

Calibration algorithms - development notes

Content

- [Content](#)
- [pnCCD overview](#)
- [Development of calibration of pnCCD](#)
 - [2014-01-22 Meeting minutes](#)
 - [Script form Chris](#)
 - [Walking and talking about unlimited pipeline \(processing\)](#)
 - [CASS Heritage](#)
 - [Online monitor](#)
 - [Data for tests](#)
 - [New modules for "old-style" calibration:](#)
 - [Detector-dependent interface](#)
 - [Pros](#)
 - [Cons](#)
 - [Detector-independent interface](#)
- [Implementation of CalibParsStore](#)
- [Module pdscalibdata::NDArriOV1](#)
 - [New approach to calibration files with header](#)
- [pnCCD analysis for users](#)
- [References](#)

pnCCD overview

[Large area pnCCD DAQ and Electronics, Lothar Struder & Robert Hartmann](#)

Development of calibration of pnCCD

2014-01-22 Meeting minutes

Hi everyone,

Here is a short summary of what I heard today for how we should start with the pnCCD.

For pnCCD algorithms:
common-mode, pedestals, hot-pixels, quadrant rotations, hit-finders,
support in mikhaïl's calibManager

For pnCCD online displays: (using matplotlib for now)
shot by shot raw data
shot by shot calibrated data
projections of the above
region-of-interest
strip-charts of interesting quantities
(also display calibration values like noise-map,pedestal-map)

After this we will work on the acqiris as well (acqiris
constant-fraction algos already exist in psana).

Attached below is a 12 line python program that plots a real pnCCD image and an x-projection (amoc0113 also has pnccd data we can look at). You can run it on a psana node by saving it to pnccd.py and doing "sit_setup" and then "ipython pnccd.py". This sort of code should work online too (although we may have to change matplotlib settings) as well as with calibrated images.

Display group (dan, mikhaïl, me) meets Thursday at 10:30. Analysis group (sebastian, ankush(?), phil, mikhaïl, me) meets Friday at 1.

See you then...

chris

Script form Chris

Use interactive psana framework ~cpo/ipsana/shm.py:

```
from psana import *

events = DataSource('shmem=1_1_XCS.0').events()
src = Source('DetInfo(XcsBeamline.1:Tm6740.5)')
import matplotlib.pyplot as plt

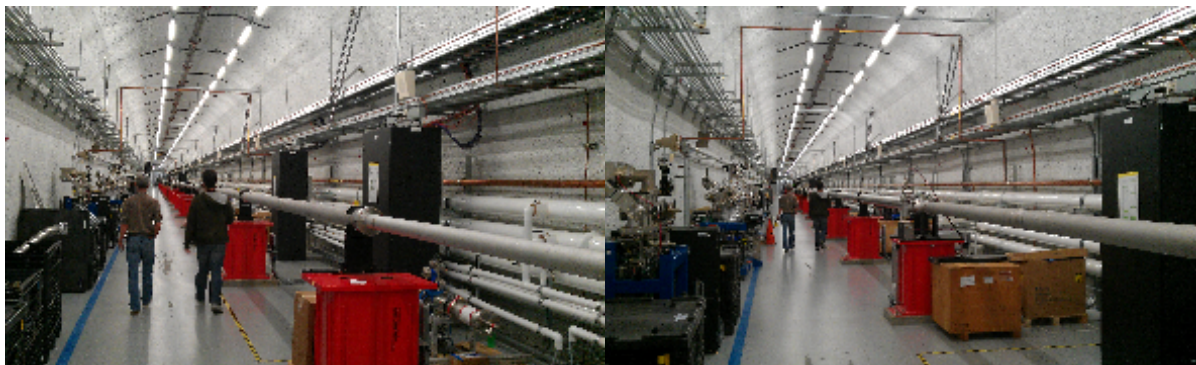
plt.ion()
fig = plt.figure('pulnix')
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # x0, y0, h, w

for i in range(100):

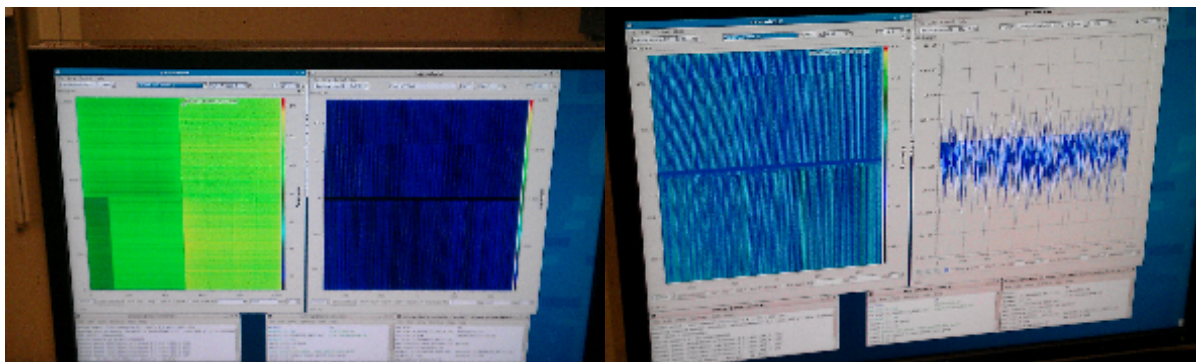
    evt = events.next()
    frame = evt.get(Camera.FrameV1, src)

    ax.cla()
    ax.imshow(frame.data16())
    fig.canvas.draw()
```

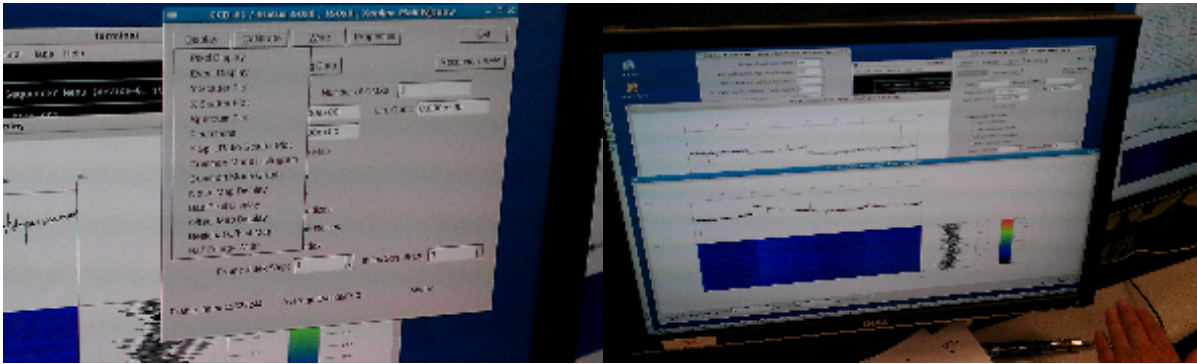
Walking and talking about unlimited pipeline (processing)



CASS Heritage



Online monitor



Data for tests

On 2014-01-27 Sebastian Carron kindly provide us with data files for pnCCD experiment amoa1214:

- Dark Run: 169, rear sensors gain 1/64, front 1/1, Imaging mode exp=amoa1214:run=169
- Run With Hits: 170 Low hit rate though, so you will have to use a hit finder of sorts exp=amoa1214:run=170

New modules for "old-style" calibration:

- `pdscalibdata/include/PnccdBaseV1.h` - baseclass for pnCCD parameters, defines Segs, Rows, Cols, Size
- `pdscalibdata/include/PnccdPedestalsV1.h` - loads pedestals from file, returns ndarray of pedestals
- `pdscalibdata/include/PnccdCommonModeV1.h` - the same for common mode
- `pdscalibdata/include/PnccdPixelGainV1.h` - the same for pixel gain
- `pdscalibdata/include/PnccdPixelRmsV1.h` - the same for pixel rms
- `pdscalibdata/include/PnccdPixelStatusV1.h` - the same for pixel status
- `PSCalib::PnccdCalibPars` - wrapper for all pnCCD types

Detector-dependent interface

Example can be found in `PSCalib/test/ex_calib_file_finder.cpp`:

```
// Assume that file is located in /reg/d/psdm/AMO/amtut13/calib/PNCCD::CalibV1/Camp.0:pnCCD.1/pedestals/1-end.
data

#include "PSCalib/PnccdCalibPars.h"

const std::string calib_dir = "/reg/d/psdm/AMO/amtut13/calib";
const std::string group = "PNCCD::CalibV1"; // or std::string()
const std::string source = "Camp.0:pnCCD.1";
unsigned long runnum = 10;
unsigned print_bits = 255;

PSCalib::PnccdCalibPars *calibpars = new PSCalib::PnccdCalibPars(calib_dir, group, source, runnum,
print_bits);

calibpars->printCalibPars();
ndarray<CalibPars::pedestals_t, 3> peds = calibpars -> pedestals_ndarr();
ndarray<CalibPars::common_mode_t, 1> cmod = calibpars -> common_mode_ndarr();
ndarray<CalibPars::pixel_status_t, 3> stat = calibpars -> pixel_status_ndarr();
ndarray<CalibPars::pixel_gain_t, 3> gain = calibpars -> pixel_gain_ndarr();
ndarray<CalibPars::pixel_rms_t, 3> gain = calibpars -> pixel_gain_ndarr();

// OR:
CalibPars::pedestals_t* p_peds = calibpars -> pedestals();
CalibPars::common_mode_t* p_cmod = calibpars -> common_mode();
CalibPars::pixel_status_t* p_stat = calibpars -> pixel_status();
CalibPars::pixel_gain_t* p_gain = calibpars -> pixel_gain();
CalibPars::pixel_rms_t* p_rms = calibpars -> pixel_rms();

const size_t ndim = ndim();
const size_t size = size();
const unsigned* shape = shape();
etc...
```

Pros

- Simple format for calibration files - just a text file with pre-defined number of values for each type:

```
973.941639 881.189675 1050.211 773.263749 899.241302 981.805836 1150.72615 993.084175 1121.15488 1029.76319
1220.14927 903.278339 1097.49944 1066.94949 1263.71044 1053.53872 1194.35915 935.320988 1317 ...
```

Cons

- Too simple calibration file format, does not allow any metadata or comments.
- Detector-dependent objects and parameters "knows" about parameters' array type and shape:
 - PSCalib::PncddCalibPars which depends on PncddPedestalsV1, PncddCommonModeV1, ..., PncddBaseV1
 - pdscalibdata::PncddPedestalsV1::pars_t = float
 - pdscalibdata::PncddCommonModeV1::pars_t = uint16_t
 - pdscalibdata::PncddPixelStatusV1::pars_t = uint16_t
 - pdscalibdata::PncddPixelGainV1::pars_t = float
 - const std::string groupName = "PNCCD::CalibV1"; - **do we really need it ?**

Detector-independent interface

- Interface is declared in the abstract base class PSCalib::CalibPars
- Access to all detector-dependent classes is hidden in the static factory class PSCalib::CalibParsStore



Factory is implemented for pnCCD only. CSPAD and CSPAD2x2 will be added soon.

```
#include "PSCalib/CalibPars.h"
#include "PSCalib/CalibParsStore.h"

// Instatiation
//Here we assume that code is working inside psana module where evt and env variables are defined through input
parameters of call-back methods.
//Code below instateates calibpars object using factory static method PSCalib::CalibParsStore::Create:

std::string calib_dir = env.calibDir(); // or "/reg/d/psdm/<INS>/<experiment>/calib"
std::string group = std::string(); // or something like "PNCCD::CalibV1";
const std::string source = "Camp.0:pnCCD.1";
const std::string key = ""; // key for raw data
Pds::Src src; env.get(source, key, &src);
PSCalib::CalibPars* calibpars = PSCalib::CalibParsStore::Create(calib_dir, group, src, PSCalib::getRunNumber
(evt));

// Access methods
calibpars->printCalibPars();
const PSCalib::CalibPars::pedestals_t* peds_data = calibpars->pedestals();
const PSCalib::CalibPars::pixel_gain_t* gain_data = calibpars->pixel_gain();
const PSCalib::CalibPars::pixel_rms_t* rms_data = calibpars->pixel_rms();
const PSCalib::CalibPars::pixel_status_t* stat_data = calibpars->pixel_status();
const PSCalib::CalibPars::common_mode_t* cmod_data = calibpars->common_mode();
```

Implementation of CalibParsStore

[Doxy doc for CalibParsStore](#)

Module pdscalibdata::NDArrIOV1

New approach to calibration files with header

In order to get rid of detector dependent types of calibration parameters we need to add metadata in the calibration file. All metadata can be listed in the header of the calibration files, for example, using keyword mapping (dictionary):

pdscalibdata/test/test.data

```
# RULES:
# Lines starting with # in the beginning of the file are considered as comments or pseudo-comments for metadata
# Lines without # with space-separated values are used for input of parameters
# Empty lines are ignored
# REF: https://confluence.slac.stanford.edu/display/PCDS/pnCCD+processing+pipeline#pnCCDprocessingpipeline-
CalibrationofpnCCD

# Optional fields:
# TITLE      This is a file with pedestals
# DATE_TIME  2014-01-30 10:21:23
# AUTHOR     someone
# EXPERIMENT amotut13
# DETECTOR   Camp.0:pnCCD.1
# CALIB_TYPE center_um

# Mandatory fields to define the ndarray<TYPE,NDIM> and its shape as unsigned shape[NDIM] = {DIM:1,DIM:2,DIM:3}
# TYPE       float
# NDIM        3
# DIM:0       3
# DIM:1       4
# DIM:2       8

21757  21769  33464  10769  68489  68561  77637  54810
21758  21773  33430  10628  68349  68345  77454  54729
21678  21820  33054  10637  68310  68254  77976  54617
21730  21755  33193  10904  68570  68456  77425  54648

33110  10457  68275  68299  56732  79628  21754  21558
33329  10446  68158  68183  56949  79783  21811  21779
33141  10369  68476  68506  57121  79700  21858  21839
33098  10477  68452  68416  56923  79666  21681  21761

    51      18     -28      18      71     -20     -15     -54
   178      61     257     247     161     231     111     106
      1     -40     104     -24      9     -82      23      6
   102      40     272     270     194     246      60     118
```

This C++ module is intended to read/write calibration files with header.

Interface description can be found in [Doxygen doc for NDArrayIOV1](#)

pnCCD analysis for users

All other staff looks useful for all users and is migrated to the PSDM page: [pnCCD processing pipeline](#)

References

- [psana - Module Examples](#)
- [Psana Module Catalog - Old](#)
- [Calibration Management Tool](#)
- [Mask Editor](#)
- [pnCCD processing pipeline](#) - part of this page, which is useful for users
- [CSPAD Calibration](#)
- [Large area pnCCD DAQ and Electronics](#), Lothar Struder & Robert Hartmann
- [doxy-doc for psana modules](#)
- [Doxygen doc for NDArrayIOV1](#)
- [Doxy doc for CalibParsStore](#)