

Calibrated Area Detector

This script lives in `/sdf/group/lcls/ds/ana/tutorials/psana1_examples/areaDetAccess.py` and demonstrates how to access calibrated "unassembled data" (3D array, no geometry applied) and "assembled images" (2D array, geometry applied). Assembled/unassembled are only different for multi-panel detectors. For monolithic 2D area detectors they are the same.

NOTE: there are many other python methods of the Detector object for Area Detectors (e.g. x,y pixel coordinates, pedestal values, masks). One can see all the methods of a class using [ipython tab completion](#), or looking at more complete documentation [here](#).

```
from psana import *
ds = DataSource('exp=xpptut15:run=54:smd')
det = Detector('cspad')
for nevent, evt in enumerate(ds.events()):
    # includes pedestal subtraction, common-mode correction, bad-pixel
    # suppression, and returns an "unassembled" 3D array of cspad panels
    calib_array = det.calib(evt)
    # this is the same as the above, but also uses geometry to
    # create an "assembled" 2D image (including "fake pixels" in gaps)
    img = det.image(evt)
    break
import matplotlib.pyplot as plt
plt.imshow(img, vmin=-2, vmax=2)
plt.show()
```

The "image" method can also be used to convert any array with the "unassembled" shape into "assembled" format (that is, applying the geometry) with a line like this:

```
assembled_array = det.image(evt, unassembled_array)
```

This can be useful for plotting masks, or determining which pixels are "real" in the assembled image:

```
real_pixels = det.image(evt, np.ones_like(unassembled_array))
```

NOTE: `det.image(evt)` maps pixels to a uniform 2D grid. For detectors with multiple physical panels this is approximate due to panel tilts, and when viewing these 2D images you may notice "jumps" in pixel locations due to the mapping of the pixels to the uniform grid. If your analysis is sensitive to these effects it is more precise to use `det.calib()` and the individual pixel coordinates return by expert methods described here: <https://lcls-psana.github.io/Detector/index.html#module-AreaDetector>.

NOTE: as of psana version 0.19.0 the behavior of the area detector "calib" and "image" methods have been changed. The new, more sustainable, policy is these methods will return the best corrections possible, which can change with new psana releases. As an example, in ana-0.19.0 we include per-pixel gain corrections (when available from flat-field data) and the configurable 6.85x per-pixel gain correction for the CSPAD (measured with low charges). If you would like your corrections to not change with new releases, you can "freeze" at a specific release by activating the appropriate conda environment.

NOTE: A table showing which corrections are applied to which detectors is shown on this page: [Method det.calib algorithms](#)

NOTE: There are two conventions for mapping pixels to an image, called "Cartesian" (row/column array indices are x/y) and "matrix" (row/column array indices are y/x) which are transposes of each other, as discussed here: <https://eli.thegreenplace.net/2014/meshgrids-and-disambiguating-rows-and-columns-from-cartesian-coordinates/>. psana uses the Cartesian convention. This is shown by the following example. If we used the matrix convention then the size of the x/y shapes would be flipped.

```
(ana-4.0.43) psanagpu101:~$ cat junk.py
from psana import *
runnum = 610
ds = DataSource('exp=xpptut15:run='+str(runnum))
det = Detector('jungfrau4M')
im_x = det.image_xaxis(runnum)
im_y = det.image_yaxis(runnum)
for evt in ds.events():
    img = det.image(evt)
    break
print(f'Image shape: {img.shape}, X shape: {im_x.shape}, Y shape: {im_y.shape}')
(ana-4.0.43) psanagpu101:~$ python junk.py
('Image shape: (2203, 2299), X shape: (2203,), Y shape: (2299,))
(ana-4.0.43) psanagpu101:~$
```

References

- [AreaDetector](#) - documentation of methods of the "Area" Detector interface.
 - [calib\(\)](#)
 - [image\(\)](#)
 - [Method det.calib algorithms](#)