PACS Photometer Pipeline
Level 0 to 0.5

Basic AOT-independent pre-processing of the data

Step-by-step execution of the script NHSC_PACS_PHOT_PS_AOT.py
or NHSC_PACS_PHOT_process_miniscan.py

I am assuming:
• HIPE is running
• You know the name of the OBSID you want to reduce
• Your data are packaged in a pool
• Your data pool is located in a local store, e.g. .hcss/lstore, or at the HSA.
Load the data in your HIPE session

Put in **YOUR OBSID**

1. **Load the Observation Context**:
   ```python
   obs = getObservation(long(obsid), poolName=myPoolName, poolLocation=pooldir, verbose=True)
   ```

2. **Load the auxiliary products**:
   - **Pointing**:
     ```python
     pp = obs.auxiliary.pointing
     ```
   - **House keeping**:
     ```python
     photHK = obs.level0.refs["HPPHK"].product.refs[0].product["HPPHKS"]
     ```
   - **Calibration database**:
     ```python
     calTree = getCalTree()
     ```

3. **Load the Level-0 frames**:
   ```python
   frames = obs.level0.refs["HPPAVGB"].product.refs[0].product
   ```
Level 0 Signal: Dimensions

Most sources will not be visible on single frames due to intrinsic offset dispersion. Signal dispersion ~ 30000 ADU.
Level 0 Signal: Timeline of typical pixel

Calibration Block

Science Block

Calibration Source 1

1 nod cycle

Calibration Source 2

25 chopper cycles

3 dither positions

Level 0 Signal [ADU] vs Readout Sequence

NHSC DP Workshop - June 2010 - PACS Pipeline - N. BilloT
Level 0 Signal : Timeline of typical pixel

**Calibration Block**

**Science Block**

(Scan Mode AOT specific)

Calibration Source 1

Calibration Source 2

**10 Hz**
frames = findBlocks(frames, calTree=calTree)

Identify Calibration Blocks and Science Frames
frames = findBlocks(frames, calTree=calTree)

Identify Calibration Blocks and Science Frames

frames = removeCalBlocks(frames)

Remove the Calibration Blocks from the frames
frames = findBlocks(frames, calTree=calTree)

Identify Calibration Blocks and Science Frames

frames = removeCalBlocks(frames)

Remove the Calibration Blocks from the frames

frames = photFlagBadPixels(frames, calTree=calTree)

Add a BADPIXEL mask to frames["Mask"] from a calibration file
frames = findBlocks(frames, calTree=calTree)

Identify Calibration Blocks and Science Frames

frames = removeCalBlocks(frames)

Remove the Calibration Blocks from the frames

frames = photFlagBadPixels(frames, calTree=calTree)

Add a BADPIXEL mask to frames[“Mask”] from a calibration file

frames = photFlagSaturation(frames, calTree=calTree, hkdata=photHK)

Add 3 SATURATION masks to frames[“Mask”] from a calibration file (Soft and Hard saturations). Saturation starts above few 100s Jy.
frames = photConvDigit2Volts(frames, calTree=calTree)

Convert bolometer signals from ADUs to Volts

signal(volts) = (signal(ADU) - offset) * gain
frames = photConvDigit2Volts(frames, calTree=calTree)

Convert bolometer signals from ADUs to Volts

signal(volts) = (signal(ADU) - offset) * gain

frames = addUtc(frames, timeCorr)

Add UTC reference time frame in frames.status
frames = photConvDigit2Volts(frames, calTree=calTree)

Convert bolometer signals from ADUs to Volts

\[
\text{signal(volts)} = (\text{signal(ADU)} - \text{offset}) \times \text{gain}
\]

frames = addUtc(frames, timeCorr)

Add UTC reference time frame in frames.status

frames = photCorrectCrosstalk(frames)

Currently disabled

Correct crosstalk at the edge of the RED sub-arrays

Ground-based correction is overcorrecting
frames = photConvDigit2Volts(frames, calTree=calTree)
-> Convert bolometer signals from ADUs to Volts
  signal(volts) = (signal(ADU) - offset) * gain

frames = addUtc(frames, timeCorr)
-> Add UTC reference time frame in frames.status

frames = photCorrectCrosstalk(frames)
-> Correct crosstalk at the edge of the RED sub-arrays
  Ground-based correction is overcorrecting

frames = photConvChopper2Angle(frames, calTree=calTree)
-> Convert Chopper position from ADUs to angles from calibration file
frames = photAddInstantPointing(frames, pointingProduct, calTree=calTree)

Assign Ra and Dec to the array virtual aperture for each frame
frames = photAddInstantPointing(frames, pointingProduct, calTree=calTree)

Assign Ra and Dec to the array virtual aperture for each frame

frames = photAssignRaDec(frames, calTree=calTree)

Assign Ra and Dec to each pixel using distortion calibration file
frames = photAddInstantPointing(frames, pointingProduct, calTree=calTree)
⇒ Assign Ra and Dec to the array virtual aperture for each frame

frames = photAssignRaDec(frames, calTree=calTree)
⇒ Assign Ra and Dec to each pixel using distortion calibration file

frames = cleanPlateauFrames(frames)
⇒ Flag readouts when the chopper transits between ON-OFF positions

Currently disabled

Fast chopper  =>  no need for this module

You’ve reached Level 0.5!
PACS Photometer Pipeline
Level 0.5 to 1

Point Source AOT specific data processing

I am assuming:
  • Point Source observation is dithered, chopped and nodded

Remarks:
  • Observation-specific improvements possible in these processing steps
    (degitching, masking bad frames/columns)
frames = photMakeDithPos(frames)

- Identify dither positions and add DithPos vector in frames.status

- Calibration blocks are gone
- Signal is in Volts
- Dither positions are identified for each nod positions
frames = photMakeRasPos(frames)

Identify Nod positions and populate the frames.status
This module has to be executed even in Point Source AOT
frames = photAvgPlateau(frames, skipFirst = True)

Compute the average signal of each chopper plateau

- Identify Chopper plateaus
- Need to skip the first readout of each chopper plateau (bolometers finite response time)
frames = photAvgPlateau(frames, skipFirst = True)

- Identify Chopper plateaus
- Need to skip the first readout of each chopper plateau (bolometers finite response time)
- Compute the average (sigma clipping option available)
- Noise map created but irrelevant (sigma derived on 3-points samples)!!
frames = photAvgPlateau(frames, skipFirst = True)

Compute the average signal of each chopper plateau

- Identify Chopper plateaus
- Need to skip the first readout of each chopper plateau (bolometers finite response time)
- Compute the average (sigma clipping option available)
- Noise map created but irrelevant (sigma derived on 3-points samples)!!
frames = photAddPointing4PointSource(frames)

Ensure differenced images have consistent pointing information
frames = photDiffChop(frames)

- Subtract OFF- from ON-position frames for each chopper cycle

- Produce differentiated chopped signals, i.e. amplitudes (in Volts)
frames = photDiffChop(frames)

Subtract OFF- from ON-position frames for each chopper cycle

- Produce differentiated chopped signals, i.e. amplitudes (in Volts)
- Offsets are gone, but pixel-to-pixel gain variations remains as well as artefacts due to non-common optical path between ON- and OFF-positions
frames = photAvgDith(frames, sigclip = 3.)

Average detector readouts per dither positions
frames = photAvgDith(frames, sigclip = 3.)

Average detector readouts per dither positions

Deglitching by sigma clipping amplitude of each dither position
frames = photDiffNod(frames)

Subtract nodded signal
frames = photDiffNod(frames)

Subtract nodded signal

After photAvgDith

After photDiffNod

4-beams pattern typical of chop-nod observations

foreground + background subtraction reveals the source
frames = photCombineNod(frames)

Average nod positions

Pixel (8,8) timeline after photDiffNod

Pixel (8,8) timeline after photCombineNod
frames = photRespFlatFieldCorrection(frames, calTree=calTree)

Apply flatfield and convert from Volts to Jy
Bolometer signal is assumed linear (no non-linearity correction available yet)
This is verified for sources with fluxes < ~50 Jy based on pre-launch estimates
frames = photRespFlatFieldCorrection(frames, calTree=calTree)

Apply flatfield and convert from Volts to Jy
Bolometer signal is assumed linear (no non-linearity correction available yet)
This is verified for sources with fluxes < ~50 Jy based on pre-launch estimates

frames = photDriftCorrection(frames, calTree=calTree)

Correct multiplicative drifts based on Calibration Blocks measurements

Currently disabled
3 Dithered Fully Calibrated Images

You’ve reached Level 1!
PACS Photometer Pipeline
Level 1 to 2

Point Source AOT specific data processing

Projecting dithered images to build the final map

Two flavors of the projection algorithm
map1 = photProject(frames, outputPixelSize = pixsize, calTree = calTree, calibration = True)

Shift by chop throw

Dither 1

Shift by chop throw

2

Shift by chop throw

3

In this projection scheme, the frame of reference is the detector array

Flux conservation

in arcsec

NHSC DP Workshop – June 2010 – PACS Pipeline - N. BilloT
map1 = photProject(frames, outputPixelsize = pixsize, calTree = calTree, calibration = True)

Homogeneous coverage (good for background statistics)

Modest S/N ratio of individual sources
map2 = photProjectPointSource(frames, allInOne=1, calTree=calTree,
outputPixelsize=pixsize, calibration=True)
map2 = photProjectPointSource(frames, allInOne=1, calTree=calTree,
                                   outputPixelsize=pixsize, calibration=True)

- Central source has highest S/N ratio
- Area of homogeneous coverage is narrow and confined around the source
- 8 artefact sources left after map reconstruction
Concluding remarks

• Good PSF stability due to small pointing errors inherent to the Point Source AOT

• High photometric accuracy for isolated sources

• Dithering is recommended to fully sample the PSF at 70 and 160 microns
  Concatenated AORs are needed to fully sample the PSF in the nod direction

• **Limited area to compute the background level for sky subtraction**
  Possible blend between positive and negative images for bright sources
  Possible blend for crowded regions or structured background

• **Homogeneous coverage limited to a small region around the source**
  Limited area for computing the noise level

• **Relatively low redundancy compared to scan mode**
  Source falls on handful of pixels
MiniScan maps are the recommended alternative to Point Source AOTs
PACS Photometer Pipeline
Level 0.5 to 2

Scan Map AOT specific data processing

Remarks:
• MapMaking is quite memory intensive for large data sets
Deglitching the data

2 non-exclusive options: Temporal AND/OR Spatial deglitching

• Temporal deglitching:
  Multi-resolution Median Transform of individual pixel timelines (Stark et al. 1998)

```python
frames = photMMTDeglitching(frames, incr_fact=2, mmt_mode='multiply', scales=3, nsigma=5)
```
Deglitching the data

• Spatial deglitching:
  Outliers detection in vector of detector pixels contributing to a given map pixel

\[\text{mi} = \text{mapIndex(frames)} \quad \# \text{Build a mapIndex}\]
\[s = \text{Sigclip}(10, 3) \quad \# 10\text{-elements sliding vector, 3-}\sigma \text{ clipping}\]
\[s.\text{setOutliers("both")} \quad \# \text{Detect both positive and negative glitches}\]
\[s.\text{setBehavior(Sigclip.CLIP)}\]
\[s.\text{setMode(Sigclip.MEDIAN)}\]
\[\text{map} = \text{deg(mi, frames, algo} = s, \text{map} = \text{False, mask} = \text{True)}\]
Treatment of 1/f noise

2 exclusive options: Highpass filtering OR MADMap

No filtering  Highpass filtering  MADMap
Highpass Filter (HPF)

frames = highpassFilter(frames, hpfwidth, maskname = “HighpassMask”)

Drift removal by subtracting a running median from each readout

*hpfwidth* parameter is specified in units of readouts

Recommended values are 20 and 15 for medium and fast scan speeds

Filter width is actually: \(2 \times hpfwidth + 1\)
frames = highpassFilter(frames, hpfwidth, maskname = "HighpassMask")

hpfwidth = 20

hpfwidth = 200

hpfwidth = 2000
frames = highpassFilter(frames, hpfwidth, maskname = “HighpassMask”)

Local median is affected by bright sources in the signal

→ Shadow in the vicinity of the source in the direction of the scan.

Source masking solves the problem ("HighpassMask"):  
  • Define circular patches around sources of known position  
  • Define mask from image (σ-clipping from image)

HPF      Masked HPF
**Projection / Map Making**

```python
map = photProject(frames, calibration=True, outputPixelSize=pixsize)
```

- **Co-adds all HPF images**
  - Conserves surface brightness
  - Uses the pixel distortion information
  - Final map pixel size in arcseconds

- **Output is a multi-layered image with:**
  - Image, WCS, error, and coverage map

! Disable glitch mask before projection! 

see demo script

Frames pixels with location (row, column, timeindex) projected onto the map pixels

NHSC DP Workshop – June 2010 – PACS Pipeline - N. BilloT