Hyperspectral Imaging Controller Software User Manual

Laurent Fasnacht

February 4, 2019

1 Introduction

In this chapter, we will present how to set up and use a minimal hyperspectral imaging controller software (hics) system. By hics system, we consider all the components which interact to enable hyperspectral data acquisition.

As seen in chapter **??**, the basic system requirement is Redis, which should be accessible by any components of the system. The exact device on which Redis runs is not important, but the interconnection should be fast enough to allow uncompressed frames to be transmitted at the correct frame rate. Usually, Gigabit ethernet is fine, but Wifi won't be sufficient.

Hics is designed to control a pushbroom hyperspectral camera system, which generally consists of:

- Camera controls (integration time, frame rate)
- Framegrabber (captures each frame coming from the camera)
- Scanner (moves the camera, the sample, or a mirror in order to get a 2D image
- Focus (selection of the depth of sharpness)
- Shutter (controls whether the light reaches the sensor or not)

The exact way these components are connected to the hics system depend on the hardware. As an example, we present the different commands that should be run in order to use the setup used in chapter **??**.

- `python3 -m hics.hardware.specimswir.camera` (on the server; camera control)
- `python3 -m hics.hardware.specimswir.scanner` (on the server; mirror scanner control)
- `python3 -m hics.hardware.specimswir.framegrabber` (on the server; frame grabber)
- `NiRawStreamer.exe img0 192.168.1.2 1234` (on the framegrabber; capture and send frames to the framegrabber. `img0` is the name of the NI interface)
- `python3 -m hics.hardware.ev3.ev3focus --redis redis://192.168.1.2` (on LEGO ev3 brick; focus control)
- `python3 -m hics.hardware.ev3.ev3rotater --redis redis://192.168.1.2` (on LEGO ev3 brick; rotating device)
- `python2 -m hics.hardware.webcam --redis redis://192.168.1.2` (on the laptop; webcam capture)
- `python3 -m hics.daemon.frameconverter` (on the server; basic correction for preview),
- `python3 -m hics.gui` (on the laptop; graphical user interface)
- `python3 -m hics.plugin.photogrammetry --redis redis://192.168.1.2` (on the laptop; capture data from webcam)
- `python3 -m hics.plugin.record --redis redis://192.168.1.2` (on the laptop; record hyperspectral frames to file)

- `python3 -m hics.plugin.autofocus --redis redis://192.168.1.2` (on any system; autofocus plugin)
- `python3 -m hics.plugin.labelprint --redis redis://192.168.1.2` (on any system; plugin to print labels)
- `python3 -m hics.plugin.autoscan --redis redis://192.168.1.2` (on any system; plugin to automate the plugins for library scanning)

As we can see, even though the command lines are quite simple (they mostly consist in indicating the redis server URL), there are numerous components. To allow simple testing, a hyperspectral camera simulator has been written. The simplest setup is to have a Redis server on the local computer, and run the following commands:

- `python3 -m hics.hardware.simulator --datafile datafile.scan` (simulate a hyperspectral camera using data from `datafile.scan`),
- `python3 -m hics.daemon.frameconverter` (basic correction for preview),
- `python3 -m hics.gui` (graphical user interface).

The simulator emulates the camera, the frame grabber, the scanner, the shutter and focus. This is in generally sufficient to test the graphical user interface and plugins.

2 Capturing data

Usually, the process for scanning a sample is the following:

1. Define the bounds for the scanner
2. Set the focus
3. Choose an integration time
4. Capture dark frames, and scan the sample

The specific setup is out of the scope of this user manual, but is instead discussed in chapter **??**. We therefore consider that the hardware is functional (and all related components are running), or that the simulator is used. The only non-hardware component required is `hics.daemon.frameconverter` to do basic dark frame subtraction for the graphical user interface. It is generally run on the same computer as the Redis server.

First, launch the Hyperspectral Imaging Control System (HICS) graphical user interface (`python3 -m hics.gui`). If everything goes well, an empty window is shown. If the redis server is not reachable, the user will be asked to provide the Redis URL (it can also be changed using the `File / Settings` menu. Usually, one would like to have at least the camera settings, the scanner settings, the camera live output, and the waterfall plot displayed (as in fig. 1). This can be done using the `Window` menu.

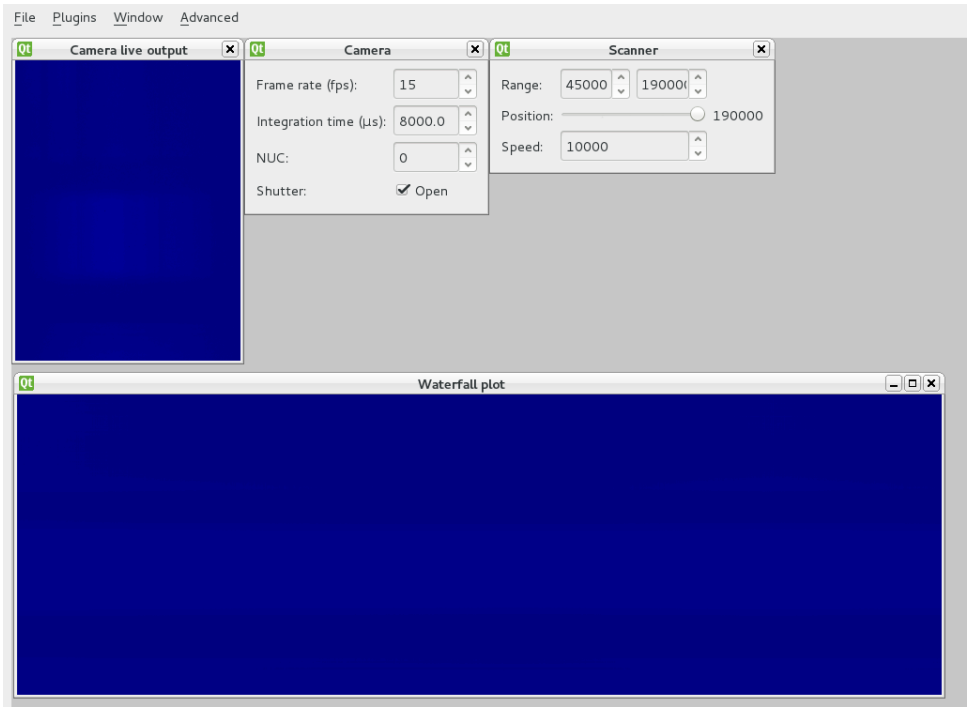

Figure 1: HICS GUI, with the common windows displayed.

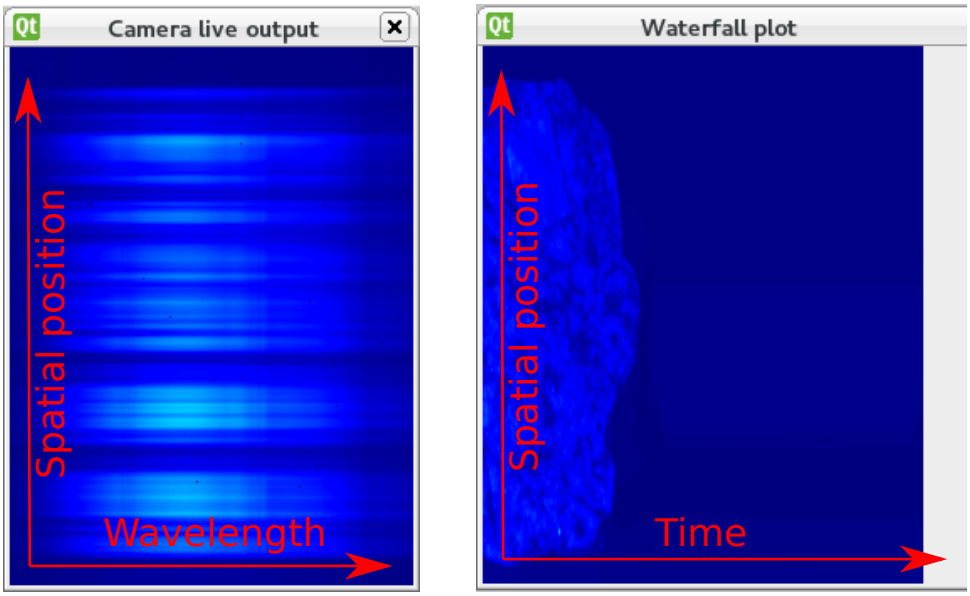

Figure 2: On the left, the `Camera live output` window shows what is captured by the sensor. The horizontal dimension is spectral, the vertical dimension is spatial. Each captured frame is averaged spectrally, and added to the left of the `waterfall plot`, enabling the operator to see a crude 2D image.

There are a few additional functions to the interface. One can click on the camera live output to get a magnitude plot (right button) or a spectrum plot (left button) for a given position. Also, the currently active plot window (or the camera live output) can be exported to Python script by using the `File / Export data` menu entry.

2.1 Estimating the scanner parameters

First, it is convenient to have the correct parameters for the scanner before working on any other settings, as it avoids getting "lost" is some unknown area.

To do so, the simplest method is to begin by settings convenient parameters:

- A human adapted frame rate (e.g. 25 fps)
- A large integration time (depends on the camera and the illumination, for example 10 000 µs)
- A fast scanning speed

Then, a large scanning range should be set. Using the position slider, the operator should try to find the beginning and the end of the region of interest, and adapt the bounds accordingly. Usually, the best approach is to follow an incremental process: first determine roughly the bounds, then lower the scanning speed, then adapt the bounds to increase their precision.

Usually, it is better to leave a few percent margin on both sides, to avoid accidental loss of the data on the sides, as the margin pixels can be masked easily in the data processing steps. On the other hand, it is not advised to have huge margins, since this will increase the size of the captured data.

2.2 Focus

Once the scanning bounds are set, it is required to focus the camera in order to get sharp pictures. If the setup supports motorized focus control, it is possible to control it using the window depicted in figure 3, otherwise it has to be done manually by the operator by turning the focus rig.

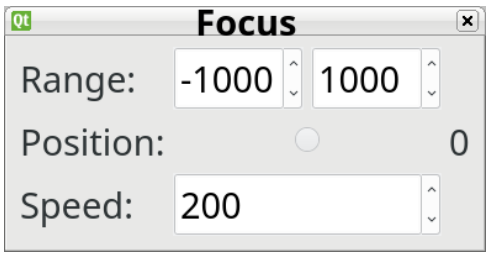

Figure 3: Window to control the focus.

There are two plugins to help the operator to set the focus, both using contrast based methods. The first one is the manual focus plugin, which helps the operator to focus the image, which works both with manual and motorized focus. The second one is the automatic focus plugin. It requires a motorized focus. Their usage is otherwise similar.

The manual focus plugin can be launched using `python3 -m hics.plugin.focushelper`, and the automatic one using `python3 -m hics.plugin.focushelper`.

The first thing to do is to decide which region should be used for focus. If there is no specific region of interest, this should be the middle of the sample. First, move the position of the scanner to this position, using the position slider. Then, launch the `Manual focus` or the `Automatic focus` plugin using the `Plugins` menu. The automatic focus will ask for the percentage of the frame which should be used for contrast computation, while the manual one will use the full frame.

A new window will open, with a live plot of the contrast of the current image returned by the camera. For the automatic plugin, it is only for monitoring and everything will happen automatically. For the manual focus, the operator then turn the focus rig, in order to maximize the contrast shown on the plot. Usually the simplest approach is to turn the rig in one direction. If the contrast decreases, then the direction should be changed, otherwise just continue till a maximum is reached, and the values start decreasing again (at that time, the rig should be turned back a little). This is exactly the strategy that is implemented in the automatic plugin. Note that the window showing the contrast can be resized to improve its visibility.

Once the focusing is done, the window can be closed. If the image focus was changed a lot, it may be required to adapt the scanning area, as changing the focus may have a zooming effect.

2.3 Integration time

Finally, the last step is to define a good integration time (exposure). This is not strictly required as it is possible to capture multiple images, and then to merge them using HDR techniques, but it helps to have a starting point. The software is provided with the `Auto-exposure plugin`, and can be started using `python3 -m hics.plugin.autoexposure`, and then called using the `Plugins` menu.

It requires 3 parameters, the percentile (p), the lowest acceptable magnitude (M_{min}) and the highest acceptable magnitude (M_{max}). It will do multiple acquisition of the image within the scanning range, and will adapt the integration time, such that the p-th percentile of the captured image is between M_{min} and M_{max}. This percentile is used to avoid the few extreme values that can arise (like a bad pixel). The default values ($p = 0.99$, $M_{min} = 0.7$, $M_{max} = 0.9$) are usually fine.

2.4 Other parameters

It is usually better to disable the camera non-uniformity correction by setting NUC to 0. To update the dark frame subtraction to the new parameters, one should toggle the shutter button in order to capture a few dark frames. The dark frame subtraction for the preview will be automatically adapted.

2.5 Scanning an image using the `record` plugin

There is a huge variety of type of scans that can be made (for example, HDR scans, time lapse, etc.). The most often used methods are implement in the `Record` plugin, which can be run using `python3 -m hics.plugin.recordscan`. Users requiring more peculiar acquisition types should implement them as plugins. This is not covered in this user manual.

A white reference should be put in the scanning range, and the scanner should be set to this position using the position slider. Then the `Record` plugin should be launched using the `Plugins` menu, and its window should appear (fig. 4).

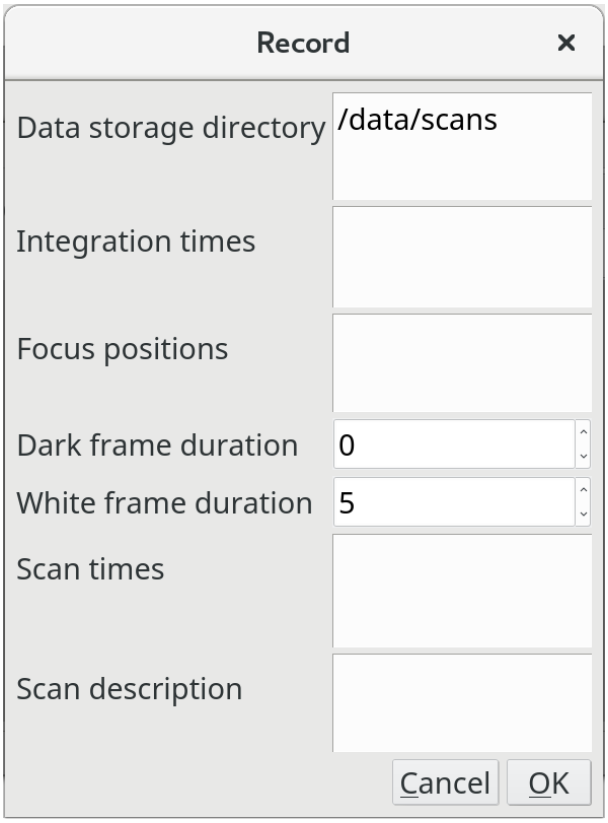

Figure 4: The record plugin

The output directory can be specified using the `Data storage directory` field. A file will be created based on the current date and time. Depending on the other parameters specified, it can contain one

or more scans. The parameters of the scans can be specified using the `Integration times`, `Focus positions` and `Scan times` fields. Each of these fields can contain either no value (will use the current value of the camera), or a comma separated list of ranges, using the following format:

- A single value (ex: `1234`)
- A range (`10:15` is equivalent to `10, 11, 12, 13, 14`)
- A range with step size (10:20:2 is equivalent to `10, 12, 14, 16, 18`)

`Scan times` can be used for a time lapse. For example specifying `Scan times = 0:3601:60` will make a capture each minute for one hour. Each of these captures will consist in possibly multiple scans for each combination of `Integration times` and `Focus positions`. Specifying multiple integration times can be useful if the sample has a large difference in luminosity between bright and dark areas. Multiple focus positions are only useful if the sample has a complex shape, in order to do focus stacking at a later stage.

`Dark frame duration` is the duration of the capture of dark frames before and after each scan, in order to estimate the average noise and variance of the dark current. To capture only a single frame, it is possible to set the value 0. `White frame duration` is the duration of the scan of the white frames. In addition to its usefulness to compute reflectance, white frames are used by calibration to estimate the noise pattern of the sensor and therefore to detect bad pixels.

Finally, `Scan description` is a field which can be freely used to write the description of the scan, this text is embedded in the scan file.

Once everything is set up, click the `OK` button, and the scan(s) will be automatically performed. The different windows can be used to monitor the status of the camera and the data captured.

3 Data processing

As many different data processing possibilities exist, we will show only a simple one, which should be generally sufficient. At every step, the file is a `mmappickle` [1] dictionary file. It can be loaded by the viewer, even if the processing is not finished. It is for example possible to display the contents of the file while the scans are being made. If `mmappickle` is not available, it can be loaded using `pickle.load`, at the expense of higher memory usage. At any steps, at least the following keys are available:

- `description`: contains a text description of the contents of the file. It is up to the user to provide a meaningful description.
- `wavelengths`: list or numpy array of the wavelengths corresponding to the bands
- `processing_steps`: list of the processing steps applied to the file. It is not present for files containing raw data only.

It is possible to list the contents of such files on the command line using the following command `python3 -m hics.datafile.ls`.

The file obtained when using the `Record` plugin contains the following keys (`#####` is the scan id, all hyperspectral data is in digital numbers):

- `scan-#####`: contains the data captured when doing the scan.
- `scan-#####-d0`, `scan-#####-d1`: dark frames captured before and after the scan
- `scan-#####-p`: properties of the scan (integration time, scanner positions, timestamps)
- `white-#####`: contains the white frame data.
- `white-#####-d0`, `white-#####-d1`: dark frames captured before and after the white frames
- `white-#####-p`: properties of the white frame scan (integration time, scanner positions, timestamps)

3.1 Removing bad pixels and computing reflectance

This step is done using the `hics.datafile.calibrate` tool. Its options can be seen using `python3 -m hics.datafile.calibrate --help`. For basic use, the default parameters are usually good enough, and it is sufficient to run `python3 -m hics.datafile.calibrate --input <filename.scan> --output <filename.calibrated>`.

The file obtained contains the following keys (`#####` is the scan id):

- `scan-#####`: contains the data captured when doing the scan, with dark frame removed and bad pixels masked. Units are $\mathrm{DN}\,\mu\mathrm{s}^{-1}$.
- `scan-#####-p`: properties of the scan (integration time, scanner positions, timestamps)
- `refl-#####`: contains the reflectance data.
- `refl-#####-var`: contains an estimate of the variance of the reflectance data.
- `refl-#####-w`: contains the white frame used to compute reflectance data.
- `refl-#####-p`: properties of the scan (integration time, scanner positions, timestamps)

3.2 Creating a HDR file

To merge the different scans, `hics.datafile.merge` can be used. The number of valid information required per pixel can be specified using the `--required_images` parameter. At least one image is required to get the value of the pixels, but if enough scans were made is it advised to increase this value in order to get a better quality HDR image. An example command line would be: `python3 -m hics.datafile.merge --input <filename.calibrated> --output <filename.hdr> --required_images 2 --clean`. The `--clean` parameter removes all intermediate data (which can be large) from the HDR file.

The file obtained contains the following keys:

- `hdr`: contains the HDR image.
- `hdr-var`: contains an estimate of the variance.

3.3 Creating and applying a mask

A mask can be created by any image manipulation software. It consists in a black and white image with same dimensions as the hyperspectral image, where white pixels correspond to pixels that should be kept.

The image can be exported as a grayscale `png` using: `python3 -m hics.datafile.maskdata --input <filename.hdr> --mask <filename.png>`.

Then, the image can be edited in order to create a mask such as the one shown in figure 5. The mask can be applied using: `python3 -m hics.datafile.maskdata --input <filename.hdr> --mask <filename.png> --output <filename.mhdr>`. The output file contains the same keys as the HDR file.

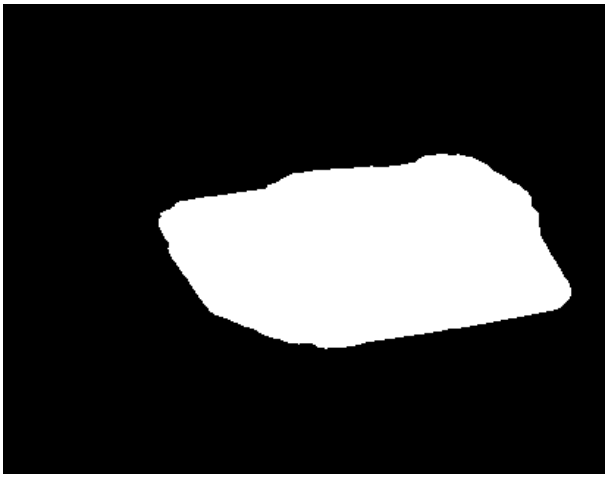

Figure 5: An example mask for a hyperspectral image.

4 Data viewer

It can be useful to visualize a data file at any step, for reasons such as ensuring that the data quality is sufficient, or to do some quick analysis. For this purpose, a simple viewer has been developed. It can be launched using `python3 -m hics.viewer <filename>`, where `<filename>` is any hyperspectral data file produced. Figure 6 shows an example view. The window is divided as follows:

- The list on the left shows the defined keys in the file.
- If the current key contains hyperspectral data, the top right area will contain 2D data, and the bottom right will contain spectra.
- If the current key contains other type of data, the right area shows directly the content of the key.

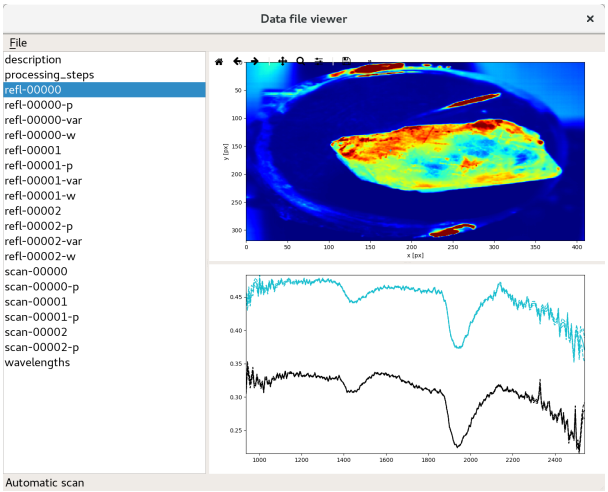

Figure 6: The data viewer window, showing a calibrated file.

For hyperspectral data, the 2D data part will show the average intensity of each pixel. The spectra for each point in the hyperspectral data can be seen by moving the mouse around in the 2D image. It is also possible to add points for which the spectra is always displayed (great for comparisons), by clicking with the right button in the 2D image, and selecting `Add point`. An example is shown in figure 7.

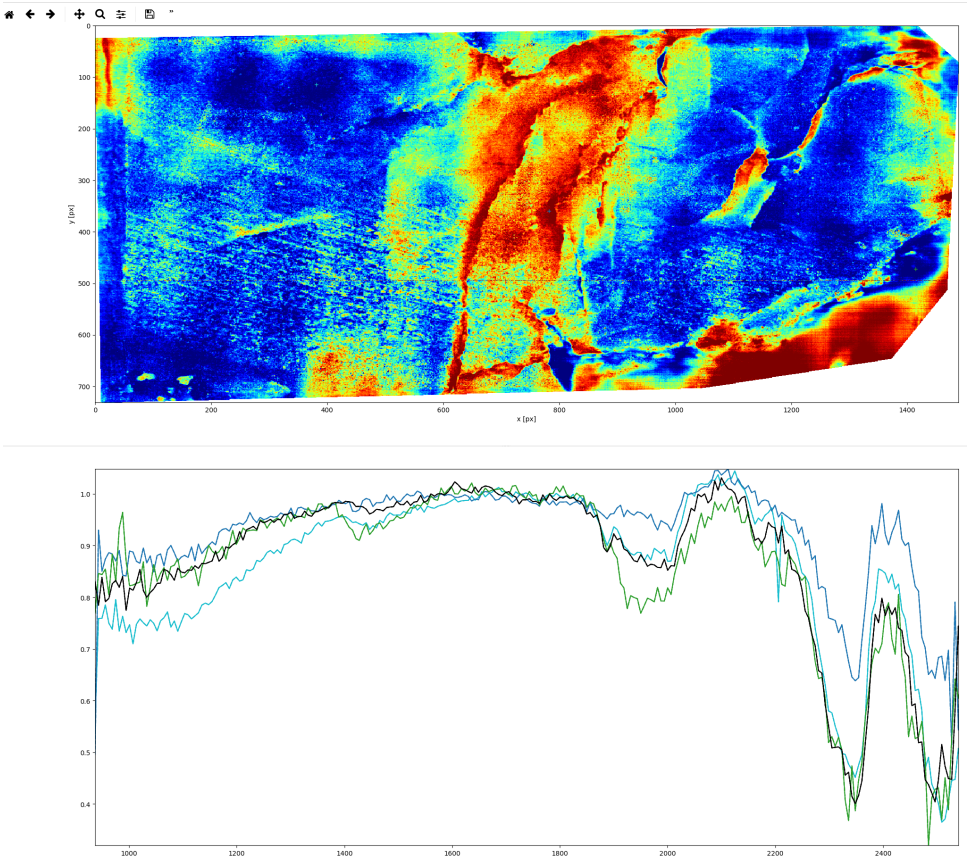

Figure 7: The data viewer window, with three data points selected. The spectrum in black correspond to the mouse cursor.

It is possible to change the bands displayed by right clicking on the spectrum displayed, and mapping a specific band to a given color. This is very useful to compare the relative intensity between bands (fig. 8.

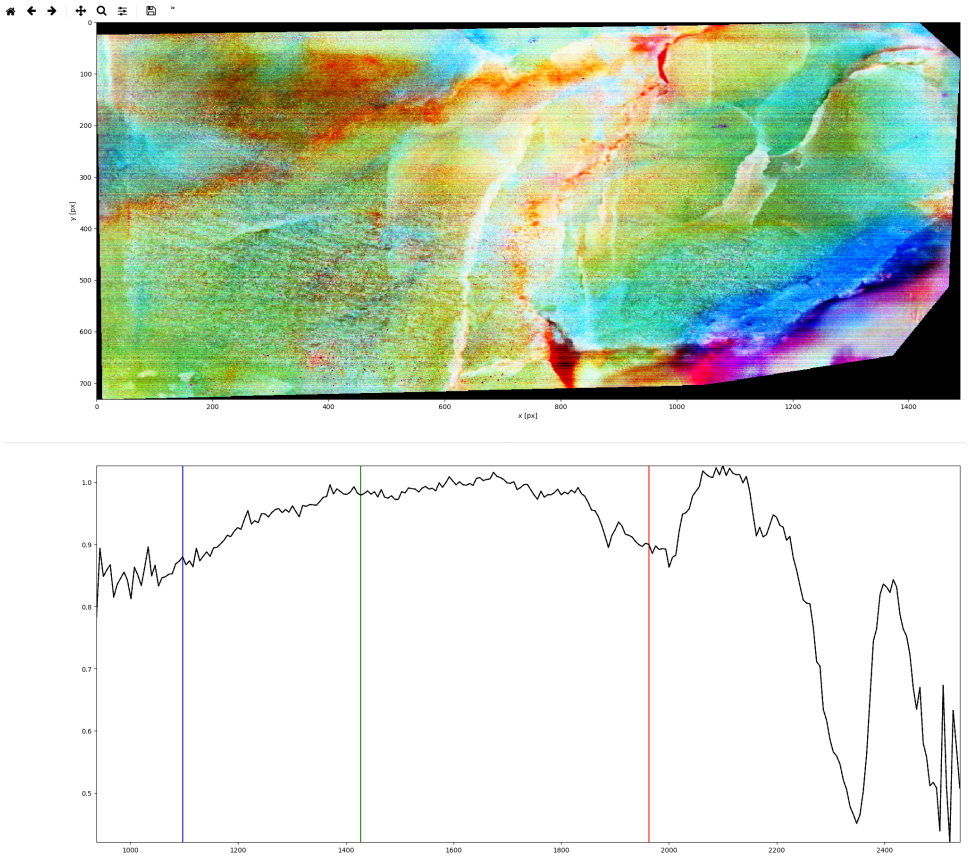

Figure 8: The data viewer window, with 3 bands selected.

The colormap can be changed by using the ⚟ button. For each of the colorband, a color-keyed histogram of the band is shown, and the curve used to map the values to the color map. The curve can be edited, using the left mouse button to add or move a point, and the right one to remove one. The results can be seen directly on the other window (fig. 9).

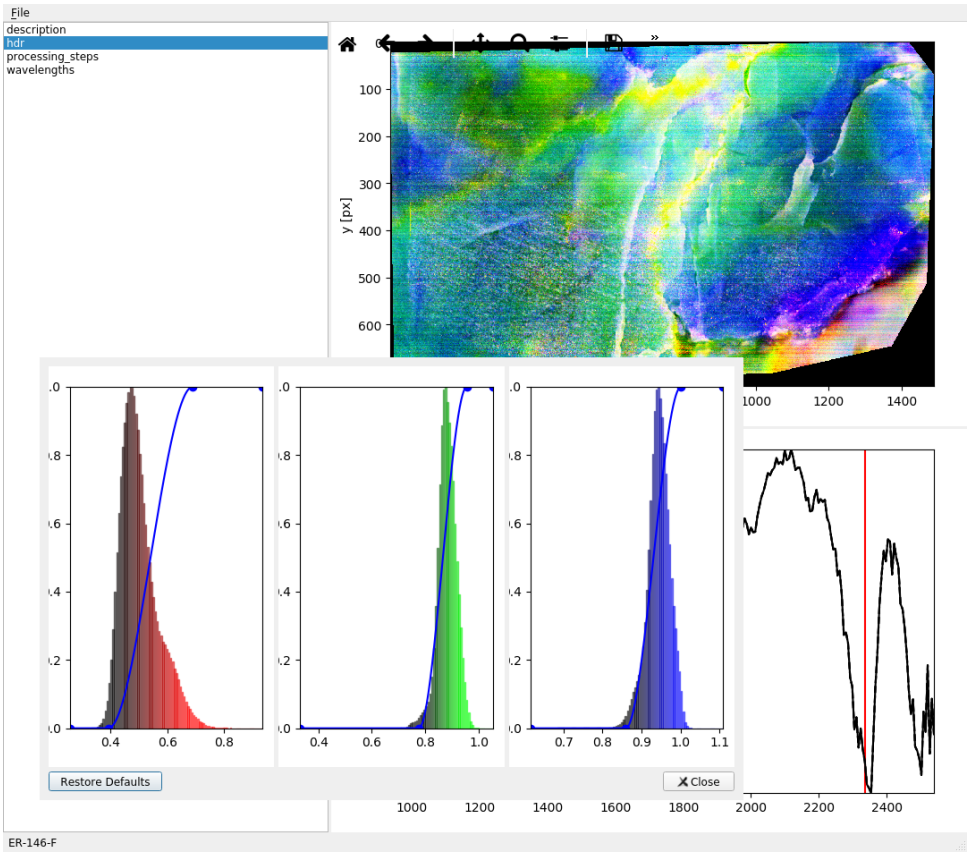

Figure 9: The data viewer window, while the colormaps are being edited.

It is possible to save the current view state, using the `File -> Export` menu entry, in order to re-use it later. Indeed, it may require some time to get the "perfect" visualization settings, so it can be useful to store it for later use (for example when discussing results with other people). A saved view state can be reloaded using `python3 -m hics.viewer <data filename> <viewstate filename>`.

References

[1] Laurent Fasnacht. mmappickle: Python 3 module to store memory-mapped numpy array in pickle format. *Journal of Open Source Software*, 3(26):651:1–651:2, June 2018. ISSN 2475-9066. doi: https://doi.org/10.21105/joss.00651. URL `http://joss.theoj.org/papers/10.21105/joss.00651`.