects). Many of those operations can be done from the GUI or from text inputs (commands or scripts), some operations produce comma-separated values (CSV) files for results, some can produce graphs directly in Siril.

## 19.2 Which version of Siril should I install?

Siril follows an even/odd versioning system, e.g. 1.2.x was a stable series whereas 1.3.x was a development series. For scientific use it is recommended to install the latest version of the latest stable series: download links can be found at siril.org. Beta versions, those distributed on the website, can also be considered quite stable for general use.

## 19.3 How can I get in touch with the developers?

The official forum is on pixls.us. If you want to contact us privately you can send us a direct message there or on gitlab. Our email addresses can also be found in the AUTHORS file. If you speak French you can find us on Astrosurf or Webastro. Feel free to reach out if you have specific needs for your research.

## 19.4 How accurate are Siril results?

Many algorithms in Siril rely on peer-reviewed published algorithms which can be found in the documentation page of the features. We are testing Siril often, sometimes comparing results with other tools, but we don't have the resources to have a automated tests on all pieces of the software, so data comes with no warranty. We know some things could be improved, for example error computation on magnitudes, which depend on camera gain, not always available in the FITS header.

Astrometric data is as good as any and supports the WCS FITS convention.

## 19.5 How can I cite Siril?

If you use Siril in your research, please cite:

> Richard, C., et al. (2024). Siril: An Advanced Tool for Astronomical Image Processing. *Journal of Open Source Software*, 9(102), 7242. https://doi.org/10.21105/joss.07242

**BibTeX format:**

```
@article{Richard2024,
  title={Siril: An Advanced Tool for Astronomical Image Processing},
  author={Richard, Cyril and others},
  journal={Journal of Open Source Software},
  volume={9},
  number={102},
  pages={7242},
  year={2024},
  publisher={The Open Journal},
  doi={10.21105/joss.07242}
}
```

## 19.6 How can I open FITS cubes with Siril?

FITS cubes come in two shapes for Siril:

- those that contain several images of the same size and format, we call those FITS sequences or fitseq for short, and use them as an alternate file representation of image sequences;
- those that contain several images of changing properties, or even tables and other data formats. Siril will only read image data, but is able to work on each image of such FITS cubes if an option is enabled in settings (named *Allow FITS cubes to have images of different sizes* in *FITS Options*). Most sequence operations will not be available on this kind of file.

Siril can also extract images from FITS cubes to individual FITS files or to other formats if needed.

## 19.7 How do I measure FWHM in an image?

Siril can give you the FWHM of a single star if you draw a selection around it in the GUI, then right click and select PSF, or for the stars in an area of the image or the full image. You can also use the *quick PSF* button in the main toolbar.

More generally: open your image and open the *Dynamic PSF tool* (from the *Tools* menu). From there clicking on the first button will start the star detection, within the selected area if you drew one. Clicking on the Sigma icon will give mean values for all detected stars. See also the next question.

## 19.8 How do I use star detection (source extraction)?

Siril's source extractor can be found in the GUI in the *Dynamic PSF tool*, it is also available as the *findstar* command. It is normally only able to detect stars and will struggle if two stars are too close from each other, or if stars have asymmetrical or very non-Gaussian shapes, but there are many settings that you could adjust to your needs, including brightness, amplitude, roundness thresholds or star shape (Gaussian or Moffat).

The GUI will display the found stars by clicking on the button on the left of the lower bar, giving quick feedback on the detection settings. The GUI and the command can produce a CSV file that can then be used to check extracted sources in an image or in a sequence of images (command only).

The list of stars can be sorted by property, clicking on a row will highlight the star in the image.

## 19.9 What astrometry tools are available?

Siril has now a very high quality astrometric solver (plate solver) and tools that interact with the astrometric solution. All is based on the FITS WCS convention and on WCSLIB. Here is a list of tools related to astrometry:

- Plate solving with local Gaia DR3 catalogue extract or with remote full Gaia DR3, PPMXL or APASS.
- Display stars from the catalogue on image, to have a visual feedback of the accuracy of the astrometric solution, using the *conesearch* command.
- *Query* online catalogues for an object using its name and the image time.
- Get centroid equatorial J2000 coordinates for any detected object.
- Show a mark on image for user provided J2000 coordinates (*show* command).
- Align images on the celestial grid, possibly creating a mosaic (an image bigger than the original).
- Correct for geometrical aberration of images, using a master correction image based on astrometry.

## 19.10 Can it handle solar system objects?

Siril can query IMCCE services to find a solar system object coordinates and display the expected position on images. It can also display the known solar system objects in an image. Observer location will be required for those uses and can be configured with MPC codes in the settings. See the *annotations* documentation page.

Soon, Siril will be able to do synthetic tracking of known solar system objects, making them appear on stacking if they are too faint to be seen on single exposures.

But Siril is not made to process closeup view of planets with multi-point stacking like other tools such as PlanetarySystemStacker (PSS) does.

## 19.11 Can Siril work with spectra?

Siril was not designed to handle spectrometry data. An *Intensity Profiling* tool that shows the graph of pixel intensities can be used for preview of a spectrum. FITS tables are not displayed in any way.

## 19.12 Can Siril replace IRAF?

Some features will be missing, many will be much more easy to do with Siril. We don't yet have a list.

Siril has been developed since 2012 and we don't plan to stop anytime soon. There are 3 active developers and thousands of amateur users.

CHAPTER

TWENTY

# SCRIPTING

This section explains the different scripting and automation methods available in Siril.

> **Tip**
>
> Python scripting was introduced in the 1.3.5 development version. It is currently marked as **EXPERIMENTAL**. This doesn't mean it will eat your data: the interface itself is robust and has been tested through development: the present experimental nature of it is more to do with the fact that we don't yet know what users will do with this new capability and whether there may be issues or limitations that we have not foreseen, perhaps due to the constraints of packaging or consistency across the different operating systems.
>
> Please try it out, either as a user by using scripts published on the scripts repository, or as a script writer. We welcome all your feedback and will aim to refine the interface throughout the 1.4 stable series and 1.5 development series.
>
> If you want to debug your Python scripts at runtime, tick the box next to *Enable Python debug mode*. A tutorial detailing the steps to attach to the Python process is shown in Siril tutorial page.

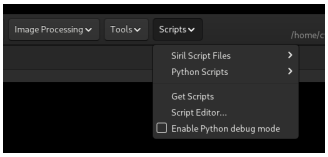



Fig. 1: Scripts menu

## 20.1 Siril Script Files

Siril Script Files are the original format for scripting Siril and remain a simple and useful way of automating a fixed set of tasks.

### 20.1.1 Siril Scripts

Siril scripts are a list of commands, either from the graphical user interface or from the command line interface. In general, commands that modify a single image work on the currently loaded image, so the use of the *load* command is required in scripts, and commands that work on a sequence of images take the name of the sequence as argument. If files are not named in a way that Siril detects as a sequence, the command *convert* will help.

> **Tip**
>
> The `Space` character is the delimiter between arguments. If you need to have spaces inside the arguments you can use the quote or double quote, just like in a shell.

Commands can be typed in the command line at the bottom of Siril's main window. Another way is to put commands in a file and execute it as a script. To execute the file from the GUI, add it to the configured script directories or from the GUI, use the `@` token of the command line like so:

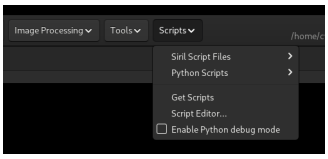
```
@file_name
```

Some commands (*calibrate*, *stack*, and all save commands) can use file names containing variables coming from the FITS header. The format of the expression is explained in details *here* and can be tested using the *parse* command.

### 20.1.2 Using scripts

There are four ways to run a script:

- from the graphical user interface, using the `@` keyword on the command line, followed by the script name in the current working directory,
- from the graphical user interface, using the *Scripts* menu,




- from the graphical user interface, using the *Script Editor* dialog which can be used to write, save and execute your own scripts or to open and edit an existing script file,
- from the command line interface (**`siril-cli`** executable), using argument `-s` followed by the script's path (see the man page for more info).

When a script is running, the user interface becomes non-interactive with the exception of the Stop button.

### 20.1.3 Populating the list of scripts

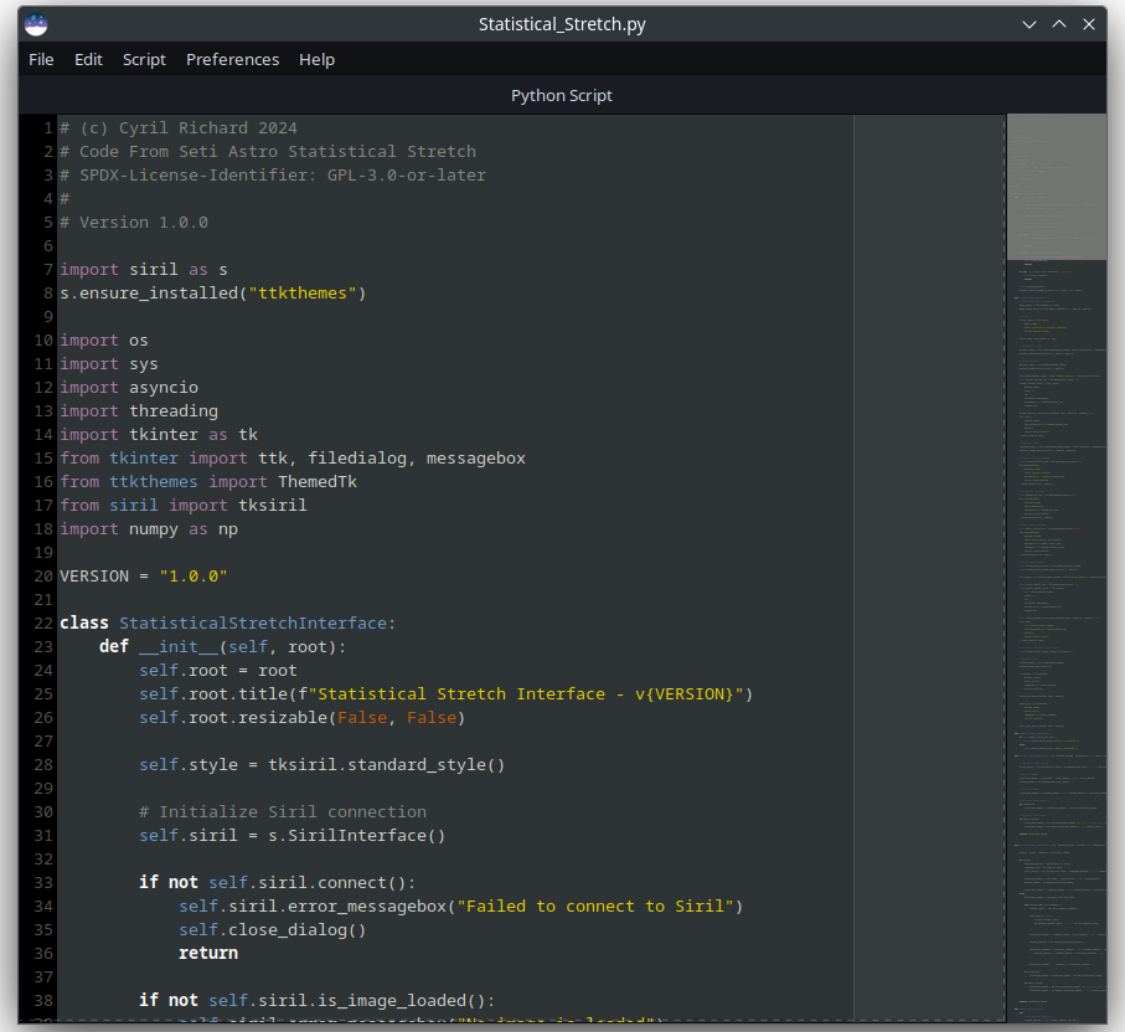
By default, when Siril is installed, a number of scripts are automatically installed. These built-in scripts, the official ones, are developed by the development team and are guaranteed to work: they are meant to cover specific use cases.

#### Adding custom scripts folders

You can, of course, *write your own* and tell Siril where to find them:

- Click on the *Burger* icon then on *Preferences* (or hit `Ctrl+P`).
- Click on the *Scripts* section.
- Copy to a new line the path to the location to store them (create a folder on your computer as required or point to an existing one).
- Click on the *Refresh* icon just below.
- Click on *Apply*.

You can have as many user-defined folders as you wish, just add them to the list.




```python
# (c) Cyril Richard 2024
# Code From Seti Astro Statistical Stretch
# SPDX-License-Identifier: GPL-3.0-or-later
#
# Version 1.0.0

import siril as s
s.ensure_installed("ttkthemes")

import os
import sys
import asyncio
import threading
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from ttkthemes import ThemedTk
from siril import tksiril
import numpy as np

VERSION = "1.0.0"

class StatisticalStretchInterface:
    def __init__(self, root):
        self.root = root
        self.root.title(f"Statistical Stretch Interface - v{VERSION}")
        self.root.resizable(False, False)

        self.style = tksiril.standard_style()

        # Initialize Siril connection
        self.siril = s.SirilInterface()

        if not self.siril.connect():
            self.siril.error_messagebox("Failed to connect to Siril")
            self.close_dialog()
            return

        if not self.siril.is_image_loaded():
```

If you have just added a new script in one of the folders and wish to refresh the menu, type the command *reloadscripts* in the command line or open the *Preferences* → *Scripts* section and use the *Refresh* icon. This scans all the folders of the list and find all the files with the *.ssf extension.

> **Warning**
>
> It is strongly advised **not** to store your custom scripts within the same folder as Siril built-in scripts. On Windows, they may get wiped when installing a newer version or prevent correct uninstall. On MacOS, it will break the bundle and prevent using Siril altogether.
>
> Don't worry, as the list of scripts locations is stored in your configuration file, you should find them back when installing a newer version.

#### Adding Scripts from the git Repository

Siril supports a git repository at https://gitlab.com/free-astro/siril-scripts This is set to auto update at startup by default, so you will always have access to the latest scripts (auto update can be disabled in preferences). To add scripts from the repository to the Scripts menu, pick the ones you want from the list available in *Preferences* → *Scripts* or via the *Scripts -> Get Scripts* menu entry. See below for full details of the git repository.

#### Run Script Files

Script files can also be run directly from the hard disk using the *Run Script Files...* script menu entry. The filechooser shown by this menu defaults to showing recently used scripts, but you can navigate to choose scripts from anywhere accessible on the filesystem.

#### Troubleshooting

For different reasons, it is possible that the *Scripts* menu is empty. This means that the scripts have not been found. If this is the case, please use the following procedure.

- Click on the *Burger* icon then on *Preferences*.
- Click on the *Scripts* section.
- Delete all the lines in the field *Script Storage Directories* as shown in the illustration below.
- If you are using the scripts repository, deselect the *Fetch and update scripts...* check box, then reselect it and select the scripts you want to use.
- Click on *Apply*.
- Close and restart Siril.

### 20.1.4 Built-in scripts

All built-in scripts must follow this file structure:

- **Mono_Preprocessing.ssf**: script for monochrome DSLR or Astro camera preprocessing, uses biases, flats and darks, registers and stacks the images. To use it: put your files (RAW or FITs) in the folders named `lights`, `darks`, `flats` and `biases` (in the Siril default working folder), then run the script.
- **OSC_Preprocessing.ssf**: same script as above but for One-Shot Color (OSC) DSLR or Astro camera. To use it: put your files (RAW or FITS) in the folders named `lights`, `darks`, `flats` and `biases` (in the Siril default working folder), then run the script.
- **OSC_Preprocessing_BayerDrizzle.ssf**: same script as above but using Bayer Drizzle to retrieve colors. To use it: put your files (RAW or FITS) in the folders named `lights`, `darks`, `flats` and `biases` (in the Siril default

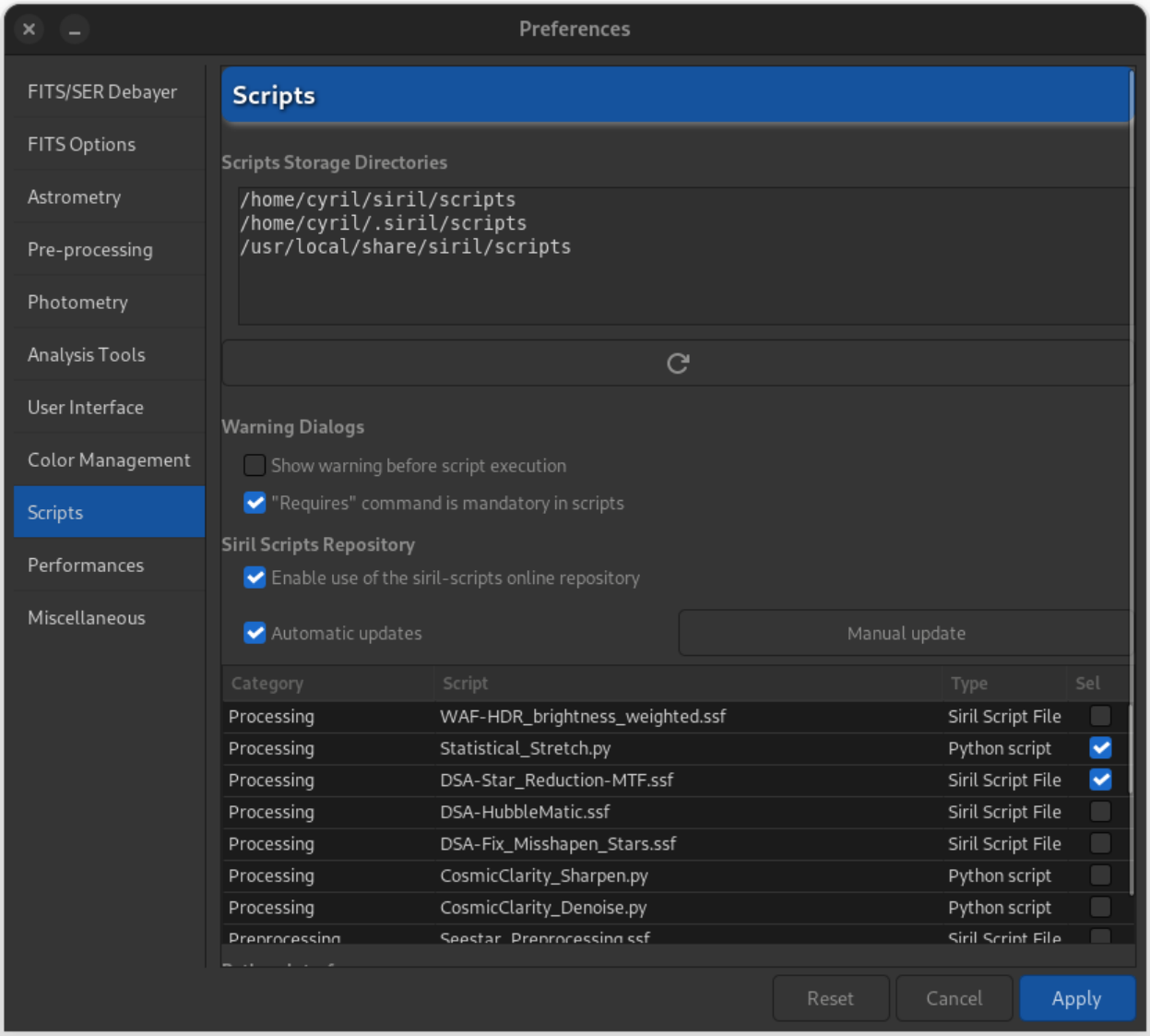
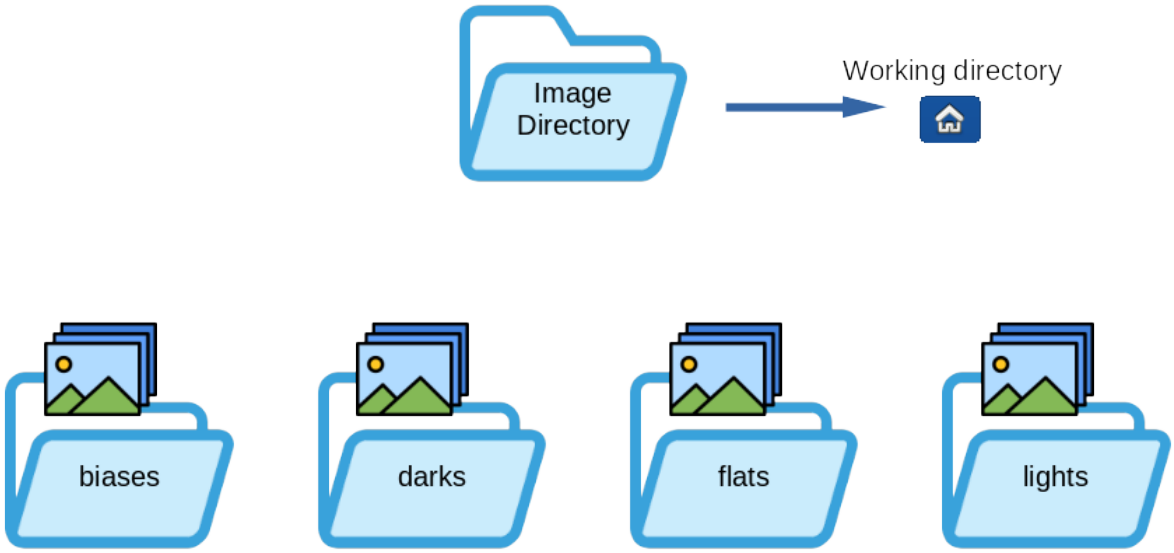



Fig. 2: Script page of preferences. The script are loaded from the paths listed in the *Script Storage Directories*.




working folder), then run the script. A large amount of data is strongly recommended to take advantage of the benefits of Bayer Drizzle and avoid unsightly artifacts.

- **OSC_Extract_Ha.ssf**: script for OSC DSLR or astro camera preprocessing, for use with Ha filter or dual-band filter. This script extracts the Ha layer of the color image. To use it: put your files (RAW or FITs) in the folders named `lights`, `darks`, `flats` and `biases` (in the Siril default working folder), then run the script.
- **OSC_Extract_HaOIII.ssf**: same script as above, but extracts Ha and OIII layers of the color image. To use it: put your files (RAW or FITs) in the folders named `lights`, `darks`, `flats` and `biases` (in the Siril default working folder), then run the script. You can also use the menu *Image Processing* then *RGB compositing* and put Ha result in Red channel and OIII result in Green and Blue layers to get an HOO image.

> **Tip**
>
> For owners of **SII** or **SII-OIII** dualband filters, the same scripts apply. In fact, it's impossible for a color sensor to see the difference between **Ha** (656.3 nm) and **SII** (671.6 nm), both of which are red.

- **RGB_Composition.ssf**: This script added in version 1.2 registers monochrome images with a global registration, reframes them to their common area, and takes the first three images to create a color image. The input images should be put alone in a directory and named `R.fit` (or with the configured extension), `G.fit` and `B.fit`. The result will be named `rgb.fit`. Make sure you remove the `process` directory between each run.

### 20.1.5 Editing Scripts

Some .ssf script files contain instructions to edit them, for example the Seestar_Preprocessing script instructs the user that if they find too many images are discarded before stacking, they should "increase the value after -filter-round= in the seqapplyreg command, line 47".

The scripts repository directory is not intended as a user-editable directory: at each update it is forcibly reset to match the state of the remote, so files saved in it are not safe. So how to make such changes?

Find the script in the list in *Preferences->Scripts* and double click it. This will open it in the Script Editor so that you can make the necessary changes and save it in a suitable location (one of the Scripts Storage Directories set at the top of *Preferences->Scripts* is ideal, but it is recommended to give it a different name so you can recognise it in your scripts menu).

> **Tip**
>
> You can also double click scripts in the list simply in order to examine the code.

#### Language of scripts

At the beginning of the scripts, and thanks to the contribution of a user, the scripts existed in two versions (English, and French). When Siril 1.2.0 was released, it was decided to keep only the English scripts for simplicity of maintenance. We encourage users to distribute translations of the official scripts to their respective communities if they deem it necessary.

### 20.1.6 Getting more scripts

There are a whole bunch of scripts that don't come with Siril installation. However, we've set up a gitlab repository for them. Everyone is free to register and propose new scripts. We'll accept them according to their relevance: the language used must be English.

Siril features git integration which means that it can download and synchronise a local copy of the repository. You can enable this by selecting the *Enable use of the siril-scripts online repository* check box in the Scripts tab of the Preferences dialog.

Selecting the check box will fetch scripts from the repository and show a list of the available scripts, categorised as either "Preprocessing" or "Processing" scripts. As there may eventually be a substantial number of scripts in the repository you need to select the ones you wish to have available in the *Scripts* menu. Click the check box next to the name of each script you wish to use and then press the *Apply* button.

The contents of any script can be viewed by double-clicking on its row in the list. It is always useful to do this to check what requirements the script may have in terms of pre-prepared directories. If you wish to modify a script, you can also use this to copy the script and paste it into your favourite text editor software. (You will need to save it in one of your local script directories.)

When the scripts repository is enabled, Siril can synchronise the local repository with the remote, either manually or automatically. If automatic updates are selected, updating will take place at application startup. Manual update is also available using the *Manual update* button. This will fetch any changes from the online repository and will show a list of the commit messages describing the changes, which the user must confirm to apply the update.

You can also refer to the address below to browse the scripts and download them manually if you don't wish to use the git integration. In that case you will need to manually place scripts you download into a script path known to Siril: https://gitlab.com/free-astro/siril-scripts.

> **Warning**
>
> Keep in mind, however, that these scripts are not necessarily maintained by the users who uploaded them, and may not be up to date. That said, have fun.

### 20.1.7 Writing your own

A script file is a simple text file with the extension *.ssf.

Writing a script is not difficult. It is a succession of calls to commands that will be executed sequentially. Each command must be executed without returning an error, otherwise the script stops. It is therefore strongly recommended to use the list of *commands* to know the syntax and the number of parameters needed. Also, some commands are not scriptable and are indicated with the icon. It can also be useful to test each script line in the Siril command line. You may wish to read the provided scripts or view (or even modify) scripts from the repository as examples.

Each script should contain a comment header containing information about the script. An example of this is provided below.

```
###########################################
#
# Script for Siril 1.0
# July 2020
# (C) Cyril Richard
# Mono_Preprocessing_WithoutDark v1.0
#
########### PREPROCESSING SCRIPT ###########
#
# Script for mono camera preprocessing
#
# Needs 3 sets of RAW images in the working
# directory, within 4 directories:
#   biases/
#   flats/
#   lights/
#
###########################################
```

Below the comment header the first command should be *requires*. This specifies the minimum version of Siril required to use the script. For example:

```
requires 0.99.4
```

After this you can start the actual script. Commands go on a line of their own, and you can comment your script using lines starting with #.

Each new script created in this way should be placed in a *user-defined folder* for Siril to find them. If you believe your script is of benefit to the wider Siril community you may submit it to the script repository. Instructions on doing so are found in the repository README.

## 20.2 Python Scripts

New to Siril 1.4, it is possible to write more advanced scripts using python.

This page is a documentation for script developers. If you want more information about what scripts are available as a user, see *the list of scripts*.

**Tip**

Python scripting is still **EXPERIMENTAL** and the API is expected to become more advanced during the 1.5 development cycle.

In its current form, python scripting allows you to:

- Execute any siril command with the added benefit of using a proper programming language so that variables passed to commands can be the result of calculations.
- Access key Siril variables and data structures to aid in scriptwriting.
- Access python extensions.
  - Using the 1.4 scripting interface it is possible to write a plugin that saves the current image, opens it using astropy and applies processing techniques provided by astropy, then saves the result and reopens it as the Siril main image.
  - The **tk** python extension can be used to create graphical user interfaces for your scripts so that user input can be obtained directly.

**Warning**

**Run-time Dependencies**

To support installation and configuration of the siril python module and its dependencies, a Python venv ("virtual environment") is configured.

**Linux**

This requires that there is a functional Python installation on the system _including the `pip` and `venv` modules. These are available by default as part of the flatpak and appimage distributions, but you will need to install them yourself if building Siril from source. On Debian-based systems you need the `python3-pip` and `python3-venv` packages installed; the installation requirements on other systems may vary.

**Windows**

If you have a Python installation in your system, the script will use that as default. If you don't, don't panic. Siril installed through the official installer also comes with a light-weight Python installation and all the required modules. That is for the official releases. If you build the development version from sources, you will need to have Python installed in your system.

**macOS**

The necessary dependencies are included in the official MacOS package. We do not provide support for homebrew building, but if you do build yourself on Mac then you will need to ensure you have a functional system python installation.

### 20.2.1 How does this relate to pySiril?

pySiril is a separate module, published by a separate development team: it is an established product and is already used to implement important helper apps such as Sirilic.

Philosophically, pySiril and the built-in Siril python module are quite different: whereas the built-in module is intended to be called from an already-running Siril to provide scripting functionality and spawns a python3 process to run the

script, pySiril is intended for buildin applications that need to call on Siril functionality, and it spawns a Siril process from within its host Python app.

These use cases are quite different, and there is no plan to merge the two modules.

### 20.2.2 Importing the Module

Siril will take care of setting up a venv (new style python virtual environment) and installing the module in it. In your script you import it using `import sirilpy`.

For convenience, you can abbreviate it as `import sirilpy as s`.

### 20.2.3 Initialising the Connection

Once you have imported the module, you need to establish the interface with Siril. This is done as follows:

```
import sirilpy as s

siril = s.SirilInterface()
try:
   siril.connect()
   print("Connected successfully!")
except SirilConnectionError as e:
   print(f"Connection failed: {e}")
```

(The status reporting is optional but is good practice.)

### 20.2.4 Executing Siril Commands

To execute a Siril command, once the interface is established, you use the `siril.cmd()` function. The command and its arguments are provided as a list, for example:

```
siril.cmd("findstar", "-maxstars=1000")
```

will run the `findstar` command, setting a limit of maximum 1000 stars.

The real power of using python to script commands is that you can use python variables in the command using the formatted string notation:

```
x = 1000
siril.cmd("findstar", f"-maxstars={x}")
```

This is a simple example, but the ability to define parameters to pass to image processing functions is a giant leap for Siril scripting compared with legacy Siril script files.

**Tip**

Most Siril commands execute an action (`register`, `stack`, `asinh` etc). These will function exactly the same when called from python.

However some simply print information to the Siril log, such as `stat`. Note that these commands do not behave any differently when called from python using `siril.cmd("stat")` or similar: they will still just print information to the Siril log and do not return any useful data to the python script.

In order to access the values from python you need to use a method that gets information about the image or sequence. So in this example, you would call `img = siril.get_image()` and then the statistics would be available in `img.stats`.

**Warning**

The `cmd()` method does not have a return value but will raise an exception if the command fails. This provides safe default behaviour - the script will stop on failure of a command and print an exception error message to the Siril console. If you want to handle failure more gracefully, for example by showing an error messagebox, you will need to use an exception handler like this:

```
try:
    siril.cmd("requires", "1.5.0")
except:
    siril.error_messagebox("This script is not compatible with this version of Siril")
    quit()
```

For debug purposes you may wish to have commands return a bool. This can be done using a simple wrapper such as the following:

```
def cmd_with_check(*args):
try:
    siril.cmd(*args)
except Exception as e:
    err_msg = ' '.join(list(args))
    print(f'Exception caught: {e}',)
    print(f"Command failed with arguments: {err_msg} but continuing...")
    return False
return True
```

### 20.2.5 Access to Siril variables and data structures

The siril module provides access to key Siril variables. The available data is as follows:

- Current Siril working directory
- Siril user config directory (for storing script-specific config)
- Current Siril image filename
- Current Siril image:
  - Image pixel data
  - Image ICC profile (as bytes: you will need to use a module such as pillow to convert this raw data to something useful)
  - Image metadata: all the metadata that Siril uses internally, including relevant keywords and image statistics. The full FITS header and a string containing keywords not recognized by Siril are also available.
- Current Siril sequence:
  - Sequence frame pixel data
  - Sequence metadata: all the metadata for the currently loaded sequence is available, with the exception of some that relates to details of the sequence format that are abstracted away by the python interface and some relating to photometry series that is not implemented yet. This includes statistics for every channel of every frame, registration data for channels for which it is available of every frame and image quality data for every frame.
  - Sequence frame ICC profiles are not available as these are not generally relevant until after the sequence processing stage. If a convincing use case emerges this could be revisited as an update to the API in the 1.5 development cycle.

  - Sequence frame HISTORY and FITS header strings are not available as they are not considered useful for sequence operations, however the keywords for a sequence frame are available using the `siril.get_seq_frame()` method, which returns a FFit object with the keywords member populated. This provides enough metadata to support any currently identified use case.

- Star modelling data. When stars have been detected in an image the modelling data is available as a list of star parameters for every detected star in the image.

This data is stored in the key data structures `FFit` representing the Siril fits data structure and `Sequence` representing the Siril sequence data structure.

```
# Get the current image
img = siril.get_image()
# Get its dimensions
siril.log(f"Current image dimensions: {img.shape[1]} x {img.shape[0]}, {img.shape[2]}␣
↪channels.")
```

Note that the image shape is stored in a typical python order: shape[0] = height, shape[1] = width, shape[2] = channels. The width, height and channels are also directly accessible using the properties `img.width`, `img.height` and `img.channels` once `img = get_image()` has been called.

When the current image is obtained using `siril.get_image()`, the pixel data can be accessed as a numpy array, allowing directly operating on pixel data.

```
import siril
import numpy as np
# Set up the interface as above
# ...

# Try to get the current image, do something to it and update it in Siril
with siril.image_lock():
   img = siril.get_image()
   img.data[:] *= 2
   siril.set_image_pixeldata(img.data)

except Exception as e:
   raise SirilException(f"Error changing pixel data: {e}")
```

We just accessed the pixel array and multiplied every pixel value by 2! Note that we have to call the `siril.set_image_pixeldata()` function to update the data back to Siril. Using `siril.set_image_pixeldata()` you can even update the image to a different size or shape, or change from 16-bit unsigned data to 32-bit floating point or vice versa.

**Tip**

In order for `siril.set_image_pixeldata()` to succeed you must claim a lock on the image. This ensures that nothing else can try to update the image at the same time as you are doing, and also ensures that you can't update the image while something else is already doing so. Note that the example above uses the context handler `siril.image_lock()` **before** getting the image from Siril, not just before updating it. This ensures the image is locked throughout the time it is being processed until it is updated back in Siril and the lock is finally released.

In a similar way you can access image statistics, FITS header information, bit depth and so on. You can also access sequence information, for example whether the *n* th image in a sequence is included, what its stats are and so on.

### 20.2.6 Installation of Python Modules

You might have noticed we are using NumPy in the previous examples. This is a dependency of the siril module and will be pulled in during initial setup of the venv. Other python modules can be imported using the standard `import` command, for example `import astropy as ap`. However, for ease of ensuring that dependencies are installed if a user doesn't already have them, the module also provides the `ensure_installed()` method. This is used as follows:

```
s.ensure_installed("astropy")
import astropy
```

(or, for another example,

```
s.ensure_installed("tifffile")
import tifffile
```

First a check is done to try importing the module (in this case, astropy or tiffile). If the module is not available for import, an attempt is made to install it. If the install fails, an exception is raised and the script will halt.

> **Tip**
>
> If a module requires installation this may take a short time, however the delay only occurs on the first run of a script. On subsequent runs the model is already installed, so the `ensure_installed()` check is almost instantaneous. A message in the log notifies the user that module installation is occurring.

If the check succeeds, the module can be imported as normal using `import`.

> **Tip**
>
> **All other non-core modules except for numpy should be checked with `ensure_installed()` as this automates the installation process for users who don't already have the required modules installed.**

> **Warning**
>
> All Siril python scripts share the same venv. When importing modules, it is important to avoid overconstraining the version requirements in order to avoid clashes between the modules required by different scripts. Only ">=" version constraints should be used, never "==" or "<=".

```
import sirilpy as s
s.ensure_installed("astropy")
import astropy as ap
```

This will automatically pull in all of astropy's dependencies as well.

### 20.2.7 Use of External Modules

> **Tip**
>
> The `siril.FFit` object type is not the same as astropy.io.fits. Whereas astropy.io.fits provides a general file-based interface to all FITS files, siril.FFit provides an interface to the data structure Siril uses to represent FITS images. The two are not directly interchangeable! Either method can be used to obtain the pixel data as a NumPy array.

Whether you obtain the pixel data directly from the `siril.FFit` object or using astropy.io.fits, siril modules such as astropy, photutils and matplotlib provide building blocks for a huge range of image analysis including source detection, image segmentation, source deblending and visualisation, and the ecosystem of python modules provides limitless capability.

```
import sirilpy as s
s.ensure_installed("astropy")
import astropy
from astropy.io import fits
with fits.open("filename.fit", mode='update') as image:
   if isinstance(image[0].data, np.ndarray):
      image[0].data *= 2
```

This looks much like our last example - all we do is multiply the pixel data by 2 - but now we have used astropy to open it directly from a FITS file.

### 20.2.8 API Reference

Note that the *API* is very new. While we will try very hard to avoid script-breaking changes to the API released with 1.4.0 throughout the stable 1.4 series, some parts of the API may evolve during the 1.5 development cycle.

## 20.3 Python Script Key Information for Users

### 20.3.1 Obtaining Scripts

End users should take care to obtain scripts from reputable sources.

- A few fundamentally important scripts are distributed as part of the core Siril package and can be found installed in the system data directory. These are tested to the same degree as the rest of Siril and the development team will treat bug reports on these scripts in exactly the same way as bug reports on any other part of Siril.
- 1.4.0 introduces the Scripts repository. We try to keep a list of the scripts with a short description *here*. Anyone may submit scripts to this repository by making an account on gitlab and submitting a merge request. The Siril team will provide a basic level of scrutiny that the scripts are not attempting to do anything malicious, but we do not accept any responsibility for the correct functionality of scripts written by other people and will not provide support for them - the author should be contacted directly.
- Script authors may choose to distribute scripts independently, as has been done in the past. This is fine: scripts can be downloaded and added to a script directory or run via the script editor from anywhere on the filesystem. In this case the Siril team have nothing at all to do with the scripts and cannot even assure you that they will do no harm, so ensure you trust the author.
- Siril does not support automatic verification of signed scripts. This is because script signing can provide a false sense o security ("the script is signed so it can't do anything bad"). This is false: all a script signature says is that the script was signed by a particular person and hasn't been changed since. In other words, it verifies the origin of a script, but says nothing about whether it functions correctly or is safe. **A signed and verified script could still destroy your data if that is what it is written to do!** In the current implementation of python scripting, the majority of scripts are distributed through the centralised script repository in which case the origin is known and scripts have been through a basic level of scrutiny by the development team prior to being merged. Some scripts may be distributed through other channels and downloaded manually, and there is nothing preventing an author providing a sha256sum or a .gpg signature for scripts distributed in this way - you will just need to verify them manually after downloading.

### 20.3.2 Troubleshooting Common Issues

Siril's python script system is quite complex: a lot goes on under the hood, and this is all coordinated by clever use of a Python venv ("virtual environment") which ensures that the python used by Siril is separated from the system Python installation and potentially from other venvs that may be in use by other software. However there are a few things to be aware of:

- venvs are tied to a specific version of python. This means that if your system updates from (e.g.) Python-3.12 to Python-3.13 the venv will no longer point to the right Python program or the right bytecode libraries for compiled modules. **When you upgrade to a new version of Python you must reset the Siril venv**.
- script dependencies are controlled using pip and the pypi archive. This is very good but not 100% foolproof, and in rare cases you may find a dependency issue has arisen that causes a problem of some kind. **If you find an unexplained problem with a script that others aren't experiencing, resetting the venv may solve it.** (And if you find a combination of scripts that can repeatably cause a dependency issue like this, please report it as an issue on the siril-scripts gitlab site as it will help us to improve advice to script authors.)

Resetting the venv is easy: click the *Get Scripts* menu item in the Scripts menu and scroll down to the bottom. Here you will find a "Reset venv" button: clicking it will reset the venv and create a new one. After doing this, the next time you run a script it will need to reinstall its dependencies so the first startup will be a bit slower than ususal (depending on network speed and whether any of the required modules are locally cached).

## 20.4 Python Script Key Information for Authors

We welcome all script authors who wish to write scripts for Siril. However supporting Siril, while fun, takes a lot of time and we have neither the time nor the desire to support other people's work as well as the core Siril software. There are therefore some guidelines to ensure that the division of responsibility is clear.

- All scripts should feature the author's name or handle, as well as a means of contact (website, YouTube channel, forum etc.) where they can be reached with comments or bug reports. For authors publishing independently this is of course guidance, however for authors wishing to publish their scripts via the siril-scripts repository this is mandatory. Scripts will not be accepted unless these details are included.
- Only open-source scripts will be accepted in the Siril scripts repository. Closed source scripts (including ones that feature a source code wrapper that calls a closed source program) must be independently published.
- Authors should also follow the script programming guidelines on the API page. These guidelines ensure that scripts can be used by end-users with minimal friction.

### 20.4.1 Siril-Scripts Repository Code Standards

As a script author you can write whatever scripts you like for your own use. However for scripts to be published to the siril-scripts repository we need to require some coding standards in order to ensure that they work with the standard Siril packages on each of the three supported operating systems. The requirements are:

- Scripts should work with versions of Python >= 3.9. This version matches the oldest Python version in our binary packages, and will be adjusted over time to match the packages.
- Scripts should be single-file scripts. The repository was originally designed for legacy .SSF scripts and the assumption of one file per script remains. We understand that it can be good for code readability to develop complex python programs by splitting them into multiple files. This is fine when working locally, however the multiple files would need to be combined into a single file for submission to the scripts repository.
- If you want to implement a GUI in your script you should use the pyQt6 toolkit.
  - pyQt6 uses the well known Qt framework and it has now been verified as installing correctly from pypi on all the supported platforms. Qt6 is much faster than Tkinter for scripts that themselves have to do significant graphics manipulation, such as pan-and-zoom previews, and it has a nice GUI designer application available in Qt Creator.

> **Warning**
>
> tkinter is now deprecated: new Siril scripts should not use it and should be written using pyQt6 instead. Although tkinter is a core Python module it is outdated, very slow and has issues with Wayland desktops on Linux. tkinter scripts will continue to work throughout the 1.4 stable series, but there is a considerable amount of added complexity in the sirilpy code to support tkinter. Since pyQt6 provides a much better alternative has now been confirmed to work on all the platforms Siril supports, the tkinter support code (the tksiril and tkfilebrowser submodules) will be removed during the 1.5 development cycle and will be removed before 1.6.0.

> **Warning**
>
> Scripts using GTK GUIs will not be accepted into the scripts repository. This GUI framework requires substantial system dependencies. Although Siril itself is coded in GTK, unfortunately the python gi packages required to use it in python scripts are broken on Windows and we are unable to bundle them. If you wish, you can install the dependencies yourself and write scripts using GTK and they may work fine on your own machine, but this is not supported. We will not provide help in getting the prebuilt packages to work with unsupported packages; if you are determined to try this, you will likely need to build Siril from source.

- Avoid import version specifications that specify a package version using "<", "<=" or "==". These constraints will result in conflicts between scripts, which all share the same venv. Specifying a package version using ">=" is fine. Remember to use `sirilpy.ensure_installed()` before importing non-core python module dependencies, to automate the process of installing them.
- Avoid importing packages that require system packages (i.e. ones that cannot be installed using `python3 -m pip install`)
- Ideally, check that your script imports will install on all three target operating systems. If not then the development team will check them after submission, and you will need to fix any issues that occur.

### 20.4.2 Known Good Modules

A list of "known good" packages that we have checked will import without problem, and with all their dependencies, on all the target operating systems, is provided below:

- astropy
- astropy-healpix
- astroquery
- ccdproc
- GaiaXPy
- matplotlib
- numpy
- opencv-python
- pandas
- photutils
- pillow
- pyfftw

- pygaia
- pyQt6
- scipy
- tk
- ttkthemes

> **Tip**
>
> For machine learning / AI scripts, onnxruntime is very strongly recommended. It is cross-platform, it is hardware-agnostic and it falls back gracefully to CPU support if a GPU or GPU system libraries are unavailable. The `sirilpy.utility` submodule provides an `ONNXHelper` class to ensure the correct hardware- and OS-specific version of onnxruntime is installed.

### 20.4.3 Machine Learning Modules

A variety of machine learning modules exist which can be used to write "AI" scripts. However the state of support for heterogenous computing (i.e. GPU use) is patchy and often requires system libraries such as CUDA or ROCm to be installed. The following modules are regarded as key for GPU acceleration and either are supported with helper classes or have helper classes in development.

- **onnxruntime** - used in GraXpert and Riccardo's AberrationRemover.py script, this package is available for all the target OSes. There are different backends to support different GPUs, but in some cases these are fragile. A helper class `ONNXHelper()` is available to help with identification and installation of the optimum package as well as ensuring correct runtime fallback to CPU. Note that currently on Windows the ONNXHelper will select the DirectML runtime for all supported GPUs. Even though in theory the CUDA-based onnxruntime-gpu package might be a bit faster, it unfortunately depends heavily on system libraries being installed and configured correctly and the installation is considered too fragile to be reliable in the kind of automated python environment used by Siril. If users wish to use it and are confident they can get it to work on their system they can install it manually using `sirilpy.ensure_installed("onnxruntime-gpu")` but this is not supported and related issues will be closed.
- **torch** - This is a heavyweight among ML frameworks and where GPU support was once very much CUDA-based it is now broadening its coverage: support for NVidia and Apple Silicon GPUs is good, and support for AMD GPUs (using ROCm) and Intel GPUs is in development (ROCm is supposedly stable enough to use on Linux, not yet on Windows). Unfortunately torch has some dependency issues (it is excessively strict about the version of CuDNN it requires) which causes problems using it in the same venv as some other modules (including jax - see below) that pull in the nvidia-cudnn package. A `TorchHelper()` class is available which manages this by first installing torch, then reinstalling it with the --no-deps flag, which prevents it complaining if another package updates the dependencies to a higher version number. This is really ugly and it's unfortunate that it's necessary, but it works. If you envisage using CUDA versions of both torch and jax, it is better to install torch first and then jax.
- **jax** - Not a dedicated ML framework as such but a means of offloading computation to the GPU. jax provides a mostly numpy-compatible module called jax.numpy which can greatly accelerate many mathematical array processing operations by offloading them to the GPU and also provides functions for gradient computation that numpy does not. Bearing in mind that the slow part of GPU processing is the data transfer between system memory and GPU memory, jax provides a jit compiler to precompile its offloaded workloads but it still requires careful algorithm design to keep data on the GPU for as long as possible, only returning the final result. Jax has good GPU support on Linux but only CPU support on Apple and Windows, although experimental Metal support and Windows GPU support is in progress. This is therefore not recommended for actual scripts at present, but is highlighted as an interesting framework to develop against in the (hopefully) near future when its support broadens. A `JaxHelper()` class is under development that will help with selection of the correct jax package to install and testing to confirm it works properly.

- **Dependencies** Note that both torch and jax use GPU library pypi packages, so installation of them may pull in very large chains of dependencies - the full set of NVIDIA packages (CUDA, CuDNN etc.) is around 1GB. So when using these in scripts be sure to warn users that they may incur a large and potentially slow download on first use!

### 20.4.4 Excluded Modules

Completely avoid using the following modules, as they are known to cause problems on one or more OSes:

- healpy (doesn't work on Windows; astropy_healpix can be used instead)
-

### 20.4.5 Modules with System Dependencies

The following modules require installation of system packages which cannot be automated using pip, and therefore should not be directly used in Siril scripts:

- `gi` (pygtk etc - requires system package installation)
-

> **Warning**
>
> **We cannot emphasise enough that such packages are not supported.** If they work for you locally, that's a lucky bonus for you. They generally won't work with any of the prebuilt binary packages (AppImage, flatpak, Windows, MacOS). These kind of packages are specifically not supported and we will not provide help with getting them to work. All tickets relating to this kind of question will be closed as a duplicate of #1527.
>
> Yes, we know it's a shame. In fact we would have liked to use GTK as the toolkit for writing sirilpy scripts, but unfortunately the binary wheel has been broken on Windows for a long time and doesn't show any sign of being fixed. We can't release core functionality that doesn't work on one of the target OSes.

### 20.4.6 Workaround for Modules with System Dependencies

It may still be possible to write projects that use these kind of packages, but you will need to package them with their dependencies and distribute them independently of Siril or the scripts repository, and the sirilpy module can still be used to achieve integration with Siril. If you want to, you can provide a wrapper script that can be distributed in the scripts repository and just initializes your main program. We will not help with any of that. As stated above, use of these packages in Siril scripts is not supported.

### 20.4.7 Closed Source Scripts

The python interface will happily run precompiled .pyc files. Siril is a Free and Open-Source software, and we therefore do not encourage development of closed-source scripts, however we don't forbid it either. Practically we can't, because even stringent interpretations of the GNU Public License allow for writing an open source shim that sits between Siril and a closed source application. Moreover we are in favour of freedom, and while we choose to release Siril under the GPL to provide freedom to our users, we also respect the freedom of developers to choose how they release their own work.

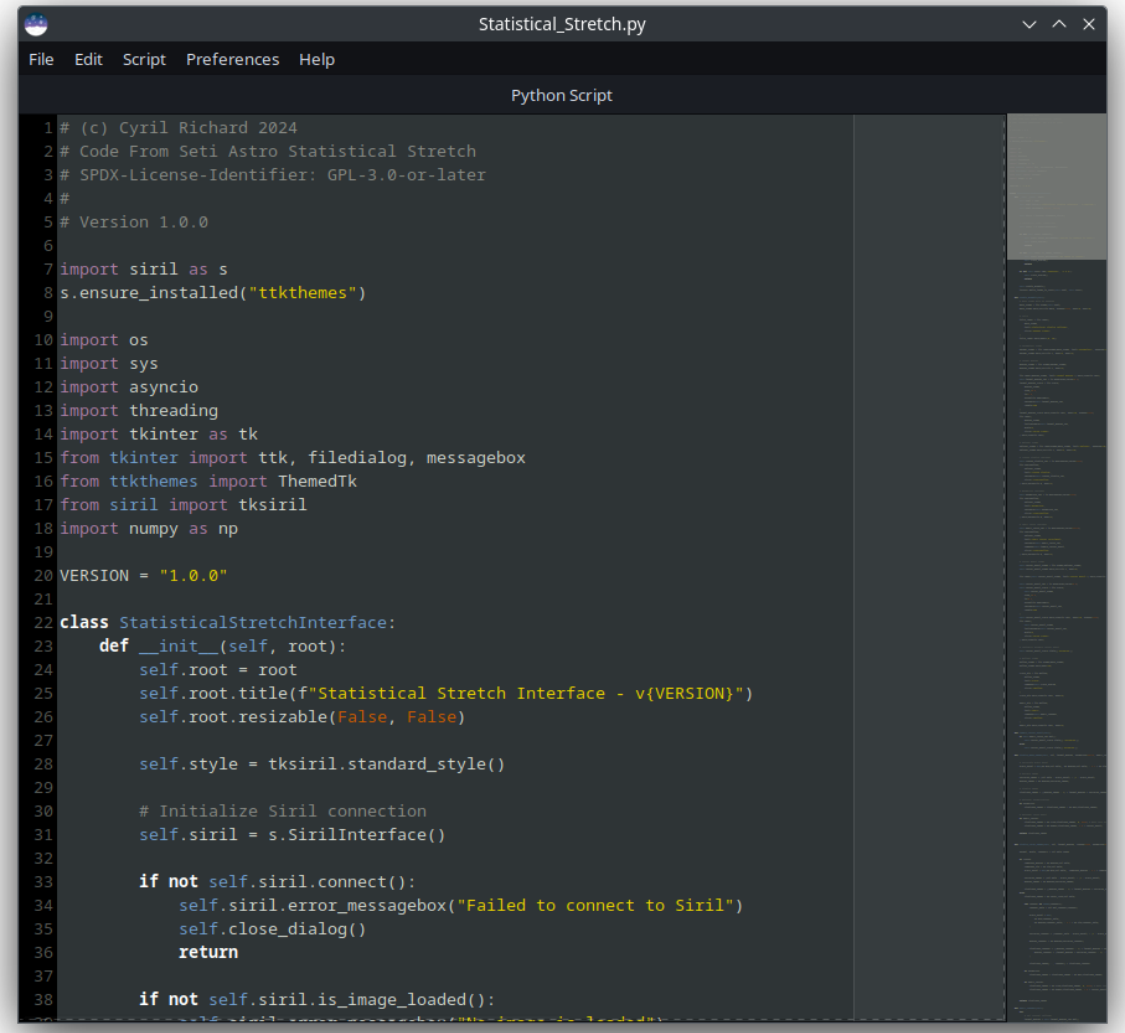
There is some history here, too: Siril has since 1.2.0 provided a built-in interface to Starnet, however Starnet was originally open source and only closed its doors later on.

However, if an author chooses to release a Siril script as a closed-source .pyc then they need to arrange all matters to do with distributing it: as we are unable to inspect the code ourselves to perform even the most cursory checks, we cannot host it in the Siril-scripts git repository. And of course, responsibility for supporting such products is entirely a matter for the author - the Siril team will offer no advice on such products.

Note that scripts must be compiled separately for each version of python: helloworld.py would be compiled with `python3 -m compileall helloworld.py` which would generate helloworld.cpython-<version>.pyc, where <version> would be 312 for Python 3.12, 313 for Python 3.13 and so on depending on the version of python used to compile the script. This is an inherent inconvenience to closed source scripts.

## 20.5 Script Editor

Siril contains a script editor, accessible via the Scripts menu. This provides a feature rich code editor tailored for writing both Siril Script Files and Python scripts. The theme (light / dark) of the script editor follows that set in the main *Preferences* dialog. The main editor window is shown below:




For most of the functionality shortcuts exist, which are as standardised as possible.

### 20.5.1 File Menu

The file menu provides the usual functionality:

- **New**: begins a new file. If there is unsaved text in the buffer a confirmation dialog will be presented to give the user a chance to save their previous work.
- **Open**: open a file from disk. If If there is unsaved text in the buffer a confirmation dialog will be presented to give the user a chance to save their previous work.
- **Recent files**: provides quick access to open recent files.
- **Save**: saves the current file with its existing filename. If the file is not yet saved then this behaves in the same way as Save as, prompting for a filename.
- **Save as**: saves the current file, prompting for the filename to save it with.
- **Close**: closes the file and the script editor window. If there is unsaved text in the buffer a dialog will be presented to give the user a chance to save their work.

Unsaved files are indicated by an asterisk (*) next to the filename in the titlebar. When changes are saved, the asterisk will disappear.

### 20.5.2 Edit Menu

The edit menu provides the following functionality:

- **Undo**: undoes the most recent change in the script. The script editor has an unlimited undo / redo buffer.
- **Redo**: redoes the most recent undo in the script.
- **Cut**: cuts the current selection to the clipboard. If no selection is made, the entire current line is cut.
- **Copy**: copies the current selection to the clipboard.
- **Paste**: pastes the current clipboard content to the cursor position.
- **Find**: Presents the *Find overlay* which provides an active search capability. The editor will scroll to the first occurrence of the search term as you type. All occurrences are highlighted in the editor view and in the minimap, if enabled.

### 20.5.3 Script Menu

The script menu provides the following functionality:

- **Run**: runs the current script. Note that this obeys the current script type selection (Siril Script File / Python Script) and this must be correct for the script you are editing. The script type is automatically set based on the file extension when loading a file, but you must set it here if you have not yet saved your script. At startup it defaults to Python Script. Note that owing to a shortcut clash in some desktop environments, as well as the listed `F5`, `Ctrl-R` also works to run the current script.
- **Python scripts**: sets the current script mode to Python scripts. This affects the syntax highlighting as well as the manner in which the script is run.
- **Siril Script Files**: sets the current script mode to Siril Script Files. This affects the syntax highlighting as well as the manner in which the script is run.
- **Enable test arguments**: Shows an entry below the main script view. This can be used to enter arguments that will be passed to the script when run from the editor. Python scripts can take arguments to set parameters: this provides an alternative to using a GUI to set parameters. Even if a GUI is provided in a script, the ability to pass arguments means that it can be used as a Siril command via the `pyscript` command, and can therefore itself be used from within scripts.

- **Enable python debug mode**: This toggle changes the behaviour of the sirilpy module to support debugging. When the SirilInterface is created at the start of the script, a modal information box will appear showing the process ID of the python interpreter, which can be used to connect a debugger. Also, all interface timeouts will be set to None, to allow for prolonged inspection of the state of a running script.

  Note that the test arguments have no effect on Siril Script File scripts, as these have no ability to handle arguments.

### 20.5.4 Preferences

The preferences menu provides the following options:

- **Highlight syntax**: when enabled, Siril will use language-aware syntax highlighting to highlight aspects of your script.
- **Enable right margin indicator**: when enabled, Siril will show a right margin indicator at a set column number. This is useful for helping to avoid excessively long lines of code and maintain readability.
- **Right margin position**: this will present a small dialog allowing the user to set the column number of the right margin indicator discussed above. This defaults to 80.
- **Enable bracket matching**: when enabled, if the cursor is on a bracket the editor will highlight the matching bracket. This can be useful to avoid bracket mismatches.
- **Show line numbers**: when enabled, line numbers are shown to the left of the editor view. This can help with debugging and navigating code.
- **Show line markers**: when enabled, line markers are shown to the left of the editor view. No functionality currently uses line marks, but the ability to view them is provided to support future developments.
- **Highlight current line**: when enabled, the current line is highlighted in the editor view.
- **Enable auto-indentation**: when enabled, on pressing Enter the new line will begin at the same level of indentation as the previous line.
- **Indent on tab**: when enabled, pressing tab with a selection made that covers multiple lines will cause all the selected lines to be indented a level, and pressing shift-tab will cause them to be unindented a level.
- **Enable smart backspace**: when enabled, pressing backspace with whitespace to the left of the cursor will delete whitespace back to the previous indentation level.
- **Smart Home / End**: when enabled, Home and End will move the cursor to the first or last non-whitespace character of the line respectively, rather than the absolute first or last character of the line.
- **Show spaces and tabs**: when selected, spaces and tabs will be shown with visible characters (central dots for spaces, right-facing arrows for tab characters).
- **Show newlines**: when selected, newlines will be shown with visible characters (down-and-left bent arrows).
- **Show minimap**: when selected, a minimap will be shown to the side of the editor view, supporting navigation and location of `Find` occurrences.

### 20.5.5 Help

The help menu provides API help for both Siril Script Files and Python scripts.

- **Python API Reference**: this opens the Python API page of the online manual in the default browser.
- **Commands Reference**: this opens the Commands Reference page of the online manual in the default browser, which is essentially the API for Siril Script Files.

## 20.6 Python Script Examples

This section highlights examples of python scripting. These are simple examples intended to provide an introduction to the API, and are explained with commentary.

For a more complete example of a script using a TKinter interface to provide a full-featured interface to an external program, see the CosmicClarity_denoise.py and CosmicClarity_sharpen.py scripts which are available in the Siril scripts repository.

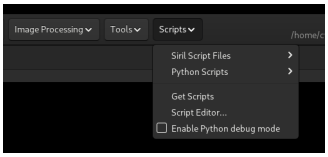



Fig. 3: Scripts menu

### 20.6.1 Hello, Siril!

It is traditional to start with a Hello, World! example, so here is "Hello, Siril!"

```
import sirilpy as s
from sirilpy import SirilConnectionError

siril = s.SirilInterface()

try:
   siril.connect()
   siril.log("Hello, Siril!")

except SirilConnectionError as e:
   print(f"An error occurred: {e}")
```

That's it. All siril python scripts should import the siril module: this is automatically provided by Siril and does not require installation.

It allows connection to Siril through the **SirilInterface** class and gives access to Siril methods: in this case the `log()` function does a tiny bit of adjustment (adding a `\n` newline character) and calls the internal function `siril_log_message()` to display the result in Siril's log tab. Note that `log()` can take an optional argument specifying the color: if you wanted "Hello, Siril!" to print in green you would change the line to `siril.log("Hello, Siril!", s.LogColor.GREEN)`.

Note that there is generally no need to call `siril.disconnect()` call, as it is not really necessary: at the end of a script the method will be called automatically in an onexit handler.

Also note that although you can initialize more than one `SirilInterface()` object, a script can only connect via one at a time (though multiple scripts can be connected to Siril simultaneously). There is generally no need to create more than one `SirilInterface()`.

### 20.6.2 Hello, astropy!

This script shows how to save a temporary working copy, open it using astropy.io.fits and modify it before saving the file and reopening it in Siril.

```
import sirilpy as s
import tempfile
```

(continues on next page)

(continued from previous page)

```
import os
import gc
import numpy as np # numpy is a dependency of the siril module and
# will always be available, so no need to use s.ensure_installed() here

s.ensure_installed("astropy")
import astropy
from astropy.io import fits
siril = s.SirilInterface()
try:
   siril.connect():
   print("Connected successfully!")
except SirilConnectionError as e:
   print(f"Connection failed: {e}")
   quit()

def hello_astropy():
   siril.log("Starting the process...")

   temp_filename = None
   try:
      # Create a temporary file
      with tempfile.NamedTemporaryFile(suffix=".fits", delete=False) as temp_file:
            temp_filename = temp_file.name
            siril.log(f"Temporary file created: {temp_filename}")

      # Save current file
      siril.cmd("save", temp_filename)
      siril.log(f"FITS file saved: {temp_filename}")

      # Open and modify FITS file using Astropy
      with fits.open(temp_filename, mode='update') as image:
            siril.log("Opened FITS file with Astropy")

            # Modify the FITS file data
            if isinstance(image[0].data, np.ndarray):
               image[0].data *= 2
               siril.log("Modified FITS file data using Astropy (multiplied pixel values␣
↪by 2)")

            image[0].header['COMMENT'] = "Modified by Astropy"
            siril.log("Added COMMENT to FITS file header")

            image.flush()  # Ensure changes are written
            siril.log(f"Changes saved to FITS file: {temp_filename}")

      # Load back into Siril
      siril.cmd("load", temp_filename)
      siril.log(f"FITS file loaded: {temp_filename}")

      siril.log("Process completed successfully")
```

(continues on next page)

(continued from previous page)

```
    except Exception as e:
        siril.log(f"An error occurred: {str(e)}")
        raise  # Optionally re-raise if needed for further handling

    finally:
        # Clean up: delete the temporary file
        if temp_filename and os.path.exists(temp_filename):
            try:
                os.remove(temp_filename)
                siril.log(f"Temporary file deleted: {temp_filename}")
            except OSError as e:
                siril.log(f"Failed to delete temporary file: {str(e)}")

        # Garbage collection using gc.collect() is not required at the end
        # of a script as it gets run automatically just after the script
        # hands back to the interpreter

# Run the function
if __name__ == "__main__":
    hello_astropy()
```

Note the use of error handling in this example, using the try: and except: blocks, and the cleanup in the finally: block.

### 20.6.3 Pixel Data and Metadata

The sirilpy python module offers direct access to a wide range of data.

```
import sirilpy as s
import numpy as np
siril = s.SirilInterface()
try:
    siril.connect():
    print("Connected successfully!")
except SirilConnectionError as e:
    print(f"Connection failed: {e}")
    quit()

def siril_module_example():
    try:
        # Lock the main image to prevent anything else altering it while we
        # are working on it
        with siril.image_lock():
            # Get the global fits object
            fit = siril.get_image()

            siril.log(f"The value of the first pixel is: {fit.data.flat[0]}")
            fit.data[:] *= 2
            siril.set_image_pixeldata(fit.data)

        siril.log(f"The value of the first pixel is now: {fit.data.flat[0]}")
        siril.log(f"Data type: {fit.data.dtype}")
        siril.log(f"Array shape: {fit.data.shape}")
```

(continues on next page)

(continued from previous page)

```
        history = fit.history
        siril.log(f"History: {history}")

    except AttributeError as e:
        siril.log(f"Error: {e}")
        fit = None  # Ensure the FFit object is released

# Run the function
if __name__ == "__main__":
    siril_module_example()
```

As before, we import the siril module: we now also import numpy, which we will use to operate on the pixel data in the image loaded in Siril. Note that there is no need to use the `import_or_install()` method, as numpy is a dependency of the siril python module and will have been preinstalled in the venv.

Next, we demonstrate various ways of interacting with the image data and metadata:

```
# Get the global fits object
with siril.image_lock():
    fit = siril.get_image(True) # True indicates that we want the pixel data
    # not just the metadata. It is not required as this is the default, but
    # if you only want the metadata you can specify False here to save a bit
    # of memory.

    siril.log(f"The value of the first pixel is: {fit.data.flat[0]}")
    fit.data[:] *= 2
    siril.set_image_pixeldata(fit.data)
    siril.log(f"The value of the first pixel is now: {fit.data.flat[0]}")
```

We get the current image using `siril.get_image()`

> **Tip**
>
> Note that `siril.fits` is not at all the same as `astropy.io.fits`. While astropy.io.fits provides a general-purpose interface to all FITS files, siril.fits provides an interface matching the fits data structure used in Siril.

We then access the pixel data as a NumPy array and carry out operations on this array to modify the image data. In this case we multiply all the pixel values by 2.

After we have finished modifying the image, we call `siril.set_image_pixeldata(fit.data)` to update the pixel data in Siril and trigger the Siril GUI to redraw to show the updated image. Along the way we use `siril.log()` to highlight the change to the first pixel.

Next we show the array datatype and shape:

```
siril.log(f"Data type: {fit.data.dtype}")
siril.log(f"Array shape: {fit.data.shape}")
```

Next we show how we can access the image history. This is provided as a list of individual HISTORY FITS headers, so we join them together and add newline characters before logging.

```
history = fit.history
siril.log(f"History: {history}")
```

Full details of the API can be obtained by importing the Siril module and calling the `help(sirilpy)` command, or more specifically by calling `help()` for the particular part of the API you want to know about, e.g. `help(sirilpy.FFit)`.

As docstrings are not translatable, for translations of the API help you will need to consult the API documentation page in this online manual.

### 20.6.4 siril.plot submodule

This script shows how to use the siril.plot submodule to create a simple plot with three series. The first is a simple x-y plot with lines; the second is an x-y plot with error bars; and the third is another simple x-y plot but shows use of numpy arrays as well as how a SeriesData object can be created and added directly to the PlotData object using `add_series_obj`.

```
import sirilpy as s
import numpy as np
siril = s.SirilInterface()
try:
   siril.connect()
except SirilConnectionError as e:
   print(f"Error: failed to connect to Siril: {e}")
   quit()

# Series data using lists of floats
plot_data = s.PlotData(
   "Example Plot",
   "X Axis",
   "Y Axis",
   "example_plot_lists",
   True,
   datamin = [0.0, 0.0]
   )

plot_data.add_series([1.0, 2.0, 3.0],
                     [4.0, 5.0, 6.0],
                     "Series 1 (Lists)")

plot_data.add_series([4.0, 5.0, 6.0],
                     [7.0, 8.0, 9.0],
                     "Series 2 (Lists with errors)",
                     s.PlotType.MARKS,
                     [0.35, 0.4, 0.45],
                     [0.35, 0.4, 0.45])

x_arr = np.array([1.0, 2.0, 3.0, 4.0], dtype=np.float32)
y_arr = np.array([1.0, 4.0, 9.0, 16.0], dtype=np.float32)
series3 = s.SeriesData(x_arr, y_arr, "Series 3 (np.arrays)")
plot_data.add_series_obj(series3)

# Serialize using lists
siril.xy_plot(plot_data)
```

### 20.6.5 Templates for Scripts with GUIs

The following templates show how to create scripts with a GUI. The first template is the simplest and only provides a GUI interface:

Listing 1: gui_template.py

```python
# Template for Siril script with GUI only (no CLI)
# SPDX-License-Identifier: # insert license identifier here
# insert author's name and details here

"""
This script template provides an exemplar for use in constructing sirilpy
scripts that provide a Tkinter GUI. Such scripts are only usable via the GUI.
This is a simplified version without command-line argument support.
"""

# Core module imports
import os
import sys
import math
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import numpy as np

import sirilpy as s
from sirilpy import tksiril, SirilError
s.ensure_installed("ttkthemes")
# Add other calls to ensure_installed() here to ensure that any non-core
# modules that are required by the script are installed and available
# to be imported.

# Note: pylint will complain that the following imports are not at the
# top of the module. This deviation from python style is required in order
# to run ensure_installed() to ensure that any non-core modules are
# available before attempting to import them.
from ttkthemes import ThemedTk
# Add any additional imports here

VERSION = "1.0.0"

def template_algorithm(fit,
                       example_float_var,
                       example_bool_var,
                       example_file_path_var):
    """
    Template processing function, insert your algorithm here.
    This is the non-GUI part of the script and is entirely up to you to write.
    Although presented in the template as a single method it may comprise
    classes, methods and anything else.
    """
    print(f"Argument: {example_float_var}")
    print(f"Bool var: {example_bool_var}")
    print(f"File path: {example_file_path_var}")
```

(continues on next page)

(continued from previous page)

```
    return fit

class TemplateScriptInterface:
    """ This class provides the GUI and related callbacks """
    def __init__(self, root):
        self.root = root
        self.root.title(f"Template Script Interface - v{VERSION}")
        self.root.resizable(False, False)
        self.style = tksiril.standard_style()

        # Initialize Siril connection
        self.siril = s.SirilInterface()

        try:
            self.siril.connect()
        except s.SirilConnectionError:
            self.siril.error_messagebox("Failed to connect to Siril")
            return

        # Initial checks: example - check if an image is loaded
        if not self.siril.is_image_loaded():
            self.siril.error_messagebox("No image is loaded")
            return

        # Check if the version of Siril is high enough
        try:
            self.siril.cmd("requires", "1.3.6")
        except s.CommandError:
            return

        # Create the UI and match its theme to Siril
        self.create_widgets()
        tksiril.match_theme_to_siril(self.root, self.siril)

    # This function is used to round values to n decimal places, it is
    # always useful if you have sliders in your UI.
    def _floor_value(self, value, decimals=2):
        """Floor a value to the specified number of decimal places"""
        factor = 10 ** decimals
        return math.floor(value * factor) / factor

    # Here is an example function to round the displayed value of a
    # variable to 2 decimal places. You will need one of these for each
    # slider control you have.
    def _update_example_float_variable_display(self, *args): # pylint: disable=unused-
↪argument
        """Update the displayed template variable value with floor rounding"""
        value = self.example_float_var.get()
        rounded_value = self._floor_value(value)
        self.example_float_var_display_var.set(f"{rounded_value:.2f}")

    def _browse_file(self):
```

(continues on next page)

(continued from previous page)

```
        """
        Use a TK filedialog to browse for a file. Note that this callback is
        specific to the variable being updated; you might need to have several
        similar callbacks if you have more than one file selection widget.
        """
        filename = filedialog.askopenfilename(
            title="Select File",
            initialdir=os.path.expanduser("~")
        )
        if filename:
            self.example_file_path_var.set(filename)

    def create_widgets(self):
        """Create the GUI's widgets, connect signals etc. """
        # Main frame with no padding
        main_frame = ttk.Frame(self.root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=0, pady=0)

        # Title
        title_label = ttk.Label(
            main_frame,
            text="Template Script",
            style="Header.TLabel"
        )
        title_label.pack(pady=(0, 20))

        # Parameters frame
        params_frame = ttk.LabelFrame(main_frame, text="Parameters", padding=10)
        params_frame.pack(fill=tk.X, padx=5, pady=5)

        ##############################################################
        # Example control with an embedded variable in its own frame
        ##############################################################

        template_variable_frame = ttk.Frame(params_frame)
        template_variable_frame.pack(fill=tk.X, pady=5)

        ttk.Label(template_variable_frame,
                  text="Template example_float_var:").pack(side=tk.LEFT)

        # Initialize variables with default values
        self.example_float_var = tk.DoubleVar(self.root, value=1.0)
        self.example_float_var_display_var = tk.StringVar(self.root, value="1.00")

        # Add trace to update display when slider changes
        self.example_float_var.trace_add("write",
                                         self._update_example_float_variable_display)

        example_float_var_scale = ttk.Scale(
            template_variable_frame,
            from_=0.0, # set your range minimum here
            to=1.0,    # set your range maximum here
```

(continues on next page)

(continued from previous page)

```
            orient=tk.HORIZONTAL, # oriented horizontally
            variable=self.example_float_var, # the tk variable the widget controls
            length=200
        )
        example_float_var_scale.pack(side=tk.LEFT, padx=10, expand=True)
        ttk.Label(
            template_variable_frame,
            textvariable=self.example_float_var_display_var, # var truncated to 2 d.p.
            width=5,
            style="Value.TLabel"
        ).pack(side=tk.LEFT)
        tksiril.create_tooltip(example_float_var_scale,
                               "Adjusts the template variable.")

        ################################
        # Add frame for other variables
        ################################
        options_frame = ttk.LabelFrame(main_frame, text="Options", padding=10)
        options_frame.pack(fill=tk.X, padx=5, pady=10)

        ##################
        # Example checkbox
        ##################
        self.example_bool_var = tk.BooleanVar(self.root, value=False)
        example_checkbox = ttk.Checkbutton(
            options_frame,
            text="Example checkbox variable",
            variable=self.example_bool_var,
            style="TCheckbutton"
        )
        example_checkbox.pack(anchor=tk.W, pady=2)
        tksiril.create_tooltip(example_checkbox, "Example checkbox.")

        ##########################################################
        # Example file selection
        # using an entry and a callback that triggers a filedialog
        # These file selector widgets have their own frame
        ##########################################################
        file_frame = ttk.Frame(options_frame)
        file_frame.pack(fill=tk.X, pady=5)

        ttk.Label(file_frame, text="File:").pack(side=tk.LEFT)

        self.example_file_path_var = tk.StringVar(self.root, value="")
        example_file_entry = ttk.Entry(
            file_frame,
            textvariable=self.example_file_path_var,
            width=40
        )
        example_file_entry.pack(side=tk.LEFT, padx=(5, 5), expand=True)

        ttk.Button(
```

(continues on next page)

(continued from previous page)

```
            file_frame,
            text="Browse",
            command=self._browse_file,
            style="TButton"
        ).pack(side=tk.LEFT)

        # The template shows examples of sliders and checkboxes
        # however you can add other sorts of TKinter widgets here

        # Buttons frame
        button_frame = ttk.Frame(main_frame)
        button_frame.pack(pady=20)

        close_btn = ttk.Button(
            button_frame,
            text="Close",
            command=self.close_dialog,
            style="TButton"
        )
        close_btn.pack(side=tk.LEFT, padx=5)
        tksiril.create_tooltip(close_btn,
                               "Close the interface and disconnect from Siril. "
                               "No changes will be made to the current image.")

        apply_btn = ttk.Button(
            button_frame,
            text="Apply",
            command=self.apply_changes,
            style="TButton"
        )
        apply_btn.pack(side=tk.LEFT, padx=5)
        tksiril.create_tooltip(apply_btn,
                               "Apply the script function with the set parameters "
                               "to the current image. Changes can be undone using "
                               "Siril's undo function.")

    def apply_changes(self):
        """ Get the necessary variables from the GUI and call the algorithm """
        try:
            # Get the thread
            with self.siril.image_lock():
                # Get values from the GUI widgets
                example_float_var = self.example_float_var.get()
                example_bool_var = self.example_bool_var.get()
                example_file_path_var = self.example_file_path_var.get()

                # Get current image
                fit = self.siril.get_image()
                fit.ensure_data_type(np.float32)

                # Save original image for undo
                self.siril.undo_save_state(f"Script algorithm: "
```

(continues on next page)

(continued from previous page)

```
                                        "arg={example_float_var:.2f}")

                # Apply script algorithm
                # Your image processing functions go here
                result = template_algorithm(fit, example_float_var, example_bool_var,
                                            example_file_path_var)

                # Clip and update image data
                fit.data[:] = np.clip(result, 0, 1)
                self.siril.set_image_pixeldata(fit.data)

        except SirilError as e:
            messagebox.showerror("Error", str(e))

    def close_dialog(self):
        """ Close dialog """
        self.root.quit()
        self.root.destroy()

def main():
    """ Main entry point """
    try:
        # Create the GUI interface
        root = ThemedTk()
        TemplateScriptInterface(root)
        root.mainloop()
    except SirilError as e:
        print(f"Error initializing script: {str(e)}", file=sys.stderr)
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Download gui_template.py

The second template is very similar, but provides an alternative interface using a command-line argument vector. This allows the script to be called with arguments using the `pyscript` Siril command: it is useful for scripting algorithms that you might wish to use as part of a bigger scripted workflow.

Listing 2: gui_and_args_template.py

```
# Template for Siril script with a GUI
# SPDX-License-Identifier: # insert license identifier here
# insert author's name and details here

"""
This script template provides an exemplar for use in constructing sirilpy
scripts that provide a Tkinter GUI but which can also be called using an
argument vector (i.e. so that they can be scripted for use with the Siril
pyscript command). A similar template exists for pure GUI scripts, which
is slightly simpler owing to not having to accommodate command-line
arguments.
"""
```

(continues on next page)

(continued from previous page)

```python

# Core module imports
import os
import sys
import math
import argparse
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import numpy as np

import sirilpy as s
from sirilpy import tksiril, SirilError
s.ensure_installed("ttkthemes")
# Add other calls to ensure_installed() here to ensure that any non-core
# modules that are required by the script are installed and available
# to be imported.

# Note: pylint will complain that the following imports are not at the
# top of the module. This deviation from python style is required in order
# to run ensure_installed() to ensure that any non-core modules are
# available before attempting to import them.
from ttkthemes import ThemedTk
# Add any additional imports here

VERSION = "1.0.0"

def template_algorithm(fit, example_float_var,
                       example_bool_var,
                       example_file_path_var):
    """
   Template processing function, insert your algorithm here.
   This is the non-GUI part of the script and is entirely up to you to write.
   Although presented in the template as a single method it may comprise
   classes, methods and anything else.
   """
    print(f"Argument: {example_float_var}")
    print(f"Bool var: {example_bool_var}")
    print(f"Bool var: {example_file_path_var}")
    return fit

class TemplateScriptInterface:
    """ This class provides the GUI and related callbacks """
    def __init__(self, root=None, cli_args=None):
        # If no CLI args, create a default namespace with defaults
        if cli_args is None:
            parser = argparse.ArgumentParser()
            parser.add_argument("-example_float_var", type=float, default=1.0)
            parser.add_argument("-example_bool_var", type=bool, default=False)
            parser.add_argument("-example_file_path_var", type=str, default="")
            # add other arguments to match the ones in main()
            cli_args = parser.parse_args([])

```

(continues on next page)

(continued from previous page)

```
        self.cli_args = cli_args
        self.root = root

        if root:
            self.root.title(f"Template Script Interface - v{VERSION}")
            self.root.resizable(False, False)
            self.style = tksiril.standard_style()

        # Initialize Siril connection
        self.siril = s.SirilInterface()

        try:
            self.siril.connect()
        except s.SirilConnectionError:
            if root:
                self.siril.error_messagebox("Failed to connect to Siril")
            else:
                print("Failed to connect to Siril")
            return

        # Initial checks: example - check if an image is loaded
        if not self.siril.is_image_loaded():
            if root:
                self.siril.error_messagebox("No image is loaded")
            else:
                print("No image is loaded")
            return

        # Check if the version of Siril is high enough
        # 1.3.6 is the baseline when python scripting became available but in time
        # you may need to check for a higher version number in order to use newer
        # features.
        try:
            self.siril.cmd("requires", "1.3.6")
        except s.CommandError:
            return

        # Create the UI and match its theme to Siril
        if root:
            self.create_widgets()
            tksiril.match_theme_to_siril(self.root, self.siril)

        # Only apply changes if CLI arguments are non-default
        if cli_args and (cli_args.example_float_var != 1.0):
            self.apply_changes(from_cli=True)

    # This function is used to round values to n decimal places, it is
    # always useful if you have sliders in your UI.
    def _floor_value(self, value, decimals=2):
        """Floor a value to the specified number of decimal places"""
        factor = 10 ** decimals
        return math.floor(value * factor) / factor
```

(continues on next page)

(continued from previous page)

```

    # Here is an example function to round the displayed value of a
    # variable to 2 decimal places. You will need one of these for each
    # slider control you have.

    def _update_example_float_variable_display(self, *args): # pylint: disable=unused-argument
        """Update the displayed template variable value with floor rounding"""
        value = self.example_float_var.get()
        rounded_value = self._floor_value(value)
        self.example_float_var_display_var.set(f"{rounded_value:.2f}")

    def _browse_file(self):
        """
        Use a TK filedialog to browse for a file. Note that this callback is
        specific to the variable being updated; you might need to have several
        similar callbacks if you have more than one file selection widget.
        """
        filename = filedialog.askopenfilename(
            title="Select File",
            initialdir=os.path.expanduser("~")
        )
        if filename:
            self.example_file_path_var.set(filename)

    def create_widgets(self):
        """Create the GUI's widgets, connect signals etc. """
        # Main frame with no padding
        main_frame = ttk.Frame(self.root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=0, pady=0)

        # Title
        title_label = ttk.Label(
            main_frame,
            text="Template Script",
            style="Header.TLabel"
        )
        title_label.pack(pady=(0, 20))

        # Parameters frame
        params_frame = ttk.LabelFrame(main_frame, text="Parameters", padding=10)
        params_frame.pack(fill=tk.X, padx=5, pady=5)

        ##############################################################
        # Example control with an embedded variable in its own frame
        ##############################################################

        template_variable_frame = ttk.Frame(params_frame)
        template_variable_frame.pack(fill=tk.X, pady=5)

        ttk.Label(template_variable_frame,
            text="Example floting point parameter:").pack(side=tk.LEFT)
```

(continues on next page)

(continued from previous page)

```
        # This is the actual float variable
        self.example_float_var = tk.DoubleVar(self.root,
            value=self.cli_args.example_float_var)
        # This is a string representation of the float var limited to 2 decimal places
        # It is used in the scale label
        self.example_float_var_display_var = tk.StringVar(
            self.root,
            value=f"{self._floor_value(self.cli_args.example_float_var):.2f}")
        # Add trace to update display when slider changes
        self.example_float_var.trace_add("write",
            self._update_example_float_variable_display)

        example_float_var_scale = ttk.Scale(
            template_variable_frame,
            from_=0.0, # set your range minimum here
            to=1.0,    # set your range maximum here
            orient=tk.HORIZONTAL, # oriented horizontally
            variable=self.example_float_var, # the tk variable the widget controls
            length=200
        )
        example_float_var_scale.pack(side=tk.LEFT, padx=10, expand=True)
        ttk.Label(
            template_variable_frame,
            textvariable=self.example_float_var_display_var, # var truncated to 2 d.p.
            width=5,
            style="Value.TLabel"
        ).pack(side=tk.LEFT)
        tksiril.create_tooltip(example_float_var_scale,
            "Adjusts the template variable.")

        ################################
        # Add frame for other variables
        ################################
        options_frame = ttk.LabelFrame(main_frame, text="Options", padding=10)
        options_frame.pack(fill=tk.X, padx=5, pady=10)

        ##################
        # Example checkbox
        ##################
        self.example_bool_var = tk.BooleanVar(self.root,
            value=self.cli_args.example_bool_var
        )
        example_checkbox = ttk.Checkbutton(
            options_frame,
            text="Example checkbox variable",
            variable=self.example_bool_var,
            style="TCheckbutton"
        )
        example_checkbox.pack(anchor=tk.W, pady=2)
        tksiril.create_tooltip(example_checkbox, "Example checkbox.")

        ##########################################################
```

(continues on next page)

(continued from previous page)

```
        # Example file selection
        # using an entry and a callback that triggers a filedialog
        ##########################################################
        file_frame = ttk.Frame(options_frame)
        file_frame.pack(fill=tk.X, pady=5)

        self.example_file_path_var = tk.StringVar(self.root,
            value=self.cli_args.example_file_path_var)
        example_file_entry = ttk.Entry(
            file_frame,
            textvariable=self.example_file_path_var,
            width=40
        )
        example_file_entry.pack(side=tk.LEFT, padx=(0, 5), expand=True)

        ttk.Button(
            file_frame,
            text="Browse",
            command=self._browse_file,
            style="TButton"
        ).pack(side=tk.LEFT)

        # The template shows examples of sliders and checkboxes
        # however you can add other sorts of TKinter widgets here

        # Buttons frame
        button_frame = ttk.Frame(main_frame)
        button_frame.pack(pady=20)

        close_btn = ttk.Button(
            button_frame,
            text="Close",
            command=self.close_dialog,
            style="TButton"
        )
        close_btn.pack(side=tk.LEFT, padx=5)
        tksiril.create_tooltip(close_btn,
                               "Close the interface and disconnect from Siril. No "
                               "changes will be made to the current image.")

        apply_btn = ttk.Button(
            button_frame,
            text="Apply",
            command=self.apply_changes,
            style="TButton"
        )
        apply_btn.pack(side=tk.LEFT, padx=5)
        tksiril.create_tooltip(apply_btn,
                               "Apply the script function with the set parameters "
                               "to the current image. Changes can be undone using "
                               "Siril's undo function.")

```

(continues on next page)

(continued from previous page)

```
    def apply_changes(self, from_cli=False):
        """
        Get the necessary variables from CLI args or the GUI and call the algorithm
        """
        try:
            # Get the thread
            with self.siril.image_lock():
                # Determine parameters: prefer CLI args if provided,
                # else use GUI values
                if from_cli and self.cli_args:
                    example_float_var = self.cli_args.example_float_var
                    example_bool_var = self.cli_args.example_bool_var
                    example_file_path_var = self.cli_args.example_file_path_var
                else:
                    example_float_var = self.example_float_var.get()
                    example_bool_var = self.example_bool_var.get()
                    example_file_path_var = self.example_file_path_var.get()

                # Get current image
                fit = self.siril.get_image()
                fit.ensure_data_type(np.float32)

                # Save original image for undo
                self.siril.undo_save_state(f"Script algorithm: "
                                           "arg={example_float_var:.2f}")

                # Apply script algorithm
                # Your image processing functions go here
                result = template_algorithm(fit, example_float_var,
                                            example_bool_var,
                                            example_file_path_var)

                # Clip and update image data
                fit.data[:] = np.clip(result, 0, 1)
                self.siril.set_image_pixeldata(fit.data)

            if from_cli:
                print("Script algorithm applied successfully.")

        except SirilError as e:
            if from_cli:
                print(f"Error: {str(e)}")
            else:
                messagebox.showerror("Error", str(e))

    def close_dialog(self):
        """ Close dialog """
        if hasattr(self, 'root'):
            self.root.quit()
            self.root.destroy()

def main():
```

(continues on next page)

(continued from previous page)

```
    """ Main entry point """
    parser = argparse.ArgumentParser(description="Template python script")
    parser.add_argument("-example_float_var", type=float, default=1.0,
                        help="Describe the variable here (0.0 to 1.0)")
    parser.add_argument("-example_bool_var", type=bool, default=False,
                        help="Describe the variable here (bool, default is False)")
    parser.add_argument("-example_file_path_var", type=str, default="",
                        help="Describe the variable here (example file path)")

    args = parser.parse_args()

    try:
        if any([args.example_float_var != 1.0, args.example_bool_var is not False]):
            # CLI mode
            TemplateScriptInterface(cli_args=args)
        else:
            # GUI mode
            root = ThemedTk()
            TemplateScriptInterface(root)
            root.mainloop()
    except SirilError as e:
        print(f"Error initializing script: {str(e)}", file=sys.stderr)
        sys.exit(1)

if __name__ == "__main__":
    main()
```

Download gui_and_args_template.py

## 20.7 Current list of Python scripts for Siril 1.4

This page is the first shot at getting a list of Python scripts that is somewhat intelligible and searchable. It was done manually and has only a very succinct description for each script, but will be improved with more words for each with time.

### 20.7.1 Image processing section

Scripts that process a single image for visual enhancements, in general a stack result.

| | |
|---|---|
| SyQon/Prism.py | SyQon Prism AI denoise. This is a paying product |
| SyQon/Starless.py | SyQon Starless AI star removal. This is a paying product |
| SyQon/Parallax.py | SyQon Parallax AI: aberration correction, star reduction and sharpening. This is a paying product |
| SyQon/SyQon_Studio_Complete. | New SyQon script that will group all paid models (Parallax, Prism, Starless) |
| RC-Astro/NoiseXTerminator.py | RC Astro AI denoise. This is a paying product |
| RC-Astro/StarXTerminator.py | RC Astro Starless AI star removal. This is a paying product |
| RC-Astro/BlurXTerminator.py | RC Astro Deconv AI: aberration correction, star reduction and sharpening. This is a paying product |
| processing/Statistical_Stretch.py | Seti Astro Statistical Stretch |

continues on next page

Table 1 – continued from previous page

| | |
|---|---|
| processing/AberrationRemover.py | Aberration Remover AI |
| processing/AstroColorMixer.py | multi-pass, hue-targeted color and luminance refinement for nonlinear RGB data. |
| processing/ContinuumSubtraction.py | Narrowband Continuum Subtraction script |
| processing/DeepSNR.py | DeepSNR AI denoise (wrapper around the third-party DeepSNR tool) |
| processing/StarNet.py | StarNet 2.5 star removal (wrapper around the third-party StarNet tool), with optional star mask generation |
| processing/GPS_Process.py | Image processing based on CosmicClarity and GraXpert (background extraction, denoise and/or sharpening) |
| processing/Narrowband_Palette_Picker.p | Helps preview different palettes for narrowband image composition |
| processing/GraXpert-AI.py | Integration of the GraXpert gradient removal tool (AI) |
| processing/CosmicClarity_Denoise.py | Cosmic Clarity Denoise |
| processing/DBXtract.py | Extract the Sulfur II, Hydrogen Alpha and Oxygen III signal from dual band filters to compose SHO images in color cameras |
| processing/Hubble_Palette_from_Band_OSC.py | Create different "Hubble-like" palettes from OSC (One-Shot Color) images acquired with a dual-band Ha/OIII filter |
| processing/PalettePicker.py | Script that help user to create its own palette for narrowband filters |
| processing/NarrowbandNormalization.py | Script that allows easy narrowband normalization on monochrome SHO combined images and on OSC dual band images |
| processing/HDR_multiscale.py | Wavelet-Based Dynamic Range Compression |
| processing/HDR_Blender.py | Inverse Variance Maximum Likelihood Estimate (MLE) HDR Blender. Blends multiple images with different exposures |
| processing/NB_2_RGB.py | Monochrome image combination into a color image |
| processing/SCUNet_Denoise.py | SCUNet image denoiser (AI) |
| processing/CosmicClarity_Superres.py | Cosmic Clarity Superres |
| processing/CosmicClarity_Native.py | Cosmic Clarity AI-powered sharpening, denoising, super resolution and star removal |
| processing/AutoBGE.py | Auto Background Extraction script |
| processing/AutoGradientRemoval.py | Auto Gradient Removal tool. This script will be implemented in the Siril code in the next version |
| processing/CosmicClarity_Satellite.py | Cosmic Clarity Satellite Removal |
| processing/CosmicClarity_Darkstar.py | Cosmic Clarity Darkstar |
| processing/ERBill_Star_Reduction.py | Script for reducing stars using pixel math |
| processing/DSAStar_Reduction.py | Script for reducing stars using pixel math |
| processing/CosmicClarity_Sharpen.py | Cosmic Clarity sharpening process |
| VeraLux/VeraLux_Alchemy.py | Linear-Phase Narrowband Normalization & Mixing |
| VeraLux/VeraLux_Nox.py | Physically-Faithful Photometric Gradient Reduction |
| VeraLux/VeraLux_StarComposer.py | High-Fidelity Star Reconstruction Engine |
| VeraLux/VeraLux_Revela.py | Photometric Local Contrast & Texture Engine |
| VeraLux/VeraLux_HyperMetric_Stre | Photometric Hyperbolic Stretch Engine |

continues on next page

Table 1 – continued from previous page

| | |
|---|---|
| VeraLux/VeraLux_Starting_Point.py | Interactive Workflow Guide & Manual |
| VeraLux/VeraLux_Curves.py | Spline-Based Photometric Sculpting Engine |
| VeraLux/VeraLux_Silentium.py | Linear-Phase Noise Suppression Engine |
| VeraLux/VeraLux_Vectra.py | Vector Color Grading & Chromatic Surgery Engine |
| processing/InteractiveHaloEditor.py | Edit halos, gradients, and artifacts using an unlimited stack of smooth ellipse masks, freehand brushes, and custom spline curves |
| processing/Lighthouse.py | Applies Lightroom-style adjustments |

### 20.7.2 Sequence pre-processing section

Scripts that process a set of images, an alternative to usual preprocessing scripts.

| | |
|---|---|
| preprocessing/AMSP.py | Siril Wizard – Automatic Multi-Session Processing. Drag and drop all files and it does the stacks |
| preprocessing/StorageFriendlyStacking.py | Storage Friendly Stacking Script |
| preprocessing/Naztronomy-Smart_Telescope_PP.py | Naztronomy - Smart Telescope Preprocessing script |
| preprocessing/GPS_Preprocess.py | Preprocessing with multi-session, mosaics and many optional steps support |
| preprocessing/ER-CometStartrail.py | Stack a startrail from a sequence |
| preprocessing/osc-multi-night-stacking-v1.2.py | Multi-night stacking |
| preprocessing/Naztronomy-OSC_PP.py | Naztronomy - OSC Image Preprocessing script with mosaics support |
| preprocessing/Naztronomy-Mono_PP.py | Naztronomy - Monochrome Image Preprocessing script with mosaics support |

### 20.7.3 Utility section

Scripts that provide some tool for data analysis or other non-processing related tasks.

| | |
|---|---|
| core/Siril_Catalog_Installer.py | Siril Catalog Installer |
| core/GPU_Manager.py | A GUI tool for managing ONNX, PyTorch, and JAX installations |
| utility/Selected_Star_Spectrum.py | Plots the Gaia DR3 continuous spectrum for any selected Gaia DR3 source |
| utility/RegistrationInspector.py | Displays the frames of selected images to check framing |
| utility/SuperStack.py | Performs superstacking (moving average) |
| utility/plot3D.py | Plots the current image or selection in 3D using matplotlib |
| utility/Sequence_Statistics_Analyzer. | Analyze frames and plot key statistics |
| utility/Workflow_Companion.py | Deep Space Astro drag-and-drop workflow companion: queue, reorder and launch scripts and Siril functions |
| utility/Workflow_Summarizer.py | Uses Google Gemini to summarize the Siril log into readable workflow documentation |
| utility/Blink_Browse_Filter_Sort.py | Image browser / filter / sorter with adaptive caching |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| utility/Svenesis-GradientAnalyzer.py | Gradient Analyzer |
| utility/Svenesis-MultipleHistogramViewer.py | Multiple Histogram Viewer |
| utility/Svenesis-CosmicDepth3D.py | CosmicDepth 3D |
| utility/Patch_Inpainting_Tool.py | Corrects defects and creates a star mask |
| utility/Svenesis-AnnotateImage.py | Full size image annotation |
| utility/AF_Multi_Crop.py | Multi-crop script |
| utility/Svenesis-BlinkComparator.py | Blink Comparator |
| utility/Satellite_Trail_Removal.py | Remove satellite trails |
| utility/Sequence_Deleter.py | GUI to delete sequences |
| utility/ImageWindow.py | Pseudo-MDI GUI for storing and swapping images |
| utility/Dwarfium_Archive_Selector.p | Prepare sessions from Dwarf telescopes |
| utility/AstroT3kFetch.py | Fetch and classify astrophotography frames |
| utility/Diffraction_Spike_Overlay.py | Add diffraction spikes |
| utility/Flat_On_Flat_Analyzer.py | Analyze flat field effectiveness |
| utility/Distortion3D.py | Plot 3D distortion map |
| utility/Autocrop.py | Autocrop stacked images |
| utility/Signature_Tool.py | Insert a signature/logo |
| utility/Galaxy_Annotations.py | Create galaxy annotations from Simbad queries |
| utility/Asteroid_Comet_Finder.py | Search for asteroids and comets |
| utility/AutoStretch_Preview.py | Interactive AutoStretch preview |
| utility/HertzsprungRussell.py | Create a Hertzsprung-Russell diagram |

**Warning**

**Not all scripts are written by, or the responsibility of, the Siril development team!**

With the introduction of Python scripting the power available to Siril script writers has increased enormously. With great power comes great responsibility!

*You*, as the user, need to ensure you trust the authors of scripts you use. 1.4.0 introduces the script repository. Some of the scripts there have been written by the Siril development team but others have been written by third party contributors. Please read the Key Info section regarding guidance for script authors, end users and the rules for the script repository.

## 20.8 Get Scripts

A small number of basic script files come packaged with Siril, however you can add more from the script repository. To do so, go to the *Scripts menu -> Get Scripts*. This will open the *Scripts* tab of the *Preferences* dialog. Here, below the list of local script directories, you will find a list of scripts in the repository.

The scripts can be sorted by category, script name, type or by whether or not they are selected.

As there may eventually be a large number of scripts, not all of which will be relevant to your workflow, you can configure the scripts you want to appear in the *Scripts* menu. Simply select the checkbox next to them and click Apply to update the menu.

**Tip**

You don't need to select scripts to appear in the menu for them to be usable with the `pyscript` command: this will search all the local script directories as well as the whole of the local scripts repository.

**Warning**

Script authors may submit updates. Occasionally, inevitably, there may be a bad update that breaks the script for you. Bugs need to be reported to the script author: for scripts written by the Siril team you can report them at the scripts GitLab repository and for those written by third party contributors you should report them at the contact details in the script.

The script can be opened in the script editor by double clicking on it in the list. This is important for several reasons:

- It allows you to find the details of the script author for bug reporting.
- When you double click the script there is an option to revert to previous versions.
- This means that if an update broke the script, you can open the previous version and save a local copy in one of your local script folders for use until the script author fixes the bad update. (The version saved in a local folder will take priority over a repository version with the same name, but it will appear in a different subfolder in the menu with the name of your local scripts folder.)

CHAPTER

TWENTYONE

# HEADLESS MODE

Siril can both operate with its graphical user interface (GUI) and with a command line interface (CLI) that does not even require having a display. It can process images for other programs, on remote or embedded computers, using either scripts or real-time generated operations called commands. The capabilities of the headless Siril are in fact those of the *available commands*. There are more than a hundred, allowing preprocessing, processing and photometry analysis to be done automatically.

Commands can also be used in the GUI version of Siril, either from the embedded command line at the bottom of the control panel, or with *scripts*. Scripts are simply a text file containing a list of commands. Reading the *scripts page* is recommended before going further.

With the headless version, commands can be executed either by passing a script to run, or by setting the standard input as a script and writing commands to it, with `-s -` command line option, or using some named pipes.

Here's an example of bash code calling headless mode, which builds the master-bias and returns its noise to the console in red:

```
#!/bin/bash
# bash commands to prepare files

initdir=$(pwd)

######## Set your own variables #############
SCRIPTS_DIRECTORY=$initdir
SIRIL_PATH=siril-cli
#############################################

# Removing process folder if exists #
rm -rf $SCRIPTS_DIRECTORY/process

echo "Running siril bash script in $initdir"
output=$($SIRIL_PATH -d $SCRIPTS_DIRECTORY -s - <<ENDSIRIL

###########################################
#
# Example of script that create a master-bias
# and print in red the noise of the image
#
###########################################

requires 1.0.0

# Convert Bias Frames to .fit files
```

(continues on next page)

(continued from previous page)

```
cd biases
convert bias -out=../process
cd ../process

# Stack Bias Frames to bias_stacked.fit
stack bias rej 3 3 -nonorm -out=master-bias
cd ../..

close
ENDSIRIL
)

log_line=$(echo "$output" | grep -o "log: Background noise value.*")
echo -e "\e[31m$log_line\e[0m"

echo done Siril part
```

## 21.1 Command Stream (Pipe)

This mode has been introduced with Siril 0.9.10. Commands can be sent through a named pipe and logs and status can be obtained through another. The mode is activated using the `-p` command line argument.

The protocol is quite simple: Siril receives commands and outputs some updates. Only commands that are marked as scriptable can be used with this system. Whenever the command inputs pipe is closed or the cancel command is given, the running command is stopped as if the stop button was clicked on in the GUI. The pipes are named `siril_command.in` and `siril_command.out` and are available in `/tmp` on Unix-based systems. Since version 1.2.0, the paths of the pipes can be configured with `-r` and `-w` options, which allows external programs to create them before starting Siril, typically with the `mkfifo` command. Also new in 1.2.0, a `ping` command will simply give a status return, indicating if siril is ready to process a command or already busy.

Outputs of siril on the pipe is a stream of one line text and formatted as follows:

- `ready` is printed on startup, indicating siril is ready to process commands
- `log:` followed by a log message
- `status: verb [subject]`, where verb can be either of `starting`, `success`, `error` or `exit` (exit message is not yet implemented). The subject is the current command name, except for exit that indicates that siril suffered a fatal error and has to exit.
- `progress: value%` is the equivalent of the progress bar, it sends percents periodically, and sometimes 0% at the end of a processing as an idle information.

CHAPTER

TWENTYTWO

# PATH PARSING

Parsing is the ability to parse, i.e. to write strings based on data contained in the *FITS* header. Path parsing, introduced in Siril 1.2.0, aims at giving more flexibility to scripting by using header data to write/read filenames or paths. For now, this is used with the following commands:

- *calibrate*
- *stack* and *stackall*
- all the *saveXXX* commands

and of course, their GUI counterparts.

## 22.1 Syntax example

We will take an easy example to begin with. Say you have a file named `light_00001.fit` and you would like to locate a masterdark from your masters library that matches the characteristics of said light. As you have chosen a convention to name your masterdarks, you know that the correct dark should be named something like:

```
DARK_"exposure"s_G"gain"_O"offset"_T"temperature"C_bin"binning".fit
```

with the terms between quotes replaced with the values read from your light header. For an exposure of 120s, temperature of -10C, gain/offset of 120/30 and binning 1, the masterdark would be named:

```
DARK_120s_G120_O30_T-10C_bin1.fit
```

Well, that's exactly what this feature allows to do. If you specify the dark name with the conventions explained just after, you can tell Siril to open the light image, read its header and use its values to write such string (and then use it to calibrate your light).

You can read the info contained in the header either through *dumpheader* command or by using the Tools menu, *Tools → FITs Header...* ` You would normally get a print like the one below (some keys removed for brevity):

```
SIMPLE  =                    T / C# FITS
BITPIX  =                   16 /
NAXIS   =                    2 / Dimensionality
NAXIS1  =                 4144 /
NAXIS2  =                 2822 /
BZERO   =                32768 /
EXTEND  =                    T / Extensions are permitted
IMAGETYP= 'LIGHT'              / Type of exposure
EXPOSURE=                120.0 / [s] Exposure duration
DATE-OBS= '2022-01-24T01:03:34.729' / Time of observation (UTC)
XBINNING=                    1 / X axis binning factor
```

(continues on next page)

(continued from previous page)

```
YBINNING=                    1 / Y axis binning factor
GAIN    =                  120 / Sensor gain
OFFSET  =                   30 / Sensor gain offset
INSTRUME= 'ZWO ASI294MC Pro'   / Imaging instrument name
SET-TEMP=                -10.0 / [degC] CCD temperature setpoint
CCD-TEMP=                -10.0 / [degC] CCD temperature
BAYERPAT= 'RGGB'               / Sensor Bayer pattern
TELESCOP= '61EDPH'             / Name of telescope
FILTER  = 'DualBand'           / Active filter name
OBJECT  = 'Rosette Nebula     '/ Name of the object of interest
OBJCTRA = '06 30 36'           / [H M S] RA of imaged object
OBJCTDEC= '+04 58 51'          / [D M S] Declination of imaged object
RA      =     97.6960081674312 / [deg] RA of telescope
DEC     =     4.99212765957446 / [deg] Declination of telescope
END
```

The format used to specify the dark name would then be:

```
DARK_$EXPTIME:%d$s_G$GAIN:%d$_O$OFFSET:%d$_T$SET-TEMP:%d$C_bin$XBINNING:%d$.fit
```

All the terms to be parsed are formed as follows: $**KEY**:**fmt**$

- **KEY** is any (valid) key from the light FITS header
- **fmt** is a format specifier.

For instance, `$EXPTIME:%d$` will be parsed to `120` if the light have been exposed for 120s. But would be parsed to `120.0` if you specify `$EXPTIME:%0.1f$`, thanks to the formatter `%x.yf`.

The full expression above would therefore evaluate to:

```
DARK_**120**s_G**120**_O**30**_T**-10**C_bin**1**.fit
```

In this first example, we have only used conversion to integers with `%d`. But there are other conventional formatters that you can use:

- `%x.yf` for floats
- `%s` for strings

> **Note**
>
> For strings, leading and trailing spaces are always removed, while spaces within the strings are replaced with `_` signs. Example: `$OBJECT:%s$` would be converted to `Rosette_Nebula`.

You can also use some less conventional formatters:

- In order to parse a date from a date-time header key, you can use the special non-standard formatter **dm12**, which means date minus 12h. In the header above, the key DATE-OBS has a value of `2022-01-24T01:03:34.729`. `$DATE-OBS:dm12$` would convert to `2022-01-23`, which was the date at the start of the night. You can also use special formatter **dm0** which will just parse the date, without substracting 12h.
- In order to parse a date-time from a date-time header key, you can use the special non-standard formatter **dt**, which just means date-time. In the header above, the key `DATE-OBS` has a value of `2022-01-24T01:03:34.729`. `$DATE-OBS:dt$` would then convert to `2022-01-24_01-03-34`.

- In order to parse RA and DEC info from OBJCTRA and OBJCTDEC header keys, you can use the special non-standard formatters **ra** and **dec**. In the header above, the keys OBJCTRA and OBJCTDEC have a value of `06 30 36` and `+04 58 51` respectively. `$OBJCTRA:ra$_$OBJCTDEC:dec$` would convert to `06h30m36s_+04d58m51s`.
- In order to parse RA and DEC info from RA and DEC header keys, when in decimal format, you can use the special non-standard formatters **ran** and **decn**. In the header above, the keys RA and DEC have a value of `97.6960081674312` and `4.99212765957446` respectively. `$RA:ran$_$DEC:decn$` would convert to `06h30m47s_+04d59m32s`.

A good example for stacking result filename is to give the following expression:

```
$OBJECT:%s$_$FILTER:%s$_$STACKCNT:%d$x$EXPTIME:%d$sec_G$GAIN:%d$_O$OFFSET:%d$_T$CCD-TEMP:
↪%d$°C_$DATE-OBS:dm12$
```

giving something like:

```
NGC_7023_L_57x120sec_G100_O50_T-9°C_2023-10-07
```

To test the syntax, you can load an image and use the *parse* command, as reminded below.

**Siril command line**

```
parse str [-r]
```

Parses the string **str** using the information contained in the header of the image currently loaded. Main purpose of this command is to debug path parsing of header keys which can be used in other commands.

Option **-r** specifies the string is to be interpreted in read mode. In read mode, all wildcards defined in string **str** are used to find a file name matching the pattern. Otherwise, default mode is write mode and wildcards, if any, are removed from the string to be parsed.

If **str** starts with *$def* prefix, it will be recognized as a reserved keyword and looked for in the strings stored in gui_prepro.dark_lib, gui_prepro.flat_lib, gui_prepro.bias_lib or gui_prepro.stack_default for *$defdark*, *$defflat*, *$defbias* or *$defstack* respectively.

The keyword *$seqname$* can also be used when a sequence is loaded

## 22.2 Finding a file with path parsing

In the example above, we have seen that we could find the name of a masterdark based on the information contained in the header of the light to be calibrated. This is what is called in the `parse` command, the `read mode`.

This behavior is mainly used in conjuction with the *calibrate* command/tab and to find master distortion files. In the `-dark=` option of the command or in the `Dark` field of the GUI, you can use the syntax described above. So that you are sure that the matching dark will be fetched to calibrate the light. The same is equally applicable to `Bias` and `Flat`. You can, of course, also give a full (or relative) path to the file. And the path can also contain expressions of the same kind.

For instance, for flats, you may want to specify the path to a library, which could contain filter or telescope information, as you may have multiple setups. A path like:

```
~/astro/masters/flats/$INSTRUME:%s$_$TELESCOP:%s$/$FILTER:%s$/FLAT_bin$XBINNING:%d$.fit
```

Would then evaluate to:

```
~/astro/masters/flats/ZWO_ASI294MC_Pro_61EDPH/DualBand/FLAT_bin1.fit
```

and is a valid value for `Flat` input.

Of course, if you were to write this as a command in your scripts or in the GUI `Flat` field every time you calibrate, that could become a bit tedious. That's when reserved keywords come to the rescue. There are 3 reserved keywords for masters:

- `$defdark`
- `$defflat`
- `$defbias`
- `$defdisto`

Their values are set in *Preferences* → *Pre-processing section*. You can also specify them through scripting thanks to *set* command. They correspond to the values of `gui_prepro.dark_lib`, `gui_prepro.flat_lib`, `gui_prepro.bias_lib`. and `gui_prepro.disto_lib`.

When their values are set and you chose to use them as defaults, they will be displayed in the fields of the *Calibration* tab. You can also start to write your scripts using these keywords. The calibration step of a new generic script for a color camera could look like:

```
calibrate light -dark=$defdark -cc=dark -flat=$defflat -cfa -equalizecfa -debayer
```

This would pick your masters directly from your libraries and make sure you never mix them up during the calibration step.

## 22.3 Writing a file with path parsing

Now, while it is handy to be able to find the files, it would be equally useful to use this syntax to store your files while stacking. That is exactly what the `write mode` is about. The syntax can be used in the `-out=` field of *stack* and *stackall* commands, or in the corresponding field in the GUI.

Let's say you want to write a generic script that prepares your master darks each time you renew your library. In the `stack` line of the script, you could write:

```
stack dark rej 3 3 -nonorm -out=$defdark
```

This line ensures that the resulting masterdark will be stored at the right location with the correct filename that can then be fetched to calibrate your lights.

In order to introduce even more flexibility with the `stack` commands, there are two more reserved keywords:

- `$defstack`
- `$sequencename$`

As for default masters, `$defstack` is configured in the same section of the Preferences, or with a *set* command on `gui_prepro.stack_default`. For instance, let's assume you have defined `$defstack` as:

```
Result_$OBJECT:%s$_$DATE-OBS:dm12$_$LIVETIME:%d$s
```

The script line:

```
stack r_pp_light rej 3 3 -norm=addscale -output_norm -out=$defstack
```

would save the stack result as:

```
Result_Rosette_Nebula_2022-01-24_12000s.fit
```

As of Siril 1.2.0, the default name for stacking is defined as `$sequencename$stacked` (the _ sign is added if not present). This is similar to the behavior in previous versions, except it is now explicit that pathparsing is used.

## 22.4 Using wildcards

It could be that you want to use some key value in your masters name that do not match the key value in the frames to be calibrated. With an example, it may be a bit clearer. Say, you want, in your masterflats names, to keep record of their exposure time. Something like:

```
FLAT_1.32s_Halpha_G120_O30_bin1.fit
```

If you put a field $EXPTIME:%0.2f$ in `$defflat`, it will end up with an error at calibration step. Simply because the EXPTIME key will be read from the light frame to be calibrated, not a flat.

To deal with this situation, you can use wildcards in the expressions to be parsed:

```
FLAT_$*EXPTIME:%0.2f$_$FILTER:%s$_G$GAIN:%d$_O$OFFSET:%d$_bin$XBINNING:%d$
```

Note the * symbol placed right before `EXPTIME`.

What this symbol means is the following:

- In `write mode`, so basically when stacking your masterflat, the field `EXPOSURE` will be used to form the filename to be saved. In the example above, you would then effectively save to `FLAT_1.32s_Halpha_G120_O30_bin1.fit`.
- In `read mode`, so when calibrating your lights, the field `EXPOSURE` will be replaced by *. When searching for such file, Siril will fetch all the files that marches the pattern `FLAT_*_Halpha_G120_O30_bin1.fit`. Hopefully, your naming convention is robust enough so that it finds only one matching file and uses it to calibrate.

  This can also be useful for specifying master distortion files. You could, for instance , use:

```
~/astro/masters/distos/DISTO_$*DATE-LOC:dm12$_$INSTRUMEN:%s$_$TELESCOP:%s$_bin$XBINNING:
↪%d$.wcs
```

Note the * symbol placed right before `DATE-LOC`.

When a wildcard is used in front of a `DATE` header key, the returned file in readmode is the file with the closest date smaller or equal to the file which header is being parsed.

> **Warning**
>
> In the event Siril finds more than one file in `read mode`, it will emit a warning in the console and select the most recent file. As it may not produce the desired outcome, you should reconsider your naming convention if this happens.

## 22.5 Handling duplicates

In some cases, there are several keywords for the same value. This is because software developers are free to use the same keywords or create new ones. Siril therefore attempts to recognize and manage duplicates. Here's a table summarizing known duplicates. If a file contains so-called "alternative" keywords, then Siril will store the value in the "primary" version.

| Primary keyword | Alternative |
|---|---|
| MIPS-HI | CWHITE |
| MIPS-LO | CBLACK |
| PROGRAM | SWCREATE |
| IMAGETYP | FRAMETYP |
| EXPTIME | EXPOSURE |
| FILTER | FILT-1 |
| FOCALLEN | FOCAL |
| CENTALT | OBJCTALT |
| CENTAZ | OBJCTAZ |
| XBINNING | BINX |
| YBINNING | BINY |
| XPIXSZ | XPIXELSZ, PIXSIZE1, PIXSIZEX, XPIXSIZE |
| YPIXSZ | YPIXELSZ, PIXSIZE2, PIXSIZEY, YPIXSIZE |
| CCD-TEMP | CCD_TEMP, CCDTEMP, TEMPERAT, CAMTCCD |
| OFFSET | BLKLEVEL |
| CVF | EGAIN |
| FOCPOS | FOCUSPOS |
| FOCTEMP | FOCUSTEM |
| STACKCNT | NCOMBINE |
| SITELAT | SITE-LAT, OBSLAT |
| SITELONG | SITE-LON, OBSLONG |

CHAPTER

TWENTYTHREE

# LIVESTACKING

Live Stacking is a technique in astrophotography that allows the real-time stacking of a series of images to produce a higher quality image. Unlike traditional image stacking, which involves combining multiple images after they have been captured, live stacking combines the images as they are being captured. This can provide an instant preview of the final image and allow the astrophotographer to make adjustments in real-time to improve the quality of the final result.

Siril 1.2.0 includes this feature, which however is **experimental** for the moment. It allows to monitor a file in real time and to stack the images that arrive as they come in. Stacking of images can be done with and without Dark/Bias/Flats. The latter must be done beforehand if you want to use them.

## 23.1 Livestacking (GUI)

> **Note**
>
> Only FITS and RAW camera images are compatible with livestacking.

To start livestacking you need to :

- Define a working directory, with the home button, in which the photos will arrive one after the other.
- Click on the framed button on the image below.

A new window pops up.

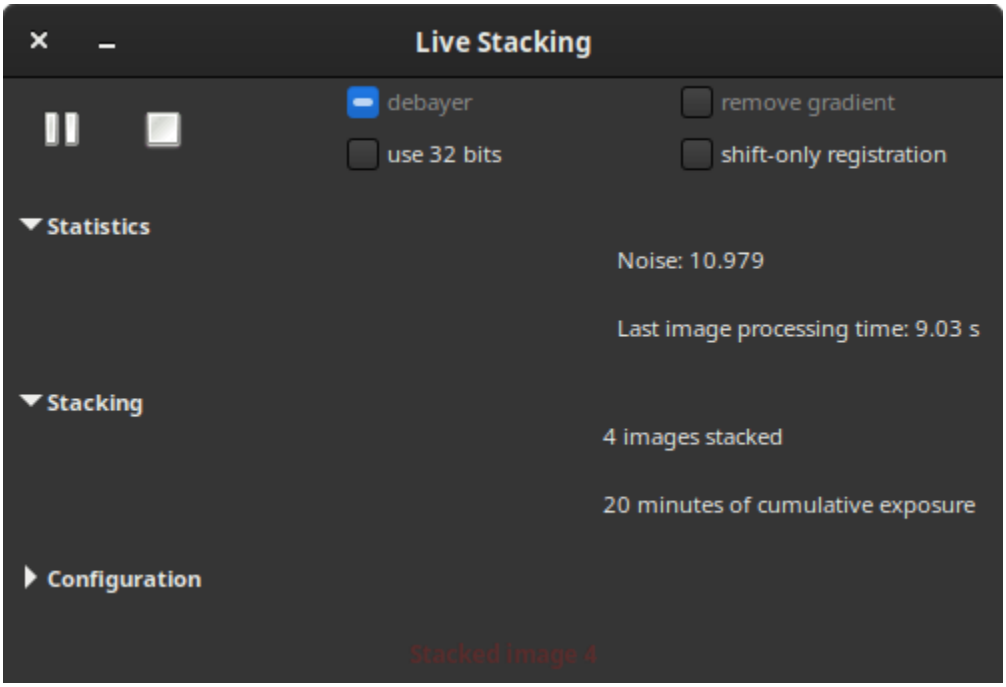
This window contains several buttons and options. A play button, which when clicked becomes a pause button, and a stop button. The first one allows to start or pause the monitoring of the working directory, and the last one to stop it.

All other options are fairly standard in the preprocessing of astronomical images:

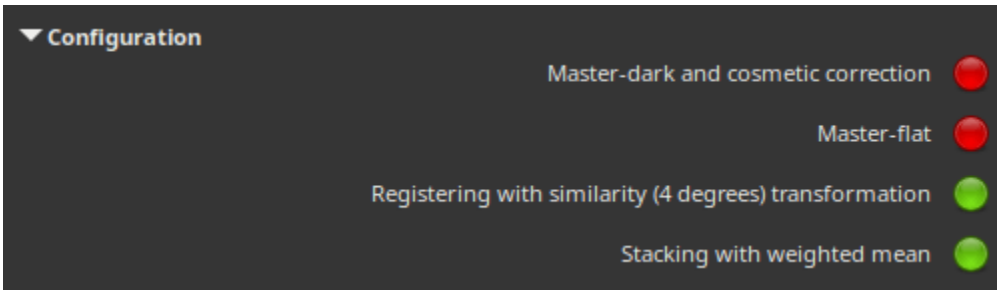
- **debayer**: The Bayer matrix is detected in files and debayer is automatically enabled. This is more of an indicator than an option.
- **use 32 bits**: Use 32 bits for image processing. This is slower and generally useless in terms of quality for live stacking.
- **remove gradient**: Apply a linear background gradient removal on calibrated input frames.
- **shift-only registration**: Only shift images instead of using rotation on registration. This should be unchecked for alt-az or heavily drifting mounts. This makes one image processing much faster.

The 3 sections below provide the information necessary for the user to follow the evolution of the stack.

- **Statistics**: This session gives the evolution of the noise level in ADU as well as the image processing time.




- **Stacking**: This section summarizes the number of images stacked and the cumulative exposure time.
- **Configuration** is a section that is not expanded by default. Once it is done, the frame gives details if the preprocessing is done using master files and the type of alignment and stacking.




> **Note**
>
> To use master files during a livestacking session, you must first have stacked your files. Then, once done and before launching the session, please load them in the *Calibration* tab. They will then be taken into account in the livestacking, and visible in the *Configuration* part of the livestack dialog.

## 23.2 Livestacking (Headless)

It is possible to use livestacking from the command line. For this, only 3 commands are necessary and explained below.

- The first, *start_ls* starts the livestacking session. It is possible to provide it with darks and flats images as arguments in order to calibrate the images during livestacking.

**Siril command line**

```
start_ls [-dark=filename] [-flat=filename] [-rotate] [-32bits]
```

Initializes a livestacking session, using the optional calibration files and waits for input files to be provided by the LIVESTACK command until STOP_LS is called. Default processing will use shift-only registration and 16-bit processing because it's faster, it can be changed to rotation with **-rotate** and **-32bits**

*Note that the live stacking commands put Siril in a state in which it's not able to process other commands. After START_LS, only LIVESTACK, STOP_LS and EXIT can be called until STOP_LS is called to return Siril in its normal, non-live-stacking, state*

Links: *livestack*, *stop_ls*, *exit*

- The *livestack* command is to be applied to each image you wish to stack.

**Siril command line**

```
livestack filename
```

Process the provided image for live stacking. Only possible after START_LS. The process involves calibrating the incoming file if configured in START_LS, demosaicing if it's an OSC image, registering and stacking. The temporary result will be in the file live_stack_00001.fit until a new option to change it is added

Links: *start_ls*

- Finally, the *stop_ls* command stops the livestacking session.

**Siril command line**

```
stop_ls
```

Stops the live stacking session. Only possible after START_LS

Links: *start_ls*

CHAPTER

# TWENTYFOUR

# COMMANDS

The following page lists all the commands available in Siril 1.4.4.

You can access an index by clicking this icon .

Commands marked with this icon can be used in scripts while those marked with this one cannot.

> **Tip**
>
> For all the sequence commands typed in the command bar of the GUI, you can replace argument **sequencename** with a `.` if the sequence to be processed is already loaded.

> **Tip**
>
> If you want to provide an argument that includes a string with spaces, for example a filename, you need to quote the entire argument not just the string. So for example you should use `command "-filename=My File.fits"`, **not** `command -filename="My File.fits"`.

addmax

```
addmax filename
```

Computes a new image combining the image in memory with the image **filename**. At each pixel location, the new value is determined as the max of value in current image and in **filename**

asinh

```
asinh [-human] stretch { [offset] [-clipmode=] }
```

Stretches the image to show faint objects using an hyperbolic arcsin transformation. The mandatory argument **stretch**, typically between 1 and 1000, will give the strength of the stretch. The black point can be offset by providing an optional **offset** argument in the normalized pixel value of [0, 1]. Finally the option **-human** enables using human

eye luminous efficiency weights to compute the luminance used to compute the stretch value for each pixel, instead of the simple mean of the channels pixel values. This stretch method preserves lightness from the L*a*b* color space. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

autoghs

```
autoghs [-linked] shadowsclip stretchamount [-b=] [-hp=] [-lp=] [-clipmode=]
```

Application of the generalized hyperbolic stretch with a symmetry point SP defined as k.sigma from the median of each channel (the provided **shadowsclip** value is the k here and can be negative). By default, SP and the stretch are computed per channel; SP can be computed as a mean of image channels by passing **-linked**. The stretch amount **D** is provided in the second mandatory argument.

Implicit values of 13 for **B**, making it very focused on the SP brightness range, 0.7 for **HP**, 0 for **LP** are used but can be changed with the options of the same names. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

autostretch

```
autostretch [-linked] [shadowsclip [targetbg]]
```

Auto-stretches the currently loaded image, with different parameters for each channel (unlinked) unless **-linked** is passed. Arguments are optional, **shadowclip** is the shadows clipping point, measured in sigma units from the main histogram peak (default is -2.8), **targetbg** is the target background value, giving a final brightness to the image, range [0, 1], default is 0.25. The default values are those used in the Auto-stretch rendering from the GUI.

Do not use the unlinked version after color calibration, it will alter the white balance

bg

```
bg
```

Returns the background level of the loaded image

bgnoise

```
bgnoise
```

Returns the background noise level of the loaded image

For more information, see the *statistics documentation*

binxy

```
binxy coefficient [-sum]
```

Computes the numerical binning of the in-memory image (sum of the pixels 2x2, 3x3..., like the analogic binning of CCD camera). If the optional argument **-sum** is passed, then the sum of pixels is computed, while it is the average when no optional argument is provided

boxselect

```
boxselect [-clear] [x y width height]
```

Make a selection area in the currently loaded image with the arguments **x**, **y**, **width** and **height**, with **x** and **y** being the coordinates of the top left corner starting at (0, 0), and **width** and **height**, the size of the selection. The **-clear** argument deletes any selection area. If no argument is passed, the current selection is printed

calibrate

```
calibrate sequencename [-bias=filename] [-dark=filename] [-flat=filename] [-cc=dark␣
↪[siglo sighi] || -cc=bpm bpmfile] [-cfa] [-debayer] [-fix_xtrans] [-equalize_cfa] [-
↪opt[=exp]] [-all] [-prefix=] [-fitseq]
```

Calibrates the sequence **sequencename** using bias, dark and flat given in argument.

For bias, a uniform level can be specified instead of an image, by entering a quoted expression starting with an = sign, such as -bias="=256" or -bias="=64*$OFFSET".

By default, cosmetic correction is not activated. If you wish to apply some, you will need to specify it with **-cc=** option.

You can use **-cc=dark** to detect hot and cold pixels from the masterdark (a masterdark must be given with the **-dark=** option), optionally followed by **siglo** and **sighi** for cold and hot pixels respectively. A value of 0 deactivates the correction. If sigmas are not provided, only hot pixels detection with a sigma of 3 will be applied.

Alternatively, you can use **-cc=bpm** followed by the path to your Bad Pixel Map to specify which pixels must be corrected. An example file can be obtained with a *find_hot* command on a masterdark.

Three options apply to color images (in CFA format): **-cfa** for cosmetic correction purposes, **-debayer** to demosaic images before saving them, and **-equalize_cfa** to equalize the mean intensity of RGB layers of the master flat, to avoid tinting the calibrated image.

The **-fix_xtrans** option is dedicated to X-Trans images by applying a correction on darks and biases to remove a rectangle pattern caused by autofocus.

It's also possible to optimize dark subtraction with **-opt**, which requires the supply of bias and dark masters, and automatically calculates the coefficient to be applied to dark, or calculates the coefficient thanks to the exposure keyword with **-opt=exp**.

By default, frames marked as excluded will not be processed. The argument **-all** can be used to force processing of all frames even if marked as excluded.

The output sequence name starts with the prefix "pp_" unless otherwise specified with option **-prefix=**.

If **-fitseq** is provided, the output sequence will be a FITS sequence (single file)

calibrate_single

```
calibrate_single imagename [-bias=filename] [-dark=filename] [-flat=filename] [-cc=dark␣
↪[siglo sighi] || -cc=bpm bpmfile] [-cfa] [-debayer] [-fix_xtrans] [-equalize_cfa] [-
↪opt[=exp]] [-prefix=]
```

Calibrates the image **imagename** using bias, dark and flat given in argument.

For bias, a uniform level can be specified instead of an image, by entering a quoted expression starting with an = sign, such as -bias="=256" or -bias="=64*$OFFSET".

By default, cosmetic correction is not activated. If you wish to apply some, you will need to specify it with **-cc=** option.

You can use **-cc=dark** to detect hot and cold pixels from the masterdark (a masterdark must be given with the **-dark=** option), optionally followed by **siglo** and **sighi** for cold and hot pixels respectively. A value of 0 deactivates the correction. If sigmas are not provided, only hot pixels detection with a sigma of 3 will be applied.

Alternatively, you can use **-cc=bpm** followed by the path to your Bad Pixel Map to specify which pixels must be corrected. An example file can be obtained with a *find_hot* command on a masterdark.

Three options apply to color images (in CFA format): **-cfa** for cosmetic correction purposes, **-debayer** to demosaic images before saving them, and **-equalize_cfa** to equalize the mean intensity of RGB layers of the master flat, to avoid tinting the calibrated image.

The **-fix_xtrans** option is dedicated to X-Trans images by applying a correction on darks and biases to remove a

rectangle pattern caused by autofocus.
It's also possible to optimize dark subtraction with **-opt**, which requires the supply of bias and dark masters, and automatically calculates the coefficient to be applied to dark, or calculates the coefficient thanks to the exposure keyword with **-opt=exp**
The output filename starts with the prefix "pp_" unless otherwise specified with option **-prefix=**

capabilities

```
capabilities
```

Lists Siril capabilities, based on compilation options and runtime

catsearch

```
catsearch name
```

Searches an object by **name** and adds it to the user annotation catalog. The object is first searched in the annotation catalogs, if not found a request is made to SIMBAD.
The object can be a solar system object, in which case a prefix, 'a:' for asteroid, 'p:' for planet, 'c:' for comet, 'dp:' for dwarf planet or 's:' for natural satellite, is required before the object name. The search is done for the date, time and observing location found in the image header, using the IMCCE Miriade service

ccm

```
ccm m00 m01 m02 m10 m11 m12 m20 m21 m22 [gamma]
```

Applies a color conversion matrix to the current image.

There are 9 mandatory arguments corresponding to the 9 matrix elements:

m00, m01, m02
m10, m11, m12
m20, m21, m22

An additional tenth argument **[gamma]** can be provided: if it is omitted, it defaults to 1.0.

These are applied to each pixel according to the following formulae:

r' = (m00 * r + m01 * g + m02 * b)^(-1/gamma)
g' = (m10 * r + m11 * g + m12 * b)^(-1/gamma)
b' = (m20 * r + m21 * g + m22 * b)^(-1/gamma)

## cd

```
cd directory
```

Sets the new current working directory.

The argument **directory** can contain the ~ token, expanded as the home directory, directories with spaces in the name can be protected using single or double quotes

## cdg

```
cdg
```

Returns the coordinates of the center of gravity of the image. Only pixels with values above 15.7% of max ADU and having four neighbors filling the same condition are used to compute it, and it is computed only if there are at least 50 of them

## clahe

```
clahe cliplimit tileSize
```

Equalizes the histogram of an image using Contrast Limited Adaptive Histogram Equalization.

**cliplimit** sets the threshold for contrast limiting.
**tilesize** sets the size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles

clear

```
clear
```

Clears the graphical output logs

clearstar

```
clearstar
```

Clears all the stars saved in memory and displayed on the screen

close

```
close
```

Properly closes the opened image and the opened sequence, if any

conesearch

```
conesearch [limit_magnitude] [-cat=] [-phot] [-obscode=] [-tag={on|off}] [-log={on|off}]␣
↪[-trix=] [-out=]
```

Displays stars from the local catalog by default for the loaded plate solved image, down to the provided **limit_magnitude** (13 by default for most catalogues, except 14.5 for aavso_chart, 20 for solsys, and omitted for pgc).
An alternate online catalog can be specified with **-cat=**, taking values
- for stars: tycho2, nomad, gaia, localgaia, ppmxl, bsc, apass, gcvs, vsx, simbad, aavso_chart
- for exoplanets: exo
- for deep-sky: pgc
- for solar system objects: solsys (closest IAU observatory code can be passed with the argument **-obscode=** for better position accuracy)

For stars catalogues containing photometric data, stars with no B-V information will be kept; they can be excluded by passing **-phot**

The argument **-trix=** can be passed instead of a catalogue followed by a number between 0 and 511 to plot stars contained in local catalogues trixel of level 3 (for dev usage mainly)

Some catalogs (bsc, gcvs, pgc, exo, aavso_chart, varisum and solsys) will also display, by default, names alongside markers in the display (GUI only) and list them in the log. For others with larger number of objects, namely vsx and simbad, the information can also be shown but, as it may clutter the display, it is not activated by default. This behavior can be toggled on/off with the options **-tag=on|off** to display names alongside markers and **-log=on|off** to list the objects in the console log

The list of items that are present in the image can optionally saved to a csv file by passing the argument **-out=**

convert

```
convert basename [-debayer] [-fitseq] [-ser] [-start=index] [-out=]
```

Converts all images of the current working directory that are in a supported format into Siril's sequence of FITS images (several files) or a FITS sequence (single file) if **-fitseq** is provided or a SER sequence (single file) if **-ser** is provided. The argument **basename** is the base name of the new sequence, numbers and the extension will be put behind it.
For FITS images, Siril will try to make a symbolic link; if not possible, files will be copied. The option **-debayer** applies demosaicing to CFA input images; in this case no symbolic link is done.
**-start=index** sets the starting index number, useful to continue an existing sequence (not used with -fitseq or **-ser**; make sure you remove or clear the target .seq if it exists in that case).
The **-out=** option changes the output directory to the provided argument.

See also CONVERTRAW and LINK

Links: *convertraw*, *link*

convertraw

```
convertraw basename [-debayer] [-fitseq] [-ser] [-start=index] [-out=]
```

Same as CONVERT but converts only DSLR RAW files found in the current working directory

Links: *convert*

cosme

```
cosme [filename].lst
```

Applies the local mean to a set of pixels on the loaded image (cosmetic correction). The coordinates of these pixels are in a text file [.lst file], the FIND_HOT command can also create it for single hot pixels, but manual operation is needed to remove rows or columns. COSME is adapted to correct residual hot and cold pixels after calibration.

Instead of providing the list of bad pixels, it's also possible to detect them in the current image using the FIND_COSME command

Links: *find_hot*, *find_cosme*

File format for the bad pixels list: * Lines in the form *P x y* will fix the pixel at coordinates (x, y) type is an optional character (C or H) specifying to Siril if the current pixel is cold or hot. This line is created by the command FIND_HOT but you also can add the two following line types manually * Lines in the form *C x 0* will fix the bad column at coordinates x. * Lines in the form *L y 0* will fix the bad line at coordinates y.

cosme_cfa

```
cosme_cfa [filename].lst
```

Same function as COSME but applying to RAW CFA images

Links: *cosme*

crop

```
crop [x y width height]
```

Crops to a selected area of the loaded image.

If a selection is active, no further arguments are required. Otherwise, or in scripts, arguments have to be given, with **x** and **y** being the coordinates of the top left corner, and **width** and **height** the size of the selection. Alternatively, the selection can be made using the BOXSELECT command

Links: *boxselect*

### ddp

```
ddp level coef sigma
```

Performs a DDP (digital development processing) on the loaded image, as described first by Kunihiko Okano. This implementation is the one described in IRIS.

It combines a linear distribution on low levels (below **level**) and a non-linear one on high levels.
It uses a Gaussian filter of standard deviation **sigma** and multiplies the resulting image by **coef**. Typical values for **sigma** are within 0.7 and 2. The level argument should be in the range [0, 65535] for 16-bit images and may be given either in the range [0, 1] or [0, 65535] for 32-bit images in which case it will be scaled automatically

### denoise

```
denoise [-nocosmetic] [-mod=m] [ -vst | -da3d | -sos=n [-rho=r] ] [-indep]
```

Denoises the image using the non-local Bayesian algorithm described by Lebrun, Buades and Morel.

It is strongly recommended to apply cosmetic correction to remove salt and pepper noise before running denoise, and by default this command will apply cosmetic correction automatically. However, if this has already been carried out earlier in the workflow it may be disabled here using the optional command **-nocosmetic**.

An optional argument **-mod=m** may be given, where 0 <= m <= 1. The output pixel is computed as : *out=m x d + (1 m) x in*, where *d* is the denoised pixel value. A modulation value of 1 will apply no modulation. If the parameter is omitted, it defaults to 1.

The optional argument **-vst** can be used to apply the generalised Anscombe variance stabilising transform prior to NL-Bayes. This is useful with photon-starved images such as single subs, where the noise follows a Poisson or Poisson-Gaussian distribution rather than being primarily Gaussian. It cannot be used in conjunction with DA3D or SOS, and for denoising stacked images it is usually not beneficial.

The optional argument **-da3d** can be used to enable Data-Adaptive Dual Domain Denoising (DA3D) as a final stage denoising algorithm. This uses the output of BM3D as a guide image to refine the denoising. It improves detail and reduces staircasing artefacts.

The optional argument **-sos=n** can be used to enable Strengthen-Operate-Subtract (SOS) iterative denoise boosting, with the number of iterations specified by n. In particular, this booster may produce better results if the un-boosted NL-Bayes algorithm produces artefacts in background areas. If both -da3d and -sos=n are specified, the last to be specified will apply.

The optional argument **-rho=r** may be specified, where 0 < r < 1. This is used by the SOS booster to determine the amount of noisy image added in to the intermediate result between each iteration. If -sos=n is not specified then the parameter is ignored.

The default is not to apply DA3D or SOS, as the improvement in denoising is usually relatively small and these techniques requires additional processing time.

In very rare cases, blocky coloured artefacts may be found in the output when denoising colour images. The optional argument **-indep** can be used to prevent this by denoising each channel separately. This is slower but will eliminate artefacts

### dir

```
dir
```

Lists files and directories in the working directory

This command is only available on Microsoft Windows, for the equivalent command on Linux and MacOS, see *ls*.

### disto

```
disto [clear]
```

Shows distortion field on a plate-solved image which solution includes distortion terms

Pass option **clear** to disable

### dumpheader

```
dumpheader
```

Dumps the FITS header of the loaded image in the console

### entropy

```
entropy
```

Computes the entropy of the loaded image on the displayed layer, only in the selected area if one has been selected or in the whole image. The entropy is one way of measuring the noise or the details in an image

epf

```
epf [-guided] [-d=] [-si=] [-ss=] [-mod=] [-guideimage=]
```

Applies an edge preserving filter. By default a bilateral filter is applied; a guided filter can be specified using the argument **-guided**. The filter diameter defaults to 3 and can be set using **-d=**. Be careful with values of d greater than 20 as the algorithm can be computationally expensive.

The intensity filtering sigma value can be set using **-si=** and the spatial sigma value can be set using **-ss=**. Sigma values represent the difference in pixel values over which the filter acts strongly: for 32-bit images the value should be between 0 and 1.0, whereas for 16-bit images it should be between 0 and 65535. The defaults if not specified are for both to be set to 11. If **-d=0** is set then the filter diameter will be set automatically based on the value of **-ss**. *Note that when applying a guided filter, only* **-sc** *applies.*

When specifying a guided filter, a guide image may be set using **-guideimage=**. The default if no guide image is specified is to perform a self-guided filter. *Note: the guide image must have the same dimensions as the image to be filtered!*

The strength of the filter can be modulated using the **-mod=** argument. If mod = 1.0 the full effect of the filter will be applied; for mod less than 1.0 a proportion of the original image will be mixed with the result, and for mod = 0.0 no filtering will be applied

exit

```
exit
```

Quits the application

extract

```
extract NbPlans
```

Extracts **NbPlans** planes of wavelet domain of the loaded image.
See also WAVELET and WRECONS. For color extraction, see SPLIT

Links: *wavelet*, *wrecons*, *split*

extract_Green

```
extract_Green
```

Extracts green signal from the loaded CFA image. It reads the Bayer matrix information from the image or the preferences and exports only the averaged green filter data as a new half-sized FITS file. A new file is created, its name is prefixed with "Green_"

extract_Ha

```
extract_Ha [-upscale]
```

Extracts H-Alpha signal from the loaded CFA image. It reads the Bayer matrix information from the image or the preferences and exports only the red filter data as a new half-sized FITS file. If the argument **-upscale** is provided, the output will be upscaled x2 to match the full sensor resolution, for example to match other images produced by the same family of sensors. A new file is created, its name is prefixed with "Ha_"

extract_HaOIII

```
extract_HaOIII [-resample=]
```

Extracts H-Alpha and O-III signals from the loaded CFA image. It reads the Bayer matrix information from the image or the preferences and exports only the red filter data for H-Alpha as a new half-sized FITS file (like EXTRACTHA) and keeps the three others for O-III with an interpolated replacement for the red pixel. The output files names start with the prefix "Ha_" and "OIII_"

The optional argument **-resample={ha|oiii}** sets whether to upsample the Ha image or downsample the OIII image to have images the same size. If this argument is not provided, no resampling will be carried out and the OIII image will have twice the height and width of the Ha image

fdiv

```
fdiv filename scalar
```

Divides the loaded image by the image given in argument. The resulting image is multiplied by the value of the **scalar** argument. See also IDIV

Links: *idiv*

ffill

```
ffill value [x y width height]
```

Same command as FILL but this is a symmetric fill of a region defined by the mouse or with BOXSELECT. Used to process an image in the Fourier (FFT) domain

Links: *fill*, *boxselect*

fftd

```
fftd modulus phase
```

Applies a Fast Fourier Transform to the loaded image. **modulus** and **phase** given in argument are the names of the saved in FITS files

ffti

```
ffti modulus phase
```

Retrieves corrected image applying an inverse transformation. The **modulus** and **phase** arguments are the input file names, the result will be the new loaded image

fill

```
fill value [x y width height]
```

Fills the loaded image entirely or only the selection if there is one with pixels having the **value** intensity expressed in ADU

find_cosme

```
find_cosme cold_sigma hot_sigma
```

Applies an automatic detection and replacement of cold and hot pixels in the loaded image, with the thresholds passed in arguments in sigma units

find_cosme_cfa

```
find_cosme_cfa cold_sigma hot_sigma
```

Same command as FIND_COSME but for CFA images

Links: *find_cosme*

find_hot

```
find_hot filename cold_sigma hot_sigma
```

Saves a list file **filename** (text format) in the working directory which contains the coordinates of the pixels which have an intensity **hot_sigma** times higher and **cold_sigma** lower than standard deviation, extracted from the loaded image. We generally use this command on a master-dark file. The COSME command can apply this list of bad pixels to a loaded image, see also SEQCOSME to apply it to a sequence

Links: *cosme*, *seqcosme*

Lines `P x y type` will fix the pixel at coordinates (x, y) type is an optional character (C or H) specifying to Siril if the current pixel is cold or hot. This line is created by the command FIND_HOT but you also can add some lines manually:
Lines `C x 0 type` will fix the bad column at coordinates x.
Lines `L y 0 type` will fix the bad line at coordinates y.

findcompstars

```
findcompstars star_name [-narrow|-wide] [-catalog={nomad|apass}] [-dvmag=3] [-dbv=0.5] [-
↪emag=0.03] [-out=nina_file.csv]
```

Automatically finds comparison stars in the field of the plate solved loaded image, for photometric analysis of a star's light curve according to
- the provided name of the star
- the field of view of the image, reduced to a diameter of its height if **-narrow** is passed, avoiding stars in the corners
- the chosen catalog (APASS by default), can be changed with **-catalog={NOMAD|APASS}**
- the difference in visual magnitude from the variable star, in the range [0, 6] with a default of 3, changed with **-dvmag=**
- the difference in color with the variable star, in the range [0.0, 0.7] of their B-V indices with a default of 0.5, changed with **-dbv=**
- the maximum allowed error on Vmag in the range [0.0, 0.1] with a default of 0.03, changed with **-emag=**.

The list can optionally be saved as a CSV file compatible with the NINA comparison stars list, specifying the file name with **-out=**. If the provided name is the special value **auto**, it is generated using the input parameters

See also LIGHT_CURVE

Links: *light_curve*

findstar

```
findstar [-out=] [-layer=] [-maxstars=]
```

Detects stars in the currently loaded image, having a level greater than a threshold computed by Siril.
After that, a PSF is applied and Siril rejects all detected structures that don't fulfill a set of prescribed detection criteria, that can be tuned with command SETFINDSTAR.
Finally, an ellipse is drawn around detected stars.

Optional parameter **-out=** allows the results to be saved to the given path.
Option **-layer=** specifies the layer onto which the detection is performed (for color images only).
You can also limit the maximum number of stars detected by passing a value to option **-maxstars=**.

See also CLEARSTAR

Links: *psf*, *setfindstar*, *clearstar*

fix_xtrans

```
fix_xtrans
```

Fixes the Fujifilm X-Trans Auto Focus pixels in the loaded image.

Indeed, because of the phase detection auto focus system, the photosites used for auto focus get a little less light than the surrounding photosites. The camera compensates for this and increases the values from these specific photosites giving a visible square in the middle of the dark/bias frames

fixbanding

```
fixbanding amount sigma [-vertical]
```

Tries to remove the horizontal or vertical banding in the loaded image.
**amount** defines the amount of correction, between 0 and 4.
**sigma** defines the highlight protection level of the algorithm, higher sigma gives higher protection, between 0 and 5. Values of 1 and 1 are often good enough.
**-vertical** option enables to perform vertical banding removal, horizontal is the default

fmedian

```
fmedian ksize modulation
```

Performs a median filter of size **ksize** x **ksize** (**ksize** MUST be odd) to the loaded image with a modulation parameter **modulation**.

The output pixel is computed as : out=mod x m + (1  mod) x in, where m is the median-filtered pixel value. A modulation's value of 1 will apply no modulation

fmul

```
fmul scalar
```

Multiplies the loaded image by the **scalar** given in argument

gauss

```
gauss sigma
```

Applies to the loaded image a Gaussian blur with the given **sigma**.

See also UNSHARP, the same with a blending parameter

Links: *unsharp*

get

```
get { -a | -A | variable }
```

Gets a value from the settings using its name, or list all with **-a** (name and value list) or with **-A** (detailed list)

See also SET to update values

Links: *set*

getref

```
getref sequencename
```

Prints information about the reference image of the sequence given in argument. First image has index 0

ght

```
ght -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -sat]␣
↪[channels]
```

Generalised hyperbolic stretch based on the work of the ghsastro.co.uk team.

The argument **-D=** defines the strength of the stretch, between 0 and 10. This is the only mandatory argument. The following optional arguments further tailor the stretch:
**B** defines the intensity of the stretch near the focal point, between -5 and 15;
**LP** defines a shadow preserving range between 0 and SP where the stretch will be linear, preserving shadow detail;
**SP** defines the symmetry point of the stretch, between 0 and 1, which is the point at which the stretch will be most intense;
**HP** defines a region between HP and 1 where the stretch is linear, preserving highlight details and preventing star bloat.
If omitted B, LP and SP default to 0.0 ad HP defaults to 1.0.
An optional argument (either **-human**, **-even** or **-independent**) can be passed to select either human-weighted or even-weighted luminance or independent colour channels for colour stretches. The argument is ignored for mono images. Alternatively, the argument **-sat** specifies that the stretch is performed on image saturation - the image must be color and all channels must be selected for this to work.
Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

grey_flat

```
grey_flat
```

Equalizes the mean intensity of RGB layers in the loaded CFA image. This is the same process used on flats during calibration when the option equalize CFA is used

help

```
help [command]
```

Lists the available commands or help for one command

histo

```
histo channel (channel=0, 1, 2 with 0: red, 1: green, 2: blue)
```

Calculates the histogram of the **layer** of the loaded image and produces file histo_[channel name].dat in the working directory.
layer = 0, 1 or 2 with 0=red, 1=green and 2=blue

iadd

```
iadd filename
```

Adds the image **filename** to the loaded image.
Result will be in 32 bits per channel if allowed in the preferences

icc_assign

```
icc_assign profile
```

Assigns the ICC profile specified in the argument to the current image.

One of the following special arguments may be provided to use the respective built-in profiles: **sRGB**, **sRGBlinear**, **Rec2020**, **Rec2020linear**, **working** to set the working mono or RGB color profile, (for mono images only) **linear**, or the path to an ICC profile file may be provided. If a built-in profile is specified with a monochrome image loaded, the Gray profile with the corresponding TRC will be used

icc_convert_to

```
icc_convert_to profile [intent]
```

Converts the current image to the specified ICC profile.

One of the following special arguments may be provided to use the respective built-in profiles: **sRGB**, **sRGBlinear**, **Rec2020**, **Rec2020linear**, **graysrgb**, **grayrec2020**, **graylinear** or **working** to set the working mono or RGB color profile, (for mono images only) **linear**, or the path to an ICC profile file may be provided. If a built-in profile is specified with a monochrome image loaded, the Gray profile with the corresponding TRC will be used.

A second argument may be provided to specify the color transform intent: this should be one of **perceptual**, **relative** (for relative colorimetric), **saturation** or **absolute** (for absolute colorimetric)

icc_remove

```
icc_remove
```

Removes the ICC profile from the current image, if it has one

idiv

```
idiv filename
```

Divides the loaded image by the image **filename**.

Result will be in 32 bits per channel if allowed in the preferences.

See also FDIV

Links: *fdiv*

imul

```
imul filename
```

Multiplies image **filename** by the loaded image.
Result will be in 32 bits per channel if allowed in the preferences

inspector

```
inspector
```

Splits the loaded image in a nine-panel mosaic showing the image corners and the center for a closer inspection (GUI only)

invght

```
invght -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -
↪sat] [channels]
```

Inverts a generalised hyperbolic stretch. It provides the inverse transformation of GHT, if provided with the same parameters, undoes a GHT command, possibly returning to a linear image. It can also work the same way as GHT but for images in negative

Links: *ght*

invmodasinh

```
invmodasinh -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -sat]␣
↪[channels]
```

Inverts a modified arcsinh stretch. It provides the inverse transformation of MODASINH, if provided with the same parameters, undoes a MODASINH command, possibly returning to a linear image. It can also work the same way as MODASINH but for images in negative

Links: *modasinh*

invmtf

```
invmtf low mid high [channels]
```

Inverts a midtones transfer function. It provides the inverse transformation of MTF, if provided with the same parameters, undoes a MTF command, possibly returning to a linear image. It can also work the same way as MTF but for images in negative

Links: *mtf*

isub

```
isub filename
```

Subtracts the loaded image by the image **filename**.

Result will be in 32 bits per channel if allowed in the preferences, so capable of storing negative values. To clip negative value, use 16 bit mode or use the THRESHLO command

Links: *threshlo*

jsonmetadata

```
jsonmetadata FITS_file [-stats_from_loaded] [-nostats] [-out=]
```

Dumps metadata and statistics of the currently loaded image in JSON form. The file name is required, even if the image is already loaded. Image data may not be read from the file if it is the current loaded image and if the **-stats_from_loaded** option is passed. Statistics can be disabled by providing the **-nostats** option. A file containing the JSON data is created with default file name '$(FITS_file_without_ext).json' and can be changed with the **-out=** option

light_curve

```
light_curve sequencename channel [-autoring] { -at=x,y | -wcs=ra,dec } { -refat=x,y | -
↪refwcs=ra,dec } ...
light_curve sequencename channel [-autoring] -ninastars=file
```

Analyses several stars with aperture photometry in a sequence of images and produces a light curve for one, calibrated by the others. The first coordinates, in pixels if **-at=** is used or in degrees if **-wcs=** is used, are for the star whose light will be plotted, the others for the comparison stars.

Alternatively, a list of target and reference stars can be passed in the format of the NINA exoplanet plugin star list, with the **-ninastars=** option. Siril will verify that all reference stars can be used before actually using them. A data file is created in the current directory named light_curve.dat, Siril plots the result to a PNG image if available

The ring radii for the annulus can either be configured in the settings or set to a factor of the reference image's FWHM if **-autoring** is passed. These autoring sizes are 4.2 time and 6.3 times the FWHM for the inner and outer radii, respectively.

See also the **setphot** command to set the same way the aperture radius size.

See also SEQPSF for operations on single star

Links: *seqpsf*

limit

```
limit { -clip | -posrescale | -rescale }
```

Limits pixel values in 32-bit images to the range 0.0 to 1.0. This command does not apply to 16-bit images as there cannot be out-of-range values. Range limiting can be done in one of the following ways:

**-clip**: this option simply clips all negative pixels to 0.0 and all pixels with a value > 1.0 to 1.0.

**-posrescale**: this option scales all positive pixel values so that the maximum value is 1.0, clipping any negative pixels to 0.0. For 3-channel images the same scaling factor is applied to all channels. If the maximum pixel value is already <= 1.0 negative pixels will still be clipped but no scaling factor will be applied to positive pixels.

**-rescale**: using this option, if there are any negative pixel values the image will have a constant value added to all pixel values so that the minimum value is 0.0. Then if the maximum pixel value is > 1.0, a scaling factor is applied so that the maximum pixel value is scaled to 1.0.

Note that if there are one or more extreme outliers (for example as a result of bad pixels) the **-rescale** and **-posrescale** options may produce an unexpected result. This can be mitigated by applying cosmetic correction to the image first

linear_match

```
linear_match reference low high
```

Computes and applies a linear function between a **reference** image and the loaded image.

The algorithm will ignore all reference pixels whose values are outside of the [**low**, **high**] range

link

```
link basename [-date] [-start=index] [-out=]
```

Same as CONVERT but converts only FITS files found in the current working directory. This is useful to avoid conversions of JPEG results or other files that may end up in the directory. The additional argument **-date** enables sorting files with their DATE-OBS value instead of with their name alphanumerically

Links: *convert*

linstretch

```
linstretch -BP= [-sat] [-clipmode=] [channels] [-clipmode=]
```

Stretches the image linearly to a new black point BP.

The argument **[channels]** may optionally be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels.

Optionally the parameter **-sat** may be used to apply the linear stretch to the image saturation channel. This argument only works if all channels are selected. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

livestack

```
livestack filename
```

Process the provided image for live stacking. Only possible after START_LS. The process involves calibrating the incoming file if configured in START_LS, demosaicing if it's an OSC image, registering and stacking. The temporary result will be in the file live_stack_00001.fit until a new option to change it is added

Links: *start_ls*

> **Warning**
>
> Note that the live stacking commands put Siril in a state in which it's not able to process other commands. After START_LS, only LIVESTACK, STOP_LS and EXIT can be called until STOP_LS is called to return Siril in its normal, non-live-stacking, state.

load

```
load filename[.ext]
```

Loads the image **filename** from the current working directory, which becomes the 'currently loaded image' used in many of the single-image commands.
It first attempts to load **filename**, then **filename**.fit, **filename**.fits and finally all supported formats.
This scheme is applicable to every Siril command that involves reading files

load_seq

```
load_seq sequencename[.ext]
```

Loads the sequence **sequencename** from the current working directory, which becomes the 'currently loaded sequence'. While Siril sequence commands require the sequence name to be specified, it is necessary to load a sequence in order to read its metadata using Python scripting

log

```
log
```

Computes and applies a logarithmic scale to the loaded image, using the following formula: log(1 - (value - min) / (max - min)), with min and max being the minimum and maximum pixel value for the channel

ls

```
ls
```

Lists files and directories in the working directory

This command is only available on Unix-like Systems, for the equivalent command on Microsoft Windows, see *dir*.

makepsf

```
makepsf clear
makepsf load filename
makepsf save [filename]
makepsf blind [-l0] [-si] [-multiscale] [-lambda=] [-comp=] [-ks=] [-savepsf=]
makepsf stars [-sym] [-ks=] [-savepsf=]
makepsf manual { -gaussian | -moffat | -disc | -airy } [-fwhm=] [-angle=] [-ratio=] [-
↪beta=] [-dia=] [-fl=] [-wl=] [-pixelsize=] [-obstruct=] [-ks=] [-savepsf=]
```

Generates a PSF for use with deconvolution, any of the three methods exposed by RL, SB or WIENER commands. One of the following must be given as the first argument: **clear** (clears the existing PSF), **load** (loads a PSF from a file), **save** (saves the current PSF), **blind** (blind estimate of tke PSF), **stars** (generates a PSF based on measured stars from the image) or **manual** (generates a PSF manually based on a function and parameters).

No additional arguments are required when using the **clear** argument.

To load a previously saved PSF the **load** argument requires the PSF *filename* as a second argument. This may be in any format that Siril has been compiled with support for, but it must be square and should ideally be odd.

To save a previously generated PSF the argument **save** is used. Optionally, a filename may be provided (this must have one of the extensions ".fit", ".fits", ".fts" or ".tif") but if none is provided the PSF will be named based on the name of the open file or sequence.

For **blind**, the following optional arguments may be provided: **-l0** uses the l0 descent method, **-si** uses the spectral irregularity method, **-multiscale** configures the l0 method to do a multi-scale PSF estimate, **-lambda=** provides the regularization constant.

For PSF from detected **stars** the only optional parameter is **-sym**, which configures the PSF to be symmetric.

For a **manual** PSF, one of **-gaussian**, **-moffat**, **-disc** or **-airy** can be provided to specify the PSF function, Gaussian by default. For Gaussian or Moffat PSFs the optional arguments **-fwhm=**, **-angle=** and **-ratio=** may be provided. For Moffat PSFs the optional argument **-beta=** may also be provided. If these values are omitted, they default to the same values as in the deconvolution dialog. For disc PSFs only the argument **-fwhm=** is required, which for this function is used to set the *diameter* of the PSF. For Airy PSFs the following arguments may be provided: **-dia=** (sets the telescope diameter), **-fl=** (sets the telescope focal length), **-wl=** (sets the wavelength to calculate the Airy diffraction pattern for), **-pixelsize=** (sets the sensor pixel size), **-obstruct=** (sets the central obstruction as a percentage of the overall aperture area). If these parameters are not provided, wavelength will default to 525nm and central obstruction will default to 0%. Siril will attempt to read the others from the open image, but some imaging software may not provide all of them in which case you will get bad results, and note the metadata may not be populated for SER format videos. You will learn from experience which are safe to omit for your particular imaging setup.

For any of the above PSF generation options the optional argument **-ks=** may be provided to set the PSF dimension, and the optional argument **-savepsf=filename** may be used to save the generated PSF: a filename must be provided and the same filename extension requirements apply as for **makepsf save filename**

Links: *psf*, *rl*, *sb*, *wiener*

merge

```
merge sequence1 sequence2 [sequence3 ...] output_sequence
```

Merges several sequences of the same type (FITS images, FITS sequence or SER) and same image properties into a new sequence with base name **newseq** created in the current working directory, with the same type. The input sequences can be in different directories, can specified either in absolute or relative path, with the exact .seq name or with only the base name with or without the trailing '_'

merge_cfa

```
merge_cfa file_CFA0 file_CFA1 file_CFA2 file_CFA3 bayerpattern
```

Builds a Bayer masked color image from 4 separate images containing the data from Bayer subchannels CFA0, CFA1, CFA2 and CFA3. (The corresponding command to split the CFA pattern into subchannels is **split_cfa**.) This function can be used as part of a workflow applying some processing to the individual Bayer subchannels prior to demosaicing. The fifth parameter **bayerpattern** specifies the Bayer matrix pattern to recreate: **bayerpattern** should be one of 'RGGB', 'BGGR', 'GRBG' or 'GBRG'

mirrorx

```
mirrorx [-bottomup]
```

Flips the loaded image about the horizontal axis. Option **-bottomup** will only flip it if it's not already bottom-up

mirrorx_single

```
mirrorx_single image
```

Flips the image about the horizontal axis, only if needed (if it's not already bottom-up). It takes the image file name as argument, allowing it to avoid reading image data entirely if no flip is required. Image is overwritten if a flip is made

mirrory

```
mirrory
```

Flips the image about the vertical axis

modasinh

```
modasinh -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -independent | -sat]␣
↪[channels]
```

Modified arcsinh stretch based on the work of the ghsastro.co.uk team.

The argument **-D=** defines the strength of the stretch, between 0 and 10. This is the only mandatory argument. The following optional arguments further tailor the stretch:
**LP** defines a shadow preserving range between 0 and SP where the stretch will be linear, preserving shadow detail;
**SP** defines the symmetry point of the stretch, between 0 and 1, which is the point at which the stretch will be most intense;

**HP** defines a region between HP and 1 where the stretch is linear, preserving highlight details and preventing star bloat.

If omitted LP and SP default to 0.0 ad HP defaults to 1.0.

An optional argument (either **-human**, **-even** or **-independent**) can be passed to select either human-weighted or even-weighted luminance or independent colour channels for colour stretches. The argument is ignored for mono images. Alternatively, the argument **-sat** specifies that the stretch is performed on image saturation - the image must be color and all channels must be selected for this to work.

Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels. The clip mode can be set using the argument **-clipmode=**: values **clip**, **rescale**, **rgbblend** or **globalrescale** are accepted and the default is rgbblend

mtf

```
mtf low mid high [channels]
```

Applies midtones transfer function to the current loaded image.

Three parameters are needed, **low**, **midtones** and **high** where midtones balance parameter defines a nonlinear histogram stretch in the [0,1] range. For an automatic determination of the parameters, see AUTOSTRETCH.

Optionally the parameter **[channels]** may be used to specify the channels to apply the stretch to: this may be R, G, B, RG, RB or GB. The default is all channels

Links: *autostretch*

neg

```
neg
```

Changes pixel values of the currently loaded image to a negative view, like 1-value for 32 bits, 65535-value for 16 bits. This does not change the display mode

new

```
new width height nb_channel [filename]
```

Creates a new image filled with zeros with a size of **width** x **height**.

Optionally the parameter **[filename]** may be used to specify the name of the new file.

The image is in 32-bit format, and it contains **nb_channel** channels, **nb_channel** being 1 or 3. It is not saved, but becomes the loaded image and it is displayed and can be saved afterwards

nozero

```
nozero level
```

Replaces null values by **level** values. Useful before an idiv or fdiv operation, mostly for 16-bit images

offline

```
offline
```

Sets Siril to offline mode. In this mode networking functions such as remote catalogue lookups, update of git repositories etc. are unavailable. Cached data is still accessible

offset

```
offset value
```

Adds the constant **value** (specified in ADU) to the current image. This constant can take a negative value.

In 16-bit mode, values of pixels that fall outside of [0, 65535] are clipped. In 32-bit mode, no clipping occurs

online

```
online
```

Sets Siril to online mode. In this mode networking functions such as remote catalogue lookups, update of git repositories etc. is allowed

parse

```
parse str [-r]
```

Parses the string **str** using the information contained in the header of the image currently loaded. Main purpose of this command is to debug path parsing of header keys which can be used in other commands.

Option **-r** specifies the string is to be interpreted in read mode. In read mode, all wildcards defined in string **str** are used to find a file name matching the pattern. Otherwise, default mode is write mode and wildcards, if any, are removed from the string to be parsed.

If **str** starts with *$def* prefix, it will be recognized as a reserved keyword and looked for in the strings stored in gui_prepro.dark_lib, gui_prepro.flat_lib, gui_prepro.bias_lib or gui_prepro.stack_default for *$defdark*, *$defflat*, *$defbias* or *$defstack* respectively.

The keyword *$seqname$* can also be used when a sequence is loaded

pcc

```
pcc [-limitmag=[+-]] [-catalog=] [-bgtol=lower,upper]
```

Run the Photometric Color Correction on the loaded plate-solved image.

The limit magnitude of stars is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.

The star catalog used is NOMAD by default, it can be changed by providing **-catalog=apass**, **-catalog=localgaia** or **-catalog=gaia**. If installed locally, the remote NOMAD (the complete version) can be forced by providing **-catalog=nomad**

Background reference outlier tolerance can be specified in sigma units using **-bgtol=lower,upper**: these default to -2.8 and +2.0

platesolve

```
platesolve [-force] [image_center_coords] [-focal=] [-pixelsize=]
platesolve ... [-noflip] [-downscale] [-order=] [-radius=] [-disto=]
```

(continues on next page)

(continued from previous page)

```
platesolve ... [-limitmag=[+-]] [-catalog=] [-nocrop]
platesolve ... [-localasnet [-blindpos] [-blindres]]
```

Plate solve the loaded image.

If the image has already been plate solved nothing will be done, unless the **-force** argument is passed to force a new solve. If WCS or other image metadata is erroneous or missing, arguments must be passed:

the approximate image center coordinates can be provided in decimal degrees or degree/hour minute second values (J2000 with colon separators), with right ascension and declination values separated by a comma or a space (not mandatory for astrometry.net).

focal length and pixel size can be passed with **-focal=** (in mm) and **-pixelsize=** (in microns), overriding values from image and settings. See also options to solve blindly with local Astrometry.net

Unless **-noflip** is specified, if the image is detected as being upside-down, it will be flipped.

For faster star detection in big images, downsampling the image is possible with **-downscale**.

The solve can account for distortions using SIP convention with polynomials up to order 5. Default value is taken form the astrometry preferences. This can be changed with the option **-order=** giving a value between 1 and 5.

When using Siril solver local catalogues or with local Astrometry.net, if the initial solve is not successful, the solver will search for a solution within a cone of radius specified with **-radius=** option. If no value is passed, the search radius is taken from the astrometry preferences. Siril near search can be disabled by passing a value of 0. (cannot be disabled for Astrometry.net).

You can save the current solution as a distortion file with the option **-disto=**.

Images can be either plate solved by Siril using a star catalog and the global registration algorithm or by astrometry.net's local solve-field command (enabled with **-localasnet**).

**Siril platesolver options:**

The limit magnitude of stars used for plate solving is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.

The choice of the star catalog is automatic unless the **-catalog=** option is passed: if local catalogs are installed, they are used, otherwise the choice is based on the field of view and limit magnitude. If the option is passed, it forces the use of the catalog given in argument, with possible values: tycho2, nomad, localgaia, gaia, ppmxl, brightstars, apass.

If the computed field of view is larger than 5 degrees, star detection will be bounded to a cropped area around the center of the image unless **-nocrop** option is passed.

**Astrometry.net solver options:**

Passing options **-blindpos** and/or **-blindres** enables to solve blindly for position and for resolution respectively. You can use these when solving an image with a completely unknown location and sampling

pm

```
pm "expression" [-rescale [low] [high]] [-nosum]
```

This command evaluates the expression given in argument as in PixelMath tool. The full expression must be between double quotes and variables (that are image names, without extension, located in the working directory in that case) must be surrounded by the token $, e.g. "$image1$ * 0.5 + $image2$ * 0.5". A maximum of 10 images can be used in the expression.

Image can be rescaled with the option **-rescale** followed by **low** and **high** values in the range [0, 1]. If no low and high values are provided, default values are set to 0 and 1. Another optional argument, **-nosum** tells Siril not to sum exposure times. This impacts FITS keywords such as LIVETIME and STACKCNT

profile

```
profile -from=x,y -to=x,y [-tri] [-cfa] [-arcsec] { [-savedat] | [-filename=] } [-
↪layer=] [-width=] [-spacing=] ["-title=My Plot"]
```

Generates an intensity profile plot between 2 points in the image, also known as a *cut*. The arguments may be provided in any order. The arguments **-to=x,y** and **-from=x,y** are mandatory.

The argument **-layer={red | green | blue | lum | col}** specifies which channel (or luminance or colour) to plot if the image is color. It may also be used with the **-tri** option, which generates 3 parallel equispaced profiles each separated by **-spacing=** pixels, but note that for tri profiles the **col** option will be treated the same as **lum**.

The option **-cfa** selects CFA mode, which generates 4 profiles: 1 for each CFA channel in a Bayer patterned image. This option cannot be used with color images or mono images with no Bayer pattern, and cannot be used at the same time as the **-tri** option.

The option **-arcsec** causes the x axis to display distance in arcsec, if the necessary metadata is available. If not provided or if metadata is not available, distance will be shown in pixel units.

The argument **-savedat** will cause the data files to be saved: the filename will be written to the log. Alternatively the argument **-filename=** can be used to specify a filename to write the data file to. (The **-filename=** option implies **-savedat**.)

The argument **"-title=My Title"** sets a custom title "My Title"

psf

```
psf [channel]
```

Performs a PSF (Point Spread Function) on the selected star and display the results. For headless operation, the selection can be given in pixels using BOXSELECT. If provided, the **channel** argument selects the image channel on which the star will be analyzed. It can be omitted for monochrome images or when run from the GUI with one of the channels active in the view

Links: *boxselect*

pwd

```
pwd
```

Prints the current working directory

pyscript

```
pyscript [-async] scriptname.py [script_argv]
```

Executes a Siril python script

An optional argument **-async** may be provided. If it is provided it **must** be the first argument. This allows the script launched by pyscript to run in the background without blocking a script that calls it. This is expected to be useful for launcher-type scripts. The default is for the script to block until completion if pyscript is run as part of a script, but not to block if run from the GUI command entry (so the -async argument makes no difference when running like that).

The script name must be provided next (or first, if **-async** is not provided). If it is not found in the current working directory, the user-defined script paths specified in Preferences and the local siril-scripts repository will be searched. All subsequent arguments will be treated as script arguments and passed to the script as its argument vector. Note that the specific script must incorporate support for reading input from the argument vector

register

```
register sequencename [-2pass] [-selected] [-prefix=] [-scale=]
register sequencename ... [-layer=] [-transf=] [-minpairs=] [-maxstars=] [-nostarlist] [-
↪disto=]
register sequencename ... [-interp=] [-noclamp]
register sequencename ... [-drizzle [-pixfrac=] [-kernel=] [-flat=]]
```

Finds and optionally performs geometric transforms on images of the sequence given in argument so that they may be superimposed on the reference image. Using stars for registration, this algorithm only works with deep sky images. Star detection options can be changed using **SETFINDSTAR** or the *Dynamic PSF* dialog.

All images of the sequence will be registered unless the option **-selected** is passed, in that case the excluded images will not be processed.

The **-2pass** option will only compute the transforms but not generate the transformed images, **-2pass** adds a preliminary pass to the algorithm to find a good reference image before computing the transforms, based on image quality and framing. To generate transformed images after this pass, use SEQAPPLYREG.

If created, the output sequence name will start with the prefix "r_" unless otherwise specified with **-prefix=** option. The output images can be rescaled by passing a **-scale=** argument with a float value between 0.1 and 3.

**Image transformation options:**

The detection is done on the green layer for colour images, unless specified by the **-layer=** option with an argument ranging from 0 to 2 for red to blue.

**-transf=** specifies the use of either **shift**, **similarity**, **affine** or **homography** (default) transformations respectively.

**-minpairs=** will specify the minimum number of star pairs a frame must have with the reference frame, otherwise the frame will be dropped and excluded from the sequence.

**-maxstars=** will specify the maximum number of stars to find within each frame (must be between 100 and 2000). With more stars, a more accurate registration can be computed, but will take more time to run.

**-nostarlist** disables saving the star lists to disk.

**-disto=** uses distortion terms from a previous platesolve solution (with a SIP order > 1). It takes as parameter either **image** to use the solution contained in the currently loaded image, **file** followed by the path to the image containing the solution or **master** to load automatically the matching distortion master corresponding to each image. When using this option, the polynomials are used both to correct star positions before computing the transformation and to undistort the images when output images are exported.

**Image interpolation options:**

By default, transformations are applied to register the images by using interpolation.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument.

**Image drizzle options:**

Otherwise, the images can be exported using HST drizzle algorithm by passing the argument **-drizzle** which can take the additional options:

**-pixfrac=** sets the pixel fraction (default = 1.0).

The **-kernel=** argument sets the drizzle kernel and must be followed by one of **point**, **turbo**, **square**, **gaussian**, **lanczos2** or **lanczos3**. The default is **square**.

The **-flat=** argument specifies a master flat to weight the drizzled input pixels (default is no flat).

Note: when using **-drizzle** on images taken with a color camera, the input images must not be debayered. In that case, star detection will always occur on the green pixels

Links: *setfindstar*, *psf*, *seqapplyreg*

reloadscripts

```
reloadscripts
```

Rescans the scripts folders and updates the Scripts menu. Note that this command may not be used headless (i.e. from siril-cli) and also that the update will happen asynchronously (i.e. the command will trigger the update but will not wait for the update to complete)

requires

```
requires min_version [obsolete_version]
```

Returns an error if the version of Siril is older than the minimum required version passed in the first argument. Optionally, takes a second argument for the Siril version at which the script is obsolete: returns an error if the version of Siril is **newer than or equal to** the one passed in the second argument.

Example: *requires 1.2.0 1.4.0* allows the script to run for all of the 1.2.x series and 1.3.x series, but will not run for any versions earlier than 1.2.0 or for version 1.4.0 or any later versions

resample

```
resample { factor | -width= | -height= | -maxdim= } [-interp=] [-noclamp]
```

Resamples the loaded image, either with a factor **factor** or for the target width or height provided by either of **-width=**, **-height=** or **-maxdim=**. This is generally used to resize images: a factor of 0.5 divides size by 2. The **-maxdim** argument can be used to resize the longest dimension of the image to a set size, which can be useful for optimizing images for certain websites, e.g. social media websites.

In the graphical user interface, we can see that several interpolation algorithms are proposed.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument

rgbcomp

```
rgbcomp red green blue [-out=result_filename] [-nosum]
rgbcomp -lum=image { rgb_image | red green blue } [-out=result_filename] [-nosum]
```

Creates an RGB composition using three independent images, or an LRGB composition using the optional luminance image and three monochrome images or a color image. Result image is called composed_rgb.fit or composed_lrgb.fit unless another name is provided in the optional argument. Another optional argument, **-nosum** tells Siril not to sum exposure times. This impacts FITS keywords such as LIVETIME and STACKCNT

rgradient

```
rgradient xc yc dR dalpha
```

Creates two images, with a radial shift (**dR** in pixels) and a rotational shift (**dalpha** in degrees) with respect to the point (**xc**, **yc**).

Between these two images, the shifts have the same amplitude, but an opposite sign. The two images are then added to create the final image. This process is also called Larson Sekanina filter

rl

```
rl [-loadpsf=] [-alpha=] [-iters=] [-stop=] [-gdstep=] [-tv] [-fh] [-mul]
```

Restores an image using the Richardson-Lucy method.

Optionally, a PSF may be loaded using the argument **-loadpsf=filename** (created with MAKEPSF).

The number of iterations is provide by **-iters** (the default is 10).

The type of regularization can be set with **-tv** for Total Variation, or **-fh** for the Frobenius norm of the Hessian matrix (the default is none) and **-alpha=** provides the regularization strength (lower value = more regularization, default = 3000).

By default the gradient descent method is used with a default step size of 0.0005, however the multiplicative method may be specified with **-mul**.

The stopping criterion may be activated by specifying a stopping limit with **-stop=**

Links: *psf* , *makepsf*

rmgreen

```
rmgreen [-nopreserve] [type] [amount]
```

Applies a chromatic noise reduction filter. It removes green tint in the current image. This filter is based on PixInsight's SCNR and it is also the same filter used by HLVG plugin in Photoshop.
Lightness is preserved by default but this can be disabled with the **-nopreserve** switch.

**Type** can take values 0 for average neutral, 1 for maximum neutral, 2 for maximum mask, 3 for additive mask, defaulting to 0. The last two can take an **amount** argument, a value between 0 and 1, defaulting to 1

rotate

```
rotate degree [-nocrop] [-interp=] [-noclamp]
```

Rotates the loaded image by an angle of **degree** value. The option **-nocrop** can be added to avoid cropping to the image size (black borders will be added).

Note: if a selection is active, i.e. by using a command `boxselect` before `rotate`, the resulting image will be a rotated crop. In this particular case, the option **-nocrop** will be ignored if passed.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.
Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument

rotatePi

```
rotatePi
```

Rotates the loaded image of an angle of 180° around its center. This is equivalent to the command "ROTATE 180" or "ROTATE -180"

Links: *rotate*

satu

```
satu amount [background_factor [hue_range_index]]
```

Enhances the color saturation of the loaded image. Try iteratively to obtain best results.

**amount** can be a positive number to increase color saturation, negative to decrease it, 0 would do nothing, 1 would increase it by 100%

**background_factor** is a factor to (median + sigma) used to set a threshold for which only pixels above it would be modified. This allows background noise to not be color saturated, if chosen carefully. Defaults to 1. Setting 0 disables the threshold.

**hue_range_index** can be [0, 6], meaning: 0 for pink to orange, 1 for orange to yellow, 2 for yellow to cyan, 3 for cyan, 4 for cyan to magenta, 5 for magenta to pink, 6 for all (default)

save

```
save filename [-chksum]
```

Saves current image to **filename**.fit (or .fits, depending on your preferences, see SETEXT) in the current working directory. The image remains loaded. **filename** can contain a path as long as the directory already exists. The **-chksum** option stores checksum keywords (CHECKSUM and DATASUM) in the FITS header

Links: *setext*

savebmp

```
savebmp filename
```

Saves current image under the form of a bitmap file with 8-bit per channel: **filename**.bmp (BMP 24-bit)

savejpg

```
savejpg filename [quality]
```

Saves current image into a JPG file: **filename**.jpg.

The compression quality can be adjusted using the optional **quality** value, 100 being the best and default, while a lower value increases the compression ratio

savejxl

```
savejxl filename [-effort=] [-quality=] [-8bit]
```

Saves current image into a JPG XL file: **filename**.jxl.

All other arguments are optional. The quality setting expresses a maximum permissible distance between the original and the compressed image: the **-quality=** argument may be provided and must be specified as a floating point number between 0.0 and 10.0. A higher quality means better quality, but larger file size. Quality = 10.0 is mathematically lossless, quality = 9.0 is visually lossless and quality = 0 is visually poor but gives very small file sizes. The default value is 9.0; typical values range from 7.0 to 10.0. The compression effort can be adjusted using the optional **-effort=** value, 9 being the most effort but very slow, while a lower value increases the compression ratio. Values above 7 are not recommended as they can be very slow and produce little if any benefit to file size, in fact sometimes effort = 9 can produce larger files. If this argument is omitted the default value of 7 is used. An option **-8bit** may be provided to force output to be 8 bits per pixel

savepng

```
savepng filename
```

Saves current image into a PNG file: **filename**.png, with 16 bits per channel if the loaded image is 16 or 32 bits, and 8 bits per channel if the loaded image is 8 bits

savepnm

```
savepnm filename
```

Saves current image under the form of a NetPBM file format with 16-bit per channel.

The extension of the output will be **filename**.ppm for RGB image and **filename**.pgm for gray-level image

savetif

```
savetif filename [-astro] [-deflate]
```

Saves current image under the form of a uncompressed TIFF file with 16-bit per channel: **filename**.tif. The option **-astro** allows saving in Astro-TIFF format, while **-deflate** enables compression.

See also SAVETIF32 and SAVETIF8

savetif32

```
savetif32 filename [-astro] [-deflate]
```

Same command as SAVETIF but the output file is saved in 32-bit per channel: **filename**.tif. The option **-astro** allows saving in Astro-TIFF format, while **-deflate** enables compression

Links: *savetif*

savetif8

```
savetif8 filename [-astro] [-deflate]
```

Same command as SAVETIF but the output file is saved in 8-bit per channel: **filename**.tif. The option **-astro** allows saving in Astro-TIFF format, while **-deflate** enables compression

Links: *savetif*

sb

```
sb [-loadpsf=] [-alpha=] [-iters=]
```

Restores an image using the Split Bregman method.

Optionally, a PSF may be loaded using the argument **-loadpsf=filename**.

The number of iterations is provide by **-iters** (the default is 1).

The regularization factor **-alpha=** provides the regularization strength (lower value = more regularization, default = 3000)

Links: *psf*

select

```
select sequencename from to
```

This command allows easy mass selection of images in the sequence **sequencename** (from **from** to **to** included).
This is a selection for later processing.
See also UNSELECT

Links: *unselect*

Examples:

*select . 0 0*
selects the first of the currently loaded sequence

*select sequencename 1000 1200*
selects 201 images starting from number 1000 in sequence named sequencename

The second number can be greater than the number of images to just go up to the end.

seqapplyreg

```
seqapplyreg sequencename [-prefix=] [-scale=] [-layer=] [-framing=]
seqapplyreg sequencename ... [-interp=] [-noclamp]
seqapplyreg sequencename ... [-drizzle [-pixfrac=] [-kernel=] [-flat=]]
seqapplyreg sequencename ... [-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-
↪filter-round=value[%|k]] [-filter-bkg=value[%|k]] [-filter-nbstars=value[%|k]] [-
↪filter-quality=value[%|k]] [-filter-incl[uded]]
```

Applies geometric transforms on images of the sequence given in argument so that they may be superimposed on the reference image, using registration data previously computed (see REGISTER).

The output sequence name starts with the prefix **"r_"** unless otherwise specified with **-prefix=** option.

The registration is done on the first layer for which data exists for RGB images unless specified by **-layer=** option (0, 1 or 2 for R, G and B respectively).

The output images can be rescaled by passing a **-scale=** argument with a float value between 0.1 and 3.

Automatic framing of the output sequence can be specified using **-framing=** keyword followed by one of the methods in the list { current | min | max | cog } :

**-framing=max** (bounding box) will project each image and compute its shift wrt. reference image. The resulting sequence can then be stacked using option **-maximize** of STACK command which will create the full image encompassing all images of the sequence.

**-framing=min** (common area) crops each image to the area it has in common with all images of the sequence.

**-framing=cog** determines the best framing position as the center of gravity (cog) of all the images.

**Image interpolation options:**

By default, transformations are applied to register the images by using interpolation.

The pixel interpolation method can be specified with the **-interp=** argument followed by one of the methods in the list **no**[ne], **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}. If **none** is passed, the transformation is forced to shift and a pixel-wise shift is applied to each image without any interpolation.

Clamping of the bicubic and lanczos4 interpolation methods is the default, to avoid artefacts, but can be disabled with the **-noclamp** argument.

**Image drizzle options:**

Otherwise, the images can be exported using HST drizzle algorithm by passing the argument **-drizzle** which can take the additional options:

**-pixfrac=** sets the pixel fraction (default = 1.0).

The **-kernel=** argument sets the drizzle kernel and must be followed by one of **point**, **turbo**, **square**, **gaussian**, **lanczos2** or **lanczos3**. The default is **square**.

The **-flat=** argument specifies a master flat to weight the drizzled input pixels (default is no flat).

**Filtering out images:**

Images to be registered can be selected based on some filters, like those selected or with best FWHM, with some of the **-filter-*** options.

Links: *register*, *stack*

With filtering being some of these in no particular order or number:

```
[-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-filter-round=value[%|k]] [-filter-
↪bkg=value[%|k]]
[-filter-nbstars=value[%|k]] [-filter-quality=value[%|k]] [-filter-incl[uded]]
```

Best images from the sequence can be stacked by using the filtering arguments. Each of these arguments can remove

bad images based on a property their name contains, taken from the registration data, with either of the three types of argument values:
- a numeric value for the worse image to keep depending on the type of data used (between 0 and 1 for roundness and quality, absolute values otherwise),
- a percentage of best images to keep if the number is followed by a % sign,
- or a k value for the k.sigma of the worse image to keep if the number is followed by a k sign.

It is also possible to use manually selected images, either previously from the GUI or with the select or unselect commands, using the **-filter-included** argument.

seqccm

```
seqccm sequencename [-prefix=]
```

Same command as CCM but for the the sequence **sequencename**. Only selected images in the sequence are processed.

The output sequence name starts with the prefix "ccm" unless otherwise specified with option **-prefix=**

Links: *ccm*

seqclean

```
seqclean sequencename [-reg] [-stat] [-sel]
```

This command clears selection, registration and/or statistics data stored for the sequence **sequencename**.

You can specify to clear only registration, statistics and/or selection with **-reg**, **-stat** and **-sel** options respectively. All are cleared if no option is passed

seqcosme

```
seqcosme sequencename [filename].lst [-prefix=]
```

Same command as COSME but for the the sequence **sequencename**. Only selected images in the sequence are processed.

The output sequence name starts with the prefix "cosme_" unless otherwise specified with option **-prefix=**

Links: *cosme*

seqcosme_cfa

```
seqcosme_cfa sequencename [filename].lst [-prefix=]
```

Same command as COSME_CFA but for the the sequence **sequencename**. Only selected images in the sequence are processed.

The output sequence name starts with the prefix "cosme_" unless otherwise specified with option **-prefix=**

Links: *cosme_cfa*

seqcrop

```
seqcrop sequencename x y width height [-prefix=]
```

Crops the sequence given in argument **sequencename**. Only selected images in the sequence are processed.

The crop selection is specified by the upper left corner position **x** and **y** and the selection **width** and **height**, like for CROP.
The output sequence name starts with the prefix "cropped_" unless otherwise specified with **-prefix=** option

Links: *crop*

seqextract_Green

```
seqextract_Green sequencename [-prefix=]
```

Same command as EXTRACT_GREEN but for the sequence **sequencename**.

The output sequence name starts with the prefix "Green_" unless otherwise specified with option **-prefix=**

seqextract_Ha

```
seqextract_Ha sequencename [-prefix=] [-upscale]
```

Same command as EXTRACT_HA but for the sequence **sequencename**.

The output sequence name starts with the prefix "Ha_" unless otherwise specified with option **-prefix=**

seqextract_HaOIII

```
seqextract_HaOIII sequencename [-resample=]
```

Same command as EXTRACT_HAOIII but for the sequence **sequencename**.

The output sequences names start with the prefixes "Ha_" and "OIII_"

seqfind_cosme

```
seqfind_cosme sequencename cold_sigma hot_sigma [-prefix=]
```

Same command as FIND_COSME but for the sequence **sequencename**.

The output sequence name starts with the prefix "cc_" unless otherwise specified with **-prefix=** option

Links: *find_cosme*

seqfind_cosme_cfa

```
seqfind_cosme_cfa sequencename cold_sigma hot_sigma [-prefix=]
```

Same command as FIND_COSME_CFA but for the sequence **sequencename**.

The output sequence name starts with the prefix "cc_" unless otherwise specified with **-prefix=** option

Links: *find_cosme_cfa*

seqfindstar

```
seqfindstar sequencename [-layer=] [-maxstars=]
```

Same command as FINDSTAR but for the sequence **sequencename**.

The option **-out=** is not available for this process as all the star list files are saved with the default name *seqname_seqnb.lst*

Links: *findstar*

seqfixbanding

```
seqfixbanding sequencename amount sigma [-prefix=] [-vertical]
```

Same command as FIXBANDING but for the sequence **sequencename**.

The output sequence name starts with the prefix "unband_" unless otherwise specified with **-prefix=** option

Links: *fixbanding*

seqght

```
seqght sequence -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as GHT but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *ght*

seqheader

```
seqheader sequencename keyword [keyword2 ...] [-sel] [-out=file.csv]
```

Prints the FITS header value corresponding to the given keys for all images in the sequence. You can write several keys in a row, separated by a space. The **-out=** option, followed by a file name, allows you to print the output in a csv file. The **-sel** option limits the output to the images selected in the sequence

seqinvght

```
seqinvght sequence -D= [-B=] [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as INVGHT but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *invght*

seqinvmodasinh

```
seqinvmodasinh sequence -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as INVMODASINH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *invmodasinh*

seqlinstretch

```
seqlinstretch sequence -BP= [channels] [-sat] [-prefix=]
```

Same command as LINSTRETCH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *linstretch*

seqmerge_cfa

```
seqmerge_cfa sequencename0 sequencename1 sequencename2 sequencename3 bayerpattern [-
↪prefixout=]
```

Merges 4 sequences of images to recombine the Bayer pattern. The sequences are specified in the arguments **sequencename0**, **sequencename1**, **sequencename2** and **sequencename3**.

The Bayer pattern to be reconstructed must be provided as the second argument as one of RGGB, BGGR, GBRG or GRBG (the order of the Bayer channels must match the order of the specified sequences).

Note: all 4 input sequences **must** be present and have the same dimensions, bit depth and number of images.

The output sequence name starts with the prefix "mCFA_" and a number unless otherwise specified with **-prefixout=** option

seqmodasinh

```
seqmodasinh sequence -D= [-LP=] [-SP=] [-HP=] [-clipmode=] [-human | -even | -
↪independent | -sat] [channels] [-prefix=]
```

Same command as MODASINH but the sequence must be specified as the first argument. In addition, the optional argument **-prefix=** can be used to set a custom prefix

Links: *modasinh*

seqmtf

```
seqmtf sequencename low mid high [channels] [-prefix=]
```

Same command as MTF but for the sequence **sequencename**.

The output sequence name starts with the prefix "mtf_" unless otherwise specified with **-prefix=** option

Links: *mtf*

seqprofile

```
seqprofile sequence -from=x,y -to=x,y [-tri] [-cfa] [-arcsec] [-savedat] [-layer=] [-
↪width=] [-spacing=] [ {-xaxis=wavelength | -xaxis=wavenumber } ] [{-wavenumber1= | -
↪wavelength1=} -wn1at=x,y {-wavenumber2= | -wavelength2=} -wn2at=x,y] ["-title=My Plot"]
```

Generates an intensity profile plot between 2 points in each image in the sequence. After the mandatory first argument stating the sequence to process, the other arguments are the same as for the **profile** command. If processing a sequence and it is desired to have the current image number and total number of images displayed in the format "My Sequence (1 / 5)", the given title should end with () (e.g. "My Sequence ()" and the numbers will be populated automatically)

seqpsf

```
seqpsf sequencename [channel] [{ -at=x,y | -wcs=ra,dec }] [-followstar]
```

Same command as PSF but runs on sequences. This is similar to the one-star registration, except results can be used for photometry analysis rather than aligning images and the coordinates of the star can be provided by options.
This command is what is called internally by the menu that appears on right click in the image, with the PSF for the sequence entry. If registration data already exist for the sequence, they will can be used to shift the search window in each image. If there is no registration data and if there is significant shift between images in the sequence, the default settings will fail to find stars in the initial position of the search area.
The follow star option can then be activated with the argument **-followstar**.

Results will be displayed in the Plot tab, from which they can also be exported to a comma-separated values (CSV) file for external analysis.

When creating a light curve, the first star for which seqpsf has been run, marked 'V' in the display, will be considered as the variable star. All others are averaged to create a reference light curve subtracted to the light curve of the variable star.

Currently, in headless operation, the command prints some analysed data in the console, another command allows several stars to be analysed and plotted as a light curve: LIGHT_CURVE. Arguments are mandatory: the sequence name must be provided ("." may be used to indicate the currently loaded sequence) and when headless it is mandatory to provide the coordinates of the star, with -at= allowing coordinates in pixels to be provided for the target star of -wcs= allowing J2000 equatorial coordinates to be provided

Links: *psf*, *light_curve*

seqplatesolve

```
seqplatesolve sequencename [image_center_coords] [-focal=] [-pixelsize=]
seqplatesolve sequencename ... [-downscale] [-order=] [-radius=] [-force] [-noreg] [-
↪disto=]
seqplatesolve sequencename ... [-limitmag=[+-]] [-catalog=] [-nocrop] [-nocache]
seqplatesolve sequencename ... [-localasnet [-blindpos] [-blindres]]
```

Plate solve a sequence. A new sequence will be created with the prefix "ps_" if the input sequence is SER, otherwise, the images headers will be updated. In case of SER, providing the metadata is mandatory and the output sequence will be in the FITS cube format, as SER cannot store WCS data.

If WCS or other image metadata are erroneous or missing, arguments must be passed:

the approximate image center coordinates can be provided in decimal degrees or degree/hour minute second values (J2000 with colon separators), with right ascension and declination values separated by a comma or a space (not mandatory for astrometry.net).

focal length and pixel size can be passed with **-focal=** (in mm) and **-pixelsize=** (in microns), overriding values from images and settings. See also options to solve blindly with local Astrometry.net

For faster star detection in big images, downsampling the image is possible with **-downscale**.

The solve can account for distortions using SIP convention with polynomials up to order 5. Default value is taken form the astrometry preferences. This can be changed with the option **-order=** giving a value between 1 and 5.

When using Siril solver local catalogues or with local Astrometry.net, if the initial solve is not successful, the solver will search for a solution within a cone of radius specified with **-radius=** option. If no value is passed, the search radius is taken from the astrometry preferences. Siril near search can be disabled by passing a value of 0. (cannot be disabled for Astrometry.net).

Images already solved will be skipped by default. This can be disabled by passing the option **-force**.

Using this command will update registration data unless the option **-noreg** is passed.

You can save the current solution as a distortion file with the option **-disto=**.

Images can be either plate solved by Siril using a star catalogue and the global registration algorithm or by astrometry.net's local solve-field command (enabled with **-localasnet**).

**Siril platesolver options:**

The limit magnitude of stars used for plate solving is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.

The choice of the star catalog is automatic unless the **-catalog=** option is passed: if local catalogs are installed, they are used, otherwise the choice is based on the field of view and limit magnitude. If the option is passed, it forces the use of the remote catalog given in argument, with possible values: tycho2, nomad, gaia, ppmxl, brightstars, apass.

If the computed field of view is larger than 5 degrees, star detection will be bounded to a cropped area around the center of the image unless **-nocrop** option is passed.

When using online catalogues, a single catalogue extraction will be done for the entire sequence. If there is a lot of

drift or different sampling, that may not succeed for all images. This can be disabled by passing the argument **-nocache**, in which case metadata from each image will be used (except for the forced values like center coordinates, pixel size and/or focal length).

**Astrometry.net solver options:**
Passing options **-blindpos** and/or **-blindres** enables to solve blindly for position and for resolution respectively. You can use these when solving an image with a completely unknown location and sampling

seqresample

```
seqresample sequencename { -scale= | -width= | -height= } [-interp=] [-prefix=]
```

Scales the sequence given in argument **sequencename**. Only selected images in the sequence are processed.

The scale factor is specified either by the **-scale=** argument or by setting the output width, height or maximum dimension using the **-width=**, **-height=** or **-maxdim=** options.

An interpolation method may be specified using the **-interp=** argument followed by one of the methods in the list **ne**[arest], **cu**[bic], **la**[nczos4], **li**[near], **ar**[ea]}.. Clamping is applied for cubic and lanczos interpolation.

The output sequence name starts with the prefix "scaled_" unless otherwise specified with **-prefix=** option

seqrl

```
seqrl sequencename [-loadpsf=] [-alpha=] [-iters=] [-stop=] [-gdstep=] [-tv] [-fh] [-mul]
```

The same as the RL command, but applies to a sequence which must be specified as the first argument

Links: *rl*

seqsb

```
seqsb sequencename [-loadpsf=] [-alpha=] [-iters=]
```

The same as the SB command, but applies to a sequence which must be specified as the first argument

Links: *sb*

seqsetmag

```
seqsetmag magnitude
```

Same as SETMAG command but for the loaded sequence.

This command is only valid after having run SEQPSF or its graphical counterpart (select the area around a star and launch the PSF analysis for the sequence, it will appear in the graphs).
This command has the same goal as SETMAG but recomputes the reference magnitude for each image of the sequence where the reference star has been found.
When running the command, the last star that has been analysed will be considered as the reference star. Displaying the magnitude plot before typing the command makes it easy to understand.
To reset the reference star and magnitude offset, see SEQUNSETMAG

Links: *setmag*, *seqpsf*, *psf*, *sequnsetmag*

seqsplit_cfa

```
seqsplit_cfa sequencename [-prefix=]
```

Same command as SPLIT_CFA but for the sequence **sequencename**.

The output sequences names start with the prefix "CFA_" and a number unless otherwise specified with **-prefix=** option.
*Limitation:* the sequence always outputs a sequence of FITS files, no matter the type of input sequence

Links: *split_cfa*

seqstarnet

```
seqstarnet sequencename [-stretch] [-upscale] [-stride=value] [-nostarmask]
```

This command calls Starnet++ to remove stars from the sequence **sequencename**. See STARNET

Links: *starnet*

seqstat

```
seqstat sequencename output_file [option] [-cfa]
```

Same command as STAT for sequence **sequencename**.

Data is saved as a csv file **output_file**.
The optional parameter defines the number of statistical values computed: **basic**, **main** (default) or **full** (more detailed but longer to compute).
\t**basic** includes mean, median, sigma, bgnoise, min and max
\t**main** includes basic with the addition of avgDev, MAD and the square root of BWMV
\t**full** includes main with the addition of location and scale.

If **-cfa** is passed and the images are CFA, statistics are made on per-filter extractions

Links: *stat*

seqsubsky

```
seqsubsky sequencename { -rbf | degree } [-nodither] [-samples=20] [-tolerance=1.0] [-
↪smooth=0.5] [-prefix=]
```

Same command as SUBSKY but for the sequence **sequencename**.
Dithering, required for low dynamic gradients, can be disabled with **-nodither**. Note that the **-existing** option is not available for sequence background removal, as the frames of a sequence are not necessarily always aligned.

The output sequence name starts with the prefix "bkg_" unless otherwise specified with **-prefix=** option. Only selected images in the sequence are processed

Links: *subsky*

seqtilt

```
seqtilt sequencename
```

Same command as TILT but for the sequence **sequencename**. It generally gives better results

Links: *tilt*

sequnsetmag

```
sequnsetmag
```

Resets the magnitude calibration and reference star for the sequence. See SEQSETMAG

Links: *seqsetmag*

sequpdate_key

```
sequpdate_key sequencename key value [keycomment]
sequpdate_key sequencename -delete key
sequpdate_key sequencename -modify key newkey
sequpdate_key sequencename -comment comment
```

Same command as UPDATE_KEY but for the sequence **sequencename**. However, this command won't work on SER sequence

Links: *update_key*

seqwiener

```
wiener sequencename [-loadpsf=] [-alpha=]
```

The same as the **WIENER** command, but applies to a sequence which must be specified as the first argument

Links: *wiener*

set

```
set { -import=inifilepath | variable=value }
```

Updates a setting value, using its variable name, with the given value, or a set of values using an existing ini file with **-import=** option.
See GET to get values or the list of variables

Links: *get*

set16bits

```
set16bits
```

Forbids images to be saved with 32 bits per channel on processing, use 16 bits instead

set32bits

```
set32bits
```

Allows images to be saved with 32 bits per channel on processing

setcompress

```
setcompress 0/1 [-type=] [q]
```

Defines if images are compressed or not.

**0** means no compression while **1** enables compression.
If compression is enabled, the type must be explicitly written in the option **-type=** ("rice", "gzip1", "gzip2").
Associated to the compression, the quantization value must be within [0, 256] range.

For example, "setcompress 1 -type=rice 16" sets the rice compression with a quantization of 16

setcpu

```
setcpu number
```

Defines the number of processing threads used for calculation.

Can be as high as the number of virtual threads existing on the system, which is the number of CPU cores or twice this number if hyperthreading (Intel HT) is available. The default value is the maximum number of threads available, so this should mostly be used to limit processing power. This is reset on every Siril run. See also SETMEM

Links: *setmem*

setext

```
setext extension
```

Sets the extension used and recognized by sequences.

The argument **extension** can be "fit", "fts" or "fits"

setfindstar

```
setfindstar [reset] [-radius=] [-sigma=] [-roundness=] [-focal=] [-pixelsize=] [-
↪convergence=] [ [-gaussian] | [-moffat] ] [-minbeta=] [-relax=on|off] [-minA=] [-
↪maxA=] [-maxR=]
```

Defines stars detection parameters for FINDSTAR and REGISTER commands.

Passing no parameter lists the current values.
Passing **reset** resets all values to defaults. You can then still pass values after this keyword.

Configurable values:

**-radius=** defines the radius of the initial search box and must be between 3 and 50.

**-sigma=** defines the threshold above noise and must be greater than or equal to 0.05.

**-roundness=** defines minimum star roundness and must between 0 and 0.95. **-maxR** allows an upper bound to roundness to be set, to visualize only the areas where stars are significantly elongated, do not change for registration.

**-minA** and **-maxA** define limits for the minimum and maximum amplitude of stars to keep, normalized between 0 and 1.

**-focal=** defines the focal length of the telescope.

**-pixelsize=** defines the pixel size of the sensor.

**-gaussian** and **-moffat** configure the solver model to be used (Gaussian is the default).

If Moffat is selected, **-minbeta=** defines the minimum value of beta for which candidate stars will be accepted and must be greater than or equal to 0.0 and less than 10.0.

**-convergence=** defines the number of iterations performed to fit PSF and should be set between 1 and 3 (more tolerant).

**-relax=** relaxes the checks that are done on star candidates to assess if they are stars or not, to allow objects not shaped like stars to still be accepted (off by default)

Links: *findstar*, *register*, *psf*

The threshold for star detection is computed as the median of the image (which represents in general the background level) plus k times sigma, sigma being the standard deviation of the image (which is a good indication of the noise amplitude). If you have many stars in your images and a good signal/noise ratio, it may be a good idea to increase this value to speed-up the detection and false positives.

It is recommended to test the values used for a sequence with Siril's GUI, available in the dynamic PSF toolbox from the analysis menu. It may improve registration quality to increase the parameters, but it is also important to be able to detect several tens of stars in each image.

## setmag

```
setmag magnitude
```

Calibrates the magnitudes by selecting a star and giving the known apparent magnitude.

All PSF computations will return the calibrated apparent magnitude afterwards, instead of an apparent magnitude relative to ADU values. Note that the provided value must match the magnitude for the observation filter to be meaningful.

To reset the magnitude constant see UNSETMAG

Links: *psf*, *unsetmag*

setmem

```
setmem ratio
```

Sets a new ratio of used memory on free memory.

**Ratio** value should be between 0.05 and 2, depending on other activities of the machine. A higher ratio should allow siril to process faster, but setting the ratio of used memory above 1 will require the use of on-disk memory, which is very slow and unrecommended, even sometimes not supported, leading to system crash. A fixed amount of memory can also be set in the generic settings, with SET, instead of a ratio

Links: *set*

setphot

```
setphot [-inner=20] [-outer=30] [-aperture=10] [-dyn_ratio=4.0] [-gain=2.3] [-min_val=0]␣
↪[-max_val=60000]
```

Gets or sets photometry settings, mostly used by SEQPSF. If arguments are provided, they will update the settings. None are mandatory, any can be provided, default values are shown in the command's syntax. At the end of the command, the active configuration will be printed.

The Aperture size is dynamic unless it is forced. If so, the **aperture** value from the settings is used. If dynamic, the radius of the aperture is defined by the supplied dynamic ratio ("radius/half-FWHM").
Allowed values for the argument **-dyn_ratio** are in the range [1.0, 5.0]. A value outside this range will automatically set the aperture to the fixed value **-aperture**.

Gain is used only if not available from the FITS header

Links: *seqpsf*

setref

```
setref sequencename image_number
```

Sets the reference image of the sequence given in first argument. **image_number** is the sequential number of the image in the sequence, not the number in the filename, starting at 1

show

```
show [-clear] [{ -list=file.csv | [name] RA Dec }] [-nolog] [-notag]
```

Shows a point on the loaded plate solved image using the temporary user annotations catalogue, based on its equatorial coordinates. The **-clear** option clears this catalogue first and can be used alone.
Several points can be passed using a CSV file with the option **-list=** containing at least ra and dec columns. If the passed file also contains a column name, names will be used as tags in the image and listed in the Console, unless toggled off with the options **-notag** and **-nolog**.

This is only available from the GUI of Siril

spcc

```
spcc [-limitmag=[+-]] [ { -monosensor= [ -rfilter= ] [-gfilter=] [-bfilter=] | -
↪oscsensor= [-oscfilter=] [-osclpf=] } ] [-whiteref=] [ -narrowband [-rwl=] [-gwl=] [-
↪bwl=] [-rbw=] [-gbw=] [-bbw=] ] [-bgtol=lower,upper] [ -atmos [-obsheight=] { [-
↪pressure=] | [-slp=] } ]
```

Run the Spectrophotometric Color Correction on the loaded platesolved image.

The limit magnitude of stars is automatically computed from the size of the field of view, but can be altered by passing a +offset or -offset value to **-limitmag=**, or simply an absolute positive value for the limit magnitude.
The star catalog used for SPCC is always Gaia DR3: by default the local Gaia DR3 xp_sampled catalog will be used if available but this can be overridden with **-catalog={gaia | localgaia}**.

The names of sensors and filters can be specified using the following options: **-monosensor=**, **-rfilter=**, **-gfilter=**, **-bfilter=** or **-oscsensor=**, **-oscfilter=**, **-osclpf=**; the name of the white reference can be specified using the **-whiteref=** option. In all cases the name must be provided exactly as it is in the combo boxes in the SPCC tool. Note that sensor, filter and white reference names may contain spaces: in this case when using them as arguments to the **spcc** command, the entire argument must be enclosed in quotation marks, for example "-whiteref=Average Spiral Galaxy".

Narrowband mode can be selected using the argument **-narrowband**, in which case the previous filter arguments are ignored and NB filter wavelengths and bandwidths can be provided using **-rwl=**, **-rbw=**, **-gwl=**, **-gbw=**, **-bwl=** and **-bbw=**.

If one of the spectral data argument is omitted, the previously used value will be used.

Background reference outlier tolerance can be specified in sigma units using **-bgtol=lower,upper**: these default to -2.8 and +2.0.

Atmospheric correction can be applied by passing **-atmos**. In this case the following optional arguments apply: **-obsheight=** specifies the observer's height above sea level in metres (default 10), **-pressure=** specifies local atmospheric pressure at the observing site in hPa, or **-slp=** specifies sea-level atmospheric pressure in hPa (default pressure is 1013.25 hPa at sea level)

spcc_list

```
spcc_list { oscsensor | monosensor | redfilter | greenfilter | bluefilter | oscfilter |␣
↪osclpf | whiteref }
```

Print a list of SPCC names available for use to define sensors, filters or white references using the **spcc** command. This command requires an argument to set which list is printed: the options are **oscsensor**, **monosensor**, **redfilter**, **greenfilter**, **bluefilter**, **oscfilter**, **osclpf** or **whiteref**.

Note that sensor, filter and white reference names may contain spaces: in this case when using them as arguments to the **spcc** command, the entire argument must be enclosed in quotation marks, for example "-whiteref=Average Spiral Galaxy"

Links: *spcc*

split

```
split file1 file2 file3 [-hsl | -hsv | -lab]
```

Splits the loaded color image into three distinct files (one for each color) and saves them in **file1**.fit, **file2**.fit and **file3**.fit files. A last argument can optionally be supplied, **-hsl**, **-hsv** or **lab** to perform an HSL, HSV or CieLAB extraction. If no option are provided, the extraction is of RGB type, meaning no conversion is done

split_cfa

```
split_cfa
```

Splits the loaded CFA image into four distinct files (one for each channel) and saves them in files

stack

```
stack seqfilename
stack seqfilename { sum | min | max } [-output_norm] [-out=filename] [-maximize] [-
↪upscale] [-32b]
stack seqfilename { med | median } [-nonorm, -norm=] [-fastnorm] [-rgb_equal] [-output_
↪norm] [-out=filename] [-32b]
stack seqfilename { rej | mean } [rejection type] [sigma_low sigma_high]  [-rejmap[s]] [-
↪nonorm, -norm=] [-fastnorm] [-overlap_norm] [-weight={noise|wfwhm|nbstars|nbstack}] [-
↪feather=] [-rgb_equal] [-output_norm] [-out=filename] [-maximize] [-upscale] [-32b]
```

Stacks the **sequencename** sequence, using options.

Rejection type:
The allowed types are: **sum**, **max**, **min**, **med** (or **median**) and **rej** (or **mean**). If no argument other than the sequence name is provided, sum stacking is assumed.

Stacking with rejection:
Types **rej** or **mean** require the use of additional arguments for rejection type and values. The rejection type is one of **n[one], p[ercentile], s[igma], m[edian], w[insorized], l[inear], g[eneralized], [m]a[d]** for Percentile, Sigma, Median, Winsorized, Linear-Fit, Generalized Extreme Studentized Deviate Test or k-MAD clipping. If omitted, the default Winsorized is used.
The **sigma low** and **sigma high** parameters of rejection are mandatory unless **none** is selected.
Optionally, rejection maps can be created, showing where pixels were rejected in one (**-rejmap**) or two (**-rejmaps**, for low and high rejections) newly created images.

Normalization of input images:
For **med** (or **median**) and **rej** (or **mean**) stacking types, different types of normalization are allowed: **-norm=add** for additive, **-norm=mul** for multiplicative. Options **-norm=addscale** and **-norm=mulscale** apply same normalization but with scale operations. **-nonorm** is the option to disable normalization. Otherwise addtive with scale method is applied by default.
**-fastnorm** option specifies to use faster estimators for location and scale than the default IKSS.
**-overlap_norm**, if passed, will compute normalization coeffcients on images overlaps instead of whole images (allowed only if **-maximize** is passed).

Other options for rejection stacking:
Weighting can be applied to the images of the sequences using the option **-weight=** followed by:
**noise** to add larger weights to frames with lower background noise.
**nbstack** to weight input images based on how many images were used to create them, useful for live stacking.
**nbstars** or **wfwhm** to weight input images based on number of stars or wFWHM computed during registration step.
**-feather=** option will apply a feathering mask on each image borders over the distance (in pixels) given in argument.

Outputs:
Result image name can be set with the **-out=** option. Otherwise, it will be named as **sequencename**_stacked.fit.
**-output_norm** applies a normalization to rescale result in the [0, 1] range (median and mean stacking only).
**-maximize** option will use registration data from the sequence to create a stacked image that encompasses all the images of the sequence (applicable to all methods except median stacking).
**-upscale** option will upscale the sequence by a factor 2 prior to stacking using the registration data (applicable to all

methods except median stacking).

**-rgb_equal** will use normalization to equalize color image backgrounds, useful if PCC/SPCC or unlinked AUTOSTRETCH will not be used.

**-32b** will override the bitdepth set in Preferences and save the stacked image in 32b.

Filtering out images:

Images to be stacked can be selected based on some filters, like manual selection or best FWHM, with some of the **-filter-*** options.

Links: *pcc*, *spcc*, *autostretch*

```
[-filter-fwhm=value[%|k]] [-filter-wfwhm=value[%|k]] [-filter-round=value[%|k]] [-filter-
↪bkg=value[%|k]]
[-filter-nbstars=value[%|k]] [-filter-quality=value[%|k]] [-filter-incl[uded]]
```

Best images from the sequence can be stacked by using the filtering arguments. Each of these arguments can remove bad images based on a property their name contains, taken from the registration data, with either of the three types of argument values:

- a numeric value for the worse image to keep depending on the type of data used (between 0 and 1 for roundness and quality, absolute values otherwise),

- a percentage of best images to keep if the number is followed by a % sign,

- or a k value for the k.sigma of the worse image to keep if the number is followed by a k sign.

It is also possible to use manually selected images, either previously from the GUI or with the select or unselect commands, using the **-filter-included** argument.

stackall

```
stackall
stackall { sum | min | max } [-maximize] [-upscale] [-32b]
stackall { med | median } [-nonorm, norm=] [-32b]
stackall { rej | mean } [rejection type] [sigma_low sigma_high] [-nonorm, norm=] [-
↪overlap_norm] [-weight={noise|wfwhm|nbstars|nbstack}] [-feather=] [-rgb_equal] [-
↪out=filename] [-maximize] [-upscale] [-32b]
```

Opens all sequences in the current directory and stacks them with the optionally specified stacking type and filtering or with sum stacking. See STACK command for options description

Links: *stack*

starnet

```
starnet [-stretch] [-upscale] [-stride=value] [-nostarmask]
```

Calls StarNet to remove stars from the loaded image.

**Prerequisite:** StarNet is an external program, with no affiliation with Siril, and must be installed correctly prior the first use of this command, with the path to its CLI version installation correctly set in Preferences / Miscellaneous.

The starless image is loaded on completion, and a star mask image is created in the working directory unless the optional parameter **-nostarmask** is provided.

Optionally, parameters may be passed to the command:
- The option **-stretch** is for use with linear images and will apply a pre-stretch before running StarNet and the inverse stretch to the generated starless and starmask images.
- To improve star removal on images with very tight stars, the parameter **-upscale** may be provided. This will upsample the image by a factor of 2 prior to StarNet processing and rescale it to the original size afterwards, at the expense of more processing time.
- The optional parameter **-stride=value** may be provided, however the author of StarNet *strongly* recommends that the default stride of 256 be used

start_ls

```
start_ls [-dark=filename] [-flat=filename] [-rotate] [-32bits]
```

Initializes a livestacking session, using the optional calibration files and waits for input files to be provided by the LIVESTACK command until STOP_LS is called. Default processing will use shift-only registration and 16-bit processing because it's faster, it can be changed to rotation with **-rotate** and **-32bits**

*Note that the live stacking commands put Siril in a state in which it's not able to process other commands. After START_LS, only LIVESTACK, STOP_LS and EXIT can be called until STOP_LS is called to return Siril in its normal, non-live-stacking, state*

Links: *livestack*, *stop_ls*, *exit*

stat

```
stat [-cfa] [main]
```

Returns statistics of the current image, the basic list by default or the main list if **main** is passed. If a selection is made, statistics are computed within the selection. If **-cfa** is passed and the image is CFA, statistics are made on per-filter extractions

stop_ls

```
stop_ls
```

Stops the live stacking session. Only possible after START_LS

Links: *start_ls*

subsky

```
subsky { -rbf | degree } [-dither] [-samples=20] [-tolerance=1.0] [-smooth=0.5] [-
↪existing]
```

Computes a synthetic background gradient using either the polynomial function model of **degree** degrees or the RBF model (if **-rbf** is provided instead) and subtracts it from the image.
The number of samples per horizontal line and the tolerance to exclude brighter areas can be adjusted with the optional arguments. Tolerance is in MAD units: median + tolerance * mad.
Dithering, required for low dynamic gradients, can be enabled with **-dither**.
For RBF, the additional smoothing parameter is also available. To use pre-existing background samples (e.g. if you have set background samples using a Python script) the **-existing** argument must be used

synthstar

```
synthstar
```

Fixes imperfect stars from the loaded image. No matter how much coma, tracking drift or other distortion your stars have, if Siril's star finder routine can detect it, synthstar will fix it. To use intensive care, you may wish to manually detect all the stars you wish to fix. This can be done using the findstar console command or the Dynamic PSF dialog. If you have not run star detection, it will be run automatically with default settings.

For best results synthstar should be run before stretching.

The output of synthstar is a fully corrected synthetic star mask comprising perfectly round star PSFs (Moffat or Gaussian profiles depending on star saturation) computed to match the intensity, FWHM, hue and saturation measured for each star detected in the input image. This can then be recombined with the starless image to produce an image with perfect stars.

No parameters are required for this command

Links: *psf*

threshlo

```
threshlo level
```

Replaces values below **level** in the loaded image with **level**

threshhi

```
threshi level
```

Replaces values above **level** in the loaded image with **level**

thresh

```
thresh lo hi
```

Replaces values below **level** in the loaded image with **level**

tilt

```
tilt [clear]
```

Computes the sensor tilt as the FWHM difference between the best and worst corner truncated mean values. The **clear** option allows to clear the drawing

trixel

```
trixel [-p]
```

For developers.

Without any argument, lists all the trixels of level 3 visible in the plate-solved image. The stars from each trixel can then be shown with command CONESEARCH using **-trix=** followed by a visible trixel number

With argument **-p**, prints out all the valid stars from all the 512 level3 trixels to file "trixels.csv"

Links: *conesearch*

unclipstars

```
unclipstars
```

Re-profiles clipped stars of the loaded image to desaturate them, scaling the output so that all pixel values are <= 1.0

unpurple

```
unpurple [-starmask] [-blue=value] [-thresh=value]
```

Applies a cosmetic filter to reduce effects of purple fringing on stars.

If the **-starmask** parameter is given, a star mask will be used to identify areas of the image to affect. If a Dynamic PSF has already been run, this will be used for the starmask, otherwise one will be created automatically. The **-mod=** parameter should be given a value somewhere around 0.14 to reduce the amount of purple. The **-thresh=** will specify the size modifier for each star in the starmask and should be large enough to cause the stars to be entirely processed without remaining purple fringing. The value should between 0 and 1, typically around 0.5.
If the **-starmask** parameter is not given, the purple reduction will be applied across the entire image for any purple pixels with a luminance value higher than the given **-thresh=**. In this case, the **-thresh=** value should be reasonably low. This mode is useful for starmasks or other images without nebula or galaxy

Links: *psf*

unselect

```
unselect sequencename from to
```

Allows easy mass unselection of images in the sequence **sequencename** (from **from** to **to** included). See SELECT

Links: *select*

unsetmag

```
unsetmag
```

Resets the magnitude calibration to 0. See SETMAG

Links: *setmag*

unsharp

```
unsharp sigma multi
```

Applies an unsharp mask, actually a Gaussian filtered image with sigma **sigma** and a blend with the parameter **amount** used as such: out = in * (1 + amount) + filtered * (-amount).

See also GAUSS, the same without blending

Links: *gauss*

update_key

```
update_key key value [keycomment]
update_key -delete key
update_key -modify key newkey
update_key -comment comment
```

Updates FITS keyword. Please note that the validity of **value** is not checked. This verification is the responsibility of the user. It is also possible to delete a key with the **-delete** option in front of the name of the key to be deleted, or to modify the key with the **-modify** option. The latter must be followed by the key to be modified and the new key name. Finally, the **-comment** option, followed by text, adds a comment to the FITS header. Please note that any text containing spaces must be enclosed in double quotation marks

visu

```
visu low high
```

Displays the loaded image with **low** and **high** as the low and high threshold, GUI only

wavelet

```
wavelet nbr_layers type
```

Computes the wavelet transform of the loaded image on (**nbr_layers**=1...n) layer(s) using linear (**type**=1) or bspline (**type**=2) version of the 'à trous' algorithm. The result is stored in a file as a structure containing the layers, ready for weighted reconstruction with WRECONS.

See also EXTRACT

Links: *wrecons*, *extract*

wiener

```
wiener [-loadpsf=] [-alpha=]
```

Restores an image using the Wiener deconvolution method.

Optionally, a PSF created by MAKEPSF may be loaded using the argument **-loadpsf=filename**.

The parameter **-alpha=** provides the Gaussian noise modelled regularization factor

Links: *psf* , *makepsf*

wrecons

```
wrecons c1 c2 c3 ...
```

Reconstructs to current image from the layers previously computed with wavelets and weighted with coefficients **c1**, **c2**, ..., **cn** according to the number of layers used for wavelet transform, after the use of WAVELET

Links: *wavelet*

CHAPTER

TWENTYFIVE

# SIRILPY PYTHON MODULE API 1.0.25 REFERENCE

The sirilpy python module supports communication with a running Siril instance. It can request representations of the current loaded image plus its metadata, including details of detected stars, as well as the current loaded sequence and most frame metadata.

This documentation is autogenerated from version 1.0.25 of the python module code.

It can also run Siril commands using the `SirilInterface.cmd()` method, and the intent is to provide a capable interface for writing advanced scripts for Siril to a level not possible with the previous simple script files.

For example, scripts can now have PyQt6 or TKinter front ends using the PyQt6 or tkinter and ttkthemes modules, and they can utilise a lot of the ecosystem of Python modules including numpy, scipy, pillow and many more.

> **Note**
>
> There are some restrictions around modules that require installation of system binary packages for the modules to work.

In the initial module release, most methods relating to the image or sequence loaded in Siril are read-only. The intent is that the parameters of the loaded image or sequence can be obtained and used as inputs to scripts, for example for calculations or input to conditionals, but the existing mature Siril command set should in most cases be used to act on the loaded image. Thus header keywords can be set using `cmd("update_key", "key", "value")`, and most built-in image operations can be carried out using the appropriate Siril command. The main exception to the rule of python methods providing read-only access is the `set_image_pixeldata()` method, which allows for setting the pixel data in the loaded image from a numpy array. This means that new pixel processing algorithms can be added using python, initially getting the pixel data from the loaded image using `get_image_pixeldata()` and setting it on completion using `set_image_pixeldata()`. Similar functions are available for getting and setting pixel data from sequence frames.

## 25.1 Dependencies

The sirilpy module depends on the following:

- numpy >= 1.20.0
- packaging >= 21.0
- pywin32 >= 300 (only on Windows)

## 25.2 Siril Script Coding Guidance

- Unlike most python environments, Siril scripts run directly from Siril and the end user may not typically know how to access or how to use the python venv from outside Siril. This means that some tasks that are fairly trivial in typical python scenarios, such as installing packages, become harder. The user cannot be expected to access the command line and install packages themselves using `python3 -m pip install`.
  - The module therefore provides the `ensure_installed()` method. This uses pip to ensure that modules are installed so that they can be imported.
- Siril scripts should **always** provide the author's name, copyright details / license information and contact details such as a YouTube channel, website or forum where they can be contacted in regard to their script. This is advisory for independently distributed scripts and mandatory for any scripts that are submitted to the scripts repository. If you write a script, you are responsible for supporting it.
- As the repository grows and the API develops, not all the scripts published will necessarily always be compatible with all versions of Siril that will be in use:
  - If your script uses Siril commands, you should use the `requires` Siril command. This can be called directly in a Siril Script File, or it can be called from a Python script using `SirilInterface.cmd("requires", "min_version", {"max_version"})` (max_version is optional) but can be used to ensure scripts designed for older versions of Siril no longer show as applicable to newer versions if the command syntax has changed.
  - If your script uses any features of the Siril python module added since the initial release, you should call the `sirilpy.check_module_version()` method to ensure the installed version meets the requriements of your script. The versions at which features were added will be listed in the API documentation for all features added after the initial public release.
  - The code that populates the Siril view of the scripts repository will automatically filter out scripts whose Siril version or python module version requirements are not met.
  - If new classes or methods are added to the module after its initial public release the version at which they were introduced will be annotated in the docstring and the online documentation generated from it.
- Siril targets Linux, Windows and MacOS. Script writers are encouraged, where possible, to ensure that their scripts run correctly on all three OSes.

## 25.3 Sirilpy Connection

This submodule provides the main SirilInterface class used for communication between Siril and the python script. All its members are made available at the module root level, there is no need to import connection separately. Connection module for Siril, providing the ability to connect to a running Siril instance and communicate with it. Includes an extensive range of methods that can be used to get and set data from / to Siril.

**class** `sirilpy.connection.`**`SirilInterface`**

Bases: `object`

SirilInterface is the main class providing an interface to a running Siril instance and access to methods to interact with it through Siril's inbuilt command system and accessing image and sequence data.

**`analyse_image_from_file`**(*filepath*)

Request Siril to load an image from a file and analyse it. This method does not change the image currently loaded in Siril. Any image format supported by Siril is supported. An ImageAnalysis object is returned, containing parameters that may be used to assess the quality of an image for use in culling.

**Parameters**

**`filepath`** (`str`) -- String specifying the path to the image file to load.

**Return type**
*ImageAnalysis*

**Returns**
ImageAnalysis object containing quality metrics for the analysed image.

**Raises**

- **FileNotFoundError** -- if the specified file does not exist,
- *DataError* -- on receipt of incorrect data,
- *SirilError* -- if an error occurs.

**clear_image_bgsamples()**

Clears all background sample points from the image.

**clear_undo_history()**

Clears the image undo history, if there is undo history to be cleared

**cmd**(**args*)

Send a command to Siril to be executed. The range of available commands can be found by checking the online documentation. The command and its arguments are provided as a list of strings.

**Parameters**
***args** (str) -- Variable number of string arguments to be combined into a command

**Raises**

- *DataError* -- If no response (or an incorrect response) was received,
- *CommandError* -- If the command returns an error status code,
- *SirilError* -- If any other error occurs during execution.

### Example

```
siril.cmd("ght", "-D=0.5", "-b=2.0")
```

**command_error_message**(*status_code*)

Provides a string describing the return status from a Siril command.

**Parameters**
**status_code** (*CommandStatus*) -- The status code returned by the Siril command handler, or by the CommandError exception.

**Returns**
A string providing a description of the error code returned by a Siril command, for use in exception handling.

**Return type**
str

**confirm_messagebox**(*title*, *message*, *confirm_label*)

Create a modal confirmation message dialog in Siril and wait for the response.

**Parameters**

- **title** (str) -- The title to display in the message box (up to 256 characters)
- **message** (str) -- The message to display in the message box (up to 1021 characters)

- **confirm_label** (str) -- The label to display in the message box confirmation button (OK, Yes, Confirm etc.) (Up to 24 characters)

**Returns**
True if the message box confirmation button was clicked, False otherwise

**Return type**
bool

**Raises**

- *DataError* -- if no response was received,
- *SirilError* -- if another error occurred.

`connect()`

Establish a connection to Siril based on the pipe or socket path.

**Return type**
bool

**Returns**
True on success

**Raises**
*SirilConnectionError* -- if a connection error occurred

`create_new_seq(seq_root)`

Creates a new .seq file with all images named seq_rootXXXXX.ext located in the current home folder. If a sequence with the same name is already loaded in Siril, it will not be recreated. This only works for FITS files, not FITSEQ nor SER. The newly created sequence is not loaded in Siril.

**Parameters**
**seq_root** (str) -- The root name of the sequence to be created.

**Returns**
True if the sequence was successfully created, False otherwise.

**Return type**
bool

**Raises**
*SirilError* -- if an error occurred.

`disconnect()`

Closes the established socket or pipe connection. Note there is not usually any need to close this unless for some reason you wish to close a connection and subsequently reopen another one. This method is automatically called at script termination using an `atexit` handler so there is no need to do so manually. Calling this method will reset the progress bar.

**Raises**
*SirilConnectionError* -- if the connection cannot be closed.

`error_messagebox(my_string, modal=False)`

Send an error message to Siril. The maximum message length is 1022 bytes: longer messages will be truncated (but this is more than enough for an error message box). Note that the error message box is not modal by default: this is intended for displaying an error message more prominently than using the Siril log prior to quitting the application.

**Parameters**

- **my_string** (str) -- The message to display in the error message box

- **modal** (Optional[bool]) -- Sets whether or not the message box should be modal and wait for completion or non-modal and allow the script to continue execution. Note that although a modal message box will block execution of the script, if a TKinter main loop is running events will continue to queue up, so if the message box is triggered by clicking a button then the user may click it while the message box is shown and trigger a second message box which will display immediately the first one is closed.

**Returns**
True if the error was successfully displayed, False otherwise

**Return type**
bool

**Raises**
*SirilError* -- if an error occurred.

**get_image**(*with_pixels=True*, *preview=False*)

Request a copy of the current image open in Siril. Requires a single image to be loaded.

**Parameters**

- **with_pixels** (Optional[bool]) -- optional bool specifying whether to get pixel data as a NumPy array, or only the image metadata. Defaults to True
- **preview** (Optional[bool]) -- optional bool specifying whether to get pixeldata as a preview (i.e. 8-bit autostretched data) or as real image data. Defaults to False (i.e. real image data)

**Return type**
Optional[*FFit*]

**Returns**
FFit object containing the image metadata and (optionally) pixel data, or None if an error occurred

**Raises**

- *NoImageError* -- if no image is loaded in Siril
- *SirilError* -- if a decoding error occurs

**get_image_bgsamples**()

Request background samples data from Siril.

**Return type**
Optional[List[*BGSample*]]

**Returns**
List of BGSamples background samples, with each set of coordinates expressed as a tuple[float, float], or None if no background samples have been set.

**Raises**

- *NoImageError* -- If no image is currently loaded,
- *DataError* -- on receipt of bad data,
- *SirilError* -- For other errors during data retrieval,

**get_image_filename**()

Request the filename of the loaded image from Siril. Requires a single image to be loaded.

**Return type**
Optional[str]

**Returns**
The filename as a string.

**Raises**

- *NoImageError* -- if no image is loaded,
- *SirilError* -- if a decoding error occurs.

`get_image_fits_header(return_as='str')`

Retrieve the full FITS header of the current image loaded in Siril. Requires a single image to be loaded.

**Parameters**
`return_as` -- Optional string specifying the format of the returned header. Can be 'str' for a string or 'dict' for a dictionary.

**Returns**
The image FITS header as a string, or None if there is no header. dict: The image FITS header as a dictionary, or None if there is no header. None: If the header is empty or not available.

**Return type**
str

**Raises**

- *NoImageError* -- If no image is currently loaded,
- *SirilError* -- For other errors during data retrieval,

`get_image_history()`

Retrieve history entries in the FITS header of the current loaded Siril image using shared memory. Requires a single image to be loaded.

**Parameters**
`none.`

**Returns**

**The HISTORY entries in the FITS header as a list of strings, or**
None if there are no HISTORY keywords.

**Return type**
list

**Raises**

- *NoImageError* -- If no image is currently loaded,
- *SirilError* -- For other errors during data retrieval.

`get_image_iccprofile()`

Retrieve the ICC profile of the current Siril image using shared memory. Requires a single image to be loaded.

Args: none.

**Returns**
The image ICC profile as a byte array, or None if the current image has no ICC profile.

**Return type**
bytes

**Raises**

- *NoImageError* -- If no image is currently loaded,

- *SirilError* -- If any other error occurs.

**get_image_keywords()**

Request FITS keywords data from Siril as a FKeywords object. Requires a single image to be loaded.

**Return type**
`Optional[FKeywords]`

**Returns**
FKeywords object containing the FITS keywords, or None if an error occurred

**Raises**
*SirilError* -- if a decoding error occurs.

**get_image_pixeldata**(*shape=None*, *preview=False*, *linked=False*)

Retrieves the pixel data from the image currently loaded in Siril.

**Parameters**

- **shape** (`Optional[list[int]]`) -- Optional list of [x, y, w, h] specifying the region to retrieve. If provided, gets pixeldata for just that region. If None, gets pixeldata for the entire image.
- **preview** (`Optional[bool]`) -- optional bool specifying whether to get pixeldata as a preview (i.e. 8-bit autostretched data) or as real image data. Defaults to False (i.e. real image data)
- **linked** (`Optional[bool]`) -- optional bool specifying whether the autostretch preview should be linked or unlinked. If preview == False then this option is ignored.

**Returns**
The image data as a numpy array

**Return type**
numpy.ndarray

**Raises**

- *NoImageError* -- If no image is currently loaded,
- **ValueError** -- If an invalid shape is provided,
- *DataError* -- if the array cannot be reshaped to the correct dimensions,
- *SirilError* -- For other errors during pixel data retrieval,

**get_image_shape()**

Request the shape of the image from Siril.

**Return type**
`Optional[Tuple[int, int, int]]`

**Returns**
A tuple (channels, height, width) representing the shape of the image, or None if no image shape is available to return.

Raises: SirilError: if an error occurred.

**get_image_stars**(*channel=None*)

Request star model PSF data from Siril.

**Parameters**
**channel** (`Optional[int]`) -- Optional int specifying the channel to retrieve from. If provided 0 = Red / Mono, 1 = Green, 2 = Blue. If the channel is omitted the default behavior will be

used: channel 0 for mono images, channel 1 (green) for color images. **channel requires sirilpy v1.0.8 or higher.**

**Return type**
`List[PSFStar]`

**Returns**
List of PSFStar objects containing the star data, or None if no stars can be found. If stars have already been detected using the *findstar* command then this list will be returned, otherwise automatic star detection will be attempted with the current star finder settings.

**Raises**

- *NoImageError* -- If no image is currently loaded,
- **ValueError** -- If an invalid channel is provided,
- *SirilError* -- For other errors during data retrieval,

**get_image_stats**(*channel*)

Request image statistics from Siril for a specific channel. Requires a single image to be loaded.

**Parameters**
**channel** (`int`) -- Integer specifying which channel to get statistics for (typically 0, 1, or 2)

**Return type**
`Optional[ImageStats]`

**Returns**
ImageStats object containing the statistics, or None if no stats are available for the selected channel

**Raises**

- *NoImageError* -- if no image is loaded,
- *SirilError* -- if an error occurs.

**get_image_unknown_keys**()

Retrieve the unknown key in a FITS header of the current loaded Siril image using shared memory. Requires a single image to be loaded.

**Parameters**
**none.**

**Returns**
The unknown keys as a string, or None if there are no unknown keys.

**Return type**
bytes

**Raises**

- *NoImageError* -- If no image is currently loaded,
- *SirilError* -- For other errors during data retrieval.

**get_selection_star**(*shape=None*, *channel=None*, *assume_centred=False*)

Retrieves a PSFStar star model from the current selection in Siril. Only a single PSFStar is returned: if there are more than one in the selection, the first one identified by Siril's internal star detection algorithm is returned. **Update**: from sirilpy 1.0.4 this method uses Siril's photometry functions to try to provide photometrically accurate values for PSFStar.mag, PSFStar.s_mag and PSFStar.SNR. If photometry succeeded and no saturated pixels were detected then PSFStar.phot_is_valid will be True, otherwise it will be False. :type shape: `Optional[list[int]]` :param shape: Optional list of [x, y, w, h] specifying the selection to

retrieve from. w x h must not exceed 300 px x 300 px. If provided, looks for a star in the specified selection If None, looks for a star in the selection already made in Siril, if one is made.

**Parameters**

- **channel** (Optional[int]) -- Optional int specifying the channel to retrieve from. If provided 0 = Red / Mono, 1 = Green, 2 = Blue. If the channel is omitted the current viewport will be used if in GUI mode, or if not in GUI mode the method will fall back to channel 0
- **assume_centred** (bool) -- Optional bool specifying whether to assume the star is already centred in the selection. Defaults to False.

**Returns**

**the PSFStar object representing the star model, or None if**
no star is detected in the selection.

**Return type**
*PSFStar*

**Raises**

- **ValueError** -- If an invalid shape is provided,
- ***NoImageError*** -- If no image is loaded,
- ***SirilConnectionError*** -- If a communication error occurs,
- ***SirilError*** -- If any other error occurred during execution.

**get_selection_stats**(*shape=None*, *channel=None*)

Retrieves statistics for the current selection in Siril. Requires a single image or a sequence to be loaded.

**Parameters**

- **shape** (Optional[list[int]]) -- Optional list of [x, y, w, h] specifying the selection to retrieve from. If provided, looks for a star in the specified selection If None, looks for a star in the selection already made in Siril, if one is made.
- **channel** (Optional[int]) -- Optional int specifying the channel to retrieve from. If provided 0 = Red / Mono, 1 = Green, 2 = Blue. If the channel is omitted the current viewport will be used if in GUI mode, or if not in GUI mode the method will fall back to channel 0

**Returns**
the ImageStats object representing the selection statistics.

**Return type**
*ImageStats*

**Raises**

- ***SirilError*** -- If an error occurred during processing,
- **ValueError** -- If an invalid shape is provided.

**get_seq**()

Request metadata for the current sequence loaded in Siril.

**Return type**
Optional[*Sequence*]

**Returns**
Sequence object containing the current sequence metadata, or None if an error occurred

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

**get_seq_distodata**(*channel*)

Request sequence distortion data from Siril

channel: Integer specifying which channel to get registration data for (typically 0, 1, or 2)

**Return type**
Optional[*DistoData*]

**Returns**
DistoData object containing the channel distortion parameters, or None if an error occurred

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

**get_seq_frame**(*frame*, *with_pixels=True*, *preview=False*, *linked=False*)

Request sequence frame as a FFit from Siril. The keywords, statistics, header and other metadata are always returned: if an ICC profile is present, this will also be populated in the resulting FFit, and optionally the pixel data can also be returned.

**Parameters**

- **frame** (int) -- Integer specifying which frame in the sequence to retrieve data for (between 0 and Sequence.number - 1). This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **with_pixels** (Optional[bool]) -- bool specifying whether or not to return the pixel data for the frame (default is True).
- **preview** (Optional[bool]) -- bool specifying whether or not to return the real pixel data or an autostretched uint8_t preview version. Only has an effect in conjunction with with_pixels = True

**Return type**
Optional[*FFit*]

**Returns**
FFit object containing the frame data

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *DataError* -- on receipt of incorrect data,
- *SirilError* -- if an error occurs.

**get_seq_frame_filename**(*frame*)

Request the filename of the specified frame of the loaded sequence from Siril. Requires a sequence to be loaded.

**Parameters**
**frame** (*int*) -- Specifies the frame index. This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.

**Return type**
Optional[str]

**Returns**
The filename as a string.

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

`get_seq_frame_header`(*frame*, *return_as='str'*)

Retrieve the full FITS header of an image from the sequence loaded in Siril.

**Parameters**

- **frame** (`int`) -- Integer specifying which frame in the sequence to retrieve data for (between 0 and Sequence.number - 1). This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **return_as** -- Optional string specifying the format of the returned header. Can be 'str' for a string or 'dict' for a dictionary.

**Returns**
The image FITS header as a string, or None if there is no header. dict: The image FITS header as a dictionary, or None if there is no header. None: If the header is empty or not available.

**Return type**
str

**Raises**

- *NoSequenceError* -- If no sequence is currently loaded,
- *SirilError* -- For other errors during data retrieval,

`get_seq_frame_pixeldata`(*frame*, *shape=None*, *preview=False*, *linked=False*)

Retrieves the pixel data from a frame in the sequence currently loaded in Siril.

**Parameters**

- **frame** (`int`) -- selects the frame to retrieve pixel data from. This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **shape** (`Optional[List[int]]`) -- Optional list of [x, y, w, h] specifying the region to retrieve. If provided, gets pixeldata for just that region. If None, gets pixeldata for the entire image.
- **preview** (`Optional[bool]`) -- optional bool specifying whether to get pixeldata as a preview (i.e. 8-bit autostretched data) or as real image data. Defaults to False (i.e. real image data).
- **linked** (`Optional[bool]`) -- optional bool specifying whether the autostretched preview should be linked or unlinked. This option is ignored if preview is not True

**Returns**
The image data as a numpy array

**Return type**
numpy.ndarray

**Raises**

- **ValueError** -- If an invalid shape is provided,
- *DataError* -- if the array cannot be reshaped to the correct dimensions,

- *SirilError* -- For other errors during pixel data retrieval.

**get_seq_imgdata**(*frame*)

Request sequence frame metadata from Siril.

**Parameters**

**frame** (int) -- Integer specifying which frame in the sequence to get image metadata for (between 0 and Sequence.number). This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.

**Return type**

Optional[*ImgData*]

**Returns**

ImgData object containing the frame metadata, or None if an error occurred

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

**get_seq_regdata**(*frame*, *channel*)

Request sequence frame registration data from Siril.

**Parameters**

- **frame** (int) -- Integer specifying which frame in the sequence to get registration data for (between 0 and Sequence.number). This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **channel** (int) -- Integer specifying which channel to get registration data for (typically 0, 1, or 2)

**Return type**

Optional[*RegData*]

**Returns**

RegData object containing the registration data, or None if no registration data is available for the specified frame and channel

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

**get_seq_stats**(*frame*, *channel*)

Request sequence frame statistics from Siril.

**Parameters**

- **frame** (int) -- Integer specifying which frame in the sequence to get statistics data for (between 0 and Sequence.number). This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **channel** (int) -- Integer specifying which channel to get statistics for (typically 0, 1, or 2)

**Return type**

Optional[*ImageStats*]

**Returns**

ImageStats object containing the statistics, or None if an error occurred

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- *SirilError* -- if a decoding error occurs.

**get_siril_active_vport()**

Request the active viewport from Siril.

**Returns**

- sirilpy.SirilVport.RED / sirilpy.SirilVport.MONO
- sirilpy.SirilVport.GREEN,
- sirilpy.SirilVport.BLUE,
- sirilpy.SirilVport.RGB

Note that RED and MONO share the same IntEnum value, so there is no difference between a test for one and the other; the two enum labels are provided solely to aid code legibility.

**Return type**
A SirilVport representing the active vport

**Raises**

- *DataError* -- if no response or an invalid response is received,
- *SirilError* -- if an error occurred.

**get_siril_config**(*group*, *key*)

Request a configuration value from Siril.

**Parameters**

- **group** (`str`) -- Configuration group name,
- **key** (`str`) -- Configuration key name within the group (Available values for group and key can be determined using the "get -A" command)

**Return type**
`Union[bool, int, float, str, List[str], None]`

**Returns**
The configuration value with appropriate Python type, or None if an error occurred.

**Raises**

- *DataError* -- if an unknown config type is encountered,
- *SirilError* -- if an error occurred getting the requested config value

**get_siril_configdir()**

Request the user config directory used by Siril.

**Return type**
`str`

**Returns**
The user config directory as a string.

**Raises**

- *DataError* -- if no response is received,
- *SirilError* -- for all other errors.

**get_siril_display_iccprofile()**

Retrieve the Siril display ICC profile.

Args: none.

**Returns**
The display ICC profile as a byte array, or None if Siril is running in headless mode.

**Return type**
bytes

**Raises**

- *NoImageError* -- If no image is currently loaded,
- *SirilError* -- If any other error occurs.

**get_siril_log()**

Retrieve the full Siril log as a text string.

**Returns**
The text of the Siril log.

**Return type**
str

**Raises**
*SirilError* -- For errors during data retrieval,

**get_siril_panzoom()**

Request the pan and zoom state from Siril.

**Return type**
`Tuple[float, float, float]`

**Returns**
A tuple (x offset, y offset, zoomlevel) representing the current image pan and zoom state.

**Raises**
*SirilError* -- if an error occurred.

**get_siril_selection()**

Request the image selection from Siril.

**Return type**
`Optional[Tuple[int, int, int, int]]`

**Returns**
A tuple (x, y, height, width) representing the current selection, or None if no selection is made.

**Raises**
*SirilError* -- if an error occurred.

**get_siril_slider_state**(*float_range=False*)

Request the display slider state from Siril.

**Return type**
`Tuple[int, int, SlidersMode]`

**Returns**
A tuple (min, max, slider mode) representing the current slider state.

**Raises**
*SirilError* -- if an error occurred.

**get_siril_stf()**

Request the Screen Transfer Function in use in Siril.

**Return type**
*STFType*

**Returns**
A STFType representing the current STF state.

**Raises**
*SirilError* -- if an error occurred.

**get_siril_stf_linked()**

Determine whether the AutoStretch STF is configured to be linked or not

**Return type**
`bool`

**Returns**
A bool representing the current STF channel-linked state.

**Raises**
*SirilError* -- if an error occurred.

**get_siril_systemdatadir()**

Request the system data directory used by Siril.

**Return type**
`Optional[str]`

**Returns**
The system data directory as a string.

**Raises**

- *DataError* -- if no response is received,
- *SirilError* -- for all other errors.

**get_siril_userdatadir()**

Request the user data directory used by Siril.

**Return type**
`str`

**Returns**
The user data directory as a string.

**Raises**

- *DataError* -- if no response is received,
- *SirilError* -- for all other errors.

**get_siril_wd()**

Request the current working directory from Siril.

**Return type**
`str`

**Returns**
The current working directory as a string.

**Raises**

- *DataError* -- if no response was obtained,
- *SirilError* -- for all other errors.

**image_lock()**

A context manager that handles claiming and releasing the processing thread.

This method is designed to be used with a *with* statement to ensure that the thread is properly claimed before processing and released after processing, even if an exception occurs during processing. It is preferable to use this context manager rather than manually calling claim_thread() and release_thread() as the context manager will ensure correct cleanup if an exception occurs.

Note that the image_lock() context should only be entered when the script itself is operating on the Siril image data. If the script is calling a Siril command to alter the Siril image then the context **must not** be entered or the Siril command will be unable to acquire the processing thread and will fail.

Example:

```
try:
    with siril.image_lock():
        # Get image data
        image_data = self.get_image_pixeldata()
        # Process image data
        processed_data = some_processing_function(image_data)
        # Set the processed image data
        siril.set_image_pixeldata(processed_data)
except ProcessingThreadBusyError:
    # Handle busy thread case
    pass
except ImageDialogOpenError:
    # Handle open dialog case
    pass
```

**Raises**

- *ProcessingThreadBusyError* -- If the thread is already in use
- *ImageDialogOpenError* -- If an image processing dialog is open
- *SirilError* -- If the thread cannot be claimed or released for other reasons

**info_messagebox**(*my_string*, *modal=False*)

Send an information message to Siril. The maximum message length is 1022 bytes: longer messages will be truncated. This is intended for displaying informational messages more prominently than using the Siril log.

**Parameters**

- **my_string** (str) -- The message to display in the info message box
- **modal** (Optional[bool]) -- Sets whether or not the message box should be modal and wait for completion or non-modal and allow the script to continue execution. Note that although a modal message box will block execution of the script, if a TKinter main loop is running events will continue to queue up, so if the message box is triggered by clicking

a button then the user may click it while the message box is shown and trigger a second message box which will display immediately the first one is closed.

**Returns**
True if the info was successfully displayed, False otherwise

**Return type**
bool

**Raises**
*SirilError* -- if an error occurred.

`is_cli()`

Check if the current instance is running in CLI mode. This method is useful to detect how the script was invoked and whether to show or not a GUI. This is False when the script is called by clicking in the Script menu, True otherwise.

**Returns**
True if running in CLI mode, False otherwise.

**Return type**
bool

`is_image_loaded()`

Check if a single image is loaded in Siril.

**Returns**
True if a single image is loaded, False if a single image is not loaded

**Return type**
bool

**Raises**

- *DataError* -- if no response is received,
- *SirilError* -- for all other errors.

`is_sequence_loaded()`

Check if a sequence is loaded in Siril.

**Returns**
True if a sequence is loaded, False if a sequence is not loaded

**Return type**
bool

**Raises**

- *DataError* -- if no response is received,
- *SirilError* -- for all other errors.

`load_image_from_file(filepath, with_pixels=True, preview=False, linked=False)`

Request Siril to load an image from a file and transfer it to sirilpy. This method does not change the image currently loaded in Siril. Any image format supported by Siril is supported. This may be used as an alternative to loading an image using astropy.io.fits, however perhaps the main benefit to using it is that it supports the preview option which can be used to obtain an 8-bit autostretched rendering of the image more quickly than is possible using astropy and applying an autostretch using numpy.

**Parameters**

- **filepath** (`str`) -- String specifying the path to the image file to load.

- **with_pixels** (Optional[bool]) -- bool specifying whether or not to return the pixel data for the image (default is True).
- **preview** (Optional[bool]) -- bool specifying whether or not to return the real pixel data or an autostretched uint8_t preview version. Only has an effect in conjunction with with_pixels = True
- **linked** (Optional[bool]) -- bool specifying whether the autostretch preview should be linked or not. Has no effect unless preview is True.

**Return type**
Optional[*FFit*]

**Returns**
FFit object containing the image data

**Raises**

- **FileNotFoundError** -- if the specified file does not exist,
- *DataError* -- on receipt of incorrect data,
- *SirilError* -- if an error occurs.

**log**(*my_string*, *color=LogColor.DEFAULT*)

Send a log message to Siril. The maximum message length is 1022 bytes: longer messages will be truncated.

**Parameters**

- **my_string** (str) -- The message to log
- **color** (*LogColor*) -- Defines the text color, defaults to white. See the documentation
- **which** (*for LogColor for an explanation of which colors should be used for*)
- **purposes.**

**Raises**
*SirilError* -- if the command fails

**Return type**
bool

**open_dialog**(*dialog*)

Opens a Siril GUI dialog. Introduced in sirilpy version 1.0.20.

**Parameters**
**dialog** (DialogID) -- Specifies the dialog to open.

**Returns**
True if the dialog-opening action succeeded False if the dialog-opening action failed (e.g. because the criteria were not met)

**Raises**

- **TypeError** -- if the parameter is not a DialogID.
- *SirilError* -- if the method is called headless or an error occurs.

**overlay_add_polygon**(*polygon*)

Adds a user polygon to the Siril display overlay :type polygon: *Polygon* :param polygon: Polygon defining the polygon to be added

**Returns**
the input updated with the ID assigned by Siril

**Return type**
*Polygon*

**Raises**

- **TypeError** -- invalid data provided,
- *SirilConnectionError* -- on a connection failure,
- *DataError* -- on receipt of invalid data,
- *SirilError* -- on any failure

**overlay_clear_polygons()**

Clears all user polygons from the Siril overlay

**Returns**
True if the command succeeded, False otherwise

**Return type**
bool

**Raises**

- *SirilError* -- if an error occurred
- *SharedMemoryError* -- if a shared memory error occurred

**overlay_delete_polygon**(*polygon_id*)

Deletes a single user polygon from the Siril overlay, specified by ID

**Parameters**
**id** -- int specifying the polygon ID to be deleted

**Raises**
*SirilError* -- on failure

**overlay_draw_polygon**(*color=16711744*, *fill=False*)

Enters a mode where the user can draw a Polygon in the Siril window by clicking the main mouse button and dragging. Releasing the mouse button finalises and closes the polygon.

**Parameters**

- **color** -- uint32 specifying packed RGBA values. Default: 0x00FF0040, 75% transparent green)
- **fill** -- bool specifying whether or not to fill the polygon (default: False)

**overlay_get_polygon**(*polygon_id*)

Gets a single user polygon from the Siril overlay, specified by ID

**Parameters**
**id** -- int specifying the polygon ID to be retrieved. The special ID -1 will

retrieve the most recently added polygon.

**Returns**
the specified Polygon if it exists, None otherwise

**Return type**
*Polygon*

**Raises**
*SirilError* -- on failure

`overlay_get_polygons_list()`

Gets a List of all user polygons from the Siril overlay

**Returns**
the list of Polygon if some exist, None otherwise

**Return type**
List[*Polygon*]

**Raises**

- *SirilError* -- if an error occurred
- *SharedMemoryError* -- if a shared memory error occurred

`pix2radec(x, y)`

Converts a pair of pixel coordinates into RA and dec coordinates using the WCS of the image loaded in Siril. This requires that an image is loaded in Siril and that it has been platesolved (i.e. it has a WCS solution).

**Parameters**

- **x** (`float`) -- float: provides the x coordinate to be converted
- **y** (`float`) -- float: provides the y coordinate to be converted

**Returns**
(RA, Dec) as a Tuple of two floats.

**Return type**
Tuple[float, float]

**Raises**

- *NoImageError* -- If no image or sequence is loaded,
- **ValueError** -- If the image or loaded sequence frame is not plate solved,
- *SirilError* -- For errors during pix2radec execution.

`radec2pix(ra, dec)`

Converts a pair of RA,dec coordinates into image pixel coordinates using the WCS of the image loaded in Siril. This requires that an image is loaded in Siril and that it has been platesolved (i.e. it has a WCS solution).

**Parameters**

- **ra** (`float`) -- float: provides the RA coordinate to be converted
- **dec** (`float`) -- float: provides the dec coordinate to be converted

**Returns**
[x, y] as a Tuple of two floats.

**Return type**
Tuple[float, float]

**Raises**

- *NoImageError* -- If no image or sequence is loaded,
- **ValueError** -- If the image or loaded sequence frame is not plate solved,

- *SirilError* -- For errors during radec2pix execution.

**redo()**

Redoes the last undone operation, if there is an undo history and an undone operation available to be redone.

**reset_progress()**

Resets the Siril progress bar.

**Parameters**
none

**Raises**
*SirilError* -- For any errors.

**Return type**
bool

**save_image_file**(*data*, *header=None*, *filename=None*)

Save image pixeldata and metadata to a FITS file. This uses Siril to save the image and can therefore be used to avoid a script dependency on astropy if it is only required for saving an image. This allows images to be saved directly from python without having to use set_image_pixeldata() and set_image_metadata_from_header_string() and saving the Siril image. It support processing multiple images without affecting the image currently loaded in Siril.

**Parameters**

- **data** -- Either a numpy.ndarray containing the image data (must be 2D or 3D array with dtype float32 or uint16), OR a FFit object containing both data and header.
- **header** -- string containing the FITS header data. Required if data is a numpy array, ignored if data is a FFit object.
- **filename** -- string containing the path where the file should be saved. Required if data is a numpy array. If data is a FFit object and filename is None, will use fit.filename.

**Returns**
True if successful, False otherwise

**Return type**
bool

**Raises**

- **ValueError** -- if the input array or header is invalid,
- **TypeError** -- if invalid parameter types are provided,
- *SirilError* -- if there was an error in handling the command.

### Examples

# Using numpy array and header string siril.save_image_file(data_array, header_string, "output.fit")

# Using FFit object siril.save_image_file(fit, filename="output.fit")

# Using FFit object with its own filename siril.save_image_file(fit)

**set_image_bgsamples**(*points*, *show_samples=False*, *recalculate=True*)

Serialize a set of background sample points and send via shared memory. Points can be provided either as: - List of (x,y) Tuples: BGSamples are created with these positions and Siril will automatically compute the statistics. - List of BGSample objects: The complete sample data is sent to Siril. By default Siril will recalculate statistics for the sample points on receipt, but this can be overridden with the argument recalculate=False

> **Note**
>
> Siril 1.4.x does not require the processing thread to be claimed for this call, but from Siril 1.5.x onwards the request is refused unless the thread is claimed. It is good practice to call this method inside a `with image_lock():` block (or between `claim_thread()` and `release_thread()`) so that scripts remain forward compatible with 1.5.x and later.

**Parameters**

- **points** (Union[List[Tuple[float, float]], List[*BGSample*]]) -- List of sample points, either as (x,y) tuples or BGSample objects
- **show_samples** (bool) -- Whether to show the sample points in Siril
- **recalculate** -- Whether to recalculate the sample points once set. This only applies if the sample points are provided as a List of BGSamples, in which case it defaults to True. If the sample points are provided as a List of (x,y) Tuples then the parameter has no effect. Setting recalculate=False is usually a bad idea but the option is provided to support potential advanced uses where the values are adjusted in python code to manipulate the background fit.

Returns: True if the command succeeded, otherwise False

**Raises**

- *NoImageError* -- if no image is loaded in Siril,
- **ValueError** -- if samples do not have valid positions,
- *SirilError* -- if there was a Siril error in handling the command.

**set_image_filename**(*filename*)

Set the image filename in Siril. :type filename: `str` :param filename: the image filename to set :type filename: str

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the filename was set successfully

**Return type**
bool

**set_image_iccprofile**(*iccprofile*)

Set the image ICC profile in Siril

> **Note**
>
> Siril 1.4.x does not require the processing thread to be claimed for this call, but from Siril 1.5.x onwards the request is refused unless the thread is claimed. It is good practice to call this method inside a `with image_lock():` block (or between `claim_thread()` and `release_thread()`) so that scripts remain forward compatible with 1.5.x and later.

**Parameters**

- **iccprofile** (*bytes*) -- The ICC profile to send to Siril. This will
- **profile** (*replace an existing ICC*)
- **None** (*if one is set. If*)

:param : :param any existing image ICC profile will be removed.:

Returns: True if the command succeeded, otherwise False

**Raises**

- ***NoImageError*** -- if no image is loaded in Siril,
- **ValueError** -- if iccprofile is not valid type,
- ***SirilError*** -- if there was a Siril error in handling the command.

**set_image_metadata_from_header_string**(*header*)

Send image metadata to Siril from a FITS header. The header can be obtained from a sirilpy FFit.header or alternatively from a FITS file obened from disk using astropy.fits.

Example:

```
hdul = fits.open('your_fits_file.fits')
# Get the header from the primary HDU (or any other HDU you want)
header = hdul[0].header
# Convert the header to string
header_string = header.tostring(sep='\\n')
# Send the metadata to Siril
siril.set_image_metadata_from_header_string(header_string)
```

> **Note**
>
> Siril 1.4.x does not require the processing thread to be claimed for this call, but from Siril 1.5.x onwards the request is refused unless the thread is claimed. It is good practice to call this method inside a `with image_lock():` block (or between `claim_thread()` and `release_thread()`) so that scripts remain forward compatible with 1.5.x and later.

**Parameters**

**header** (`str`) -- string containing the FITS header data

**Returns**

True if successful, False otherwise

**Return type**

bool

**Raises**

- **TypeError** -- invalid parameter provided,
- ***NoImageError*** -- if no image is loaded in Siril,
- ***SirilError*** -- if an error occurs.

**set_image_pixeldata**(*image_data*)

Send image data to Siril using shared memory.

> **Note**
>
> This method modifies the loaded image and requires the processing thread to be claimed: call it inside a `with image_lock():` block (or between `claim_thread()` and `release_thread()`), otherwise Siril will refuse the request with an error.

**Parameters**

**image_data** (ndarray) -- numpy.ndarray containing the image data. Must be 2D (single channel) or 3D (multi-channel) array with dtype either np.float32 or np.uint16.

**Raises**

- *NoImageError* -- if no image is loaded in Siril,
- **ValueError** -- if the input array is invalid,
- *SirilError* -- if there was an error in handling the command.

**Return type**

bool

**set_seq_frame_incl**(*index*, *incl*)

Set whether given frame(s) are included in the currently loaded sequence in Siril. This method is intended for use in creating custom sequence filters.

**Parameters**

- **index** (Union[int, List[int]]) -- integer or list of integers specifying which frame(s) to set the inclusion status for. This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame number 1. Passing a list is available since sirilpy 1.0.17
- **incl** (bool) -- bool specifying whether the frame(s) are included or not.

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- **TypeError** -- if index is not an int or list of ints,
- *SirilError* -- on failure.

**set_seq_frame_pixeldata**(*index*, *image_data*, *prefix*)

Send sequence frame image data to Siril using shared memory. Note that this method only works with sequences of FITS images: it does **not** work with FITSEQ, SER or AVI single-file sequences. The image_lock() context manager is not required in order to use this method.

**Parameters**

- **index** (int) -- integer specifying which frame to set the pixeldata for. This uses a 0-based indexing scheme, i.e. the first frame is frame number 0, not frame numer 1.
- **image_data** (ndarray) -- numpy.ndarray containing the image data. Must be 2D (single channel) or 3D (multi-channel) array with dtype either np.float32 or np.uint16.
- **prefix** (str) -- String prefix to use when saving the file to make a new sequence. Note that saving sequence frames with a new prefix does not by itself create a new sequence: once all the frames have been saved with the new sequence prefix, `sirilpy.SirilInterface.create_new_seq()` must be called to create the actual sequence file. Note that while it is permitted to pass prefix=None, this will overwrite the existing sequence and is not typically what is wanted, therefore the parameter is not optional and must be passed explicitly.

**Raises**

- *NoSequenceError* -- if no sequence is loaded in Siril,
- **ValueError** -- if the input array is invalid,
- *SirilError* -- if there was an error in handling the command.

**Return type**
bool

**set_siril_pan**(*xoff*, *yoff*)

Set the display offset (pan) in Siril to (xoff, yoff). :type xoff: float :param xoff: x display offset :type xoff: float :type yoff: float :param yoff: y display offset :type yoff: float

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the display offset was set successfully

**Return type**
bool

**set_siril_selection**(*x=None*, *y=None*, *w=None*, *h=None*, *selection=None*)

Set the image selection in Siril using the provided coordinates and dimensions.

**Parameters**

- **x** (Optional[int]) -- X-coordinate of the selection's top-left corner (must be provided with y, w, h)
- **y** (Optional[int]) -- Y-coordinate of the selection's top-left corner (must be provided with x, w, h)
- **w** (Optional[int]) -- Width of the selection (must be provided with x, y, h)
- **h** (Optional[int]) -- Height of the selection (must be provided with x, y, w)
- **selection** (Optional[Tuple[int, int, int, int]]) -- A tuple of (x, y, w, h) as returned by get_siril_selection()

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the selection was set successfully

**Return type**
bool

**set_siril_slider_lohi**(*lo=None*, *hi=None*)

Set the slider values in Siril using the provided lo and hi values. If the sliders mode is not already set to USER, it is set to that mode as setting slider values is only relevant in that mode. :type lo: Union[float, int] :param lo: Low value for the slider (float [0,1] or uint16) :type hi: Union[float, int] :param hi: High value for the slider (float [0,1] or uint16)

**Raises**

- *SirilError* -- if an error occurred.

- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the slider state was set successfully

**Return type**
bool

**set_siril_slider_mode**(*mode*)

Set the slider state in Siril using the provided lo, hi and mode values. :type mode: *SlidersMode* :param mode: SlidersMode enum value

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the slider state was set successfully

**Return type**
bool

**set_siril_stf**(*mode*)

Set the screen transfer function in Siril using the provided enum value. :type mode: *SlidersMode* :param mode: STFType enum value

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the slider state was set successfully

**Return type**
bool

**set_siril_stf_linked**(*state*)

Set the screen transfer function in Siril using the provided enum value. :type state: `bool` :param state: bool

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the slider state was set successfully

**Return type**
bool

**set_siril_zoom**(*zoom*)

Set the display zoom. Passing any negative value will set 'zoom to fit'. :type zoom: `float` :param zoom: zoom level :type zoom: float

**Raises**

- *SirilError* -- if an error occurred.
- **ValueError** -- if parameters are not properly provided.

**Returns**
True if the display offset was set successfully

**Return type**
bool

`undo()`

Undoes the last operation, if there is an undo history available.

`undo_save_state(my_string)`

Saves an undo state. The maximum message length is 70 bytes: longer messages will be truncated. Requires a single image to be loaded.

**Parameters**
**my_string** (`str`) -- The message to log in FITS HISTORY

**Returns**
True if the message was successfully logged, False if a logging error occurred or None if the script is headless (undo does not happen when headless)

**Return type**
Optional[bool]

**Raises**
*SirilError* -- if an error occurred.

`update_progress(message, progress)`

Send a progress update to Siril with a message and completion percentage.

**Parameters**

- **message** (`str`) -- Status message to display,
- **progress** (`float`) -- Progress value in the range 0.0 to 1.0. The following special values can be used: -1.0 will pulsate the progress bar, and -2.0 will update the progress bar text but will not update the progress shown in the bar.

**Raises**

- **ValueError** -- If the progress argument is out of range,
- *SirilError* -- For any other errors.

**Return type**
`bool`

`warning_messagebox(my_string, modal=False)`

Send a warning message to Siril. The maximum message length is 1022 bytes: longer messages will be truncated. This is intended for displaying warning messages more prominently than using the Siril log.

**Parameters**

- **my_string** (`str`) -- The message to display in the warning message box
- **modal** (`Optional[bool]`) -- Sets whether or not the message box should be modal and wait for completion or non-modal and allow the script to continue execution. Note that although a modal message box will block execution of the script, if a TKinter main loop is running events will continue to queue up, so if the message box is triggered by clicking a button then the user may click it while the message box is shown and trigger a second message box which will display immediately the first one is closed.

**Returns**
True if the warning was successfully displayed, False otherwise

**Return type**
bool

**Raises**
*SirilError* -- if an error occurred.

**xy_plot**(*plot_data*, *display=True*, *save=False*)

Serialize plot data and send via shared memory. See the sirilpy.plot submodule documentation for how to configure a PlotData object for use with SirilInterface.xy_plot()

**Parameters**

- **plot_metadata** -- PlotMetadata object containing plot configuration
- **display** -- bool indicating whether to display the plot on screen (defaults to True)
- **save** -- bool indicating whether to save to the file specified in PlotData.savename (defaults to False)

**Raises**

- *DataError* -- if invalid xy_plot data is received via shared memory,
- *SirilError* -- If an error occurs.

## 25.4 Sirilpy Data Models

This submodule provides dataclasses to represent the main Siril data structures. Most dataclasses have corresponding deserialization methods that are used by the SirilInterface methods. All its members are made available at the module root level, there is no need to import models separately.

**class** sirilpy.models.**BGSample**(*x=None*, *y=None*, *position=None*, *size=25*, ***kwargs*)

Bases: object

Python equivalent of the Siril background_sample struct. Used to hold background sample data obtained from Siril, or to generate or modify background sample data to set in Siril. A BGSample can be constructed as: - s1 = BGSample(x=1.0, y=2.0) - s2 = BGSample(position=(1.0, 2.0)) - s3 = BGSample(x=1.0, y=2.0, mean=0.5, size=31)

**classmethod deserialize**(*data*)

Deserialize a portion of a buffer into a BGSample object

**Parameters**
**data** (*bytes*) -- The full binary buffer containing BGSample data

**Returns**
A BGSample object

**Return type**
*BGSample*

**Raises**

- **ValueError** -- If the buffer slice size does not match the expected size.
- **struct.error** -- If there is an error unpacking the binary data.

**max: float = 0.0**

**mean: float = 0.0**

**median: Tuple[float, float, float] = (0.0, 0.0, 0.0)**

Median values for R, G and B channels. For mono images only median[0] is used.

**min: float = 0.0**

**position: Tuple[float, float] | None = None**

Position in (x, y) image coordinates

**size: int = 25**

The default size matches the size of Siril bg samples.

**valid: bool = True**

Samples default to being valid

**class** sirilpy.models.**DistoData**(*index=DistoType.DISTO_UNDEF*, *filename=''*, *velocity=(0, 0)*)

Bases: object

Python equivalent of Siril disto_params structure

**filename: str = ''**

filename if DISTO_FILE or DISTO_MASTER (and optional for DISTO_FILE_COMET)

**index: *DistoType* = 0**

Specifies the distrosion type

**velocity: Tuple[float, float] = (0, 0)**

shift velocity if DISTO_FILE_COMET

**class** sirilpy.models.**FFit**(*bitpix=None*, *orig_bitpix=None*, *naxis=0*, *_naxes=(0, 0, 0)*, *keywords=<factory>*, *checksum=False*, *header=None*, *unknown_keys=None*, *stats=<factory>*, *mini=0.0*, *maxi=0.0*, *neg_ratio=0.0*, *_data=None*, *top_down=False*, *_focalkey=False*, *_pixelkey=False*, *history=<factory>*, *color_managed=False*, *_icc_profile=None*)

Bases: object

Python equivalent of Siril ffit (FITS) structure, holding image pixel data and metadata.

**allocate_data**()

Allocate memory for image data with appropriate type. self.width, self.height, self.naxis, self.naxes and self.dtype must be set before calling this method.

**Raises**

**ValueError** -- if self.bitpix is not set to BitpixType.USHORT_IMG or BitpixType.FLOAT_IMG

**ensure_data_type**(*target_type=None*)

Ensure data is in the correct type with proper scaling

**Parameters**

**target_type** -- Optional np.dtype to convert to. If None, uses self.type

**Raises**

**ValueError** -- if the conversion is between data types that are not internally used by Siril for calculation

**estimate_noise**(*array*, *nullcheck=True*, *nullvalue=0.0*)

Estimate the background noise in the input image using the sigma of first-order differences.

noise = 1.0 / sqrt(2) * RMS of (flux[i] - flux[i-1])

**Parameters**

- **array** (*np.ndarray*) -- 2D array of image pixels (np.uint16 or np.float32).
- **nullcheck** (*bool*) -- If True, check for null values.
- **nullvalue** (Optional[float]) -- The value of null pixels (only used if nullcheck is True).

**Returns**
Estimated noise value.

**Return type**
float

**Raises**
**ValueError** -- if the array is the wrong shape

**get_channel**(*channel*)

Get a specific channel of the pixel data. Note that this does not pull pixel data directly from the image loaded in Siril: that must previously have been obtained using get_image_pixeldata() or get_image()

**Return type**
ndarray

**update_stats**()

Update image statistics for all channels. Note that this only updates the statistics based on the NumPy array representing pixel data in the python FFit object, it does not update the statistics of the image in Siril.

**bitpix: *BitpixType* | None = None**

FITS header specification of the image data type.

**property channels: int**

Image channels

**checksum: bool = False**

Whether Siril will write FITS data checksums for this file.

**color_managed: bool = False**

Specifies whether the image is color managed or not.

**property data: ndarray | None**

The pixel data of the current image loaded in Siril, stored as a NumPy array

**header: str | None = None**

The FITS header as a string.

**property height: int**

Image height

**history: list[str]**

Contains a list of strings holding the HISTORY entries for this image.

**property icc_profile: bytes | None**

The ICC profile as raw bytes data. This may be converted for use by modules such as pillow which can handle ICC profiles.

**keywords: *FKeywords***

A FKeywords object containing FITS header keywords.

**maxi: float = 0.0**

The maximum value across all image channels.

**mini: float = 0.0**

The minimum value across all image channels.

**property naxes: Tuple[int, int, int]**

The naxes tuple holds the image dimensions as width x height x channels. Note that the axis ordering differs between Siril representation as held in naxes and numpy representation as held in _data.shape (which is channels x height x width)

**naxis: int = 0**

The number of axes (2 for a mono image, 3 for a RGB image). Corresponds to the FITS kwyword NAXIS.

**neg_ratio: float32 = 0.0**

The ratio of negative pixels to the total pixels.

**orig_bitpix: *BitpixType* | None = None**

FITS header specification of the original image data type.

**stats: List[*ImageStats* | None]**

A list of ImageStats objects, one for each channel.

**top_down: bool = False**

Specifies the ROWORDER for this image. The FITS specification directs that FITS should be stored bottom-up, but many CMOS sensors are natively TOP_DOWN and capture software tends to save FITS images captured by these sensors as TOP_DOWN.

**unknown_keys: str | None = None**

All unknown FITS header keys as a string. This gives access to header cards that Siril does not use internally.

**property width: int**

Image width

**class** sirilpy.models.**FKeywords**(*bscale=1.0, bzero=0.0, lo=0, hi=0, flo=0.0, fhi=0.0, program='', filename='', row_order='', filter='', image_type='', object='', instrume='', telescop='', observer='', bayer_pattern='', sitelat_str='', sitelong_str='', focname='', date=None, date_obs=None, data_max=0.0, data_min=0.0, pixel_size_x=0.0, pixel_size_y=0.0, binning_x=1, binning_y=1, expstart=0.0, expend=0.0, bayer_xoffset=0, bayer_yoffset=0, airmass=1.0, focal_length=0.0, flength=0.0, iso_speed=0.0, exposure=0.0, aperture=0.0, ccd_temp=0.0, set_temp=0.0, livetime=0.0, stackcnt=0, cvf=0.0, gain=0, offset=0, focuspos=0, focussz=0, foctemp=0.0, centalt=0.0, centaz=0.0, sitelat=0.0, sitelong=0.0, siteelev=0.0, objctra=None, objctdec=None, ra=None, dec=None, pltsolvd=False, pltsolvd_comment=None*)

Bases: *object*

Python equivalent of Siril fkeywords structure. Contains the FITS header keyword values converted to suitable data types.

**classmethod deserialize**(*data*)

Deserialize binary keyword-block (the block produced by keywords_to_py()) into an FKeywords object. Raises ValueError on size mismatch and SirilError for other unpacking issues.

**Return type**
*FKeywords*

**FLEN_VALUE: ClassVar[int] = 71**

```
KEYWORDS_FORMAT: ClassVar[str] = '!
71s71s71s71s71s71s71s71s71s71s71s71s71s71sddQQddddddQQddddddddqqddddddddddQdqqqqdqqdd?
'
```

```
KEYWORDS_SIZE: ClassVar[int] = 1465
```

**airmass: float = 1.0**

Airmass at frame center (Gueymard 1993)

**aperture: float = 0.0**

Aperture value as a float

**bayer_pattern: str = ''**

Bayer color pattern

**bayer_xoffset: int = 0**

X offset of the Bayer pattern

**bayer_yoffset: int = 0**

Y offset of the Bayer pattern

**binning_x: int = 1**

XBINNING FITS header card as an int

**binning_y: int = 1**

YBINNING FITS header card as an int

**bscale: float = 1.0**

Offset data range to that of unsigned short

**bzero: float = 0.0**

Default scaling factor

**ccd_temp: float = 0.0**

CCD temperature as a float

**centalt: float = 0.0**

[deg] Altitude of telescope

**centaz: float = 0.0**

[deg] Azimuth of telescope

**cvf: float = 0.0**

Conversion factor (e- / ADU)

**data_max: float = 0.0**

used to check if 32b float is in the [0, 1] range

**data_min: float = 0.0**

used to check if 32b float is in the [0, 1] range

**date: datetime | None = None**

UTC date that FITS file was created

**date_obs: datetime | None = None**

ss observation start, UT

**Type**

YYYY-MM-DDThh

**Type**
mm

**dec: float | None = None**
Dec as float, if available

**expend: float = 0.0**
Exposure end as a Julian date

**exposure: float = 0.0**
Exposure time as a float (s)

**expstart: float = 0.0**
Exposure start as a Julian date

**fhi: float32 = 0.0**
MIPS-Hi key in FITS file, "Upper visualization cutoff (float)"

**filename: str = ''**
Original Filename

**filter: str = ''**
Active filter name

**flength: float = 0.0**
[mm] Focal length

**flo: float32 = 0.0**
MIPS-LO key in FITS file, "Lower visualization cutoff (float)"

**focal_length: float = 0.0**
[mm] Focal length

**focname: str = ''**
Focusing equipment name

**foctemp: float = 0.0**
Focuser temperature

**focuspos: int = 0**
Focuser position

**focussz: int = 0**
[um] Focuser step size

**gain: int = 0**
Gain factor read in camera

**hi: int = 0**
MIPS-HI key in FITS file, "Upper visualization cutoff"

**image_type: str = ''**
Type of image

**instrume: str = ''**
Instrument name

**iso_speed: float = 0.0**
ISO speed value as a float

**livetime: float = 0.0**

Sum of exposure times (s)

**lo: int = 0**

MIPS-LO key in FITS file, "Lower visualization cutoff"

**objctdec: str | None = None**

object Dec as string, if available

**objctra: str | None = None**

object RA as string, if available

**object: str = ''**

Name of the object of interest

**observer: str = ''**

Observer name

**offset: int = 0**

Offset value read in camera

**pixel_size_x: float = 0.0**

XPIXSZ FITS header card as a float

**pixel_size_y: float = 0.0**

YPIXSZ FITS header card as a float

**pltsolvd: bool = False**

whether plate solved or not

**pltsolvd_comment: str | None = None**

plate solution comment

**program: str = ''**

Software that created this HDU

**ra: float | None = None**

RA as float, if available

**row_order: str = ''**

Order of the rows in image array

**set_temp: float = 0.0**

CCD set temperature as a float

**siteelev: float = 0.0**

[m] Observation site elevation

**sitelat: float = 0.0**

**sitelat_str: str = ''**

**sitelong: float = 0.0**

[deg] Observation site longitude

**sitelong_str: str = ''**

**stackcnt: int = 0**

Number of stacked frames

**telescop: str = ''**

Telescope used to acquire this image

**class** sirilpy.models.**FPoint**(*x*, *y*)

Bases: object

Represents a 2D point with float x and y coordinate values in the Siril image.

**x: float**

x co-ordinate

**y: float**

y co-ordinate

**class** sirilpy.models.**Homography**(*h00=0.0*, *h01=0.0*, *h02=0.0*, *h10=0.0*, *h11=0.0*, *h12=0.0*, *h20=0.0*, *h21=0.0*, *h22=0.0*, *pair_matched=0*, *Inliers=0*)

Bases: object

Python equivalent of the Siril Homography structure. Contains coefficients for the Homography matrix that maps a sequence frame onto the reference frame.

**Inliers: int = 0**

number of inliers kept after RANSAC step

**h00: float = 0.0**

Homography matrix H00

**h01: float = 0.0**

Homography matrix H01

**h02: float = 0.0**

Homography matrix H02

**h10: float = 0.0**

Homography matrix H10

**h11: float = 0.0**

Homography matrix H11

**h12: float = 0.0**

Homography matrix H12

**h20: float = 0.0**

Homography matrix H20

**h21: float = 0.0**

Homography matrix H21

**h22: float = 0.0**

Homography matrix H22

**pair_matched: int = 0**

number of pairs matched

**class** sirilpy.models.**ImageAnalysis**(*bgnoise=0.0*, *fwhm=0.0*, *wfwhm=0.0*, *nbstars=0*, *roundness=0.0*, *imagetype=0*, *timestamp=0*, *channels=0*, *height=0*, *width=0*, *filter=''*)

Bases: object

Structure to hold image analysis data, used for culling

**classmethod deserialize**(*data*)

Unpack a network-safe binary struct into an ImageAnalysis instance.

**Return type**
*ImageAnalysis*

**serialize**()

Pack the dataclass into a network-safe binary struct.

**Return type**
bytes

**FLEN: ClassVar[int] = 71**

**bgnoise: float = 0.0**

RMS background noise

**channels: int = 0**

number of channels in the image

**filter: str = ''**

filter name (fixed-length string from C, max 70 chars)

**fwhm: float = 0.0**

Mean fwhm

**height: int = 0**

image height

**imagetype: *ImageType* = 0**

Image type enum

**nbstars: int = 0**

Number of stars detected

**roundness: float = 0.0**

Mean star roundness

**timestamp: int = 0**

00 UTC)

**Type**
UNIX timestamp (64-bit seconds since 1970/1/1 00

**wfwhm: float = 0.0**

Mean weighted fwhm

**width: int = 0**

image width

**class** sirilpy.models.**ImageStats**(*total=0*, *ngoodpix=0*, *mean=0.0*, *median=0.0*, *sigma=0.0*, *avgDev=0.0*, *mad=0.0*, *sqrtbwmv=0.0*, *location=0.0*, *scale=0.0*, *min=0.0*, *max=0.0*, *normValue=0.0*, *bgnoise=0.0*)

Bases: object

Python equivalent of Siril imstats structure. Contains statistics for a particular channel of a Siril image.

**classmethod deserialize**(*data*)

Deserialize binary data into an ImageStats object.

**Parameters**
**data** (bytes) -- (bytes) Binary data to unpack

**Return type**
*ImageStats*

Returns: ImageStats object

**Raises: SirilError: If received data size is incorrect**
struct.error: If unpacking fails

**avgDev: float = 0.0**

average deviation of pixels

**bgnoise: float = 0.0**

RMS background noise

**location: float = 0.0**

location of pixel values

**mad: float = 0.0**

mean average deviation of pixels

**max: float = 0.0**

maximum pixel value

**mean: float = 0.0**

mean value of pixels

**median: float = 0.0**

median value of pixels

**min: float = 0.0**

minimum pixel value

**ngoodpix: int = 0**

number of non-zero pixels

**normValue: float = 0.0**

norm value of the pixels

**scale: float = 0.0**

scale value of the pixels

**sigma: float = 0.0**

standard deviation of pixels

**sqrtbwmv: float = 0.0**

square root of the biweight midvariance of pixel values

**total: int = 0**

total number of pixels

**class** sirilpy.models.**ImgData**(*filenum=0*, *incl=False*, *date_obs=None*, *airmass=0.0*, *rx=0*, *ry=0*)

Bases: object

Python equivalent of Siril imgdata structure

**classmethod deserialize**(*response*)

Deserialize binary response into an ImgData object.

**Parameters**
**response** (*bytes*) -- Binary data to unpack.

**Returns**
An ImgData object with deserialized data.

**Return type**
*ImgData*

**Raises**

- **ValueError** -- If received data size is incorrect.
- **struct.error** -- If unpacking fails.

**airmass: float = 0.0**

airmass of the image

**date_obs: datetime | None = None**

date of the observation

**filenum: int = 0**

real file index in the sequence

**incl: bool = False**

selected in the sequence

**rx: int = 0**

width

**ry: int = 0**

height

**class** sirilpy.models.**PSFStar**(*star_name=None*, *B=0.0*, *A=0.0*, *x0=0.0*, *y0=0.0*, *sx=0.0*, *sy=0.0*, *fwhmx=0.0*, *fwhmy=0.0*, *fwhmx_arcsec=0.0*, *fwhmy_arcsec=0.0*, *angle=0.0*, *rmse=0.0*, *sat=0.0*, *R=0*, *has_saturated=False*, *beta=0.0*, *profile=StarProfile.GAUSSIAN*, *xpos=0.0*, *ypos=0.0*, *mag=0.0*, *Bmag=0.0*, *s_mag=999.99*, *s_Bmag=999.99*, *SNR=0.0*, *phot_is_valid=False*, *BV=0.0*, *B_err=0.0*, *A_err=0.0*, *x_err=0.0*, *y_err=0.0*, *sx_err=0.0*, *sy_err=0.0*, *ang_err=0.0*, *beta_err=0.0*, *layer=0*, *units=None*, *ra=0.0*, *dec=0.0*)

Bases: object

Python equivalent of the Siril fwhm_struct structure. Contains data on a modelled fit to a star identified in the image.

**classmethod deserialize**(*data*)

Deserialize a portion of a buffer into a PSFStar object.

**Parameters**
**data** (bytes) -- (bytes) The full binary buffer containing PSFStar data.

**Return type**
*PSFStar*

**Returns**
PSFStar object

**Raises**

- **ValueError** -- If the buffer slice size does not match the expected size.
- **struct.error** -- If there is an error unpacking the binary data.

**A: float = 0.0**

amplitude

**A_err: float = 0.0**

error in A

**B: float = 0.0**

average sky background value

**BV: float = 0.0**

only used to pass data in photometric color calibration

**B_err: float = 0.0**

error in B

**Bmag: float = 0.0**

B magnitude

**R: int = 0**

Optimized box size to enclose sufficient pixels in the background

**SNR: float = 0.0**

SNR of the star

**ang_err: float = 0.0**

error in angle

**angle: float = 0.0**

angle of the x and yaxes with respect to the image x and y axes

**beta: float = 0.0**

Moffat equation beta parameter

**beta_err: float = 0.0**

error in beta

**dec: float = 0.0**

Declination

**fwhmx: float = 0.0**

FWHM in x axis in pixels

**fwhmx_arcsec: float = 0.0**

FWHM in x axis in arc seconds

**fwhmy: float = 0.0**

FWHM in y axis in pixels

**fwhmy_arcsec: float = 0.0**

FWHM in y axis in arc seconds

**has_saturated: bool = False**

Shows whether the star is saturated or not

**layer: int = 0**

image channel on which the star modelling was carried out

**mag: float = 0.0**

(V) magnitude, approximate or accurate

**phot_is_valid: bool = False**

True if mag, Bmag, s_mag, s_Bmag and SNR are photometrically accurate with no saturated pixels. Introduced in sirilpy 1.0.4.

**profile: *StarProfile* = 0**

**ra: float = 0.0**

Right Ascension

**rmse: float = 0.0**

RMSE of the minimization

**s_Bmag: float = 999.99**

error on the B magnitude

**s_mag: float = 999.99**

error on the (V) magnitude

**sat: float = 0.0**

Level above which pixels have satured

**star_name: str | None = None**

**sx: float = 0.0**

Size of the fitted function on the x axis in PSF coordinates

**sx_err: float = 0.0**

error in sx

**sy: float = 0.0**

Size of the fitted function on the y axis in PSF coordinates

**sy_err: float = 0.0**

error in sy

**units: str | None = None**

Units

**x0: float = 0.0**

x coordinate of the peak

**x_err: float = 0.0**

error in x

**xpos: float = 0.0**

x position of the star in the image

**y0: float = 0.0**

y coordinate of the peak

**y_err: float = 0.0**

error in y

**ypos: float = 0.0**

y position of the star in the image

**class** sirilpy.models.**Polygon**(*points*, *polygon_id=0*, *color=4294967295*, *fill=False*, *legend=None*)

Bases: `object`

Represents a user-defined polygon. These can be filled or outline-only, and can have any color and transparency (alpha) value. They can also have an optional label which is displayed centred on the polygon.

Note that Polygons should be considered transitory if used with the overlay - they can be used to display information to the user but they may be cleared at any time if the user toggles the overlay button in the main Siril interface to clear the overlay.

**classmethod deserialize_polygon**(*data*)

Creates a Polygon object by deserializing a byte array.

**Returns**

**A Polygon object and any remaining bytes in the byte** array. (The remaining bytes are for use in deserialize_polygon_list and can be safely ignored if deserializing a single polygon.)

**Return type**
Tuple

**Raises**
**ValueError** -- If there is insufficient data to deserialize.

**classmethod deserialize_polygon_list**(*data*)

Creates a List of Polygon objects by deserializing a byte array.

**Returns**
A List of Polygon objects.

**Return type**
List

**Raises**
**ValueError** -- If there is invalid data to deserialize.

**classmethod from_rectangle**(*rect*, ***kwargs*)

Create a Polygon from a rectangle of the kind returned by sirilpy.connection.get_siril_selection().

**Parameters**

- **rect** (*Tuple[int, int, int, int]*) -- Rectangle as (x, y, width, height)
- ****kwargs** -- Additional keyword arguments to pass to Polygon constructor (polygon_id, color, fill, legend)

**Returns**
A new Polygon instance representing the rectangle

**Return type**
*Polygon*

**contains_point**(*x*, *y*)

Determine if a point is inside the polygon using Dan Sunday's optimized winding number algorithm.

This algorithm is more robust than ray casting for complex polygons and handles edge cases better, including points on edges and self-intersecting polygons.

**Parameters**

- **x** (*float*) -- X coordinate of the point to test.
- **y** (*float*) -- Y coordinate of the point to test.

**Returns**
True if the point is inside the polygon, False otherwise.

**Return type**
bool

`get_bounds()`

Get the bounding box of the polygon.

**Returns**
(min_x, min_y, max_x, max_y)

**Return type**
Tuple[float, float, float, float]

**Raises**
`ValueError` -- If the polygon has no points.

`get_max_x()`

Get the maximum x coordinate of the polygon.

**Return type**
`float`

`get_max_y()`

Get the maximum y coordinate of the polygon.

**Return type**
`float`

`get_min_x()`

Get the minimum x coordinate of the polygon.

**Return type**
`float`

`get_min_y()`

Get the minimum y coordinate of the polygon.

**Return type**
`float`

`serialize()`

Serializes a single Polygon object into a byte array.

**Returns**
A byte array representing the serialized polygon data.

**Return type**
bytes

**Raises**
`ValueError` -- If the number of points exceeds the allowed limit.

`color: int = 4294967295`

32-bit RGBA color (packed, uint_8 per component. Default value is 0xFFFFFFFF)

**fill: bool = False**

whether or not the polygon should be filled when drawn

**legend: str = None**

an optional legend

**points: List[*FPoint*]**

List of points defining the polygon's shape

**polygon_id: int = 0**

unique identifier

**class** sirilpy.models.**RegData**(*fwhm=0.0*, *weighted_fwhm=0.0*, *roundness=0.0*, *quality=0.0*, *background_lvl=0.0*, *number_of_stars=0*, *H=<factory>*)

Bases: object

Python equivalent of Siril regdata structure

**classmethod deserialize**(*data*)

Deserialize a binary response into a RegData object.

**Parameters**
**data** (*bytes*) -- Binary data to unpack

**Return type**
*RegData*

Returns: RegData object

**Raises: SirilError if the received data doesn't match the expected size'**
struct.error If unpacking fails

**H: *Homography***

Stores a homography matrix describing the affine transform from this frame to the reference frame

**background_lvl: float32 = 0.0**

background level

**fwhm: float = 0.0**

copy of fwhm->fwhmx, used as quality indicator

**number_of_stars: int = 0**

number of stars detected in the image

**quality: float = 0.0**

measure of image quality

**roundness: float32 = 0.0**

fwhm->fwhmy / fwhm->fwhmx, 0 when uninit, ]0, 1] when set

**weighted_fwhm: float32 = 0.0**

used to exclude spurious images

**class** sirilpy.models.**Sequence**(*seqname=''*, *number=0*, *selnum=0*, *fixed=0*, *nb_layers=-1*, *rx=0*, *ry=0*, *is_variable=False*, *bitpix=0*, *reference_image=0*, *imgparam=None*, *regparam=None*, *stats=None*, *distoparam=None*, *beg=0*, *end=0*, *exposure=0.0*, *fz=False*, *type=None*, *cfa_opened_monochrome=False*, *current=0*)

Bases: `object`

Python equivalent of Siril sequ structure

**beg: int = 0**

imgparam[0]->filenum

**bitpix: int = 0**

image pixel format, from fits

**cfa_opened_monochrome: bool = False**

CFA SER opened in monochrome mode

**current: int = 0**

file number currently loaded

**distoparam: List[*DistoData*] = None**

distortion data for the sequence [nb_layers]

**end: int = 0**

imgparam[number-1]->filenum

**exposure: float = 0.0**

exposure of frames

**fixed: int = 0**

fixed length of image index in filename

**fz: bool = False**

whether the file is compressed

**imgparam: List[*ImgData*] = None**

a structure for each image of the sequence [number]

**is_variable: bool = False**

sequence has images of different sizes

**nb_layers: int = -1**

number of layers embedded in each image file

**number: int = 0**

number of images in the sequence

**reference_image: int = 0**

reference image for registration

**regparam: List[List[*RegData*]] = None**

registration parameters for each layer [nb_layers][number]

**rx: int = 0**

first image width

**ry: int = 0**

first image height

**selnum: int = 0**

number of selected images

**seqname: str = ''**

name of the sequence

**stats: List[List[*ImageStats*]] = None**

statistics of the images for each layer [nb_layers][number]

**type: *SequenceType* = None**

the type of sequence

## 25.5 Sirilpy Enums

This module provides all the enums that are used within sirilpy. Enums submodule for Siril. This submodule contains all the enums used within sirilpy.

**class** sirilpy.enums.**BitpixType**(*value*)

Bases: IntEnum

Mimics the Siril bitpix enum. Note that although Siril can handle opening FITS files of any data type, internally it processes images only as USHORT_IMG (uint16) or FLOAT_IMG (float32).

**BYTE_IMG = 8**

**DOUBLE_IMG = -64**

**FLOAT_IMG = -32**

**LONG_IMG = 32**

**SHORT_IMG = 16**

**USHORT_IMG = 20**

**class** sirilpy.enums.**CommandStatus**(*value*)

Bases: IntEnum

Contains Siril command status codes, matching the values returned internally within Siril. These can be used for error handling. CMD_OK and CMD_NO_WAIT are no-error codes; all the other codes represent command errors. These are available through the CommandError exception and may generally be handled without being regarded as fatal to the script.

**CMD_ALLOC_ERROR = 1048576**

**CMD_ARG_ERROR = 32**

**CMD_DIR_NOT_FOUND = 4194304**

**CMD_FILE_NOT_FOUND = 131072**

**CMD_FOR_CFA_IMAGE = 65536**

**CMD_FOR_PLATE_SOLVED = 262144**

**CMD_GENERIC_ERROR = 128**

**CMD_IMAGE_NOT_FOUND = 256**

**CMD_INVALID_IMAGE = 1024**

**CMD_LOAD_IMAGE_FIRST = 2048**

**CMD_NEED_INIT_FIRST = 524288**

**CMD_NOT_FOR_MONO = 16384**

**CMD_NOT_FOR_RGB = 32768**

**CMD_NOT_FOR_SINGLE = 8192**

**CMD_NOT_FOUND = 1**

**CMD_NOT_SCRIPTABLE = 8**

**CMD_NO_CWD = 4**

**CMD_NO_WAIT = 2**

**CMD_OK = 0**

**CMD_ONLY_SINGLE_IMAGE = 4096**

**CMD_SELECTION_ERROR = 64**

**CMD_SEQUENCE_NOT_FOUND = 512**

**CMD_THREAD_RUNNING = 2097152**

**CMD_WRONG_N_ARG = 16**

**class** sirilpy.enums.**DialogID**(*value*)

Bases: IntEnum

**classmethod from_name**(*name*)

Safely create from a str containing a potential dialog name (e.g. "ASINH_DIALOG", returning None if the name no longer exists.

**Return type**
*DialogID* | None

**ABERRATION_DIALOG = 19**

**ABOUT_DIALOG = 0**

**ANNOTATE_DIALOG = 1**

**ASINH_DIALOG = 2**

**ASTROMETRY_DIALOG = 3**

**BACKGROUND_EXTRACTION_DIALOG = 4**

**BINXY_DIALOG = 6**

**CANON_FIXBANDING_DIALOG = 7**

**CCM_DIALOG = 8**

**CLAHE_DIALOG = 9**

**COLOR_CALIBRATION = 10**

**COMPOSITION_DIALOG = 12**

COMPSTARS_DIALOG = 11

COSMETIC_DIALOG = 13

CURVES_DIALOG = 14

CWD_DIALOG = 15

DECONV_DIALOG = 5

DENOISE_DIALOG = 16

DIALOG_FFT = 17

DIALOG_STAR_REMIX = 18

EPF_DIALOG = 20

EXTRACT_CHANNEL_DIALOG = 21

EXTRACT_WAVELETS_LAYERS_DIALOG = 22

FILE_INFORMATION = 23

GHT_DIALOG = 24

HISTOGRAM_DIALOG = 25

ICC_DIALOG = 26

KEYWORDS_DIALOG = 27

LINEARMATCH_DIALOG = 28

MEDIAN_DIALOG = 29

MERGE_CFA_DIALOG = 30

NINA_LIGHT_CURVE = 31

OPEN_DIALOG = 32

PCC_DIALOG = 33

PIXEL_MATH_DIALOG = 34

PREFS_DIALOG = 35

RESAMPLE_DIALOG = 36

RGRADIENT_DIALOG = 37

ROTATION_DIALOG = 38

SATU_DIALOG = 40

SAVEAS_DIALOG = 41

SCNR_DIALOG = 42

SEQLIST_DIALOG = 43

**SPLIT_CFA_DIALOG = 44**

**STARNET_DIALOG = 45**

**STARS_LIST_WINDOW = 46**

**STAT_WINDOW = 47**

**S_PCC_DIALOG = 39**

**UNPURPLE_DIALOG = 48**

**WAVELETS_DIALOG = 49**

**class** sirilpy.enums.**DialogReq**(*value*)

Bases: IntEnum

Represents the requirements to open a dialog represented by a DialogID

**ANY = 1**

Image or sequence loaded

**IMG = 2**

Image loaded

**MONO = 3**

Mono image loaded

**NONE = 4**

No requirement, this action works at any time

**PLTSOLVD = 5**

Image or sequence frame loaded and plate solved

**RGB = 6**

RGB image loaded

**RGBPLTSOLVD = 7**

RGB image or sequence frame loaded and plate solved

**SEQ = 8**

Sequence loaded

**class** sirilpy.enums.**DistoType**(*value*)

Bases: IntEnum

Python equivalent of the Siril disto_source enum

**DISTO_FILE = 2**

Distortion from given file

**DISTO_FILES = 4**

Distortion stored in each file (true only from seq platesolve, even with no distortion, it will be checked upon reloading)

**DISTO_FILE_COMET = 5**

special for cometary alignement, to be detected by apply reg

**DISTO_IMAGE = 1**

Distortion from current image

**DISTO_MASTER = 3**

Distortion from master files

**DISTO_UNDEF = 0**

No distortion

**class** sirilpy.enums.**ImageType**(*value*)

Bases: IntEnum

enum representing image functional types

**BIAS = 4**

**DARK = 2**

**FLAT = 3**

**LIGHT = 1**

**UNKNOWN = 0**

**class** sirilpy.enums.**LogColor**(*value*)

Bases: IntEnum

Defines colors available for use with SirilInterface.log() For consistency LogColor.Default should be used for normal messages, LogColor.Red should be used for error messages, LogColor.Salmon should be used for warning messages, LogColor.Green should be used for completion notifications, and LogColor.Blue should be used for technical messages such as equations, coefficients etc.

**BLUE = 4**

**DEFAULT = 0**

**GREEN = 3**

**RED = 1**

**SALMON = 2**

**class** sirilpy.enums.**PlotType**(*value*)

Bases: IntEnum

Enumeration of available plot types for visualizing data series.

**HYPHENS = 2**

**LINES = 3**

**LINESHYPHENS = 6**

**LINESMARKS = 5**

**LINESPOINTS = 4**

**MARKS = 1**

**POINTS = 0**

**class** sirilpy.enums.**STFType**(*value*)

Bases: IntEnum

enum representing STF types

**ASINH_DISPLAY = 4**

**AUTOSTRETCH_DISPLAY = 5**

**HISTEQ_DISPLAY = 6**

**LINEAR_DISPLAY = 0**

**LOG_DISPLAY = 1**

**SQRT_DISPLAY = 2**

**SQUARED_DISPLAY = 3**

**class** sirilpy.enums.**SequenceType**(*value*)

Bases: IntEnum

Python equivalent of the Siril sequence_type enum

**SEQ_AVI = 3**

**SEQ_FITSEQ = 2**

**SEQ_INTERNAL = 4**

**SEQ_REGULAR = 0**

**SEQ_SER = 1**

**class** sirilpy.enums.**SirilVport**

Bases: object

Defines the Siril viewports

**BLUE = 2**

**GREEN = 1**

**MONO = 0**

**RED = 0**

**RGB = 3**

**class** sirilpy.enums.**SlidersMode**(*value*)

Bases: IntEnum

**MINMAX = 1**

**MIPSLOHI = 0**

**USER = 2**

**class** sirilpy.enums.**StarProfile**(*value*)

Bases: IntEnum

Python equivalent of the Siril starprofile enum. Used to identify the type of fit used to model a star in the image. Note that MOFFAT_FIXED is currently not used in Siril, but is reserved for future use for Moffat stars modelled with a fixed beta parameter

**GAUSSIAN = 0**

**MOFFAT = 1**

**MOFFAT_FIXED = 2**

## 25.6 Sirilpy Plotting

This submodule provides classes and method to access the native Siril plotting functionality. Of course, you can also use matplotlib but this submodule provides access to the same plotting capabilities as used internally within Siril for a more seamless result. All its members are made available at the module root level, there is no need to import models separately.

Once populated, the PlotData object can be plotted using `SirilInterface.xy_plot()`. Plot submodule for Siril, providing classes for plot data representation and serialization. This submodule enables users to create and configure various types of plots with customizable appearance and error bars.

**class** sirilpy.plot.**PlotData**(*title='Data Plot'*, *xlabel='X'*, *ylabel='Y'*, *savename='plot'*, *show_legend=True*, *datamin=None*, *datamax=None*)

Bases: object

Metadata container for plot configuration. The actual series data are held in SeriesData objects and can be added using the Class methods add_series or add_series_obj after initialization of the PlotData.

**Members:**

title: Plot title xlabel: X-axis label ylabel: Y-axis label savename: Save filename (extension is added automatically) show_legend: bool indicating whether to show legend datamin: List [xmin, ymin] forcing the bottom left coordinate to show. If omitted, the range is set to the data range. datamax: List [xmax, ymax] forcing the top right coordinate to show. If omitted, the range is set to the data range.

**classmethod serialize**(*plot_data*)

Serialize plot data for shared memory transfer using network byte order.

**Parameters**

**plot_data** (*PlotData*) -- PlotData object containing plot configuration

**Return type**

Tuple[bytes, int]

**Returns**

Tuple of serialized bytes and total length

**add_series**(*x_coords*, *y_coords*, *label=None*, *plot_type=PlotType.LINES*, *n_error=None*, *p_error=None*)

Add a new series to the plot metadata.

**Returns**

the created SeriesData object for further manipulation if needed.

**Return type**

*SeriesData*

**add_series_obj**(*series*)

Add a pre-created SeriesData object to the plot metadata.

Returns: None

**Return type**
None

**class** sirilpy.plot.**SeriesData**(*x_coords*, *y_coords*, *label=None*, *plot_type=PlotType.LINES*, *n_error=None*, *p_error=None*)

Bases: object

Represents a single data series for plotting.

**Members:**

x_coords: Either a List[float] or a np.ndarray containing the values for the x coordinates for this series

y_coords: Either a List[float] or a np.ndarray containing the values for the y coordinates for this series

label: A str containing a label for the series (shown in the plot legend)

plot_type: a PlotType setting the type of marks to use

n_error: Either a List[float] or a np.ndarray containing values for the y-axis negative errors for this series

p_error: Either a List[float] or a np.ndarray containing values for the y-axis positive errors for this series

## 25.7 Sirilpy Utility Methods

This submodule provides utility methods for use in Siril python scripts. The most important is `ensure_installed()` but there are also methods such as `download_with_progress()` which provides a highly robust method for downloading large files with good error recovery through a retries and resume mechanism. Utility module for Siril Python interface providing helper functions for file operations, package management, and standard I/O control to support Siril's scripting capabilities.

**class** sirilpy.utility.**SuppressedStderr**

Bases: object

This context manager allows suppression of the script's stderr, which can be useful if you are using module functions that are known to produce warnings that you want to avoid distracting the user with, such as FutureWarnings of features that have become deprecated but are in a dependency rather than your own code. The class should be used sparingly and should **not** be used to hide evidence of bad code.

**class** sirilpy.utility.**SuppressedStdout**

Bases: object

This context manager allows suppression of the script's stdout, which can be useful to avoid flooding the log with stdout messages from an excessively verbose module used in the script.

### Example

```
import sirilpy as s
siril = s.SirilInterface()
print("This message will appear in the Siril log")
with s.SuppressedStdout():
    print("This message will not appear")
print("This message will appear again")
```

`sirilpy.utility.check_module_version(requires=None)`

Check the version of the Siril module is sufficient to support the script. This is not mandatory if you are only using classes, methods etc. that are provided in the initial public release, but if you rely on methods that are noted int he API documentation as having been added at a particular version of the module then you must check the running sirilpy module supports your script by calling this function.

Siril will not show scripts with unsatisfied check_module_version() calls in the repository list in 'Get Scripts', so if you want a function that enables different code paths for different versions you should use *needs_module_version()* instead.

**Parameters**

**requires** (*str*) -- A version format specifier string following the same format used by pip, i.e. it may contain '==1.2', '!=3.4', '>5.6', '>=7.8', or a combination such as '>=1.2,<3.4'

**Returns**

True if requires = None or if the available sirilpy module version satisfies the version specifier, otherwise False

**Raises**

**ValueError** -- if requires is an invalid version specifier.

`sirilpy.utility.download_with_progress(siril, url, file_path, max_retries=3, retry_delay=5, resume=True)`

Robust file download method with native Siril progress tracking and error handling using retries and a resume mechanism.

**Return type**

bool

`sirilpy.utility.ensure_installed(*packages, version_constraints=None, reinstall=False)`

Ensures that the specified package(s) are installed and meet optional version constraints.

**Parameters**

- ***packages** (*str or List[str]*) -- Name(s) of the package(s) to ensure are installed.
- **version_constraints** (*str or List[str], optional*) -- Version constraint string(s) (e.g. ">=1.5", "==2.0"). Can be a single constraint or a list matching packages.
- **reinstall** (*bool, optional*) -- Forces reinstallation. Defaults to False.

**Returns**

True if all packages are successfully installed or already meet constraints.

**Return type**

bool

**Raises**

- *SirilError* -- If package installation fails.
- **ValueError** -- If a different number of constraints is provided to the number of packages to be installed.
- **TimeoutError** -- If pip fails with an apparent timeout.
- **SystemExit** -- If a package that is already imported needs to be reinstalled with a different version.

`sirilpy.utility.human_readable_size(bytes_size)`

Convert bytes to human-readable format.

**Parameters**

**bytes_size** (*int*) -- Size in bytes

**Returns**
Formatted size with appropriate unit (B, KB, MB, GB, TB)

**Return type**
str

**Raises**
**TypeError** -- on incorrect input type

`sirilpy.utility.needs_module_version(requires=None)`

Check the version of the Siril module is sufficient to support a feature. This allows writing optional code paths for different sirilpy API levels. The function works the same as check_module_version() but the presence of an unsatisfied needs_module_version() call will not prevent a script from showing up in the list in 'Get Scripts'

**Parameters**
**requires** (*str*) -- A version format specifier string following the same format used by pip, i.e. it may contain '==1.2', '!=3.4', '>5.6', '>=7.8', or a combination such as '>=1.2,<3.4'

**Returns**
True if requires = None or if the available sirilpy module version satisfies the version specifier, otherwise False

**Raises**
**ValueError** -- if requires is an invalid version specifier.

`sirilpy.utility.parse_fits_header(header_text, include_comments=False)`

Parse FITS header from text content into a dictionary with support for HIERARCH and CONTINUE keywords.

Handles Siril's newline-separated FITS header format and converts it to a dictionary compatible with astropy.wcs.WCS. Supports extended FITS keywords via HIERARCH and long string values via CONTINUE.

**Parameters**

- **header_text** (`str`) -- Raw header string with header cards separated by newlines. Expected format: "KEYWORD = value / comment"
- **include_comments** (`bool`) -- If True, includes COMMENT and HISTORY cards in the output. If False (default), these cards are skipped.

**Return type**
`Dict[str, Union[str, int, float, bool]]`

**Returns**
Dictionary mapping FITS header keywords (str) to their parsed values. Values are converted to appropriate Python types: - 'T'/'F' -> bool - Quoted strings -> str (quotes removed, with CONTINUE support for long strings) - Numeric strings -> int or float - Everything else -> str - COMMENT/HISTORY cards -> str (content after keyword, if include_comments=True) - HIERARCH keywords -> str (full hierarchical keyword preserved)

### Notes

- Filters out warning messages, tracebacks, and other non-header content
- By default ignores COMMENT and HISTORY cards (set include_comments=True to include)
- Skips malformed cards or invalid keywords
- Standard keywords must be 8 characters and alphanumeric (plus underscore/hyphen)
- HIERARCH keywords can be longer and contain spaces/dots
- CONTINUE cards are automatically merged with the previous string value

- Comments after '/' are ignored for regular cards

### Example

```
>>> header_str = '''SIMPLE  = T / file conforms to FITS standard
... BITPIX  = -32 / bits per pixel
... HIERARCH ESO DET CHIP1 NAME = 'CCD #1' / detector name
... LONGSTR = 'This is a very long string that needs to be '
... CONTINUE  'continued on the next line'
... COMMENT Test comment'''
>>> result = parse_fits_header(header_str)
>>> result['SIMPLE']
True
>>> result['BITPIX']
-32
>>> result['HIERARCH ESO DET CHIP1 NAME']
'CCD #1'
>>> result['LONGSTR']
'This is a very long string that needs to be continued on the next line'
```

`sirilpy.utility.`**`safe_subprocess_Popen`**(*command*, ***kwargs*)

Provides a safe alternative to subprocess.Popen, accounting for the fact that a subprocess command line may be run from a flatpak environment and require use of flatpak-spawn to ensure it runs outside the sandbox, avoiding issues with subprocesses that rely on environment variables and so on. safe_subprocess_Popen is a drop-in replacement for subprocess.Popen for use in Siril scripts.

Introduced in sirilpy version 1.0.3

`sirilpy.utility.`**`safe_subprocess_run`**(*command*, ***kwargs*)

Provides a safe alternative to subprocess.Popen, accounting for the fact that a subprocess command line may be run from a flatpak environment and require use of flatpak-spawn to ensure it runs outside the sandbox, avoiding issues with subprocesses that rely on environment variables and so on. safe_subprocess_run is a drop-in replacement for subprocess.run for use in Siril scripts.

Introduced in sirilpy version 1.0.3

`sirilpy.utility.`**`truncate_utf8`**(*data*, *max_bytes*)

Truncates utf8 input. Accepts either bytes or str as input and returns data in the same format as the input.

**Parameters**
**data** (*bytes or str*) -- The data to be truncated

**Returns**
The truncated data

**Return type**
bytes or str

**Raises**
**TypeError** -- if the input was not bytes or str

`sirilpy.utility.`**`uninstall_package`**(*package_name*)

Uninstall a package using pip, streaming output to stdout. Introduced in sirilpy 1.0.15.

**Parameters**
**package_name** (*str*) -- Name of the package to uninstall.

**Raises**
**subprocess.CalledProcessError** -- If pip uninstallation fails.

## 25.8 Sirilpy GPU Helpers

This submodule provides helper classes to make it easier to manage GPU framework packages such as ONNX, Torch etc. for use in Siril python scripts. The landscape of these frameworks' support for different GPU architectures on different OSes is rapidly developing and the helpers aim to suggest reliable packages / configurations. This means that the proposed configurations may in some cases be conservative: the aim is to provide a good and robust level of GPU support for as many users as possible in an automated python environment rather than the absolute best, but perhaps fragile, optimisation. GPU helper module for Siril Python interface providing helper functions for detection, installation and testing of GPU-related modules. Initial scope is ONNX, torch and jax

**class** `sirilpy.gpuhelper.`**`JaxHelper`**

Bases: `object`

A helper class for detecting, installing, and testing JAX with appropriate hardware acceleration.

This class automatically detects the system configuration and installs the correct JAX variant (CPU, CUDA, ROCm, etc.) based on available hardware.

**`ensure_jax()`**

Wrapper for install_jax() that only installs it if needed, with negligible overhead if it is already installed.

**Return type**
`bool`

**`get_recommended_backend()`**

Determine the recommended JAX backend and installation parameters. When multiple GPUs are present, uses priority system to choose the best one.

JAX has limited platform support: - NVIDIA: CUDA on Linux (Windows support experimental/limited) - AMD ROCm: Linux only - Apple Silicon: Metal backend - Intel: Experimental plugin with dependency issues

**Return type**
`Dict[str, Any]`

**Returns**
Dict with 'backend', 'packages', 'extra_index_url' keys

**`install_jax`**(*force_variant=None*, *version_constraint=None*, *force_reinstall=False*)

Install JAX with the appropriate variant for the detected hardware. Use this instead of ensure_installed() to ensure that jax is installed correctly for the given hardware / OS

**Parameters**

- **`force_variant`** (`Optional[str]`) -- Override auto-detection with specific variant (e.g., 'jax[cpu]')
- **`version_constraint`** (`Optional[str]`) -- Version constraint string (e.g., '>=0.4.0')

force_reinstall: Forces a reinstallation

**Returns**
True if installation succeeded, False otherwise

**Return type**
bool

**`is_jax_installed()`**

Check if PyTorch is installed without importing it.

**Return type**
`bool`

**status()**

Prints the current status of the Jax Helper class in regard to its support for different OSes, GPUs. The world of heterogenous computing is developing rapidly and while support for some of the frameworks for which helpers are available is not yet universally available, hopefully it will improve in the future.

**test_jax()**

Test JAX functionality and return execution provider.

**Returns**
the bool returned is True if jax works or False if it does not, and the str is "gpu" if JAX is using GPU, "cpu" if using CPU or None if

**Return type**
Tuple[bool,str]

**Raises**

- **RuntimeError** -- If JAX computation fails or accuracy check fails
- **ImportError** -- If JAX is not installed

**uninstall_jax**(*dry_run=False*)

Detect and uninstall any existing JAX-related packages.

This is useful when you need to clean up a problematic JAX installation before installing a different variant (e.g., falling back from GPU to CPU).

**Parameters**
**dry_run** (bool) -- If True, only detect packages without uninstalling them

**Return type**
Dict[str, Any]

**Returns**
Dict containing information about detected and uninstalled packages

**class** sirilpy.gpuhelper.**ONNXHelper**

Bases: object

A class to handle detection and installation of the appropriate ONNX Runtime package based on the system hardware and configuration.

Example usage (this should be used instead of `sirilpy.ensure_installed("onnxruntime")` to install the correct package for the user's system.)

```
oh = sirilpy.ONNXHelper()
oh.ensure_onnxruntime()
```

**ensure_onnxruntime()**

Wrapper for install_onnxruntime() that only installs it if needed, with negligible overhead if it is already installed.

**Return type**
bool

**get_execution_providers_ordered**(*ai_gpu_acceleration=True*, *force_check=False*)

Get execution providers ordered by priority. This function returns a list of available ONNX Runtime execution providers in a reasonable order of priority, covering major GPU platforms: The CPU provider is always included as the final fallback option.

**Parameters**

- **ai_gpu_acceleration** (*bool*) -- Whether to include GPU acceleration providers. Defaults to True.
- **force_check** (*bool*) -- Whether to force re-checking even if a cached config exists. Defaults to False.

**Returns**
Ordered list of available execution providers.

**Return type**
list

**get_recommended_backend**()

Determine the recommended ONNX Runtime backend based on hardware. When multiple GPUs are present, uses priority system to choose the best one.

Note: ONNX Runtime has platform-specific backend support: - Windows: DirectML for all GPUs (NVIDIA, AMD, Intel) - more reliable, no driver dependencies - Linux: CUDA for NVIDIA, CPU for AMD (no ROCm support yet), OpenVINO for Intel

**Returns**
'gpu', 'directml', 'openvino', 'coreml', or 'cpu'

**Return type**
Backend name

**import_onnxruntime**()

Import onnxruntime, add it to the global dict, test if it's built against CUDA and if so preload the CUDA and CUDNN libraries to improve the chances of finding them if Torch[CUDA] happens to be installed.

**install_onnxruntime**(*force=False*)

Detect system configuration and install the appropriate ONNX Runtime package.

**Parameters**
**force** -- bool: If True, force reinstallation even if onnxruntime is already installed.

**Returns**
True if installation was successful or already installed, False otherwise.

**Return type**
bool

**Raises**
**TimooutError** -- if a TimeoutError occurs in ensure_installed() - this avoids falling back to the CPU-only package purely because of network issues

**is_onnxruntime_installed**()

Check if any onnxruntime package is already installed and usable.

**Returns**

**(is_installed, package_name) where package_name could be**
'onnxruntime', 'onnxruntime-gpu', 'onnxruntime-silicon', etc.

**Return type**
tuple

**run**(*session*, *model_path*, *output_names*, *input_feed*, *run_options=None*, *return_first_output=False*)

Run inference with automatic CPU fallback if the session fails.

**Parameters**

- **session** -- The ONNX runtime inference session

- **model_path** (*str*) -- Path to the ONNX model file (needed for CPU fallback)
- **output_names** -- Names of the outputs to compute, or None for all outputs
- **input_feed** -- Dictionary mapping input names to input tensors
- **run_options** -- Optional run options for the inference session
- **return_first_output** (*bool*) -- If True, return only the first output instead of the full list

**Returns**

**(result, session) where result is the inference output (or first output if return_first_output=True) and**
session is the (potentially updated) inference session

**Return type**
tuple

`status()`

Prints the current status of the ONNX Helper class in regard to its support for different OSes, GPUs. The world of heterogenous computing is developing rapidly and while support for some of the frameworks for which helpers are available is not yet universally available, hopefully it will improve in the future.

`test_onnxruntime(ort=None)`

Test an imported onnxruntime.

**Args:install_torch(cuda_version=cuda_version)**

**ort: The ONNX runtime module to test. If None, the method will**
attempt to import onnxruntime for the test.

**Returns**
a list of confirmed working ONNXRuntime ExecutionProviders in priority order

**Return type**
list

`uninstall_onnxruntime()`

Detects and uninstalls all variants of onnxruntime packages. Checks for any package starting with 'onnxruntime'.

**Returns**
A list of uninstalled packages

**Return type**
list

`class sirilpy.gpuhelper.TorchHelper`

Bases: `object`

Helper class for PyTorch detection, installation and testing.

`ensure_torch(cuda_version=None)`

Ensure PyTorch is installed with the appropriate backend.

**Parameters**
**cuda_version** (`Optional[str]`) -- Optional CUDA version to override auto-detection (e.g., 'cu118', 'cu126', 'cu128')

**Return type**
`bool`

Returns: True on success, False on failure

**get_recommended_backend()**

Determine the recommended PyTorch backend and installation parameters. When multiple GPUs are present, uses priority system to choose the best one.

PyTorch has good cross-platform support: - NVIDIA: CUDA on all platforms - AMD ROCm-compatible: ROCm on Linux AND Windows - Intel Arc: XPU backend - Apple Silicon: MPS backend

**Return type**
`Dict[str, Any]`

**Returns**
Dict with 'backend', 'cuda_version', 'extra_index_url', 'packages' keys

**get_torch_device**(*use_gpu=True*)

Obtains a suitable torch device based on the capabilities of the installed torch package. if use_gpu is False, torch.device("cpu") will be returned. This function is available since sirilpy 1.0.17

**Return type**
torch.device

**install_torch**(*cuda_version=None*, *extra_index_url=None*, *packages=None*)

Install PyTorch with specified configuration.

**Parameters**

- **cuda_version** (`Optional[str]`) -- CUDA version (e.g., 'cu118', 'cu126', 'cu128')
- **extra_index_url** (`Optional[str]`) -- PyTorch wheel repository URL
- **packages** (`Optional[list]`) -- List of packages to install

**is_torch_installed()**

Check if PyTorch is installed without importing it.

**Return type**
`bool`

**status()**

Prints the current status of the Torch Helper class in regard to its support for different OSes, GPUs. The world of heterogenous computing is developing rapidly and while support for some of the frameworks for which helpers are available is not yet universally available, hopefully it will improve in the future.

**test_torch()**

Run tests to verify that torch is installed and runs correctly.

**uninstall_torch()**

Detects and uninstalls PyTorch and related packages. Checks for torch ecosystem packages. :returns: A list of uninstalled packages :rtype: list

sirilpy.gpuhelper.**detect_gpu_capabilities()**

Comprehensive GPU detection for all supported hardware.

**Return type**
`Dict[str, Any]`

**Returns**
Dict containing detected hardware information and recommendations for PyTorch, ONNX, and JAX backends.

`sirilpy.gpuhelper.get_gpu_detection_report()`

Generate a comprehensive GPU detection report suitable for display in a GUI.

**Return type**
`Dict[str, Any]`

**Returns**
Dict containing all detected hardware and recommended configurations for ONNX, PyTorch, and JAX, including GPU priority information.

## 25.9 Sirilpy TK GUI

> **Warning**
>
> **This submodule is now deprecated.** There are no longer any scripts in the repository that use tksiril and submissions to the siril-scripts repository will no longer be accepted if they use tksiril. You should migrate any personal scripts to use PyQt6 instead. **The tksiril submodule will be removed in a future version of Siril.**

This submodule provides some helper functions to support consistent GUI implementation using `tkinter`. It must be explicitly imported in order to be used. Note that when writing TKinter GUIs you should import ThemedTK from the ttkthemes module, because the bare TKinter UI shows up poorly on MacOS. ThemedTK and the methods available in the `tksiril` module help to provide a consistent look for Siril script GUIs on all platforms.

> **Warning**
>
> Linux users running a Wayland desktop should note that Tkinter does not yet have support for pure Wayland. In order to use python scripts utilising Tkinter GUIs you **must** have the XWayland compatibility package installed. If you don't, you will see errors about DISPLAY being unavailable.

```
from sirilpy import tksiril
```

TKsiril submodule for Siril, providing utility methods to achieve consistent script GUI appearance using the TKinter toolkit.

DEPRECATION WARNING: the tksiril submodule is deprecated from Siril 1.4.3. It will remain part of sirilpy throughout the stable 1.4 series but has been removed from master and will therefore disappear with the release of Siril 1.6.0.

No new scripts will be accepted in the siril-scripts repository if they use the tksiril submodule.

**class** `sirilpy.tksiril.ScrollableFrame(container, *args, **kwargs)`

Bases: `Frame`

A scrollable frame widget that can contain other widgets.

This class creates a frame with vertical scrolling capability using a Canvas widget and Scrollbar. It supports both scrollbar and mouse wheel scrolling across all platforms (Windows, Mac, Linux).

**Usage:**

scrollable = ScrollableFrame(parent) scrollable.pack(fill="both", expand=True)

# Add widgets to scrollable.scrollable_frame label = ttk.Label(scrollable.scrollable_frame, text="Hello") label.pack()

# Optionally bind mouse wheel to child widgets scrollable.add_mousewheel_binding(label)

**add_mousewheel_binding**(*widget=None*)

Add mouse wheel scrolling support to a widget and its children.

This method recursively binds mouse wheel events to the specified widget and all its child widgets. It uses platform detection to apply the appropriate event bindings for each operating system.

**Parameters**

**widget** -- The tkinter widget to bind mouse wheel events to. The binding will be applied recursively to all children. If no widget is specified it will default to the ScrollableFrame itself.

### Example

# Add a complex widget to the scrollable frame frame = ttk.Frame(scrollable.scrollable_frame) label = ttk.Label(frame, text="Hello") button = ttk.Button(frame, text="Click me")

# Bind mouse wheel to the entire widget hierarchy scrollable.add_mousewheel_binding(frame)

sirilpy.tksiril.**create_tooltip**(*widget*, *text*, *wrap_length=250*)

Create a tooltip for a given Tkinter widget.

**Parameters**

- **widget** (*tk.Widget*) -- The widget to attach the tooltip to
- **text** (*str*) -- The tooltip text to display
- **max_width** (*int, optional*) -- Maximum width of the tooltip. Defaults to 300.
- **wrap_length** (*int, optional*) -- Length at which text wraps. Defaults to 250.

**Raises**

**TypeError** -- If text is not a string or the provided widget is not a valid Tkinter widget

sirilpy.tksiril.**elevate**(*root*)

Raises the Tk root window to the top of the window stack. Useful after calls to sirilpy methods that present child windows of the main Siril window such as info_messagebox().

NOTE: For this to work on KDE desktops, focus-stealing prevention must be disabled.

sirilpy.tksiril.**match_theme_to_siril**(*themed_tk*, *s*, *on_top=False*)

Match the Tkinter theme to the Siril configuration and set the script dialog to have topmost status, meaning that it will remain in front of other non-topmost windows.

**Parameters**

- **s** (SirilInterface) -- sirilpy.SirilInterface class to provide the Siril theme (light or dark) to match
- **themed_tk** (*ThemedTk*) -- ThemedTk instance to apply the theme to
- **on_top** -- whether the script window should be always on top of other windows

**Raises**

- **TypeError** -- If input arguments are of incorrect type
- **ValueError** -- If the theme configuration is not 0 or 1
- **AttributeError** -- If required methods are not available
- **RuntimeError** -- If there are errors installing or setting the theme

`sirilpy.tksiril.standard_style()`

Provide a standardised ttk style to allow consistent visual appearance between different Siril python scripts.

**Parameters**
none

**Raises**
*SirilError* -- If the style creation or configuration fails

## 25.10 Sirilpy Tkfilebrowser

**Warning**

**This submodule is now deprecated.** There are no longer any scripts in the repository that use tkfilebrowser and submissions to the siril-scripts repository will no longer be accepted if they use tkfilebrowser. You should migrate any personal scripts to use PyQt6 instead. **The tkfilebrowser submodule will be removed in a future version of Siril.**

This submodule is a fork of tkfilebrowser. The fork addresses a bug in the original code where duplicate device entries could cause errors in generating the filebrowser, and ensures the code can be mantained as the upstream package was last updated several years ago.

Documentation for tkfilebrowser can be found here. Note that some compatibility improvements have been made in the version included in sirilpy:

- `sirilpy.tkfilebrowser.askdirectory()` has been added as an alias for `tkfilebrowser.askopendirname()` to maintain compatibility with `tk.filedialog.askdirectory()`.
- In filefilter specifications `tkfilebrowser` requires multiple extensions to be provided separated by | whereas filedialog requires space-separated extensions: `sirilpy.tkfilebrowser` has been adapted to accept either format.

This submodule serves the sole purpose of being a drop-in replacement for the standard Tk filedialog on Linux, as the standard Tk filedialog is horrible on Linux. It can be used as a replacement like this:

```
if sys.platform.startswith("linux"):
    import sirilpy.tkfilebrowser as filedialog
else:
    from tkinter import filedialog
```

## 25.11 Sirilpy Exceptions

This submodule provides some customized exceptions to support Siril-specific error handling. All its members are made available at the module root level, there is no need to import it separately.

The sirilpy exceptions policy is as follows:

- At a low level within sirilpy methods a variety of exception types may be raised (those shown below as well as exceptions raised from other modules such as struct.error). Internal exception types are all descended from SirilError and may therefore be caught using *except SirilError*. Other exception types are re-raised as a SirilError to show the method where they were generated, but the underlying error can still be seen either in a traceback or using the Exception `__cause__` property.

- Some error types are recoverable errors such as NoImageError, NoSequenceError and CommandError. These exception types can be handled at script level (for example by showing a warning dialog reminding the user to load an image).
- Other error types are typically not recoverable, such as SharedMemoryError or SirilConnectionError.

Exceptions submodule for Siril, providing exception classes for use in exception raising within the sirilpy module.

**exception** sirilpy.exceptions.**CommandError**(*message='Command execution failed'*, *status_code=CommandStatus.CMD_GENERIC_ERROR*)

Bases: *SirilError*

Raised when a command sent to Siril fails to execute properly. (Note: 'command' in this case refers to internal commands sent from the python module to the Siril python handler, not Siril commands of the type that might be entered in the Siril command entry.) The full set of command status codes is shown in the CommandStatus enum. These exceptions are often recoverable and should therefore be handled before generically handling other SirilError types that are considered fatal.

**status_code**

(CommandStatus) Indicates the status code returned by the Siril command. This may be used in error handlers to allow scripts to handle some types of command error and continue (e.g. by prompting a user intervention).

**exception** sirilpy.exceptions.**DataError**(*message='Error handling data'*)

Bases: *SirilError*

Raised when there are problems with data handling. This includes cases like:

- Invalid image data
- Data conversion errors
- Memory allocation failures
- Buffer overflows

**exception** sirilpy.exceptions.**ImageDialogOpenError**(*message='Siril image dialog is open'*)

Bases: *SirilError*

Exception raised when an image processing dialog is open.

**exception** sirilpy.exceptions.**MouseModeError**(*message='Siril mouse mode error'*)

Bases: *SirilError*

Exception raised when Siril is in the wrong mouse mode.

**exception** sirilpy.exceptions.**NoImageError**(*message='No Siril image loaded'*)

Bases: *SirilError*

Raised when a method requires an image to be loaded but no image is loaded. These exceptions are often recoverable and should therefore be handled before generically handling other SirilError types that are considered fatal.

**exception** sirilpy.exceptions.**NoSequenceError**(*message='No Siril sequence loaded'*)

Bases: *SirilError*

Raised when a method requires a sequence to be loaded but no sequence is loaded. These exceptions are often recoverable and should therefore be handled before generically handling other SirilError types that are considered fatal.

**exception** sirilpy.exceptions.**ProcessingThreadBusyError**(*message='Siril processing thread is busy'*)

Bases: *SirilError*

Exception raised when the processing thread is already in use.

**exception** sirilpy.exceptions.**SharedMemoryError**(*message='Siril shared memory error'*)

Bases: *SirilError*

Raised when there are problems connecting to or communicating with Siril using shared memory.

SharedMemoryError is not raised directly but will be wrapped in a SirilError. It should generally be regarded as fatal and the script should shut down gracefully if possible or just stop.

**exception** sirilpy.exceptions.**SirilConnectionError**(*message='Failed to connect to Siril'*)

Bases: *SirilError*

Raised when there are problems connecting to or communicating with Siril. This includes cases like:

- Siril not running
- Socket connection failures
- Communication protocol errors
- Unexpected disconnections

SirilConnectionError is not raised directly but will be wrapped in a SirilError. It should generally be regarded as fatal and the script should shut down gracefully if possible or just stop.

**exception** sirilpy.exceptions.**SirilError**(*message='An error occurred'*)

Bases: `Exception`

Base exception class for all Siril-related errors.

All other Siril exceptions inherit from this class, making it easy to catch any Siril-related error with a single except clause.

CHAPTER

TWENTYSIX

# HOW TO REPORT ISSUES

If, despite studying the documentation and tutorials, you're experiencing strange behavior, this page will guide you on how to proceed. Keep in mind that if something doesn't work, it doesn't necessarily mean there's a bug. More often than not, issues arise due to incorrect usage. In such cases, it's crucial to provide precise information about the problem, which will help us determine if it's a bug or a misunderstanding. Refer to the section on how to *send us useful information* for guidance. Indeed, just saying it doesn't work won't help us find a solution. We're not soothsayers and need information to identify the problem and devise a solution. So it's important that you tell us what you were doing at the time the problem occurred, what operating system you're using, what version of Siril you're using and most importantly logs!

## 26.1 Check changelogs and bug trackers

First of all, if you think you found a bug in Siril, it may already have been reported (sometimes literally dozens of times). We therefore ask you to check first, for example by looking at changelogs, or tickets that have already been **opened**, and even **closed**.

## 26.2 Send us useful information

As mentioned in the introduction, we need useful information to help solve the problem:

1. What operating system you're using? Because Siril can behave very differently on Windows, Linux or macOS, we need to know this information. Please be as precise as possible.
2. Which version of Siril do you use? And how did you get it? Package downlaoded on the website? Through a third party? Compiled by yourself? Here again, please, be as precise as possible.
3. Sometimes it's useful to share screenshots. However, please **DO NOT TAKE YOUR SCREENSHOTS WITH YOUR SMARTPHONE** - it's illegible. Your operating system is capable of making screenshots very easily (Google can help you with this) and Siril also offers such a feature (the button with the camera). Finally, the desired formats are image formats: `jpg`, `bmp` or `png`, but absolutely not `pdf`.
4. Send us logs. Ideally, we prefer English logs! Just go to Siril preferences and change the language to English in the *User Interface* tab. Also, There are two types of logs: the one displayed in the Siril console, which describes the steps performed by the software and can help us debug, and the internal logs that are visible when Siril is run from the command line:
   - The former are very useful in most cases, and can be exported very easily using the button at the bottom right of them. This creates a file that can easily be sent to us.
   - However, when the software crashes (i.e. suddenly closes without warning), you need to start Siril from the command line, trying to reproduce the crash and retrieve the logs. Here, the method depends on the operating system.
     - Microsoft Windows: Open a cmd window (type `cmd` in Windows Search bar) and type the following:

```
"C:\Program Files\Siril\bin\siril.exe" 2>&1 >output.log
```

This will save the file output.log to the folder where the terminal was started (in most cases in `%USERPROFILE%` folder).

- macOS: If you've installed Siril in the Applications folder, as is generally recommended, then start by opening the Terminal application from the Utilities folder within Applications, then copy and paste the following line:

```
/Applications/Siril.app/Contents/MacOS/siril > ~/Desktop/output.log 2>&1
```

After the crash, the logs will be available on the desktop in the file `output.log`.

- GNU/Linux: Simply start Siril in a terminal. Usually, the binary is located in the `$PATH` variable, in which case type:

```
siril > output.log 2>&1
```

is all you need to get the logs redirected in the file `output.log`. This will save the file output.log to the folder where the terminal was started

5. Send us your pic. If you find your image strange, don't hesitate to share it with us, usually in FITS format. It's always more interesting than a screenshot. To do this, you need to use a large file exchange service. There are lots of them, and we can suggest WeTransfer, for example. In that case, upload your data to WeTransfer and get a download link to share.

## 26.3 How to contact us?

There are several ways to contact us and report a bug. The easiest is to find us on the forum. But it is also possible to open a ticket on our gitlab repository. In this case, please check first that the same ticket has not already been opened. It may even have been closed because it has been resolved, in which case, a short description with a ticket number will be shown in the Changelog. To visualize the ticket (even a closed one) and confirm you are, or not, experiencing the same issue, go to the address https://gitlab.com/free-astro/siril/-/issues/XXXX with XXXX the ticket number.

**Tip**

As Siril users and developers are of different nationalities, the language used to report a bug must be English.

# BIBLIOGRAPHY


[Fischler1981] Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381-395.

[Valdes1995] Valdes, F. G., Campusano, L. E., Velasquez, J. D., & Stetson, P. B. (1995). FOCAS automatic catalog matching algorithms. Publications of the Astronomical Society of the Pacific, 107(717), 1119.

[Losson2010] Olivier Losson, Ludovic Macaire, Yanqin Yang. Comparison of color demosaicing methods. Advances in Imaging and Electron Physics, 2010, 162, pp.173-265, section 2.2.2. https://hal.science/hal-00683233/document

[FruchterHook1997] A. S. Fruchter and R. N. Hook. (1997) *A novel image reconstruction method applied to deep Hubble Space Telescope images.* Proc. S.P.I.E. vol. 3164. https://arxiv.org/abs/astro-ph/9708242

[Avila2015] R. J. Avila, A. Koekemoer, J. Mack, A. Fruchter. (2015) *Optimizing pixfrac in Astrodrizzle: An Example from the Hubble Frontier Fields.* https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2015/WFC3-2015-04.pdf

[Peter2009] Peter J. Huber and E. Ronchetti (2009), Robust Statistics, 2nd Ed., Wiley

[ConejeroPI] Juan Conejero, ImageIntegration, Pixinsight Tutorial

[Rosner1983] Rosner, B. (1983). *Percentage points for a generalized ESD many-outlier procedure*. Technometrics, 25(2), 165-172.

[Brown2007] Brown, M., & Lowe, D. G. (2007). *Automatic panoramic image stitching using invariant features*. International journal of computer vision, 74, 59-73.

[GaiaDR3] Vallenari, A., et al. "Gaia Data Release 3-Summary of the content and survey properties." Astronomy & Astrophysics 674 (2023): A1. 99(613), 191.

[SWIRE] https://www.iasf-milano.inaf.it/~polletta/templates/swire_templates.html

[Stellar] https://www.eso.org/sci/facilities/paranal/decommissioned/isaac/tools/lib.html

[Wiki_HEALPIX] https://en.wikipedia.org/wiki/HEALPix

[Young1994] https://opg.optica.org/ao/viewmedia.cfm?uri=ao-33-6-1108&seq=0

[Anger2018] Anger, J., Facciolo, G., & Delbracio, M. (2018). *Estimating an image's blur kernel using natural image statistics, and deblurring it: an analysis of the Goldstein-Fattal method.* Image Processing On Line, 8, 282-304. https://doi.org/10.5201/ipol.2018.211

[Anger2019] Anger, J., Facciolo, G., & Delbracio, M. (2019). Blind image deblurring using the l0 gradient prior. Image processing on line, 9, 124-142. https://doi.org/10.5201/ipol.2019.243

[Goldstein2012] Goldstein, A., & Fattal, R. (2012, October). *Blur-kernel estimation from spectral irregularities.* In European Conference on Computer Vision (pp. 622-635). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33715-4_45

[Lucy1974] Lucy, L. B. (1974). *An iterative technique for the rectification of observed distributions.* The astronomical journal, 79, 745. https://doi.org/10.1086/111605.

[Lebrun2013] Lebrun, M., Buades, A., & Morel, J. M. (2013) *Implementation of the "Non-Local Bayes" (NL-Bayes) Image Denoising Algorithm.* Image Processing On Line, 3 , pp. 1–42. https://doi.org/10.5201/ipol.2013.16

[Pierazzo2017] Pierazzo, N., & Facciolo, G. (2017). *Data adaptive dual domain denoising: a method to boost state of the art denoising algorithms.* Image Processing On Line, 7, 93-114. https://doi.org/10.5201/ipol.2017.203

[Mäkitalo2011] Mäkitalo, M., & Foi, A. (2012, March). *Poisson-gaussian denoising using the exact unbiased inverse of the generalized anscombe transformation.* In 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 1081-1084). IEEE. https://doi.org/10.1109/ICASSP.2012.6288074

[Mäkitalo2012] Makitalo, M., & Foi, A. (2011). *A closed-form approximation of the exact unbiased inverse of the Anscombe variance-stabilizing transformation.* IEEE transactions on image processing, 20(9), 2697-2698. https://doi.org/10.1109/TIP.2011.2121085

[Romano2015] Romano, Y., & Elad, M. (2015). *Boosting of image denoising algorithms.* SIAM Journal on Imaging Sciences, 8(2), 1187-1219. https://doi.org/10.1137/140990978

[Wright2003] Wright, Grady Barrett. Radial basis function interpolation: numerical and analytical developments. University of Colorado at Boulder, 2003.

[Stetson1987] Stetson, P. B. (1987). DAOPHOT: A computer program for crowded-field stellar photometry. Publications of the Astronomical Society of the Pacific, 99(613), 191.

# PYTHON MODULE INDEX

## S

sirilpy.connection, 544
sirilpy.enums, 587
sirilpy.exceptions, 606
sirilpy.gpuhelper, 598
sirilpy.models, 570
sirilpy.plot, 593
sirilpy.tksiril, 603
sirilpy.utility, 594

# COMMANDS INDEX


## A

addmax, 471
asinh, 471
autoghs, 472
autostretch, 472

## B

bg, 472
bgnoise, 473
binxy, 473
boxselect, 473

## C

calibrate, 473
calibrate_single, 474
capabilities, 475
catsearch, 475
ccm, 475
cd, 476
cdg, 476
clahe, 476
clear, 477
clearstar, 477
close, 477
conesearch, 477
convert, 478
convertraw, 478
cosme, 478
cosme_cfa, 479
crop, 479

## D

ddp, 480
denoise, 480
dir, 481
disto, 481
dumpheader, 481

## E

entropy, 481
epf, 482
exit, 482
extract, 482
extract_Green, 483
extract_Ha, 483
extract_HaOIII, 483

## F

fdiv, 484
ffill, 484
fftd, 484
ffti, 484
fill, 485
find_cosme, 485
find_cosme_cfa, 485
find_hot, 485
findcompstars, 486
findstar, 486
fix_xtrans, 487
fixbanding, 487
fmedian, 488
fmul, 488

## G

gauss, 488
get, 488
getref, 489
ght, 489
grey_flat, 489

## H

help, 490
histo, 490

## I

iadd, 490
icc_assign, 490
icc_convert_to, 491
icc_remove, 491
idiv, 491
imul, 492
inspector, 492
invght, 492

invmodasinh, 492
invmtf, 493
isub, 493

## J

jsonmetadata, 493

## L

light_curve, 494
limit, 494
linear_match, 495
link, 495
linstretch, 495
livestack, 495
load, 496
load_seq, 496
log, 496
ls, 497

## M

makepsf, 497
merge, 498
merge_cfa, 498
mirrorx, 499
mirrorx_single, 499
mirrory, 499
modasinh, 499
mtf, 500

## N

neg, 500
new, 500
nozero, 501

## O

offline, 501
offset, 501
online, 501

## P

parse, 502
pcc, 502
platesolve, 502
pm, 503
profile, 504
psf, 504
pwd, 505
pyscript, 505

## R

register, 505
reloadscripts, 507
requires, 507
resample, 507
rgbcomp, 507
rgradient, 508
rl, 508
rmgreen, 509
rotate, 509
rotatePi, 509

## S

satu, 510
save, 510
savebmp, 510
savejpg, 510
savejxl, 511
savepng, 511
savepnm, 511
savetif, 512
savetif32, 512
savetif8, 512
sb, 512
select, 513
seqapplyreg, 513
seqccm, 515
seqclean, 515
seqcosme, 515
seqcosme_cfa, 516
seqcrop, 516
seqextract_Green, 516
seqextract_Ha, 517
seqextract_HaOIII, 517
seqfind_cosme, 517
seqfind_cosme_cfa, 517
seqfindstar, 518
seqfixbanding, 518
seqght, 518
seqheader, 519
seqinvght, 519
seqinvmodasinh, 519
seqlinstretch, 519
seqmerge_cfa, 520
seqmodasinh, 520
seqmtf, 520
seqprofile, 521
seqpsf, 521
seqplatesolve, 522
seqresample, 523
seqrl, 523
seqsb, 523
seqsetmag, 524
seqsplit_cfa, 524
seqstarnet, 524
seqstat, 525
seqsubsky, 525
seqtilt, 525

sequnsetmag, 526
sequpdate_key, 526
seqwiener, 526
set, 527
set16bits, 527
set32bits, 527
setcompress, 527
setcpu, 528
setext, 528
setfindstar, 528
setmag, 529
setmem, 529
setphot, 530
setref, 530
show, 531
spcc, 531
spcc_list, 532
split, 532
split_cfa, 532
stack, 533
stackall, 534
starnet, 535
start_ls, 535
stat, 535
stop_ls, 536
subsky, 536
synthstar, 536

## T

threshlo, 537
threshhi, 537
thresh, 537
tilt, 537
trixel, 538

## U

unclipstars, 538
unpurple, 538
unselect, 539
unsetmag, 539
unsharp, 539
update_key, 539

## V

visu, 540

## W

wavelet, 540
wiener, 540
wrecons, 541

# INDEX

## A


A (*sirilpy.models.PSFStar attribute*), 581
A_err (*sirilpy.models.PSFStar attribute*), 581
ABERRATION_DIALOG (*sirilpy.enums.DialogID attribute*), 588
ABOUT_DIALOG (*sirilpy.enums.DialogID attribute*), 588
add_mousewheel_binding() (*sirilpy.tksiril.ScrollableFrame method*), 603
add_series() (*sirilpy.plot.PlotData method*), 593
add_series_obj() (*sirilpy.plot.PlotData method*), 593
addmax, **471**
airmass (*sirilpy.models.FKeywords attribute*), 574
airmass (*sirilpy.models.ImgData attribute*), 580
allocate_data() (*sirilpy.models.FFit method*), 571
analyse_image_from_file() (*sirilpy.connection.SirilInterface method*), 544
ang_err (*sirilpy.models.PSFStar attribute*), 581
angle (*sirilpy.models.PSFStar attribute*), 581
ANNOTATE_DIALOG (*sirilpy.enums.DialogID attribute*), 588
ANY (*sirilpy.enums.DialogReq attribute*), 590
aperture (*sirilpy.models.FKeywords attribute*), 574
asinh, **471**
ASINH_DIALOG (*sirilpy.enums.DialogID attribute*), 588
ASINH_DISPLAY (*sirilpy.enums.STFType attribute*), 592
ASTROMETRY_DIALOG (*sirilpy.enums.DialogID attribute*), 588
autoghs, **472**
autostretch, **472**
AUTOSTRETCH_DISPLAY (*sirilpy.enums.STFType attribute*), 592
avgDev (*sirilpy.models.ImageStats attribute*), 579


## B


B (*sirilpy.models.PSFStar attribute*), 581
B_err (*sirilpy.models.PSFStar attribute*), 581
BACKGROUND_EXTRACTION_DIALOG (*sirilpy.enums.DialogID attribute*), 588
background_lvl (*sirilpy.models.RegData attribute*), 585
bayer_pattern (*sirilpy.models.FKeywords attribute*), 574
bayer_xoffset (*sirilpy.models.FKeywords attribute*), 574
bayer_yoffset (*sirilpy.models.FKeywords attribute*), 574
beg (*sirilpy.models.Sequence attribute*), 586
beta (*sirilpy.models.PSFStar attribute*), 581
beta_err (*sirilpy.models.PSFStar attribute*), 581
bg, **472**
bgnoise, **473**
bgnoise (*sirilpy.models.ImageAnalysis attribute*), 578
bgnoise (*sirilpy.models.ImageStats attribute*), 579
BGSample (*class in sirilpy.models*), 570
BIAS (*sirilpy.enums.ImageType attribute*), 591
binning_x (*sirilpy.models.FKeywords attribute*), 574
binning_y (*sirilpy.models.FKeywords attribute*), 574
binxy, **473**
BINXY_DIALOG (*sirilpy.enums.DialogID attribute*), 588
bitpix (*sirilpy.models.FFit attribute*), 572
bitpix (*sirilpy.models.Sequence attribute*), 586
BitpixType (*class in sirilpy.enums*), 587
BLUE (*sirilpy.enums.LogColor attribute*), 591
BLUE (*sirilpy.enums.SirilVport attribute*), 592
Bmag (*sirilpy.models.PSFStar attribute*), 581
boxselect, **473**
bscale (*sirilpy.models.FKeywords attribute*), 574
BV (*sirilpy.models.PSFStar attribute*), 581
BYTE_IMG (*sirilpy.enums.BitpixType attribute*), 587
bzero (*sirilpy.models.FKeywords attribute*), 574


## C


calibrate, **473**
calibrate_single, **474**
CANON_FIXBANDING_DIALOG (*sirilpy.enums.DialogID attribute*), 588
capabilities, **475**
catsearch, **475**
ccd_temp (*sirilpy.models.FKeywords attribute*), 574
ccm, **475**
CCM_DIALOG (*sirilpy.enums.DialogID attribute*), 588
cd, **476**
cdg, **476**
centalt (*sirilpy.models.FKeywords attribute*), 574

centaz (*sirilpy.models.FKeywords attribute*), 574
cfa_opened_monochrome (*sirilpy.models.Sequence attribute*), 586
channels (*sirilpy.models.FFit property*), 572
channels (*sirilpy.models.ImageAnalysis attribute*), 578
check_module_version() (*in module sirilpy.utility*), 594
checksum (*sirilpy.models.FFit attribute*), 572
clahe, **476**
CLAHE_DIALOG (*sirilpy.enums.DialogID attribute*), 588
clear, **477**
clear_image_bgsamples() (*sirilpy.connection.SirilInterface method*), 545
clear_undo_history() (*sirilpy.connection.SirilInterface method*), 545
clearstar, **477**
close, **477**
cmd() (*sirilpy.connection.SirilInterface method*), 545
CMD_ALLOC_ERROR (*sirilpy.enums.CommandStatus attribute*), 587
CMD_ARG_ERROR (*sirilpy.enums.CommandStatus attribute*), 587
CMD_DIR_NOT_FOUND (*sirilpy.enums.CommandStatus attribute*), 587
CMD_FILE_NOT_FOUND (*sirilpy.enums.CommandStatus attribute*), 587
CMD_FOR_CFA_IMAGE (*sirilpy.enums.CommandStatus attribute*), 587
CMD_FOR_PLATE_SOLVED (*sirilpy.enums.CommandStatus attribute*), 587
CMD_GENERIC_ERROR (*sirilpy.enums.CommandStatus attribute*), 587
CMD_IMAGE_NOT_FOUND (*sirilpy.enums.CommandStatus attribute*), 587
CMD_INVALID_IMAGE (*sirilpy.enums.CommandStatus attribute*), 587
CMD_LOAD_IMAGE_FIRST (*sirilpy.enums.CommandStatus attribute*), 587
CMD_NEED_INIT_FIRST (*sirilpy.enums.CommandStatus attribute*), 587
CMD_NO_CWD (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NO_WAIT (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NOT_FOR_MONO (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NOT_FOR_RGB (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NOT_FOR_SINGLE (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NOT_FOUND (*sirilpy.enums.CommandStatus attribute*), 588
CMD_NOT_SCRIPTABLE (*sirilpy.enums.CommandStatus attribute*), 588
CMD_OK (*sirilpy.enums.CommandStatus attribute*), 588
CMD_ONLY_SINGLE_IMAGE (*sirilpy.enums.CommandStatus attribute*), 588
CMD_SELECTION_ERROR (*sirilpy.enums.CommandStatus attribute*), 588
CMD_SEQUENCE_NOT_FOUND (*sirilpy.enums.CommandStatus attribute*), 588
CMD_THREAD_RUNNING (*sirilpy.enums.CommandStatus attribute*), 588
CMD_WRONG_N_ARG (*sirilpy.enums.CommandStatus attribute*), 588
color (*sirilpy.models.Polygon attribute*), 584
COLOR_CALIBRATION (*sirilpy.enums.DialogID attribute*), 588
color_managed (*sirilpy.models.FFit attribute*), 572
command_error_message() (*sirilpy.connection.SirilInterface method*), 545
CommandError, 606
CommandStatus (*class in sirilpy.enums*), 587
COMPOSITION_DIALOG (*sirilpy.enums.DialogID attribute*), 588
COMPSTARS_DIALOG (*sirilpy.enums.DialogID attribute*), 588
conesearch, **477**
confirm_messagebox() (*sirilpy.connection.SirilInterface method*), 545
connect() (*sirilpy.connection.SirilInterface method*), 546
contains_point() (*sirilpy.models.Polygon method*), 583
convert, **478**
convertraw, **478**
cosme, **478**
cosme_cfa, **479**
COSMETIC_DIALOG (*sirilpy.enums.DialogID attribute*), 589
create_new_seq() (*sirilpy.connection.SirilInterface method*), 546
create_tooltip() (*in module sirilpy.tksiril*), 604
crop, **479**
current (*sirilpy.models.Sequence attribute*), 586
CURVES_DIALOG (*sirilpy.enums.DialogID attribute*), 589
cvf (*sirilpy.models.FKeywords attribute*), 574
CWD_DIALOG (*sirilpy.enums.DialogID attribute*), 589

## D

DARK (*sirilpy.enums.ImageType attribute*), 591
data (*sirilpy.models.FFit property*), 572
data_max (*sirilpy.models.FKeywords attribute*), 574
data_min (*sirilpy.models.FKeywords attribute*), 574
DataError, 606
date (*sirilpy.models.FKeywords attribute*), 574
date_obs (*sirilpy.models.FKeywords attribute*), 574
date_obs (*sirilpy.models.ImgData attribute*), 580

ddp, **480**
dec (*sirilpy.models.FKeywords attribute*), 575
dec (*sirilpy.models.PSFStar attribute*), 581
DECONV_DIALOG (*sirilpy.enums.DialogID attribute*), 589
DEFAULT (*sirilpy.enums.LogColor attribute*), 591
denoise, **480**
DENOISE_DIALOG (*sirilpy.enums.DialogID attribute*), 589
deserialize() (*sirilpy.models.BGSample class method*), 570
deserialize() (*sirilpy.models.FKeywords class method*), 573
deserialize() (*sirilpy.models.ImageAnalysis class method*), 577
deserialize() (*sirilpy.models.ImageStats class method*), 578
deserialize() (*sirilpy.models.ImgData class method*), 579
deserialize() (*sirilpy.models.PSFStar class method*), 580
deserialize() (*sirilpy.models.RegData class method*), 585
deserialize_polygon() (*sirilpy.models.Polygon class method*), 583
deserialize_polygon_list() (*sirilpy.models.Polygon class method*), 583
detect_gpu_capabilities() (*in module sirilpy.gpuhelper*), 602
DIALOG_FFT (*sirilpy.enums.DialogID attribute*), 589
DIALOG_STAR_REMIX (*sirilpy.enums.DialogID attribute*), 589
DialogID (*class in sirilpy.enums*), 588
DialogReq (*class in sirilpy.enums*), 590
dir, **481**
disconnect() (*sirilpy.connection.SirilInterface method*), 546
disto, **481**
DISTO_FILE (*sirilpy.enums.DistoType attribute*), 590
DISTO_FILE_COMET (*sirilpy.enums.DistoType attribute*), 590
DISTO_FILES (*sirilpy.enums.DistoType attribute*), 590
DISTO_IMAGE (*sirilpy.enums.DistoType attribute*), 590
DISTO_MASTER (*sirilpy.enums.DistoType attribute*), 590
DISTO_UNDEF (*sirilpy.enums.DistoType attribute*), 591
DistoData (*class in sirilpy.models*), 571
distoparam (*sirilpy.models.Sequence attribute*), 586
DistoType (*class in sirilpy.enums*), 590
DOUBLE_IMG (*sirilpy.enums.BitpixType attribute*), 587
download_with_progress() (*in module sirilpy.utility*), 595
dumpheader, **481**

## E

elevate() (*in module sirilpy.tksiril*), 604
end (*sirilpy.models.Sequence attribute*), 586
ensure_data_type() (*sirilpy.models.FFit method*), 571
ensure_installed() (*in module sirilpy.utility*), 595
ensure_jax() (*sirilpy.gpuhelper.JaxHelper method*), 598
ensure_onnxruntime() (*sirilpy.gpuhelper.ONNXHelper method*), 599
ensure_torch() (*sirilpy.gpuhelper.TorchHelper method*), 601
entropy, **481**
epf, **482**
EPF_DIALOG (*sirilpy.enums.DialogID attribute*), 589
error_messagebox() (*sirilpy.connection.SirilInterface method*), 546
estimate_noise() (*sirilpy.models.FFit method*), 571
exit, **482**
expend (*sirilpy.models.FKeywords attribute*), 575
exposure (*sirilpy.models.FKeywords attribute*), 575
exposure (*sirilpy.models.Sequence attribute*), 586
expstart (*sirilpy.models.FKeywords attribute*), 575
extract, **482**
EXTRACT_CHANNEL_DIALOG (*sirilpy.enums.DialogID attribute*), 589
extract_Green, **483**
extract_Ha, **483**
extract_HaOIII, **483**
EXTRACT_WAVELETS_LAYERS_DIALOG (*sirilpy.enums.DialogID attribute*), 589

## F

fdiv, **484**
ffill, **484**
FFit (*class in sirilpy.models*), 571
fftd, **484**
ffti, **484**
fhi (*sirilpy.models.FKeywords attribute*), 575
FILE_INFORMATION (*sirilpy.enums.DialogID attribute*), 589
filename (*sirilpy.models.DistoData attribute*), 571
filename (*sirilpy.models.FKeywords attribute*), 575
filenum (*sirilpy.models.ImgData attribute*), 580
fill, **485**
fill (*sirilpy.models.Polygon attribute*), 584
filter (*sirilpy.models.FKeywords attribute*), 575
filter (*sirilpy.models.ImageAnalysis attribute*), 578
find_cosme, **485**
find_cosme_cfa, **485**
find_hot, **485**
findcompstars, **486**
findstar, **486**
fix_xtrans, **487**
fixbanding, **487**
fixed (*sirilpy.models.Sequence attribute*), 586
FKeywords (*class in sirilpy.models*), 573

FLAT (*sirilpy.enums.ImageType attribute*), 591
FLEN (*sirilpy.models.ImageAnalysis attribute*), 578
FLEN_VALUE (*sirilpy.models.FKeywords attribute*), 573
flength (*sirilpy.models.FKeywords attribute*), 575
flo (*sirilpy.models.FKeywords attribute*), 575
FLOAT_IMG (*sirilpy.enums.BitpixType attribute*), 587
fmedian, **488**
fmul, **488**
focal_length (*sirilpy.models.FKeywords attribute*), 575
focname (*sirilpy.models.FKeywords attribute*), 575
foctemp (*sirilpy.models.FKeywords attribute*), 575
focuspos (*sirilpy.models.FKeywords attribute*), 575
focussz (*sirilpy.models.FKeywords attribute*), 575
FPoint (*class in sirilpy.models*), 577
from_name() (*sirilpy.enums.DialogID class method*), 588
from_rectangle() (*sirilpy.models.Polygon class method*), 583
fwhm (*sirilpy.models.ImageAnalysis attribute*), 578
fwhm (*sirilpy.models.RegData attribute*), 585
fwhmx (*sirilpy.models.PSFStar attribute*), 581
fwhmx_arcsec (*sirilpy.models.PSFStar attribute*), 581
fwhmy (*sirilpy.models.PSFStar attribute*), 581
fwhmy_arcsec (*sirilpy.models.PSFStar attribute*), 581
fz (*sirilpy.models.Sequence attribute*), 586

## G

gain (*sirilpy.models.FKeywords attribute*), 575
gauss, **488**
GAUSSIAN (*sirilpy.enums.StarProfile attribute*), 593
get, **488**
get_bounds() (*sirilpy.models.Polygon method*), 584
get_channel() (*sirilpy.models.FFit method*), 572
get_execution_providers_ordered() (*sirilpy.gpuhelper.ONNXHelper method*), 599
get_gpu_detection_report() (*in module sirilpy.gpuhelper*), 602
get_image() (*sirilpy.connection.SirilInterface method*), 547
get_image_bgsamples() (*sirilpy.connection.SirilInterface method*), 547
get_image_filename() (*sirilpy.connection.SirilInterface method*), 547
get_image_fits_header() (*sirilpy.connection.SirilInterface method*), 548
get_image_history() (*sirilpy.connection.SirilInterface method*), 548
get_image_iccprofile() (*sirilpy.connection.SirilInterface method*), 548
get_image_keywords() (*sirilpy.connection.SirilInterface method*), 549
get_image_pixeldata() (*sirilpy.connection.SirilInterface method*), 549
get_image_shape() (*sirilpy.connection.SirilInterface method*), 549
get_image_stars() (*sirilpy.connection.SirilInterface method*), 549
get_image_stats() (*sirilpy.connection.SirilInterface method*), 550
get_image_unknown_keys() (*sirilpy.connection.SirilInterface method*), 550
get_max_x() (*sirilpy.models.Polygon method*), 584
get_max_y() (*sirilpy.models.Polygon method*), 584
get_min_x() (*sirilpy.models.Polygon method*), 584
get_min_y() (*sirilpy.models.Polygon method*), 584
get_recommended_backend() (*sirilpy.gpuhelper.JaxHelper method*), 598
get_recommended_backend() (*sirilpy.gpuhelper.ONNXHelper method*), 600
get_recommended_backend() (*sirilpy.gpuhelper.TorchHelper method*), 602
get_selection_star() (*sirilpy.connection.SirilInterface method*), 550
get_selection_stats() (*sirilpy.connection.SirilInterface method*), 551
get_seq() (*sirilpy.connection.SirilInterface method*), 551
get_seq_distodata() (*sirilpy.connection.SirilInterface method*), 552
get_seq_frame() (*sirilpy.connection.SirilInterface method*), 552
get_seq_frame_filename() (*sirilpy.connection.SirilInterface method*), 552
get_seq_frame_header() (*sirilpy.connection.SirilInterface method*), 553
get_seq_frame_pixeldata() (*sirilpy.connection.SirilInterface method*), 553
get_seq_imgdata() (*sirilpy.connection.SirilInterface method*), 554
get_seq_regdata() (*sirilpy.connection.SirilInterface method*), 554
get_seq_stats() (*sirilpy.connection.SirilInterface method*), 554
get_siril_active_vport() (*sirilpy.connection.SirilInterface method*), 555
get_siril_config() (*sirilpy.connection.SirilInterface method*), 555
get_siril_configdir() (*sirilpy.connection.SirilInterface method*), 555
get_siril_display_iccprofile() (*sirilpy.connection.SirilInterface method*), 555
get_siril_log() (*sirilpy.connection.SirilInterface method*), 556
get_siril_panzoom() (*sirilpy.connection.SirilInterface method*), 556
get_siril_selection() (*sirilpy.connection.SirilInterface method*), 556

get_siril_slider_state() (*sirilpy.connection.SirilInterface method*), 556
get_siril_stf() (*sirilpy.connection.SirilInterface method*), 557
get_siril_stf_linked() (*sirilpy.connection.SirilInterface method*), 557
get_siril_systemdatadir() (*sirilpy.connection.SirilInterface method*), 557
get_siril_userdatadir() (*sirilpy.connection.SirilInterface method*), 557
get_siril_wd() (*sirilpy.connection.SirilInterface method*), 557
get_torch_device() (*sirilpy.gpuhelper.TorchHelper method*), 602
getref, **489**
ght, **489**
GHT_DIALOG (*sirilpy.enums.DialogID attribute*), 589
GREEN (*sirilpy.enums.LogColor attribute*), 591
GREEN (*sirilpy.enums.SirilVport attribute*), 592
grey_flat, **489**

## H

H (*sirilpy.models.RegData attribute*), 585
h00 (*sirilpy.models.Homography attribute*), 577
h01 (*sirilpy.models.Homography attribute*), 577
h02 (*sirilpy.models.Homography attribute*), 577
h10 (*sirilpy.models.Homography attribute*), 577
h11 (*sirilpy.models.Homography attribute*), 577
h12 (*sirilpy.models.Homography attribute*), 577
h20 (*sirilpy.models.Homography attribute*), 577
h21 (*sirilpy.models.Homography attribute*), 577
h22 (*sirilpy.models.Homography attribute*), 577
has_saturated (*sirilpy.models.PSFStar attribute*), 581
header (*sirilpy.models.FFit attribute*), 572
height (*sirilpy.models.FFit property*), 572
height (*sirilpy.models.ImageAnalysis attribute*), 578
help, **490**
hi (*sirilpy.models.FKeywords attribute*), 575
HISTEQ_DISPLAY (*sirilpy.enums.STFType attribute*), 592
histo, **490**
HISTOGRAM_DIALOG (*sirilpy.enums.DialogID attribute*), 589
history (*sirilpy.models.FFit attribute*), 572
Homography (*class in sirilpy.models*), 577
human_readable_size() (*in module sirilpy.utility*), 595
HYPHENS (*sirilpy.enums.PlotType attribute*), 591

## I

iadd, **490**
icc_assign, **490**
icc_convert_to, **491**
ICC_DIALOG (*sirilpy.enums.DialogID attribute*), 589
icc_profile (*sirilpy.models.FFit property*), 572
icc_remove, **491**
idiv, **491**
image_lock() (*sirilpy.connection.SirilInterface method*), 558
image_type (*sirilpy.models.FKeywords attribute*), 575
ImageAnalysis (*class in sirilpy.models*), 577
ImageDialogOpenError, 606
ImageStats (*class in sirilpy.models*), 578
ImageType (*class in sirilpy.enums*), 591
imagetype (*sirilpy.models.ImageAnalysis attribute*), 578
IMG (*sirilpy.enums.DialogReq attribute*), 590
ImgData (*class in sirilpy.models*), 579
imgparam (*sirilpy.models.Sequence attribute*), 586
import_onnxruntime() (*sirilpy.gpuhelper.ONNXHelper method*), 600
imul, **492**
incl (*sirilpy.models.ImgData attribute*), 580
index (*sirilpy.models.DistoData attribute*), 571
info_messagebox() (*sirilpy.connection.SirilInterface method*), 558
Inliers (*sirilpy.models.Homography attribute*), 577
inspector, **492**
install_jax() (*sirilpy.gpuhelper.JaxHelper method*), 598
install_onnxruntime() (*sirilpy.gpuhelper.ONNXHelper method*), 600
install_torch() (*sirilpy.gpuhelper.TorchHelper method*), 602
instrume (*sirilpy.models.FKeywords attribute*), 575
invght, **492**
invmodasinh, **492**
invmtf, **493**
is_cli() (*sirilpy.connection.SirilInterface method*), 559
is_image_loaded() (*sirilpy.connection.SirilInterface method*), 559
is_jax_installed() (*sirilpy.gpuhelper.JaxHelper method*), 598
is_onnxruntime_installed() (*sirilpy.gpuhelper.ONNXHelper method*), 600
is_sequence_loaded() (*sirilpy.connection.SirilInterface method*), 559
is_torch_installed() (*sirilpy.gpuhelper.TorchHelper method*), 602
is_variable (*sirilpy.models.Sequence attribute*), 586
iso_speed (*sirilpy.models.FKeywords attribute*), 575
isub, **493**

## J

JaxHelper (*class in sirilpy.gpuhelper*), 598
jsonmetadata, **493**

## K

keywords (*sirilpy.models.FFit attribute*), 572
KEYWORDS_DIALOG (*sirilpy.enums.DialogID attribute*), 589

KEYWORDS_FORMAT (*sirilpy.models.FKeywords attribute*), 573
KEYWORDS_SIZE (*sirilpy.models.FKeywords attribute*), 574

## L

layer (*sirilpy.models.PSFStar attribute*), 581
legend (*sirilpy.models.Polygon attribute*), 585
LIGHT (*sirilpy.enums.ImageType attribute*), 591
light_curve, **494**
limit, **494**
LINEAR_DISPLAY (*sirilpy.enums.STFType attribute*), 592
linear_match, **495**
LINEARMATCH_DIALOG (*sirilpy.enums.DialogID attribute*), 589
LINES (*sirilpy.enums.PlotType attribute*), 591
LINESHYPHENS (*sirilpy.enums.PlotType attribute*), 591
LINESMARKS (*sirilpy.enums.PlotType attribute*), 591
LINESPOINTS (*sirilpy.enums.PlotType attribute*), 591
link, **495**
linstretch, **495**
livestack, **495**
livetime (*sirilpy.models.FKeywords attribute*), 575
lo (*sirilpy.models.FKeywords attribute*), 576
load, **496**
load_image_from_file() (*sirilpy.connection.SirilInterface method*), 559
load_seq, **496**
location (*sirilpy.models.ImageStats attribute*), 579
log, **496**
log() (*sirilpy.connection.SirilInterface method*), 560
LOG_DISPLAY (*sirilpy.enums.STFType attribute*), 592
LogColor (*class in sirilpy.enums*), 591
LONG_IMG (*sirilpy.enums.BitpixType attribute*), 587
ls, **497**

## M

mad (*sirilpy.models.ImageStats attribute*), 579
mag (*sirilpy.models.PSFStar attribute*), 582
makepsf, **497**
MARKS (*sirilpy.enums.PlotType attribute*), 591
match_theme_to_siril() (*in module sirilpy.tksiril*), 604
max (*sirilpy.models.BGSample attribute*), 570
max (*sirilpy.models.ImageStats attribute*), 579
maxi (*sirilpy.models.FFit attribute*), 572
mean (*sirilpy.models.BGSample attribute*), 570
mean (*sirilpy.models.ImageStats attribute*), 579
median (*sirilpy.models.BGSample attribute*), 570
median (*sirilpy.models.ImageStats attribute*), 579
MEDIAN_DIALOG (*sirilpy.enums.DialogID attribute*), 589
merge, **498**
merge_cfa, **498**
MERGE_CFA_DIALOG (*sirilpy.enums.DialogID attribute*), 589
min (*sirilpy.models.BGSample attribute*), 571
min (*sirilpy.models.ImageStats attribute*), 579
mini (*sirilpy.models.FFit attribute*), 572
MINMAX (*sirilpy.enums.SlidersMode attribute*), 592
MIPSLOHI (*sirilpy.enums.SlidersMode attribute*), 592
mirrorx, **499**
mirrorx_single, **499**
mirrory, **499**
modasinh, **499**
module
    sirilpy.connection, 544
    sirilpy.enums, 587
    sirilpy.exceptions, 606
    sirilpy.gpuhelper, 598
    sirilpy.models, 570
    sirilpy.plot, 593
    sirilpy.tksiril, 603
    sirilpy.utility, 594
MOFFAT (*sirilpy.enums.StarProfile attribute*), 593
MOFFAT_FIXED (*sirilpy.enums.StarProfile attribute*), 593
MONO (*sirilpy.enums.DialogReq attribute*), 590
MONO (*sirilpy.enums.SirilVport attribute*), 592
MouseModeError, 606
mtf, **500**

## N

naxes (*sirilpy.models.FFit property*), 573
naxis (*sirilpy.models.FFit attribute*), 573
nb_layers (*sirilpy.models.Sequence attribute*), 586
nbstars (*sirilpy.models.ImageAnalysis attribute*), 578
needs_module_version() (*in module sirilpy.utility*), 596
neg, **500**
neg_ratio (*sirilpy.models.FFit attribute*), 573
new, **500**
ngoodpix (*sirilpy.models.ImageStats attribute*), 579
NINA_LIGHT_CURVE (*sirilpy.enums.DialogID attribute*), 589
NoImageError, 606
NONE (*sirilpy.enums.DialogReq attribute*), 590
normValue (*sirilpy.models.ImageStats attribute*), 579
NoSequenceError, 606
nozero, **501**
number (*sirilpy.models.Sequence attribute*), 586
number_of_stars (*sirilpy.models.RegData attribute*), 585

## O

objctdec (*sirilpy.models.FKeywords attribute*), 576
objctra (*sirilpy.models.FKeywords attribute*), 576
object (*sirilpy.models.FKeywords attribute*), 576
observer (*sirilpy.models.FKeywords attribute*), 576

offline, **501**
offset, **501**
offset (*sirilpy.models.FKeywords attribute*), 576
online, **501**
ONNXHelper (*class in sirilpy.gpuhelper*), 599
OPEN_DIALOG (*sirilpy.enums.DialogID attribute*), 589
open_dialog() (*sirilpy.connection.SirilInterface method*), 560
orig_bitpix (*sirilpy.models.FFit attribute*), 573
overlay_add_polygon() (*sirilpy.connection.SirilInterface method*), 560
overlay_clear_polygons() (*sirilpy.connection.SirilInterface method*), 561
overlay_delete_polygon() (*sirilpy.connection.SirilInterface method*), 561
overlay_draw_polygon() (*sirilpy.connection.SirilInterface method*), 561
overlay_get_polygon() (*sirilpy.connection.SirilInterface method*), 561
overlay_get_polygons_list() (*sirilpy.connection.SirilInterface method*), 562

## P

pair_matched (*sirilpy.models.Homography attribute*), 577
parse, **502**
parse_fits_header() (*in module sirilpy.utility*), 596
pcc, **502**
PCC_DIALOG (*sirilpy.enums.DialogID attribute*), 589
phot_is_valid (*sirilpy.models.PSFStar attribute*), 582
pix2radec() (*sirilpy.connection.SirilInterface method*), 562
PIXEL_MATH_DIALOG (*sirilpy.enums.DialogID attribute*), 589
pixel_size_x (*sirilpy.models.FKeywords attribute*), 576
pixel_size_y (*sirilpy.models.FKeywords attribute*), 576
platesolve, **502**
PlotData (*class in sirilpy.plot*), 593
PlotType (*class in sirilpy.enums*), 591
PLTSOLVD (*sirilpy.enums.DialogReq attribute*), 590
pltsolvd (*sirilpy.models.FKeywords attribute*), 576
pltsolvd_comment (*sirilpy.models.FKeywords attribute*), 576
pm, **503**
POINTS (*sirilpy.enums.PlotType attribute*), 591
points (*sirilpy.models.Polygon attribute*), 585
Polygon (*class in sirilpy.models*), 583
polygon_id (*sirilpy.models.Polygon attribute*), 585
position (*sirilpy.models.BGSample attribute*), 571
PREFS_DIALOG (*sirilpy.enums.DialogID attribute*), 589
ProcessingThreadBusyError, 606
profile, **504**
profile (*sirilpy.models.PSFStar attribute*), 582
program (*sirilpy.models.FKeywords attribute*), 576
psf, **504**
PSFStar (*class in sirilpy.models*), 580
pwd, **505**
pyscript, **505**

## Q

quality (*sirilpy.models.RegData attribute*), 585

## R

R (*sirilpy.models.PSFStar attribute*), 581
ra (*sirilpy.models.FKeywords attribute*), 576
ra (*sirilpy.models.PSFStar attribute*), 582
radec2pix() (*sirilpy.connection.SirilInterface method*), 562
RED (*sirilpy.enums.LogColor attribute*), 591
RED (*sirilpy.enums.SirilVport attribute*), 592
redo() (*sirilpy.connection.SirilInterface method*), 563
reference_image (*sirilpy.models.Sequence attribute*), 586
RegData (*class in sirilpy.models*), 585
register, **505**
regparam (*sirilpy.models.Sequence attribute*), 586
reloadscripts, **507**
requires, **507**
resample, **507**
RESAMPLE_DIALOG (*sirilpy.enums.DialogID attribute*), 589
reset_progress() (*sirilpy.connection.SirilInterface method*), 563
RGB (*sirilpy.enums.DialogReq attribute*), 590
RGB (*sirilpy.enums.SirilVport attribute*), 592
rgbcomp, **507**
RGBPLTSOLVD (*sirilpy.enums.DialogReq attribute*), 590
rgradient, **508**
RGRADIENT_DIALOG (*sirilpy.enums.DialogID attribute*), 589
rl, **508**
rmgreen, **509**
rmse (*sirilpy.models.PSFStar attribute*), 582
rotate, **509**
rotatePi, **509**
ROTATION_DIALOG (*sirilpy.enums.DialogID attribute*), 589
roundness (*sirilpy.models.ImageAnalysis attribute*), 578
roundness (*sirilpy.models.RegData attribute*), 585
row_order (*sirilpy.models.FKeywords attribute*), 576
run() (*sirilpy.gpuhelper.ONNXHelper method*), 600
rx (*sirilpy.models.ImgData attribute*), 580
rx (*sirilpy.models.Sequence attribute*), 586
ry (*sirilpy.models.ImgData attribute*), 580
ry (*sirilpy.models.Sequence attribute*), 586

## S


s_Bmag (*sirilpy.models.PSFStar attribute*), 582
s_mag (*sirilpy.models.PSFStar attribute*), 582
S_PCC_DIALOG (*sirilpy.enums.DialogID attribute*), 590
safe_subprocess_Popen() (*in module sirilpy.utility*), 597
safe_subprocess_run() (*in module sirilpy.utility*), 597
SALMON (*sirilpy.enums.LogColor attribute*), 591
sat (*sirilpy.models.PSFStar attribute*), 582
satu, **510**
SATU_DIALOG (*sirilpy.enums.DialogID attribute*), 589
save, **510**
save_image_file() (*sirilpy.connection.SirilInterface method*), 563
SAVEAS_DIALOG (*sirilpy.enums.DialogID attribute*), 589
savebmp, **510**
savejpg, **510**
savejxl, **511**
savepng, **511**
savepnm, **511**
savetif, **512**
savetif32, **512**
savetif8, **512**
sb, **512**
scale (*sirilpy.models.ImageStats attribute*), 579
SCNR_DIALOG (*sirilpy.enums.DialogID attribute*), 589
ScrollableFrame (*class in sirilpy.tksiril*), 603
select, **513**
selnum (*sirilpy.models.Sequence attribute*), 586
SEQ (*sirilpy.enums.DialogReq attribute*), 590
SEQ_AVI (*sirilpy.enums.SequenceType attribute*), 592
SEQ_FITSEQ (*sirilpy.enums.SequenceType attribute*), 592
SEQ_INTERNAL (*sirilpy.enums.SequenceType attribute*), 592
SEQ_REGULAR (*sirilpy.enums.SequenceType attribute*), 592
SEQ_SER (*sirilpy.enums.SequenceType attribute*), 592
seqapplyreg, **513**
seqccm, **515**
seqclean, **515**
seqcosme, **515**
seqcosme_cfa, **516**
seqcrop, **516**
seqextract_Green, **516**
seqextract_Ha, **517**
seqextract_HaOIII, **517**
seqfind_cosme, **517**
seqfind_cosme_cfa, **517**
seqfindstar, **518**
seqfixbanding, **518**
seqght, **518**
seqheader, **519**
seqinvght, **519**
seqinvmodasinh, **519**
seqlinstretch, **519**
SEQLIST_DIALOG (*sirilpy.enums.DialogID attribute*), 589
seqmerge_cfa, **520**
seqmodasinh, **520**
seqmtf, **520**
seqname (*sirilpy.models.Sequence attribute*), 586
seqplatesolve, **522**
seqprofile, **521**
seqpsf, **521**
seqresample, **523**
seqrl, **523**
seqsb, **523**
seqsetmag, **524**
seqsplit_cfa, **524**
seqstarnet, **524**
seqstat, **525**
seqsubsky, **525**
seqtilt, **525**
Sequence (*class in sirilpy.models*), 585
SequenceType (*class in sirilpy.enums*), 592
sequnsetmag, **526**
sequpdate_key, **526**
seqwiener, **526**
serialize() (*sirilpy.models.ImageAnalysis method*), 578
serialize() (*sirilpy.models.Polygon method*), 584
serialize() (*sirilpy.plot.PlotData class method*), 593
SeriesData (*class in sirilpy.plot*), 594
set, **527**
set16bits, **527**
set32bits, **527**
set_image_bgsamples() (*sirilpy.connection.SirilInterface method*), 563
set_image_filename() (*sirilpy.connection.SirilInterface method*), 564
set_image_iccprofile() (*sirilpy.connection.SirilInterface method*), 564
set_image_metadata_from_header_string() (*sirilpy.connection.SirilInterface method*), 565
set_image_pixeldata() (*sirilpy.connection.SirilInterface method*), 565
set_seq_frame_incl() (*sirilpy.connection.SirilInterface method*), 566
set_seq_frame_pixeldata() (*sirilpy.connection.SirilInterface method*), 566
set_siril_pan() (*sirilpy.connection.SirilInterface method*), 567
set_siril_selection() (*sirilpy.connection.SirilInterface method*), 567
set_siril_slider_lohi() (*sirilpy.connection.SirilInterface method*), 567
set_siril_slider_mode() (*sirilpy.connection.SirilInterface method*), 568

set_siril_stf() (*sirilpy.connection.SirilInterface method*), 568
set_siril_stf_linked() (*sirilpy.connection.SirilInterface method*), 568
set_siril_zoom() (*sirilpy.connection.SirilInterface method*), 568
set_temp (*sirilpy.models.FKeywords attribute*), 576
setcompress, **527**
setcpu, **528**
setext, **528**
setfindstar, **528**
setmag, **529**
setmem, **529**
setphot, **530**
setref, **530**
SharedMemoryError, 607
SHORT_IMG (*sirilpy.enums.BitpixType attribute*), 587
show, **531**
sigma (*sirilpy.models.ImageStats attribute*), 579
SirilConnectionError, 607
SirilError, 607
SirilInterface (*class in sirilpy.connection*), 544
sirilpy.connection
    module, 544
sirilpy.enums
    module, 587
sirilpy.exceptions
    module, 606
sirilpy.gpuhelper
    module, 598
sirilpy.models
    module, 570
sirilpy.plot
    module, 593
sirilpy.tksiril
    module, 603
sirilpy.utility
    module, 594
SirilVport (*class in sirilpy.enums*), 592
siteelev (*sirilpy.models.FKeywords attribute*), 576
sitelat (*sirilpy.models.FKeywords attribute*), 576
sitelat_str (*sirilpy.models.FKeywords attribute*), 576
sitelong (*sirilpy.models.FKeywords attribute*), 576
sitelong_str (*sirilpy.models.FKeywords attribute*), 576
size (*sirilpy.models.BGSample attribute*), 571
SlidersMode (*class in sirilpy.enums*), 592
SNR (*sirilpy.models.PSFStar attribute*), 581
spcc, **531**
spcc_list, **532**
split, **532**
split_cfa, **532**
SPLIT_CFA_DIALOG (*sirilpy.enums.DialogID attribute*), 589
SQRT_DISPLAY (*sirilpy.enums.STFType attribute*), 592
sqrtbwmv (*sirilpy.models.ImageStats attribute*), 579
SQUARED_DISPLAY (*sirilpy.enums.STFType attribute*), 592
stack, **533**
stackall, **534**
stackcnt (*sirilpy.models.FKeywords attribute*), 576
standard_style() (*in module sirilpy.tksiril*), 604
star_name (*sirilpy.models.PSFStar attribute*), 582
starnet, **535**
STARNET_DIALOG (*sirilpy.enums.DialogID attribute*), 590
StarProfile (*class in sirilpy.enums*), 592
STARS_LIST_WINDOW (*sirilpy.enums.DialogID attribute*), 590
start_ls, **535**
stat, **535**
STAT_WINDOW (*sirilpy.enums.DialogID attribute*), 590
stats (*sirilpy.models.FFit attribute*), 573
stats (*sirilpy.models.Sequence attribute*), 587
status() (*sirilpy.gpuhelper.JaxHelper method*), 598
status() (*sirilpy.gpuhelper.ONNXHelper method*), 601
status() (*sirilpy.gpuhelper.TorchHelper method*), 602
status_code (*sirilpy.exceptions.CommandError attribute*), 606
STFType (*class in sirilpy.enums*), 591
stop_ls, **536**
subsky, **536**
SuppressedStderr (*class in sirilpy.utility*), 594
SuppressedStdout (*class in sirilpy.utility*), 594
sx (*sirilpy.models.PSFStar attribute*), 582
sx_err (*sirilpy.models.PSFStar attribute*), 582
sy (*sirilpy.models.PSFStar attribute*), 582
sy_err (*sirilpy.models.PSFStar attribute*), 582
synthstar, **536**

## T

telescop (*sirilpy.models.FKeywords attribute*), 576
test_jax() (*sirilpy.gpuhelper.JaxHelper method*), 599
test_onnxruntime() (*sirilpy.gpuhelper.ONNXHelper method*), 601
test_torch() (*sirilpy.gpuhelper.TorchHelper method*), 602
thresh, **537**
threshhi, **537**
threshlo, **537**
tilt, **537**
timestamp (*sirilpy.models.ImageAnalysis attribute*), 578
top_down (*sirilpy.models.FFit attribute*), 573
TorchHelper (*class in sirilpy.gpuhelper*), 601
total (*sirilpy.models.ImageStats attribute*), 579
trixel, **538**
truncate_utf8() (*in module sirilpy.utility*), 597
type (*sirilpy.models.Sequence attribute*), 587

## U

unclipstars, **538**
undo() (*sirilpy.connection.SirilInterface method*), 569
undo_save_state() (*sirilpy.connection.SirilInterface method*), 569
uninstall_jax() (*sirilpy.gpuhelper.JaxHelper method*), 599
uninstall_onnxruntime() (*sirilpy.gpuhelper.ONNXHelper method*), 601
uninstall_package() (*in module sirilpy.utility*), 597
uninstall_torch() (*sirilpy.gpuhelper.TorchHelper method*), 602
units (*sirilpy.models.PSFStar attribute*), 582
UNKNOWN (*sirilpy.enums.ImageType attribute*), 591
unknown_keys (*sirilpy.models.FFit attribute*), 573
unpurple, **538**
UNPURPLE_DIALOG (*sirilpy.enums.DialogID attribute*), 590
unselect, **539**
unsetmag, **539**
unsharp, **539**
update_key, **539**
update_progress() (*sirilpy.connection.SirilInterface method*), 569
update_stats() (*sirilpy.models.FFit method*), 572
USER (*sirilpy.enums.SlidersMode attribute*), 592
USHORT_IMG (*sirilpy.enums.BitpixType attribute*), 587

## V

valid (*sirilpy.models.BGSample attribute*), 571
velocity (*sirilpy.models.DistoData attribute*), 571
visu, **540**

## W

warning_messagebox() (*sirilpy.connection.SirilInterface method*), 569
wavelet, **540**
WAVELETS_DIALOG (*sirilpy.enums.DialogID attribute*), 590
weighted_fwhm (*sirilpy.models.RegData attribute*), 585
wfwhm (*sirilpy.models.ImageAnalysis attribute*), 578
width (*sirilpy.models.FFit property*), 573
width (*sirilpy.models.ImageAnalysis attribute*), 578
wiener, **540**
wrecons, **541**

## X

x (*sirilpy.models.FPoint attribute*), 577
x0 (*sirilpy.models.PSFStar attribute*), 582
x_err (*sirilpy.models.PSFStar attribute*), 582
xpos (*sirilpy.models.PSFStar attribute*), 582
xy_plot() (*sirilpy.connection.SirilInterface method*), 570

## Y

y (*sirilpy.models.FPoint attribute*), 577
y0 (*sirilpy.models.PSFStar attribute*), 582
y_err (*sirilpy.models.PSFStar attribute*), 582
ypos (*sirilpy.models.PSFStar attribute*), 582