NHSC/PACS Web Tutorials

Running PACS photometer pipelines

PACS-201 (for Hipe 9.0)

Level 0 to Level 2 processing:

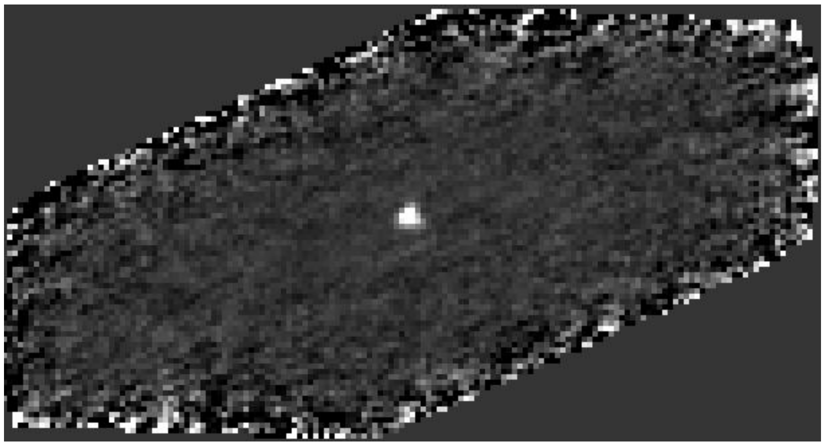
From raw data to calibrated maps

Prepared by Roberta Paladini

September 2012

Introduction

This tutorial applies to reducing large or mini scan maps.
They both share many common processing steps

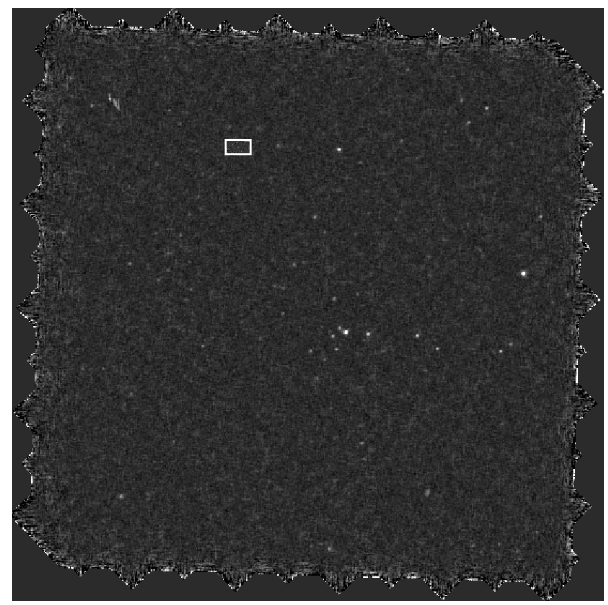


Mini scan map

with a single source in the center of the field

Large scan map

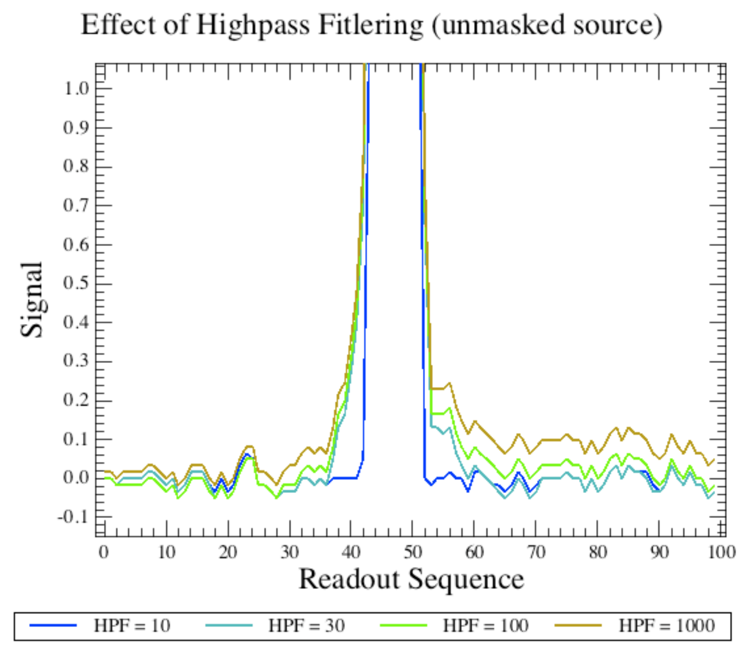
With multiple sources distributed in the field

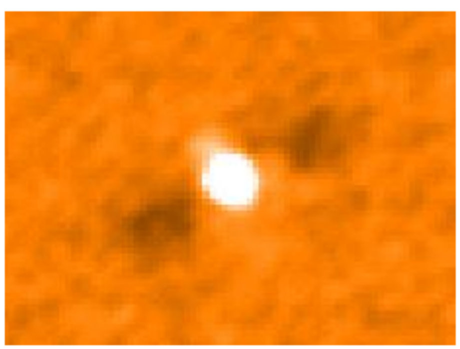


Global drift (correlated noise) and *1/f* (uncorrelated) noise are corrected with <u>High-Pass Filter</u>

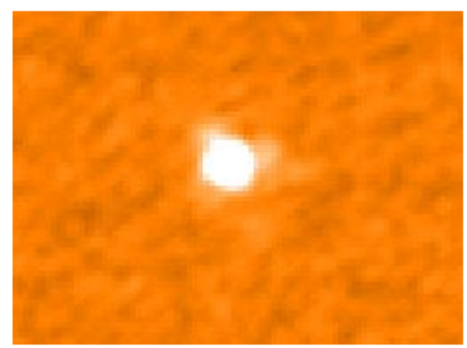
Main Idea:
sliding median-filter on individual pixel timelines to remove large scale drifts

When a bright source enters the filter box, it alters the estimate of the median and thus the drift removal





Unmasked Highpass Fitlering



Masked Highpass Filtering

The large/mini-scan map pipeline consists of 4 sections:

SECTION 0: section containing settings definitions

- obsids, camera, hpfradius, outpixsz, pixfrac, …

SECTION 1: from Level 0 to Level 0.5

- extract frames (e.g raw data) from Observation Context
- extract auxiliary information (pointing products, calibration tree, etc.)
- remove calibration blocks
- flag bad/crosstalk/saturated pixels
- convert digital units into Volts
- convert chopper angles into sky angles (degrees)
- add coordinates to reference pixel (center of the detector)

SECTION 2: from Level 0.5 to Level 1

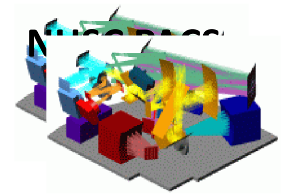
- glitch (MMT) or ….not deglitch (do 2nd level deglitching later – default) ?
- apply flat-field and convert Volts into Jy/pixel
- apply non-linearity correction

SECTION 3: from Level 1 to Level 2

- set high-pass filter radius (hpfradius) – depends on scan speed
- mask the source (3 options)
- run high-pass filter with generated mask
- select only frames at constant speed
- run 2nd level deglitching (unless you decide otherwise)
- make your map (photProject)

LEGEND:

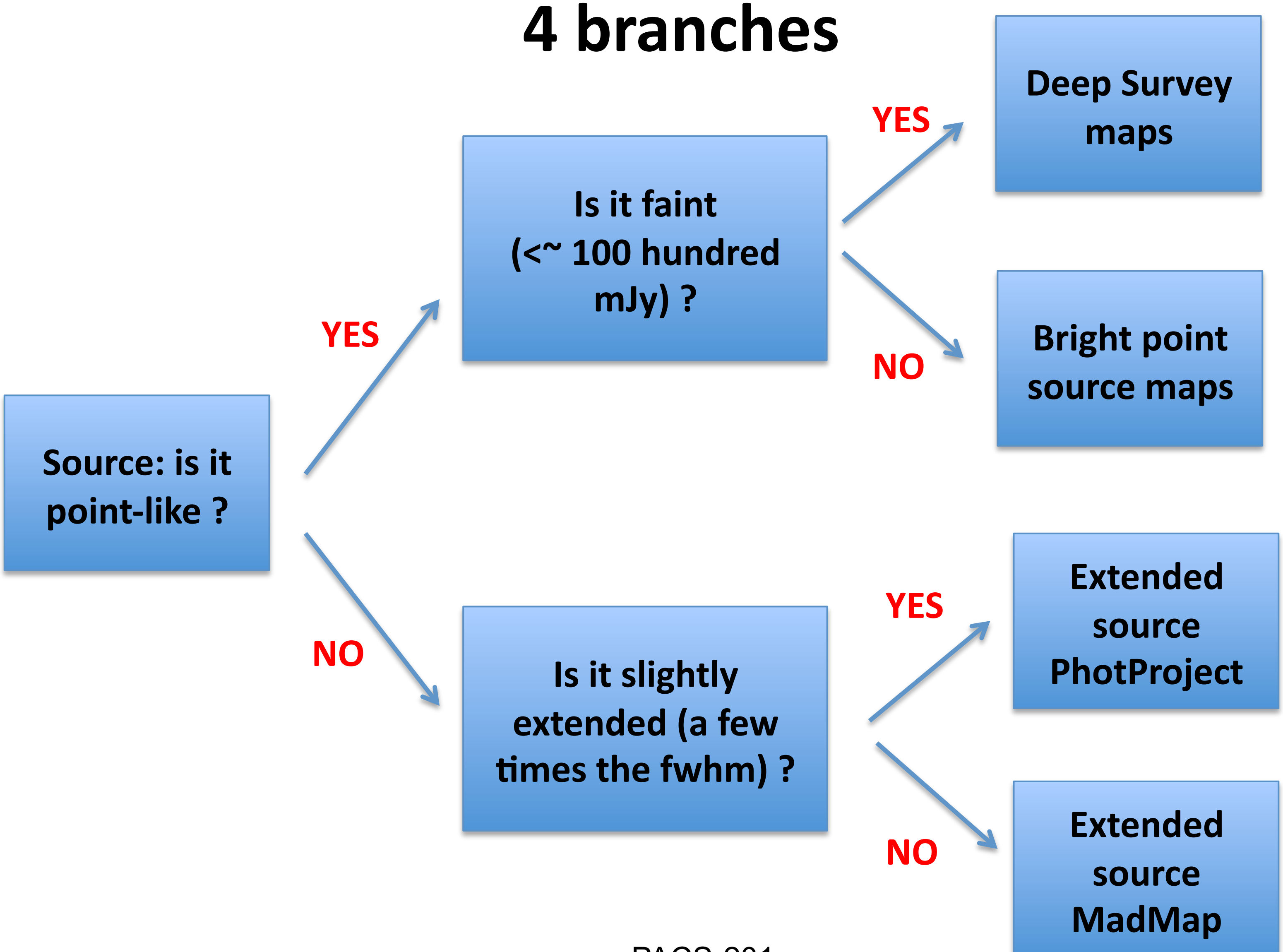
Blue → "iterative" parts of the pipeline: the user is required to provide inputs and/or take decisions !



The user has <u>3 major decisions</u> to take when he/she reduces mini/large-scan map data:

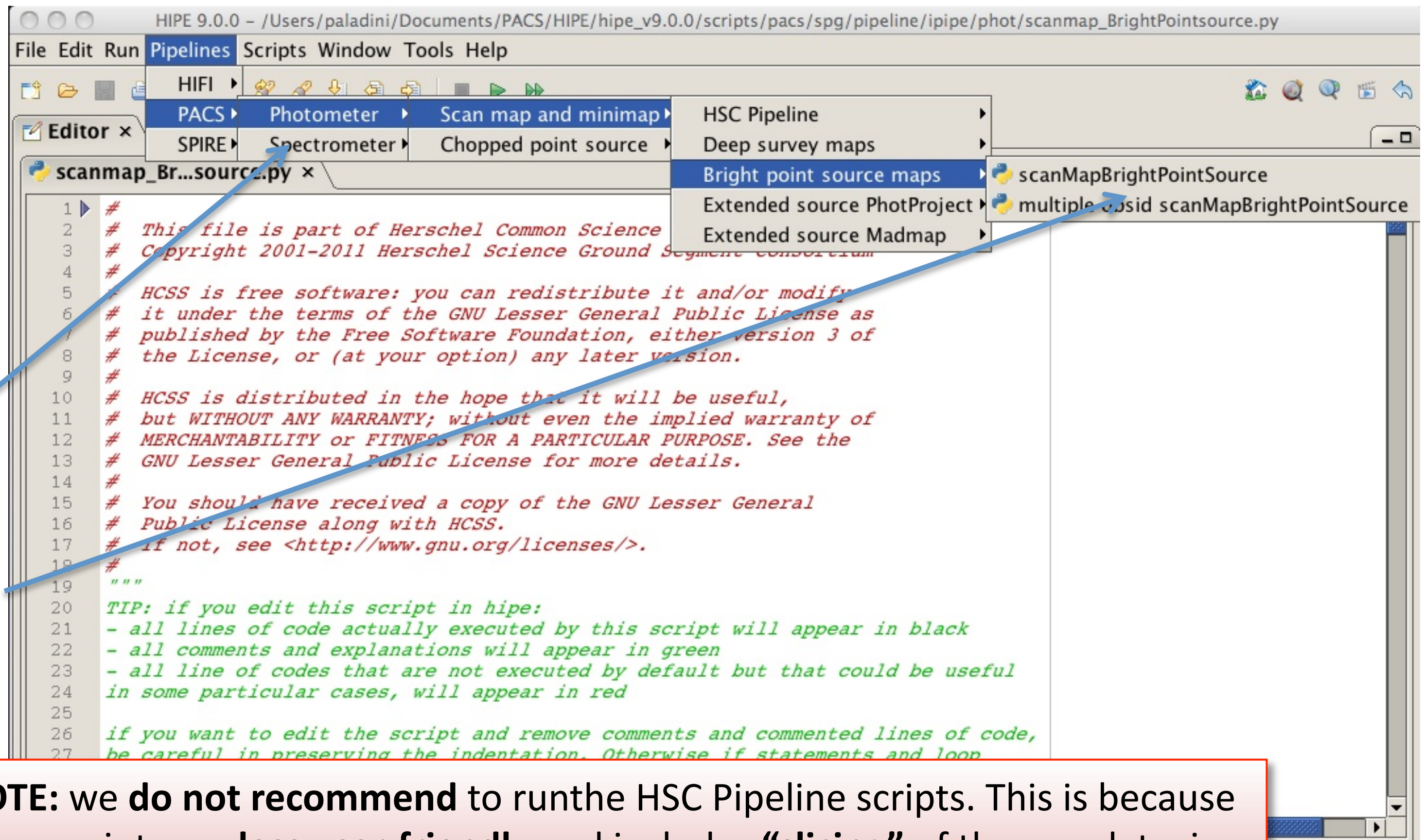
1. How to set High-pass filter radius
2. How to set mask radius to "protect" source from high-pass filtering
3. When to deglitch (from Level 0.5 to 1 <u>or</u> 1 to 2) and how (2nd level deglitching – default – <u>or</u> MMT deglitching)

PACS Photometer Pipeline: 4 branches



For this tutorial, we use the ipipe script: scanMapBrightPointSource

From the Pipeline Menu,
Select PACS > Photometer >
Scan map and mini map >
Bright Point Source maps >
scanMapBrightPointSource



NOTE: we **do not recommend** to runthe HSC Pipeline scripts. This is because these scripts are **less user-friendly** and includes **"slicing"** of the raw data, i.e. an operation designed only with the intent of saving on memory which will disappear in future versions of the pipeline.

In the script, comment out line # 219:

```
Console ×
HIPE> #getObservation(obsid, useHsa=True, instrument='PACS')
```

Uncomment and edit lines #98:

```
Console ×
HIPE> direc = ”/specify here the name of the input/output directory/"
```

NOTE: directories must end with a trailing slash “/”

Uncomment and edit line # 106:

```
Console ×
HIPE> obsid= “1342189191"  #give here the obsid number
```

Uncomment and edit line # 226 and 227:

```
Console ×
HIPE > dir = '/specify here the name of the directory where data reside/’
HIPE > obs = getObservation(obsid, poolLocation=dir)
```

Section 1:
From Level 0 to Level 0.5

1. extract frames (e.g raw data) from Observation Context

2. extract auxiliary information (pointing products, housekeeping, etc.)

3. extract calibration tree

4. filter slew to target (new in Hipe 9.0)

5. remove calibration blocks

6. flag bad/crosstalk/saturated pixels (new in Hipe 9.0)

7. convert digital units into Volts

8. convert chopper angles into sky angles (degrees)

9. add coordinates to reference pixel (center of the detector)

1.1 Extract the Level 0 data cube (*frames*) from the ObservationContext

Level 0: raw data cube

Syntax for Blue/Red array

Console

```
HIPE> frames = obs.level0.refs["HPPAVGB"].product.refs[0].product
```

HPPAVGB/R: **H**erschel **P**ACS **P**hotometer **AVG**erage **B**lue/**R**ed
This is the signal downlinked from the spacecraft after on-board averaging

1.2. Extract the auxiliary data from the ObservationContext

Console

```
HIPE> pp = obs.auxiliary.pointing

HIPE> photHK = obs.level0.refs["HPPHK"].product.refs[0].product["HPPHKS"]

HIPE> oep = obs.auxiliary.orbitEphemeris
```

pp: **p**ointing **p**roduct
photHK: **phot**ometer **H**ouse**K**eeping
oep: **o**rbit**e**phemeris **p**roduct

We recommend not to modify these variables, they contain critical information required in subsequent processing modules.

1.3. Load the calibration files (*calTree*)

The Calibration Tree (*calTree)* contains all the files necessary to process your data

Console

```
HIPE> calTree = getCalTree( time = frames.startDate )
```

Some calibration files changed with time, e.g. the SIAM or pointing calibration file, so it is necessary to provide the date of observation for retrieving the appropriate calFiles

1.4. Filter slew to target

Console

```
HIPE> frames  = filterslew(frames)
```

1.5. Identify Blocks in the observation and remove the Calibration Blocks

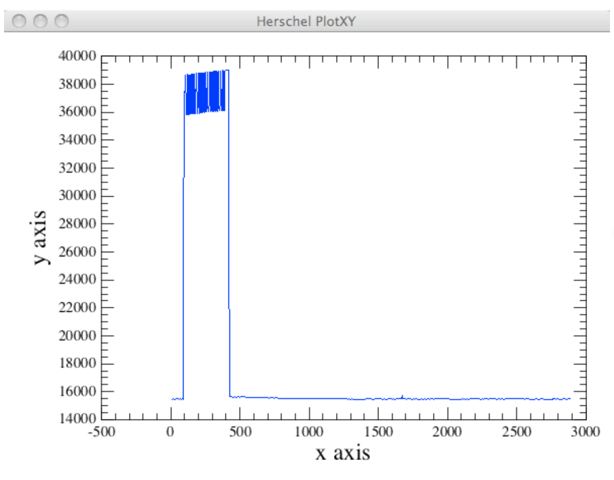
Console

```
HIPE> frames = findBlocks(frames, calTree=calTree)

HIPE> frames = detectCalibrationBlock(frames)

HIPE> frames = removeCalBlocks(frames)
```

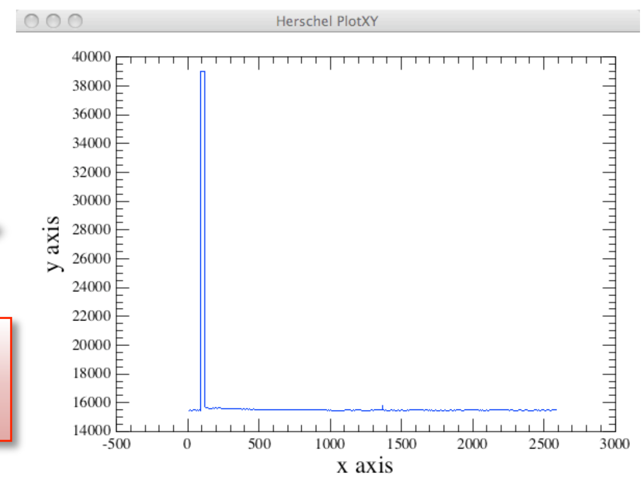
Before



Console

```
HIPE> PlotXY(frames.signal[8,8,:])
```

After



The chopping pattern from the Calibration Block is gone.

1.6. Flag Bad/Crosstalk/Saturated Pixels and populate *frames.mask*

```
HIPE > frames = photFlagBadPixels(frames, calTree=calTree)
HIPE > frames = photMaskCrosstalk(frames)
HIPE > frames = photFlagSaturation(frames, calTree=calTree, hkdata=photHK)
```

Electronic crosstalk was identified during ground tests . It is present also in on-flight data. It affects especially column 0 of all modules in the red channel.

```
HIPE> blue_badpix = calTree.photometer.badPixelMask.blue   # get the mask
HIPE> blue_badpix[2,30] = 1   # This flips the value of pixel(2,30) from False to True
HIPE> frames.setMask("BADPIXELS", blue_badpix)
```

Optional addition of rogue pixels to the BADPIXEL mask

1.7. Convert the signal from digital units (ADU) into physical units (Volts)

Console

```
HIPE> frames = photConvDigit2Volts(frames, calTree=calTree)
```

1.8. Convert the chopper angle from digital units (ADU) into physical units (degrees)

Console

```
HIPE> frames = convertChopper2Angle(frames, calTree=calTree)
```

1.9. Add the pointing information to the *frames*

```
HIPE> frames = photAddInstantPointing(frames, pp, calTree=calTree, orbitEphem=oep)
```

Provide the telescope pointing product and the orbit ephemeris that were extracted from the ObservationContext

This module adds the spacecraft pointing to the *frames.status* as coordinates and additional pointing information

Save your Level 0.5 products before further processing

Console

```
HIPE> simpleFitsWriter(frames,"/my/directory/my_frames.fits”)
```

The *frames* are saved in standard fits format
The saved file can be read back into HIPE or IDL

Console

```
HIPE> frames = simpleFitsReader("/my/directory/my_frames.fits”)
```

Terminal — idl — 74×5

idl

```
IDL> a = mrdfits("frames.fits", 0, header)
MRDFITS: Null image, NAXIS=0
IDL> b = mrdfits("frames.fits", 1)
MRDFITS: Image array (2586,32,16)  Type=Real*8
IDL>
```

This is the extension of the fits file

Section 2:
From Level 0.5 to Level 1

1. glitch (MMT) or ….not deglitch (do 2^{nd} level deglitching later – default) ?
2. apply flat-field and convert Volts into Jy/pixel
3. apply non-linearity correction (new in Hipe 9.0)

2.1. Shall we deglitch NOW….. or LATER ?? (default: LATER, e.g. Level 1 to 2)

There are two non-exclusive deglitching algorithms available in HIPE:
Spatial (DEFAULT) and/or temporal deglitching

Spatial (2nd Level Deglitching) approach identifies glitches by exploiting spatial redundancy

- 🙂 **Reliable** even in the presence of strong signal gradients, e.g. with bright compact sources or extended emission
- 🙁 The algorithm is quite memory intensive for large data sets
  It requires a high level of spatial redundancy

Temporal (MMT) 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

Option A. If we have faint sources, or not enough redundancy, we deglitch now, applying the MMT deglitching task:

```
HIPE> frames = photMMTDeglitching(frames, incr_fact=2, mmt_mode='multiply', scales=3, nsigma=5)
```

The set of parameters provided here works well with most observations

Option B. Otherwise (i.e. bright sources, strong gradient, poor redundancy), we skip step A and go to next task → <u>HSC pipeline default !</u>

2.2. Apply the flat-field correction and convert the signal from Volt/pixel into Jy/pixel

```
HIPE> frames = photRespFlatfieldCorrection(frames, calTree = calTree)
```

2.3. Apply non-linearity correction

This correction has no influence on point-sources fainter than ~ 100 Jy

```
HIPE> frames = photNonLinearityCorrection(frames, calTree = calTree)
```

You have reached Level 1

The frames are now calibrated and ready for generating the map

Save Level 1 products before further processing

Console

```
HIPE> simpleFitsWriter(frames,"/my/directory/my_frames.fits”)
```

The *frames* are saved in standard fits format
The saved file can be read back into HIPE or IDL

Console

```
HIPE> frames = simpleFitsReader("/my/directory/my_frames.fits”)
```

Terminal — idl — 74×5

idl

```
IDL> a = mrdfits("frames.fits", 0, header)
MRDFITS: Null image, NAXIS=0
IDL> b = mrdfits("frames.fits", 1)
MRDFITS: Image array (2586,32,16)  Type=Real*8
IDL>
```

This is the extension of the fits file

Section 3:
From Level 1 to Level 2

1. Set high-pass filter radius (hpfradius) – depends on scan speed

2. mask the source (3 options)

3. run high-pass filter with generated mask

4. Select only frames at constant speed

5. run 2nd level deglitching (unless you decide otherwise)

6. make your map (photProject)

3.1. Choice of high-pass filter radius: hpfradius

The width of the high-pass filter depends on the science case and scan speed.

Ex: For a scan speed of 20''/sec (i.e. medium scan speed in "PACS only Photometry Mode" or low scan speed for "Parallel Mode"), the best values, for the blue and red camera are, respectively:

```
HIPE> if camera=='blue':
HIPE >   hpfradius=15
HIPE > elif camera=='red':
HIPE >   hpfradius=25
```

→ These values (in readouts) allow us to remove the 1/f noise while preserving as much as possible the flux in the wings of the PSF (Point Spread Function)

RULE OF THUMB:

```
HIPE > If camera == 'blue':
HIPE >   hpfradius = int(CEIL(15.* 20./speed))
HIPE > elif camera =='red':
HIPE >   hpfradius = int(CEIL(25.*20./speed))
```

NOTE: to get the 'speed' information:

- for PACS only photometry mode:
  speed = frames.meta["mapScanSpeed"].value
- for parallel mode:
  speed = frames.meta["mapScanRate"]. value

3.2. Choice of masking radius: HighPassMask

OPTION 1: observation of a point source of known coordinates → mask blindly all the pixels within a given radius around these coordinates

OPTION 2: the user has pre-existing mask (from other data or external catalog) → user only needs to use this mask

OPTION 3 (default) : this is the case for unknown coordinates of the source → user needs to create the map from the scratch and directly from the observations

3.2. Choice of masking radius – OPTION 1: *known coordinates*

```
HIPE> frames = photAssignRaDec(frames, calTree=calTree) → only OPTION 1
HIPE>  rasource  = obs.meta["raNominal"].value
HIPE>  decsource = obs.meta["decNominal"].value
HIPE > cosdec=COS(decsource*Math.PI/180.)
HIPE > on_source=SQRT(((frames.ra-rasource)*cosdec)**2+(frames.dec-decsource)
         **2) < masking_radius/3600.
```

To define, attach and set HighPassMask in the frames:

```
HIPE > if (frames.getMask().containsMask("HighpassMask") == False):
HIPE >     frames.addMaskType("HighpassMask","Masking the source for High pass")
HIPE >    frames.setMask('HighpassMask',on_source)
```

3.2. Choice of masking radius – OPTION 2:
Pre-existing mask

The user has already a mask called HighpassMask. The file containing the mask is loaded as "maskfile”:

```
HIPE > mask=simpleFitsReader(maskfile)
```

To attach HighPassMask in the frames:

```
HIPE > frames  = photReadMaskFromImage(frames, si=mask,
         extendedMasking=True,maskname="HighpassMask")
```

3.2. Choice of masking radius – OPTION 3 (DEFAULT): *unknown coordinates - I*

1. **High-pass filtering without mask**
2. **Select only frames for which the telescope is slewing at a constant speed**
3. **Create a preliminary map**
4. **….next slide**

Console

```
HIPE> frames = highpassFilter(frames, hpfwidth)

HIPE> frames = filterOnScanSpeed(frames,limit=10)

HIPE> map = photProject(frames, calTree = calTree, calibration=True)
```

The task removes all readouts with a velocity 10% higher or lower than then scan speed

NOTE: This map is NOT good for photometry: It must be used *only* for identifying bright sources and for sigma-clipping

3.2. Choice of masking radius – OPTION 3 (DEFAULT):
unknown coordinates - II

4. Identify the region of the map with high coverage
5. Use this region to estimate the signal standard deviation (stdev)
6. Define the threshold as cutlevel*stdev
7. ….next slide

Console

```
HIPE >  med=STDDEV(map.coverage[map.coverage.where(map.coverage > 0.)])

HIPE > signal_stdev=STDDEV(map.image[map.image.where(map.coverage > med)])

HIPE > cutlevel=3.0  (2.0 is DEFAULT)

HIPE > threshold=cutlevel*signal_stdev
```

3.2. Choice of masking radius – OPTION 3 (DEFAULT): *unknown coordinates - III*

7. Mask everything above the threshold → these are the "sources"
8. Mask in the timeline all readouts at the same coordinates of the map pixels with signal above the threshold

Console

```
HIPE > mask=map.copy()mask=map.copy()
HIPE > mask.image[mask.image.where(map.image > threshold)] = 1.0
HIPE > mask.image[mask.image.where(map.image < threshold)] = 0.0
HIPE > frames  = photReadMaskFromImage(frames, si=mask, extendedMasking=True,maskname="HighpassMask")
```

For bright sources, it is recommended to make more than 1 iteration. Normally 3 iterations are sufficient to obtain good results

3.3. Run High-pass filter with generated mask

```
HIPE  > frames  = highpassFilter(frames,hpfradius,maskname="HighpassMask", interpolateMaskedValues=True)
```

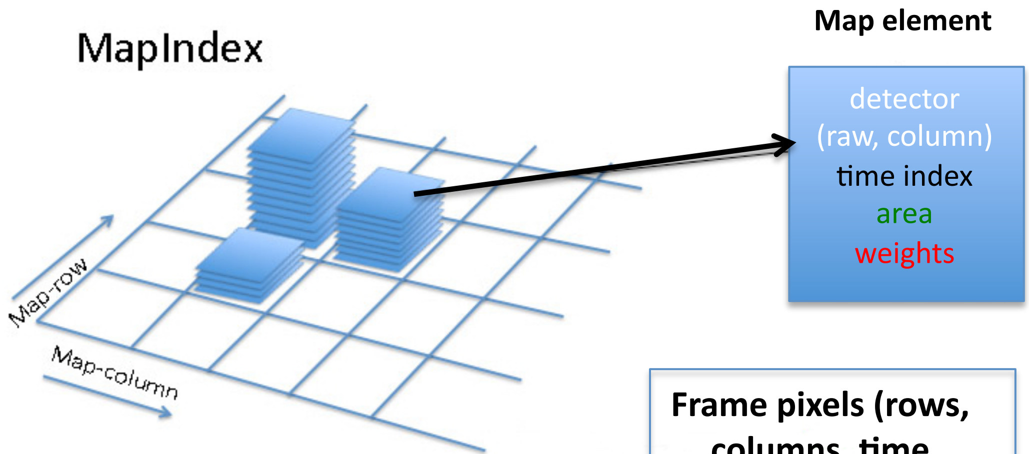
We set **InterpolateMaskedValues** to True when hpfradius < masking radius. In this case, part of the source flux is removed by the filtering process. Setting the parameter to True allows to "jump" the source when applying high-pass filtering

3.4. Select frames at constant speed

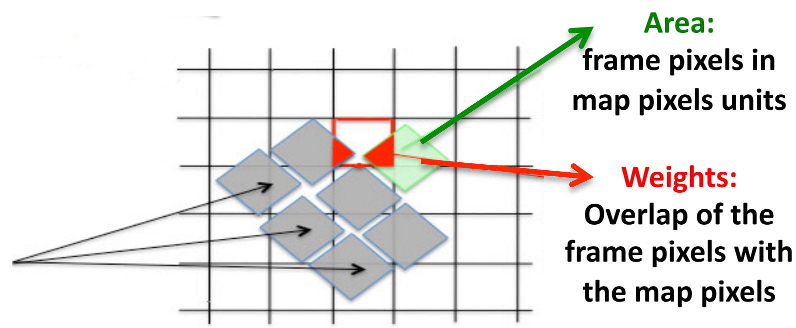
```
HIPE  > framesframes= filterOnScanSpeed(frames, limit=10)
```

3.5. 2nd Level Deglitching – I

The **Second Level Deglitching** relies on spatial redundancy to detect outliers. It makes use of a MapIndex variable. The MapIndex is populated with the signal contributions from all detector pixels for each individual map pixels



Frame pixels (rows, columns, time indices) projected onto map pixels



3.5. 2nd Level Deglitching – II

```
HIPE > from herschel.pacs.signal import MapIndex
HIPE > from herschel.pacs.spg.phot import MapIndexTask
HIPE > from herschel.pacs.spg.phot import IIndLevelDeglitchTask
HIPE > mi = MapIndexTask()
HIPE > iind = IIndLevelDeglitchTask()
HIPE> mapToCubeIdx = mi(frames,slimindex=True)
HIPE > s = Sigclip(nsigma=10,behavior="clip",outliers="both",mode=Sigclip.MEDIAN)
HIPE > deg = iind(mapToCubeIdx,frames,map=False,mask=True,maskname='SecondGlitchmask',algo=s)
```

This task needs a lot of RAM. Alternatively, you can use MapDeglitchTask , although it is more time-consuming

Outliers are detected with a **sigma-clipping** algorithm and flagged as glitches. Both **positive** and **negative** outliers are detected. By default, outliers are detected with respect to the **median**.

3.6. Making the map

```
HIPE  > map=photProject(frames, outputPixelsize=outpixsz, calTree=calTree, pixfrac=pixfrac)
```

The photProject task performs a simple co-addition of the images using the drizzle method (Fruchter and Hook, 2002, PASP, 114, 144). The key parameters are the output pixel size and the drop size (**pixfrac**). **A small pixfrac value can help to reduce the correlated noise due to the projection.**

DEFAULT VALUES:

```
HIPE > If camera == 'blue':
HIPE >    outpxsz = 2.
HIPE > elif camera =='red':
HIPE >    outpixsz = 3.

HIPE > pixfrac = 0.1
```

What's new in Hipe 9.0 – I

PACS -> Pipeline -> Photometry

New calibration files for :

- **flat-field** (flatField v4)
- **responsivity** (v7):

  → the change in photometry for extended sources is about +3% (red)), + 1.5% (green), 0% (blue)

  → the change in photometry for point-sources is 2% (red) and < 1% (green/ blue)
- **aperture correction** (apertureCorrection v3)
- **non-linearity correction** (nonLinearCoef v2)

What's new in Hipe 9.0 – II

PACS -> Pipeline -> Photometry

• <u>Updated ipipe scripts with:</u>

→ now using the **filterSlew** and **photMaskCrosstalk** tasks

→ **filterOnScanSpeed** task now only masks the turnover frames instead of deleting them

→ optimized mask generation for the high-pass filtering

→ optimized second order deglitching with the **MapDeglitch** task

Thank you !