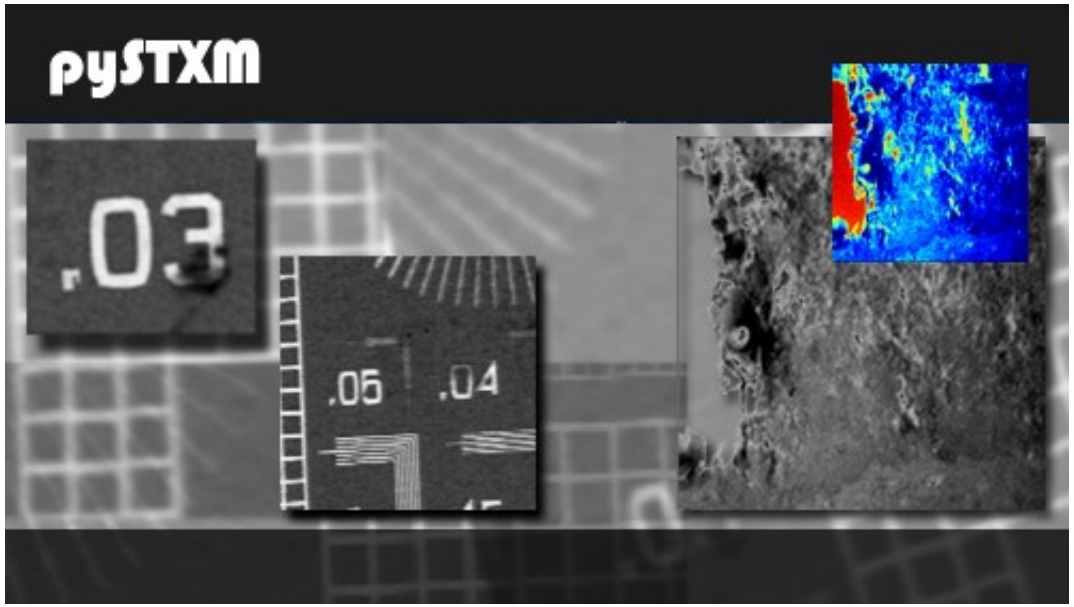
Canadian Light Source | Centre canadien de rayonnement synchrotron

pySTXM

Scanning microscope data collection with Python , Qt and BlueSky



Epics Collaboration Meeting July 2021

Russ Berg: russ.berg@lightsource.ca

Canadian Light Source | Centre canadien de rayonnement synchrotron

pySTXM

Scanning microscope data collection with Python , Qt and BlueSky

The goal of this talk is to present some ideas on facilitating user “experience and efficiency” that is not 100% dependent on the experience level of the user on an Epics beamline end station, hopefully some of these idea’s translate and are of some value to data collection software others are working.

I will cover:

- What is STXM?
- High level overview of the software architecture
- UI idea’s that were implemented focusing on streamlining the “User Experience”

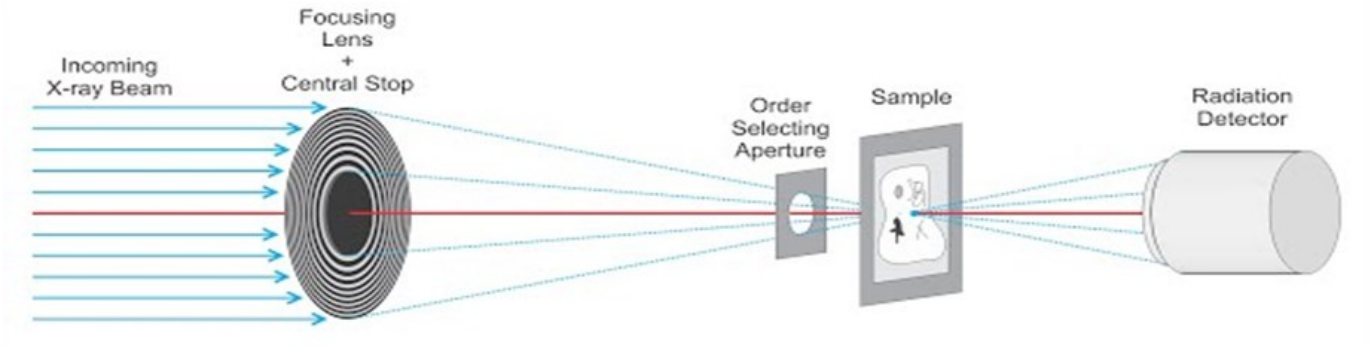
what is STXM?

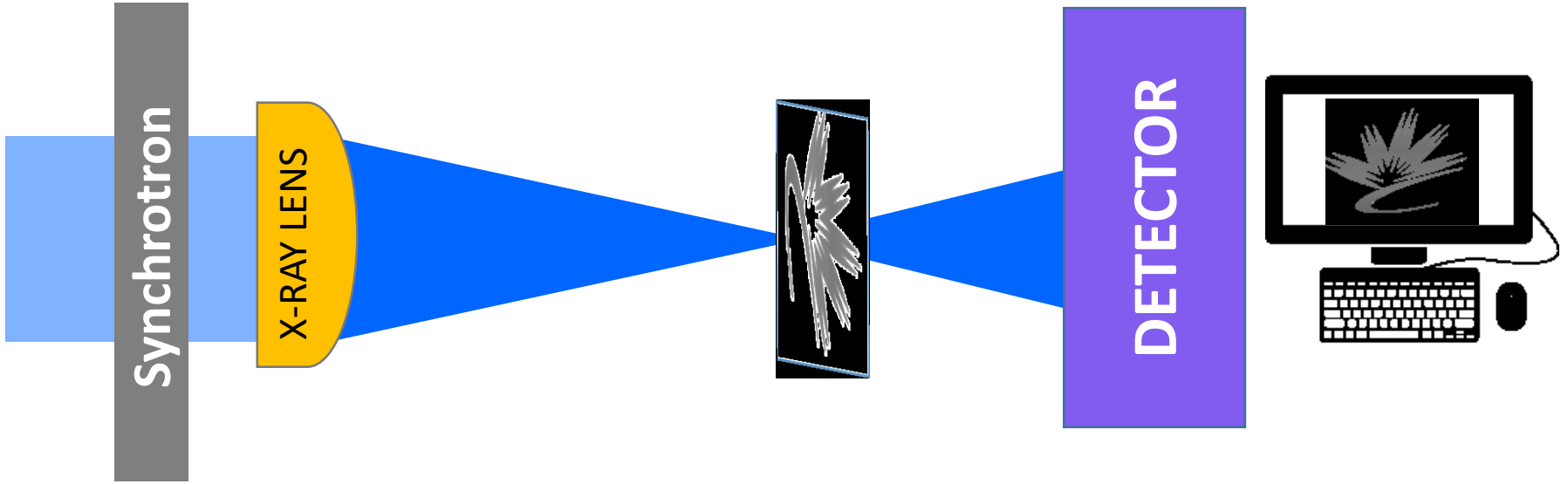
Scanning

Transmission

X-ray

Microscopy

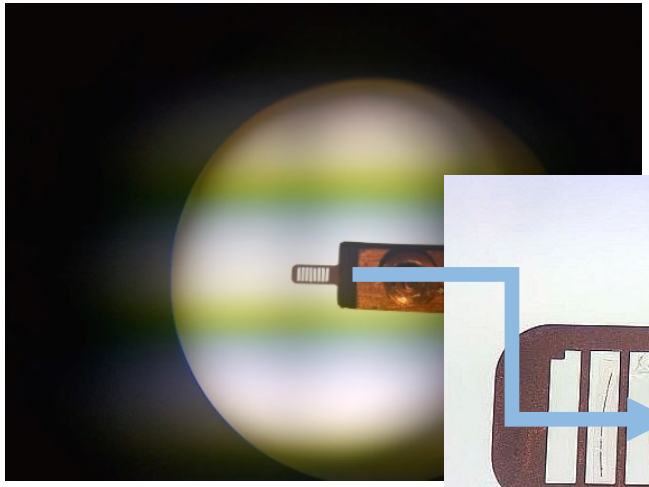


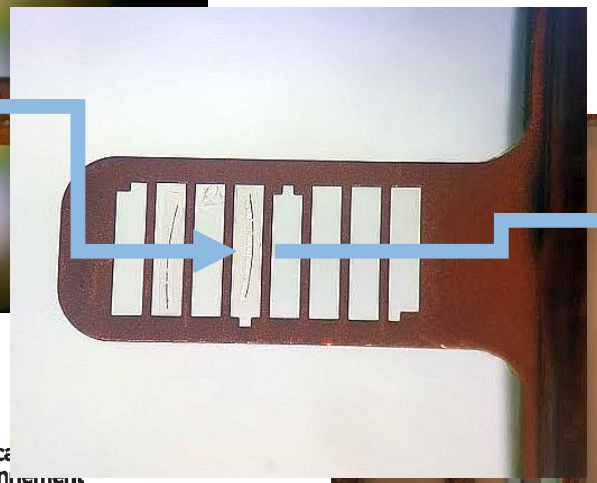


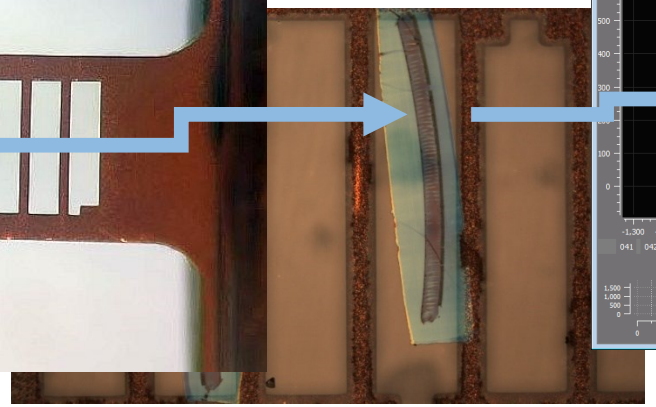
What is STXM?

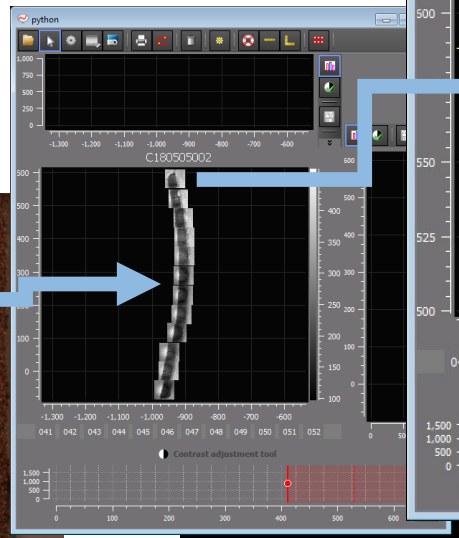
Scanning
Transmission
X-ray
Microscopy

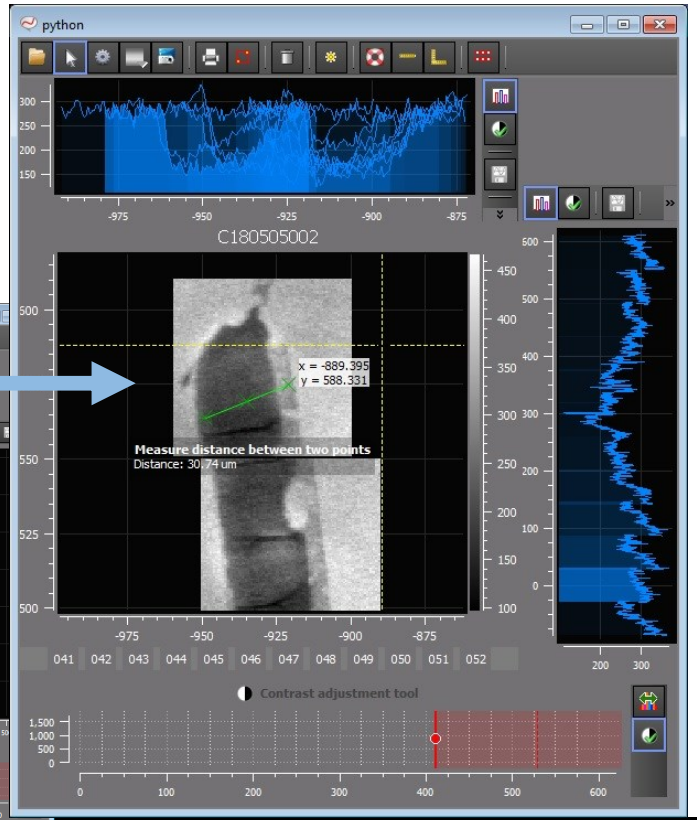
It's a microscope





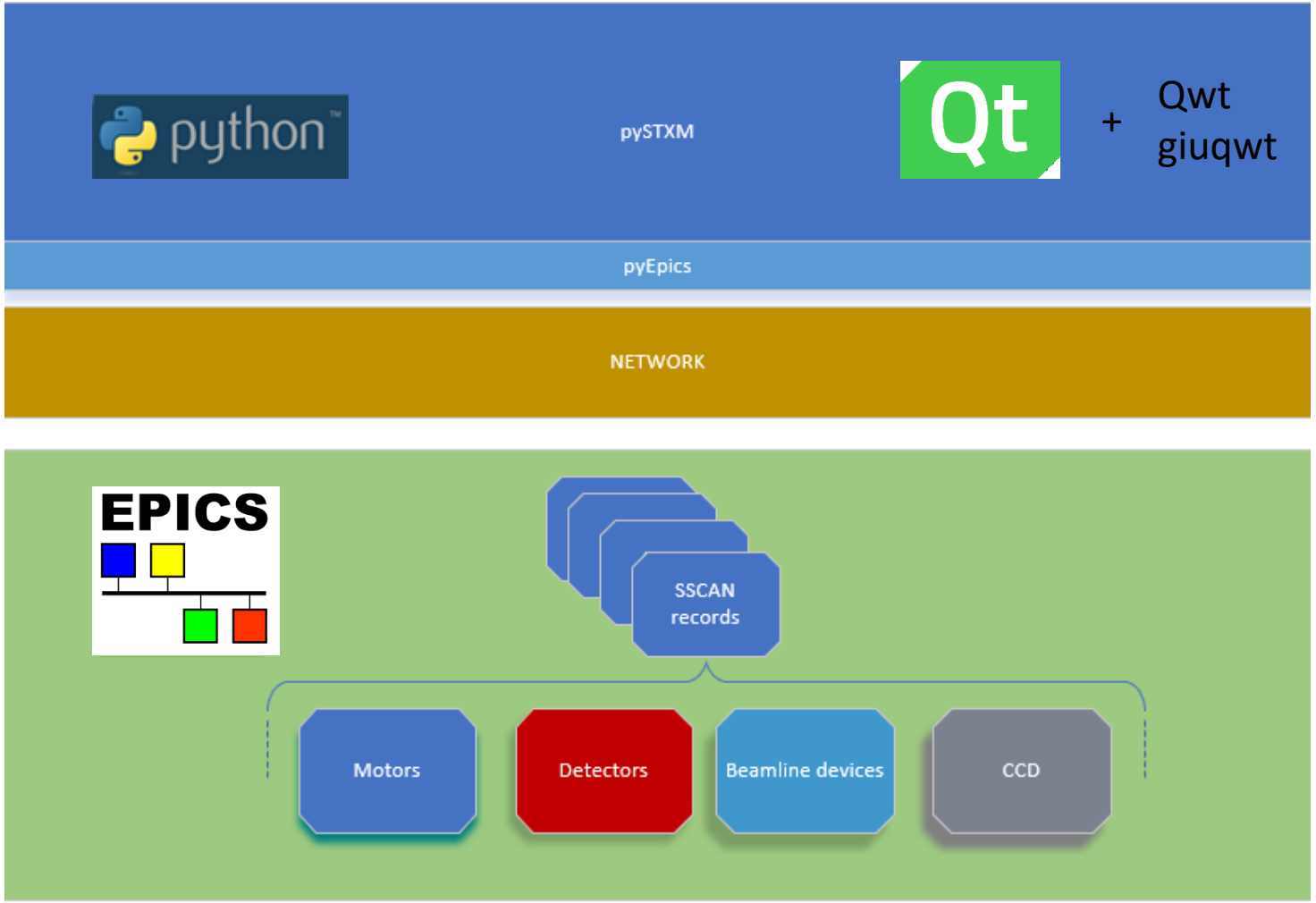




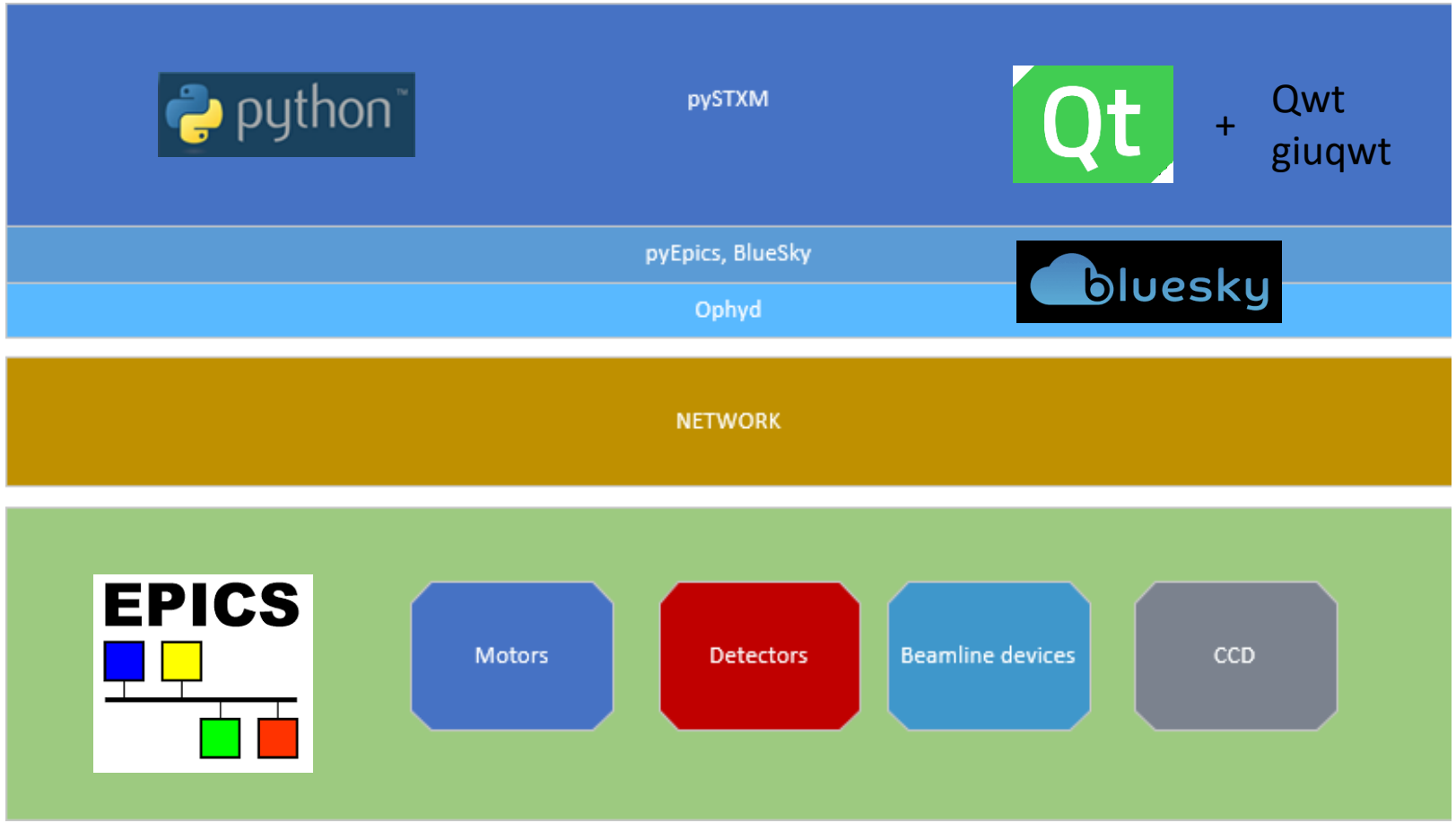


Canadian Light Source | Centre canadien de rayonnement synchrotron

pySTXM:
General software structure, initial implementation

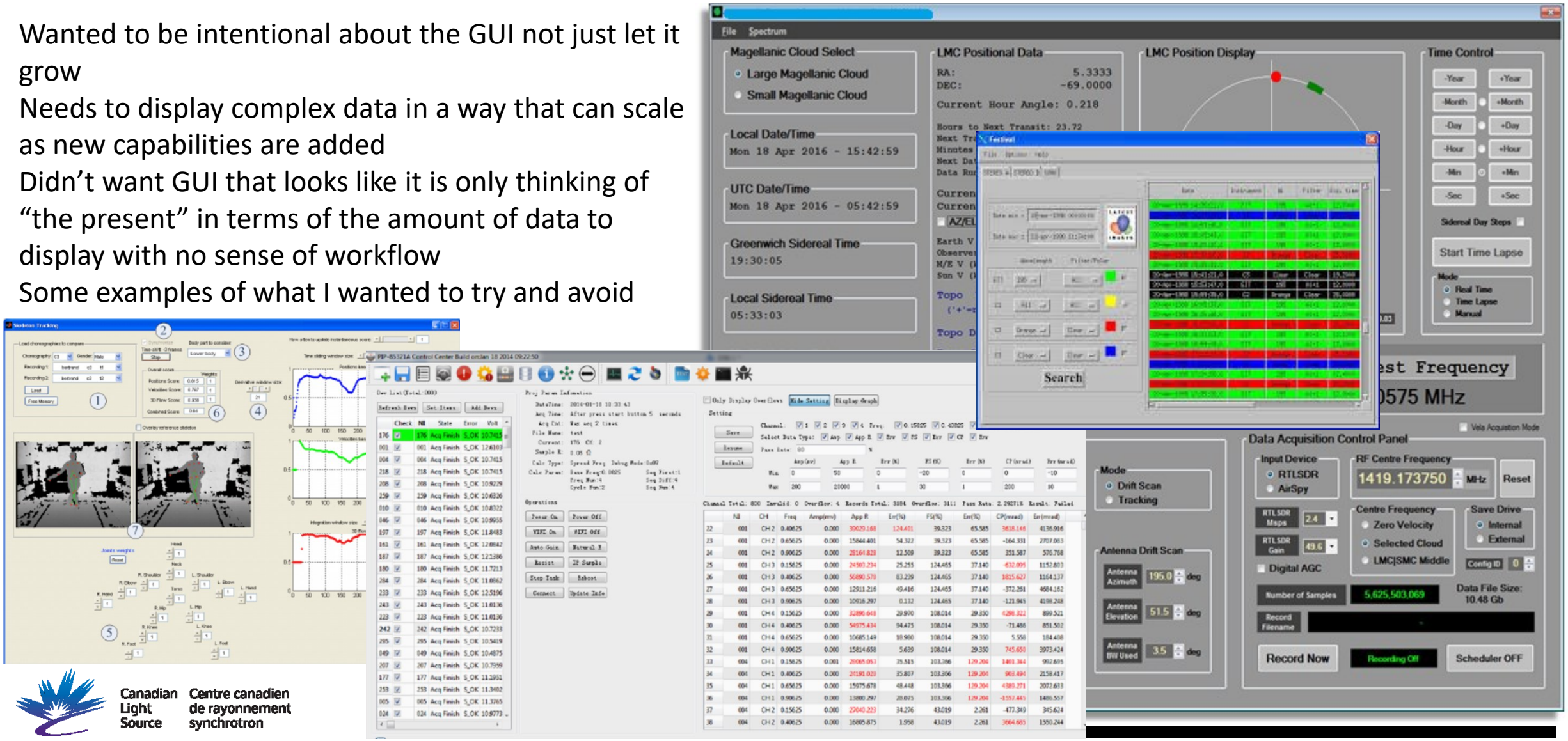


pySTXM:
General software structure, ported to use BlueSky



GUI layout Philosophy: How to avoid clutter as app scales?

- Wanted to be intentional about the GUI not just let it grow
- Needs to display complex data in a way that can scale as new capabilities are added
- Didn't want GUI that looks like it is only thinking of "the present" in terms of the amount of data to display with no sense of workflow
- Some examples of what I wanted to try and avoid

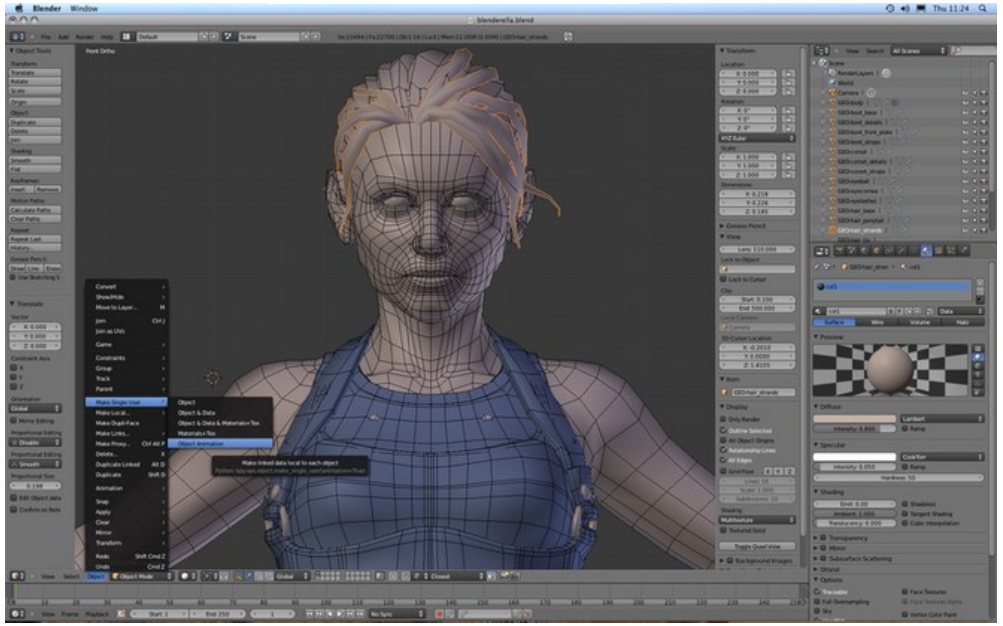


Canadian Light Source | Centre canadien de rayonnement synchrotron

GUI layout and goals: take inspiration from successful software

- Layout that facilitates workflow
- “like typed” information organized into panel areas on the screen
- Minimal color pallet used to express as few different types of information as possible
- Open as few new screens as possible to get at the data
- Inspiration from:

blender™



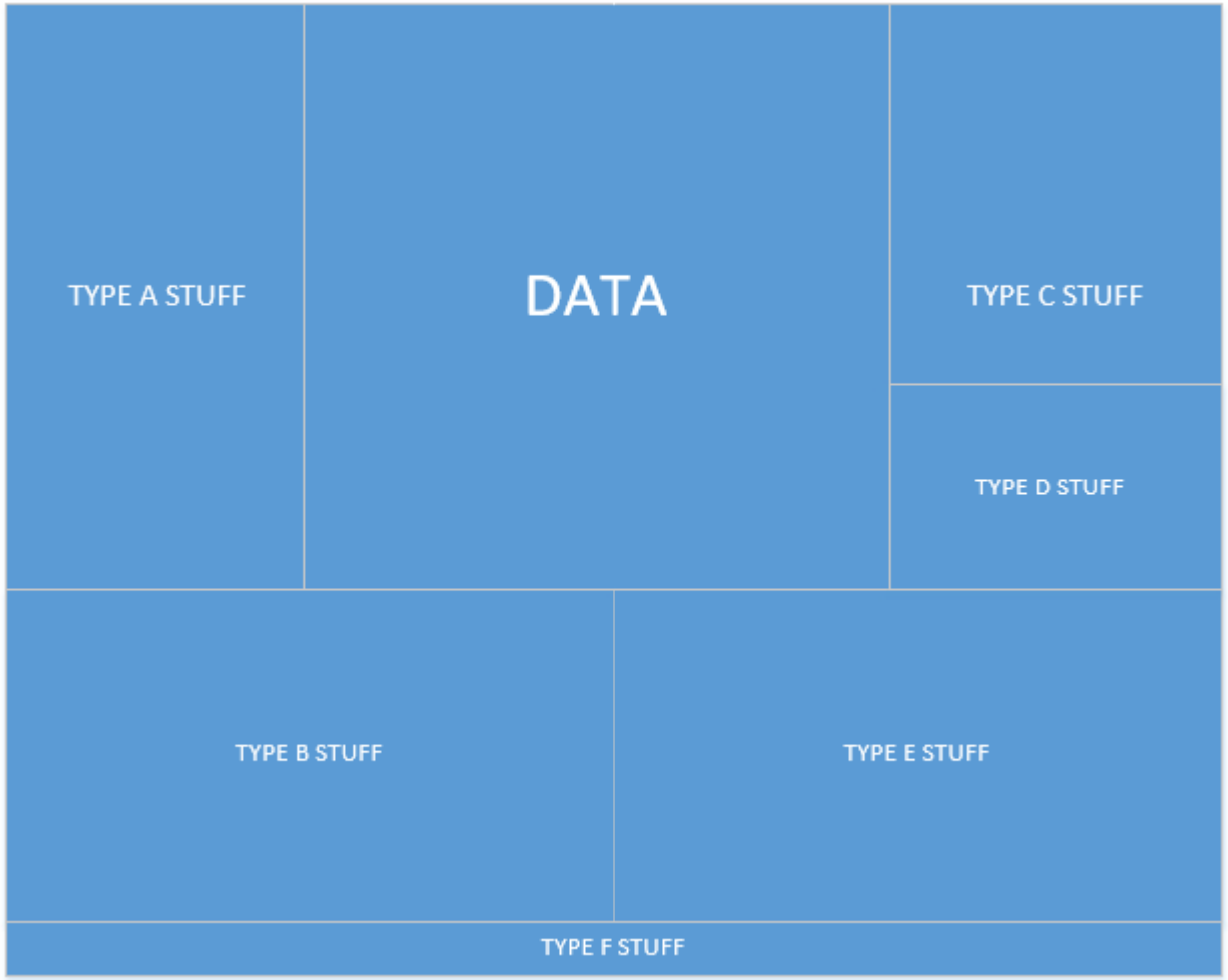
Adobe Photoshop

GUI layout and goals: what I did want

- wanted a structure that would scale as new features and capabilities were realized

- **Type A Stuff:**
  - Scan definition
  - User Preferences

- **Type B Stuff:**
  - Scan control



- **Type C Stuff:**
  - Data that has been collected

- **Type D Stuff:**
  - Device Control

- **Type E Stuff:**
  - Log, Info

- **Type F Stuff:**
  - Status bar

pySTXM GUI

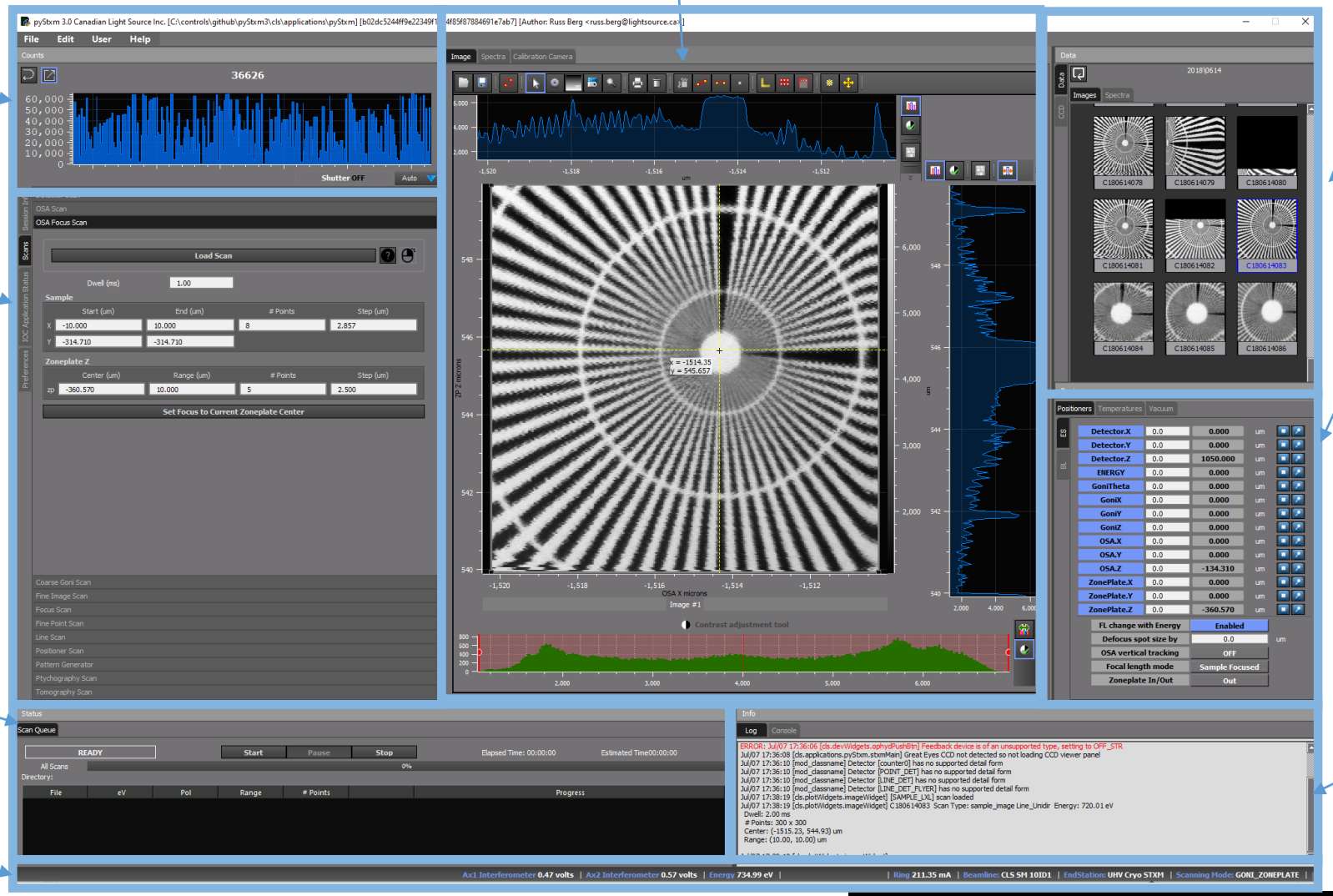
- Image data
- Spectra data
- Calibration Camera

Live PMT

- Scan plugins
- Session info
- Preferences

Scan control + feedback

Status Bar



Data in current data directory

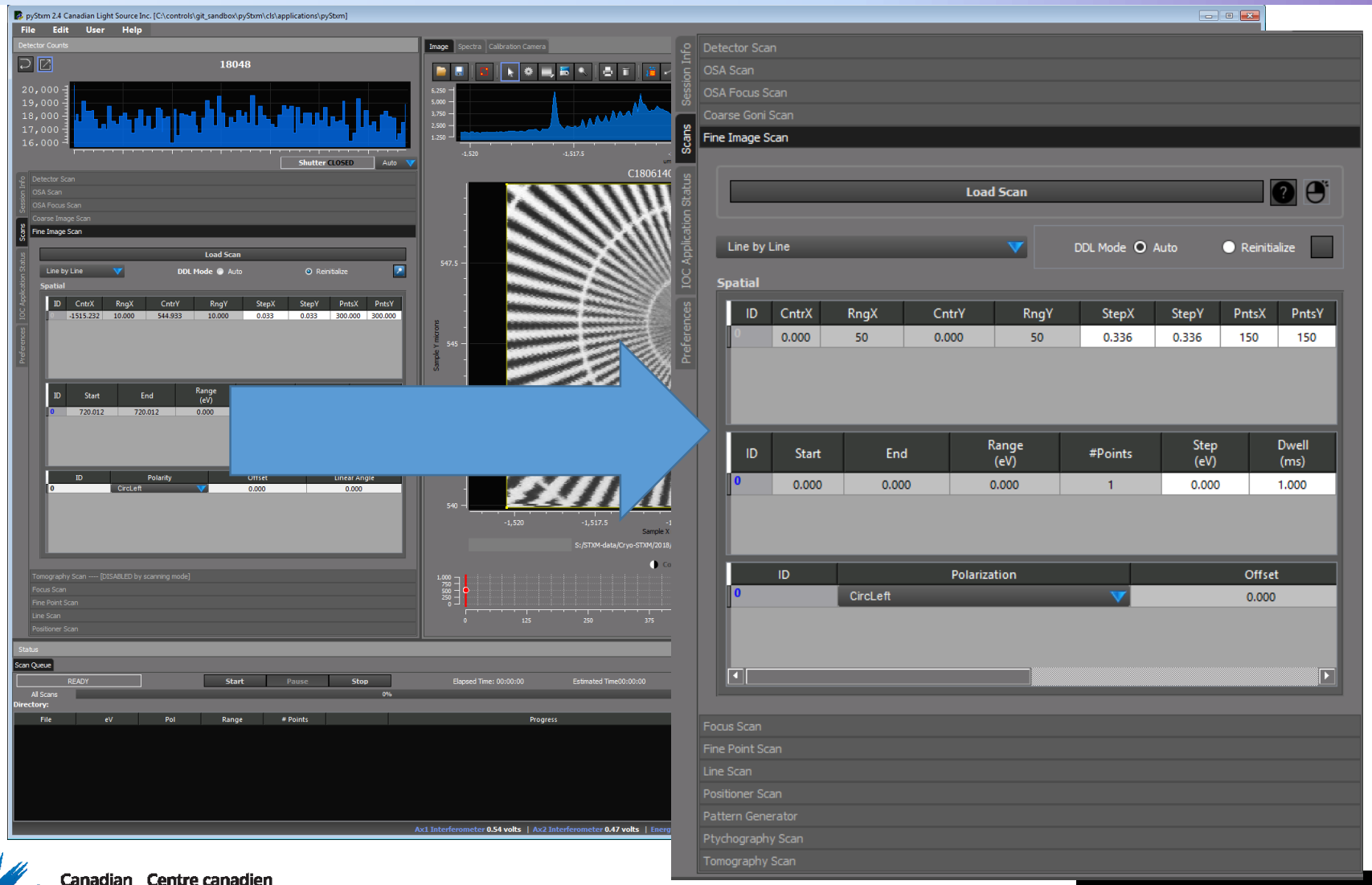
- Updates dynamically

Endstation/Beamline Devices

- Positioners
- Vacuum
- Temperatures

- Log window
- Python console
- Status

Scan plugin layout -> workflow



Scans vertical tab with each individual scan using a horizontal toolbar.

Scan plugins ordered from top to bottom in usual order a user needs to work to collect data.

Top -> alignment/focus scans
Down -> data collect/refine scans

BlueSky plans combined to create higher level STXM scans

One or more base level pre-assembled plans used to create whatever scan is needed

```
66
67  def make_pxp_scan_plan(self, dets, gate, md=None, bi_dir=False):
68      dev_list = self.main_obj.main_obj[DEVICES].devs_as_list()
69      self._bi_dir = bi_dir
70
71      if (md is None):
72          md = {'metadata': dict_to_json(
73              self.make_standard_metadata(entry_name='entry0', scan_type=self.scan_type, dets=dets))}
74
75      @bpp.baseline_decorator(dev_list)
76      #@bpp.stage_decorator(dets)
77      #@bpp.run_decorator(md={'entry_name': 'entry0', 'scan_type': scan_types.OSA_FOCUS})
78      def do_scan():
79          mtr_x = self.main_obj.device(DNM_OSA_X)
80          mtr_y = self.main_obj.device(DNM_OSA_Y)
81          mtr_z = self.main_obj.device(DNM_ZONEPLATE_Z_BASE)
82          shutter = self.main_obj.device(DNM_SHUTTER)
83
84          x_traj = cycler(mtr_x, self.x_roi[SETPOINTS])
85          y_traj = cycler(mtr_y, self.y_roi[SETPOINTS])
86          zz_traj = cycler(mtr_z, self.zz_roi[SETPOINTS])
87
88          yield from bps.stage(gate)
89          shutter.open()
90          # the detector will be staged automatically by the grid_scan plan
91          yield from scan_nd(dets, zz_traj * (y_traj + x_traj), md=md)
92
93          shutter.close()
94          # yield from bps.wait(group='e712_wavgen')
95          yield from bps.unstage(gate)
96
97          print('OsaFocusScanClass: make_scan_plan Leaving')
98
99      return (yield from do_scan())
100
```

Pre-assembled Plans

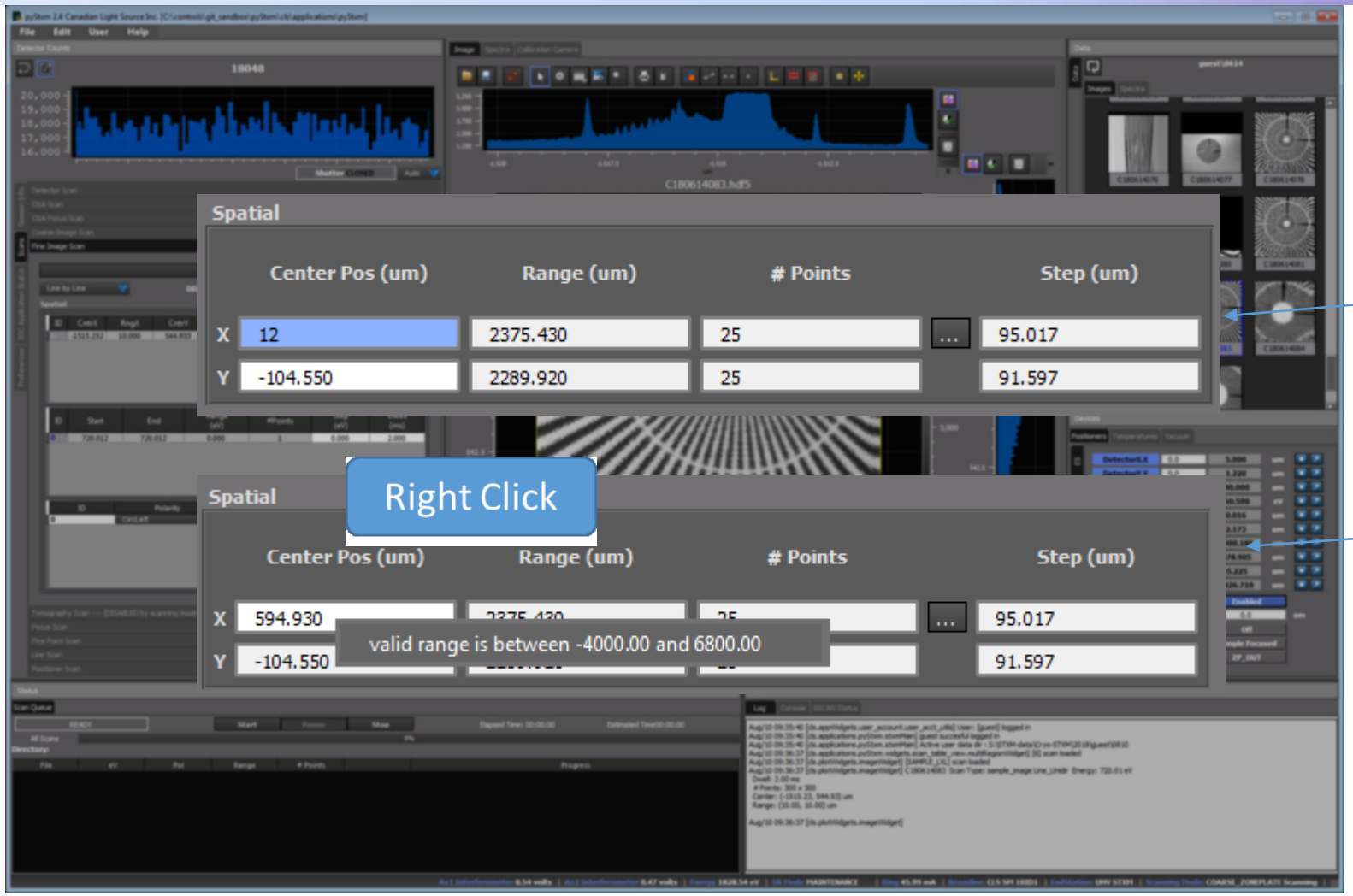
Below this summary table, we break the down the plans by category and show examples with figures.

Summary

Notice that the names in the left column are links to detailed API documentation.

`count`	Take one or more readings from detectors.
`scan`	Scan over one multi-motor trajectory.
`rel_scan`	Scan over one multi-motor trajectory relative to current position.
`list_scan`	Scan over one or more variables in steps simultaneously (inner product).
`rel_list_scan`	Scan over one variable in steps relative to current position.
`list_grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`rel_list_grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`log_scan`	Scan over one variable in log-spaced steps.
`rel_log_scan`	Scan over one variable in log-spaced steps relative to current position.
`grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`rel_grid_scan`	Scan over a mesh relative to current position.
`scan_nd`	Scan over an arbitrary N-dimensional trajectory.
`spiral`	Spiral scan, centered around (x_start, y_start)

Knowledgeable text edit fields

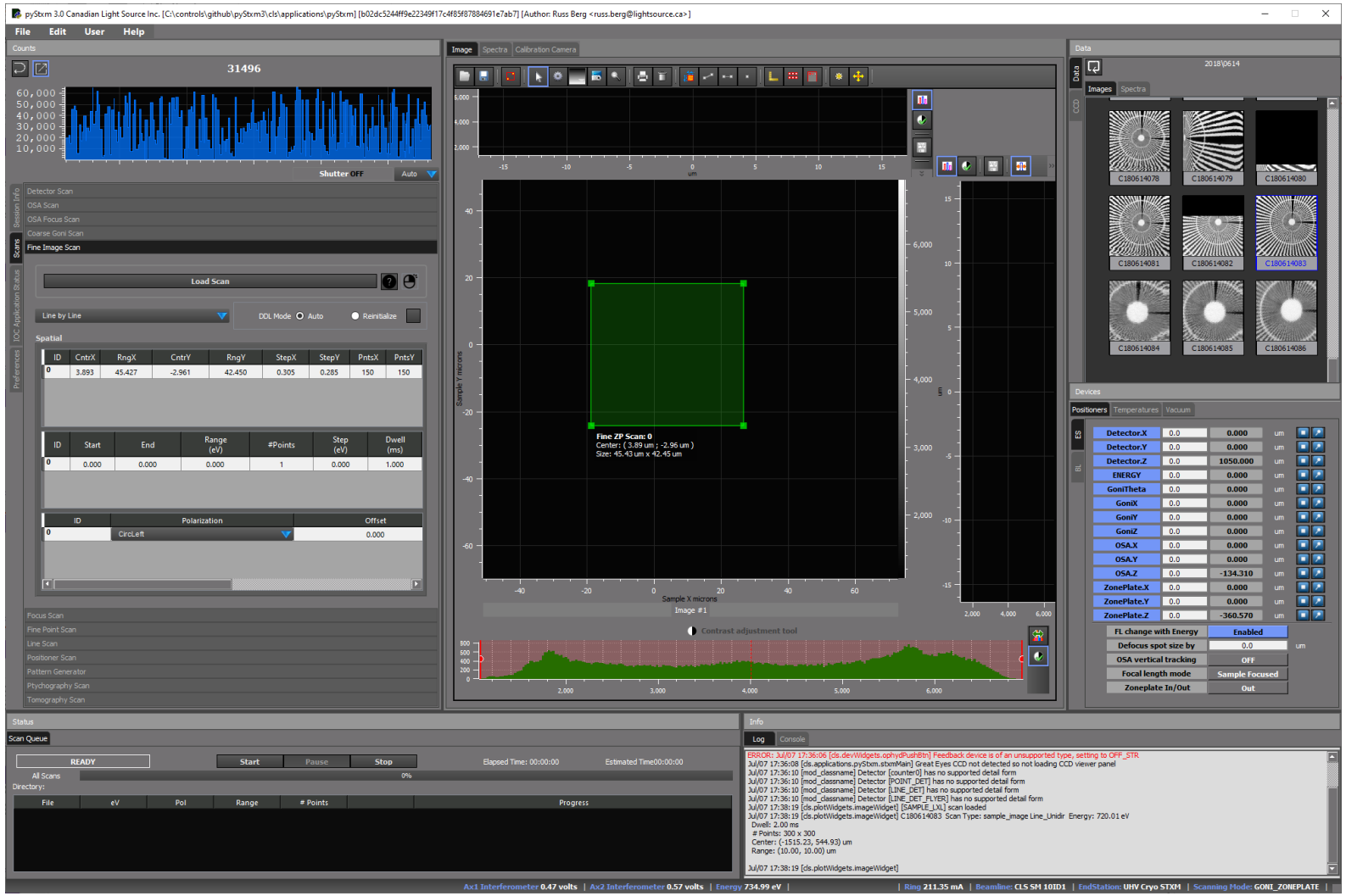


Smart text edit fields:

- Color
- Snap back to initial value if focus changed
- Int/Float validation

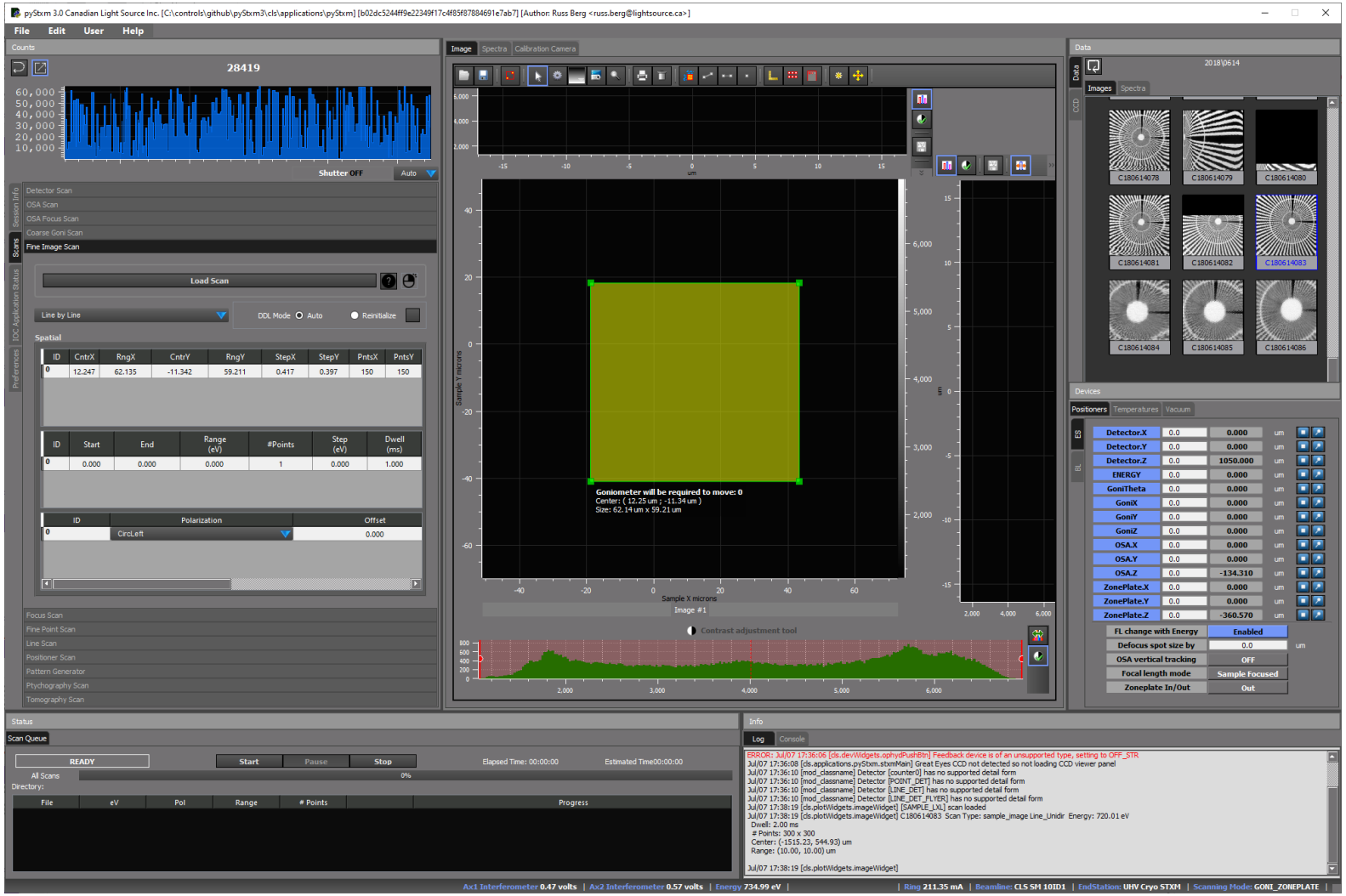
Limit values entered to the soft limits of that positioner

Only allow valid scans to be configured/executed



Visual cues to indicate an impossible scan

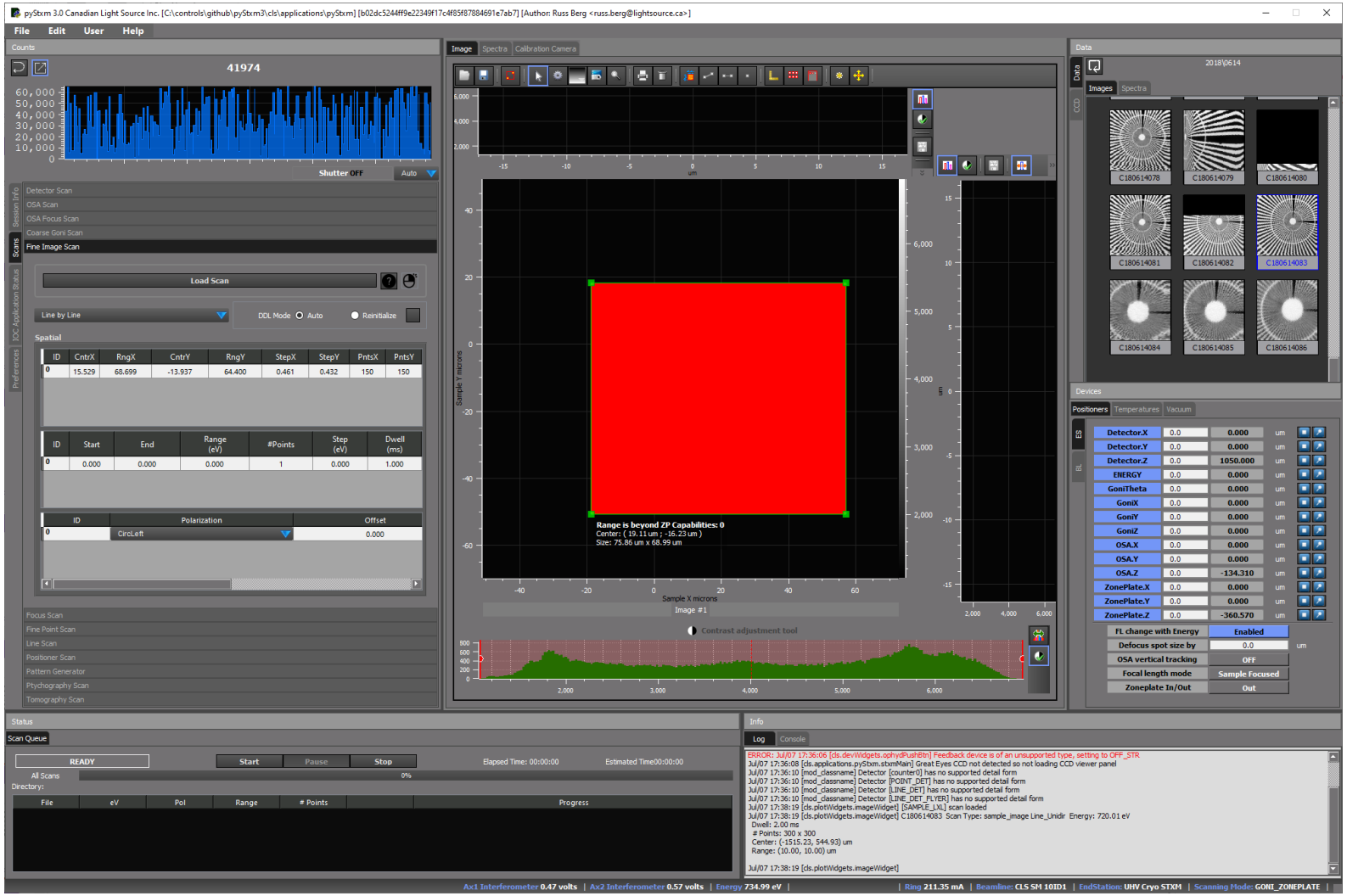
Only allow valid scans to be configured/executed



Visual cues to indicate an impossible scan

- When nearing the max range of the positioner the selection area background changes to solid yellow indicating the situation

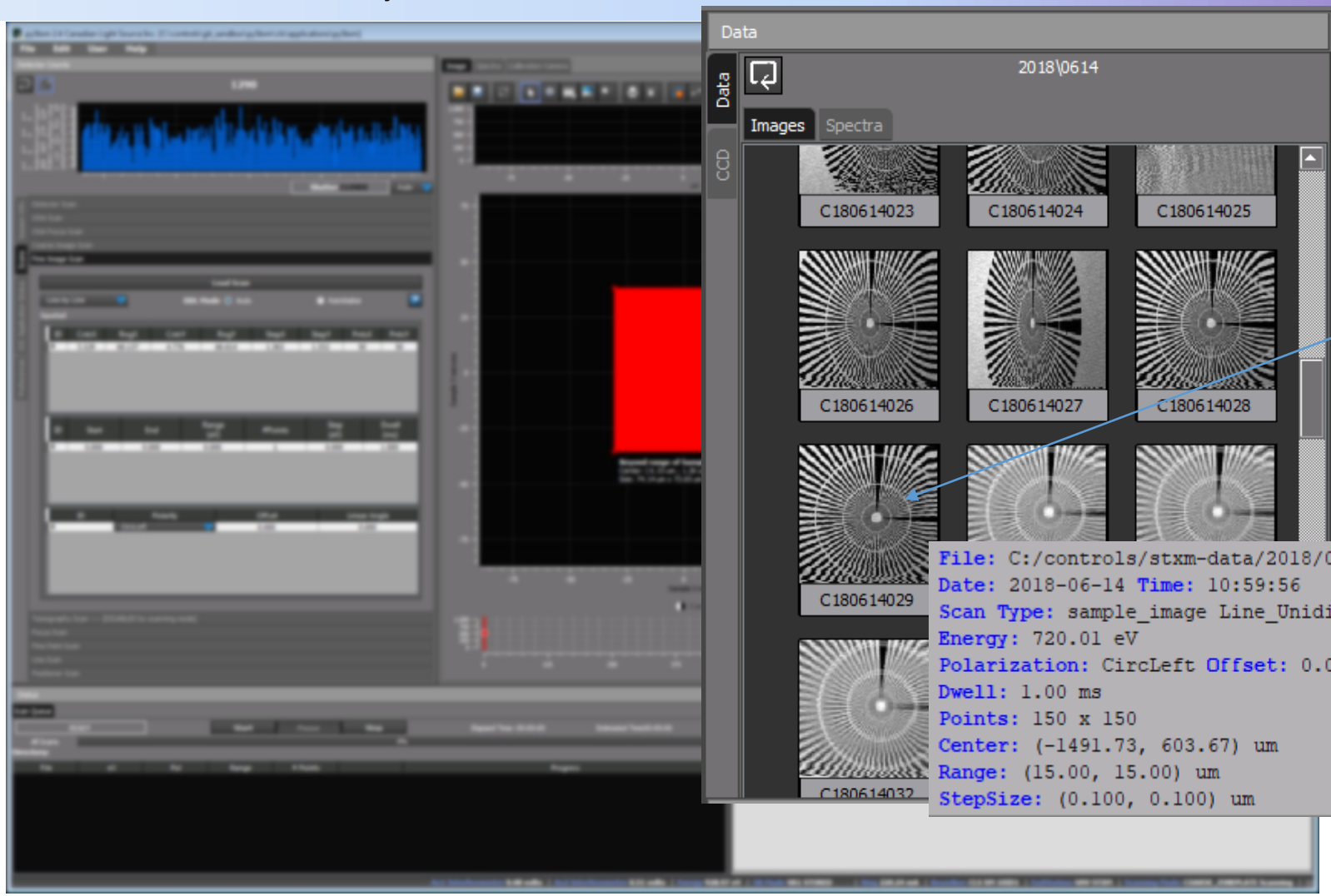
Only allow valid scans to be configured/executed



Visual cues to indicate an impossible scan

- When the max range of the positioner would be exceeded the selection area background changes to solid red and the scan range fields are no updated until scan range is valid again

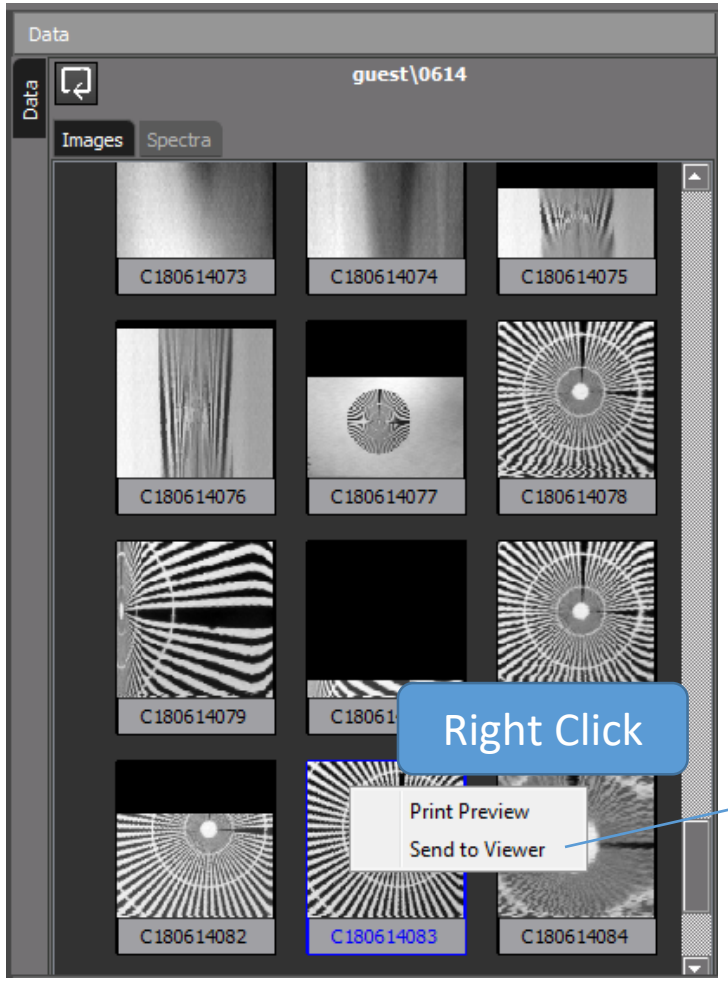
Easy access to data, Thumbnail viewer

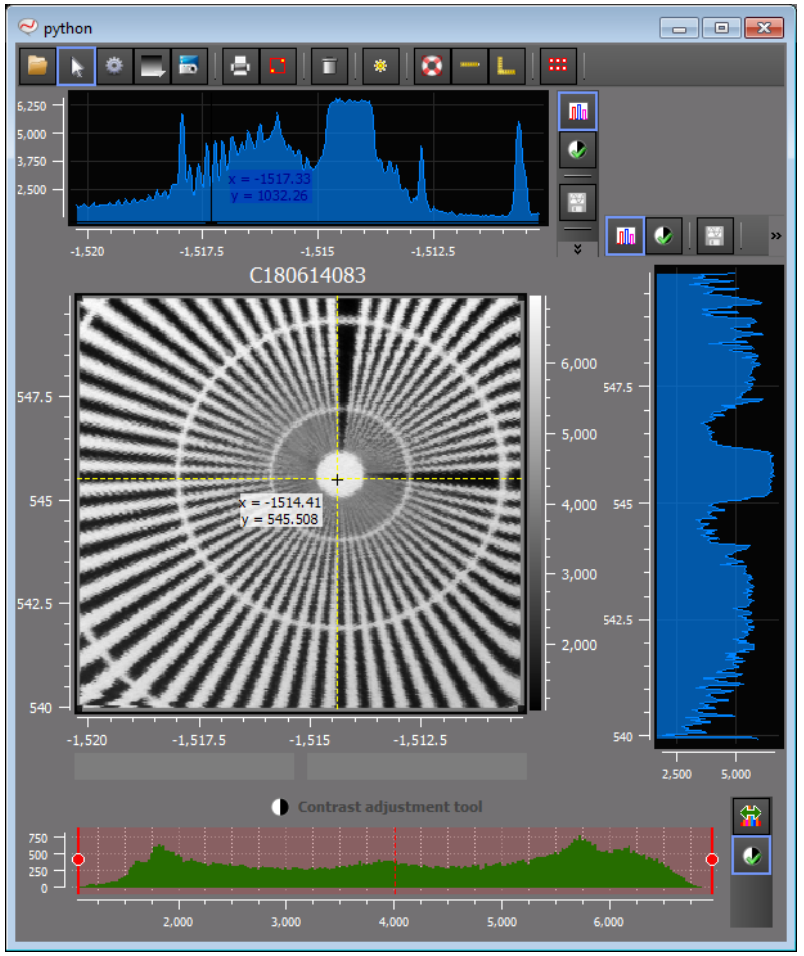


Loads data from disk as it becomes available

Mouse over brings up tooltip of metadata mined from the hdf5 file(s)

Easy access to data, print or load into a viewer



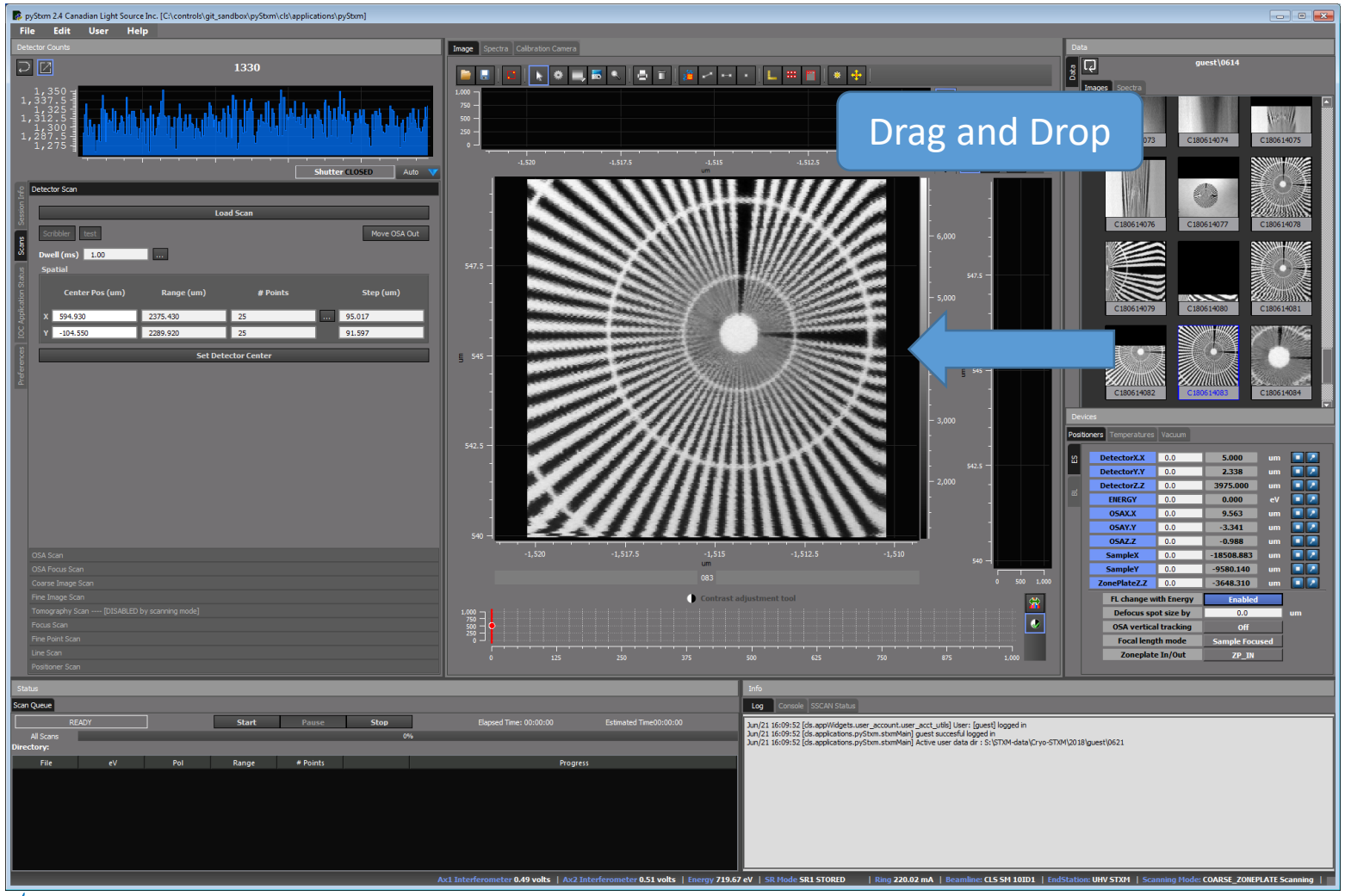


Right click menu:

- "**print**" data small enough to fit in log book
- "**send to Viewer**" to load data into separate viewer

pySTXM

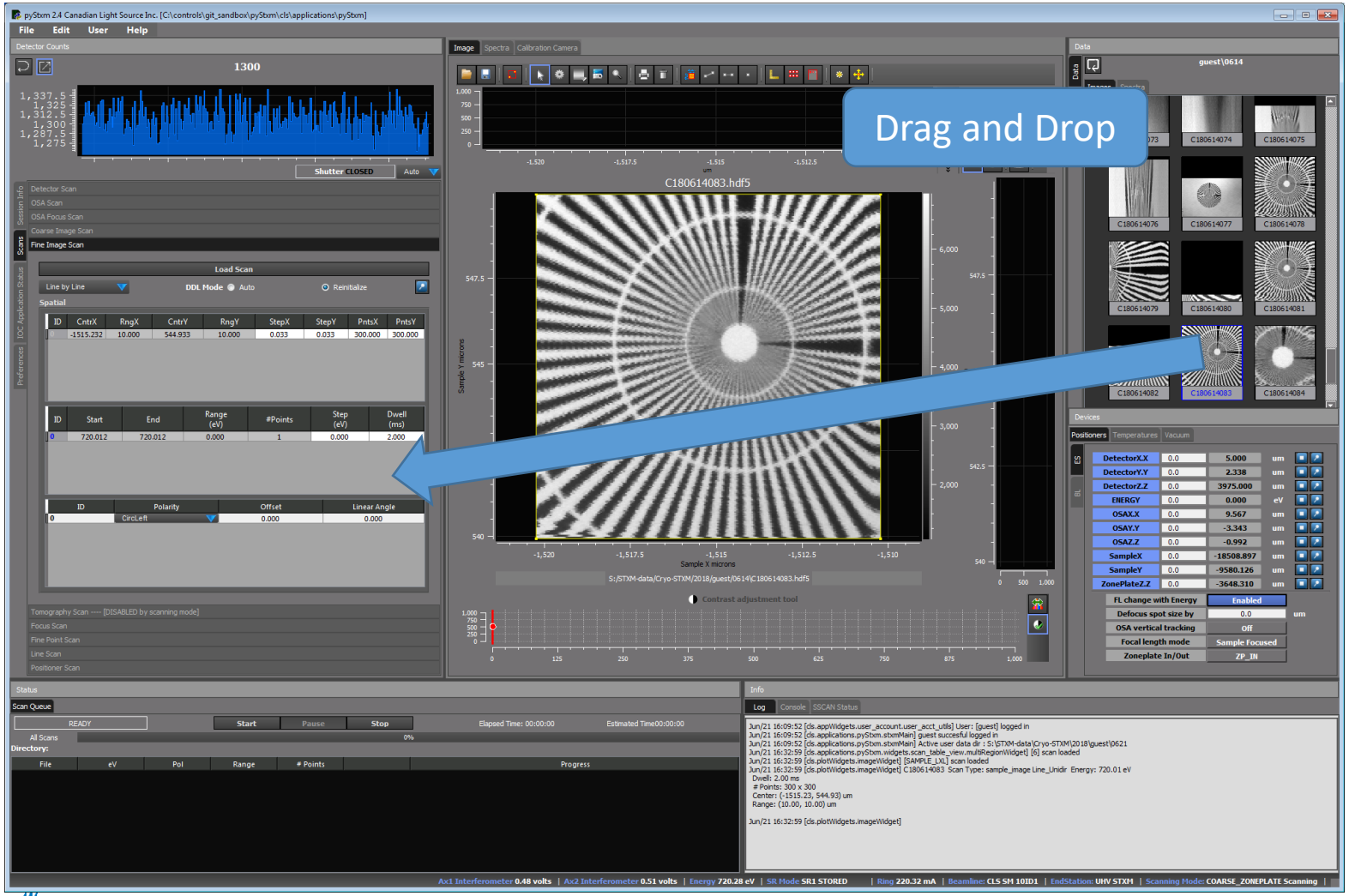
Easy access to data, quickly load a previous scan



Load previous scan and use to select scan area

Drag and drop data from Thumbnail viewer to central plot to be used to select ROI for a different scan

Easy access to data, Load previous scan for re-execution



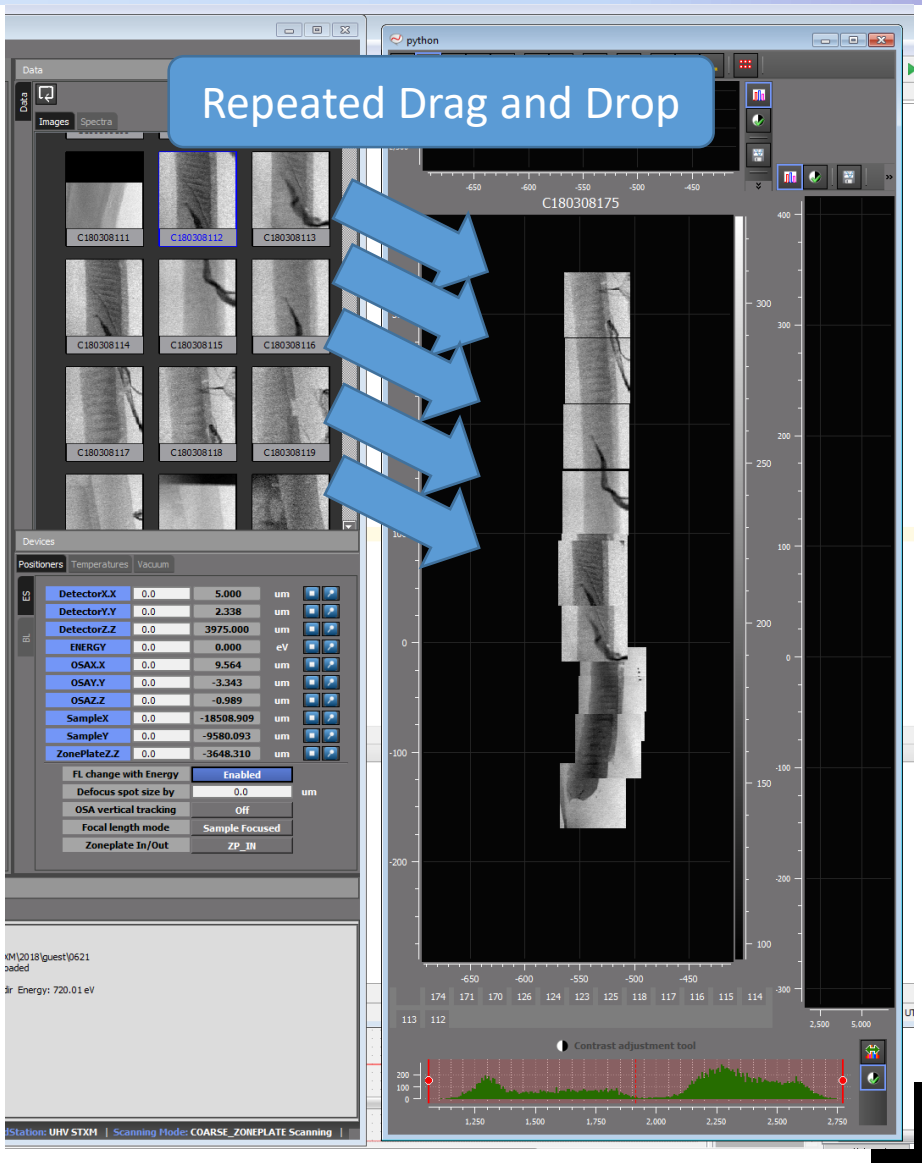
Drag and drop data from Thumbnail viewer to central plotter

Or

Onto the scan plugins pane to be reloaded

pySTXM

Interactively build larger view from smaller images

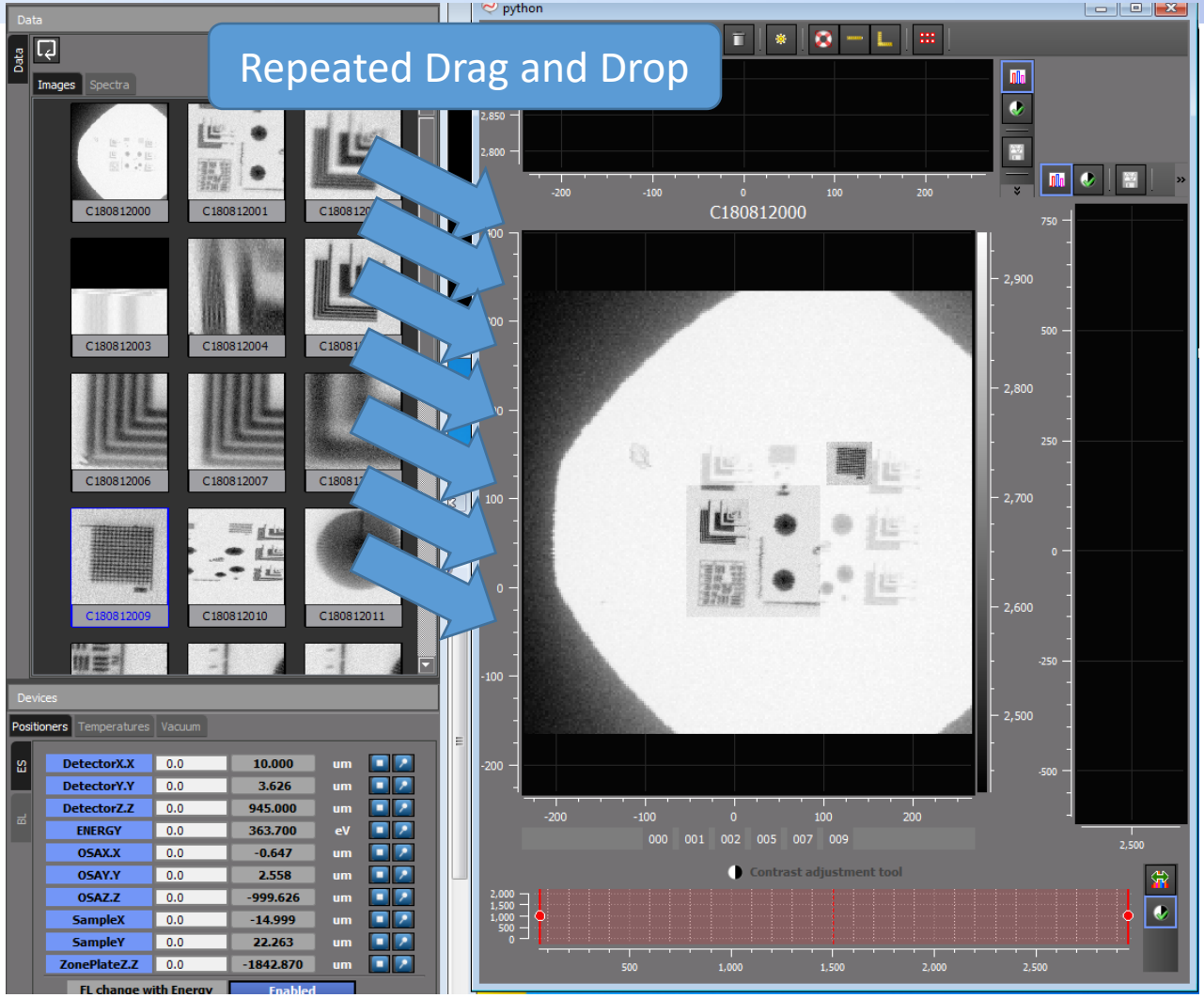


Drag and drop multiple images from thumbnail viewer onto another viewer building up a mosaic of a number of scans

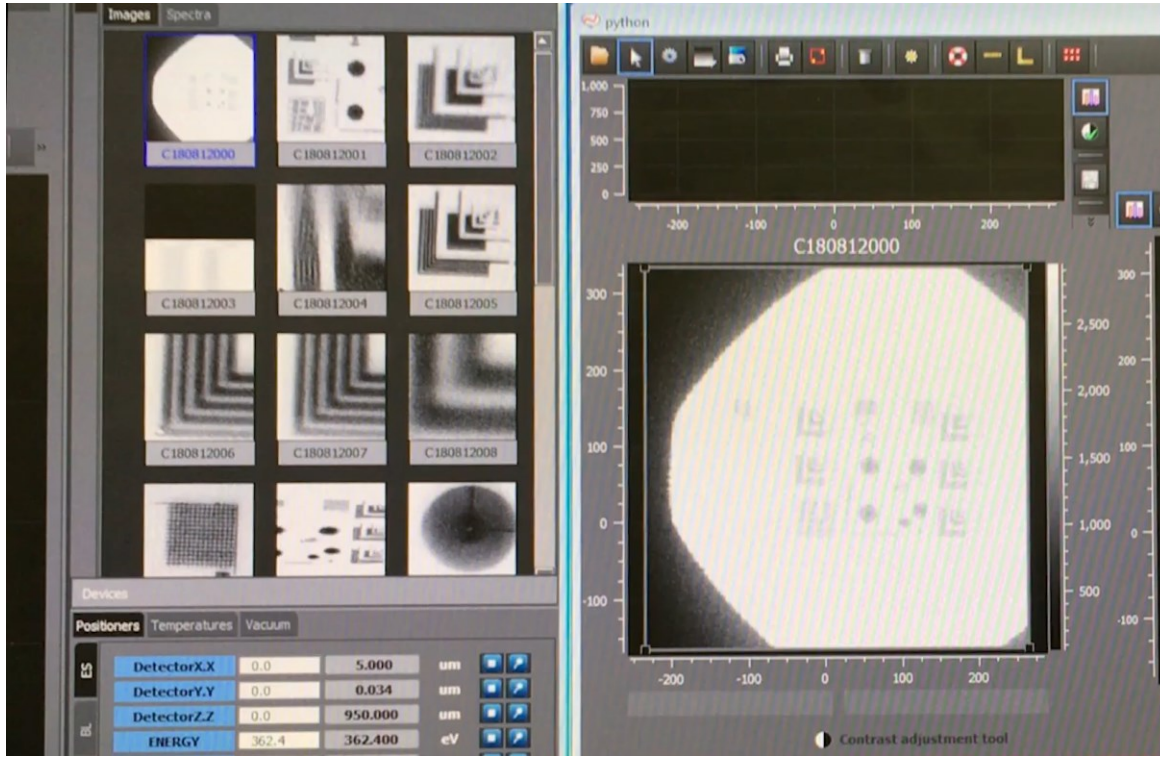
Or select and load multiple files from disk

pySTXM

combine coarse and fine images



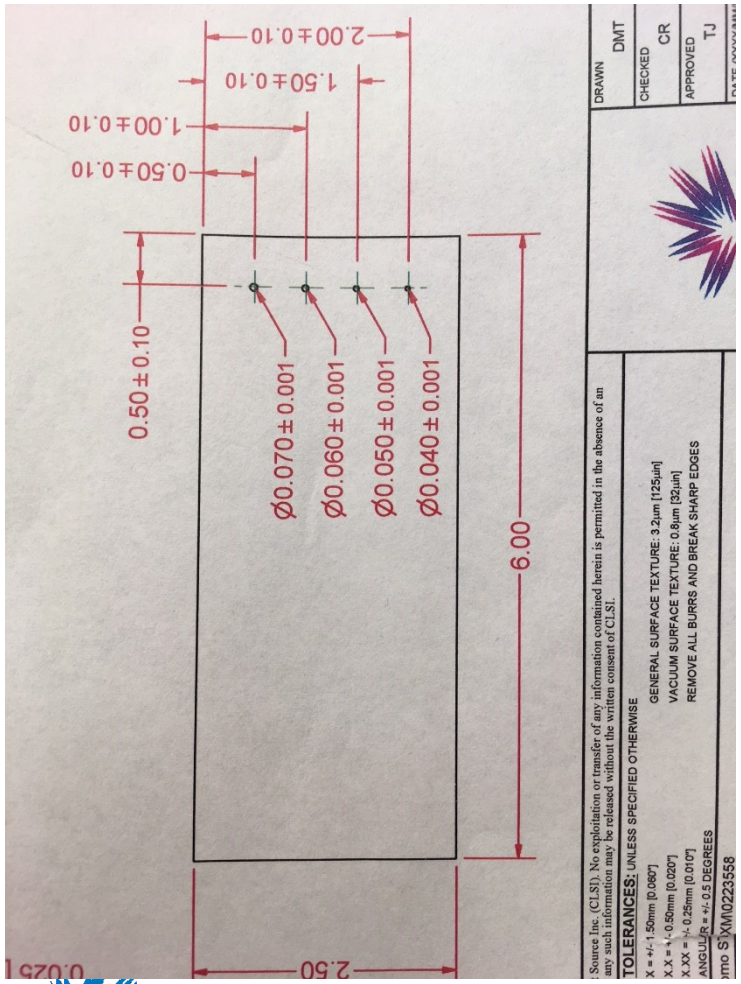
Images are placed on top of each other based on scan resolution and scan range



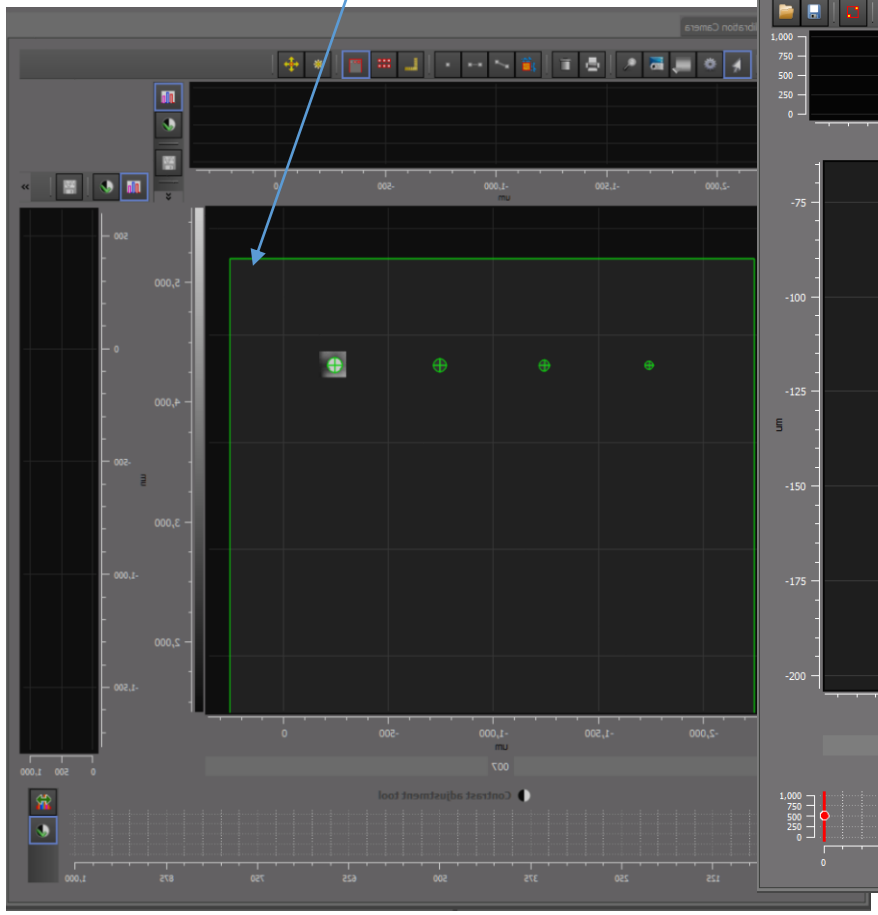
pySTXM

visual tools - Overlay images of OSA, Sample Holder

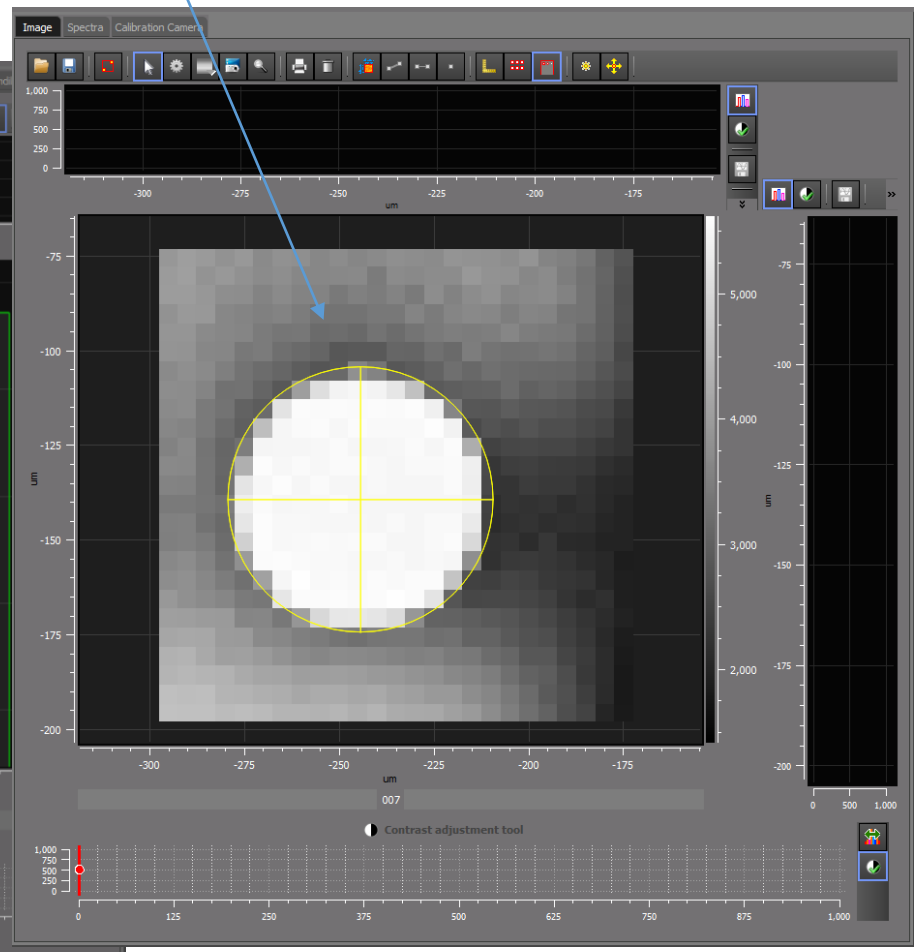
CAD drawing of 4 hole OSA



Recreated as a drag 'able overlay drawing

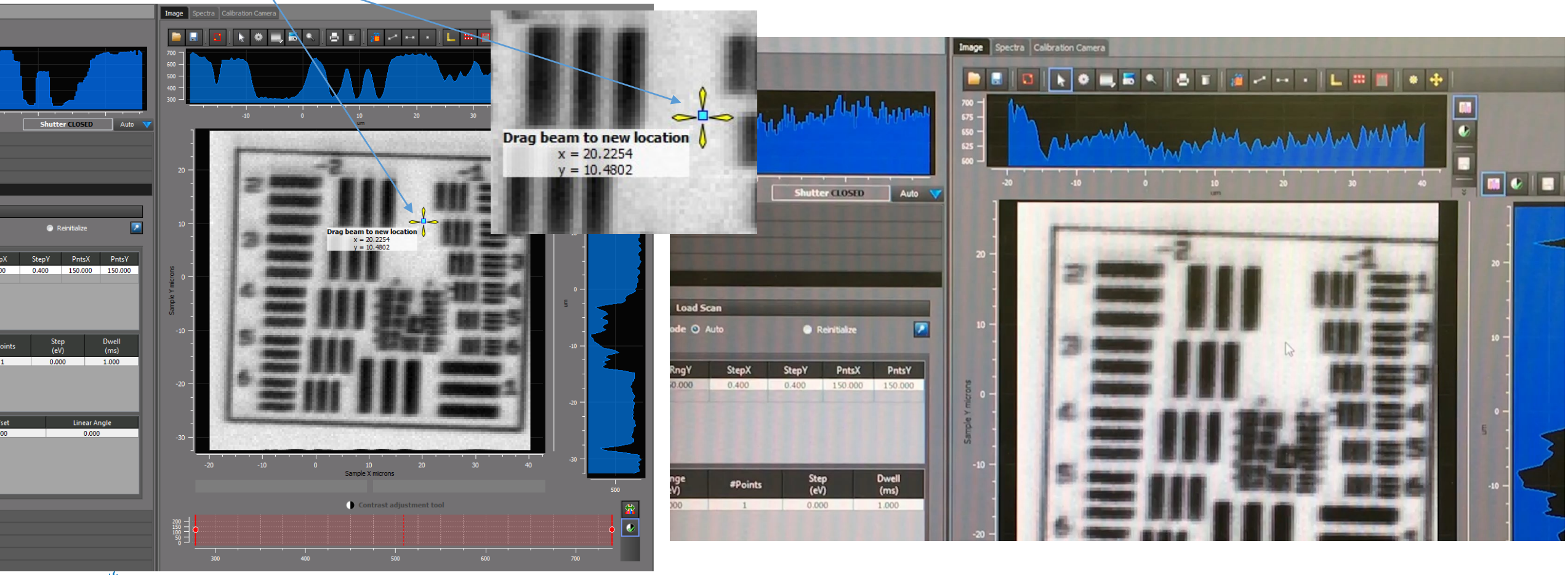


OSA scan image

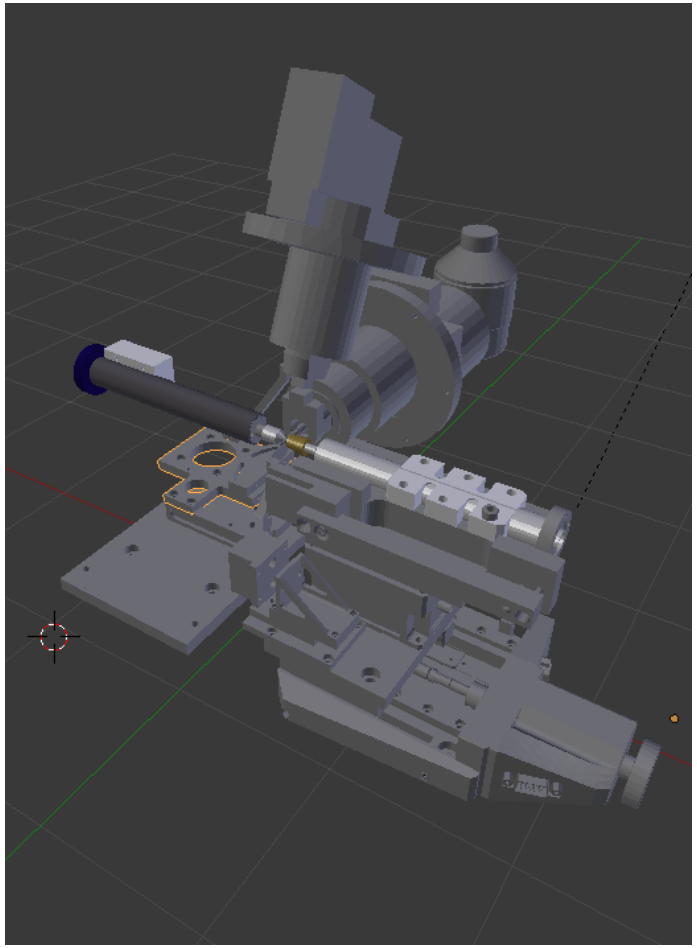


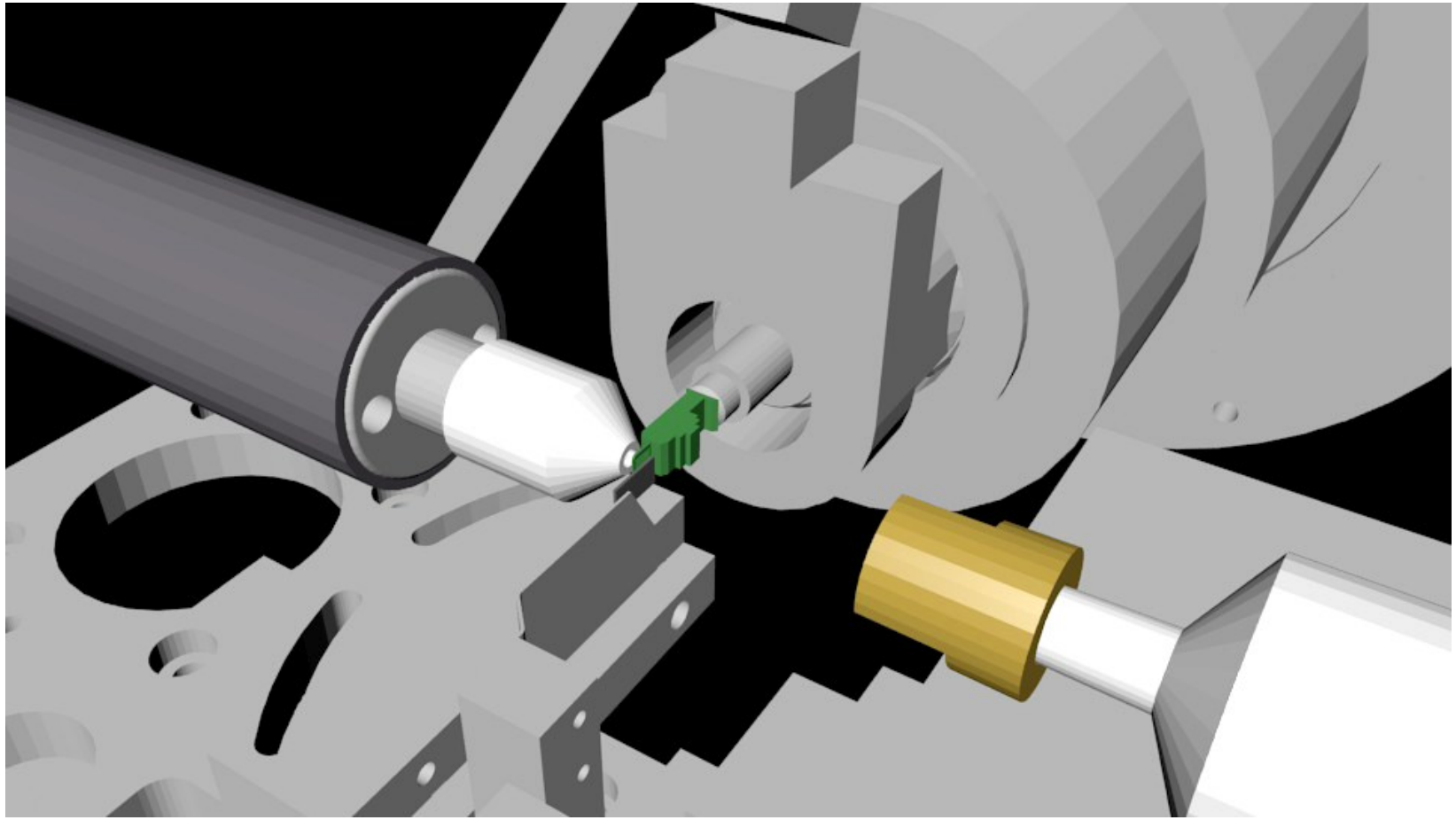
Interactive tools – Drag the beam

Target shows current position of beam, then use tool to drag it to desired location, ex: I0 optimization



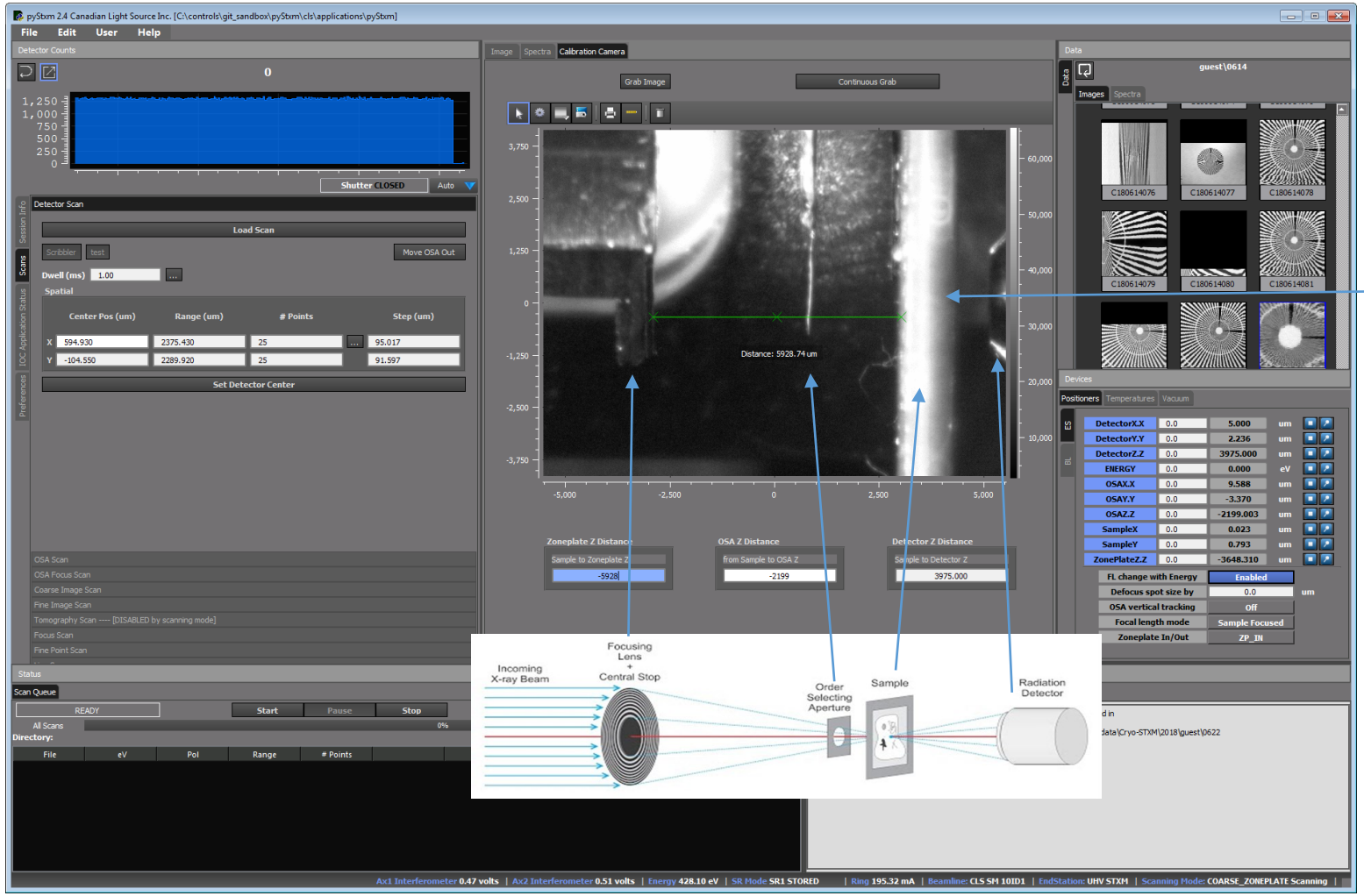
To prevent collisions, typically requires an expert to calibrate the system





pySTXM

Make use of CCD camera that was there for a different purpose



Calibration Camera, roughly calibrate 3 devices in seconds

Frame of video taken from fire wire camera looking inside the UHV STXM tank.
Used to set the initial distance of:

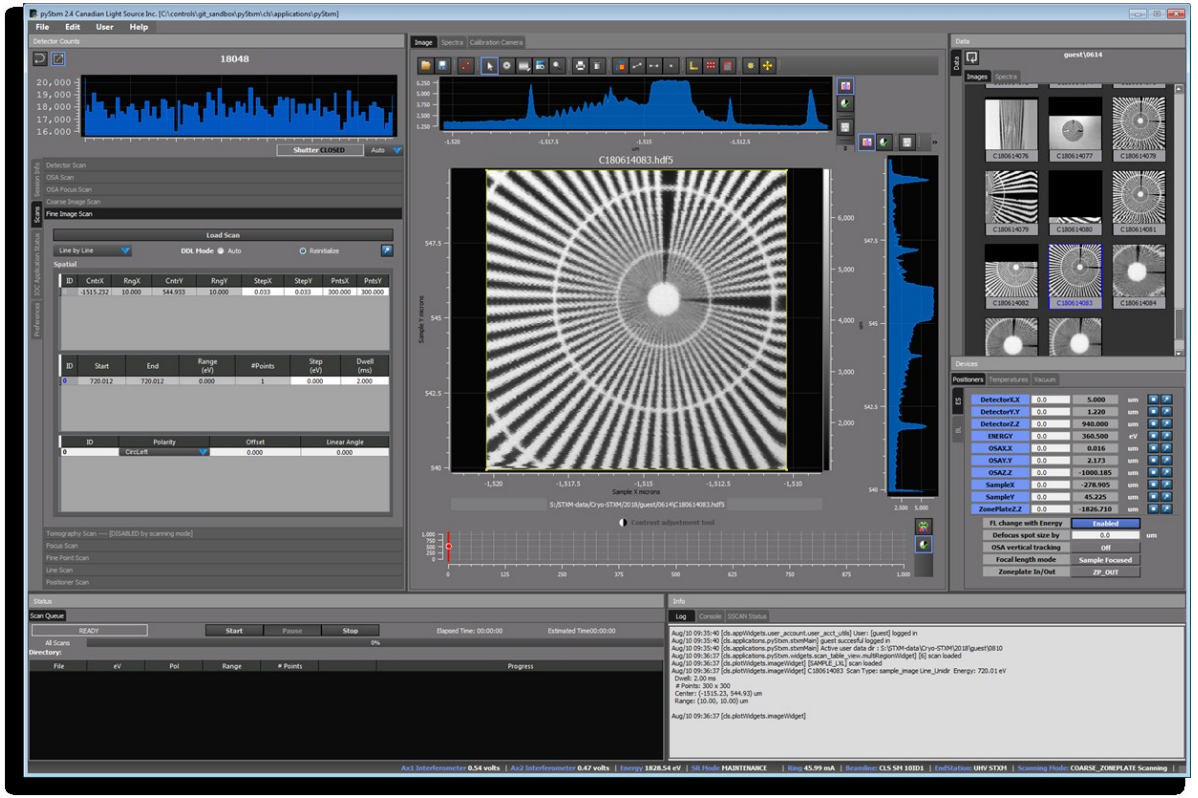
- Zoneplate Z
- OSA Z
- Detector Z

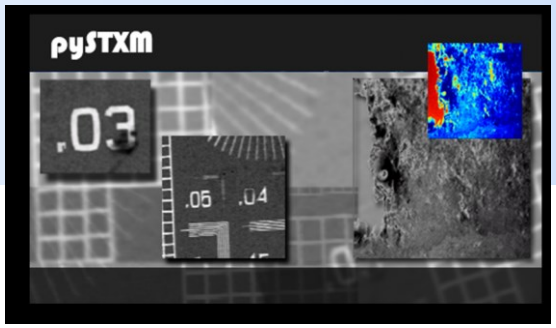
Future for the project

- Install as part of a current beamline upgrade 10ID1
- Extend to include Ptychography (started)
- Automate Tomography

- Have interest from other labs, no simple way **yet** for people to install/configure without my help
- Goal of making this an inter lab collaboration

https://github.com/RussBerg/pyStxm3.git





pySTXM

Scanning microscope data collection with Python , Qt and BlueSky

Acknowledgments

- **10ID1 Beamline:** Jian wang, Chithra Karunakaran, Charan Kupili
- **Canadian Light Source:** Adam Leontowich, Jan Geilhufe, Michel Fodje
- **BlueSky/Ophyd:** Daniel Allan, Tom Caswell, Maksim Rakitin

Thank you very much for listening!