pylablib cam-control Documentation

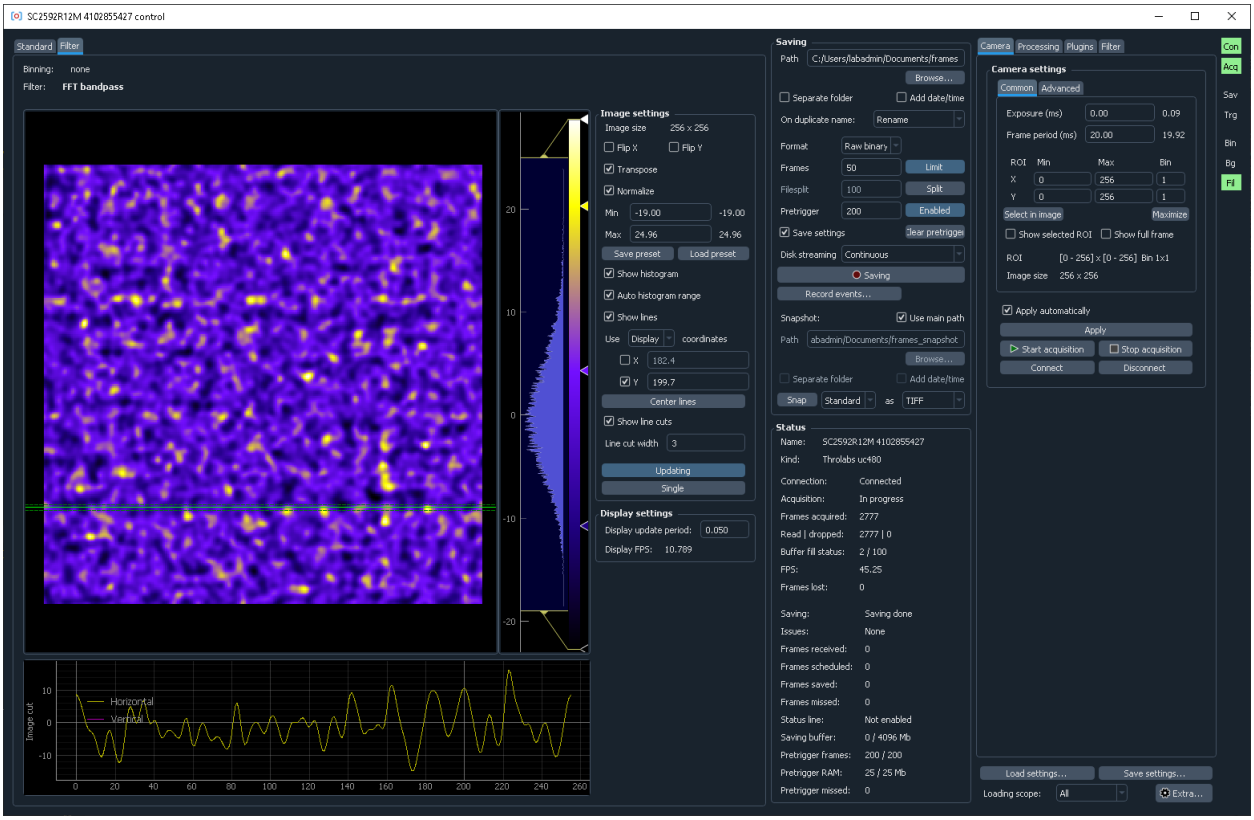
Release 2.2.1

Alexey Shkarin

Oct 06, 2022

CONTENTS:

PyLabLib cam-control aims to provide a convenient interface for controlling cameras and acquiring their data in a unified way.



Features:

- Communication with a variety of cameras: Andor, Hamamatsu, Thorlabs, IDS, PCO, Photometrics, Princeton Instruments, PhotonFocus with IMAQ and Silicon Software frame grabbers.
- Operating at high frame (100 kFPS) and data (1 Gb/s) rates.
- On-line data processing and analysis: *binning*, *background subtraction*, simple built-in *image filters* (Gaussian blur, Fourier filtering), *playback slowdown* for fast process analysis.
- Customization using *user-defined filters* (simple Python code operating on numpy arrays) or control from other software via a *TCP/IP interface*.
- Flexible data acquisition: *pre-trigger buffering*, initiating acquisition on *timer, specific image property*, or *external software signals*.

To install cam-control, download the latest version from GitHub as a self-contained Zip file and then follow further *instructions* for how to run it.

CHAPTER

ONE

OVERVIEW

1.1 Installation

Download the zip file with the latest release and unpack it to the desired location. The software comes with its own isolated Python interpreter with all the basic packages installed, so no additional installations are necessary. Download links to older version are available on the *release history* page.

To run the software, simply execute `control.exe`. On the first run it will suggest to either load settings file (`settings.cfg`) from the previous version, or to detect all connected supported cameras and store their connection parameters. Make sure that at this point all cameras are connected, turned on, and not used in any other software (e.g., Andor Solis, Hokawo, or NI MAX). If new cameras are added to the PC, they can be re-discovered by running `detect.exe`.

If only one camera is found, running `control.exe` will automatically connect to it. Otherwise, a dropdown menu will show up allowing selection of specific cameras.

1.2 Software requirements

Cam-control is built using pylablib and includes all of the cameras supported there. All of these cameras need some kind of drivers and API installed:

- Andor cameras: Andor Solis or Andor SKD.
- Hamamatsu/DCAM cameras: Hamamatsu-supplied software such as Hokawo, or the freely available DCAM API. Keep in mind, that you also need to install the drivers for the corresponding camera type (USB, Ethernet, IEEE 1394). These drivers are in the same installer, but need to be installed separately.
- Thorlabs uc480 and Thorlabs scientific imaging cameras: freely available ThorCam software (no older than v3.5.0 for the scientific imaging cameras).
- IDS uEye cameras: freely available (upon registration) IDS Software Suite.
- PCO cameras: freely available pco.camware software.
- IMAQdx cameras: all the necessary code is contained in the freely available Vision Acquisition Software. However, the IMAQdx part of the software is proprietary, and needs to be purchased to use.
- IMAQ frame grabbers: freely available Vision Acquisition Software. In addition, you would also need to specify the correct camera file, which describes the camera communication protocol details.
- Silicon Software frame grabbers: freely available (upon registration) Silicon Software Runtime Environment (the newest version for 64-bit Windows is 5.7.0).
- PhotonFocus: on top of IMAQ or Silicon Software requirements, it needs freely available (upon registration) PFInstaller software.

- PICam (Princeton Instruments) cameras: freely available PICam software.
- PVCAM (Photometrics) cameras: freely available (upon registration) PVCAM software.
- Basler cameras: freely available (upon registration) Basler pylon Camera Software Suite (the current latest version is 7.1.0).
- BitFlow Axion frame grabbers require several steps:
  - First, you need to install freely available BitFlow SDK 6.5, preferrably into its default folder or, at least, using the default folder name `BitFlow SDK 6.5`.
  - Second, you have to download manufacturer-provided BitFlow Python 3.8 package, extract `BFModule-1.0.1-cp38-cp38-win_amd64.whl` file from there to the `python` folder within `cam-control` (it should contain files like `python.exe`), and run `install-dependencies.bat` script contained in the same folder.
  - Next, you need to specify the correct camera file for your camera using `SysReg` utility located in the `Bin64` folder of your BitFlow installation (by default, `C:\BitFlow SDK 6.5\Bin64`).
  - Afterwards, you should copy this camera file into the main `cam-control` folder (it should contain files like `settings.cfg`). The camera file is located in `Config\Axn` folder within the BitFlow installation (by default, `C:\BitFlow SDK 6.5\Config\Axn`) and has `.bfml` extension, e.g., `PhotonFocus-MV1-D1024E-160-CL.bfml`.
  - You can now search for the cameras by running either `control.exe` for the first time, or `detect.exe` at any point.

Note: Cam-control is only available in 64-bit version, which in most cases means that you must install 64-bit versions of the software described above.

Note: It is strongly recommended that you install the manufacturer software (especially Andor Solis and ThorCam) into their default locations. Otherwise, cam-control might not be able to find the required DLLs and communicate with the cameras. In this case, you would need to specify the location of the necessary DLLs in the *settings file*.

In most cases, you already have the necessary software installed to communicate with the cameras to begin with. As a rule of thumb, if you can open the camera in the manufacturer-supplied software, you can open it in the cam-control as well.

In rare cases you might also need to install Microsoft Visual C++ Redistributable Package. You can obtain it on the Microsoft website.

1.2.1 Specifying camera files

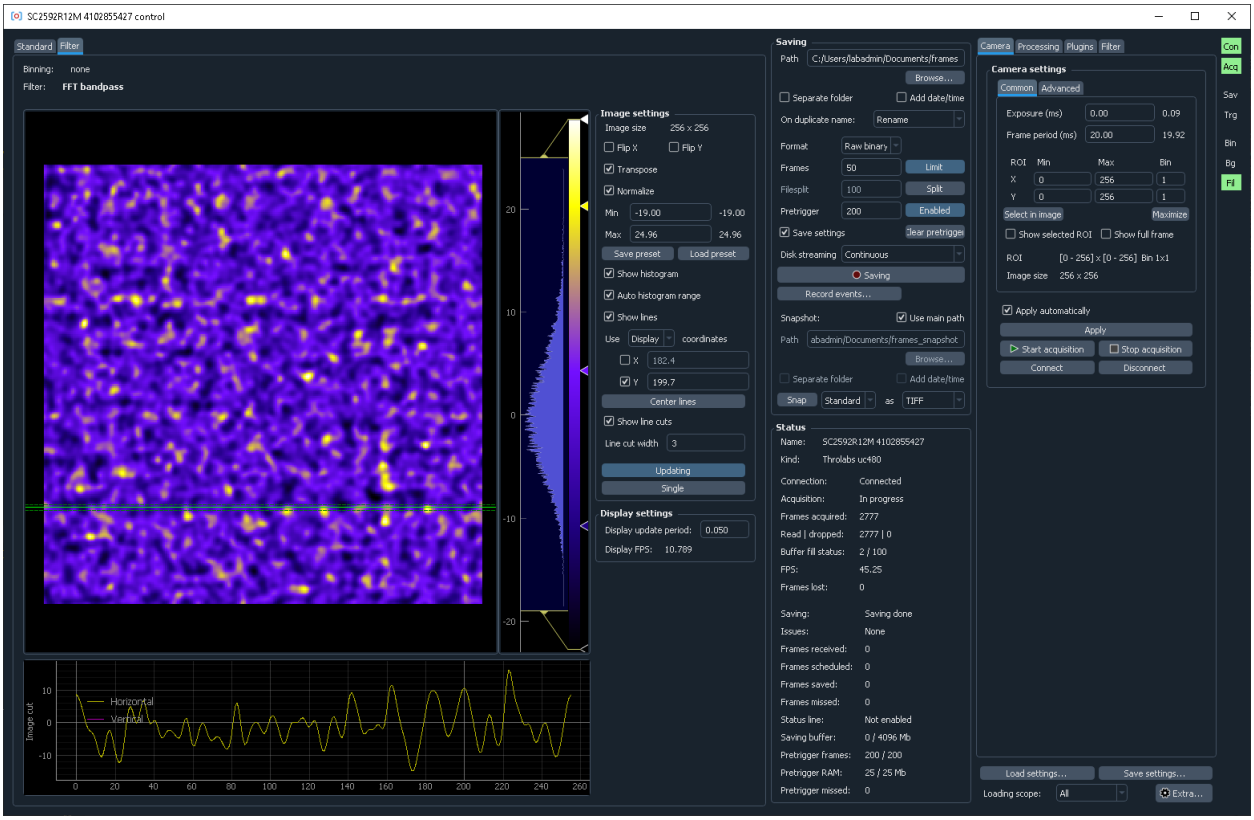
Cameras using NI-IMAQ frame grabbers also need the correct `.icd` camera file for the connected camera, which describes the communication protocol details, sensor size, possible bit depths, etc. Some of the common camera files are already provided when you install Vision Acquisition Software, others need to be obtained from the manufacturer. To specify them, follow the following steps:

1) If you are using a manufacturer-provided file, copy it into the NI-IMAQ camera file storage. By default it is located at `C:\Users\Public\Documents\National Instruments\NI-IMAQ\Data` (the folder should already exist and have many `.icd` files).
2) Run NI MAX, and there find the camera entry in `Devices and Interfaces -> NI-IMAQ Devices -> img<n>: <Frame grabber name>` (e.g., `img0: NI PCIe-1433`) -> `Channel <n>` (0 for single-channel frame grabbers).

3) Right-click on the camera entry and there select `Camera -> <Manufacturer> -> <Camera model corresponding to the file>`.

PhotonFocus provides camera files with PFRemote, and they can be found in `<Photon Focus folder>\PFRemote\fg_files`. There are several files with names like `pfFg_ni_2tap_8bit.icd`, which should be selected based on the desired bit depth (usually 12 bit is preferable, if it is available for your camera) and the number of CameraLink taps (specified in the technical specification found in the camera manual; e.g., MV1-D1024E-160-CL has 2 taps). After specifying the file, you need to also specify the camera pixel depth using PFRemote. The correct setting is located at `Data Output -> Output Mode -> Resolution`.

1.3 General layout

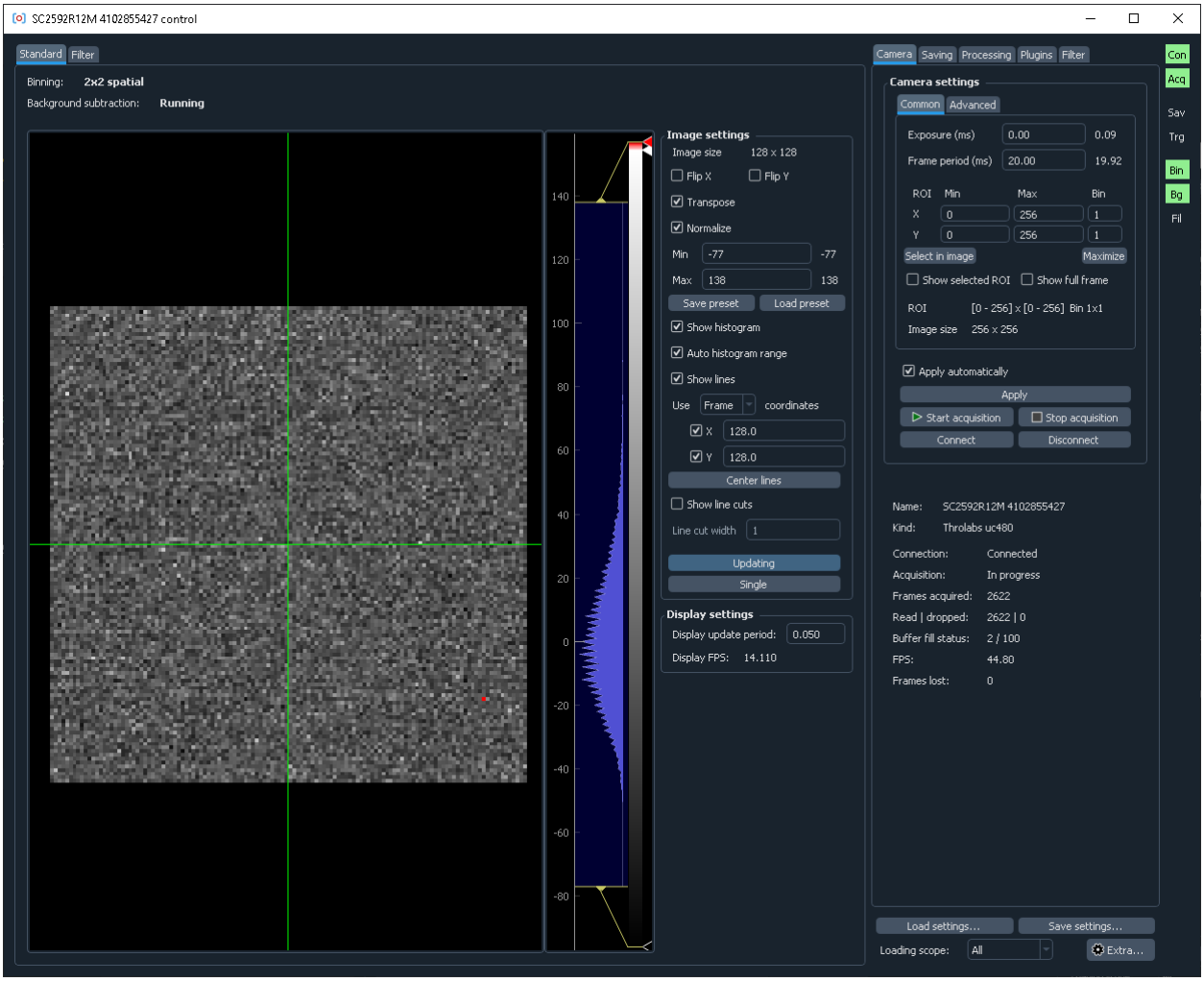


The window is split into two parts. The left half shows the images, possibly with several tabs to show several different kinds of images (e.g., if filters are used). The right half controls data saving, shows camera status (in the middle column), and has additional controls for camera, on-line processing, or additional plugins (rightmost column).

The acquisition is started by pressing `Start acquisition` button. Note that you might need to adjust camera parameters (e.g., specify exposure, ROI, binning, pixel readout rate) to get reasonable image quality and performance.

All of the entered values are automatically saved on exit and restored on the next restart. It is also possible to *save the settings to a file* and load them later, which is helpful for working in several different regimes.

You can find more information either on the *interface page*, or in the built-in *tutorial*.



In case the interface takes too much space and does not fit in the screen, you can enable the compact mode in the *preferences*.

The software uses a dark color theme by default. You can change it in the *preferences*.

1.4 Support and feedback

If you have any issues, suggestions, or feedback, you can either raise an issue on GitHub at https://github.com/SandoghdarLab/pyLabLib-cam-control/issues, or send an e-mail to pylablib@gmail.com.

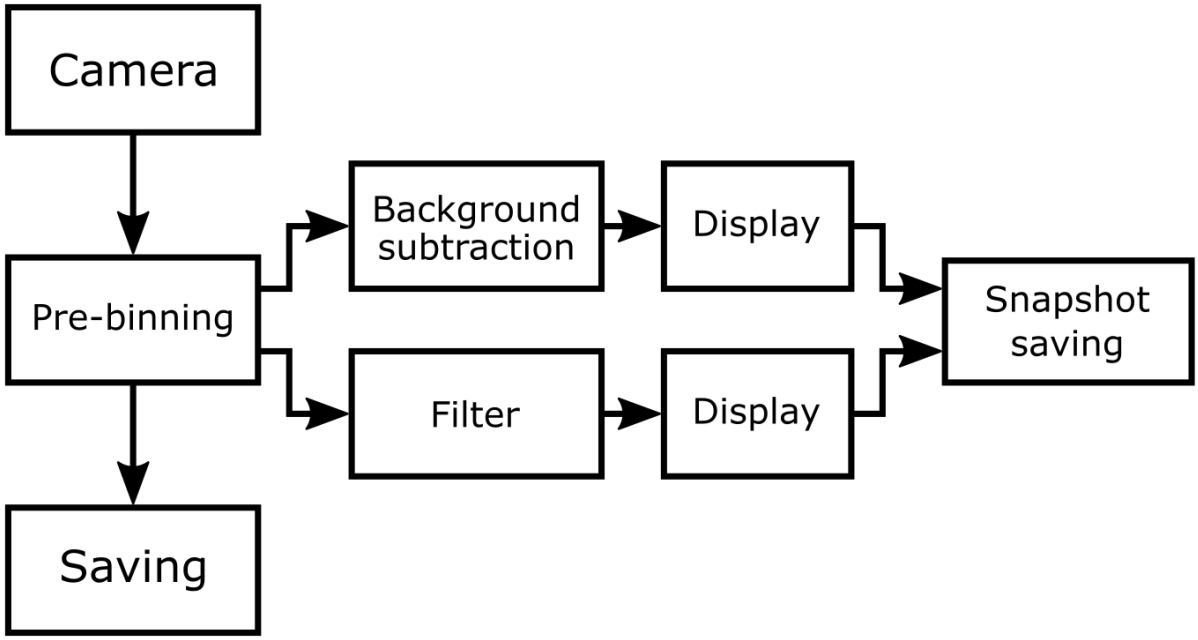
CHAPTER

TWO

BASIC DATA FLOW

Here we describe the basic stages that acquired frames go through, what they do, and how they can be controlled.

2.1 Overview



After camera acquisition, the frames undergo several processing stages:

- First, the raw *camera* frames go through the optional *pre-binning* step, where they are binned spatially and temporally. This step is mostly intended to reduce the load on the further processing or streaming stages which saves processing time and storage space.
- After that, the frames are sent directly for *saving*. All other on-line processing steps (background subtraction, filters, frames slowdown, etc.) only affect the displayed images and snapshot saving, but not the standard data saving.
- At the same time, the pre-binned frames are sent to *processing* stages. One possibility is the built-in background subtraction. The other is custom filters, e.g., Fourier filtering or running average. These are then displayed in corresponding image tabs.

- Finally, these displayed images can be saved directly using the *snapshot saving*. Unlike the standard saving, it saves the processed images exactly like they are displayed, but only one image per file.

These stages above are described in more details below.

2.2 Camera

Generally, cameras continuously generate frames at a constant rate as long as the acquisition is running. In addition to the frames themselves, there is an option (enabled i the `Advanced` tab of the *camera settings*) to query additional frame info such as shape, index, time stamp (both low-resolution based on PC time and, for some cameras, high-resolution camera-baseed one), etc. Additional details about camera functions such as frame buffer, ROI, or external triggering are available in pylablib documentation.

2.3 Pre-binning

This is the first stage after camera acquisition, where the generated frame stream is reduced by binning it. The binning is either temporal, i.e., combining several frames into a single frame, or spatial, combining several pixels in to a single pixel. The available binning modes are skipping (only take the first frame or pixel), summing, taking mean, min, or max.

Pre-binning is generally designed to reduce load on the further processing stages and to shrink the resulting file size. Spatial pre-binning can also partially substitute for camera binning: the effect is mostly the same, although built-in binning typically yields higher signal-to-noise ratio. Temporal sum or mean binning can somewhat increase the camera dynamic range because, unlike increasing exposure, it can avoid saturation.

Note that sum and mean binning can, in principle, result in values not fitting into the standard 16-bit integer used for the frame data, either because of the rounding (for mean) or potential integer overflow (for sum). In this case, there is an option to convert frames to float for further processing and saving. However, this is not always necessary. For example, if the camera has 12-bit pixel depth, you can sum up to a factor of 16 without the fear of overflow, since the maximal image pixel values is 1/16 of the 16-bit integer span.

2.4 Saving

2.4.1 Pre-trigger buffer

This is a fairly useful but rarely implemented feature, which essentially allows you to start saving *before* the saving button was pressed.

The mechanism is similar to the standard operation of any modern digital oscilloscope. The software stores some number `n` of most recently acquired frames in a buffer, called a pre-trigger buffer. Once the save trigger arrives (e.g., when the `Saving` button is pressed), the frames from the buffer are stored first, followed by the frames acquired by the camera. This way, the stored data starts `n` frames before the trigger arrives, which looks as if the saving started in the past.

The amount of the pre-recorded data is controlled by the pre-trigger buffer size `n`. For example, if the camera is acquiring at the rate 5 kFPS and the pre-trigger buffer size is 10,000 frames, the saved file starts 2 seconds before the trigger.

This feature is very useful when recording rare events. First, it allows recording some amount data before the event is seen clearly, which helps studying how it arises. Second, it means that you do not have to have a fast reaction time and press the button as quickly as possible to avoid lost data.

2.4.2 Naming and file arrangement

Saving can result in one or several data files, depending on the additional settings. By default, the main data file is named exactly like in the specified path, the settings file has suffix `_settings`, frame info has suffix `_frameinfo`, and background, correspondingly, `_background`. Furthermore, if snapshot saving is used, suffix `_snapshot` is added automatically.

Alternatively, all of the files can be stored in a separate newly created folder with the specified path. In this case, the main data file i s imply named `frames`, and all auxiliary files lack prefixes.

If the file with the given name already exists and can be overwritten, appended, or the new file will be renamed by adding a number to its name (e.g., `frames000.tiff`). Appending is generally not recommended, since it is not compatible with all file formats and saving modes, and in any case all auxiliary files such as settings and frame info are completely overwritten.

To avoid name conflicts, it is possible to generate new unique file names based on the specified path and the current date and time. This option allows for easy saving of multiple datasets to unique destinations without changing the destination path. By default, the date and time are added as a suffix, but they can also be added as a suffix or as a new folder, based on the *settings file*.

2.4.3 File formats

Currently two basic file formats are supported: raw binary and Tiff/BigTiff.

Raw binary is the simplest way to store and load the data. The frames are directly stored as their binary data, without any headers, metadata, etc. This makes it exceptionally easy to load in code, as long as the data shape (frames dimensions and number) and format (number of bytes per pixel, byte order, etc.) are known. For example, in Python one can simply use `numpy.fromfile` method. On the other hand, it means that the shape and format should be specified elsewhere, so the datafile alone might not be enough to define the content. Note that the settings file (whose usage is highly recommended) describes all the necessary information under `save/frame/dtype` and `save/frame/shape` keys.

Tiff and BigTiff formats are more flexible: they store all of the necessary metadata, allow for frames of different shapes to be stored in the same file, and are widely supported. On the other hand, it is a bit slower to write, and require additional libraries to read in code.

Note: Tiff has a limitation of 2 Gb per single file. If file exceeds this size, the saving is interrupted, and the data might be potentially corrupted (either Tiff file by itself will not load, or the frame indices stored in the settings and frame info files will be out-of-sync with the frames). To avoid that, you can either use BigTiff, which does not have this restriction, or file splitting, as described in the *saving interface*.

In the end, raw binary is more convenient when the data is processed using custom scripts, while Tiff is easier to handle for external software such as ImageJ.

2.4.4 Saving buffer

Many high-speed cameras can generate data faster than it can be saved (about 60 Mb/s for an HDD and about 500 Mb/s for an SSD). In this case, the acquired but not yet saved data is stored in the intermediate saving buffer. This buffer is being simultaneously both filled from the camera and emptied to the drive, which means that its size is only growing as the difference of the two rates. For example, if the camera generates data at 80 Mb/s and it is saved to the drive at 60 Mb/s, the buffer only grows at 20Mb/s. Similarly, if camera generates data at 40 Mb/s, which is lower than the saving rate, then the buffer stays empty, and the streaming can go indefinitely long.

Note: It is important to keep in mind, that the saving is marked as done when all the necessary frames have been placed into the saving buffer, but not necessarily saved. If the frames buffer has some data in it at that point, it will take additional time to save it all to the drive. If another saving is started in the meantime, those unsaved frames will be lost. The filling of the saving buffer can be seen in the *saving status*.

2.4.5 Snapshot saving

In addition to streaming complete data, there is an option to save the data which is currently displayed. This is done through a separate path, which is independent from the main saving routine; hence, it can be done during ongoing data streaming. Note that, like the standard save, the snapshot also stores all of the additional settings data, but not the frame info or the background.

2.5 Background subtraction

Background subtraction can be done in two different flavors. In the "snapshot" subtraction a fixed background frame is acquired and calculated once. In the "running" subtraction a set of `n` immediately preceding frames is used to generate the background, so it is different for different frames. In either case, the background is usually generated by acquiring `n` consecutive frames and calculating their mean, median, or (per-pixel) minimal value. Combining several frames usually leads to smoother and more representative background, thus improving the quality, but taking more resources to compute.

The running background subtraction can usually be fairly well reproduced from the saved data, as long as its length is much longer than the background window. However, the same can not be said about the snapshot background, which could have been acquired under different conditions. To account for that, there is also an option to store the snapshot background when saving the data. This saving can be done in two ways: either only the final background frame, or the complete set of `n` frames which went into its calculation.

2.6 Filters

Filters are pieces of Python code, which process the acquired frames and output new images. There are some *built-in filters*, but they can also be *written by a user*.

CHAPTER

THREE

ADVANCED FEATURES

3.1 Playback slowdown

Fast cameras are typically used to analyze fast events. However, cam-control still has to display data in real time, which means that this fast acquisition potential is lost during on-line analysis. Of course, it still saves all the frames for further examination, but it usually takes some time to load and go through them.

For a quick check, there is an option to temporarily slow data display to the desired frame rate, slowing down the playback. For example, if the camera operates at 10 kFPS and the playback is set to work at 100 FPS, the process is seen 100 times slower.

The way it works is by storing all incoming camera frames into a temporary buffer, and then taking these frames from the buffer at a lower rate. Since the size of this buffer is limited, the slowdown naturally can only proceed for a finite time. It is easy to calculate that, e.g., for the same 10 kFPS camera speed and 100 FPS playback speed the buffer of 1000 frames will take 0.1s of real time and stretch it into 10s of display time.

Note that the slowdowns happens after the pre-binning, but before filters and background subtraction. Hence, it affects all of the displayed frames, but not the saving, which still happens at the regular rate.

This feature controls are on the *Processing tab*.

3.2 Time plot

Sometimes it is useful to look at how the image values evolve in time. Cam-control has basic capabilities for plotting the mean value of the frame or a rectangular ROI within it as a function of time or frame number. It can be set in two slightly different ways: either plot averages of displayed frames vs. time, or averages of all camera frames vs. frame index.

This feature is only intended for a quick on-line data assessment, so there is currently no provided way to save these plots. As an alternative, you can either save the whole move, or use *time map filter* and save the resulting frame.

This feature controls are on the *Processing tab*.

3.3 Saving trigger

Often we would like to automate data acquisition. There are two basic built-in ways to do that in cam-control.

The first is simple timer automation, where a new data set is acquired with a given period. It is useful when monitoring relatively slow processes, when recording data continuously is excessive.

The second is based on the acquired images themselves. Specifically, it is triggered when any pixel in a displayed image goes above a certain threshold value. Since multiple consecutive frames can trigger saving, this method also includes a dead time: a time after triggering during which all triggers are ignored. This way, the resulting datasets can be spaced wider in time, if required. However, even with zero dead time (or zero period for timer trigger) the recording can only start after the previous recording is finished, so that each saved dataset is complete.

The image-based method strongly benefits from two other software features: *pre-trigger buffer* and *filters*. The first one allows to effectively start saving some time before the triggering image, to make sure that the data preceding the event is also recorded. The second one adds a lot of flexibility to the exact triggering conditions. Generally, it is pretty rare that one is really interested in the brightest pixel value. Using filters, you can transform image to make the brightest pixel value more relevant (e.g., use transform to better highlight particles, or use temporal variations to catch the moment when the image starts changing a lot), or even create a "fake" filter output a single-pixel 0 or 1 image, whose sole job is to trigger the acquisition.

Both timed and image trigger also support a couple common features. They both can trigger either standard save for more thorough data acquisition, or snapshot to get a quick assessment. And both can take a limit on the total number of saving events.

This feature controls are on the *Plugins tab*.

3.4 Filters

Filters provide a flexible way to perform on-line image processing. They can be used to quickly assess the data in real time or even to *automate data acquisition*.

They are primarily designed for *expanding by users*. Nevertheless, there are several pre-made filters covering some basic spatial and temporal image transforms:

- **Gaussian blur**: standard image blur, i.e., spatial low-pass filter. The only parameter is the blur size.
- **FFT filter**: Fourier domain filter, which is a generalization of Gaussian filter. It involves both low-pass ("minimal size") and high-pass ("maximal size") filtering, and can be implemented either using a hard cutoff in the Fourier space, or as a Gaussian, which is essentially equivalent to the Gaussian filter above.
- **Moving average**: average several consecutive frames within a sliding window together. It is conceptually similar to *time pre-binning*, but only affects the displayed frames and works within a sliding window. It is also possible to take only every n'th frame (given by `Period` parameter) to cover larger time span without increasing the computational load.
- **Moving accumulator**: a more generic version of moving average. Works very similarly, but can apply several different combination methods in addition to averaging: taking per-pixel median, min, max, or standard deviation (i.e., plot how much each pixel's value fluctuates in time).
- **Moving average subtraction**: combination of the moving average and the time derivative. Averages frames in two consecutive sliding windows and displays their difference. Can be thought of as a combination of a moving average and a sliding *background subtraction*. This approach was used to enhance sensitivity of single protein detection in interferometric scattering microscopy (iSCAT) [Young2018], and it is described in detail in [Dastjerdi2021].
- **Time map**: a 2D map which plots a time evolution of a line cut. The cut can be taken along either direction and possibly averaged over several rows or columns. For convenience, the `Frame` display mode shows the frames

with only the averaged part visible. This filter is useful to examine some time trends in the data in more details than the simple local average plot.

- **Difference matrix**: a map for pairwise frames differences. Shows a map `M[i,j]`, where each element is the RMS difference between `i`'th and `j`'th frames. This is useful for examining the overall image evolution and spot, e.g., periodic disturbances or switching behavior.

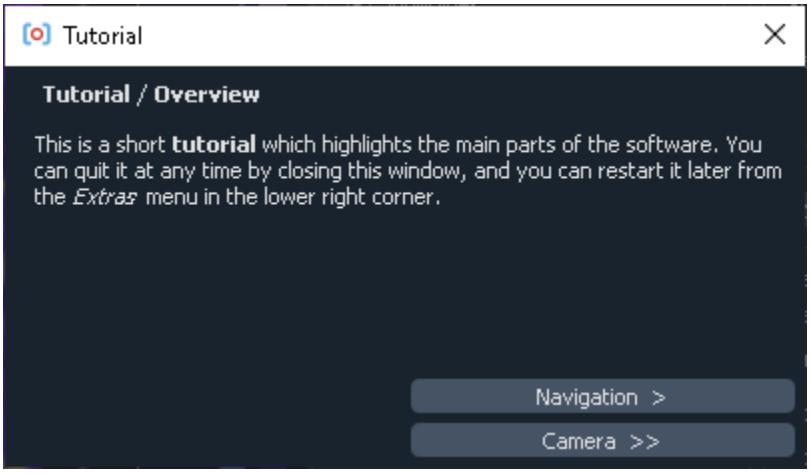
This feature controls are on the *Filter tab*.

CHAPTER

FOUR

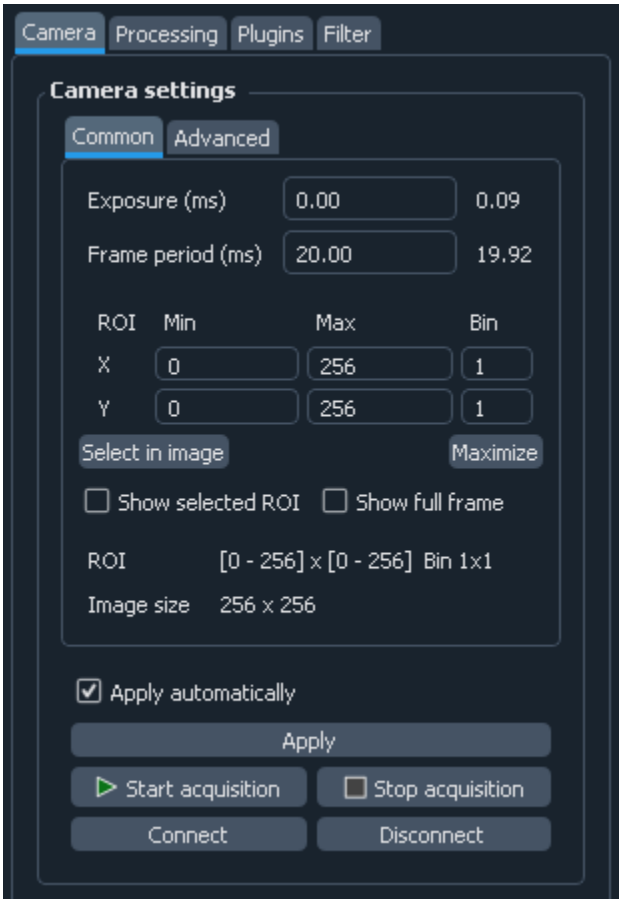
DETAILED INTERFACE DESCRIPTION

4.1 Tutorial



The first time you start the software, you are greeted with the tutorial. It walks you through the interface and the basic concepts. It is recommended that you take it at some point to get familiar with all of the software capabilities. If you want to revisit it, you can find it under the *Extra button*.

4.2 Camera control



The `Camera` tab controls and displays the camera parameters. To reduce the screen space, the settings are split into two categories: `Common` and `Advanced`. The separation is purely cosmetic, and all the settings are treated in exactly the same way.

The settings are applied directly after changing the control. Sometimes (e.g., when using IMAQ frame grabbers at very high frame rates) applying settings might take several seconds, which makes this default behavior annoying. In this cases, one can uncheck `Apply automatically` checkbox, so the setting will apply only on pressing `Apply` button.

Most parameters have an indicator next to the control, which shows the current parameter state. The state might be different from the entered control value, either when the parameter has not been applied, or if the entered value is out of range, and has to be coerced to the nearest valid value. For example while entering exposure value of 0 is allowed, it is never set exactly to zero, but rather to the lowest possible value given all other parameters.

The settings and their exact behavior vary a bit between different cameras. Below are the most common:

- `Exposure`: exposure time (in ms)
- `Frame period`: frame period (in ms). Ultimately determines the camera frame rate (as shown in the status table)
- `ROI`: region of interest and binning of the camera. `X` coordinate is horizontal from the left, `Y` coordinate is vertical from the top. For example, to get the bottom half of the 512x512 sensor, one needs to enter `Xmin=0, Xmax=512, Ymin=256, Ymax=512`. The coordinates are given in raw (un-binned) camera pixels, i.e., full ROI for 2048x2048 sensor with 4x binning corresponds to `Xmax=Ymax=2048`. Depending on the camera, binning can be independent for two axes, the same for two axes, applied to only one axis, or completely disabled. Hence, one

or both of the `Bin` controls can be disabled.

`Select in image` button below tje ROI controls on the left lets you select the ROI in the image display by dragging the mouse to select a rectangular region. Note that it only works in `Standard` display and not, e.g., in `Filter` display. `Maximize` button sets the ROI to the full frame, while preserving the binning.

The two checkboxes, `Show selected ROI` and `Show full frame`, display two rectangles in the main image view which show, correspondingly, the currently entered (but not applied) ROI and the whole camera sensor. `Show full frame` in combination with `Select by image` makes it especially convenient to select ROI relative to the full camera frame.

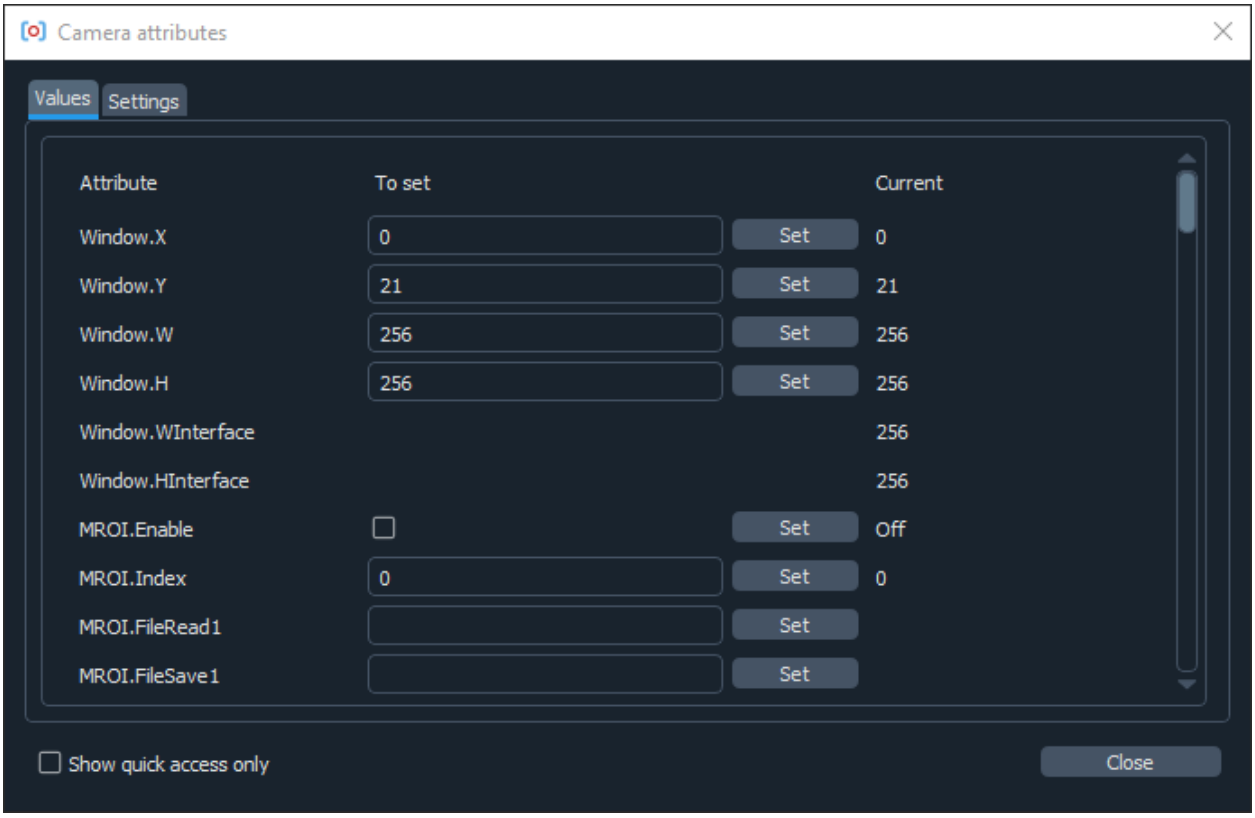
In the bottom of ROI controls are two indicators. The first, `ROI`, shows the actually applied ROI parameters in raw (i.e., un-binned) camera pixels. These can be different from the specified, as cameras often have restrictions on the minimal and maximal size, or the ROI position. The second indicator, `Image size`, shows the resulting image size, taking binning into account.

- `Trigger mode` (on `Advanced` tab): allows to set internal/external frame trigger.

The acquisition process is controlled by `Start Acquisition` and `Stop Acquisition` buttons. At the start of the software, the camera is not acquiring frames.

Finally, `Connect` and `Disconnect` button control the camera connection. By default, the camera is connected on startup. However, if it is temporarily required in some other software, it can be disconnected, and then re-connected again.

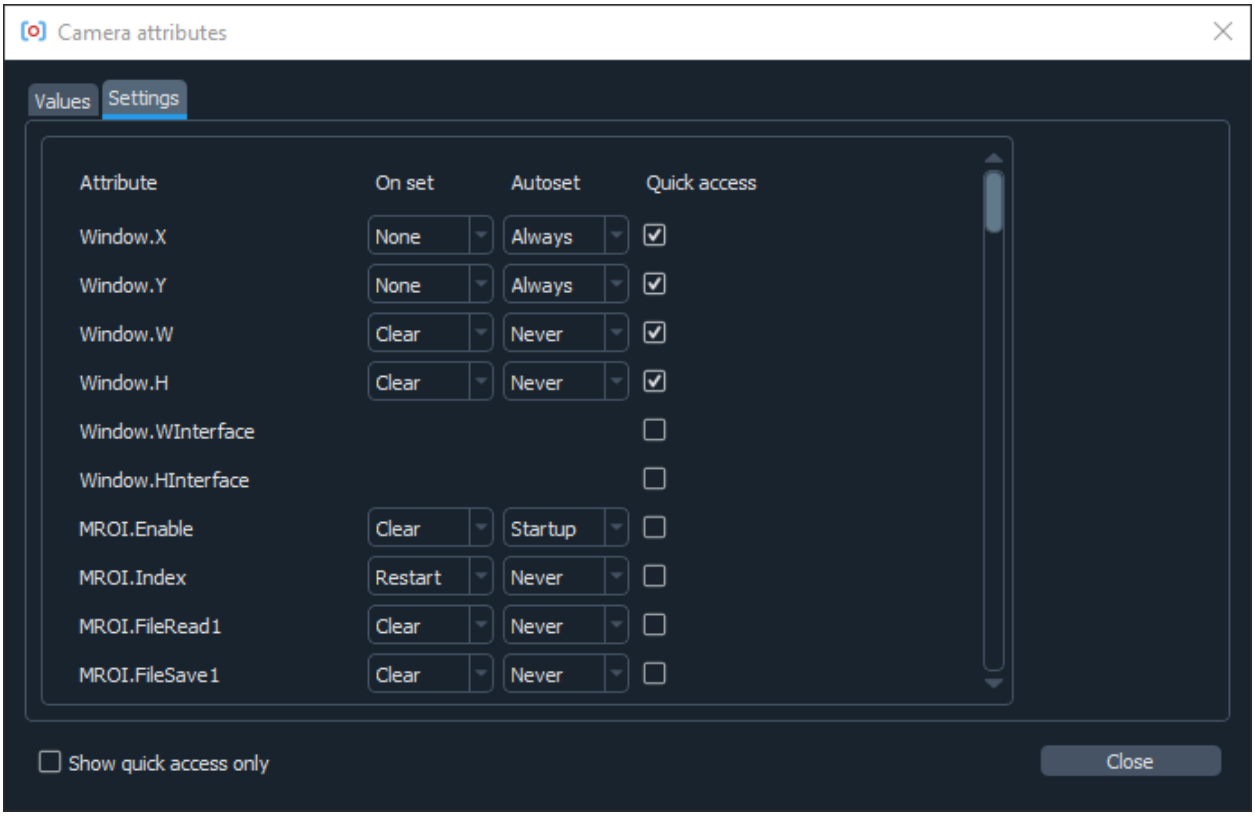
4.3 Camera attributes



Some cameras such as Andor SDK3, Hamamatsu, Photometrics and Princeton Instruments support camera attributes in their API. In principle, they provide almost complete control over camera's hardware, but their selection, naming

and meaning varies a lot between different manufacturers and event different camera models. Therefore, the software provides a generic interface to access these attributes, but does not guard about potential errors arising due to improper settings, which can include the software shutdown if, e.g., the settings interfere with the readout parameters. Hence, one needs to be somewhat careful when using these, and expect potential issues.

Th attributes window is accessed via `Show attributes` button in the advanced camera settings. It has two tabs. The first one gives access to the attribute values. In it the first columns (`Attribute`) shows the attribute name, the next one (`To set`) lets you select a new attribute value (the control is absent if the attribute is read-only), which is followed by a `Set` button, which sets the attribute after entry, and the indicator showing the current attribute value. Finally, if setting the attribute resulted in an error, it will be indicated in the last column, with the explanation appearing when the mouse is hovered over.



The second tab defines some of the attributes behavior. The first control, `On set`, describes how the acquisition is affected during the attribute settings. It can be `Clear`, which means that it is temporarily stopped and the frame buffer is cleared (the safest but also the slowest option), `Restart`, in which the acquisition is stopped but the buffer remains allocated (this is somewhat faster but, among other things, requires the frame size to remain constant), and `None`, meaning that the camera acquisition is not affected (the fastest, but not supported by all cameras).

The second control, `Autoset`, defines when the attribute is set. It can be `None` (only set when the `Set` button is pressed), `Startup` (set on the software setup to, e.g., set up some basic parameters), or `Always` (set on startup and any time the entered value is changed).

Finally, `Quick access` cna select whether the attribute should show up in the "quick access" mode. Since there are tens or even hundreds of attributes, and only a handful of them are usually relevant, there's a "quick access" mode enabled by a checkbox in the lower left corner, which switches to showing only the marked attributes. This provides more convenience in dealing with the same recurring set of attributes.

4.4 Camera status

The top half of the status table deals with the camera status:

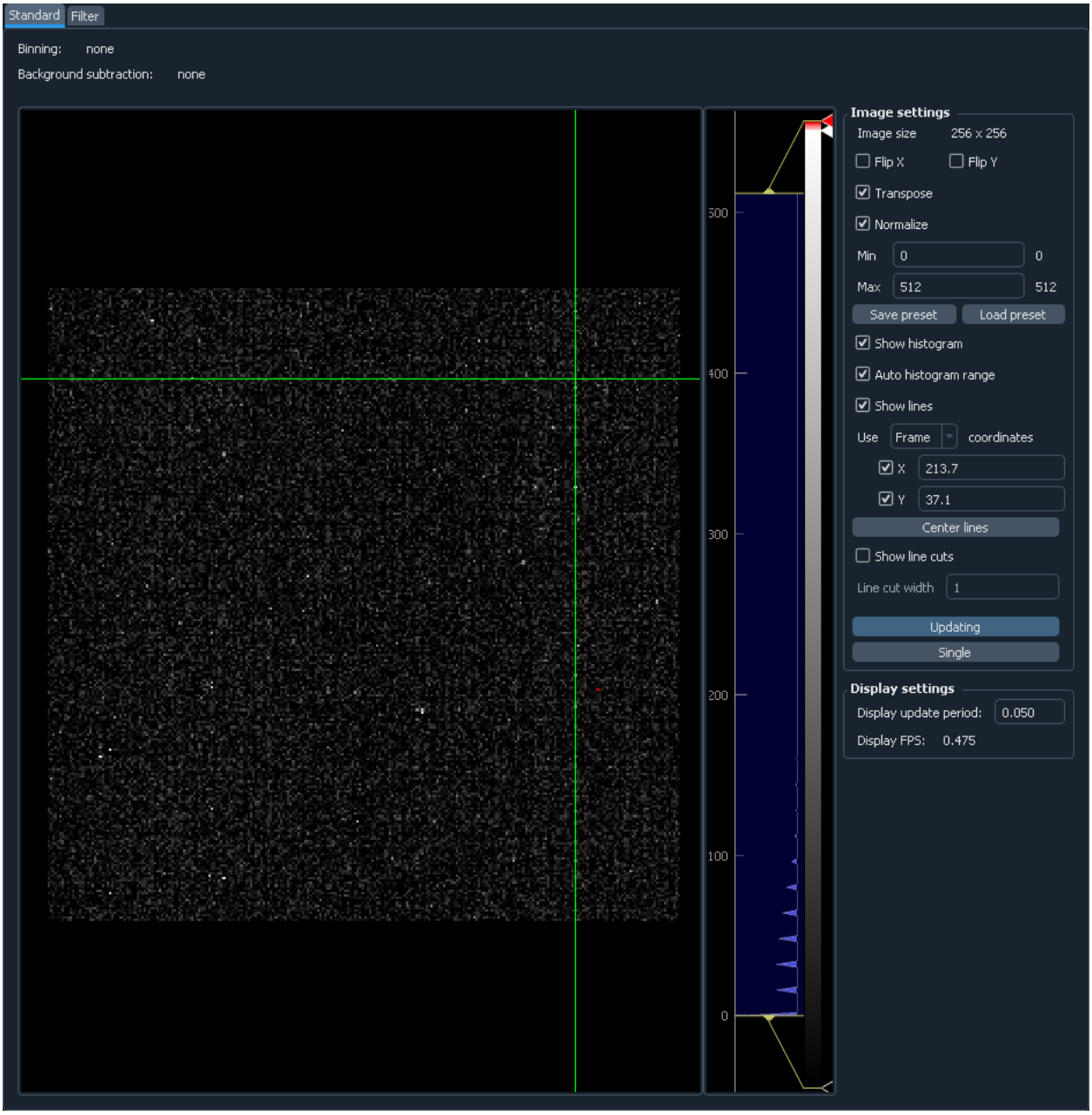
- `Name`: camera name, as defined in the settings file
- `Kind`: camera kind; usually, the camera interface kind, e.g., `Andor iXON` or `Thorlabs uc480`
- `Connection`: shows whether camera is connected or not. Normally the camera should always be connected. It is only disconnected if it has been disconnected manually (by pressing `Disconnect` button on the camera control), or if the software couldn't connect to the camera on the startup. The latter usually means that the camera is turned off, disconnected, or used by a different program
- `Acquisition`: acquisition status
- `Frames acquired`: number of frames acquired by the camera in the current acquisition session, i.e., since the last press of the `Start Acquisition` button or the last change of parameters
- `Read / dropped`: number of frames which have been successfully read from the camera and, correspondingly, lost in transmission. Ideally the dropped frames counter is always zero, which means that the number of read and acquired frames is the same.
- `Buffer fill status`: status of the camera frame buffer; shows number of unread frames in the buffer and the total number of frames in the buffer
- `FPS`: calculated camera frame generation rate

Frames buffer, and the difference between acquired and read frames require some explanation.

Typically, after the camera acquires a frame and sends it to the PC, the camera driver places it into RAM almost immediately. However, the software working with the frames can't guarantee to react to each frame individually and quickly enough before the next frame arrive. To deal with that, the camera driver allocates a large area of RAM (called a buffer), where it places all the acquired frames in sequence. Afterwards, when the user software checks for new frames, it can read multiple frames acquired since the last check, and deal with all of them in one go. In other words, it can process acquired frames in "bursts", keeping the required throughput (total number of processed frames per second), but reducing the required reaction time and frequency of checks for the new frames.

However, this only works if the buffer doesn't overflow between the checks, i.e., if the number of frames acquired since the last check doesn't exceed the size of the buffer. If that happens, some of the acquired frames will be lost, which causes the difference between acquired frames (frames read by the camera and placed into the buffer), and the read frames (frames extracted from the buffer and sent further down the road). Hence, at high frame rates and high data rates it is important to monitor the buffer fill status and make sure that the buffer is not too full, and especially that the buffer fill level is not steadily rising.

4.5 Image display



The image control is based on pyqtgraph `ImageView` control, which is used for displaying the image and the intensity histogram. In addition, there are some controls to change the image appearance and to enable additional features:

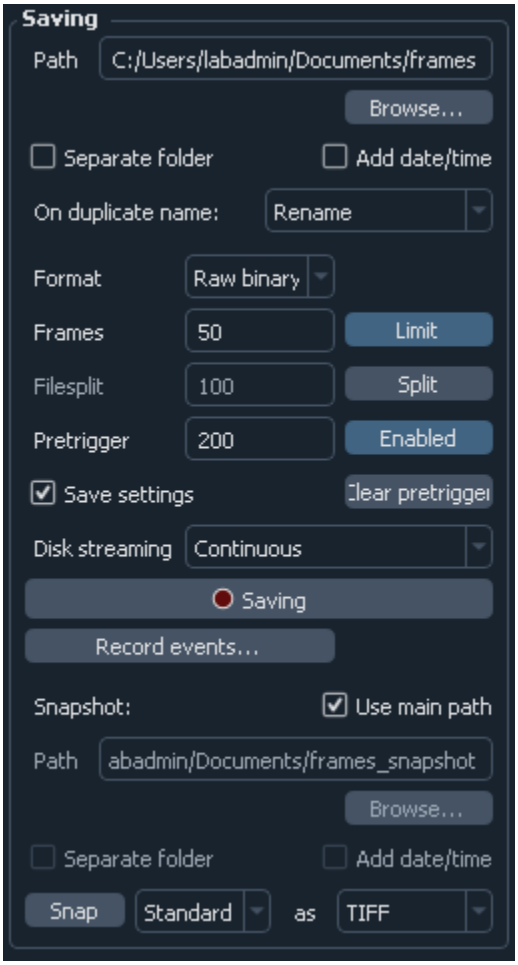
- `Binning`, `Background subtraction`, `Filter` (above the image): quick overview of all processing steps for the displayed frame
- `Image size`: displays image size in pixels
- `Flip X`, `Flip Y`, `Transpose`: correspondingly, flips the image along the given axis, or transposes it (flips along the diagonal); note that these settings only affect display, and do not change the way images are saved
- `Normalize`: controls whether the image levels are automatically normalized to cover the full image range, or if

they stay fixed

- `Min`, `Max`: minimal and maximal intensity levels, if `Normalize` is off
- `Save preset`, `Load preset`: it is possible to store a single "preset" combination of intensity levels and recall them later; can be used to, e.g., temporarily switch to the `Normalize` mode to assess the whole image, but then quickly switch back to the previously used levels
- `Show histogram`: controls whether the image value histogram on the right is shown; turning it off gives more space to the image and somewhat improves the update rate
- `Auto histogram range`: controls whether the histogram plot is rescaled with every new frame; this is different from the `Normalize` option, which control whether the image level range (shown with a transparent box in the histogram) gets similarly rescaled
- `Show lines`: controls whether the green cross is shown in the plot; it can be used to mark or extract positions of features in the image
- `Use <name> coordinates`: controls the coordinate system used for the lines. By default the included systems are `Display` (display image coordinates after applying flipping and rotations), `Image` (image coordinates, before flipping and rotations), and `Frame` (camera frame coordinates, which account for ROI and binning settings; only available in the `Standard` image display).
- `X`, `Y`: enable or disable individual lines and control their positions in the specified coordinate system. The lines can also be moved in the plot, or centered to a given position with a double-click.
- `Center lines`: move the cross to the center of the images
- `Show line cuts`: when activated, shows a small additional plot with line cuts along the displayed lines
- `Line cut width`: if more than 1, it specifies a band thickness to average for a single line cut; this might reduce noisiness of the cuts
- `Updating`: controls whether the image view updates on the new frames, or simply shows the last frame; can be used to improve performance, or to closer inspect a single image
- `Single`: when pressed, grabs a single image and stops updating
- `Display update period`: maximal image update period. Can be increased if the software is to laggy, e.g., if large frames or large data rates are used.
- `Display FPS`: actual display update frame rate.

The colored gradient bar in the intensity histogram shows the current color scheme and allows to change it. It can be done either by right-clicking on it and selecting one of the presets, or manually adding, dragging, and changing color of the markers.

4.6 Saving control



Here the *saving* parameters, such as path, format, and number of frames to save, are controlled:

- `Path`: path for saving the frames. If the containing folder does not exist, it is created automatically; if the extension is not specified, it is added automatically. Note that if `Add date/time` is activated, the actual path will be somewhat different.
- `Separate folder`: if activated, then the supplied path is treated as a folder, and all of the data is stored inside under standard names (`frames.bin` or `frames.tiff` for main frames data, `settings.dat` for settings, etc.) This option allows for better data organizing when each dataset has multiple files (e.g., main data, settings, frame info, background, several split files).
- `Add date/time`: if activated, create a unique name by appending current date and time to the specified path. By default, the date and time are added as a suffix, but this behavior can be changed in the *preferences*.
- `On duplicate name`: determines what happens if the files with the specified name already exists; can be `Rename` (add a numeric suffix to make a new unique name), `Overwrite` (overwrite the existing data), or `Append` (append the existing data)
- `Format`: saving format; so far, only raw binary, tiff, and big tiff (BTF) are supported
- `Frames limit`: if activated, only save the given number of frames; otherwise, keep streaming data until saving is manually stopped

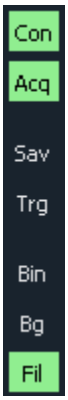
- `Filesplit`: if activated, saved frames are split into separate files of the specified size instead of forming a single large file; this is useful when continuously acquiring very large amounts of data to avoid creating huge files
- `Pretrigger`: set up the *pretrigger* buffer size
- `Clear pretrigger`: clear the accumulated pretrigger buffer
- `Save settings`: if checked, then in addition to the frame saves a text file containing all of the related information: camera settings, GUI state, frame counters, frame shape and data format, etc. Highly recommended to use.
- `Disk streaming`: selects the way of data streaming to the disk. `Continuous` is as described in the *saving buffer* explanation, with frames being constantly streamed to the disk as quickly as possible while the overhead is stored in the buffer. Alternatively, `Single-shot` mode does not write data during acquisition, but only starts streaming to the disk after the necessary number of frames has been accumulated (or the saving has been stopped manually). Unlike the `Continuous` mode, it can not work indefinitely, since its stores in RAM all the data to be saved. However, for very high-performance cameras working at >1Gb/s (e.g., Photometrix Kinetix) this mode is more reliable and has lower chances of dropping frames during acquisition.
- `Saving`: the main button which initiates and stops data streaming; while streaming, changing of any other saving parameters is not allowed
- `Record events...`: opens a small window which lets one record various events during data acquisition. The events are tagged by the global OS timestamp, time since the recording start, and the frame number. The event file is automatically created when the first message is added.
- `Snapshot`: *snapshot* saving parameters
- `Use main path`: if checked, snapshot image path will be the same as the main image path, just with `_snapshot` appended to the end; all of the modifying parameters (`Separate folder` and `Add date/time`) are also the same
- `Path`, `Separate folder`, `Add date/time`: same meaning as above, but applied to the snapshot saving; only active if `Use main path` is not checked.
- `Snap`: pressing it saves a single image from the specified source (usually either `Standard` or `Filter`) in the specified image format

4.7 Saving status

The bottom half of the status table deals with the saving status:

- `Saving`: saving status; can be `Saving in progress` during the saving process, `Finishing saving` when finishing writing the data to the hard drive, or `Saving done` when done.
- `Frames received`: number of frames received for saving during the current saving session
- `Frames scheduled`: number of frames which have been scheduled for saving to the drive
- `Frames saved`: number of frames stored to the drive
- `Frames missed`: number of frames which were missed in saving; this includes both frames that were received but not saved (e.g., due to save buffer overfill) and frames missed on camera readout
- `Status line`: some cameras provide a status line within their frames (currently only PhotonFocus is supported). This status line allows one to do the last-minute check of the frames consistency, whose results are shown here.
- `Saving buffer`: fill status of the *save buffer* and its maximal size in Mb. This maximal size can be changed in *preferences*.
- `Pretrigger frames`: fill status of the *pre-trigger buffer*
- `Pretrigger RAM`: same as `Pretrigger frames`, but expressed in memory size; useful to keep an eye on it in case the requested pre-trigger buffer size gets too large
- `Pretrigger skipped`: number of skipped frames in the pre-trigger buffer, which arose during the camera readout

4.8 Activity overview



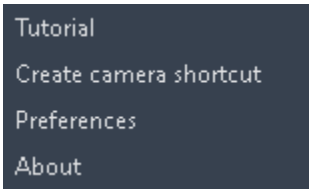
In the upper right corner you can find indicators for the basic software activities: camera connection and acquisition, saving, background subtraction, filters, etc. These give a fast overview and help to, e.g., notices that some process is stopped (e.g., saving is done), or if it uses resources unnecessarily (e.g., running filters).

4.9 Settings saving and extras

The small box in the lower right corner allows to save the application settings to a file and subsequently load them. This lets you quickly switch between several working modes. `Loading scope` selects the loaded settings scope: only camera settings, everything except for the camera, or all settings.

If you want to only load some of the settings, you can manually edit saved settings files. It is a human-readable table, and the parameter names are relatively straightforward to decipher. Note that you can also load settings from the `*_settings.dat` file accompanying the saved data, as long as it was obtained using the same version of the software. This may be useful to make sure that you save the data with exactly the same parameters as before.

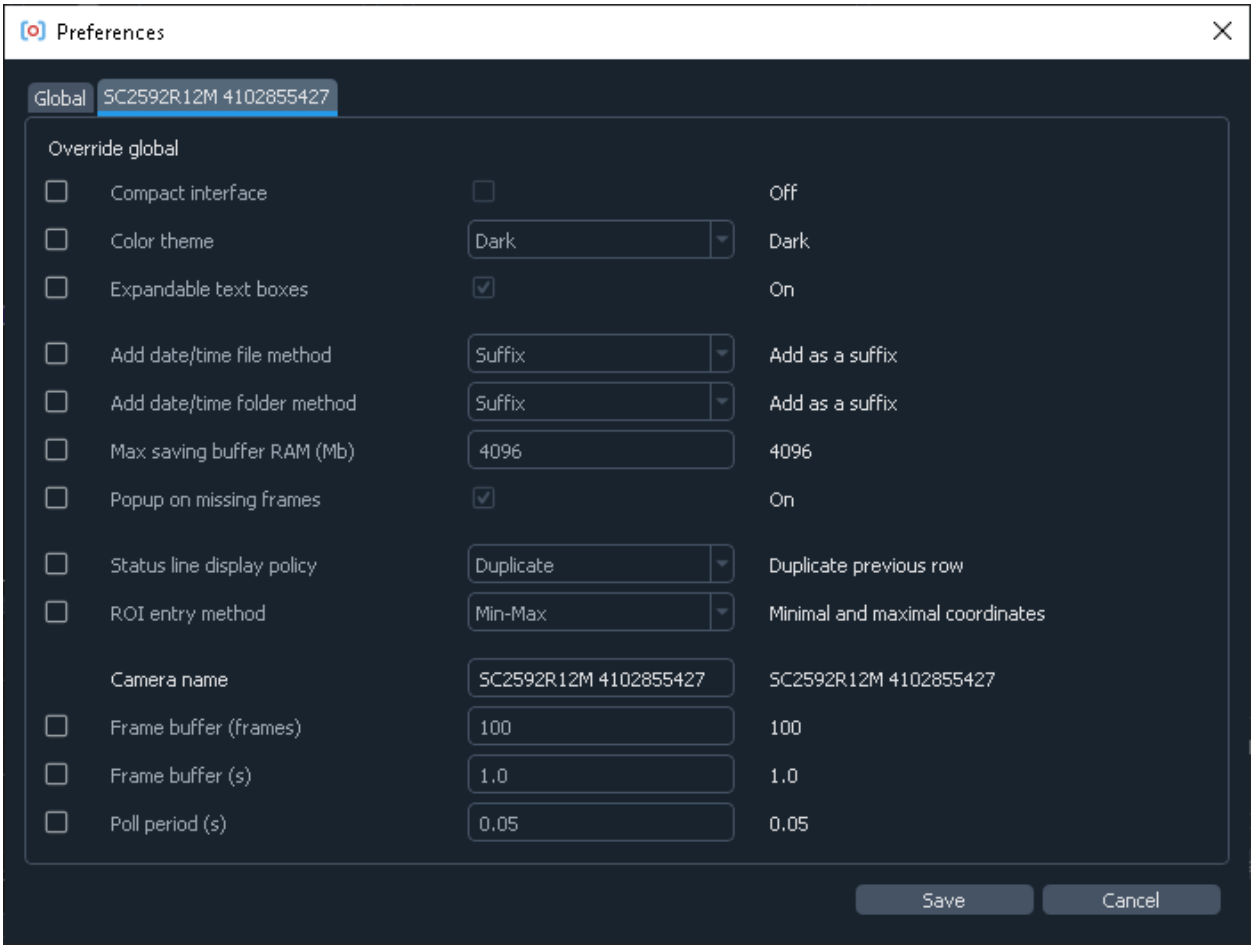
4.10 Extras



The `Extra...` button in the *footer* contains additional infrequently used features:

- `Tutorial`: interface and operation *tutorial*, which automatically shows up during the first run of the software
- `Create camera shortcut`: if there are multiple cameras, this button allows to create a shortcut which connects to a particular camera. This skips the camera selection window on the application start and immediately runs the specific camera.
- `Preferences`: opens the *settings and preferences editor*.
- `About`: opens the `About` window with the version information and useful links.

4.11 Settings and preferences



The preferences window can be opened using the *Extras* button. Here you can edit the general software settings. These cover all the same items as the *settings file*, but provides a user-friendly interface for editing these settings. This windows has several tabs. The first tab controls general settings, which affect all cameras by default. The other tabs (one per cameras) allow you override these settings for specific cameras (e.g., choose different color schemes for different cameras), as well as control some camera-specific settings. Here are the generic settings:

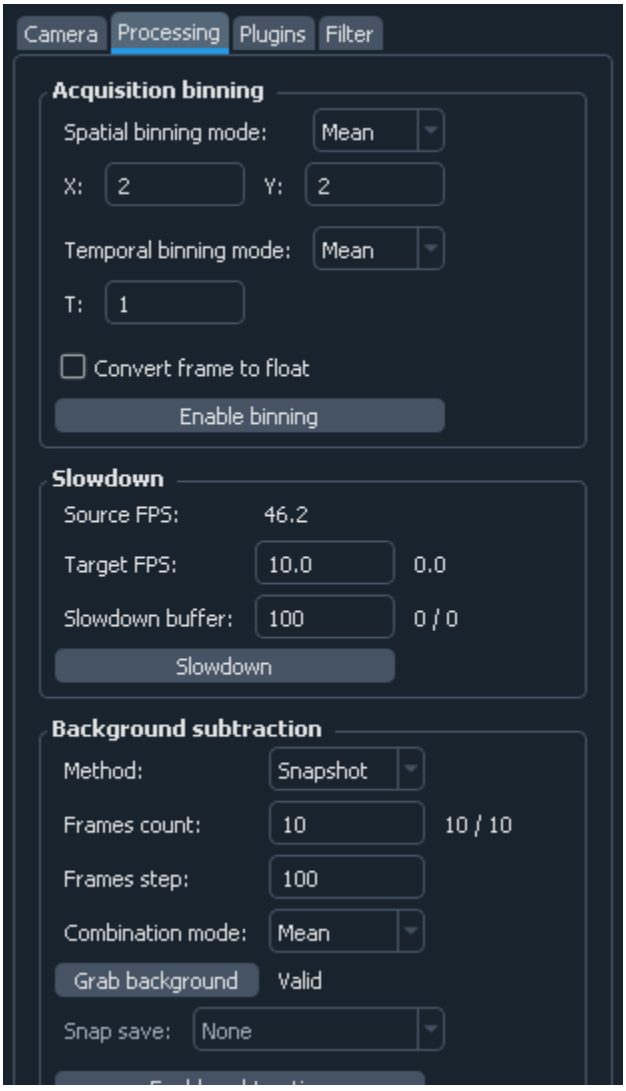
- `Compact interface`: switche between the standard four-panel and the more compact three-panel layouts.
- `Color theme`: select different interface and color therems (based of qdarkstyle).
- `Expandable text boxes`: enable or disable expandable text boxes for paths and event logs.
- `Add date/time file method`: method to generate file names when `Add date/time` is selected but `Create separate folder` is not. Can be `Prefix` (add date as a prefix, e.g., `20210315_120530_video.bin`), `Suffix` (add date as a suffix, e.g., `video_20210315_120530.bin`), or `Folder` (create folder with the datetime as name, e.g., `20210315_120530/video.bin`).
- `Add date/time folder method`: same but when both `Add date/time` and `Create separate folder` are selected.
- `Max saving buffer RAM (Mb)`: maximal size of the saving buffer in megabytes. Makes sense to increase if large movies are saved to slow drive, or if large pre-trigger buffer is used (the size of the saving queue must be larger than the pre-trigger buffer). Makes sense to decrease if the PC has small amount of RAM.

- `Popup on missing frames`: whether to show a pop-up message in the end of saving if the saved data contains missing frames.
- `Status line display policy`: method to deal with a status line (on PhotonFocus or PCO edge cameras) when displaying frames. Only affects the displayed image.
- `ROI entry method`: ROI entry method in camera control. Can be `Min-Max` (enter minimal and maximal coordinates), or `Min-Size` (enter minimal coordinates and size).

In addition, there are several camera-specific parameters:

- `Camera name`: the name associated with the camera, which is displayed in the window title, camera status, or in the dropdown menu during camera selection. By default, autogenerated based on the camera model and serial number.
- `Frame buffer (frames)`: minimal camera frame buffer size defined in terms of number of frames.
- `Frame buffer (s)`: minimal camera frame buffer size defined in terms of acquisition time (in seconds). For example, the size of 1 second would be result in 100 frame for 100 FPS frame rate and 1000 frames for 1 kFPS frame rate.
- `Poll period (s)`: the period at which camera is polled for new frames. Lower period increases the image update frame rate, but might decrease the overall performance.

4.12 Processing controls

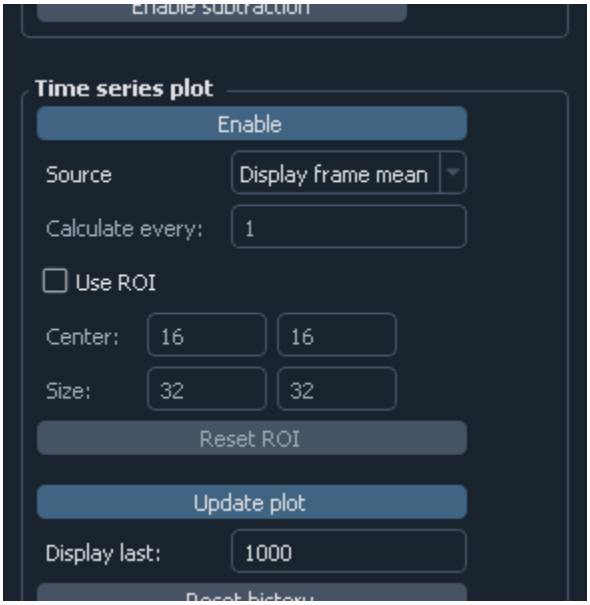


The top half of the `Processing` tab controls *pre-binning*, *slowdown*, and *background subtraction*:

- `Acquisition binning`: controls pre-binning
  - `Spatial binning mode`: determines the mode of the spatial (i.e., within-frame) binning, which reduces the frame size
  - `X` and `Y`: binning factor along the two directions
  - `Temporal binning mode`: determines the mode of the temporal binning, which reduces the frame rate
  - `T`: temporal binning factor
  - `Convert frame to float`: if checked, the frames fed to later stages (including saving) are converted to float instead of staying as integer; useful when `Mean` or `Sum` binning modes are used
  - `Enable binning`: enables or disables the binning
- `Slowdown`: controls the display slowdown

  - `Source FPS`: displays the current frame rate; normally it is equal to the camera FPS divided by the temporal binning factor
  - `Target FPS`: reduced frame rate; the slowdown factor is then roughly equal to the ratio of the source to the target FPS
  - `Slowdown buffer`: size and current status of the slowdown buffer; the status shows the number of already displayed frames from the buffer and the total number of frames acquired so far, while the edit box control the maximal size of the buffer
  - `Slowdown`: enables or disables the slowdown
- `Background subtraction`: controls the background subtraction options
  - `Method`: subtraction method, which can be `Snapshot` (a single fixed frame) or `Running` (dynamically generated from some number of previous frames)
  - `Frames count`: number of frames to combine for the background
  - `Combination mode`: method of combining the frames; note that `Median` works significantly slower than all other methods, and should be avoided for large frame counts (typically, above 100-1000 frames) in the `Running` mode
  - `Grab background`: if `Snapshot` method is used, pressing it initializes the new snapshot background acquisition; while it is in progress, `Frames count` status shows the number of frames acquired so far
  - `Snap save`: determines whether the snapshot background is saved together with the main data; only active when `Snapshot` method is used and the subtraction is active
  - `Enable subtraction`: enable or disable the background subtraction

4.13 Time plot

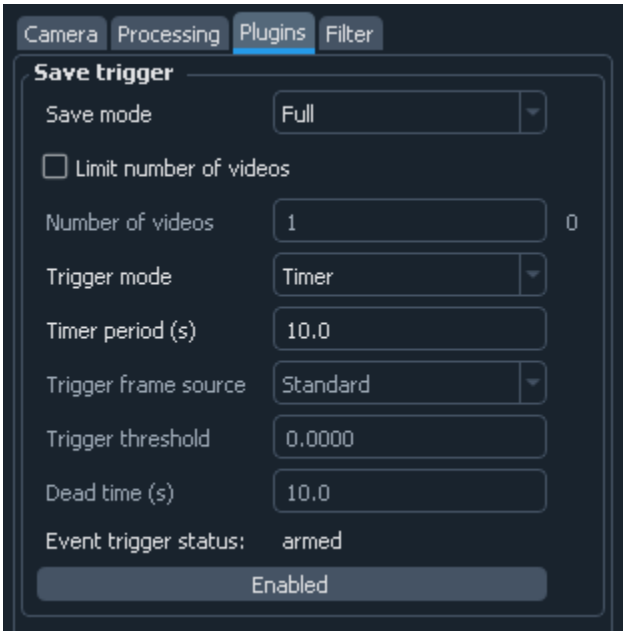


This part controls the *time series plotting*:

- `Enable`: enable or disable the time series plot
- `Source`: plot source; can be either `Display frames` or `Raw frames`

- `Calculate every`: if raw frames are used, the averaging might be computationally expensive for high frame rates; this parameter allows to average only some frames with the given periodicity
- `Use ROI`: enable or disable averaging in a given region of interest (ROI); if disabled, average the whole frame
- `Center`, `Size`: controls the averaging ROI
- `Reset ROI`: reset ROI to the full frame
- `Update plot`: enable or disable plot update
- `Display last`: number of points to display
- `Reset history`: reset the displayed points

4.14 Saving trigger

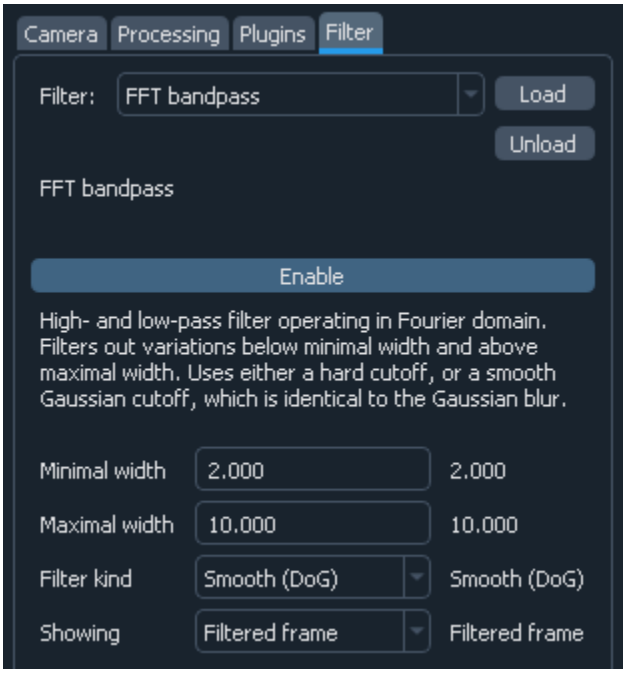


The top part of the `Plugins` tab controls the *saving trigger*:

- `Save mode`: the kind of saving that happens on the trigger; can be `Full` (standard saving, equivalent to pressing `Saving` button) or `Snap` (snapshot saving, equivalent to pressing `Snap` button)
- `Limit number of videos`: if enabled, limits the total number of saved videos
- `Number of videos`: maximal number of saved videos; the indicator shows the number saved so far
- `Trigger mode`: the source of the trigger; can be `Timer` for periodic timed acquisition or `Frame` for a frame-triggered acquisition
- `Trigger frame source`: the source of the triggering frame, either `Standard` for the standard processing pipeline (including background subtraction) or `Filter` for the filter frame
- `Time period (s)`: for timer acquisition, the trigger period
- `Dead time (s)`: for frame trigger, the amount of dead time, i.e., the time after the trigger when the subsequent triggers are ignored. If the save mode is `Full`, it is recommended that the period and the dead time are longer than the length of the single movie

- `Trigger threshold`: frame trigger threshold; the trigger is activated when any image pixel is above this threshold
- `Event trigger status`: frame trigger status, either `armed` (waiting for trigger), `triggered` (triggered recently), or `dead` (dead time period)

4.15 Filter



The `Filter` selects the *filter* and controls its parameters:

- `Filter`: selected filter to load; currently loaded and active filter is shown above the `Enable` button
- `Load`: load the selected filter, or reload if it is already loaded; reloading can be used to, e.g., clear the accumulated frames in the buffer
- `Unload`: unload the filter
- `Enable`: enable or disable the filter; note that while it stops frames from being fed to the filter, it preserves all of the accumulated data

Below this button is the filter description and the list of filter parameters and indicators. Both depend on the exact filter.

CHAPTER

FIVE

EXPANDING AND MODIFYING

5.1 Accessing and modifying the distribution

Cam-control runs in an isolated Python interpreter (located in `python` subfolder within the main folder), which has only basic dependencies installed. Sometimes you might want to modify the installed packages, for example, to add the ones required by your custom code (such as a *custom filter*), or to replace or update some of them. To do that, you can run `local-python.bat` located in the `python` folder, which launches a command line interface referenced to the local Python interpreter. From there you can simply use `python -m pip` to modify the distribution, e.g., run `python -m pip install -U <pkg>` to install or update any desired package, or `python -m pip uninstall <pkg>` to uninstall a currently present package.

> **Warning:** Due to some specifics of path handling in Python scripts on Windows, simply running `pip` will not work. You should always use `python -m pip` instead.

As an alternative for your own code (such as filters), you can request additional packages to be installed during the execution. This can be achieved by adding the following lines to your code file:

```
from pylablib.core.utils import module as module_utils
module_utils.install_if_older("some-package")
```

where `"some-package"` is the name of the package to install (e.g., `"scikit-image"`). This code checks if the package is already installed, and runs `pip install` if it is missing. Note that this code should be included before the required package is first imported.

5.1.1 Running from source

It is also possible to run cam-control in your own Python environment. All of the required code is contained in `cam-control` folder and can be obtained either on GitHub or directly from the folder. To run it, you also need to install the necessary dependencies: NumPy, SciPy, pandas, Numba, RPyC, PyQt5 (or PySide2 with shiboken2), pyqtgraph, and imageio. All of the dependencies are included in `requirements.txt` file inside the `cam-control` folder (it can also be extracted by running `python -m pip freeze` in the local python command line). In addition, the GitHub-hosted version requires pylablib v1.4.1 (not included in `requirements.txt`).

5.2 Custom filters

Filters give a way to add custom on-line image processing. They can be used to quickly assess the data in real time or to *automate data acquisition*.

The filter is defined as a subclass of `IFrameFilter` class with methods to specify its parameters (displayed in cam-control), receive frames from the camera and generate output.

5.2.1 Single-frame filter

In the simplest scenario, filters simply take a single frame and output the transformed image. In this case, the filter can inherit from a more specific `ISingleFrameFilter` class and only define a couple of new methods. For example, this is the built-in Gaussian blur filter:

```
### The class inherits not from the most generic frame class,
### but from ISingleFrameFilter, meaning that it transforms a single frame
class GaussianBlurFilter(ISingleFrameFilter):
    """
    Filter that applies Gaussian blur to a frame.
    """
    _class_name = "blur"  # unique filter identifier; should obey usual variable naming␣
↪rules
    _class_caption = "Gaussian blur"  # caption which shows up in the filter selection␣
↪menu
    _class_description = "Standard convolution Gaussian blur filter"  # detailed␣
↪description

    ### This method sets up the filter upon loading
    ### Usually it defines GUI parameters and internal variables
    def setup(self):
        super().setup()  # call setup from the parent class
        # set up the width parameter
        self.add_parameter("width", label = "Width", limit = (0,None), default = 2)

    ### This methods processes a frame (2D numpy array) and returns the result
    def process_frame(self, frame):
        # self.p is used to access previously defined parameters;
        # frame is converted into float, since it can also be an integer array
        return scipy.ndimage.gaussian_filter(frame.astype("float"), self.p["width"])
```

The main method for a single-frame filter is `process_frame`, which takes a single frame as a 2D array (integer or float) and returns either a processed frame as a 2D array, or `None`, meaning that a new frame is not available. The other important method is `setup`, which is used to initialize variables and define the filter parameters. Finally, each filter class should define `_class_name`, `_class_caption` and, if appropriate, `_class_description` strings.

5.2.2 Multi-frame filter

In a more complicated case, a filter takes several most recent frames and combines them together to get a result. This can be handled by inheriting from `IMultiFrameFilter` class. For example, here is the built-in moving average filter, which simply calculates the averages of the last `n` frames:

```
### This class still inherits from a helper IMultiFrameFilter class,
### which keeps a buffer of the last several frames
class MovingAverageFilter(IMultiFrameFilter):
    """
    Filter that generates moving average (averages last ``self.p["length"]`` received␣
↪frames).
    """
    _class_name = "moving_avg"
    _class_caption = "Moving average"
    _class_description = ("Averages a given number of consecutive frames into a single␣
↪frame. "
        "Frames are averaged within a sliding window.")

    def setup(self):
        # set up the buffer filter; process_incomplete = True means
        #   that it will work even with a partially filled buffer
        super().setup(process_incomplete = True)
        # add the parameters
        self.add_parameter("length", label = "Number of frames", kind = "int",
            limit = (1,None), default = 1)
        self.add_parameter("period", label = "Frame step", kind = "int",
            limit = (1,None), default = 1)

    ### This methods is called whenever a parameter is changed in GUI
    ### Normally it simply updates self.p dictionary,
    ### but in this case it also changes the buffer parameters if necessary
    def set_parameter(self, name, value):
        super().set_parameter(name, value)
        # update the buffer parameters
        buffer_size = value if name == "length" else None
        buffer_step = value if name == "period" else None
        self.reshape_buffer(buffer_size, buffer_step)

    ### This method is called when a new frame is requested
    ### The argument is the buffer, which is a list of 2D numpy arrays
    def process_buffer(self, buffer):
        # if buffer is empty, return None (no new frame to show)
        if not buffer:
            return None
        return np.mean(buffer, axis = 0)
```

The first difference from the previous example is the different calculation method, which is now called `process_buffer`, and which takes a list of 2D arrays instead of a singe array. The second is the redefined `set_parameter` method. This method is called every time a user changes a parameter value in the GUI. By default, it simply updates `self.p` attribute, which can be used when calculating the frame, like in the Gaussian filter example. However, here it also updates the buffer parameters.

5.2.3 Filter storage

To appear in the cam-control, the file defining one or more custom filter classes should simply be added to the `plugins/filter` folder inside the main `cam-control` directory. For further examples, you can examine files already in that folder: `builtin.py` for *built-in filters*, `examples.py` for several example classes, and `template.py` for a template file containing a single filter class.

5.2.4 Debugging

Debugging a filter class by running it from the camera control might be cumbersome. Instead, it might be more convenient to test it on generated or pre-loaded frames. Here is a short snippet for doing that:

```
### Simple filter testing script
# Assume that it is a separate .py file located in the main
#   cam-control folder (the one with control.py)

# import the filters module
from plugins.filters import examples

import numpy as np

# create and setup the filter
flt = examples.MovingAverageFilter()
flt.setup()

# specify filter parameters
flt.set_parameter("length", 100)
flt.set_parameter("step", 10)

# generate input frames (here 300 random 256x256 frames);
#   must be a 3D array where the first axis is frame index
frames = np.random.randint(0, 256, size = (300,256,256), dtype = "uint16")
# feed the frames to the filter
flt.receive_frames(frames)
# calculate the result
result = flt.generate_frame()
```

5.3 Control server

To allow platform-independent external control of the software, there is an option to run a TCP/IP server. This server can receive commands from and send data to any other software running either locally on the same PC, or within the local network. The communication is mostly done via a text JSON protocol, which is straightforward to parse in most common languages.

5.3.1 Setting up and connection

By default, the server is not enabled. To activate it, you need to add the following line into the *settings file*:

```
plugins/serv/class  server
```

The default port on which the server runs is `18923`. If this port is occupied (e.g., by another instance of cam-control running), it tries to connect to the next 10 ports in ascending order. If that is still unsuccessful, it does not run at all. The port can also be manually set in the settings file with the following line:

```
plugins/serv/parameters/port        23456
```

If you use more than one camera, it might make sense to assign them different well-defined ports using the *camera-specific settings*:

```
css/ppimaq_0/plugins/serv/parameters/port   23456
css/uc480_0/plugins/serv/parameters/port    23467
```

Finally, if the PC has several network interfaces, you can specify the IP address in the same way as the port:

```
plugins/serv/parameters/ip  127.0.0.1
```

After the server is set up and software is started, the server starts automatically. If it is running, you can see its status on the bottom of the `Plugins` tab. It shows the server IP address and port, as well as the number of currently connected clients. Several clients can be operating simultaneously.

5.3.2 General message format

The protocol is based on JSON messages. Almost all of the messages sent to and from the server are simple JSON data. The only exception are the messages containing large data (e.g., frames), in which case the message consists of a JSON header and appended binary data, whose format is described within the header.

The first message kind is the one establishing the protocol. It has a simple format `{"protocol": "1.0"}`, where instead of `1.0` it can have any protocol version. The reply has the same format, which specifies the actual protocol used by the server. Currently only a single protocol (version `1.0`) is supported, so this message is not necessary. However, it is still recommended to start with it to make sure that the server runs the specified version and future-proof the applications.

Apart from this message, other messages follow the same general structure:

```
{
    "id": 0,
    "purpose": "request",
    "parameters": {
        "name": "cam/param/get",
        "args": {
```

(continues on next page)

(continued from previous page)

```
            "name": "exposure"
        }
    }
}
```

The first field is "id", which can contain a message ID. If it is defined, then the reply to this message will have the same value of the "id" field. If it is omitted, then "id" is omitted in the reply as well.

The second field is "purpose", which specifies the purpose of the message. The messages sent to the server have purpose "request", which is assumed by default if this field is omitted. The other possibilities used in the server-sent messages are "reply" for a reply to the request or "error" if an error arose.

The next field is "parameters", which describe the parameters of the request, reply, or error. Request parameters have two sub-fields "name" and "args" specifying, correspondingly, request name and its arguments. Depending on the request, the arguments might also be omitted.

The last possible field (not shown above) is "payload", which signifies that the JSON message is followed by a binary payload and describes its parameters. It is encountered only in some special replies and is described in detail later.

The requests and replies normally have the same format, with the reply typically having the same name but different set of arguments. The error messages have "name" parameter describing the kind of error (e.g., "wrong_request or "wrong_argument"), "description" field with the text description and "args" field with further arguments depending on the error kind.

Finally, note again that in request only "parameters" field is necessary. Hence, the command above can be shortened to {"parameters":{"name":"cam/param/get","args":{"name":"exposure"}}} and, e.g., to start camera acquisition you can simply send {"parameters":{"name":"cam/acq/start"}}.

5.3.3 Requests description

GUI requests

These requests directly address the GUI. They are almost directly analogous to entering values and pressing buttons in the GUI or reading values of controls or indicators:

- "gui/get/value": get value of a GUI parameter
  - *Request args*:
    - "name": parameter name; by default, return all parameters
  - *Reply args*:
    - "name": parameter name, same as in request
    - "value": parameter value; can be a dictionary
  - *Examples*:
    - {"name": "gui/get/value", "args": {"name": "cam/save/path"}} requests the saving path
    - {"name": "gui/get/value"} requests all GUI values
- "gui/get/indicator": get value of a GUI indicator; not that many indicators (e.g., anything in the status tables) are still technically values, and their values should be requested using "gui/get/value"
  - *Request args*:
    - "name": indicator name; by default, return all indicators

  - *Reply args*:
    * `"name"`: indicator name, same as in request
    * `"value"`: indicator value; can be a dictionary
  - *Examples*:
    * `{"name": "gui/get/indicator", "args": {"name": "cam/cam/frame_period"}}` requests the camera frame period indicator
    * `{"name": "gui/get/indicator"}` requests all GUI indicators
- `"gui/set/value"`: set value of a GUI parameter
  - *Request args*:
    * `"name"`: parameter name
    * `"value"`: parameter value
  - *Reply args*:
    * `"name"`: parameter name, same as in request
    * `"value"`: set parameter value; normally the same as in request, but can differ if, e.g., the range was coerced
  - *Examples*:
    * `{"name": "gui/set/value", "args": {"name": "cam/save/batch_size", "value": 100}}` sets the saving frames limit to 100
    * `{"name": "gui/set/value", "args": {"name": "plugins/trigger_save.ts/params/period", "value": 2}}` sets the period of the saving trigger to 2 seconds

To initiate a button press, you need to set its value to `True`.

Save requests

These requests initiate or stop data streaming to the drive:

- `"save/start"`: start the standard saving with the specified parameters; the parameters which are not specified are taken from the GUI
  - *Request args*:
    * `"path"`: save path
    * `"batch_size"`: number of frames per saved video (`None` for no limit)
    * `"append"`: determines whether the data is appended to the existing file
    * `"format"`: file format; can be `"raw"`, `"tiff"`, or `"bigtiff"`
    * `"filesplit"`: number of frames to save per file (`None` if no splitting is active)
    * `"save_settings"`: determines whether the settings are saved
  - *Reply args*:
    * `"result"`: should be `"success"` if the saving was successful
  - *Examples*:
    * `{"name": "save/start"}` starts saving with all parameters as specified in the GUI

        * `{"name": "save/start", "args": {"batch_size": 10}}` starts saving of 10 frames with all other parameters as specified in the GUI

- `"save/stop"`: stop the standard saving if it is running; no parameters are specified
  - *Reply args*:
    * `"result"`: should be `"success"` if the saving was successful
- `"save/snap"`: perform a snapshot saving with the specified parameters; the parameters which are not specified are taken from the GUI
  - *Request args*:
    * `"source"`: snapshot frame source; normally either `"standard"` (frame from the `Standard` image tab) or `"filter.filt"` (frame from the `Filter` image tab)
    * `"path"`: save path
    * `"format"`: file format; can be `"raw"`, `"tiff"`, or `"bigtiff"`
    * `"save_settings"`: determines whether the settings are saved
  - *Reply args*:
    * `"result"`: should be `"success"` if the saving was successful
  - *Examples*:
    * `{"name": "save/snap"}` snaps the image with all parameters as specified in the GUI
    * `{"name": "save/start", "args": {"source": "filter.filt"}}` snaps an image from the filter tab with all other parameters as specified in the GUI

Note that if the path is explicitly specified in the request, then this exact path is used. That is, of `On duplicate name` is set to `Rename` in the interface, it will not take an effect.

Camera requests

These requests directly control the camera:

- `"acq/stop"`: start the camera acquisition; no parameters are specified
  - *Reply args*:
    * `"result"`: should be `"success"` if the saving was successful
- `"acq/stop"`: stop the camera acquisition if it is running; no parameters are specified
  - *Reply args*:
    * `"result"`: should be `"success"` if the saving was successful
- `"acq/param/get"`: set the camera parameter
  - *Request args*:
    * `"name"`: parameter name; by default, return all parameters
  - *Reply args*:
    * `"name"`: parameter name, same as in request
    * `"value"`: parameter value; can be a dictionary
  - *Examples*:

        * `{"name": "cam/param/get", "args": {"name": "exposure"}}` requests the camera exposure
        * `{"name": "cam/param/get"}` requests all camera parameters
- `"acq/param/set"`: set the camera parameter
    - *Request args* contain camera parameters and their values (parameter names are the same as given by `acq/param/get` command)
    - *Reply args*:
        * `"result"`: should be `"success"` if the saving was successful
    - *Examples*:
        * `{"name": "cam/param/set", "args": {"exposure": 0.1, "roi": [0, 256, 0, 256]}}` set the camera exposure to 0.1 s and ROI to span from 0 to 256 on both axes

Streaming requests

These requests control transfer of the camera data.

Some of the replies can contain binary frame data, so their format differs from other replies. First, in addition to `"purpose"` and `"parameters"` field they also contain `"payload"` field with the information regarding the binary data. For example, the full JSON header can look like

```
{
    "purpose": "reply",
    "parameters": {
        "args": {
            "first_index": 41849,
            "last_index": 41948
        },
        "name": "stream/buffer/read"
    },
    "payload": {
        "nbytes": 13107200,
        "dtype": "<u2",
        "shape": [100, 256, 256]
    }
}
```

Payload description has 3 fields. First, `"nbytes"` specifies the total payload size in bytes. In the example above it states that this message is followed by `13107200` bytes of binary data. Next ,``"dtype"`` specifies the binary data format in the standard numpy format. Here `"<u2"` means that the data is 2-byte unsigned integer withe the little-endian byte order (the system default). Finally, `"shape"` specifies the shape of the result, i.e., dimensions along each axis when it is represented as a multidimensional array. In the example the shape is `[100, 256, 256]`, which means a 3D 100x256x256 array. In this particular reply the first axis is the frame index and the other 2 are the frame axes, i.e., the data contains 100 of 256x256 frames.

The streaming is done through requests, which means that it requires an intermediate buffer to store the frames between these requests (similar to, e.g., camera frame buffer). Hence, one first need to setup this buffer using `"stream/buffer/setup"` command, and then request the frames with `"stream/buffer/read"` command:

- `"stream/buffer/setup"`: setup the streaming buffer or clear it if it is already set up
    - *Request args*:

        - `"size"`: number of frames in the buffer; if not specified, set to 1 if the buffer is not set up or keep the current value if it is (in which case it just clears the buffer)
    - *Reply args*: same as `"stream/buffer/status"` (see below)
    - *Examples*:
        - `{"name": "stream/buffer/setup", "args": {"size": 100}}` sets up the buffer with 100 frames
- `"stream/buffer/clear"`: clear the streaming buffer
    - *Request args*: no arguments required
    - *Reply args*: same as `"stream/buffer/status"` (see below)
- `"stream/buffer/status"`: get the buffer status
    - *Request args*: no arguments required
    - *Reply args*:
        - `"filled"`: number of unread frames in the buffer
        - `"size"`: total size of the buffer (as specified with `"stream/buffer/setup"`)
        - `"first_index"`: index of the oldest frame in the buffer
        - `"last_index"`: index of the newest frame in the buffer
- `"stream/buffer/read"`: read some frames from the buffer
    - *Request args*:
        - `"n"`: number of frames to read; if not specified, read all frames; otherwise, read `n` oldest frames
        - `"peek"`: if `True`, return the frames but keep them in the buffer; otherwise (default), the frames are removed from the buffer after transfer
    - *Reply args*:
        - `"first_index"`: index of the first transferred frame
        - `"last_index"`: index of the last transferred frame
        - the frames data is contained in the payload as described above
    - *Examples*:
        - `{"name": "stream/buffer/read", "args": {"n": 10}}` requests 10 oldest frames from the buffer

CHAPTER

SIX

CONFIGURING

6.1 Command line arguments

`control.exe` takes several command line arguments to customize the software launch sequence:

- `--camera <camera name>`, `-c <camera name>`: select a specific camera to control; `<camera name>` is the name of the camera used within the settings file, such as `ppimaq_0`. Normally, if several cameras are available, the software first shows the dropdown menu to choose the camera to control. However, if this argument is specified, this camera will be selected automatically. Can be used to, e.g., create separate shortcuts for controlling different cameras.
- `--config-file <file>`, `-cf <file>`: specify a configuration file if it is different from the default `settings.cfg`. The path is always specified relative to `control.py` location.

6.2 Settings file

Software settings are defined in `settings.cfg` file inside the main folder. These include the camera connection parameters, as well as some additional interface and camera settings. The settings are described in a plain text format with one parameter per row. The most important settings are either defined by default, do not need explicit definition, or filled in by the camera detection. However, you can also edit them manually. Below is the list of available parameters.

6.3 General parameters

`interface/compact`

Switch to compact three-panel layout with a smaller controls area.
Values: `True`, `False`
Default: `False`

`interface/color_theme`

Color theme (based of qdarkstyle).
Values: `dark`, `light`, `standard` (standard Windows style)
Default: `dark`

`interface/expandable_edits`

Enable or disable expandable text boxes for paths and event logs.
Values: `True`, `False`
Default: `True`

`interface/plotter/binning/max_size`

Maximal size of the image shown in the plotter window (any image larger than this size gets binned). Only affect plotting, and only useful to speed up image display.
Values: any positive integer
Default: not defined (no maximal size)

`interface/plotter/binning/mode`

Binning mode for plotting display, if the frame needs to be binned. Works in conjunction with `interface/plotter/binning/max_size`.
Values: `mean`, `min`, `max`, `skip`
Default: `mean`

`interface/popup_on_missing_frames`

Show a pop-up message in the end of saving if the saved data contains missing frames.
Values: `True`, `False`
Default: `True`

`interface/datetime_path/file`

Template to generate file names when `Add date/time` is selected but `Create separate folder` is not.
Values: `pfx` (add date as a prefix, e.g., `20210315_120530_video.bin`), `sfx` (add date as a suffix, e.g., `video_20210315_120530.bin`), or `folder` (create folder with the datetime as name, e.g., `20210315_120530/video.bin`)
Default: `sfx`

`interface/datetime_path/folder`

Template to generate folder names when both `Add date/time` and `Create separate folder` are selected.
Values: `pfx` (add date as a prefix, e.g., `20210315_120530_video/`), `sfx` (add date as a suffix, e.g., `video_20210315_120530/`), or `folder` (create folder with the datetime as name, e.g., `20210315_120530/video/`)
Default: `sfx`

`interface/cam_control/roi_kind`

ROI entry method in camera control.
Values: `minmax` (ROI is defined by minimal and maximal coordinates), `minsize` (ROI is defined by minimal coordinates and size), or `centersize` (ROI is defined by center coordinates and size)
Default: `minsize` for PhotonFocus cameras, `minmax` for all other cameras

`frame_processing/status_line_policy`

Method to deal with a status line (on PhotonFocus or PCO edge cameras) for the raw image display. Only affects the displayed image.

Values: `keep` (keep as is), `cut` (cut off rows with the status line), `zero` (set status line pixels to zero), `median` (set status line pixels to the image median),or `duplicate` (replace status line with pixels from a nearby row)

Default: `duplicate`

saving/max_queue_ram

Maximal size of the saving buffer in bytes. Makes sense to increase if large movies are saved to slow drive, or if large pre-trigger buffer is used (the size of the saving queue must be larger than the pre-trigger buffer). Makes sense to decrease if the PC has small amount of RAM.

Values: any positive integer

Default: `4294967296` (i.e., 4 GB)

6.4 Camera-related parameters

In this section `<camera name>` stands for any camera name, e.g., `ppimaq_0`. For example, `cameras/<camera name>/params` can become `cameras/ppimaq_0/params`.

css/<camera name>/<key>

Camera-specific parameter. Allows for any generic parameter (such as GUI parameters described above) to take different values for different cameras. For example, if there are two cameras named `ppimaq_0` and `uc480_0` defined in the file, then one can use `css/ppimaq_0/saving/max_queue_ram` to specify the saving buffer size for the first camera, and `css/uc480_0/saving/max_queue_ram` for the second one. Either of those can also be omitted, in which case the generic value (either specified or default) will be used instead. Note that this option is not supported for `select_camera`, `cameras`, and `dlls` parameters (`css` entries are ignored for those).

select_camera

Default selected camera. If this parameter is set, then this camera is automatically selected even if several cameras are present (i.e., the camera select menu doesn't show up). In this case, the only way to start other cameras is by using `--camera` *command line argument*.

Values: any camera name (e.g., `ppimaq_0`)

cameras/<camera name>/params

Parameters for camera initialization (interface name, index, etc.) Created automatically by the `detect` script, and usually does not need to be changed

Values: depends on the camera

cameras/<camera name>/display_name

Camera name to be shown in the camera select window (if multiple cameras are available) and in the window header

Values: any text

Default: automatically filled by the `detect` script based on the camera kind, model, serial number, etc.

cameras/<camera name>/params/misc

Additional minor camera parameters

Values: depends on the camera (see generic parameters below)

`cameras/<camera name>/params/misc/buffer/min_size/time`

Minimal camera frame buffer size defined in terms of acquisition time (in seconds). For example, for `time = 0.5` the frame buffer size would be 50 frame for 100 FPS frame rate and 500 frames for 1 kFPS frame rate.

Values: any positive floating point number

Default: 1 second for most cameras

`cameras/<camera name>/params/misc/buffer/min_size/frames`

Minimal camera frame buffer size defined in terms of number of frames.

Values: any positive integer

Default: camera-dependent; usually, between 100 and 1000

For any given FPS the maximal of the two declared buffer sizes is used. For example, if `time = 1` and `frames = 100`, then at 50 FPS the frame buffer size is 100 (defined through `frames`), and at 1000 FPS the frame buffer size is 1000 (defined through `time`).

`cameras/<camera name>/params/misc/loop/min_poll_period`

The period to polled the camera for new frames. The new frames are read out from this camera with this period, which means that the *display* period is limited by the poll. However, since multiple frames are read out at once, the overall readout frame rate does not depend on the poll period. Lower number results in higher image update rates but also, usually, in somewhat lower performance.

Values: any positive number

Default: 0.05 (corresponding to the maximum of 20 FPS update rate)

`cameras/<camera name>/params/misc/trigger/in/src`

Source of the input trigger for cameras supporting several trigger sources

Values: camera-dependent. For IMAQ cameras (e.g., using NI frame grabber) a tuple `(kind, index)`, where `kind` can be `"ext"` (external SMB connector), `"rtsi"` (RTSI connection), or `"iso_in"` (ISO connection), and `line` is an integer line number. For example, `("ext",0)` is the default external SMB connector, and `("rtsi",4)` is the RTSI line 4.

Default: `("ext",0)`

`cameras/<camera name>/params/misc/trigger/out/src`

Destination of the output trigger for cameras supporting several trigger destinations

Values: camera-dependent. For IMAQ cameras (e.g., using NI frame grabber) a tuple `(kind, index)`, where `kind` can be `"ext"` (external SMB connector), `"rtsi"` (RTSI connection), or `"iso_out"` (ISO connection), and `line` is an integer line number. For example, `("ext",0)` is the default external SMB connector, and `("rtsi",4)` is the RTSI line 4.

Default: `("ext",0)`

6.5 Specific system parameters

dlls/<camera interface>

Paths to camera-specific DLL locations, if different from the device location. <camera interface> can stand for one of the following:

- andor_sdk2: path to atmcd64d.dll for Andor SDK2. By default, search in the default location of Andor Solis.
- andor_sdk3: path to atcore.dll and related DLLs for Andor SDK3. By default, search in the default location of Andor Solis.
- dcamapi: path to dcamapi.dll and related DLLs for Hamamatsu/DCAM cameras. By default, search in System32 folder, where it is placed after installing DCAM API or Hokawo software.
- niimaq: path to imaq.dll for NI IMAQ frame grabber interface. By default, search in System32 folder, where it is placed after installing NI Vision Acquisition Software.
- niimaqdx: path to niimaqdx.dll for NI IMAQdx frame grabber interface. By default, search in System32 folder, where it is placed after installing NI Vision Acquisition Software.
- pco_sc2: path to SC2_Cam.dll for PCO cameras. By default, search in the default location of pco.camware or pco.sdk.
- picam; path to picam.dll for Princeton Instruments cameras. By default, search in the default location of Princeton Instruments PICam Runtime.
- pfcam: path to pfcam.dll for PhotonFocus cameras. By default, search in PFRemote folder specified in the PATH environment variable.
- pvcam; path to pvcam64.dll for Photometrics cameras. By default, search in System32 folder, where it is placed after installing PVCAM software.
- sisofgrab: path to fglib5.dll for Silicon Software frame grabber interface. By default, search in Silicon Software Runtime Environment folder specified in the PATH environment variable.
- thorlabs_tlcam: path to thorlabs_tsi_camera_sdk.dll and related DLLs for Thorlabs Scientific Cameras. By default, search in the default location of ThorCam.
- uc480: path to uc480_64.dll and related DLLs for uc480 camera interface. By default, search in the default location of ThorCam.
- ueye: path to ueye_api_64.dll and related DLLs for uEye camera interface. By default, search in the default location of IDS Software Suite.

CHAPTER

SEVEN

USE CASES

Here we describe some basic use cases and applications of the cam-control software.

- **Standard acquisition**

  This is the standard mode which requires minimal initial configuration. Simply setup the *camera parameters*, configure the *saving*, start the acquisition, and press `Saving`.

- **Recording of rare events**

  Manual recording of rare and fast events is often challenging. Usually you have to either record everything and sift through the data later, or hope to press `Saving` quickly enough to catch most of it. Neither option is ideal: the first method takes a lot of extra storage space, while the second requires fast reaction. *Pretrigger buffer* takes the best of both worlds: it still allows to only record interesting parts, put lets you start saving a couple seconds before the button is pressed, so it is not necessary to press it as quickly as possible. The buffer is set up in the *saving parameters*.

- **Periodic acquisition**

  To record very slow processes, you might want to just occasionally take a snapshot or a short movie. You can use *saving trigger* for that.

- **Image-based acquisition start**

  Another *saving trigger* application is to automate data acquisition. It gives an option to start acquisition based on the frame values. Combined with *custom filters*, it provides a powerful way to completely automate acquisition of interesting events.

- **Background subtraction**

  Oftentimes the camera images contain undesired static or slowly changing background. It is possible to get rid of it using the built-in basic *background subtraction*. Keep in mind that it only affects the displayed data (and, as such, is not applied to frames supplied to filters).

- **On-line image processing**

  More complicated on-line processing is available through filters. Cam-control already comes with several basic *built-in filters*, but the real power in custom application comes from the ability to easily write *custom filters*.

- **On-line analysis of fast processes**

  To do a quick on-line analysis of fast processes, you can use the *frame slowdown* capability.

- **Interfacing with external software**

  Cam-control provides a *TCP/IP server control* which allows one to control GUI, start or stop acquisition and saving, and directly acquire frames from the camera. Since this is implemented as a server, it is platform- and software-independent, so it can interface with any custom software written in, e.g.,

C++, Java, LabView, or Matlab. It can be used to synchronize data acquisition with other devices, implement more sophisticated automation, or provide frames for custom image-processing software.

- **Controlling several cameras**

  Cam-control allows for control of several connected cameras. If more than one camera is specified in the settings file (typically these are found by running the `detect` script), then every time the software is started, it allows to select which camera to control. Several instances can run simultaneously for different cameras without interference. The default GUI parameters are stored independently for all cameras.

  The selection window can be avoided by creating a specific camera shortcut in the *extras menu*. It creates a shortcut which uses the `--camera` *argument* (e.g., run `control.exe --camera ppimaq_0`) to supply the specific camera name.

CHAPTER

EIGHT

TROUBLESHOOTING

- **Camera is not detected by the software**
  - Camera is disconnected, turned off, its drivers are missing, or it is used by a different program: Andor Solis, Hokawo, NI MAX, PFRemote, PCO Camware, ThorCam, etc. Check if it can be opened in its native software.
  - Software can not find the libraries. Make sure that the *native camera software* is installed in the default path, or *manually specify the DLL paths*.
  - Frame grabber cameras and IMAQdx cameras currently have only limited support. Please, *contact the developer* to see if it is possible to add your specific camera to the software.
- **Camera camera controls are disabled and camera status says "Disconnected"**
  - Camera is disconnected, turned off, its drivers are missing, or it is used by a different program. Check if it can be opened in its native software.
- **The image is not updated when the camera acquisition is running**
  - Make sure that you are looking at the correct image tab, and that the `Update` button is pressed.
  - The camera is in the external trigger mode and no triggering signal is incoming. Switch the camera to the internal trigger mode.
  - In some cases certain camera setting combinations result in unstable behavior. Check if these exact camera settings work in the native software.
- **The operation or frames update is slow or laggy**
  - Frame plotting takes up too much resources. Reduce the frame update period or, if necessary, turn off the display update.
  - Additional plotting features take up too much resources. Turn off frame cuts plotting, color histogram, and switch to the plain grayscale color scheme.
  - On-line processing takes up too much resources. Turn off pre-binning, background subtraction, filters (disable or unload), and mean frame plots.
  - Slowdown is turned on, camera frame rate is low, or time frame binning is enabled with a large factor. All of these can lead to actually low generated frame rate.
  - Selected display update period (selected under the plot parameters) is too large.
  - By default, the camera is polled for new frames every 50ms, so the frame rate is limited by 20 FPS. To increase it, you can specify the `loop/min_poll_period` parameter the *settings file*.
- **Acquisition is frozen** - Acquisition is still being set up, which can take up to several seconds in some cases. This can be confirmed by `Acquisition` camera status saying `Setting up...`. - Camera buffer overflow on Andor SDK3 cameras (e.g., Zyla) can sometimes lead to such behavior. Reduce the data transfer rate using smaller

frame rate, smaller ROI or larger binning, and then restart the acquisition. - Some NI IMAQ frame grabbers can freeze if the camera data transfer rate is too high (~200Mb/s). In this case it will freeze the acquisition shortly after start. This can be confirmed by the disagreement between the expected frame rate from the `Frame period` indicator and the actual frame rate from the `FPS` camera status. If this is the case, reduce the data transfer rate using smaller frame rate, smaller ROI or larger binning, and then restart the acquisition.

- **Missing frames are reported after saving**
  - Acquisition can not deal with high data or frame rate. Check if the *frame buffer* is full or constantly increasing. If so, reduce the frame rate or frame size.
  - Frame info acquisition takes too much time. On some cameras (e.g., uc480 and Silicon Software frame grabbers) acquiring frame information can take a significant fraction of the frame readout, especially for small frames and high readout rates. If this is the case, you need to turn the frame info off.
  - Frames pre-binning can not deal with high data rate. Check if the frame buffer is full or constantly increasing. If so, reduce the frame rate or frame size, or turn the pre-binning off.
  - Software as a whole can not deal with high data or frame rate. Minimize the load from unnecessary programs running on this PC. Avoid using remote control software (e.g., TeamViewer or Windows Remote Desktop) during the saving sessions.
  - Acquisition has been restarted during saving (e.g., due to parameters change).
  - PhotonFocus cameras can generate frames faster than some frame grabbers (e.g., SiliconSoftware microEnable 4) can manage. This shows up as lower frame rate than expected from the frame period. If this is the case, reduce the frame rate.
  - The data rate is higher than the drive writing rate, and the *save buffer* is overflown. Reduce the size of a single saving session, switch to a faster drive (SSD), or increase the save buffer size.
  - (especially if missing frames occur right at the beginning) Obtaining camera settings for saving takes too much time. Increase the buffer size using `misc/buffer/min_size/time` parameter in the *settings file*, or turn off the settings file (not recommended).
- **Saving in Tiff format ends abruptly or produces corrupted files**
  - Tiff format does not support files larger than 2 Gb. Either split data in smaller files (e.g., using the *file split* settings), or use other format such as BigTiff.
- **Control window is too large and does not fit into the screen**
  - You can enable the compact mode in the *setting file*.
- **Camera performance is lower than can be achieved in the native software**
  - Make sure that all available settings (including advanced settings such as readout speed, pixel clock, etc.) are the same in both cases.
  - Some specific cameras might not be fully supported. Please, *contact the developer* to add necessary settings of your specific camera to the software.

CHAPTER

NINE

RELEASE HISTORY

9.1 Version 2.2.1

2022-10-05

- Added Basler pylon-compatible camera, BitFlow frame grabbers with PhotonFocus cameras, and support for AlliedVision Bonito cameras with National Instruments frame grabbers.
- Added basic beam profiler filter.
- Minor bugfixes.

Download

9.2 Version 2.2.0

2022-04-04

- Added Photometrics cameras.
- Added camera attributes browser to Andor SDK3, Hamamatsu, Photometrics, Princeton Instruments, and IMAQdx cameras.
- Added lines coordinate systems to the plotter, added ROI selection in the image.
- Added a separate event logging window for a more convenient logging of larger messages.
- Added single-shot data recording for high-performance saving.
- Altered the save trigger behavior: now the new save can start only after the previous is done.
- Minor bugfixes.

Download

9.3 Version 2.1.2

2021-12-23

- Graphics and UI update: added launcher executables, “About” windows, popup error messages.

Download

9.4 Version 2.1.1

2021-12-12

- Added preferences editor.
- Added option for importing settings from a previous software version.
- Added creation of camera-specific shortcuts.
- Minor GUI bugfixes.

Download

9.5 Version 2.1.0

2021-12-04

- Added tutorial.
- Added activity indicator.
- Implemented color themes.
- Added IDS uEye and Princeton Instruments cameras.
- Slightly reorganized GUI.
- Minor bug fixes, wider cameras support.

Download

9.6 Version 2.0.1

2021-10-05

- Changed event log format: added recorded frame number and column name header.
- Reorganized snapshot saving controls.
- Added processing steps indicators above the image plots.
- Minor bug fixes.

Download

9.7 Version 2.0.0

2021-09-29

Major upgrade, first publicly released version.

Download

BIBLIOGRAPHY

[Young2018] Gavin Young et al., "Quantitative mass imaging of single biological macromolecules," *Science* **360**, 423-427 (2018)

[Dastjerdi2021] Houman Mirzaalian Dastjerdi, Mahyar Dahmardeh, André Gemeinhardt, Reza Gholami Mahmoodabadi, Harald Köstler, and Vahid Sandoghdar, "Optimized analysis for sensitive detection and analysis of single proteins via interferometric scattering microscopy," *bioRxiv doi*: 10.1101/2021.08.16.456463