NHSC/PACS Web Tutorials

Running PACS photometer pipelines

PACS-201 (for Hipe 11.0.1)

Level 1 to Level 2 processing:*

The High-Pass Filter pipeline

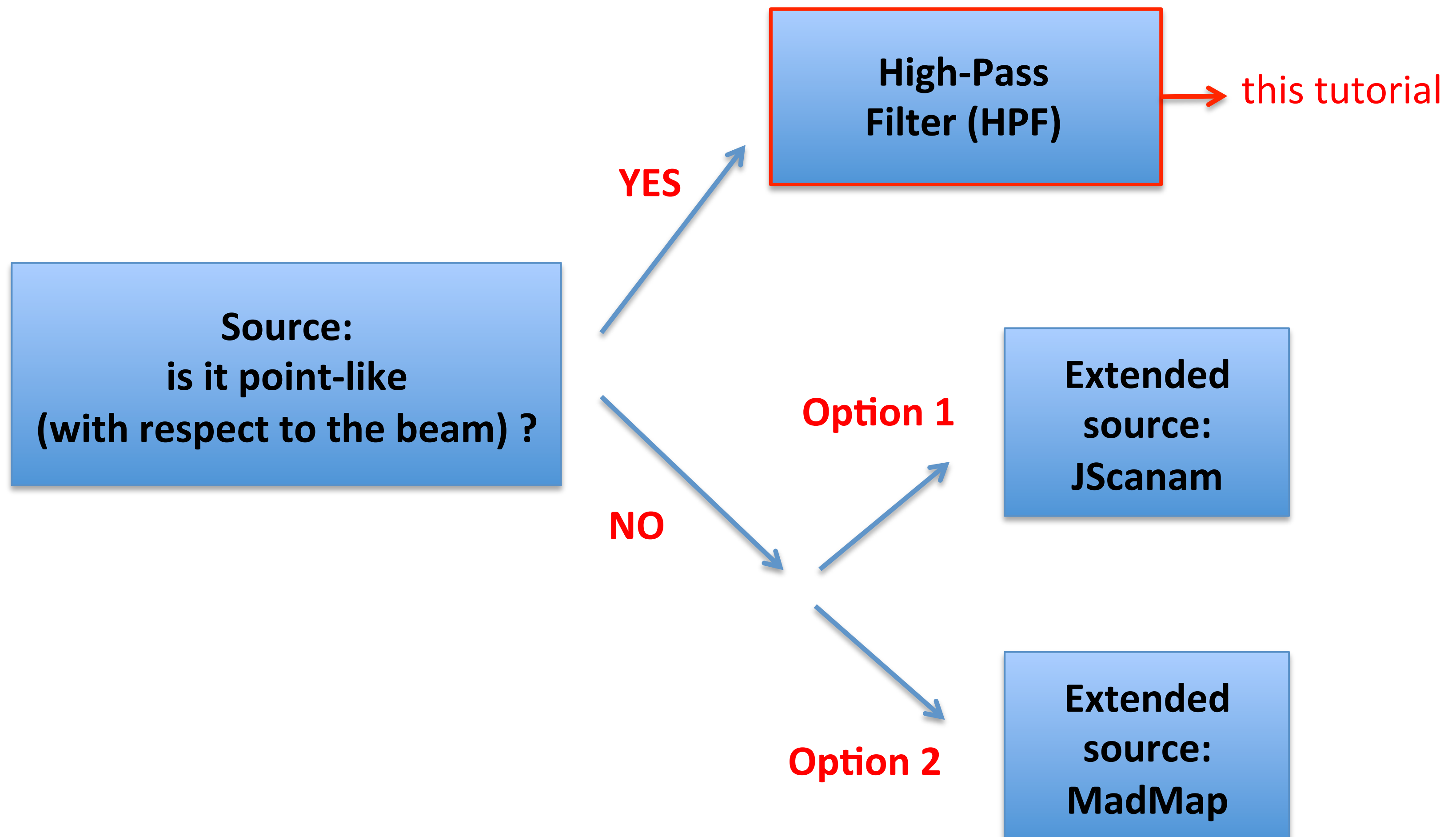
Prepared by Roberta Paladini

September 2013

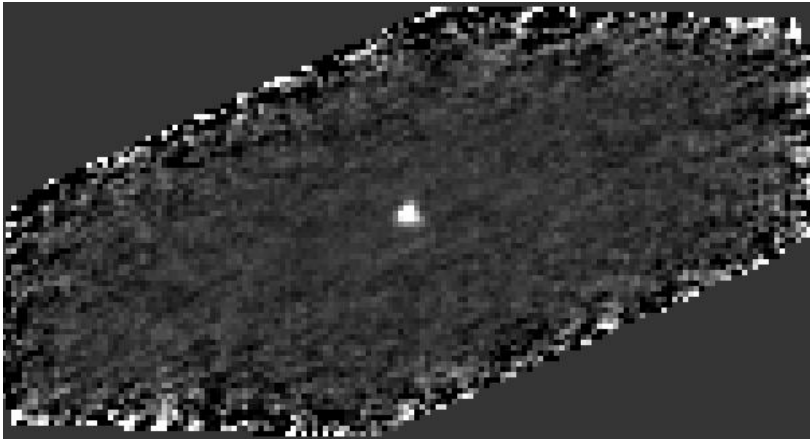
Structure of this tutorial

- Slide 3 to 35: philosophy and step-by-step walk-through of the PACS photometer High-Pass Filter pipeline
- Slide 36 to 39: Deglitching
- Slide 40: High-Pass Filter radius
- Slide 41: outpix & pixfrac
- Slide 42: turnaround removals

PACS Photometer Pipeline: 2 main branches



The HPF branch is optimal for reducing mini scan maps or large scan maps where the focus of the science is on point sources

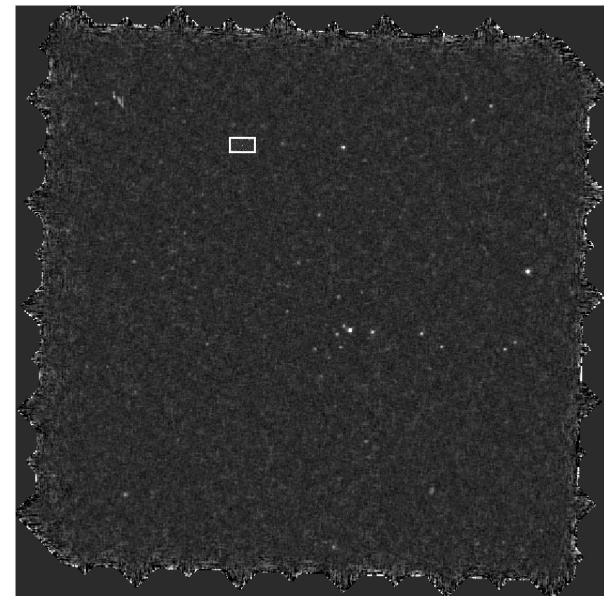


Mini scan map

a single source in the center of the field

Large scan map

multiple sources distributed in the field

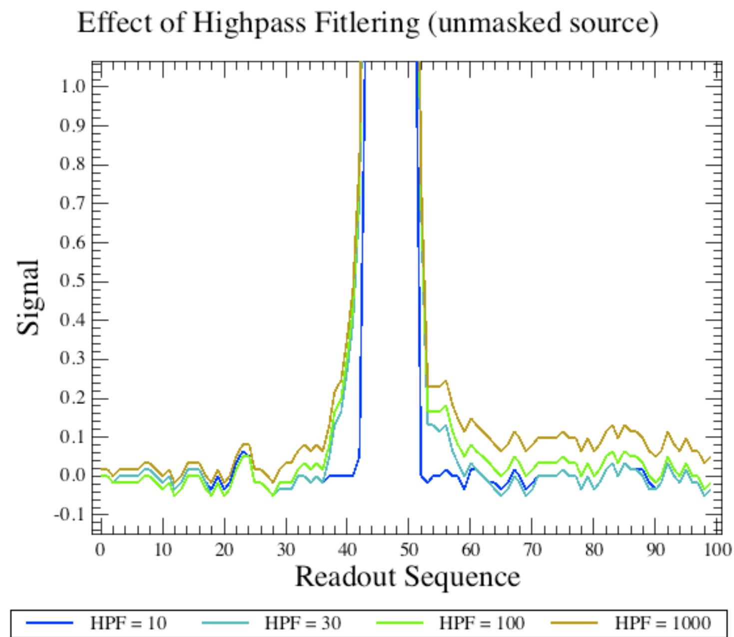


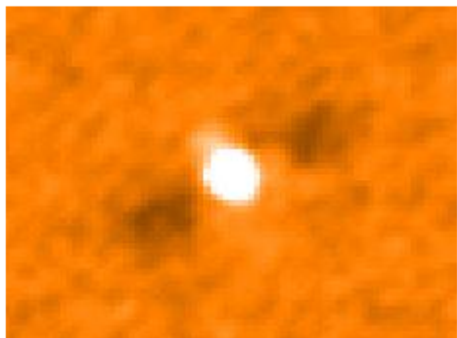
nhsc NASA Herschel Science Center

The High-Pass Filter concept

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

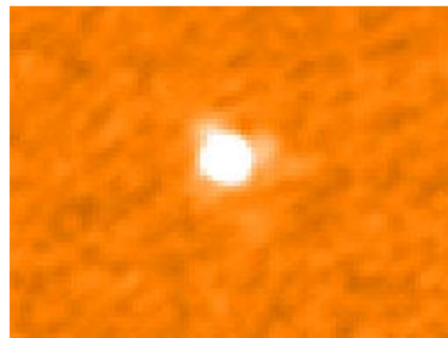
Note: When a bright source enters the filter box, it alters the estimate of the median and thus the drift removal





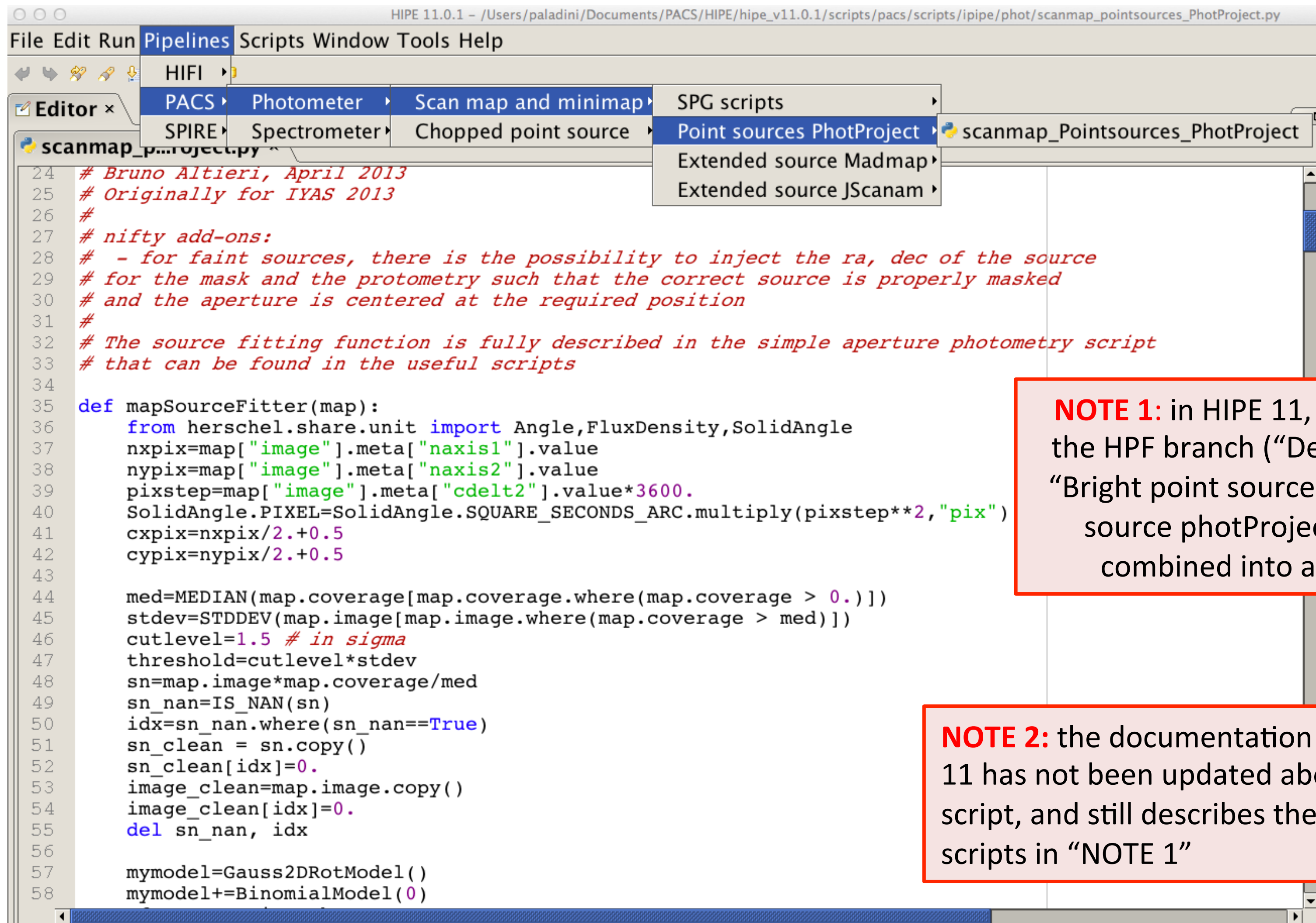
Unmasked Highpass Fitlering

Sources have to be masked !



Masked Highpass Filtering

The ipipe script for HPF processing is: scanmap_Pointsources_PhotProject



NOTE 1: in HIPE 11, 3 ipipe script for the HPF branch ("Deep survey map", "Bright point source map" "Extended source photProject") have been combined into a single script.

NOTE 2: the documentation of HIPE 11 has not been updated about this script, and still describes the 3 scripts in "NOTE 1"

Level 1 to Level 2 processing:*

NOTE: Starting with HIPE 11, the PACS photometer *ipipe* script for the High-Pass Filter branch starts from Level 1 (*calibrated cubes)* instead of Level 0 (*raw data*).

This is because the pipeline is now very stable between Level 0 and Level 1 and the user does not need to tweak the processing at the level of raw calibrated data.

HPF Pipeline: Main Concept

The main idea of the HPF pipeline revolves around generating a mask as accurate as possible to "protect" the source/s of interest when the high-pass filtering (which allows the correction for the noise) is applied.

This operation is done in multiple (3) steps.

Structure of the ipipe script:

The script is organized in 3 + 1 major blocks:

- Part 1: 1st mask guess: for each OBSID, a mask is generated by statistically identifying sources as “peaks” above the (median) background of the scan. HPF is then applied with this mask

- Part 2: 2nd updated mask: all the individual maps are combined together. A mask is built from these combined maps, and HPF is re-applied to each OBSID (e.g. Level 1) with this improved mask. Deglitching is also applied

- Part 3: 3rd updated mask: generated by using direct information on the coordinates of the source/s of interest. The mask is added to the ones previously generated, and HPF is re-applied (e.g. to Level 1 data) using this final, global mask

- Photometry: at the end of the script, aperture photometry is performed on the final map (→ THIS PART OF THE SCRIPT IS NOT TREATED IN THIS TUTORIAL)

Part 1

(from line 185 to 277)

Generate a mask for each OBSID and use this for 1st pass with HPF

- ✓ the Level 1 data of each OBSID are loaded into HIPE
- ✓ depending on the brightness of the source, the script decides how to deglitch the data (see section on Deglitching)
- ✓ the calTree, which contains all the calibration files, is loaded into HIPE
- ✓ HPF is run on the Level 1 data without masking the sources
- ✓ a map is generated from these 1st pass HPF data
- ✓ the high-coverage pixels of this map are used to identify outliers. These outliers are the "sources" which allow the generation of a mask
- ✓ this mask is applied to the data before re-running HPF
- ✓ a map from the 2nd pass HPF data is created

At the beginning of Part 1, the user has to set:

- target name, e.g. M31: object (line # 143)
- OBSID numbers: obsidall (line # 145)
- working directory, i.e. directory where to store generated maps: direc (line # 153)
- band name, i.e. blue (70 μm), green (100 μm) or red (160 μm): camera (line # 156)
- turnaround removals: lowScanSpeed/highScanSpeed or limits (line # 180/181 or 182)

… and then decide:

- whether to do (aperture) photometry as well as processing: doPhotometry (line # 166)
- use 2nd level deglitching or MMT deglitching: iindDeg (line # 167, see slide 36 to 39)
- do 2-d gaussian fit to determine source centroid and improve mask: doSourceFit (line # 168)
- get coordinates of the source/s from external file to improve mask: fromAFile (line # 169)
- If previous line set to 'True', enter filename: tfile (line # 170)

At this stage, the script starts looping over the list of OBSIDs:

1. The Level 1 data of each OBSID, *i,* in the "*obsidall*" list, is loaded into HIPE. First load the observation context:

Console ×

```
HIPE> obs = getObservation(obsidall[i], useHsa=True, instrument='PACS')
```

2. then extract the Level1:

Syntax for Blue/Red array

Console ×

```
HIPE>  frames=obs.level1.refs["HPPAVGB"].product.refs[0].product
```

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

NHSC PACS

3. Load the Calibration Tree (calTree):

Console

```
HIPE> calTree = getCalTree(obs=obs)
```

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

Now that we have the basic pieces, we can get to the core of Part 1, i.e.:

the generation of the mask for each OBSID

At this stage the mask is created **blindly**. This means that the assumption is that one does not know the location (coordinates) of the sources, and these have to be identified as "peaks" above the median background.

NHSC PACS

Let's see how this works:

4. First apply high-pass filtering without a mask
5. Then create a preliminary map
6. ….next slide

see slide 40

Console

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

HIPE> ….

HIPE> map1 = photProject(frames, calTree = calTree, calibration=True, outputPixelsize=outpixsz)
```

see slide 41

NOTE: This map is NOT good for photometry: It must be used *only* for identifying sources

6. Identify the region of the map with high coverage (i.e. coverage > “med”)
7. Use this region to estimate the signal standard deviation of the map (stdev)
8. Set a threshold (i.e. cutlevel) above which map outliers are identified
9. ….next slide

Console

```
HIPE >  med=STDDEV(map1.coverage[map1.coverage.where(map1.coverage > 0.)])
HIPE > index = map1.image.where((map1.coverage > med) & (ABS(map1.image) < 1e-2))
HIPE > signal_stdev=STDDEV(map1.image[index])
HIPE > cutlevel=3.0
HIPE > threshold=cutlevel*signal_stdev
```

9. Mask everything above the threshold → **these are the "sources"**
10. Save the mask in a .fits file
11. Mask in the timeline (i.e. in each frame, see slide 10) all readouts at the same coordinates of the map pixels with signal above the threshold

The *frames* are saved in standard fits format. The saved file can be read back into HIPE:
HIPE> frames = simpleFitsReader(maskfile)

Console

```
HIPE > mask=map1.copy()
HIPE > mask.image[mask.image.where(map1.image > threshold)] = 1.0
HIPE > mask.image[mask.image.where(map1.image < threshold)] = 0.0
HIPE > simpleFitsWriter(mask,maskfile)
HIPE > frames  = photReadMaskFromImage(frames, si=mask,maskname="HighpassMask", extendedMasking=True, calTree=calTree)
```

This is how a mask is "attached" to the Level 1 frames !

9. now that you have a mask, re-run high-pass filtering applying it
10. create map from high-pass filtered data
11. save map in a .fits file

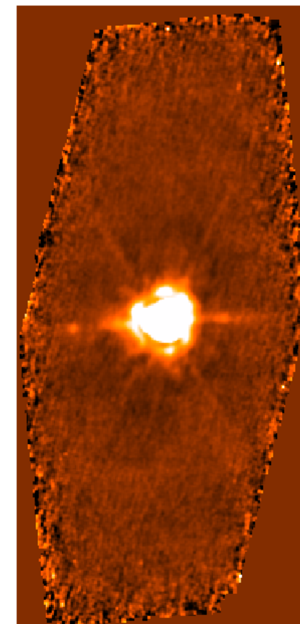
see slide 40

Console

```
HIPE  > frames  = highpassFilter(frames,hpfradius,maskname="HighpassMask", interpolateMaskedValues=True)
HIPE > .....
HIPE > map2   = photProject(frames, calibration=True,outputPixelsize=outpixsz,calTree=calTree)
HIPE > simpleFitsWriter(map2, outfile)
```

see slide 41

Map from an individual OBSID (e.g. map2)



Part 2

(from line 282 to 365)

Derive a mask from combined maps and then step back, i.e. apply the newly improved mask to do HPF on individual OBSIDs

- ✓ the maps generated in Part 1 are read-in, co-added
- ✓ the new combined map is used to generate an improved mask, following the same procedure as in Part 1 for the individual OBSID maps: the high-coverage pixels of this map are used to identify outliers, an these outliers are the "sources" which allow the generation of a mask
- ✓ with this improved mask, step back to Level 1 data of each OBSID: apply HPF using the new mask, and create the map
- ✓ co-add again the individual maps

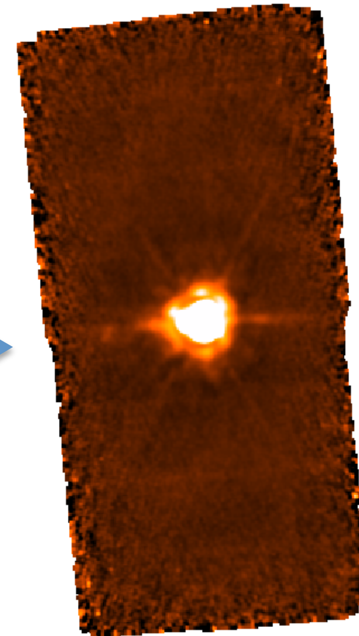
1. First, loop over the OBSIDs and co-add them:

NOTE: in Jython the loop is denoted with an indentation

Console

```
HIPE  > for i in range(len(obsidall))
HIPE >      ima=simpleFitsReader(file=direc+'Map_'+camera+'_'+str(obsidall[i])+'_maskedHPF.fits')
HIPE >      images.add(ima)
HIPE >      mosaic1=MosaicTask()(images=images,oversample=0)
```

Map from combined OBSIDs (e.g. mosaic1)



2. Then generate a mask using the combined map. The procedure is the same as in Part 1 (slide # 15 to 17):

→ Identify the region of the map with high coverage (i.e. coverage > “med”)
→ Use this region to estimate the signal standard deviation of the map (stdev)
→ Set a threshold (i.e. cutlevel) above which map outliers are identified

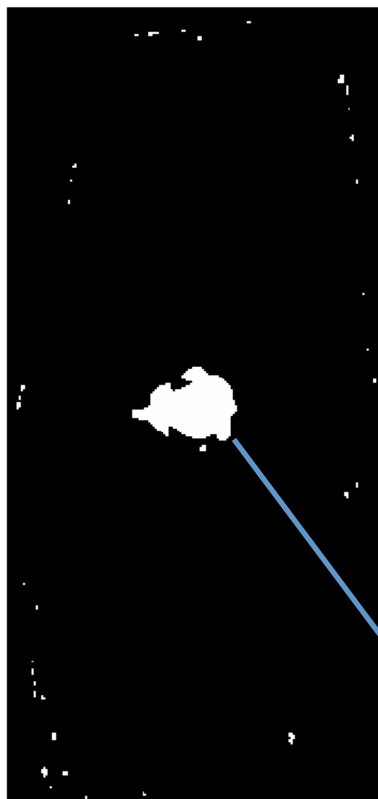
….and then:

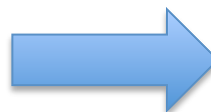
→ Mask everything above the threshold → **these are the “sources”**
→ Save the mask in a .fits file
→ Mask in the timeline all readouts at the same coordinates of the map pixels with signal above the threshold

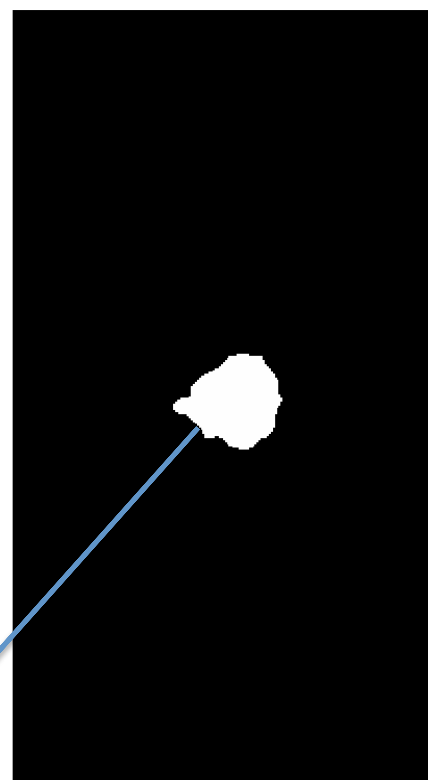
Note: in Part 2, “cutlevel” is set to 2.0 instead of 3.0 as in Part 1

Mask from Part 1:
One for each OBSID

Updated Mask from Part 2:
combined OBSIDs

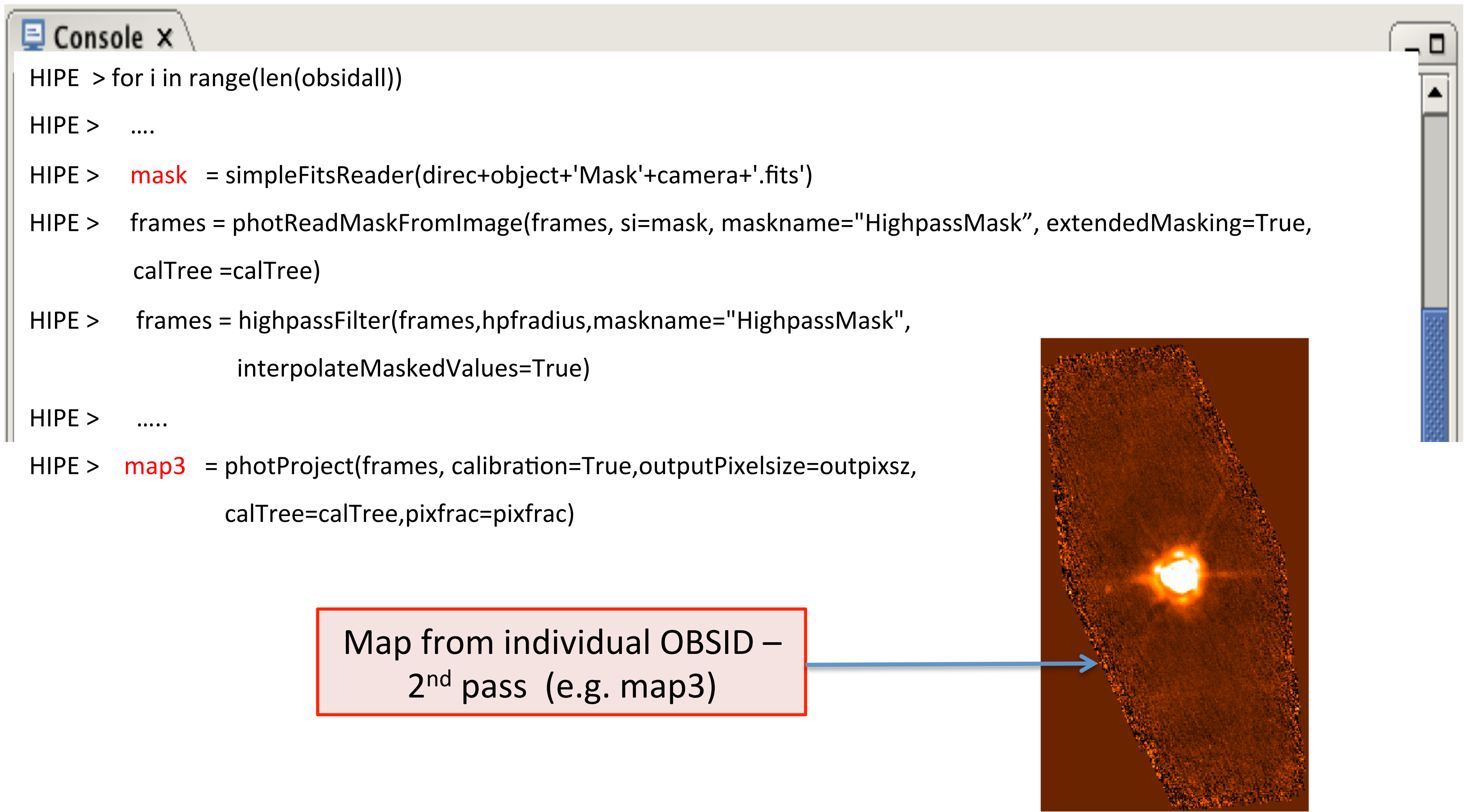






source

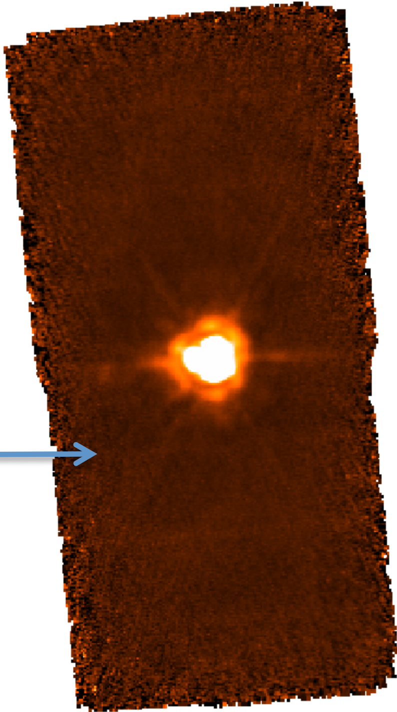
3. Step back to Level 1 data for each OBSID and apply HPF with improved mask



4. Now co-add the 2nd pass HPF maps from individual OBSIDs

```
HIPE  > images=ArrayList()
HIPE > for i in range(len(obsidall)):
HIPE >     ......
HIPE >     images.add(ima)
HIPE > mosaic2=MosaicTask()(images=images,oversample=0)
```

Map from combined OBSIDs from 2nd pass (e.g. mosaic2)



Part 3

(from line 375 to 479)

After generating a "blind" mask in Part 1 and 2, the user can now try to improve on that mask by using the information on the location of the sources (e.g. coordinates), if this information is available.

✓ To use the known coordinates of the sources, the script provides two options:

A. The user input these coordinates (e.g. from a file) and a 2-d gaussian procedure is used to "refine" the input source/s coordinates. The fitted centroids are then used to generate a mask;

B. The input coordinates (from a file or from the metadata) are used to directly generate a mask at these locations;

Option A:
2-d gaussian fitting

To use this option, doSourceFit (line # 168) has to be set to True

Then:

- ✓ Read-in the file with the input coordinates
- ✓ For each input source, create a postage stamp using a given “cropsize”
- ✓ do a 2-d gaussian fit on each postage stamp
- ✓ use fitted “rasource” and “decsource” to generate a mask

NOTE: option A also includes the alternative case (line # 397 to 399) in which the user, with no a-priori information on the location of the sources (i.e. no input file), performs a “blind” 2-d gaussian image on the map (e.g. mosaic2, see slide #) and uses the fitted centroids to generate a mask.

Option A: in practice..

NHSC PACS

1. Input source file:

The “readTargetList” function is defined at the beginning of the script (line # 104 to 134)

```
HIPE> tlist,ralist,declist=readTargetList(tfile)
```

2. For each source, convert ra/dec into pixel coordinates:

```
HIPE> pixcoor = mosaic2.wcs.getPixelCoordinates(ralist[0],declist[0])
```

Note: the current script has a bug, as only the **first** source is considered. A loop on the sources is missing !

3. Define boundaries of postage stamp in pixel coordinates: r1, r2, c1, c2.
The postage stamp size is defined by “cropsize” (default = 20 pixels), e.g:

```
HIPE> r1 = int(pixcoor[0]-cropsize/2.)
```

4. Create postage stamp:

```
HIPE > cmap    = crop(image=mosaic2,row1=int(pixcoor[0]-cropsize/2.), \
row2=int(pixcoor[0]+cropsize/2.),    \
column1=int(pixcoor[1]-cropsize/2.) \
column2=int(pixcoor[1]+cropsize/2.))
```

5. For each postage stamp, do a 2-d gaussian fit:

```
HIPE> sfit = mapSourceFitter(cmap)
```

The “mapSourceFitter” function is defined at the beginning of the script (line # 35 to 102)

6. Get the coordinates centroid, ra/dec, from the fit:

```
HIPE > rasource = Double1d([sfit["Parameters"].data[1]])
HIPE > decsource = Double1d([sfit["Parameters"].data[2]])
```

Option B:
Source coordinates from file or metadata

In this case, doSourceFit (line # 168) has to be set to False

Then:

- ✓ If input catalog file is provided, read-in the file with the input coordinates
- ✓ use the input coordinates to generate a mask

Or:

- ✓ get source/s coordinates from metadata
- ✓ use these coordinates to generate a mask

Option B: in practice..

→ **If input catalog file is available:**

```
HIPE> tlist,ralist,declist=readTargetList(tfile)
HIPE > rasource = Double1d(ralist)
HIPE > decsource = Double1d(declist)
```

→ **alternatively, if source/s coordinates are read-in from metadata:**

```
HIPE > rasource  = Double1d([obs.meta['ra'].value])
HIPE > decsource = Double1d([obs.meta['dec'].value])
```

In common to option A and B:

→ With the source/s coordinates ("rasource", "decsource") – either from 2-d gaussian fit/s or direct input/s from catalog/metadata – the new mask can now be generated:

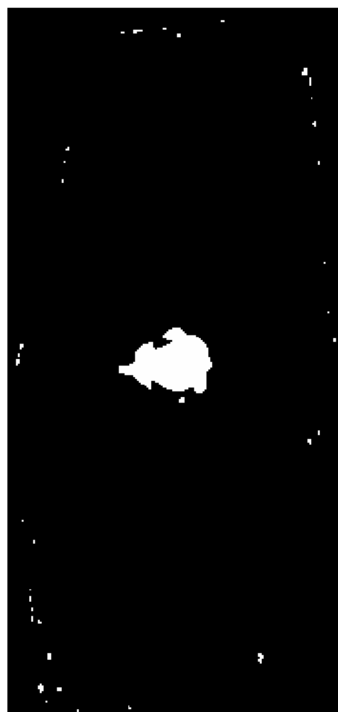
Console ×

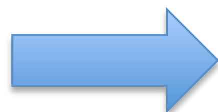
```
HIPE > mfc = MaskFromCatalogueTask()

HIPE > mask1 = mask.copy()

HIPE >  cmap  = mfc(mask1,rasource,decsource,Double1d(len(rasource),radius),copy=1)
```

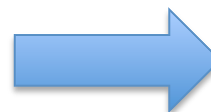
Mask from Part 1:
from **each** OBSID

Updated Mask from Part 2:
from **combined** OBSIDs

Updated Mask from Part 3:
From **known** source/s coords.





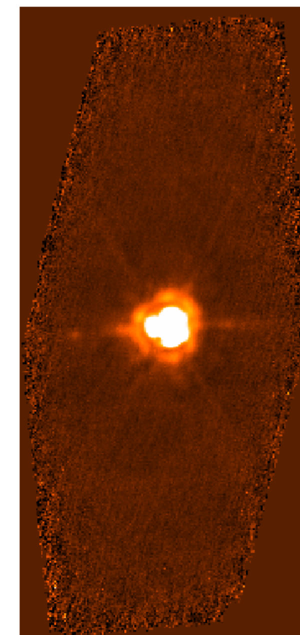


→ Now step back and apply this new mask (improved version of the mask from Part 1, 2 and 3) to do HPF on the Level 1 data of each OBSID:

Console X

```
HIPE > for i in range(len(obsidall)):
HIPE >     ...
HIPE >     frames = photReadMaskFromImage(frames, si=cmap, maskname="HighpassMask", extendedMasking=True,calTree = calTree)
HIPE >    frames = highpassFilter(frames,hpfradius,maskname="HighpassMask",interpolateMaskedValues=True)
HIPE >    ....
HIPE >    map4    = photProject(frames, calibration=True,outputPixelsize=outpixsz, calTree=calTree,pixfrac=pixfrac)
```

Map from individual OBSID – 3rd pass (e.g. map4)

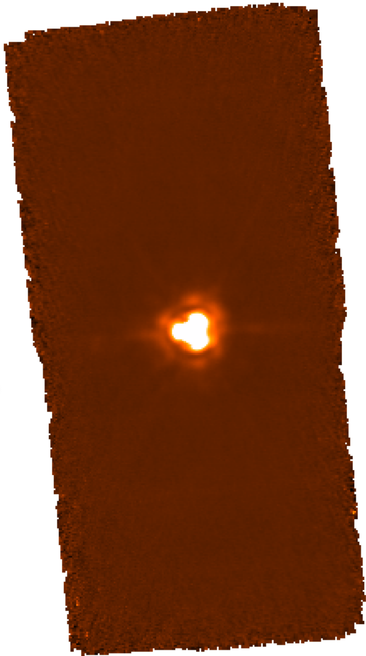


→ now co-add the 3rd pass HPF maps from individual OBSIDs: this is the FINAL map !

Console X

```
HIPE > images=ArrayList()
HIPE > for i in range(len(obsidall)):
HIPE >    ......
HIPE >    images.add(ima)
HIPE > .....
HIPE > mosaic3=MosaicTask()(images=images,oversample=0)
```

Map from combined OBSIDs - 3rd pass (e.g. mosaic3)



This concludes the walk through the PACS photometer HPF pipeline !

Deglitching
(more information in tutorial # 402)

- At the beginning of the script (line # 167), the user has to set the switch iindDeg to either True or False;
- this means that the user has to decide whether to use 2nd Level Deglitching or MMT Deglitching;
- The default is to use MMT Deglitching (→ iindDeg = False)
- Note that, in the script, deglitching is performed in Part 2 (line # 326 to 333)

What are 2nd Level Deglitching and MMT Deglitching?

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 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

MMT Deglitching

For faint sources, or not enough redundancy, we deglitch by applying the MMT deglitching task:

Console ×

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

The set of parameters provided above works well with most observations

2nd Level Deglitching

For relatively bright sources and high redundancy, we deglitch by applying the 2nd Level Deglitching task:

Console

```
HIPE > s = Sigclip(10, 30)
HIPE > s.behavior = Sigclip.CLIP
HIPE > s.outliers = Sigclip.BOTH_OUTLIERS
HIPE >  s.mode = Sigclip.MEDIAN
HIPE > if ( iindDeg ):
HIPE >      mapDeglitch(frames, algo = s, deglitchvector="timeordered", calTree=calTree)
```

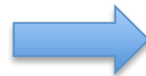
Outliers are detected with a **sigma-clipping** algorithm and flagged as glitches. Both **positive** and **negative** outliers are detected. By default, outliers are detected with respect to the **median**.

nhsc
NASA Herschel Science Center

High-Pass Filter Radius: hpfradius

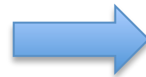
The default values are:

Averagely bright source



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

Bright source



```
HIPE> if camera=='blue':
HIPE >   hpfradius=25
HIPE > elif camera=='red':
HIPE >   hpfradius=40
```

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

NOTE: the values above are optimized for point-sources. If the source of interest is slightly extended (a few times the fwhm), the use of larger “hpfradius” is recommended.

Making the map: outpixsz & pixfrac

NHSC PACS

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.**

Console

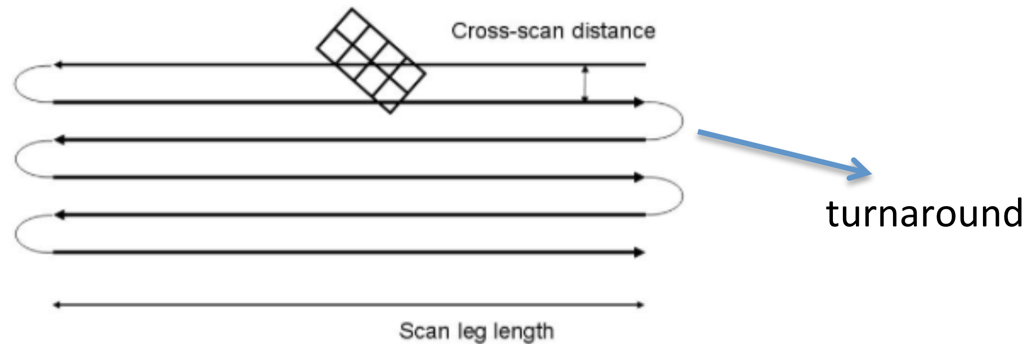
```
HIPE  > map=photProject(frames, outputPixelsize=outpixsz, calTree=calTree, pixfrac=pixfrac)
```

DEFAULT VALUES:

```
HIPE > If camera == 'blue':
HIPE >     outpixsz = 1.2
HIPE > elif camera =='red':
HIPE >     outpixsz = 2.4.

HIPE > pixfrac = 0.1
```

Turnaround removal

NOTE: in the script, the frames corresponding to the telescope turnaround are removed each time before creating the map. <u>Turnaround frames are characterized by much lower coverage (hence sensitivity) then normal science frames.</u>



turnaround

```
HIPE > frames = filterOnScanSpeed(frames,limit=limits)
```

Turnaround frames are executed at different scan speed than normal science frames

The parameter "limit" is set at the beginning of the script (line # 182).
"Limits = 10" means: remove all the frames with a scan speed 10% higher/lower than science frames scan speed.

Thank you !