TomoScan and TomoStream Python Software for Tomography Data Collection

Mark Rivers (University of Chicago)

Francesco DeCarlo (APS)

Viktor Nikitin (APS)

March 17, 2021

Introduction

- TomoScan was a collaboration with Francesco
- TomoStream is entirely Francesco and Viktor, not me
- May be a dedicated presentation on that in the future

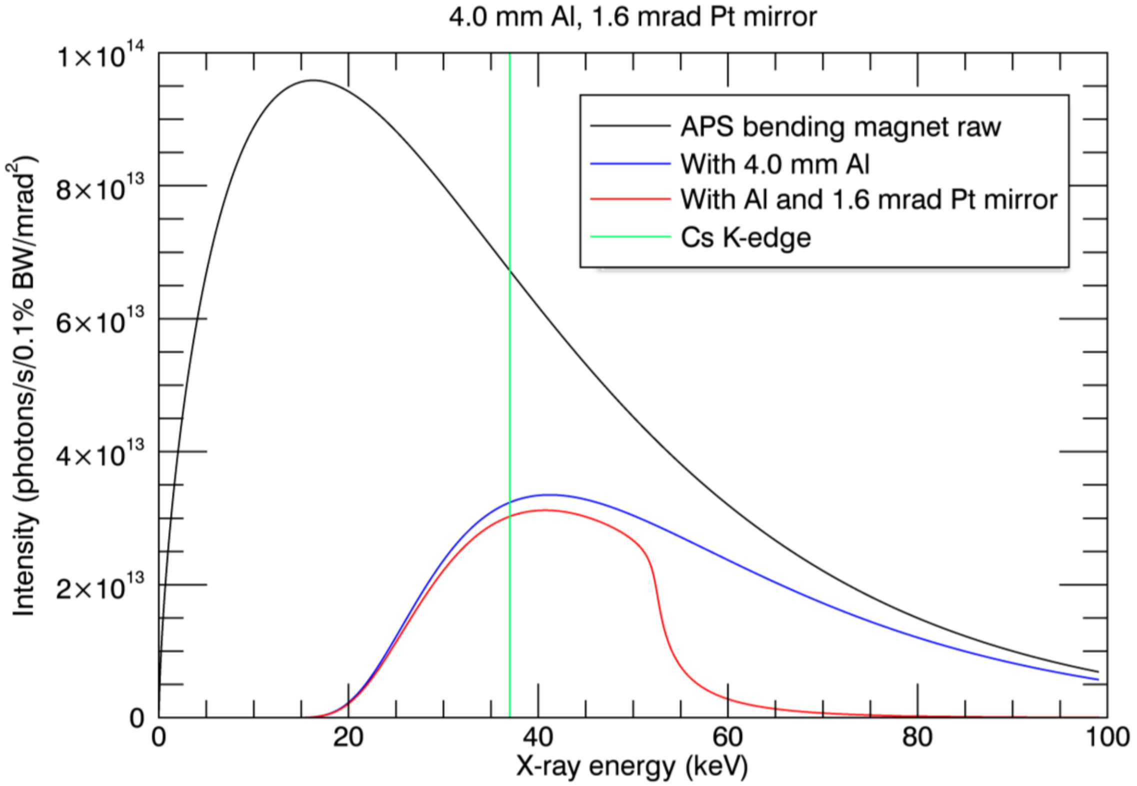
- Only a few beamlines run tomography
- But the concepts presented here can almost certainly be useful for other techniques

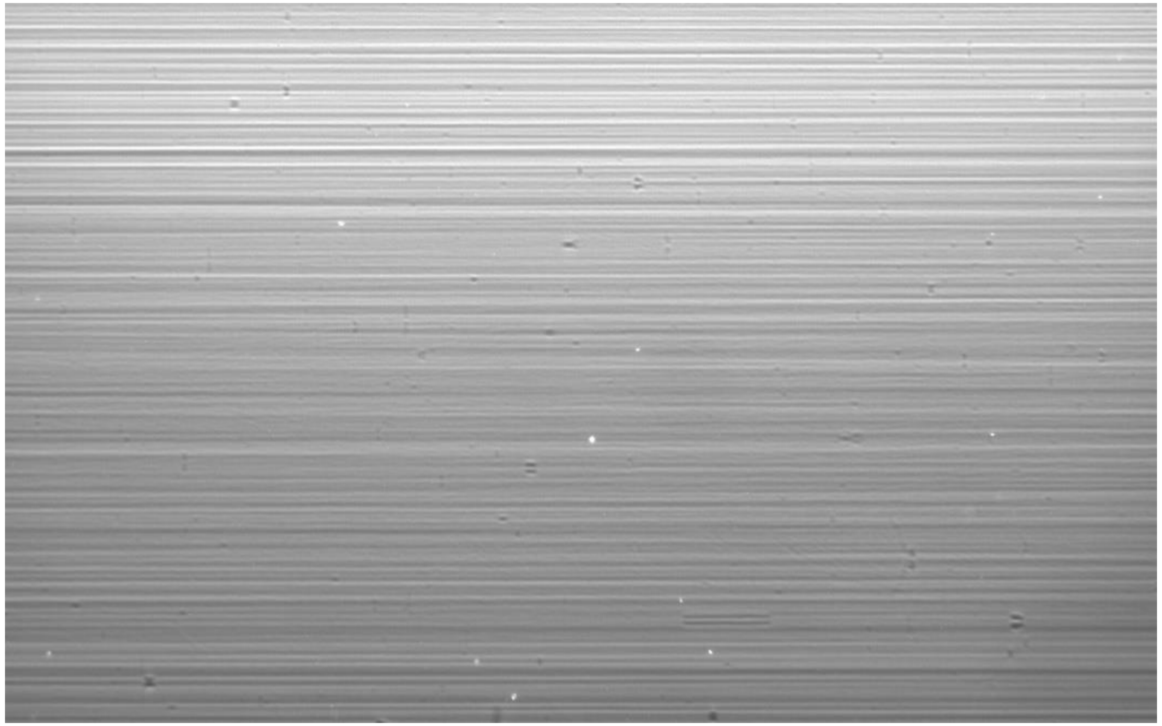
Tomography at APS Beamline 13-BM-D

- Bending magnet source, critical energy ~20 keV
- Beamline modes:
  - Monochromatic beam, 10-80 keV, Si (111)
  - Pink beam, 1.1 m long vertical mirror bounces down
    - Can be bent to focus or defocus
  - White beam
- Both ambient and very high-pressure tomography
- Ambient runs about 30% of the time
  - Several non-tomography experiments in same station

Pink Beam, Mirror=1.6 mrad

- Mirror angle=1.6 mrad
- 4 mm Al absorber
- 2 ms exposure time, 66 frames/s, 13.6 seconds total
- 8 mm x 5 mm field of view shown

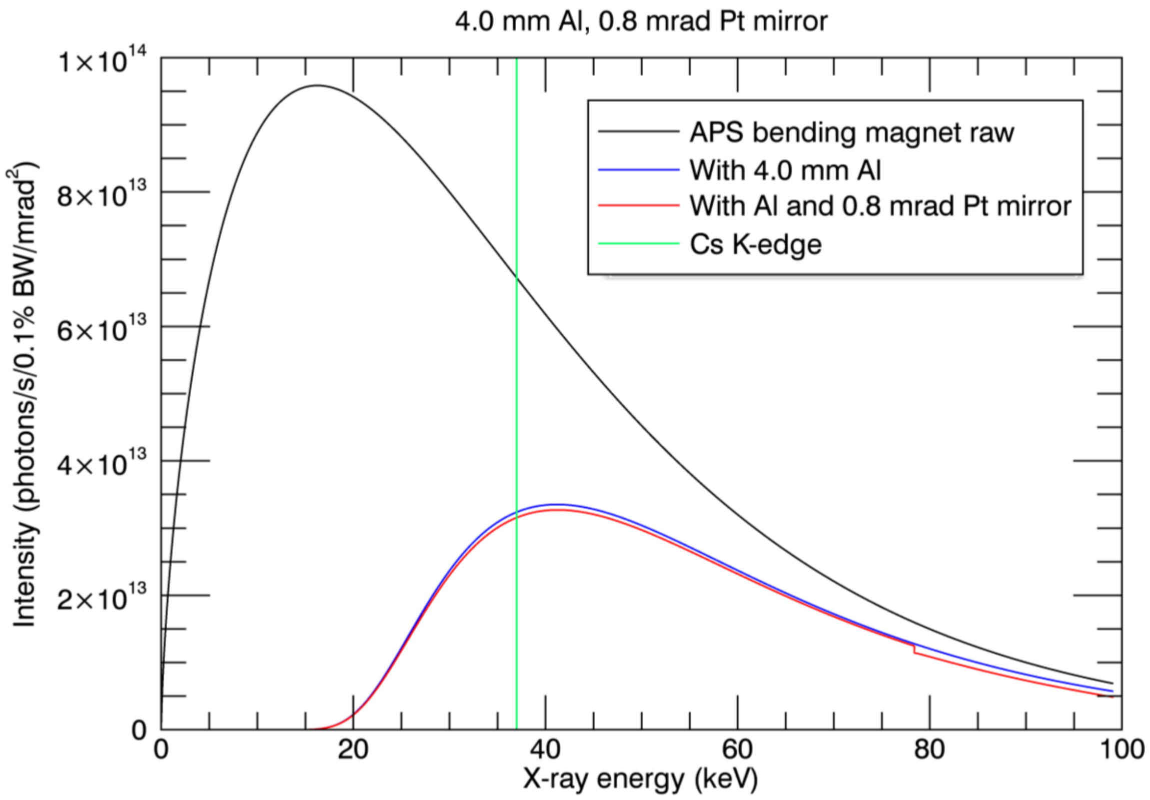


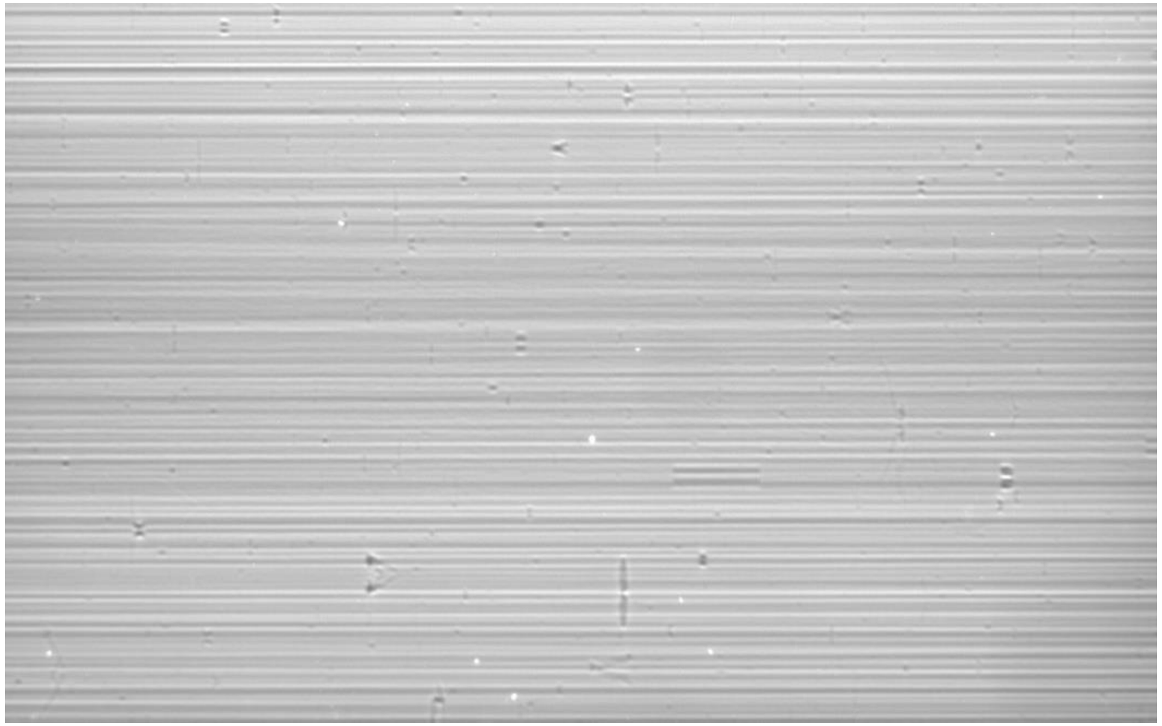


Flat field

Pink Beam, Mirror=0.8 mrad

- Mirror angle=0.8 mrad
- 4 mm Al absorber
- 8 mm x 5 mm field of view shown

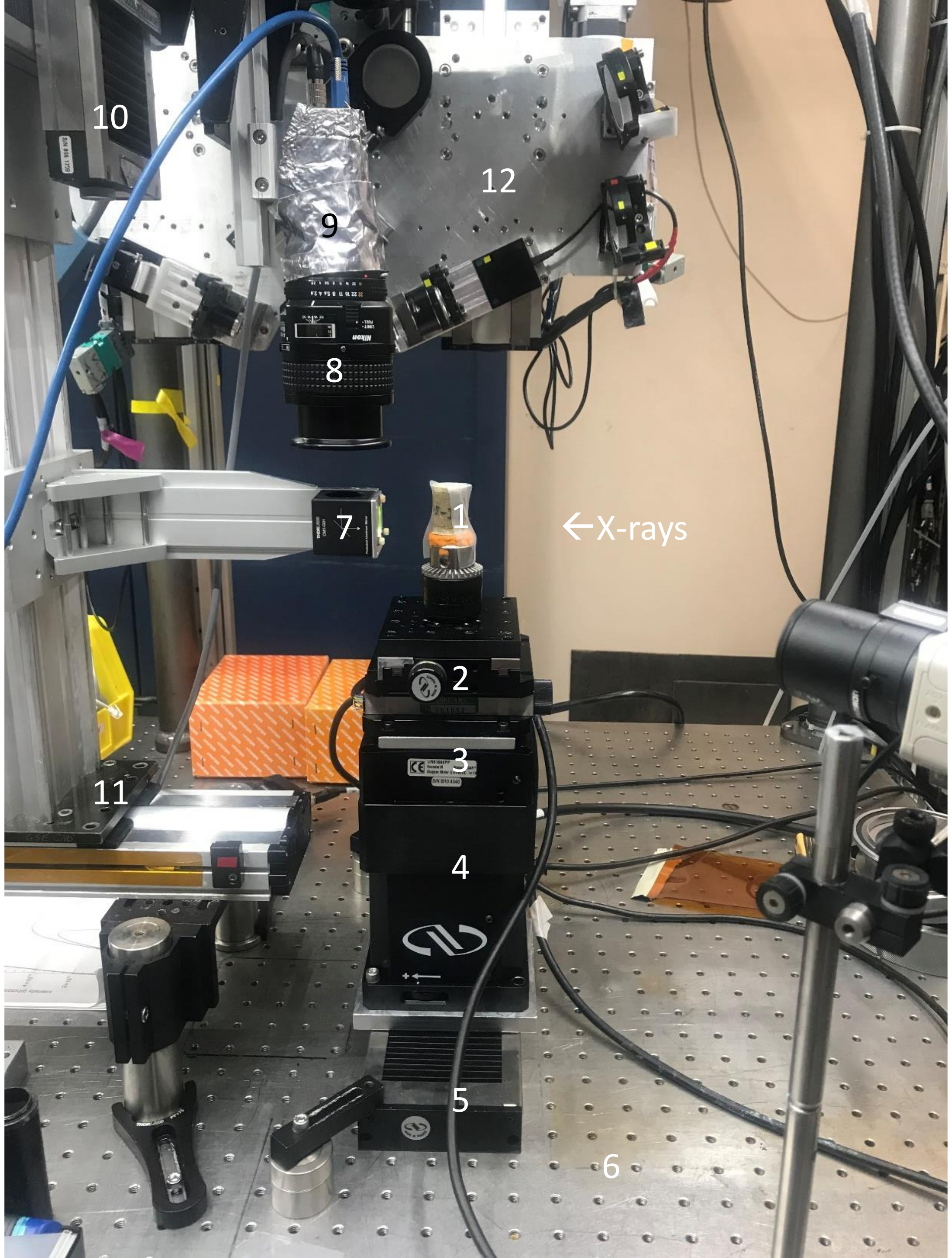




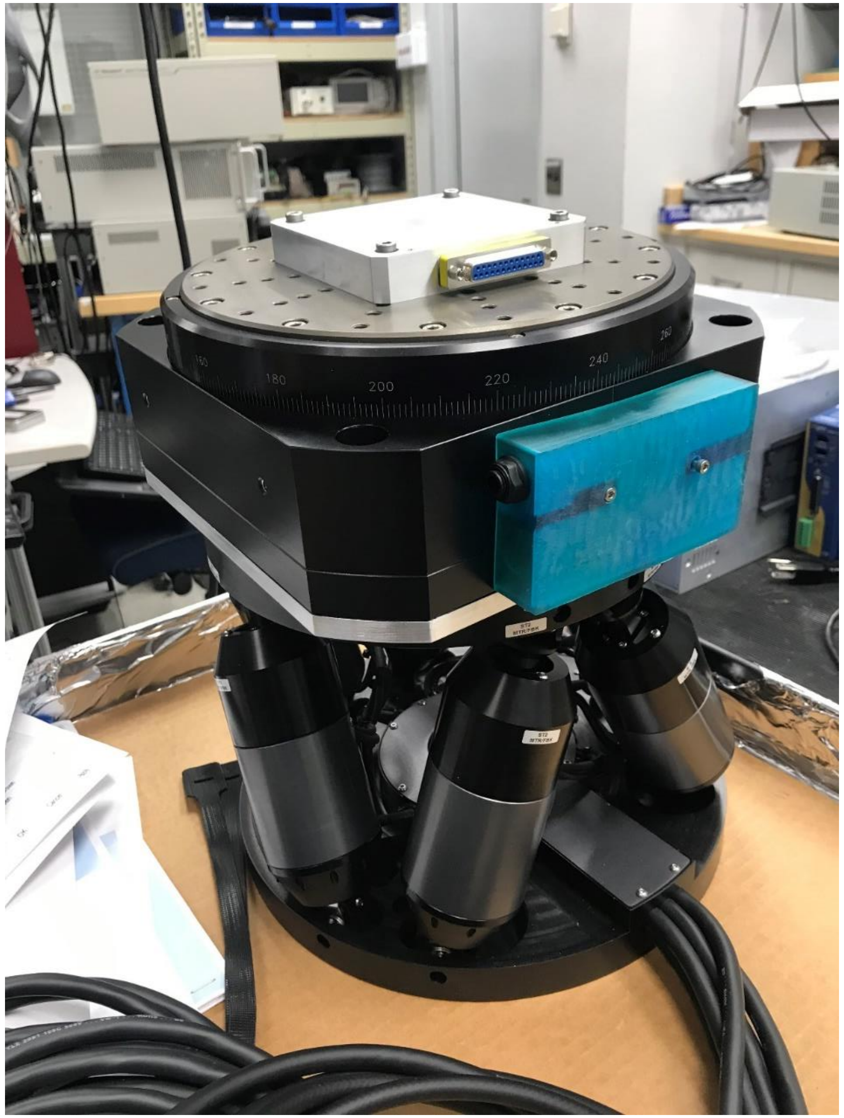
Flat field

Tomography Apparatus in 13-BM-D Before January 2021

1. Sample at x-ray beam height
2. X-Z translation stages above rotation stage, 25mm travel
3. Rotation stage
4. Vertical translation stage, 30 mm travel
5. Horizontal translation stage, 100 mm travel
6. Optical table, 5 degrees of freedom (X, Y, roll, pitch, yaw)
7. Scintillator and 45 degree mirror
8. Nikon macro lens (others lenses available for higher magnification)
9. CMOS camera, 1920x1200 pixels, 163 frames/s maximum
10. X-Y-Z-theta stage to position camera
11. Z stage to change scintillator to sample distance for phase contrast
12. Brillouin spectroscopy optics for diamond anvil cell, not used for tomography

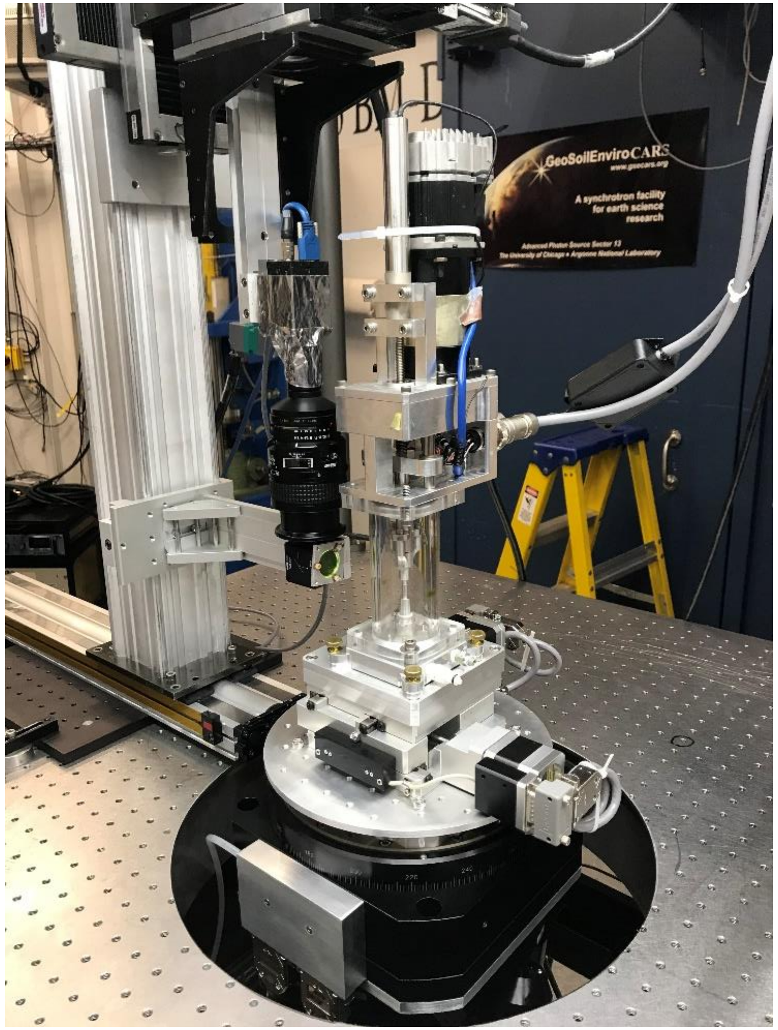


New Tomography Sample Stage

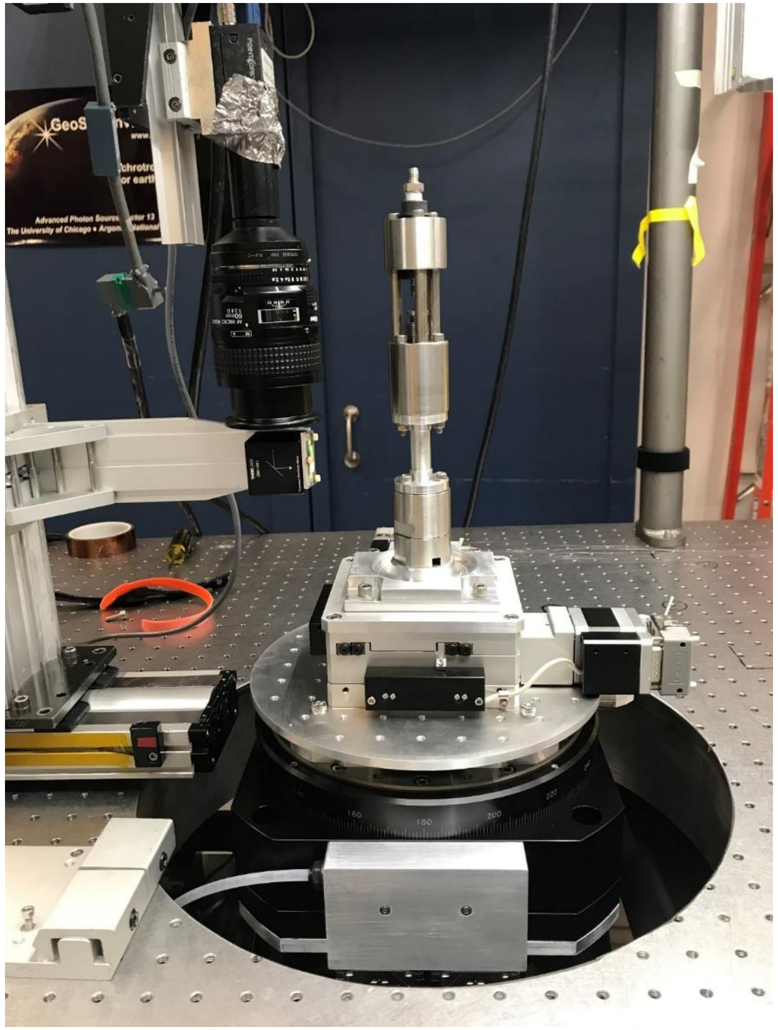


- Old stage
  - < 3 kg load
  - Maximum distance from pink beam to stage is ~75 mm
  - Cannot use large in-situ apparatus
  - Ball bearing stage, > 1 μm runout
- New stage
  - 25 kg load
  - Hexpod base, 6 degrees of freedom
  - Air bearing rotation stage, 0.25 μm runout
- Finished 2021-1 run with new stage February 26, 2021, greatly improved resolution and stiffness

In-situ Cells on New Stage

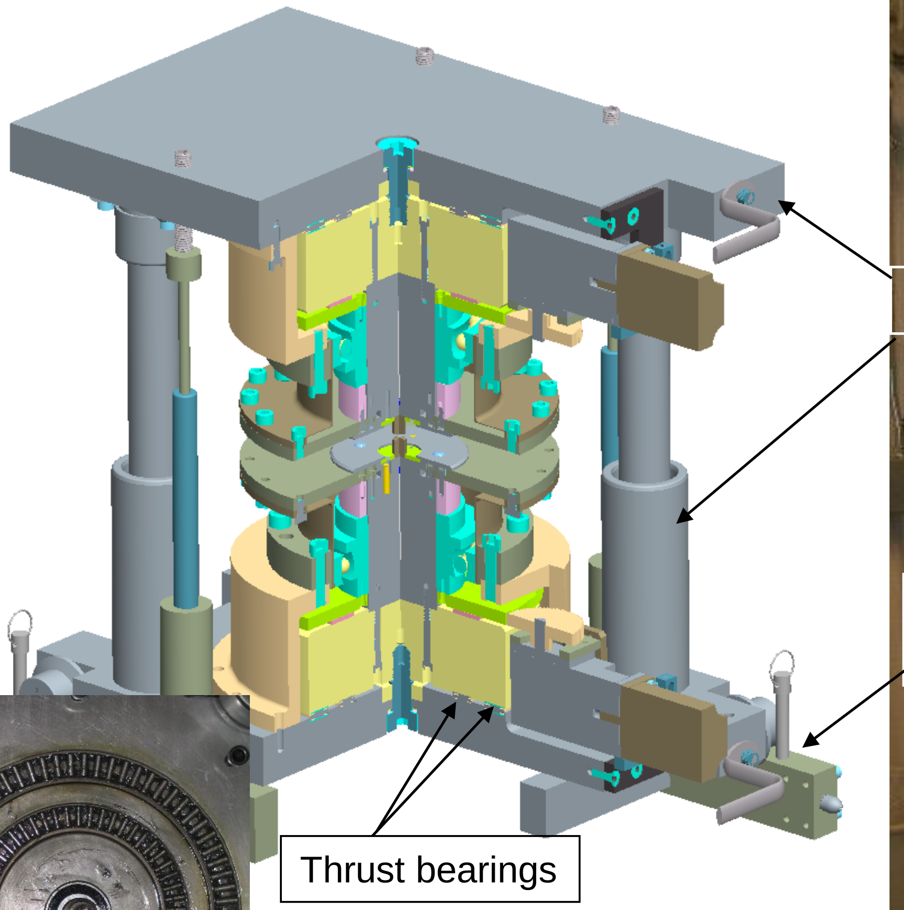


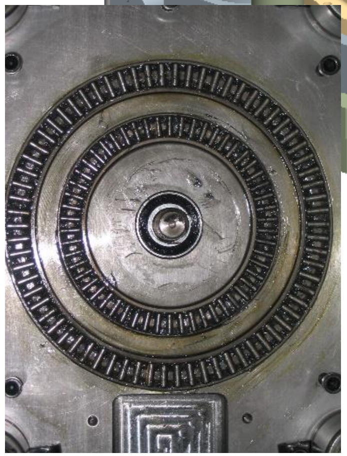
Uniaxial load cell



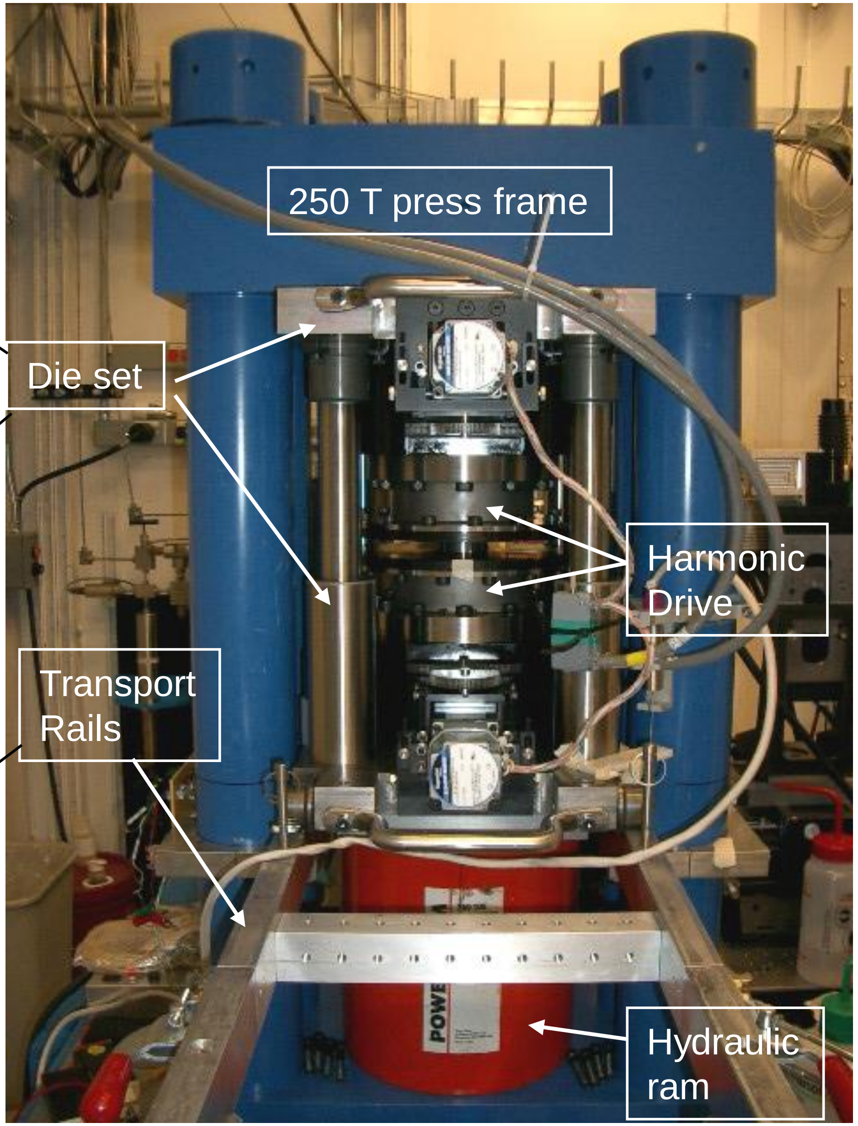
Triaxial high-pressure load cell

High-P tomography: Instrumentation





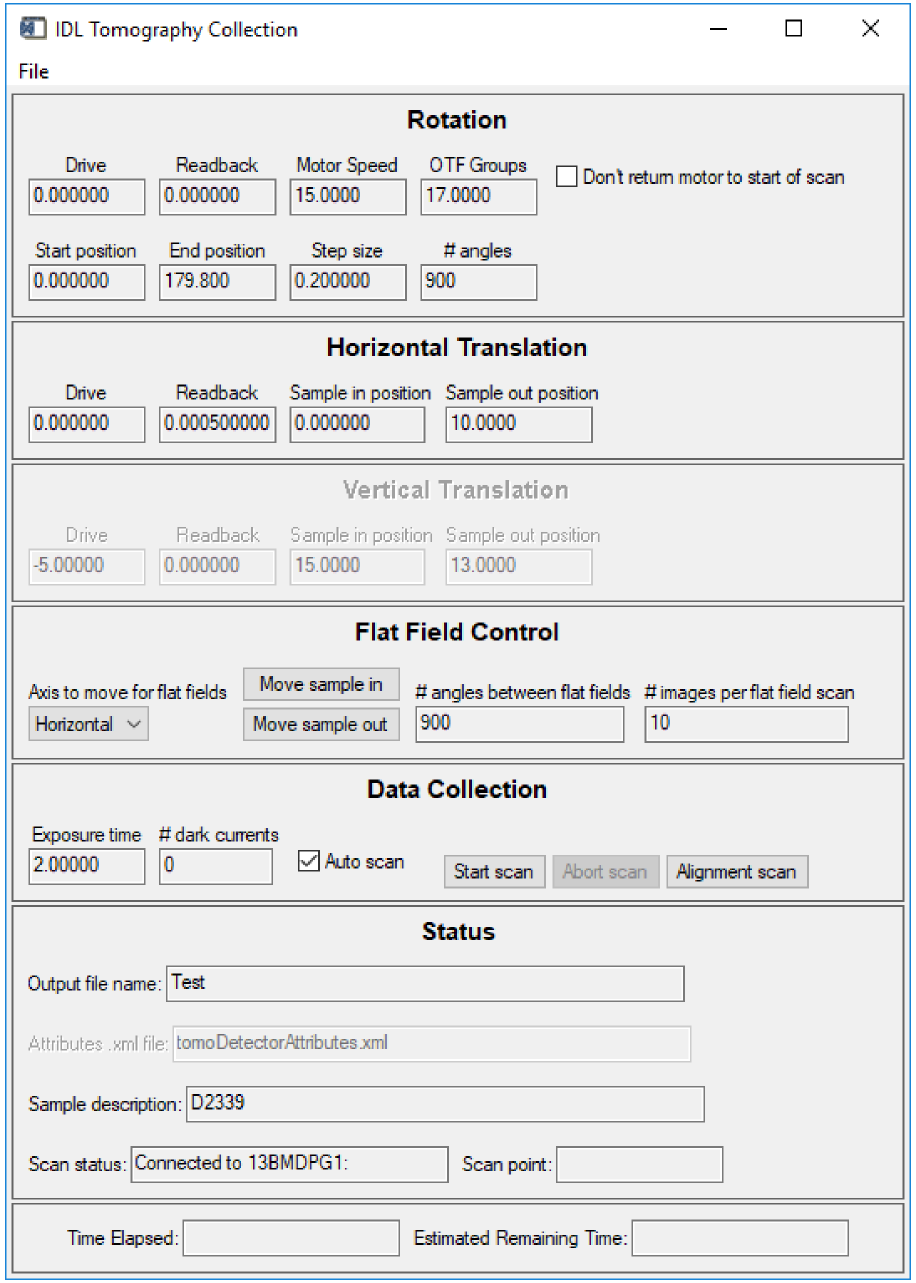
Max. load 50 tons

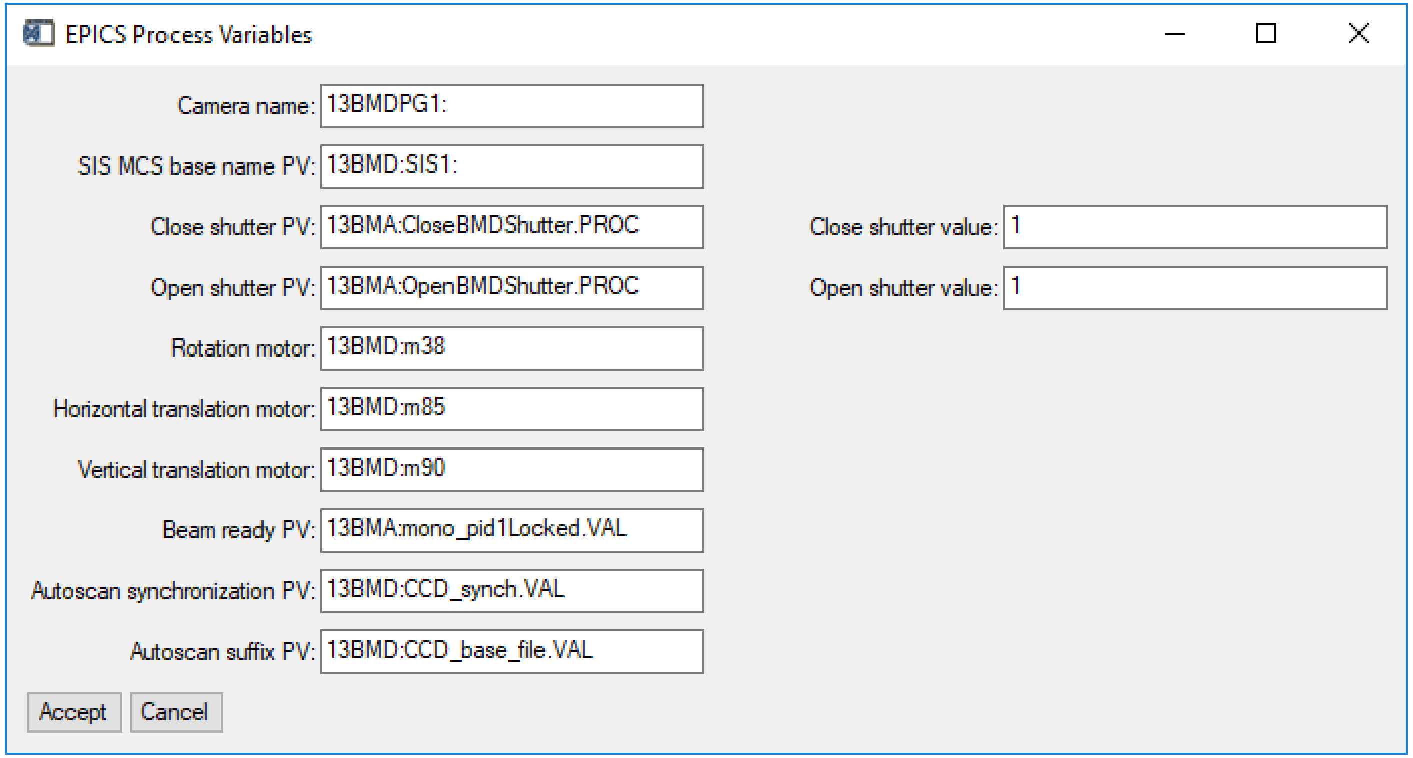


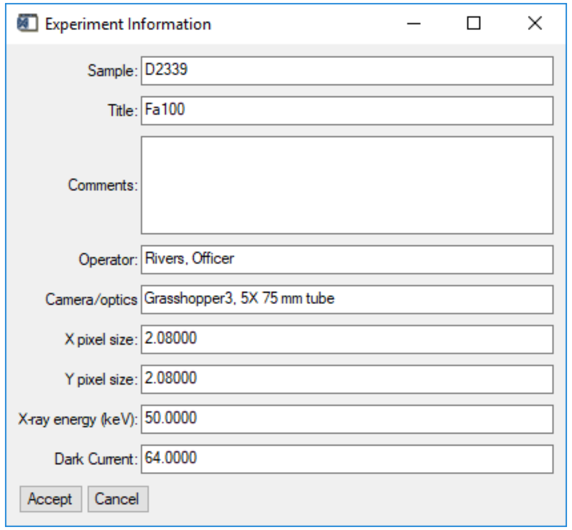
Tomography Data Collection History (13-BM-D)
TomoCollect

- Object-oriented code written in IDL
- Simple Graphical User Interface
- Started as step-scanning, but evolved to only on-the-fly scanning by 2014.
- Used successfully for 14 years from 2006-2020.

TomoCollect







TomoCollect Strengths

- Hardware trigger of detector based on rotation stage position
- Simple GUI very easy for users to learn, 1-2 hours to run independently.
- Small code, 2500 lines including GUI.
- Code functions as a **tomography scan server** that can be run from any EPICS client.
  - Its only job is to collect a single tomography dataset.
  - Knows nothing about beamline energy, sample height, sample temperature, etc.
  - Clients written any language (Python, IDL, etc.) control those parameters and then commands TomoCollect to collect a dataset.

TomoCollect Weaknesses

- The only thing controllable from EPICS was the file name and starting acquisition.
  - Could not script the exposure time, number of projections, location of rotation stage, etc.
- 13-BM-D was the only beamline using this software, no community development
- IDL is no longer popular, needed to be ported to Python.

Data Collection History (2-BM, 7-BM, 32-ID)
Python programs

Python scan programs were used on each of these beamlines

Weaknesses

- Not a clean object oriented design
- Programs grew organically with time, became very large and diverged for each beamline.
- Hard to maintain, changes made on one beamline could not be easily used on the others

TomoScan

New Python Scanning Software

- In April 2020 Francesco and I took advantage of the COVID shutdown at APS to devote time to developing new Python scanning software.
- Started with the 2-BM Python code, but did a major refactoring.

TomoScan
Architecture

- Beamline independent base classes
- Beamline dependent derived classes
- Functions as a “tomography scan server”, only job is to collect a single tomography dataset.
- All scan parameters are EPICS Process Variables (PVs)
  - Can be scripted from any client.
  - Can use any EPICS Operator Interface client (medm, CSS, caQtDM) as the GUI.
- Provides a simple EPICS IOC application with databases and OPI screens that can be used at any beamline.
- Runs on Linux or Windows.

TomoScan Assumptions and Limitations

- Designed to function only with the EPICS control system
- Assumes motors are using the EPICS motor record
- Assumes the detector is using the EPICS areaDetector package
- Currently only implements on-the-fly scanning (continuous rotation)
  - Step scanning will be implemented for 32-ID nanotomography
- No other assumptions about hardware or software

tomoscan.py
Primary base class

Methods

- move_sample_in(), move_sample_out()
- open_shutter(), close_shutter()
- set_exposure_time(), set_flat_exposure_time()
    - Copies the desired exposure time to the camera
- compute_frame_time()
    - Computes the minimum time between triggers based on the exposure time
    - Used to set the velocity of the rotation stage
- collect_dark_fields(), collect_flat_fields(), collect_projections()
- wait_camera_done()
    - Waits for a series of images to be collected, or an abort or timeout
- begin_scan(), end_scan(), abort_scan()
    - Performs operations that need to be done at the beginning and end of a scan, or when aborting a scan.
- fly_scan(), run_fly_scan()
- pv_callback()

tomoscan.py methods (continued)

fly_scan()

- Performs the operations for a tomography fly scan, i.e. with continuous rotation.
- This base class method does the following:
  - Moves the rotation motor to position defined by the RotationStart PV.
  - Calls begin_scan()
  - If the DarkFieldMode PV is ‘Start’ or ‘Both’ calls collect_dark_fields()
  - If the FlatFieldMode PV is ‘Start’ or ‘Both’ calls collect_flat_fields()
  - Calls collect_projections()
  - If the FlatFieldMode PV is ‘End’ or ‘Both’ calls collect_flat_fields()
  - If the DarkFieldMode PV is ‘End’ or ‘Both’ calls collect_dark_fields()
  - Calls end_scan
- If there is either CameraTimeoutError exception or ScanAbortError exception during the scan, it jumps immediate to calling end_scan() and returns.
- Derived classes generally do not need to override this method, but they are free to do so if required.

run_fly_scan()

- Runs fly_scan() in a separate thread
- pv_callback()

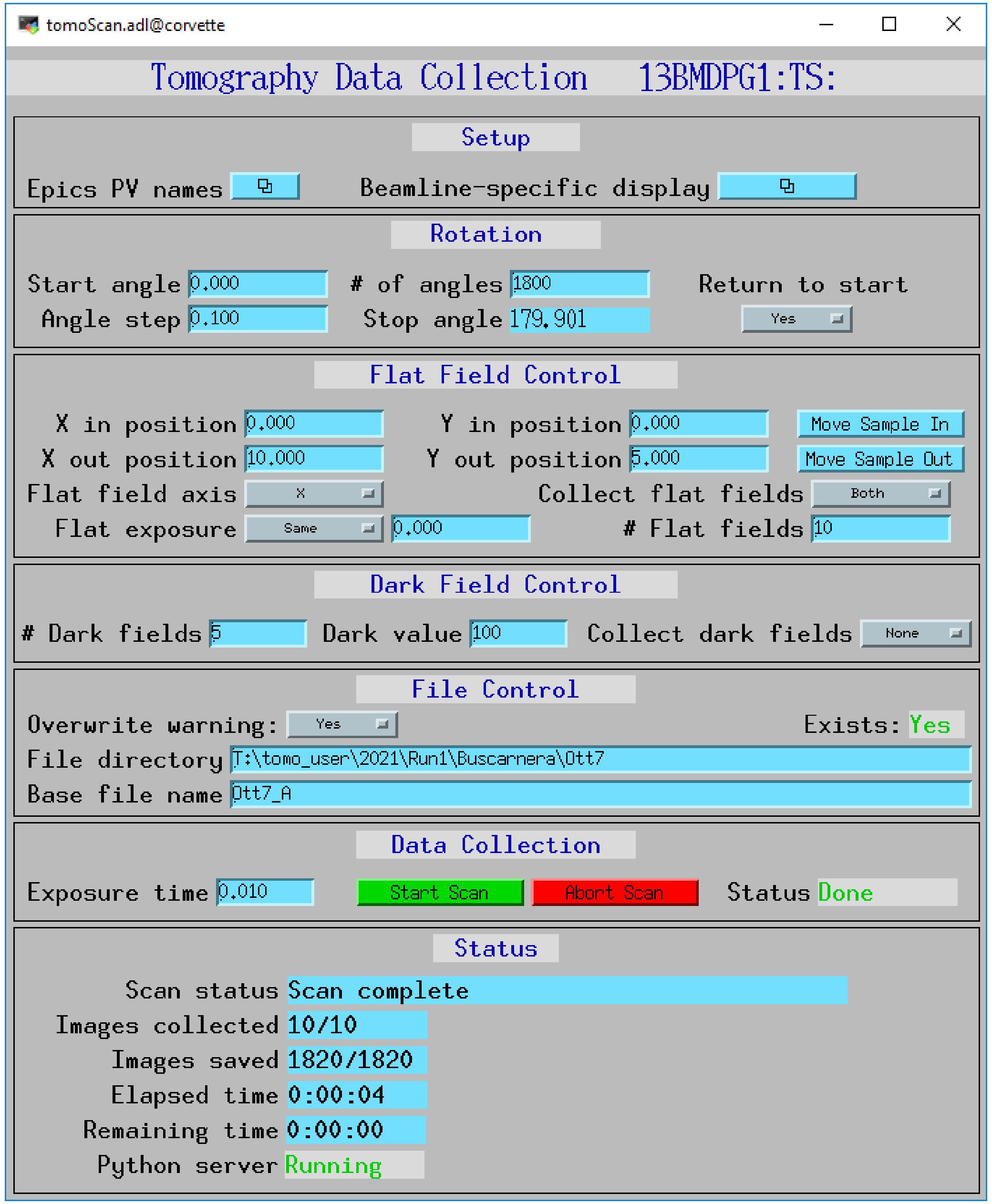
tomoscan.py Method (continued)

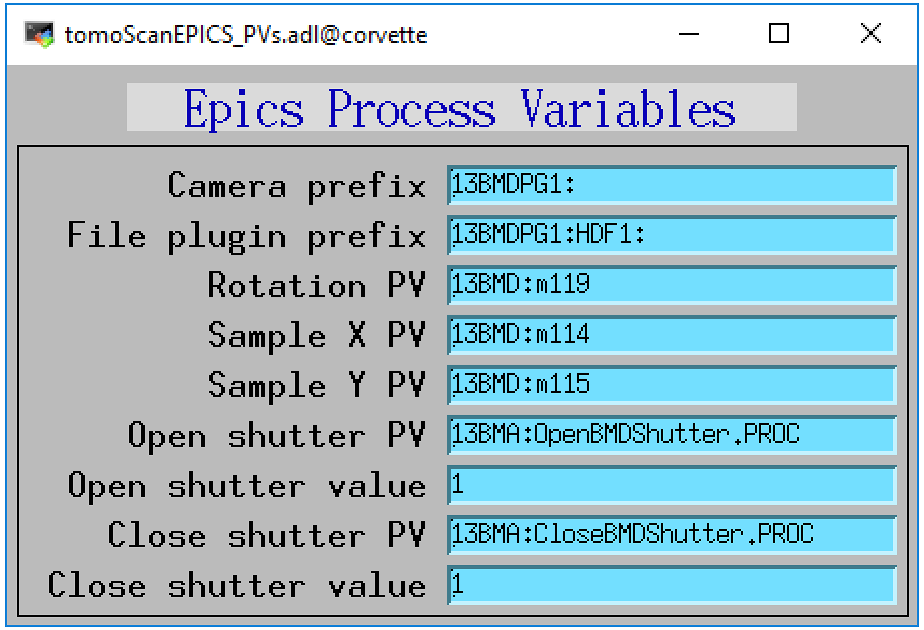
pv_callback()

- Callback function that is called by pyEpics when certain EPICS PVs are changed
- The PVs that are handled are:
  - StartScan : Calls run_fly_scan()
  - AbortScan : Calls abort_scan()
  - MoveSampleIn : Runs MoveSampleIn() in a new thread.
  - MoveSampleOut : Runs MoveSampleOut() in a new thread.
  - ExposureTime : Runs set_exposure_time() in a new thread.
  - FilePath : Runs copy_file_path() in a new thread.
  - FPFilePathExists : Runs copy_file_path_exists() in a new thread.

- ~900 lines of code

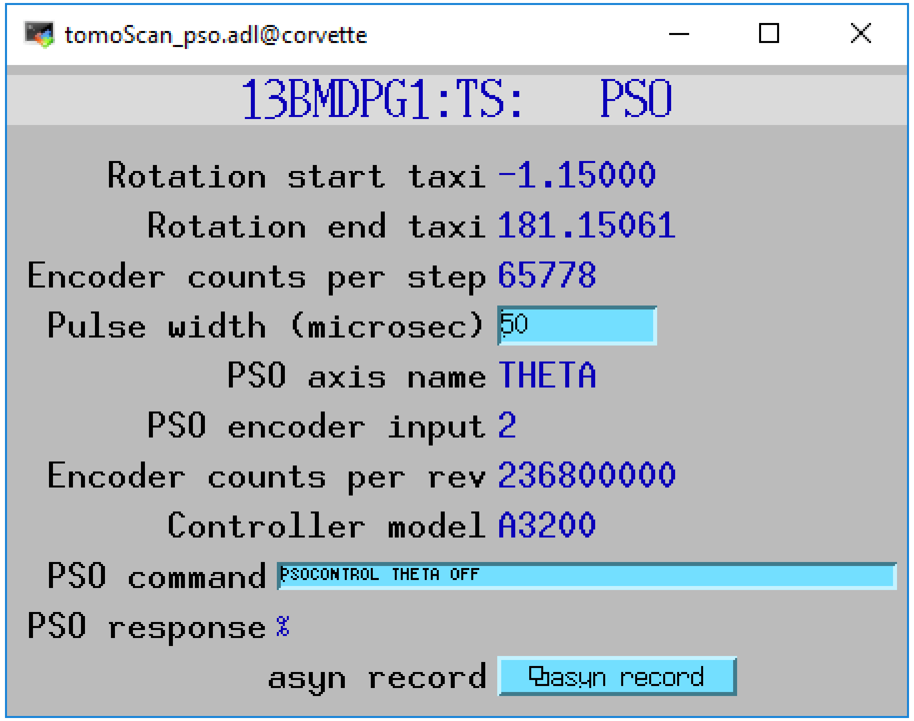
tomoscan base class medm screens





tomoscan_pso.py

- Intermediate base class for Aerotech rotation stages using Position Synchronized Output (PSO) to trigger detector
- Most APS tomography beamlines use Aerotech air-bearing rotation stages, so having a base class for this makes sense.
- Implements the methods to collect dark fields, flat fields and projections
- Uses the PSO output to trigger the detector based on projection interval
- Can program the pulse width for camera-specific requirements
- ~300 lines of code



Beamline Dependent Derived Classes

tomoscan_13bm_pso

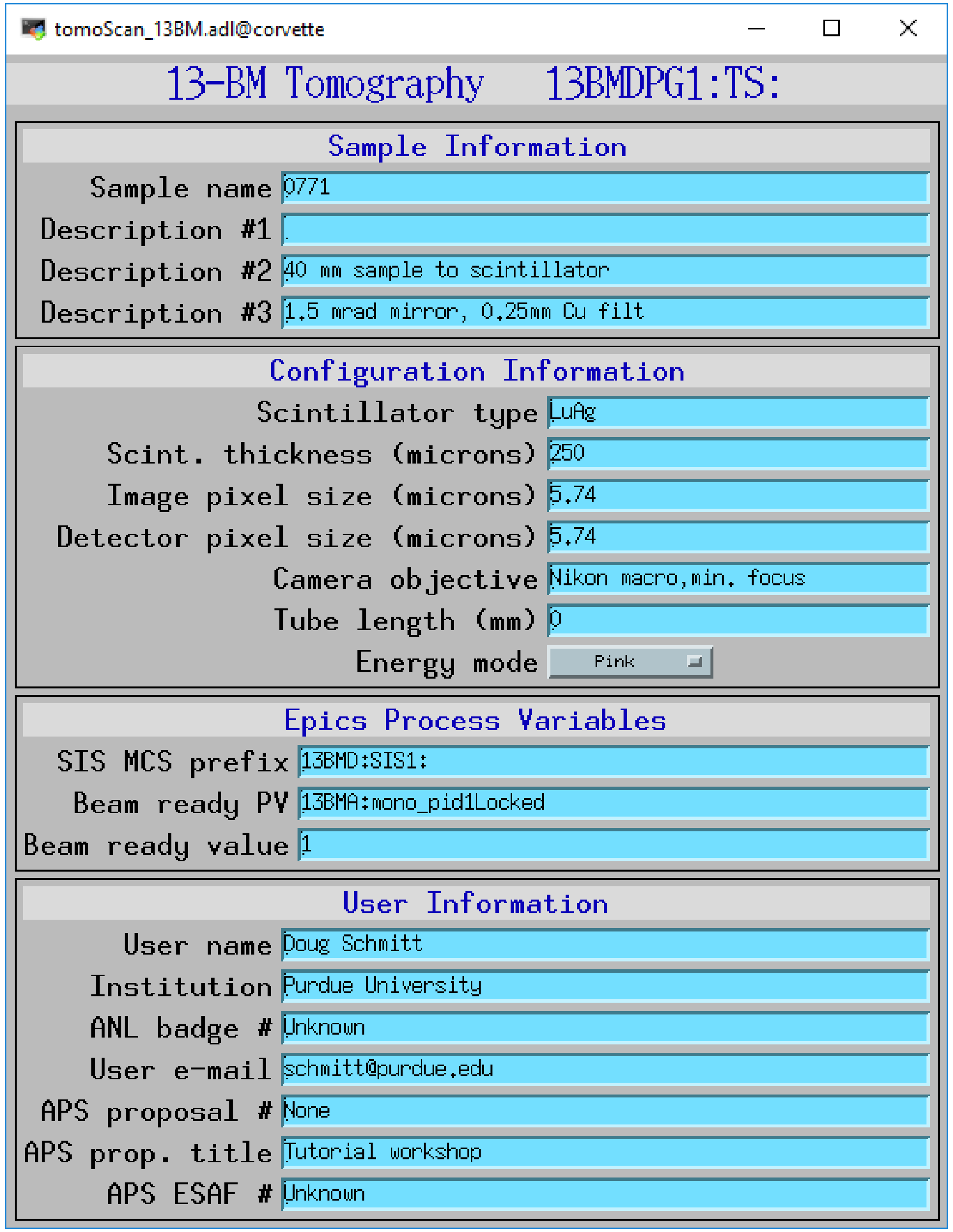
- Derived from tomoscan_pso class.
- Only implements set_trigger_mode() because our FLIR Grasshopper 3 camera needs to take 3 dummy images when switching from Internal Trigger to External Trigger.
- 76 lines of code.

tomoscan_13bm_mcs.py

- Implements methods that are specific to using an SIS3820 to divide stepper motor pulses by N for detector triggering.
- These methods do something beamline-specific and thencall the base-class version in many cases
- Used to be used for main tomography data collection, but that now uses Aerotech rotation stage and PSO version above.
- Used for high-pressure tomography
- 247 lines of code.

- Also beamline-dependent classes for 2-BM-A, 2-BM-B, and 7-BM.

13-BM Beamline-specific medm screen



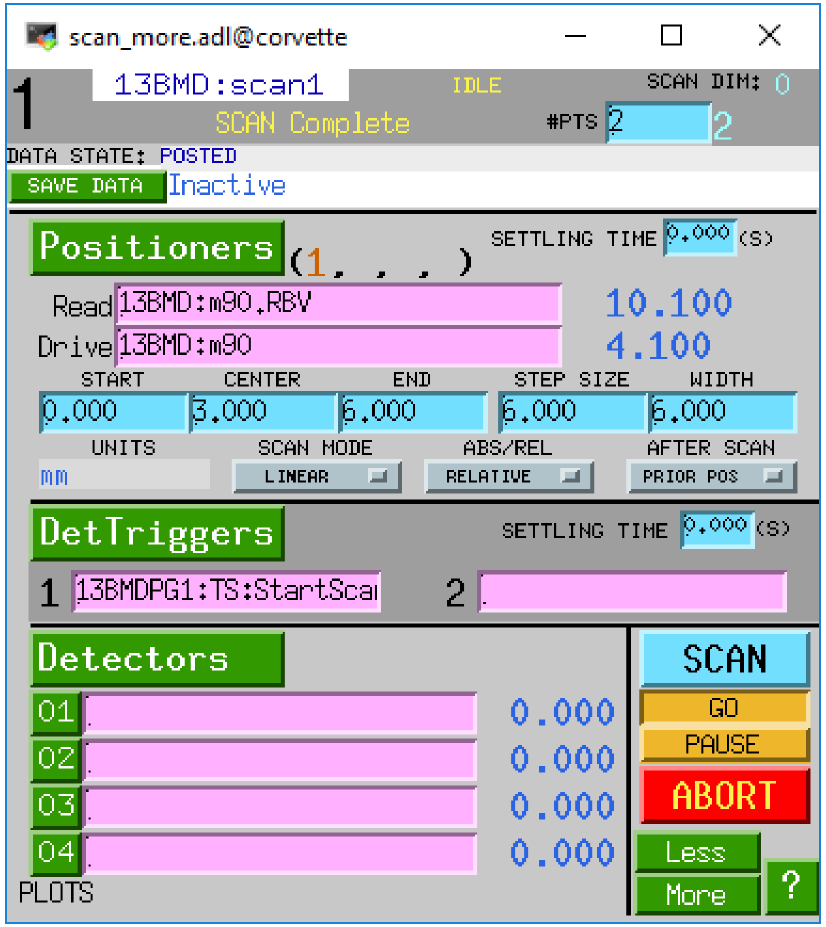
- Metadata is saved for both user-entered information shown here, as well as many EPICS PVs for the state of the storage ring, beamline, sample stage, etc.
- Can add additional metadata for a specific experiment (temperature, etc.)

Scanning

- Any EPICS client can change the tomoscan scan parameters (file name, exposure time, etc.) and then write 1 to the StartScan PV to perform a complete tomography scan.
- StartScan is an EPICS “busy” record so ca_put_callback will not return until the scan is complete, including the file-writer having finished writing all data.

Scanning with EPICS scan record

- Very mature tool
- EPICS scan record can scan any EPICS PV and collect a tomography dataset at each point in the scan.
- Vertical sample position scanned here
- Could scan monochromator energy, sample temperature, etc.



Scanning with Python script

```python
import epics
def scan_demo(tomo_prefix, exposure_time, scan_pv, start, step, points):
    """Demonstrates collecting a series of tomography datasets while scanning an EPICS PV.

       Parameters
       ----------
       tomo_prefix : str
           The EPICS PV prefix for the tomoScan database
       exposure_time : float
           The exposure time per projection for the tomography datasets
       scan_pv : str
           The name of the EPICS PV to scan
       start : float
           The starting value for the scanned PV
       step : float
           The step size for the scanned PV
       points : int
           The number of points in the scan
    """

    epics.caput(tomo_prefix + 'ExposureTime', exposure_time, wait=True)
    file_plugin_prefix = epics.caget(tomo_prefix + 'FilePluginPVPrefix')
    # Set the initial file number back to 1 and make sure AutoIncrement is enables
    epics.caput(file_plugin_prefix + 'FileNumber', 1)
    epics.caput(file_plugin_prefix + 'AutoIncrement', 'Yes')

    for i in range(1, points+1):
        epics.caput(scan_pv, start + step*i, wait=True)
        epics.caput(tomo_prefix + 'StartScan', 1, wait=True, timeout=100)
        print('Completed dataset %s' % epics.caget(file_plugin_prefix + 'FullFileName_RBV', as_string=True))
```

Streaming model with Communication via EPICS pvAccess

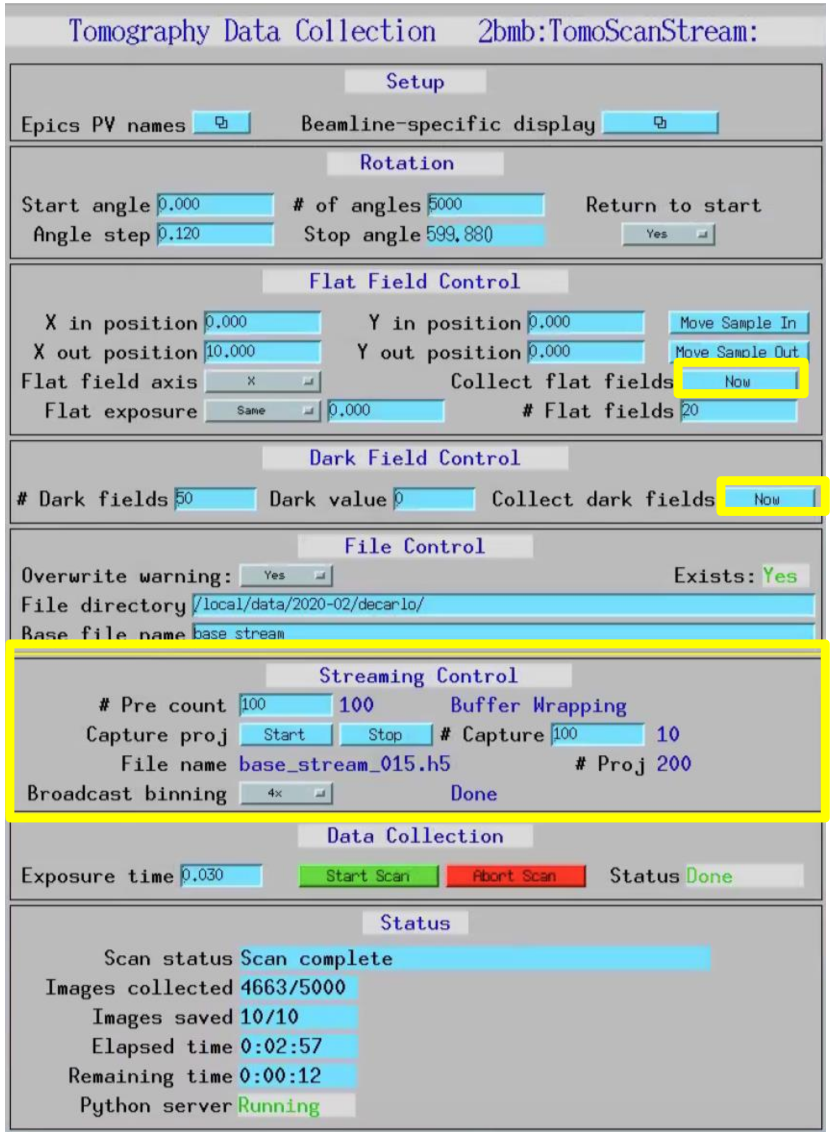
1 . Detector machine

EPICS AreaDetector

- preprocessing projections
- capture to an hdf5
- circular buffer
- broadcasting projections

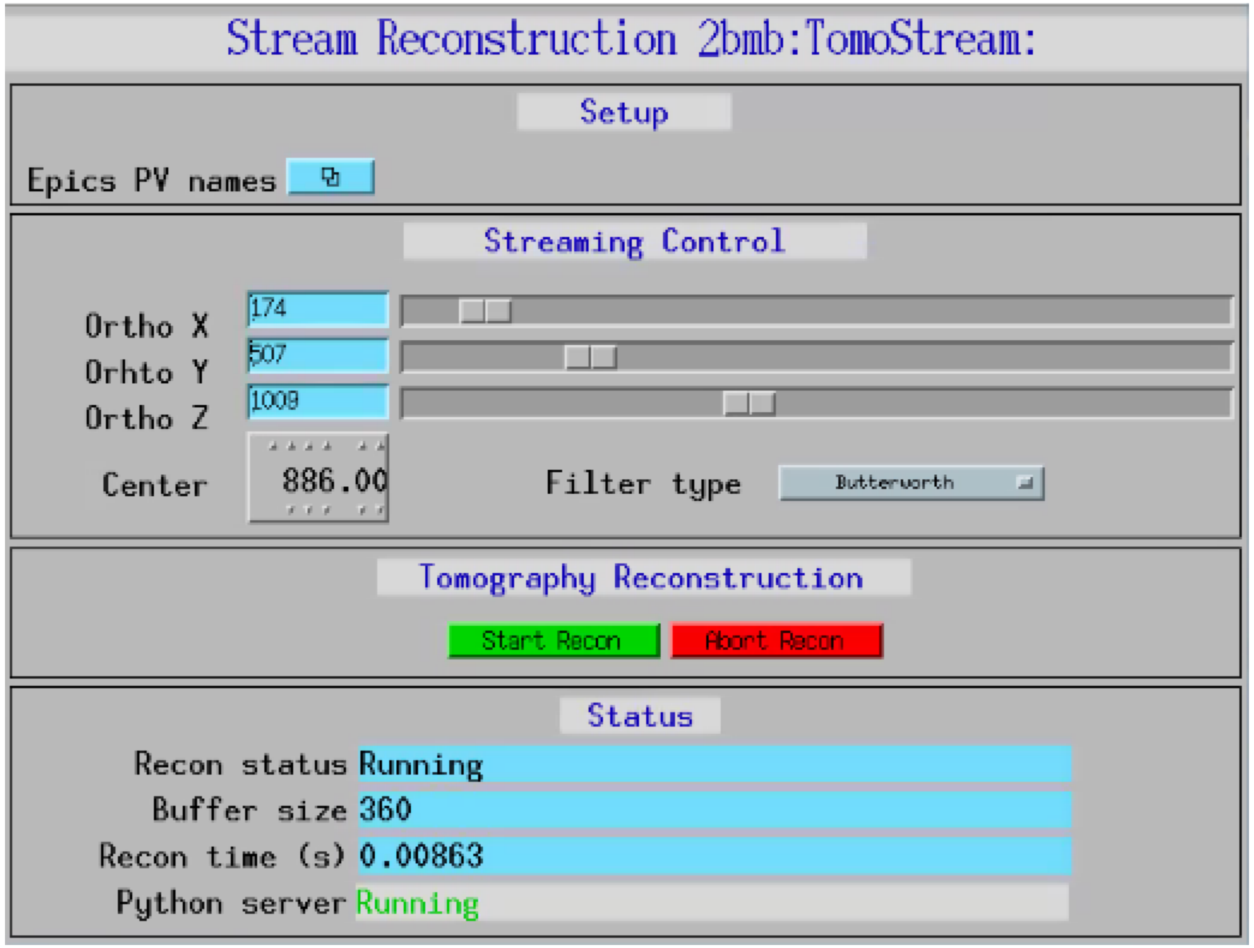
TomoScanStream(Tomoscan)

- scanning control
- data capturing control
- broadcasting (binned) darks/flats/angles with pvAccess



Streaming model with Communication via EPICS pvAccess

2. Processing machine with GPU



Tomostream

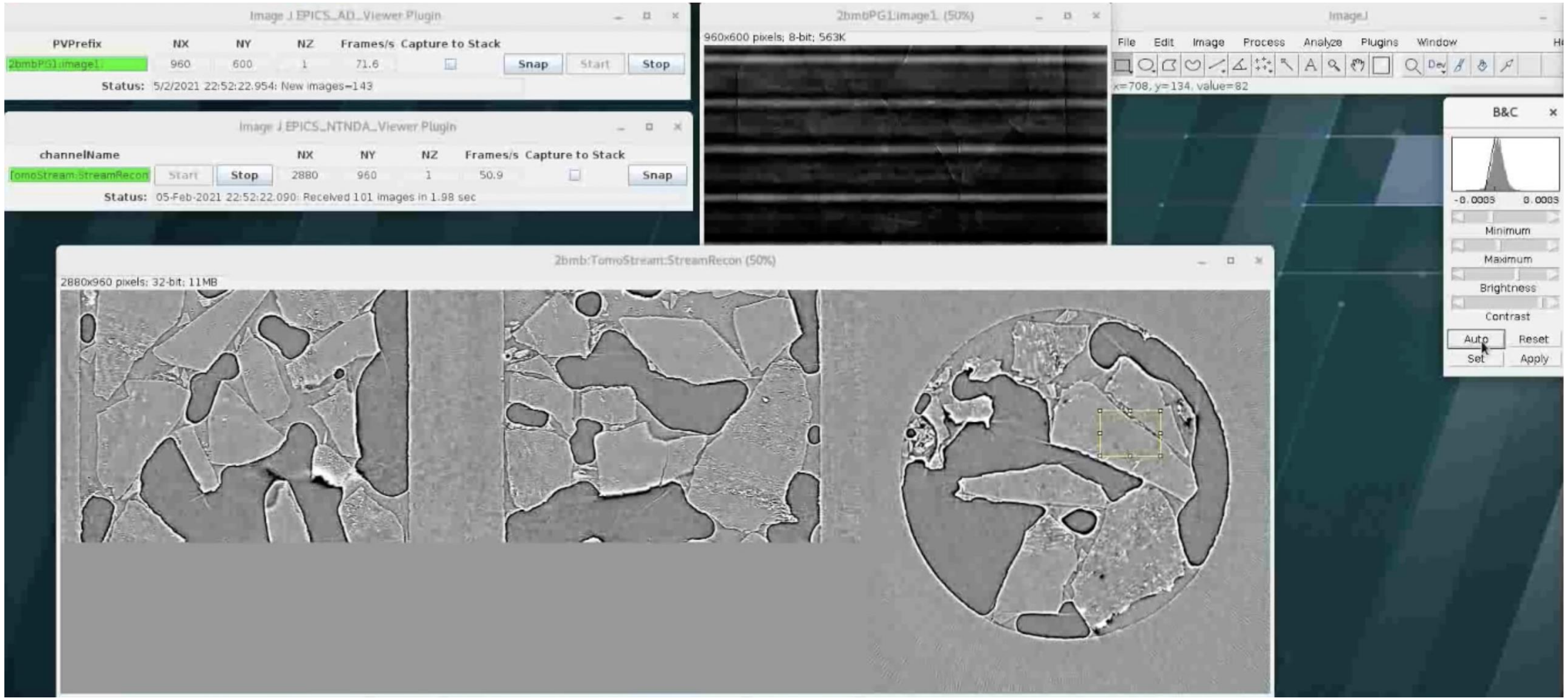
- ortho-slice reconstruction (3 slices)
- broadcasting reconstructions with Channel Acces and pvAccess

Streaming model with Communication via EPICS pvAccess

3. Observer machine

Visualization of projections/reconstructions

Uses ImageJ with ADViewer or NTDAViewer plugins



TOMOSCAN+TOMOSTREAM MODEL

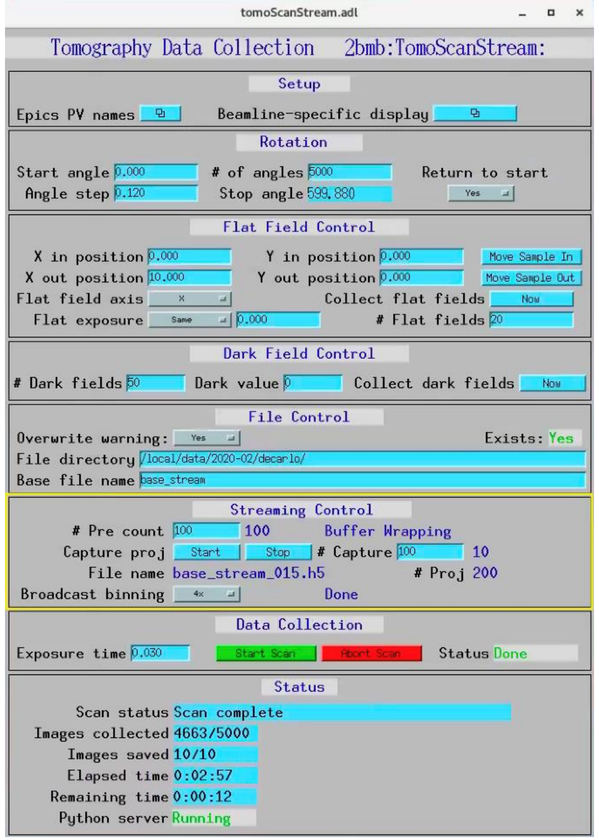
Highlights

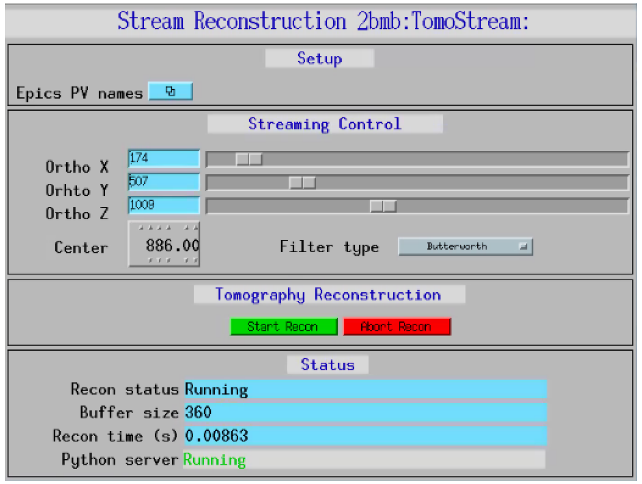
Streaming data

1. Continuous data collection
2. Capture projections to hdf5 file on demand
3. Circular buffer to store projections for some period
4. Re-take flat/dark fields on demand
5. Broadcasting projections, darks, and flats via network
6. Visualization of projections in ImageJ

Streaming reconstruction

1. Real-time orthogonal slices reconstruction
2. Broadcasting reconstruction via network
3. Visualization of reconstructions in ImageJ





NEW OPPORTUNITIES WITH STREAMING

- Real-time alignment of the acquisition system
- Real-time positioning of the sample
- Real-time adjustment of acquisition parameters
- Real-time monitoring of sample changes
- Focusing to the regions of interest
- Saving data only when the dynamic process occurs
- Use of Machine Learning techniques to automatically detect sample changes, apply segmentation and quantitative analysis

https://tomostream.readthedocs.io/

Thanks for Your Attention !!!