Deglitching PACS Photometer Data

Presentation based on tutorial PACS-402
accessible from the NHSC website at
https://nhscsci.ipac.caltech.edu/sc/index.php/Pacs/DataProcessing
Introduction
What is a glitch?

A glitch occurs when a cosmic ray hits the detectors, which generates a sudden signal change.

PACS Bolometer arrays have a low glitch rate: less than 1% of the data is affected

Glitch rate is on average around 3 (1) hits/frame in the Blue (Red) channel

Glitch energy distribution:
Few energetic impacts, numerous small glitches

Thermal cross-talk:
In rare occasions, strong impacts might affect several contiguous pixels with negative glitches.
**Introduction**

**Glitch Zoology**

- Positive glitch (~85%)
- Negative glitch (~15%)
- Break glitches (rare)

---

**Components:**

- **Inter-pixel wall**
  - (large heat capacity)
- **Absorbing grid**
  - (low heat capacity)
- **Readout electronics**
  - (transistor small cross-section)
There are two non-exclusive deglitching algorithms available in HIPE: Temporal and/or spatial deglitching

**Temporal** approach identifies glitches from individual pixel timelines

- Excellent performance for deep observations of faint sources
- Bright sources are erroneously flagged as glitches since they “look” like glitches when scanned, it is thus necessary to disable the Glitchmask on-source

**Spatial** approach identifies glitches by exploiting spatial redundancy

- Reliable even in the presence of strong signal gradients, e.g. with bright compact sources or structured extended emission
- The algorithm is quite memory intensive for large data sets
- It requires a high level of spatial redundancy
The standard pipeline uses a time-domain deglitching algorithm at Level 0.5: The MMTDegitching uses a Multi-resolution Median Transform to identify glitches on individual pixel timelines (Stark et al. 1998)

```python
HIPE> frames = photMMTDeglitching(frames, incr_fact=2, mmt_mode='multiply', scales=3, nsigma=5, imagenoisearray = frames.noise[:,:,0])
```

The set of parameters given here work well with most observations

Complete documentation on MMTDeglitching:
PACS Pipeline Reference Manual Section 3.7.8
PACS User’s Reference Manual Section 1.146
Deglitching
1. Temporal approach

Inspect the *MMT_Glitchmask*

HIPE> print frames.mask

---

*MMT_Glitchmask* was created in *frames.mask*
Deglitching

1. Temporal approach

Inspect the **MMT_Glitchmask** with MaskViewer

- Group of pixels (source) erroneously flagged as glitches (see tutorial PACS-202)
- Randomly distributed genuine single event glitches
- Scroll to explore the 3rd dimension of the mask cube
The spatial deglitching algorithm, or **Second Level Deglitching**, is not yet part of the standard pipeline, but you can integrate it in your own data processing scripts. It relies on spatial redundancy to detect outliers.

The Second Level Deglitching works on a *mapIndex* variable. The mapIndex for each individual map pixel is populated with the signal contributions from all detector pixels.
Deglitching

2. Spatial approach

Outliers are detected with a sigma-clipping algorithm and flagged as glitches.

Signal from the sky (contributions from various detector pixels at different times). The relatively low dispersion is due to the intrinsic instrumental noise.
Deglitching
2. Spatial approach

Syntax for running the second level deglitching on frames

```python
from herschel.ia.numeric.toolbox.basic import Sigclip
from herschel.pacs.spg.phot import IIndLevelDeglitchTask
from herschel.pacs.spg.phot import MapIndexTask
mapIndex = MapIndexTask()
mi = mapIndex(frames, slimindex= False)
envSize = 10
nsigma = 5
s = Sigclip(envSize, nsigma, outliers = "both", behavior=Sigclip.CLIP, mode=Sigclip.MEDIAN)
deg = IIndLevelDeglitchTask()
map = deg(mi, frames, mask = True, map = False, algo = s, deglitchvector="timeordered")
```

Complete documentation on Second Level Deglitching:
http://www.herschel.be/twiki/bin/view/Public/SecondLevelDeglitching
Deglitching

2. Spatial approach

Second level deglitching is quite memory intensive. You can save on the memory by dividing the map into several tiles and deglitch them independently in a loop.

```python
mapIndex = MapIndexTask()
mapIndex.inframes = frames
mapIndex.slimindex = False
mapIndex.no_slicerows = 4
mapIndex.no_slicecols = 4
mapIndex.slices = Intld.range(16)
envSize = 10
nsigma = 5
s = Sigclip(envSize, nsigma, outliers = "both", behavior=Sigclip.CLIP, mode=Sigclip.MEDIAN)
deg = IIIndLevelDeglitchTask()
img = None
# Now loop over the number of tiles to deglitch
for mi in mapIndex:
    img = deg(mi, frames, mask = True, map = False, algo = s, deglitchvector="timeordered", partialmap=img)
del(mi)  # to free the memory
Define the number of tiles on which to run the second level deglitching
```
Second level deglitching is quite memory intensive. You can save on the memory by using the minimum information required to build the glitchmask (not the map).

```python
mapIndex = MapIndexTask()
mapIndex.inframes = frames
mapIndex.slimindex = False
mapIndex.no_slicerows = 4
mapIndex.no_slicecols = 4
mapIndex.slices = Int16.range(16)
envSize = 10
nsigma = 5
s = Sigclip(envSize, nsigma, outliers = "both", behavior=Sigclip.CLIP, mode=Sigclip.MEDIAN)
deg = IIIndLevelDeglitchTask()
img = None
# Now loop over the number of tiles to deglitch
for mi in mapIndex:
    img = deg(mi, frames, mask = True, map = False, algo = s, deglitchvector="timeordered", partialmap=img)
del(mi)  # to free the memory
```

Set **Slimindex = True** to minimize the size of the MapIndex product.
Deglitching

2. Spatial approach

Inspect the 2\textsuperscript{nd} level glitchmask

HIPE> print frames.mask

Access data from this mask as boolean array with bool4mask =
Ramps.mask["masktype"].data
Alternatively use getMask(...) -methods on the PacsContainer classes (i.e. Frames,
Ramps), for example Ramps.getMaskld("masktype", row, col).
HIPE>

2\textsuperscript{nd} level glitchmask was created in frames.mask
The second Level deglitching relies on spatial redundancy. If you have scan and cross-scan observations on the same object/field, then you should **concatenate** (*join*) the two frames before applying the second level deglitching (See tutorial PACS-202).

```plaintext
HIPE> print frames_scan.dimensions ; print frames_Xscan.dimensions
array([32, 64, 1000], int)
array([32, 64, 2100], int)

HIPE> frames_Xscan.join(frames_scan)

HIPE> print frames_scan.dimensions ; print frames_Xscan.dimensions
array([32, 64, 1000], int)
array([32, 64, 2100], int)
```
Deglitching

2. Spatial approach

MapIndexViewer: an interactive second level deglitching

HIPE> from herschel.pacs.spot.gui.mapindexview import MapIndexViewer

HIPE> MapIndexViewer(mi, frames)
Most relevant documentation for de-glitching PACS data:
- PACS Data Reduction Guide
- PACS User’s Reference Manual

General documentation necessary to manipulate PACS data:
- HIPE Owner’s Guide
- Herschel Data Analysis Guide
- Scripting and Data Mining