pdsspect Documentation

Release 0.1.0

PlanetaryPy

Jul 25, 2017

Contents:

1 pdsspect - A Python PDS Image Region of Interest Selection Tool **1**
- 1.1 Features . . . 1
- 1.2 Install . . . 1
- 1.3 Quick Tutorial . . . 2
- 1.4 Supported Instruments . . . 19

2 pdsspect **21**

3 pdsspect_image_set **23**

4 pdsspect_view **25**

5 pan_view **27**

6 pds_image_view_canvas **29**

7 selection **31**

8 transforms **33**

9 roi **35**

10 basic **37**

11 histogram **39**

12 roi_plot **41**

13 roi_histogram **43**

14 roi_line_plot **45**

15 set_wavelength **47**

16 Instrument Models **49**
- 16.1 Supported Instruments . . . 49
- 16.2 get_wavelength . . . 49
- 16.3 instrument . . . 49
- 16.4 mastcam . . . 50

16.5 pancam . . . 50
16.6 cassini_iss . . . 50

17 Contributing **51**
17.1 Types of Contributions . . . 51
17.2 Get Started! . . . 52
17.3 Pull Request Guidelines . . . 53
17.4 Tips . . . 53

18 Credits **55**
18.1 Development Lead . . . 55
18.2 Contributors . . . 55

19 History **57**
19.1 0.1.1 (“2017-08-21”) . . . 57
19.2 0.1.0 (“2017-08-20”) . . . 57

20 Indices and tables **59**

Python Module Index **61**

CHAPTER 1

pdsspect - A Python PDS Image Region of Interest Selection Tool

NOTE: This is Alpha quality software that is being actively developed, use at your own risk. This software is not produced by NASA.

- Free software: BSD license
- Documentation: https://pdsspect.readthedocs.org.

Features

- NASA PDS Image Viewer

NOTE: This is alpha quality software. It lacks many features and lacks support for many PDS image types. This software is not produced by NASA.

Install

On OS X you must first install the Qt UI toolkit using Homebrew (http://brew.sh/). After installing Homebrew, issue the following command:

```
brew install qt
```

Install Using Pip

Install pdsspect using pip:

```
pip install pdsspect
```

Then install your choice of pyside, pyqt4, or pyqt5

Install for Development

Create a new virtual environment, install the *pdsspect* module with git, and setup the PySide environment. You must install either PySide, PyQt5, or PyQt4 as well (recommend PyQt5):

```
Make a clone of ``pdsspect`` and change to main directory. We recommend
making a virtual environment for to install ``pdsspect`` in.

pip install -e .
pip install PyQt5
```

Now you should be able to run the *pdsspect* program.

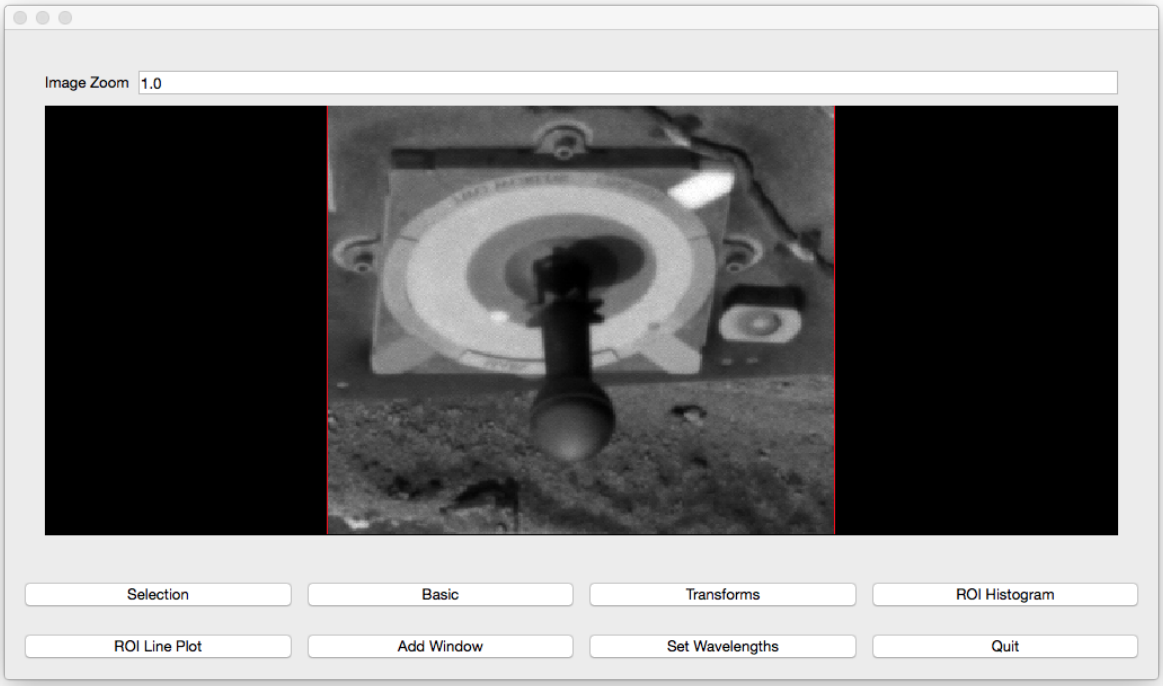
This works on Linux as well (Ubuntu 14.04).

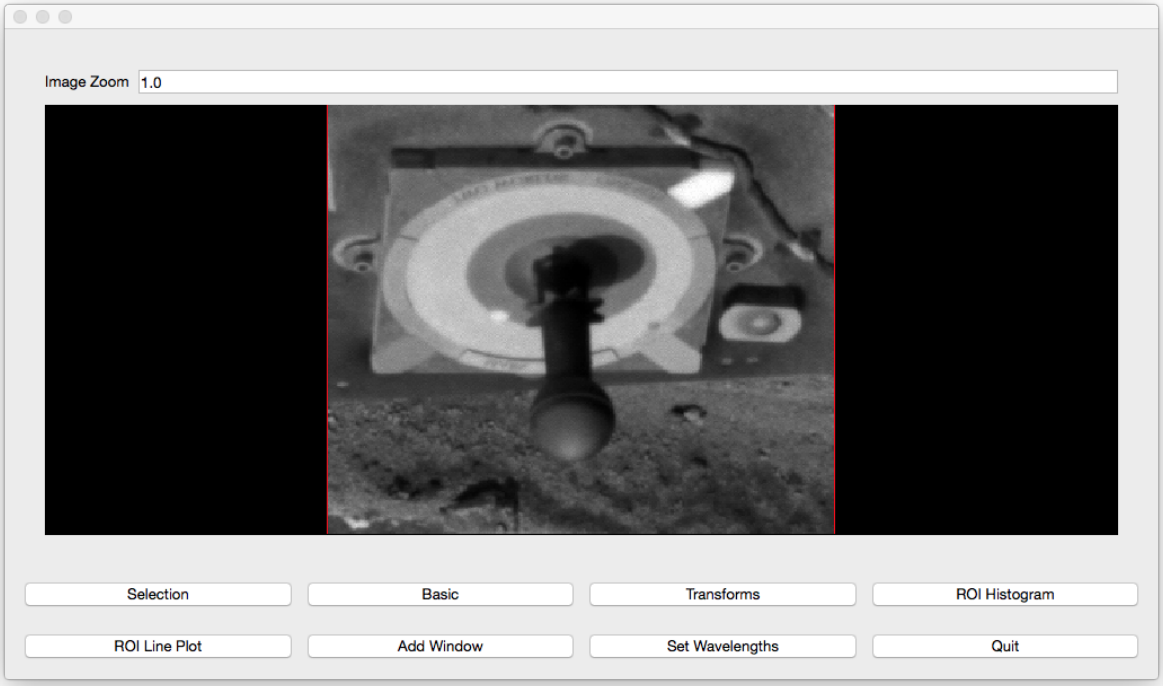
Quick Tutorial

Open an image in the command line:

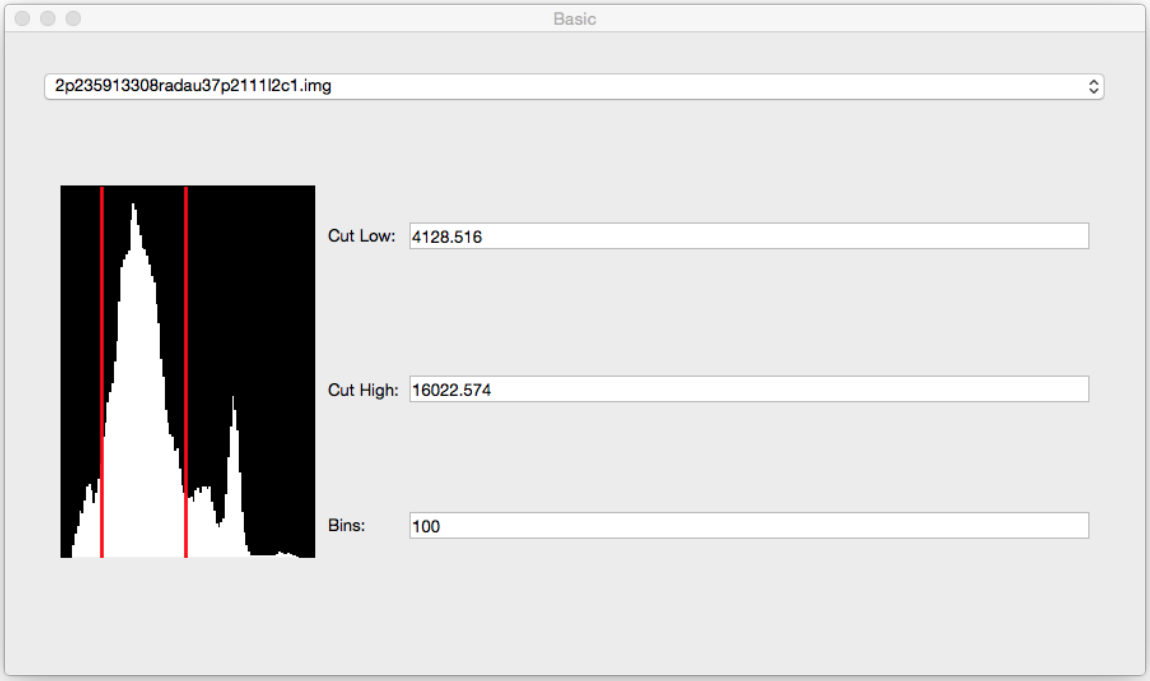
```
pdsspect tests/mission_data/2m132591087cfd1800p2977m2f1.img
```

This will open the default window:



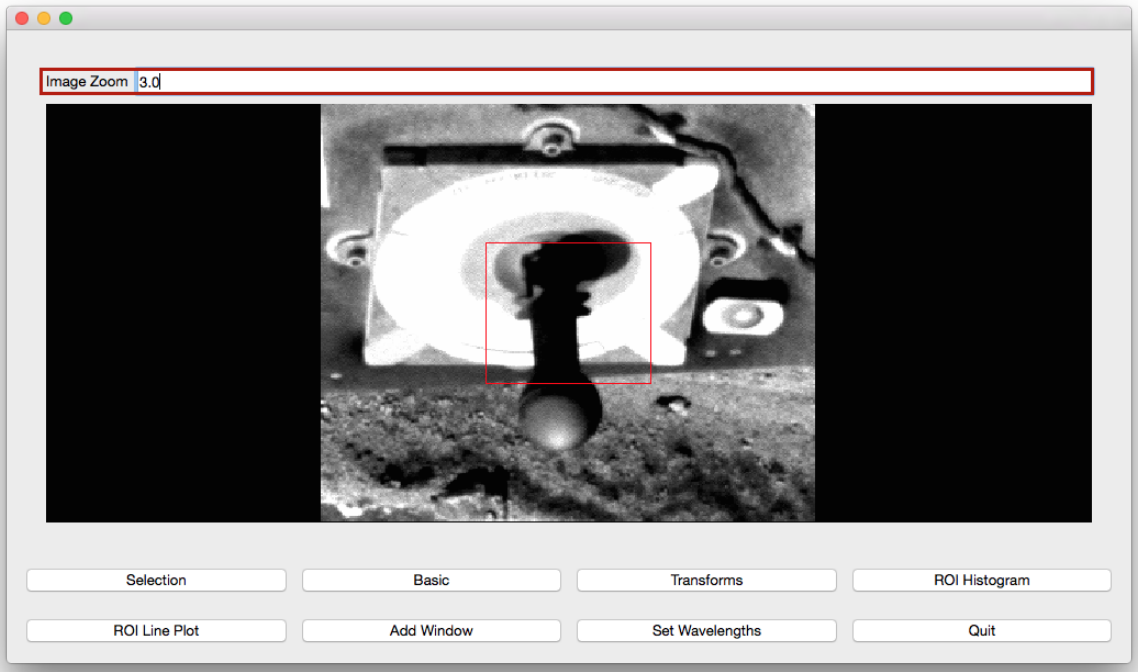


The bottom left window is considered the main window. In this window, the user can adjust the position of the pan and open other windows. The bottom right window is the `basic` window. Pressing the `basic` button will open this window if closed. However, it starts out open. In this window, the user can change the image in the views and adjust the cut levels by either moving the red lines or typing in the numbers in the cut boxes:



The top window is the `pan window` which displays the data in the main window's red box. The main function of this window is to make Region of Interest (ROI) selections.

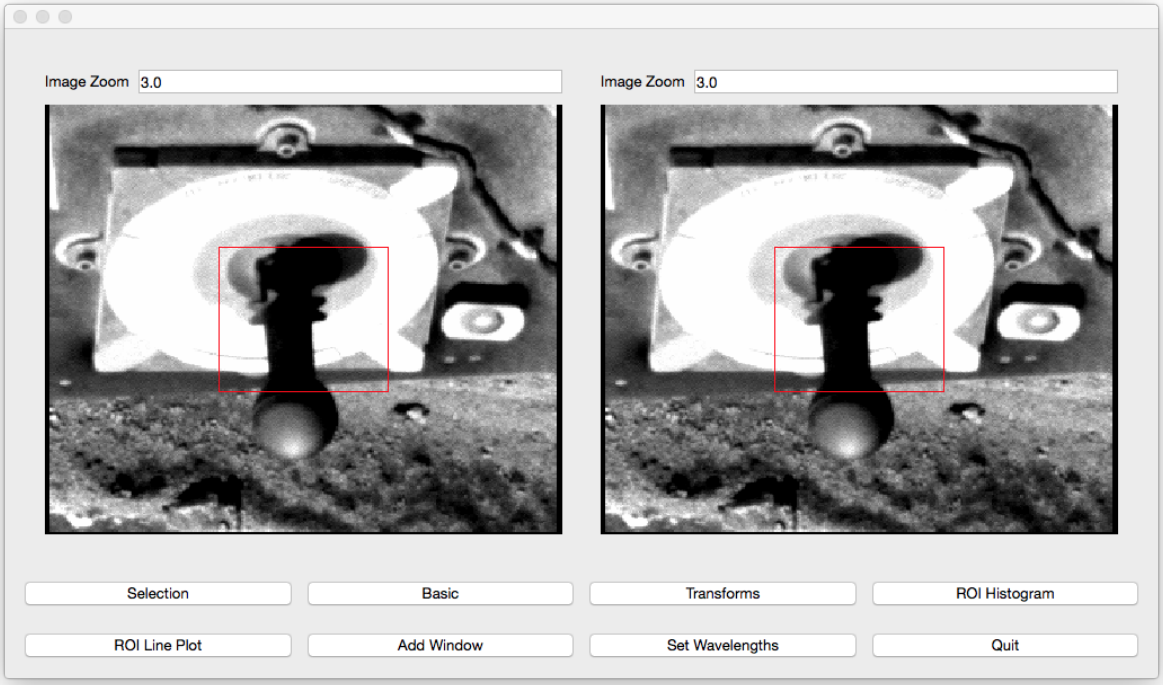
In the `zoom` box in the main window, the user can change the size of the box and the data in the pan view:



the mouse wheel can also be used to change the zoom. Rolling the wheel foward and backwards will adjust the zoom amount by `+1` or `-1` respectively. The user can adjust the position of the box by clicking in the main window where the center of the pan should be. Using the arrow keys will also adjust the position of the box by `1` in the direction of

the arrow key.

Clicking the `Add Window` button will open another view. This view will have the same image, cut levels, and zoom by default.



If the image's are the same, chaning the cut levels on one image will automatically change the cut levels on another image. However, one can change the zoom on one view without changing the zoom another view.

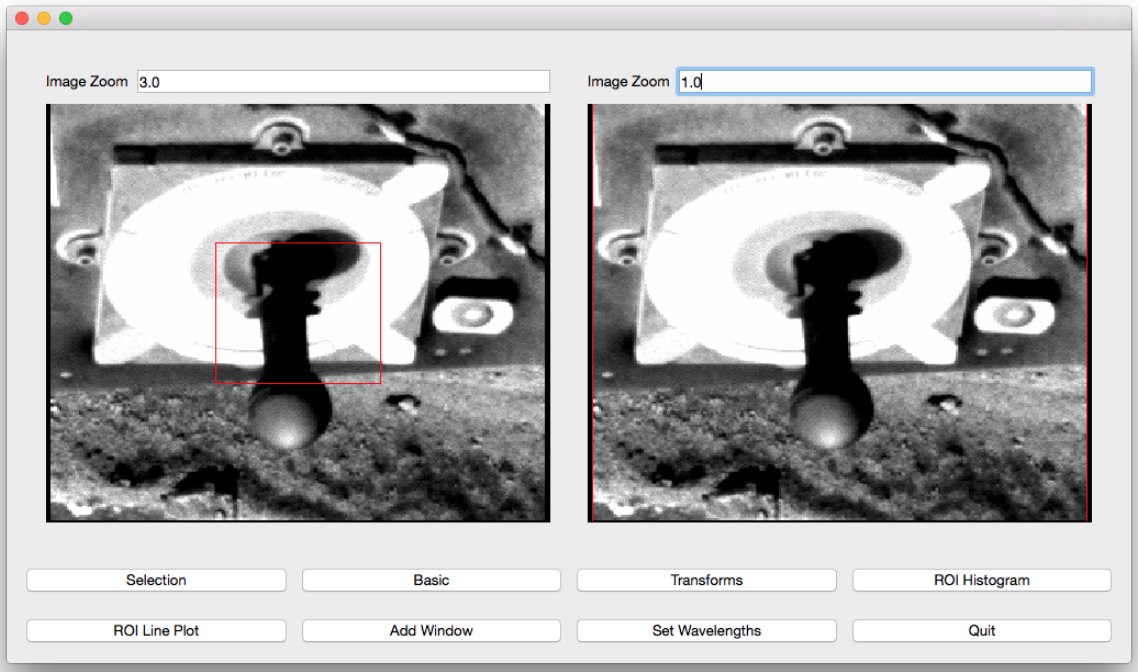
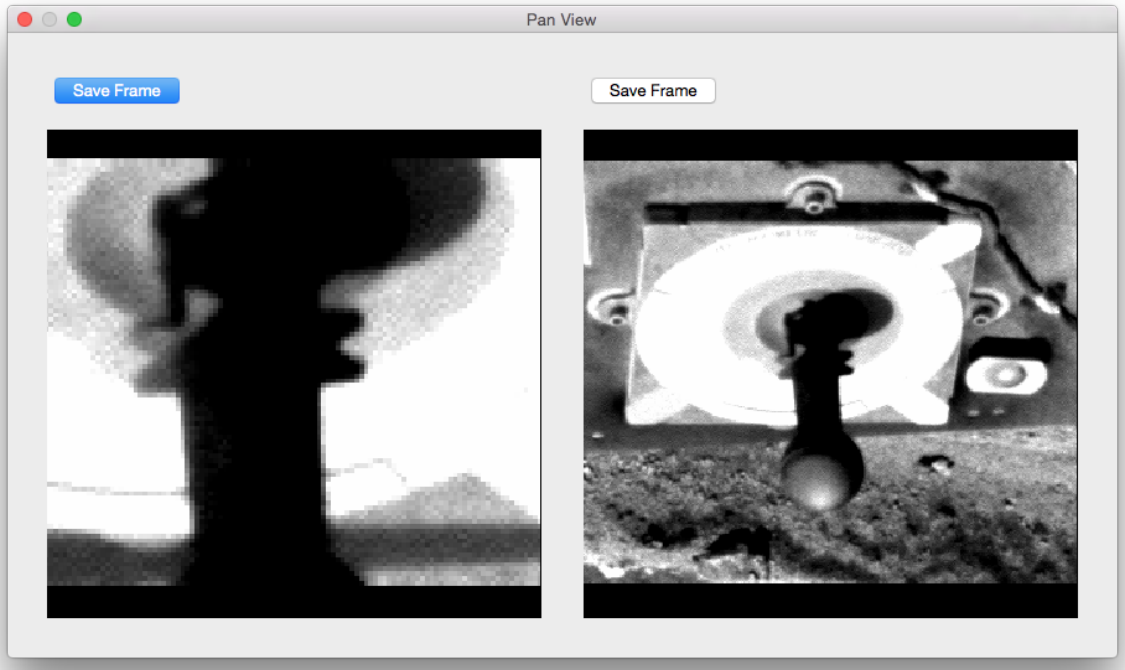
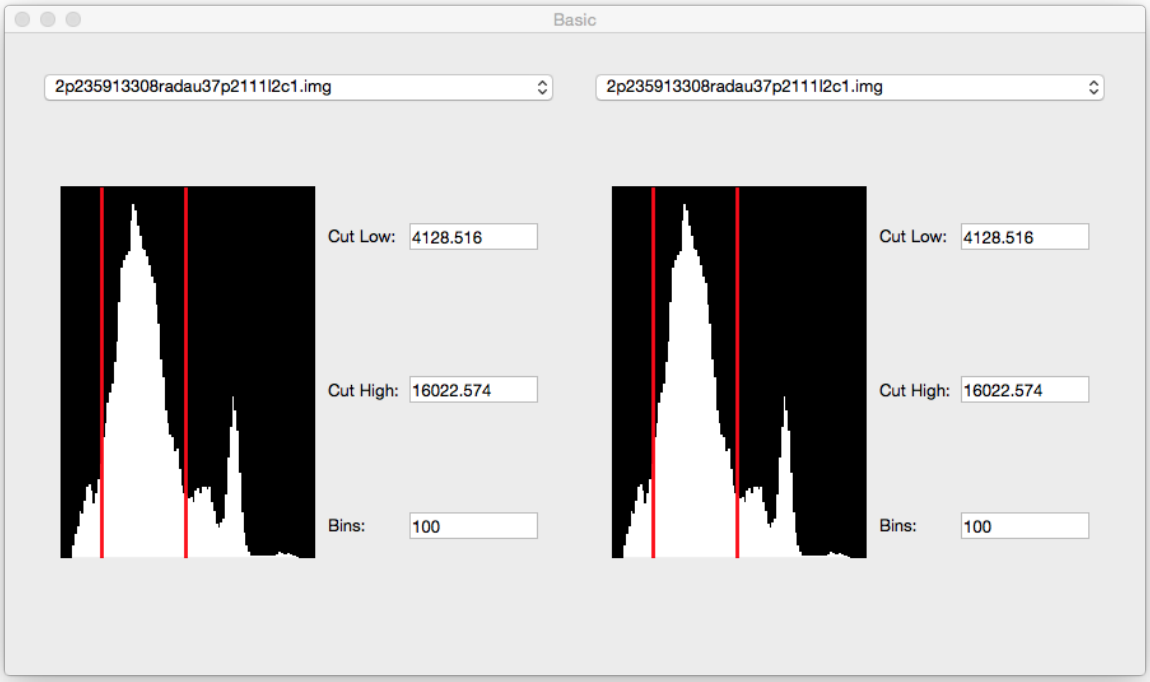
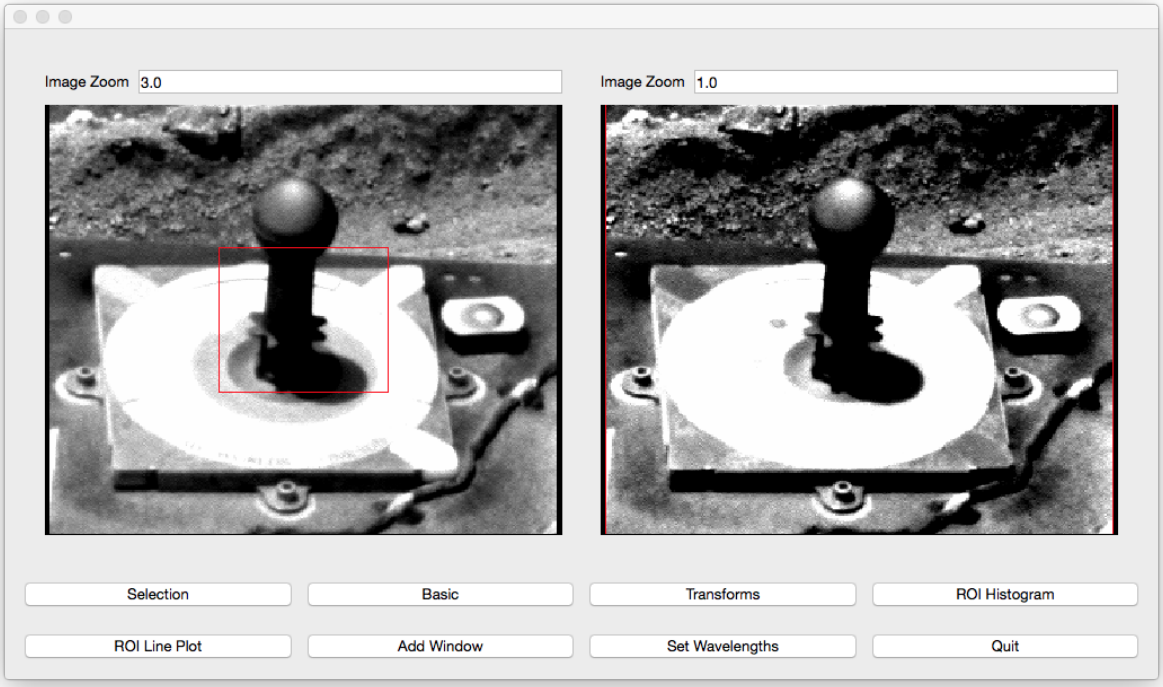


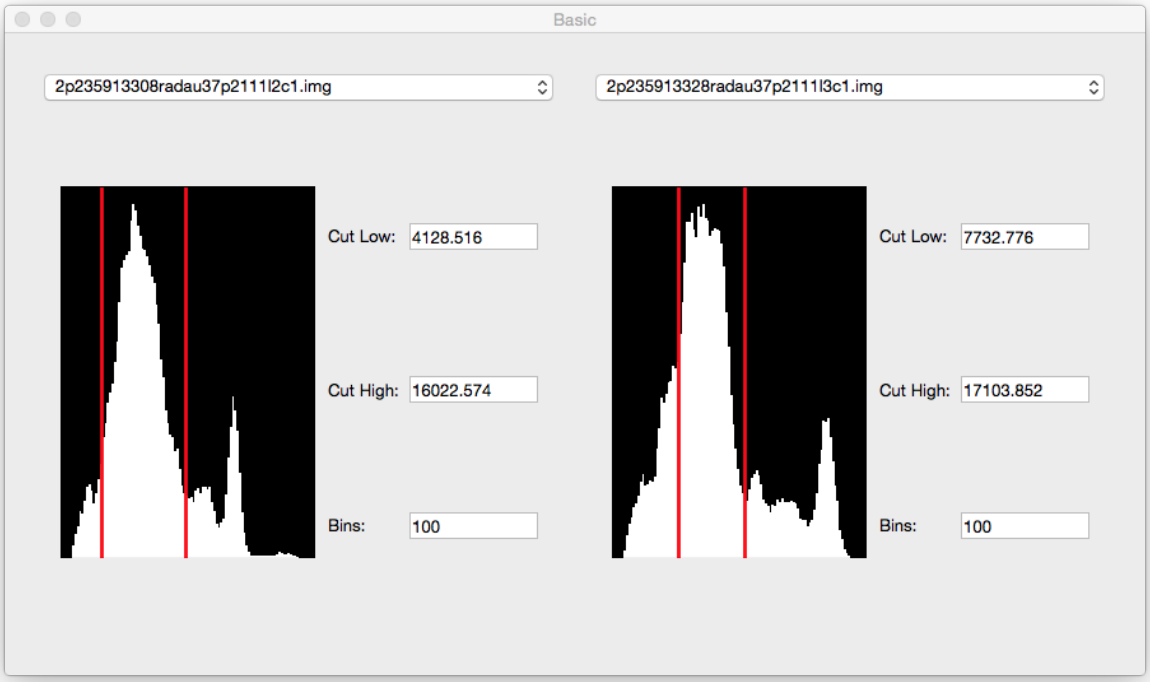
Image Zoom 3.0
Image Zoom 1.0
Selection
Basic
Transforms
ROI Histogram
ROI Line Plot
Add Window
Set Wavelengths
Quit


Pan View
Save Frame
Save Frame

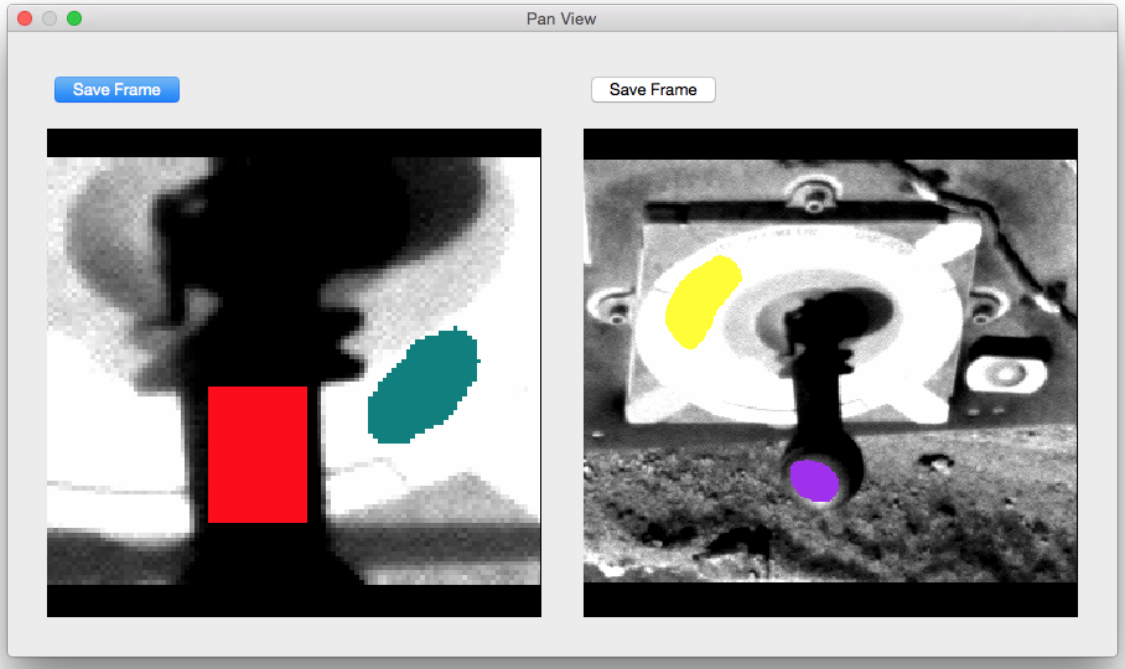


When the images are different, adjusting the cut levels on one image will only change the cut levels on that image:

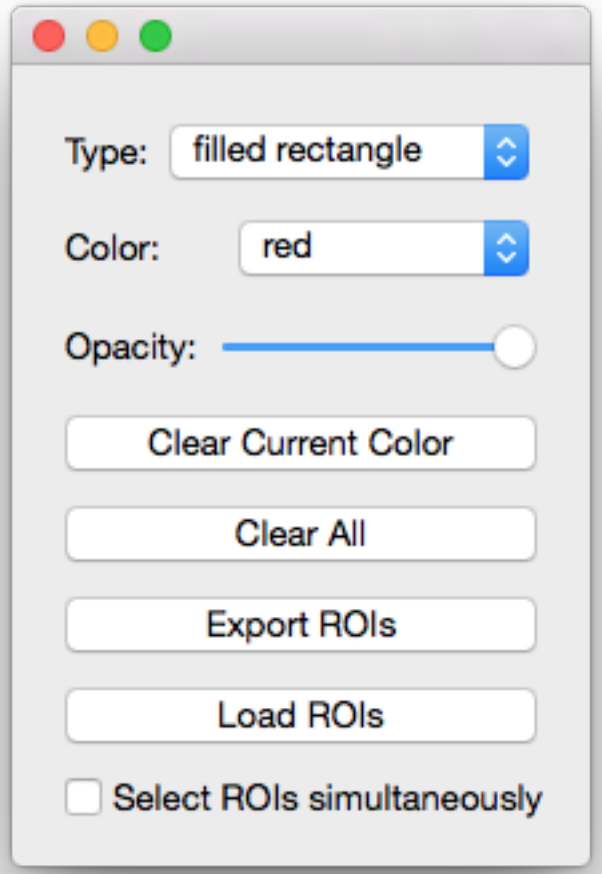




The user can create separate ROIs in each view:



Clicking the `Selection` button will open the `Selections Window`:



In this window, the user can choose the color of the ROI. The possible choices for colors: `red`, `brown`, `lightblue`, `lightcyan`, `darkgreen`, `yellow`, `pink`, `teal`, `goldenrod`, `sienna`, `darkblue`, `crimson`, `maroon`, `purple`, and `eraser (black)`. The selection type can be changed in this window as well. The possible types are `filled rectangle`, `filled polygon`, and `pencil` (single points).

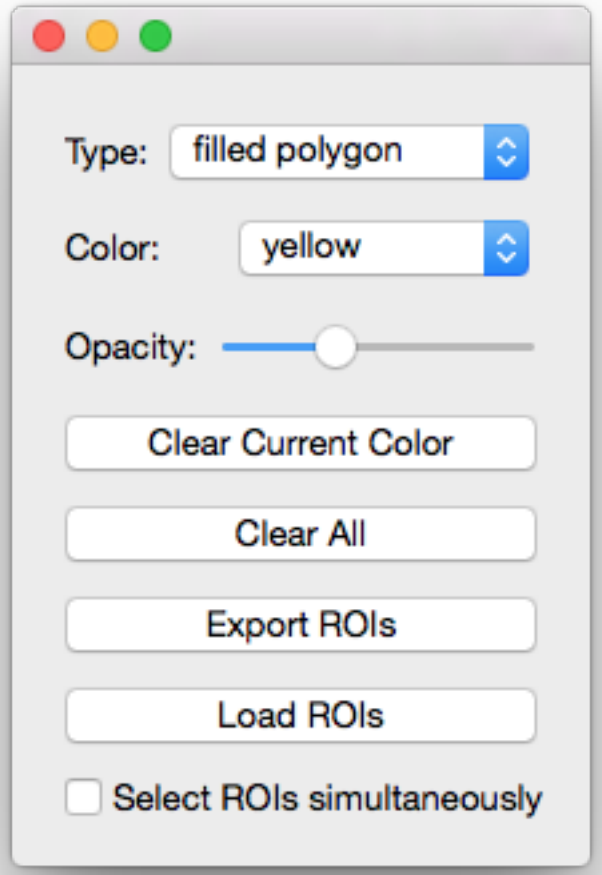
Furthermore, in this window, the user can clear the current color or clear all ROIs. Most importantly, the user can export ROIs to `.npz` files. These files contain boolean masks and of the images and a list of files open at the time of export. The ROIs in the 2nd, 3rd, 4th, etc. views will be labeled as `color#view` while the ROIs in the first view is still labeled as `color`. For example, to see the data in an example file `example.npz`, use numpy load method to view and utilize data.

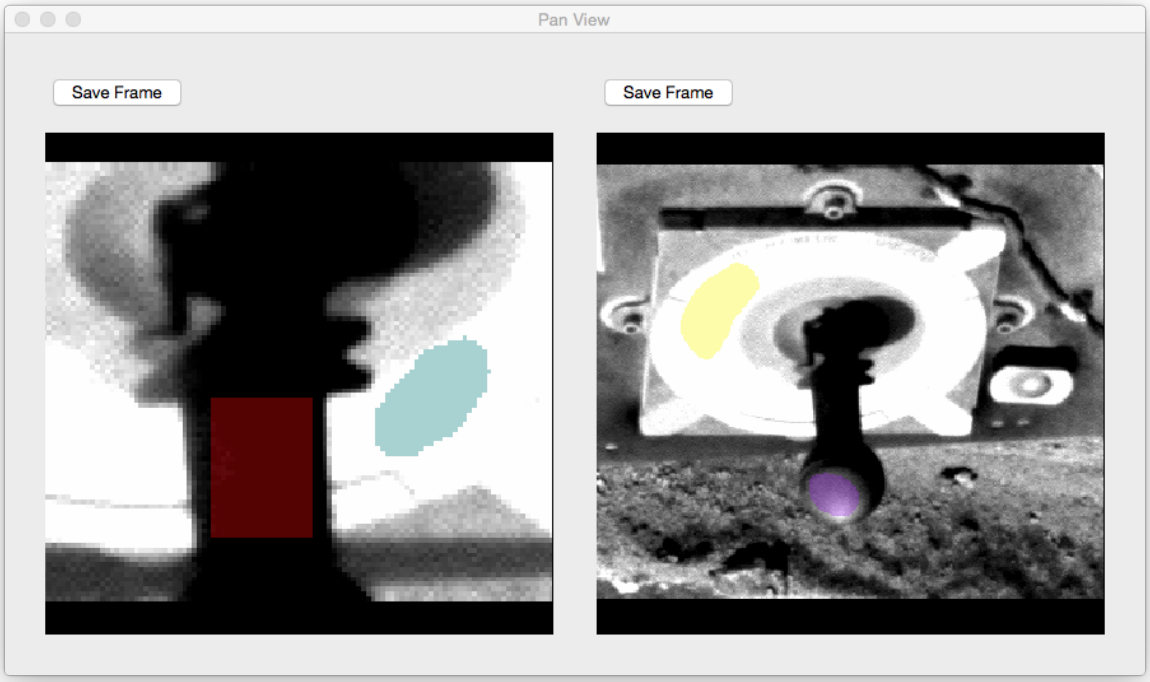
```
>>> import numpy as np
>>> selections = np.load('example.npz')
>>> selections['red'][114:118, 142:146]
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]], dtype=bool)
```

```
>>> selections['purple2'][48:52, 146:150]
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]], dtype=bool)
```

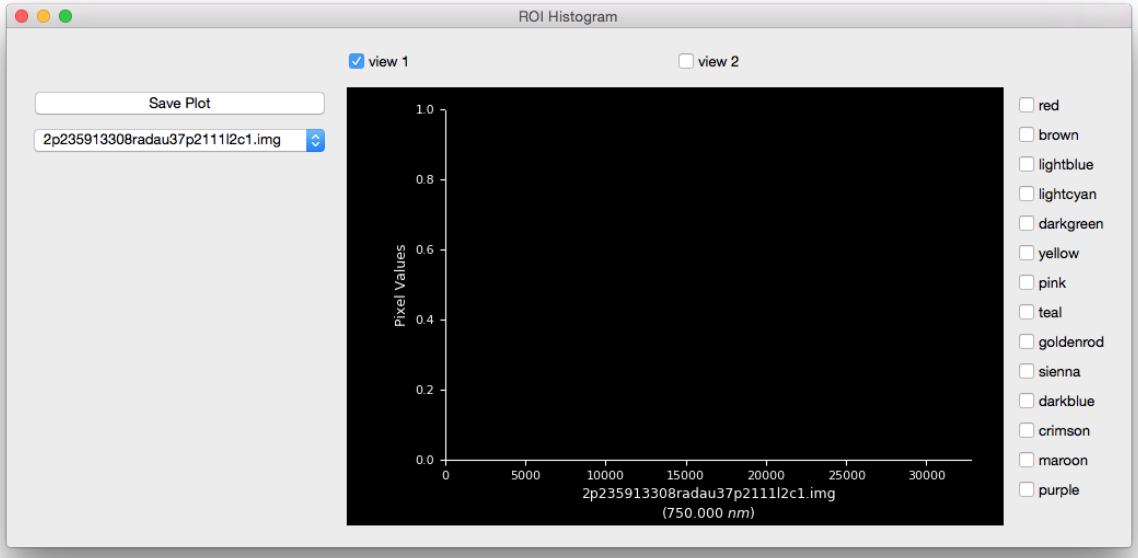
The user can also import ROI selections. However the images that are open must be in the `files` list in the `.npz` file.

Changing the opacity in the Selecitons window will change the opacity on all the ROIs in every view:

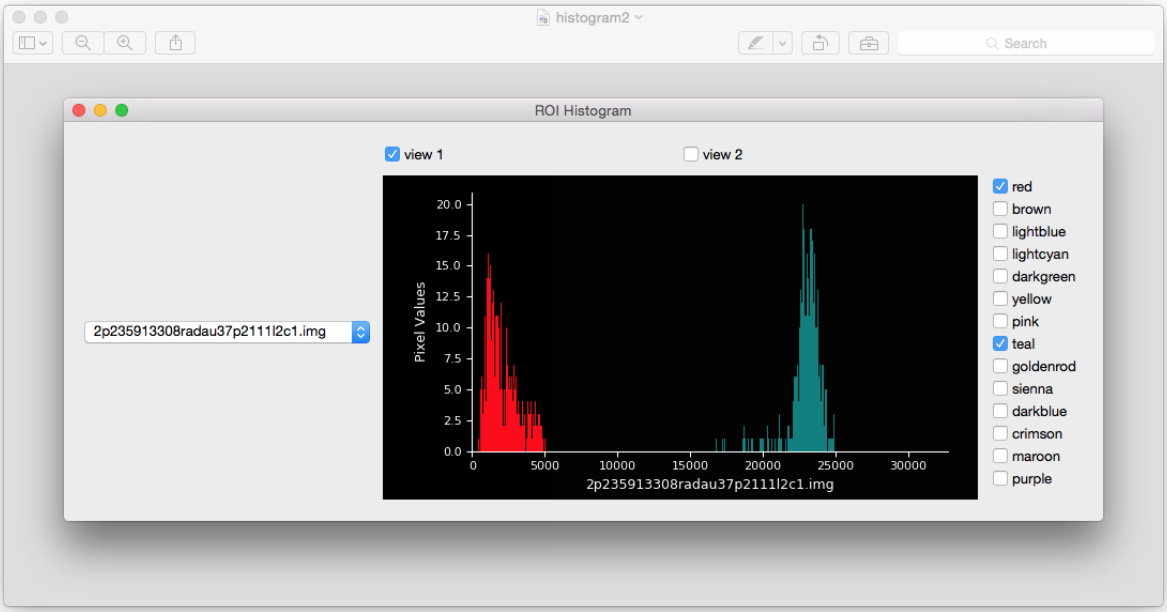




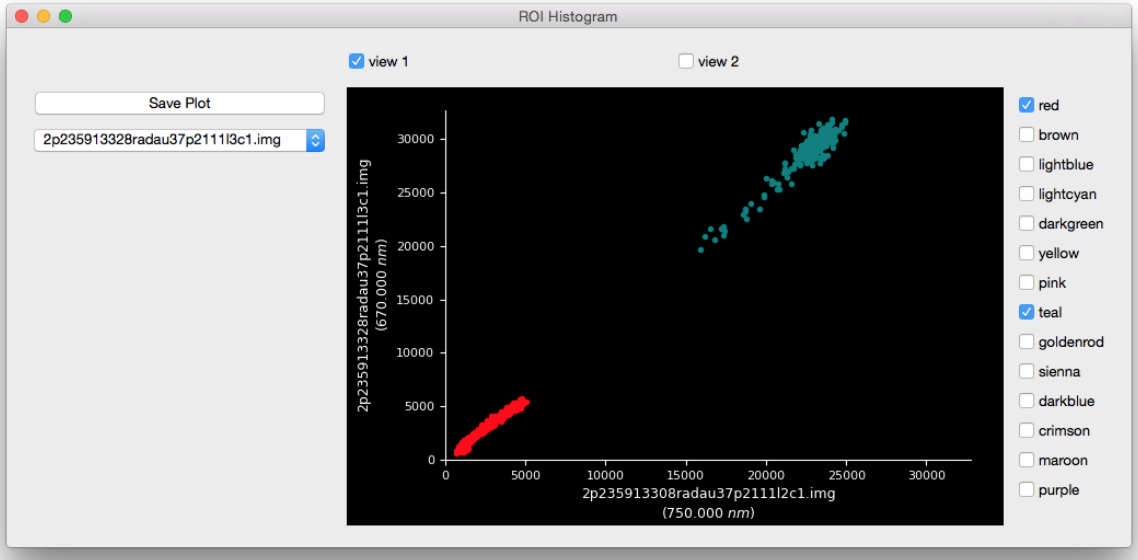
You can view the data within the ROIs with the `ROI Histogram` window. Open the window by pressing the `ROI Histogram` button in the main viewer.



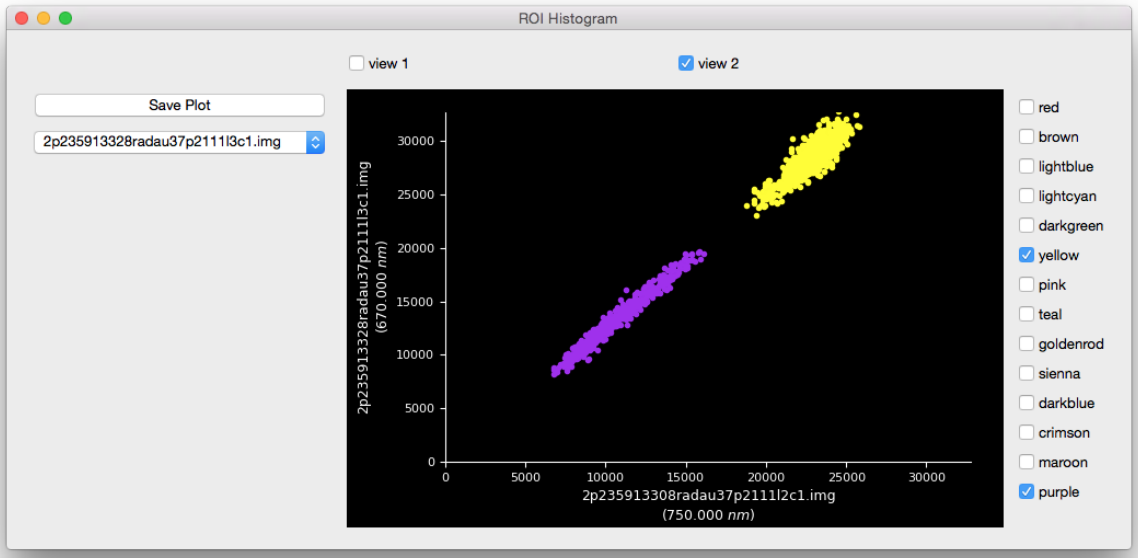
Display the data in the ROI by color by checking the checkbox next to the color. When the image in the menu and the current image in the checked view are the same, the plot will be a histogram:



When the menu and the current image are different, the plot will compare the data:

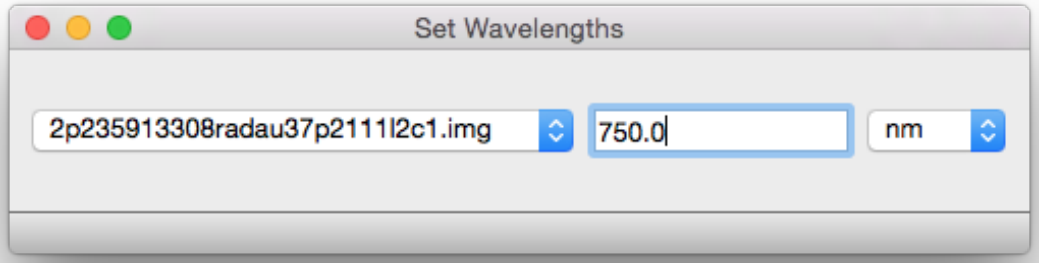


To view the data in the other view, check the view number:

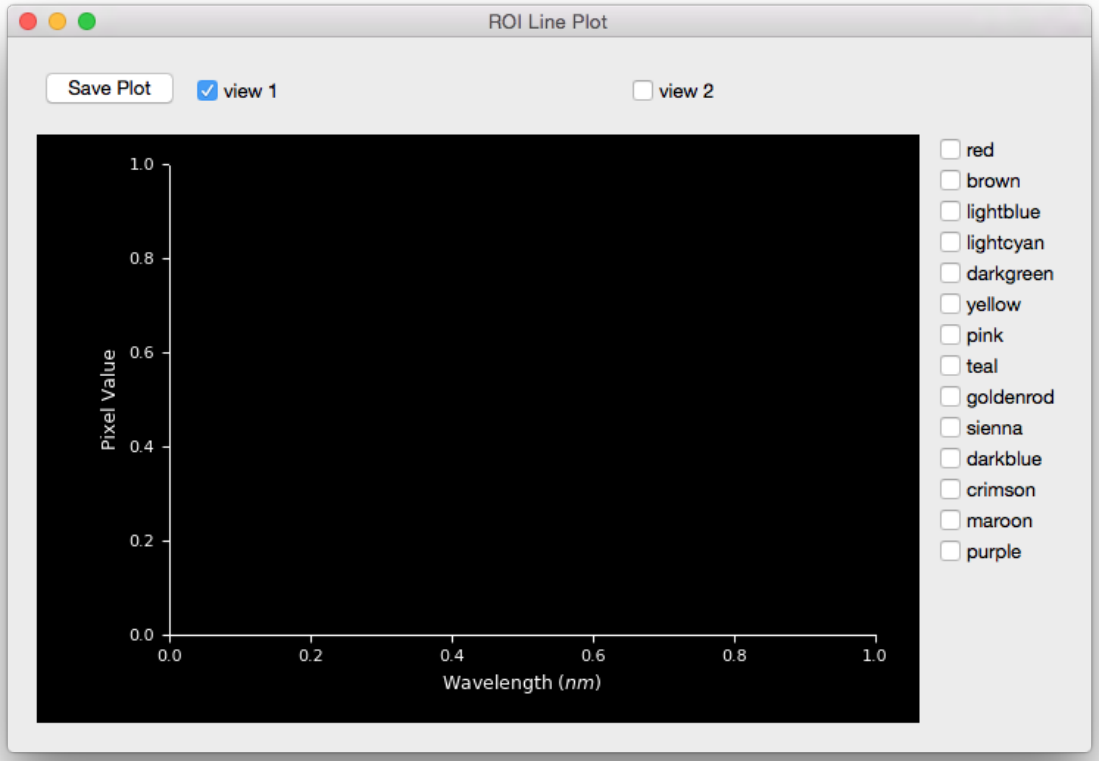


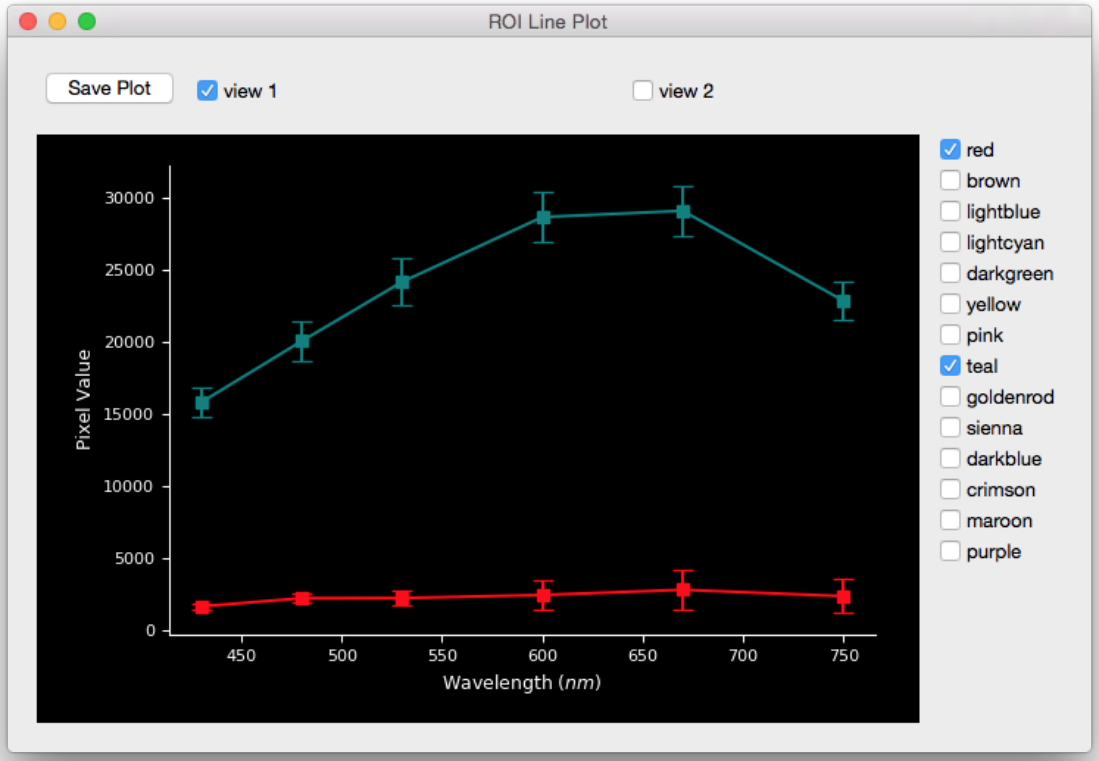
Overlay ROIs by checking other boxes. The order (depth) of the histogram data will be in the order that the user checks the boxes (i.e., checking red and then purple will result in purple overlaying the red).

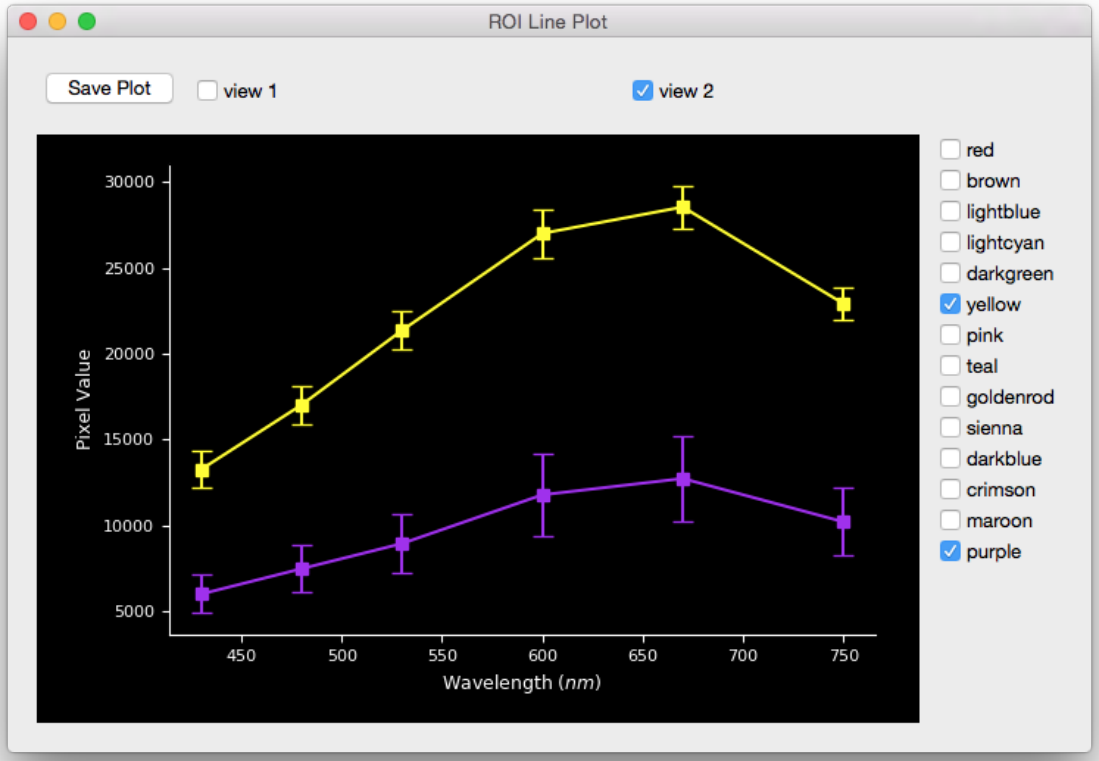
To perform multispectral analysis use `ROI Line Plot`. If analyzing images that are not *fully supported (see here for list of instruments supported by pdsspect)* the user must manually input the image wavelength with `Set Wavelength` widget:



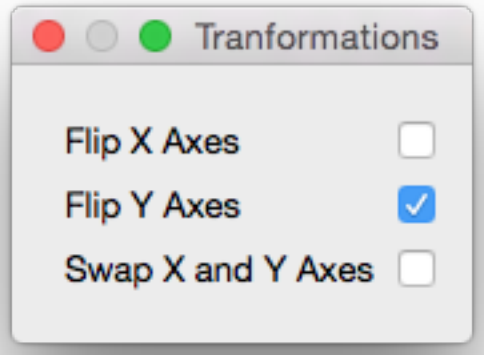
`ROI Line Plot` works similar to that of the histogram plot except it will compare each image with an associated wavelength.

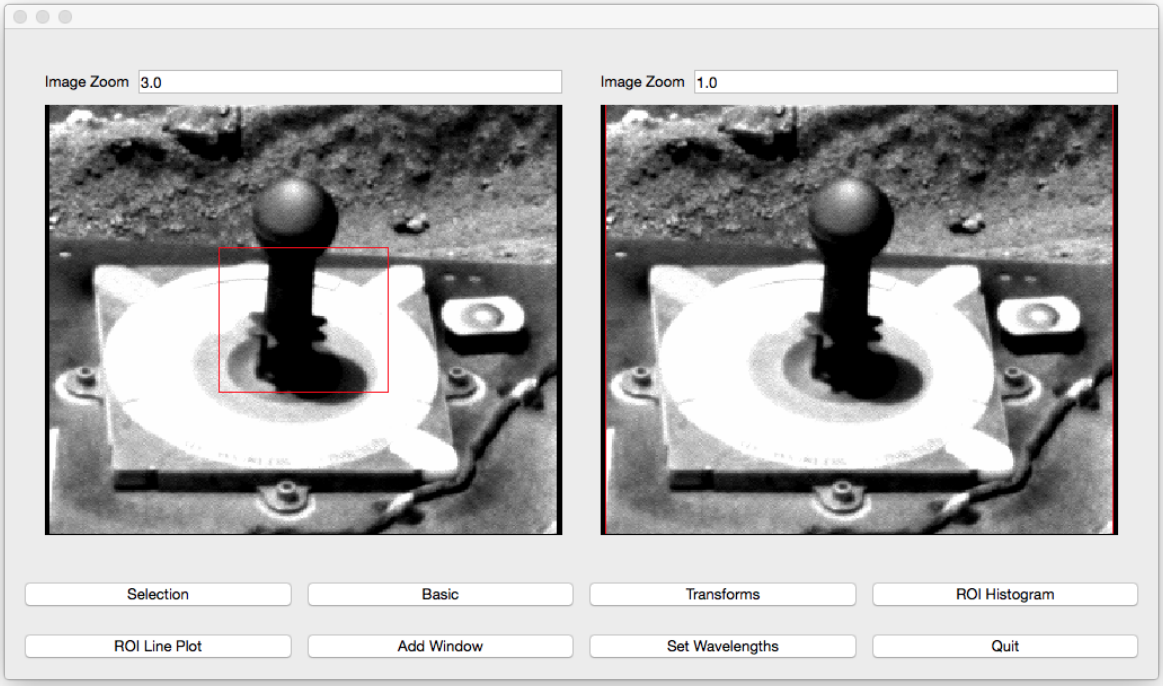

ROI Line Plot
Save Plot
view 1
view 2
red
brown
lightblue
lightcyan
darkgreen
yellow
pink
teal
goldenrod
sienna
darkblue
crimson
maroon
purple
Pixel Value
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Wavelength (nm)

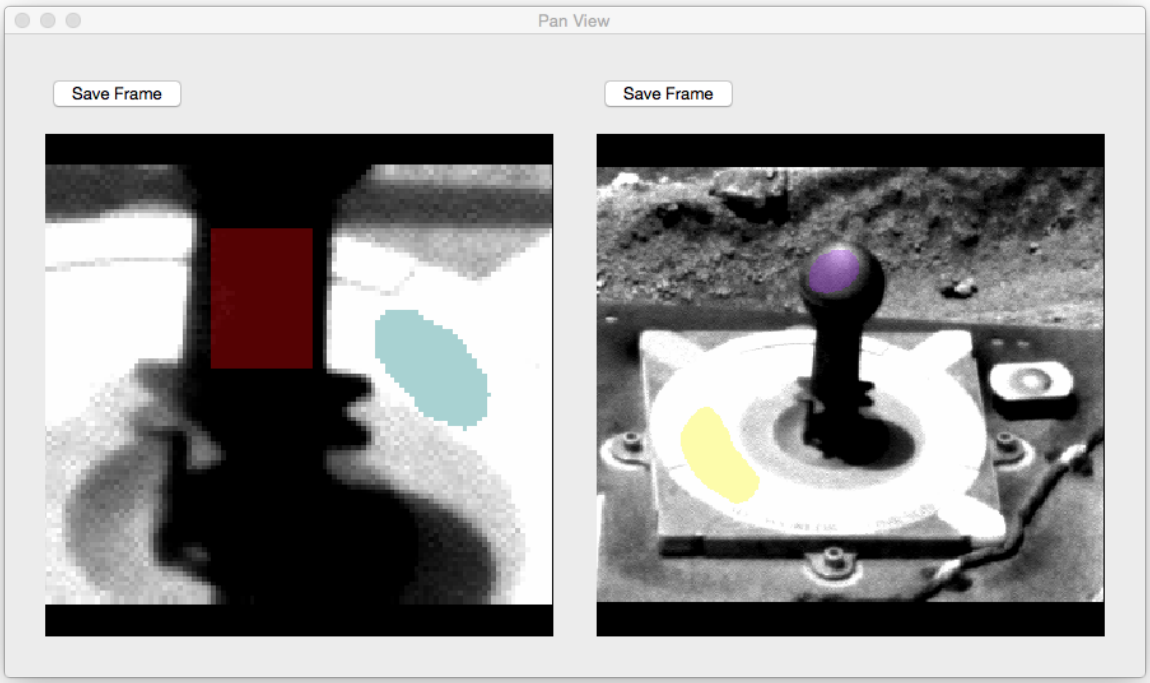

ROI Line Plot
Save Plot
view 1
view 2
red
brown
lightblue
lightcyan
darkgreen
yellow
pink
teal
goldenrod
sienna
darkblue
crimson
maroon
purple
Pixel Value
30000
25000
20000
15000
10000
5000
0
450
500
550
600
650
700
750
Wavelength (nm)



The user can flip the image over different axis with the Transforms window. The transformation will apply to each image in all the views:







Note that when opening multiple images at the same time, it is best practice that they are all the same shape. Otherwise the images will have the smallest common shape and not look as expected (i.e., If when loading two images where one image has a shape of `(63, 36)` and the other image has a shape of `(24, 42)`, the displayed shape will be `(24, 36)`. This will cause the first image to have the right side cut off and the second image to have the top cut off). This

is done so all ROIs created can apply to the entire list of images. To avoid this behavior, either only open images that have the same shape or open images one at a time.

Images In Example

- 2p235913308radau37p2111l2c1.img
- 2p235913328radau37p2111l3c1.img
- 2p235913348radau37p2111l4c1.img
- 2p235913368radau37p2111l5c1.img
- 2p235913399radau37p2111l6c1.img
- 2p235913431radau37p2111l7c1.img

Supported Instruments

- **MER**
  - Pancam
- **MSL**
  - Mastcam
- **Cassini**
  - Imaging Science Subsystem (ISS)

Adding More Instruments

We welcome anyone to create more models for instruments that are not yet supported. Please follow the Pull Request guide to make sure your model is compatible with the rest of the models. See Pull Request #20 as an example.

Pull Request Checklist

Please include the following checklist in your PR so we know you have completed each step:

```
- [ ] Created model as subclass of [InstrumentBase](https://github.com/planetarypy/
↪pdsspect/blob/master/instrument_models/instrument.py#L7)
- [ ] Added model to [get_wavelength](https://github.com/planetarypy/pdsspect/blob/
↪master/instrument_models/get_wavelength.py)
- [ ] Documented Model
- [ ] Tested Model
- [ ] Added model to [test_get_wavelength](https://github.com/planetarypy/pdsspect/
↪blob/master/tests/test_get_wavelength.py) test
- [ ] Added instrument to supported_instruments.rst list
```

Style

- Set PR label to `Instrument Model`

- If an issue was created, please include `Fixes #<issue_number>` at the top of the PR to automatically close the issue
- Please include a link to any documents used to find the filter wavelength. Follow the example for Mastcam and/or Pancam
- When documenting your model, use numpy docs.See these examples. Also add to instrument_models.rst following the format of the other models
- For tests, if one of the core mission_data images is not from your instrument, create a minimal label in the tests__init__.py. You must test the model itself and test that it works in test_get_wavelength
- Add the mission and instrument to the `supported_instruments.rst` file following the set format.

CHAPTER 2

pdsspect

CHAPTER 3

pdsspect_image_set

CHAPTER 4

pdsspect_view

CHAPTER 5

pan_view

CHAPTER 6

pds_image_view_canvas

CHAPTER 7

selection

CHAPTER 8

transforms

CHAPTER 9

roi

CHAPTER 10

basic

CHAPTER 11

histogram

CHAPTER 12

roi_plot

CHAPTER 13

roi_histogram

CHAPTER 14

roi_line_plot

CHAPTER 15

set_wavelength

CHAPTER 16

Instrument Models

Supported Instruments

- **MER**
  - Pancam
- **MSL**
  - Mastcam
- **Cassini**
  - Imaging Science Subsystem (ISS)

get_wavelength

instrument

Provide base class for all instrument models

class instrument_models.instrument.**InstrumentBase**(*label*)

Abstract Base Class for all instrument models

Parameters **label** (pvl.PVLModule) – Image's label

label

pvl.PVLModule – Image's label

get_wavelength(*unit*, **args*, ***kwargs*)

Abstract method to get the image's wavelength

Parameters **unit** (str [nm]) – The wavelength unit. Best practice for unit to exist in pdsspect.pdsspect_image_set.ImageStamp.accepted_units

Returns **wavelength** – The image's filter wavelength

Return type `float`

mastcam

pancam

cassini_iss

CHAPTER 17

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. To understand and better read the code, you should have an understanding of the Model-View-Controller (MVC) architecture.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/planetarypy/pdsspect/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

pdsspect could always use more documentation, whether as part of the official pdsspect docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/planetarypy/pdsspect/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *pdsspect* for local development.

1. Fork the *pdsspect* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pdsspect.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pdsspect
$ cd pdsspect/
$ pip install -r requirements.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test
$ make test-all
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/planetarypy/pdsspect/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
py.test tests/
```

CHAPTER 18

Credits

Development Lead

- PlanetaryPy Developers <contact@planetarypy.com>

Contributors

- Perry Vargas <perrybvargas@gmail.com>
- Austin Godber <godber@uberhip.com>

CHAPTER 19

History

0.1.1 (“2017-08-21”)

- Make compatible to be opened by other programs like pystamps

0.1.0 (“2017-08-20”)

- First release on PyPi

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

instrument_models.instrument, 49

Index

G

get_wavelength() (instrument_models.instrument.InstrumentBase method), 49

I

instrument_models.instrument (module), 49
InstrumentBase (class in instrument_models.instrument), 49

L

label (instrument_models.instrument.InstrumentBase attribute), 49