PACS Performance Verification Phase Photometer Point Source AOT Evaluation: dithered vs. non-dithered Test case: asteroid 360 Carlova

Markus Nielbock, Ulrich Klaas
Max–Planck–Institut für Astronomie, Königstuhl 17, D-69117 Heidelberg, Germany

1 Introduction

The PACS Photometer point source AOT can be executed with and without dithering using the focal plane chopping mirror. Two chopper offsets of ±7/3 pixels around the nominal throw are used for this. This report deals with the evaluation and comparison of these two observing modes based on a dedicated observing programme. It was executed on OD 108 during the PACS PV phase using the asteroid 360 Carlova as the prime target. The details of the observations are summarised in Tab. 1. The expected noise levels and signal-to-noise ratios are given in Tab. 2.

Table 1: Details of the observations.

OBSID	AOR label	Filters	Dithering	Execution Time [s]	
				Total	On Source
1342182969	PVPhotAOTVal_511B_StdPSndith_blu_cld_Carlova_0001	blue/red	no	516	248
1342182970	PVPhotAOTVal_511B_StdPSndith_grn_cld_Carlova_0001	green/red	no	516	248
1342182971	PVPhotAOTVal_511B_StdPSdith_grn_cld_Carlova_0001	green/red	yes	516	248
1342182972	PVPhotAOTVal_511B_StdPSdith_blu_cld_Carlova_0001	blue/red	yes	516	248

Table 2: Expected noise levels and signal-to-noise ratios as estimated by HSPOT. The noise predictions were modified to correspond to the effective on-source time that is reduced by discarding the first of four readouts per chopper plateau during the data reduction. Hence, the listed 1σ noise values are higher by a factor of $\sqrt{4/3}$ than what was derived by HSPOT. The flux estimates are based on applying a *complex shape model*, private communication by T. Müller.

OBSID	blue/green			red		
	Expected flux [mJy]	1σ noise [mJy]	S/N	Expected flux [mJy]	1σ noise [mJy]	S/N
1342182969	3485	2.34	1489	884	3.59	246
1342182970	2024	2.49	813	884	3.59	246
1342182971	2024	2.49	813	884	3.59	246
1342182972	3485	2.34	1489	884	3.59	246

2 Product generation up to level 2

The data were processed starting from the TM files. The standard PHOT point source AOT pipeline script (`pacs/toolboxes/spg/pacsphotpointsource.py`) with modifications[1] introduced by B. Altieri was used. The initial calibration block processing was omitted as well as the `photMMTDeglitching` and `cleanPlateauFrames` tasks. The `photAvgPlateau` task was executed with the `skipFirst=True` option in order to discard the first detector readout after a chopper transition which is severely affected by slow detector time constants and reflecting the correct signal level. This leads to a reduction of the effective on-source time that has to be considered when comparing the results with the HSPOT predictions. The `photAvgDith` task was applied with a $3\,\sigma$ deglitching algorithm. The final result was a level 2 product that had shifted and combined the source signal of all individual frames using the `allInOne=1` setting in the `photProjectPointSource` task. The resulting default pixel size of $3\farcs2$ for the blue and green spectral ranges, and $6\farcs4$ for the red spectral range was set by specifying `outputPixelsize`. The task was executed with the `calibration=True` option.

```
map = photProjectPointSource(frames,allInOne=1,outputPixelsize=3.2,calTree=calTree,calibration=True)
```

3 Analysis

The subsequent analysis was done outside of HCSS using the IRAF software. Aperture photometry was performed on the co-added image of the source. Aperture radii as specified in Tab. 3 were used in order to be as comparable with the HSPOT instrument model assumptions as possible. Fig. 1 displays the aperture selection for the blue detector.

3.1 Details of applied aperture photometry

In order to be comparable to the assumptions made for the noise assessment in HSPOT, we have to apply the same apertures. They are based on the model calculation as presented in "Summary and Conclusions", A. Poglitsch, SVRIII presentation, 9 November 2007[2] (see Tab. 3). The chosen apertures only include a certain fraction of the measured source flux (c.f. PICC-ME-TN-033, V 0.2, 27-Oct-2009). The real source flux can be obtained by applying a correction factor according to the fractions provided.

Table 3: Details of aperture photometry consistent with the PACS instrument model used in HSPOT. The encircled energy fractions have been obtained from the analysis in PICC-ME-TN-033.

	Pixel size	Aperture			Encircled energy
Filter	[″]	Area [pix]	Radius [pix]	Radius [″]	Fraction [%]
blue	3.2	6.17	1.40	4.48	59
green	3.2	8.85	1.68	5.38	61
red	6.4	5.52	1.33	8.51	61

3.2 Sky noise and coverage correction

The background level and residual sky noise were evaluated employing an annulus around the source. It was chosen to be wide enough to cover the blank sky without contaminations by the extended low level PSF emission. The pixel coverage varies across the final image, as shown in Fig. 1 (right panel) with the coverage being highest at the target position but being already decreased for the sky backfound reference position. Hence, the rms noise determined by this method is worse than the real sky noise that affects the target. Therefore, it had to be corrected by introducing the pixel coverages at the target position and on the sky. The mean pixel coverages were measured by aperture photometry on the coverage map using apertures i) as applied for the target photometry, ii) with the inner and outer radius of the sky annulus. The latter two results were subtracted, providing the mean pixel coverage within the sky annulus. The square root of the ratio of mean pixel coverages

[1] `http://www.herschel.be/twiki/bin/view/HSC/HowToPACSPointSourcePipeline`
[2] `http://pacs.ster.kuleuven.be/SVRIII/FMILT_reports/WBS/SVRIIIpres/003/001/27%20AP_Summary_Conclusions.pdf`

of the photometry aperture and the sky annulus is then taken as the conversion factor to correct the measured residual sky noise.

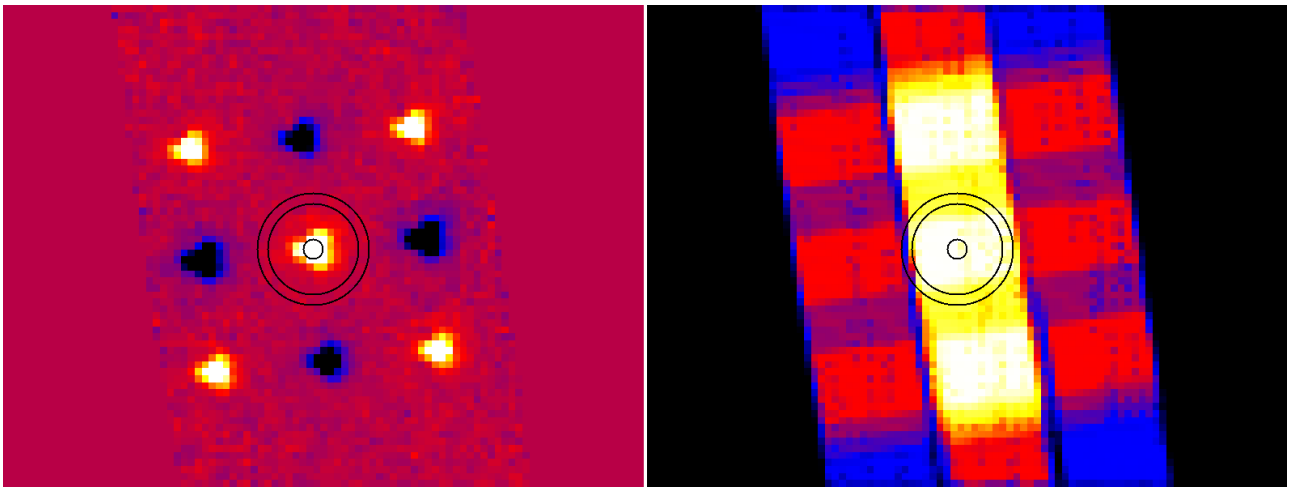

Figure 1: Synopsis of the level 2 products of the blue, dithered Carlova point source observation. Left: co-added image, right: coverage map. The coverage map displays the number of pixels that have seen a certain fraction of the sky shown in the image. The black circles represent the aperture with a radius of 1.4 pixels, and a sky annulus with an inner radius of 7.0 pixels and a width of 1.5 pixels. In order to correct for the different pixel coverages in the aperture and the sky annulus that is used to determine the residual sky noise, we calculated the mean pixel coverage both in the target aperture and in the sky annulus. The noise determined within the annulus was then corrected by dividing it by the square root of the ratio of the mean pixel coverages.

3.3 Calculation of residual noise and the signal-to-noise ratio

One has to distinguish between the S/N ratio of the source peak ($S/N_{\rm peak}$) and the integrated S/N ratio ($S/N_{\rm int}$). The former is a measure of the detectability of the source in the image, while the latter represents the accuracy of the measured integrated flux density.

$$S/N_{\rm peak} = \frac{F_{\rm peak}}{\sigma} \tag{1}$$

The pixel noise σ is the standard deviation of the background noise. Its unit is given in mJy per pixel.

$$S/N_{\rm int} = \frac{F_{\rm int}}{\Delta F} \tag{2}$$

In this case, ΔF is defined as the uncertainty of the photometry within an aperture of n pixels. It can be derived by:

$$\Delta F = \frac{\sigma \cdot n}{\sqrt{n}} = \sigma \cdot \sqrt{n} \tag{3}$$

This results in:

$$S/N_{\rm int} = \frac{F_{\rm int}}{\sigma \cdot \sqrt{n}} \tag{4}$$

Table 4: Pixel coverage photometry. The pixel size of the blue and green filter observations is 3.″2, the red detector has a pixel size of 6.″4.

OBSID	Filter	Dithering	Aperture radius [pix]	Number of pixels	Pixel coverage integrated
1342182969	blue	no	1.4	6.3992	55.849
			7.0	154.1851	1308.580
			8.5	227.2160	1780.286
1342182972	blue	yes	1.4	6.4153	54.457
			7.0	154.1394	1265.687
			8.5	227.2455	1748.869
1342182970	green	no	1.68	9.1734	80.047
			6.50	133.0533	1127.089
			8.00	201.2534	1600.578
1342182971	green	yes	1.68	9.1928	78.399
			6.50	133.0861	1097.368
			8.00	201.3863	1554.883
1342182969	red	no	1.33	5.9831	52.504
			3.50	38.8574	325.687
			4.50	63.2675	482.721
1342182970	red	no	1.33	5.9831	52.495
			3.50	38.8574	325.714
			4.50	63.2675	482.860
1342182971	red	yes	1.33	5.9831	52.429
			3.50	38.8574	316.054
			4.50	63.2675	483.092
1342182972	red	yes	1.33	5.9831	52.461
			3.50	38.8574	315.880
			4.50	63.2675	483.112

Table 5: Pixel coverage noise correction. The results of the sky annuli were obtained by subtracting the number of pixels and integrated pixel coverages obtained for the matching aperture radii.

OBSID	Filter	Dithering	Image region	Number of pixels	Pixel coverage integrated	Pixel coverage mean	Sky noise correction
1342182969	blue	no	aperture	6.3992	55.849	8.728	0.860
			sky annulus	73.0309	471.706	6.459	
1342182972	blue	yes	aperture	6.4153	54.457	8.489	0.882
			sky annulus	73.1061	483.182	6.609	
1342182970	green	no	aperture	9.1734	80.047	8.762	0.892
			sky annulus	68.2001	473.489	6.943	
1342182971	green	yes	aperture	9.1928	78.399	8.528	0.886
			sky annulus	68.3002	457.515	6.699	
1342182969	red	no	aperture	5.9831	52.504	8.775	0.856
			sky annulus	24.4101	157.034	6.433	
1342182970	red	no	aperture	5.9831	52.495	8.778	0.856
			sky annulus	24.4101	157.145	6.438	
1342182971	red	yes	aperture	5.9831	52.429	8.763	0.884
			sky annulus	24.4101	167.038	6.843	
1342182972	red	yes	aperture	5.9831	52.461	8.768	0.884
			sky annulus	24.4101	167.232	6.851	

4 Results

The results of the aperture photometry is given in Tab. 6. The derived values reflect the pure numerical result without any correction applied.

Table 6: Results of the uncorrected aperture photometry.

OBSID	Filter	Measured flux density [mJy*]		Measured noise [mJy*]		S/N	
		Peak [mJy*/pix]	Integrated aperture	1σ noise [mJy*/pix]	Photometric error on sky	Peak 1 pixel	Integrated point source
1342182969	blue	436.2	1301.035	0.620	1.568	704	830
1342182972	blue	449.2	1343.560	0.633	1.603	710	838
1342182970	green	241.1	1023.226	0.755	2.286	320	448
1342182971	green	239.1	1032.926	0.938	2.845	255	363
1342182969	red	168.5	429.083	1.336	3.269	126	131
1342182970	red	174.4	454.680	1.755	4.292	99	106
1342182971	red	177.1	456.563	1.479	3.619	120	126
1342182972	red	177.4	466.675	1.993	4.874	89	96

* based on on-ground calibration

The real S/N ratios are obtained by applying a coverage correction for the derived residual sky noise, presented in Tab. 7. The sky noise determined during the photometry is based on a poorer pixel coverage than achieved at the position of the target. The applicable correction factors are provided in Tab. 5.

Table 7: Results of the aperture photometry with the sky noise being corrected for the pixel coverage.

OBSID	Filter	Measured flux density [mJy*]		Corrected noise [mJy*]		S/N		
		Peak [mJy*/pix]	Integrated aperture	1σ noise [mJy*/pix]	Photometric error on sky	Peak 1 pixel	Integrated point source	Fraction HSPOT
1342182969	blue	436.2	1301.035	0.533	1.348	818	965	65%
1342182972	blue	449.2	1343.560	0.558	1.414	805	950	64%
1342182970	green	241.1	1023.226	0.673	2.039	358	501	62%
1342182971	green	239.1	1032.926	0.832	2.522	287	410	50%
1342182969	red	168.5	429.083	1.144	2.799	147	153	62%
1342182970	red	174.4	454.680	1.503	3.676	116	124	50%
1342182971	red	177.1	456.563	1.307	3.198	135	143	58%
1342182972	red	177.4	466.675	1.761	4.308	101	108	44%

* based on on-ground calibration

Table 8: Results of the photometry after applying the aperture correction.

OBSID	Filter	Dithering	Measured flux density [mJy*]			Fraction	
			small aperture	corrected (Tab. 3)	large aperture	model	small/large
1342182969	blue	no	1301.035	2205.144	2127.198	59%	62%
1342182972	blue	yes	1343.560	2277.220	2244.891	59%	60%
1342182970	green	no	1023.226	1677.420	1626.092	61%	63%
1342182971	green	yes	1032.926	1693.321	1563.584	61%	66%
1342182969	red	no	429.083	703.415	619.522	61%	69%
1342182970	red	no	454.680	745.377	678.854	61%	67%
1342182971	red	yes	456.563	748.464	663.232	61%	69%
1342182972	red	yes	466.675	765.041	708.894	61%	66%

* based on on-ground calibration

In order to compare the fluxes obtained by aperture photometry with the expected total fluxes of the object, an aperture correction has to be applied (c.f. Tab. 3). Theoretically, this should compensate for the flux that is outside of the aperture used in the photometry. However, by measuring the target fluxes within an aperture that includes the entire PSF of the source (as verified by visual inspection), we typically achieve lower flux

values than the aperture correction suggests. This is indicated in Tab. 8 in the column "'Fraction"', where the "'small/large"' ratio gives the suggested ration from the measurement and the "'model"' is taken from Tab. 3).

5 Conclusions

1. The size of the low level component of the source PSF impedes a consistent determination of the background flux and noise level. It must be expected that the PSFs of the images of bright sources produced by the chop-nod observing technique may blend into each other. This would make it difficult to select a proper region for the calculation of the sky noise.

2. The signal-to-noise (S/N) ratios of the dithered and non-dithered observations do not differ significantly. Due to the pixel under-sampling of the PSF in the blue filter, dithered observations should be preferred to better recover the source flux.

3. When employing suitable photometric techniques, the S/N ratios predicted by HSPOT for the PACS PHOT point source AOT are recovered at a level of 44 to 65% by the real measurements. This is based on the assumption that the source flux of variable targets predicted for the epoch of the observation is correct.

Appendices

A Carlova images

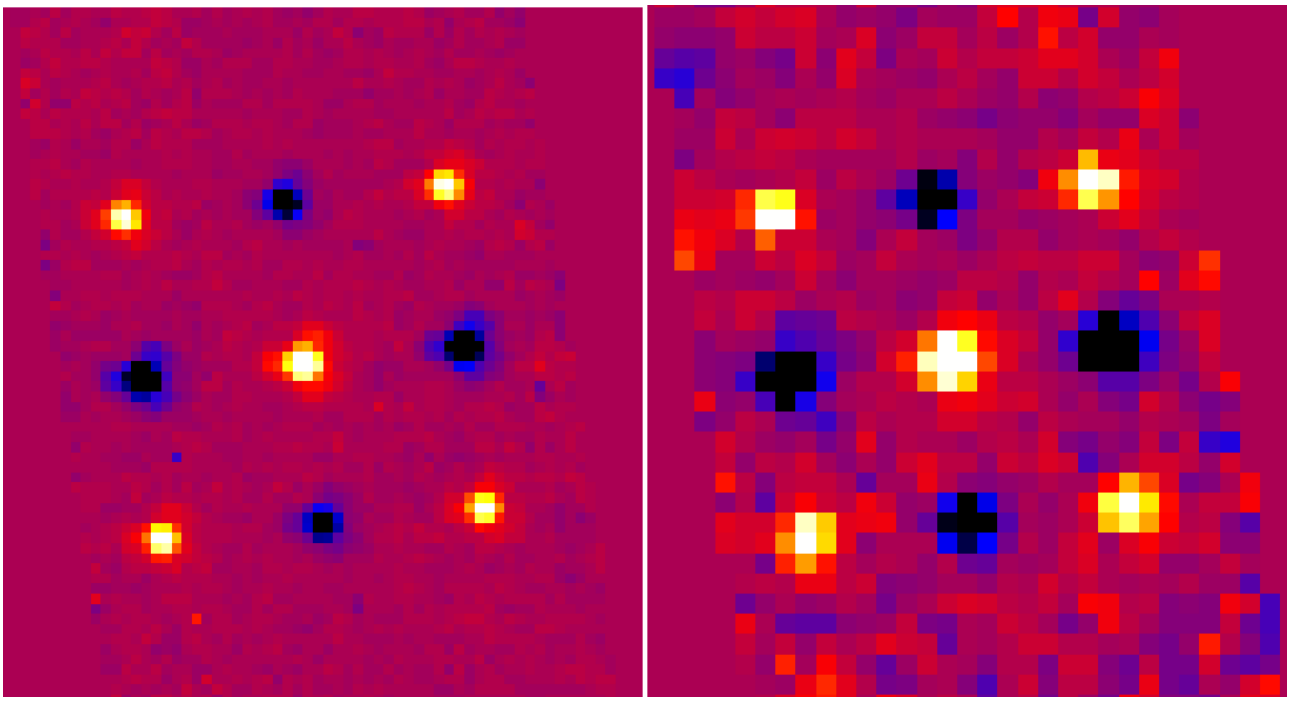

Figure 2: Synopsis of the level 2 products of the green and red, dithered Carlova point source observation (OBSID 1342182971). Left: green spectral band, right: red spectral band.

B HCSS reduction script: Carlova_OD108_tm.py

```
from herschel.ia.obs                import ObservationContext
from herschel.ia.pal.pool.lstore    import *
from herschel.ia.pal.pool.db        import DbPool
from herschel.ia.pal                import PoolManager
from herschel.ia.pal.pool.lstore    import *
from herschel.ia.pal.pool.db        import DbPool
from herschel.ia.pal.browser        import ProductBrowser
from herschel.ia.pal.browser        import ProductBrowserFrame
from herschel.ia.pg                 import PgPluginManager
from herschel.ia.pg                 import ProcessEnvironment
from herschel.share.util            import Configuration

from herschel.pacs.spg              import PacsLevel0Plugin
from herschel.pacs.spg              import PacsCalPlugin
from herschel.pacs.cal              import PacsCal

from herschel.share.fltdyn.time     import FineTime
from herschel.store.api             import *

from herschel.ia.pal.pool.lstore    import LocalStoreFactory
from herschel.ia.pal                import ProductStorage
from herschel.ia.pal.browser        import *
from herschel.pacs.toolboxes.all    import *
from java.util                      import Date
from java.lang                      import Runtime

calTree=getCalTree()
storedir="/home/nielbock/PACS/DATA/PV-OD0108/"

# Indicate which detector data to process: blue/red
filter="blue"

# TM files
tmfile=['OD0108_OBSID1342182969_PACS_Calibration_PVPhotAOTVal_2-
     PVPhotAOTVal_511B_StdPSndith_blu_cld_Carlova_0001.tm','
     OD0108_OBSID1342182970_PACS_Calibration_PVPhotAOTVal_2-
     PVPhotAOTVal_511B_StdPSndith_grn_cld_Carlova_0001.tm','
     OD0108_OBSID1342182971_PACS_Calibration_PVPhotAOTVal_2-
     PVPhotAOTVal_511B_StdPSdith_grn_cld_Carlova_0001.tm','
     OD0108_OBSID1342182972_PACS_Calibration_PVPhotAOTVal_2-
     PVPhotAOTVal_511B_StdPSdith_blu_cld_Carlova_0001.tm']

# enter OBSIDs, dithering information and photometric range for each observation to process
obsid=[1342182969,1342182970,1342182971,1342182972]
dither=["ndith","ndith","dith","dith"]
if (filter == "blue"):
   channel=["blue","green","green","blue"]
   pixsize=3.2
if (filter == "red"):
   channel=["red","red","red","red"]
   pixsize=6.4

# retrieve latest pointing product
ppfits=storedir+"pp_OD108.fits"
pp = simpleFitsReader(file=ppfits)

for i in range(len(obsid)):
   #
   # extract frames from TM file
   print "Extract detector data : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
        1000)
   seq=readTm(tmfile[i])
   pdfs = extractDataframes(seq)
   mix  = decomposeDataframes(pdfs,filter)
   products=mix.getProductNames()
   frames=mix[products[0]]
   frames = addObcp2Frames(frames,seq)

   print "Extract HK data : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
   photHK = getPhotHk(seq=seq)

   print "Plot chopper position : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
        1000)
   ftime=frames.getStatus("FINETIME")
   dtimes=(ftime-ftime[0])*1e-6
   cpr=frames.getStatus("CPR")
   PlotXY(dtimes,cpr)
```

```
#
################################################
# Level 0 -> Level 0.5
################################################
#
print "Add pointing : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photAddInstantPointing(frames,pp, calTree=calTree)
#
print "findBlocks : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = findBlocks(frames, calTree=calTree)
#
print "photFlagBadPixels : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photFlagBadPixels(frames, calTree=calTree)
#
print "photFlagSaturation : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photFlagSaturation(frames, calTree=calTree)
#
print "photConvDigit2Volts : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photConvDigit2Volts(frames, calTree=calTree)
#
print "photCorrectCrosstalk : " , Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
    1000)
frames = photCorrectCrosstalk(frames, calTree= calTree)
#
#print "photMMTDeglitching : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
#frames = photMMTDeglitching(frames, scales=1, nsigma=5)
#
#print "addUtc : " , Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
#frames = addUtc(frames)
#
print "convertChopper2Angle : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
    1000)
frames = convertChopper2Angle(frames, calTree=calTree)
#
print "photAssignRaDec : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photAssignRaDec(frames, calTree=calTree)
#
#print "cleanPlateauFrames : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
#frames = cleanPlateauFrames(frames, calTree = calTree)
#
################################################
# Level 0.5 -> Level 1
################################################
#
print "photMakeDithPos : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photMakeDithPos(frames)
#
print "photMakeRasPosCount : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photMakeRasPosCount(frames)
#
print "photAvgPlateau : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photAvgPlateau(frames,skipFirst=True)
#
print "photAddPointings4PointSource : ", Date(), " - " , Runtime.getRuntime().freeMemory() /
    (1024 * 1000)
frames  = photAddPointings4PointSource(frames)
#
print "photDiffChop : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photDiffChop(frames)
#
print "photAvgDith : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photAvgDith(frames,sigclip=3.)
#
print "photDiffNod : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
frames = photDiffNod(frames)
#
signal_chopnod_0=frames.getImage(0)
signal_chopnod_1=frames.getImage(1)
signal_chopnod_2=frames.getImage(2)
signal_chopnod_3=frames.getImage(3)
signal_chopnod_4=frames.getImage(4)
signal_chopnod_5=frames.getImage(5)
im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod0.fits"
simpleFitsWriter(product=signal_chopnod_0,file=im_fits)
im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod1.fits"
simpleFitsWriter(product=signal_chopnod_1,file=im_fits)
im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod2.fits"
simpleFitsWriter(product=signal_chopnod_2,file=im_fits)
im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod3.fits"
simpleFitsWriter(product=signal_chopnod_3,file=im_fits)
```

```
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod4.fits"
    simpleFitsWriter(product=signal_chopnod_4,file=im_fits)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_chopnod5.fits"
    simpleFitsWriter(product=signal_chopnod_5,file=im_fits)
    #
    print "photCombineNod : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 * 1000)
    frames = photCombineNod(frames)
    #
    signal_dith_0=frames.getImage(0)
    signal_dith_1=frames.getImage(1)
    signal_dith_2=frames.getImage(2)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_dith0.fits"
    simpleFitsWriter(product=signal_dith_0,file=im_fits)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_dith1.fits"
    simpleFitsWriter(product=signal_dith_1,file=im_fits)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_dith2.fits"
    simpleFitsWriter(product=signal_dith_2,file=im_fits)
    #
    print "photRespFlatfieldCorrection : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024
        * 1000)
    frames = photRespFlatfieldCorrection(frames, calTree = calTree)
    #
    signal_dith_0=frames.getImage(0)
    signal_dith_1=frames.getImage(1)
    signal_dith_2=frames.getImage(2)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_flat_dith0.fits"
    simpleFitsWriter(product=signal_dith_0,file=im_fits)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_flat_dith1.fits"
    simpleFitsWriter(product=signal_dith_1,file=im_fits)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_flat_dith2.fits"
    simpleFitsWriter(product=signal_dith_2,file=im_fits)
    #
    ################################################
    # Level 1 -> Level 2
    ################################################
    print "photProjectPointSource : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
        1000)
    map = photProjectPointSource(frames,allInOne=1,outputPixelsize=pixsize,calTree=calTree,
        calibration=True)

    # Write results to FITS files (indicate storage directory and file names)
    print "Writing final images : ", Date(), " - " , Runtime.getRuntime().freeMemory() / (1024 *
        1000)
    im_fits=storedir+"OD0108_"+str(obsid[i])+"_Carlova_"+channel[i]+"_"+dither[i]+"_allinone.fits"
    simpleFitsWriter(product=map,file=im_fits)

System.gc()
```