PACS Photometric Check : MADmap vs Photproject

Nicolas BILLOT & Babar ALI
NHSC
nbillot@ipac.caltech.edu

Résumé

This document presents a comparison of point source photometric measurements carried out on maps generated by MADmap (1st and 3rd generation INVNTT files) and Photproject. The aim is to characterize the photometric robustness of MADmap as a mapmaker against the better tested Photproject/highpass-filtering combination.
It appears that MADmap absolute photometry is well within 5% of Photproject fluxes in both channels. The different flavors of MADMap preprocessings do not alter point source photometry. Yet, the third generation of MADMap Calibration files do change the photometry by 10-25% with respect to the first generation of INVNTT files. Lastly, uncertainties on MADmap measurements are on average systematically higher than Photprojects' by $\sim 20-40\%$ due to the very aggressive filter settings used in combination with Photproject.

1 Introduction

The photometric calibration of the PACS Photometer uses the highpass-filtering/Photproject combination. Highpass filtering is used to filter out the long timescale variations from the data cube that may originate from the detectors 1/f noise, or from the actual extended emission present in the sky when the observation involves telescope motion. If well parameterized, the output of the highpass-filtering/Photproject data reduction is a flat-background map, devoid of any extended emission or stripping, containing contrasted compact sources.
There are several mapmakers capable of filtering out instrumental low-frequency noise while leaving the celestial extended emission unaltered. MADmap is one of them, and it is the official PACS pipeline mapmaker for extended emission. As such, we need to characterize the photometric quality of this algorithm as far as point sources are concerned.
As of today, MADmap shows strong cross-like artifacts arounds bright sources, which prevents us from doing any reliable point-source photometry on bright sources. It was therefore decided to use observations of faint extragalactic sources to ensure point sources are not affected by this artifact. I chose three deep fields of galaxy clusters to ensure many faint sources are in the field.

Here are the OBSIDs of the observations :

– [1342189941, 1342189942] for PEP - GOODS-S scan and cross-scan observations at 70 and 160 microns.
– [1342189151, 1342189152] for the Herschel Lensing Survey (HLS) scan and cross-scan observations on Abell 383 at 100 and 160 microns.
– [1342188464, 1342188465] for LoCuSS scan and cross-scan observations of Abell 1914 at 100 and 160 microns.

2 The data reduction

The aim of this report is to compare the photometric quality of two mapmaking approaches. Maps are therefore created from the same calibrated level 1 data cube so that any differences in the final maps would originate from the mapmakers only . Each OBSID is reduced to level1, scan and cross-scan OBSIDs are concatenated, and maps are created with the two different mapmakers. The data reduction was carried out with HIPE 5.0 975.

2.1 Pre-processing

The script *toLevel1.py* (see section B.1 in the appendix) that takes the data from level0 to level1 contains all the steps of the standard pipeline, including the MMT deglitching of the timelines and the second order deglitching of the map index. In addition, I removed pixel-to-pixel offsets as well as the global drift of the whole focal plane as part of the pre-processing [1] necessary for the MADmap algorithm (baseline removal is required to mitigate correlated thermal noise in the focal plane).

I tried different flavors of the baseline removal preprocessing :

– **Default pre-processing** - The median signal of the whole focal plane, as a function of time, is fitted with a second order polynomial. This polynomial is then subtracted from each pixel timelines. This is done by the module *photGlobalDriftCorrection* which is distributed with the MADMap tar ball, thus the default pre-processing.
– **Segmented pre-processing** - In certain cases, especially right after a cryocooler recycling, the beginning of an observation may exhibit strong drifts. We therefore wrote a routine to segment an observation, and apply a linear fit to the individual segments. The length of the segments is always an integer number of scan legs so that the break between two consecutive segments, which would introduce discontinuities in the drift-corrected timelines, occurs outside of the map. This type of pre-processing is quite efficient at correcting the relatively steep signal drifts after cooler recycling.
– **Smooth pre-processing** - In the present case, assuming there is no extended emission in the chosen deep fields, any slow drift in the signal comes solely from the electronics. I used a BoxCarFilter of width 10000 to obtain a smoothed version of the median timeline, and I subtracted it from each pixel timelines. This approach is only possible because there is no extended emission in the fields considered for this analysis. The smooth pre-processing is also

1. Note that this pre-processing will not impair the highpass-filtering/Photproject reduction since the high-pass filtering is in fact a lot more aggressive, in terms of filtering spatial scales, than the global baseline removal.

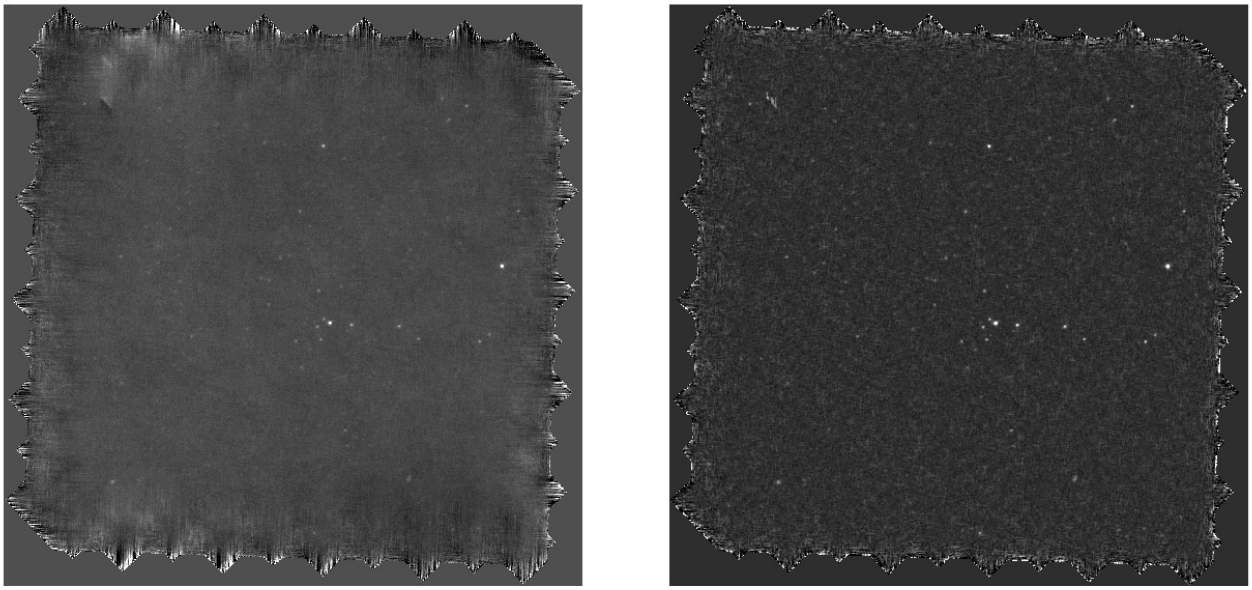

FIGURE 1 – Maps obtained in the green band (100 μm) from LoCuSS observations 0.6×0.6 deg^2 (P.I. G. Smith). MADmap and HPF/Photproject reductions are on the left and right respectively. The MADMap reduction was obtained with a smooth pre-processing and version 1 of the INVNTT files.

very close to the high-pass filtering, but with a quite larger filter width. It is certainly the most accurate baseline removal approach in this case.

2.2 MapMaking

For both MADMap and HPF/photproject approaches, the final map pixel size is set to be the native bolometer pixel size (3.2" and 6.4" for the blue and red channel, respectively).
In the case of Photproject, a first map is created from which a 3D mask is derived with thresholds of 0.008 and 0.0025 in the BL and R filters, respectively (see script *makeMaps.py* in section B.2 of the appendix). This mask is collapsed to a 2D map to check that point sources only have been properly thresholded, and that a reasonable fraction of the final map is masked for the subsequent highpass filtering. The *extendedMasking = True* keyword is used in the *photReadMaskFromImage* module. Filter widths of 20 and 30 frames are used for the BL and R datasets, respectively. As for MADmap, a maximum relative error of 10^{-5} is set as the convergence criterion, and the version 1 and 3 of the calibration files (InvNTT noise filters) were tested. Figure 1 presents typical maps obtained for the LOCUSS field in the 100 μm band.

It takes about 7 hours to run MADMap or HPF/photproject (including 2 passes for masking purposes + 2D mask creation) to create maps for the 3 fields in the two bands when allocating 64Gb of memory on our big compute machine *nhsccmp4*.

3 The photometry

I exported the generated maps, and carried out aperture photometry in IDL using the centroid (CNTRD.pro) to center the aperture, and APER.pro to measure the flux within 5" and 9" apertures in the Blue and Red channels, respectively. I therefore applied aperture corrections of 1.558, 1.680 and 1.524 in the Blue, Green and Red filters according to the information published on the HSC website in the release note [2].

Over the 3 fields, I measured a total of 37 sources in both bands, in both mapmaking styles with the 3 types of pre-processing mentioned in section 2.1. The results for a smooth pre-processing and version 1 of the INVNTT files are presented in figure 2 and in table 1 in the appendix. We find a tight correlation, over almost two orders of magnitudes, between the fluxes measured on MADmaped and Photprojected maps (left column on the figure). The figure also shows the histogram and statistics of the flux ratios between MADmap and Photproject fluxes by filter. The standard deviation of this ratio is $\sim$2% (5%) in the Blue (Red) channel. The median flux ratio in the Red filter is 1.07, which might indicate a slight systematic overestimation of the fluxes in MADmaps.

I have repeated this analysis for 3 different apertures (5"->10"->20" / 9"->15"->25" in the blue/red channel), and it shows that the correlation stays very tight. However larger apertures tends to increase the photometric uncertainty as more noisy pixels are added to the total flux. The final fluxes also tend to slightly increase with larger apertures, but this is possibly due to inappropriate aperture correction factors (the aperture correction I use was derived on a PSF scan at slow speed, i.e narrower core).

In addition, the measurement uncertainties (right column on figure 2) are systematically larger in the case of MADmap. This is primarily due to the INVNTT files correlation lengths being longer than the width of the highpass-filter, so that a significant fraction of the middle-frequency noise is left in the maps created with MADMap while it is rejected with the HPF/photproject combination. Highpass filter widths of 20 and 30 in the blue and red channels are indeed pretty aggressive settings. For the settings used in this analysis, MADmap uncertainties are typically about 20-40% higher than Photproject uncertainties.

Furthermore, I have checked that the photometry is not altered by the type of pre-processing applied to the data cube (see section 4.1 for details).

Lastly, I have compared the photometry for maps created from MADmap with the first and third generation of calibration files, or INVNTT files. Figure 3 shows that the two versions of the INVNTT files do not give consistent results. It turns out that the photometry from 3rd generation maps is systematically lower than 1st generation maps by 10-25%. This results was unexpected and is still under investigation.

2. http ://herschel.esac.esa.int/Docs/AOTsReleaseStatus/PACS_ScanMap_ReleaseNote_23Feb2010.pdf

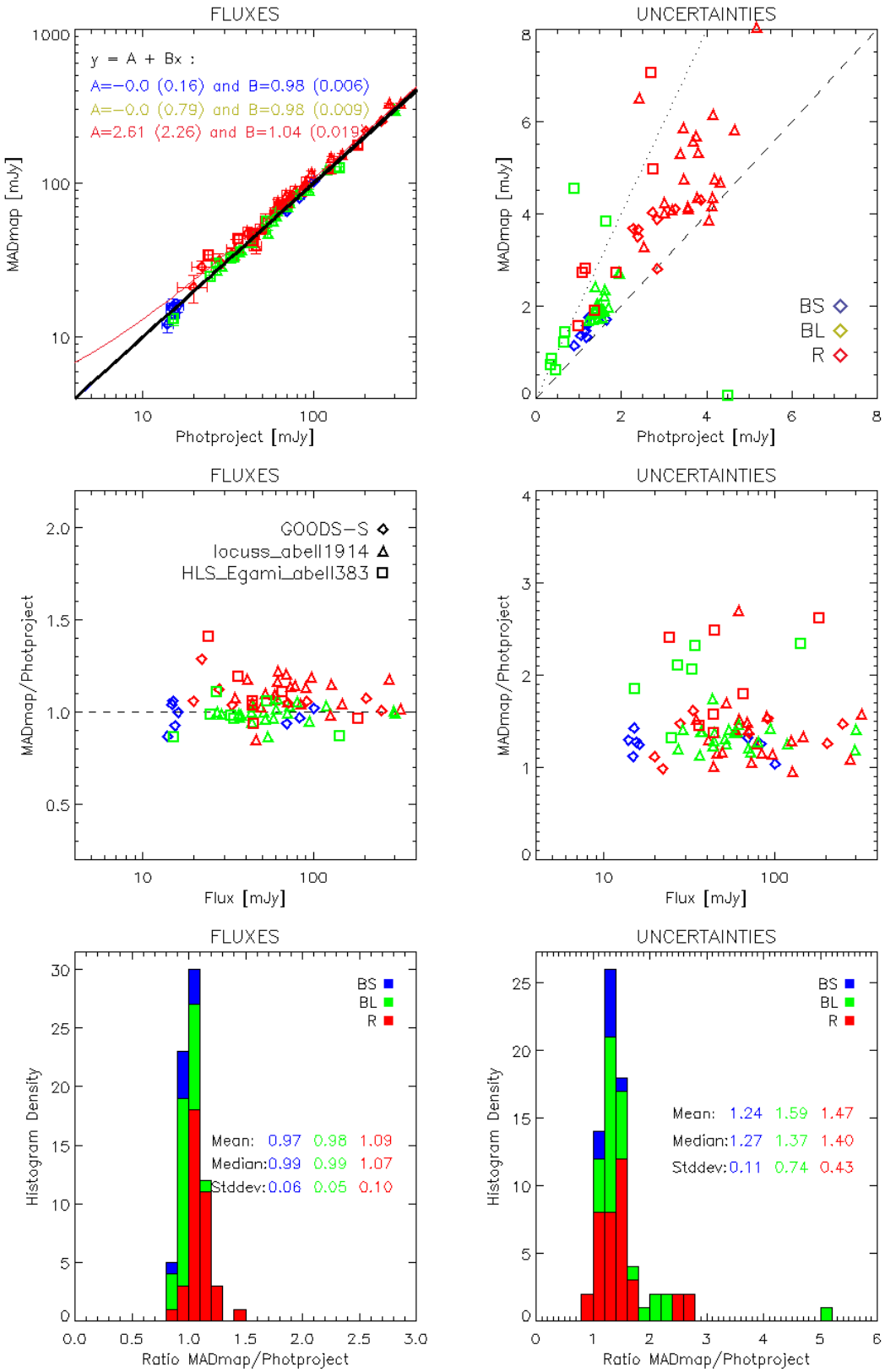

FIGURE 2 – Fluxes and corresponding uncertainties measured in the three fields for the two mapmaking approaches. The two lines on the left plot are linear fits to the data per filter, the fit parameters are shown on the graph. The two black lines on the right plot have slopes of 1 and 2 for comparison purposes. MADMap reductions were obtained with a smooth pre-processing and version 1 of the calibration files.

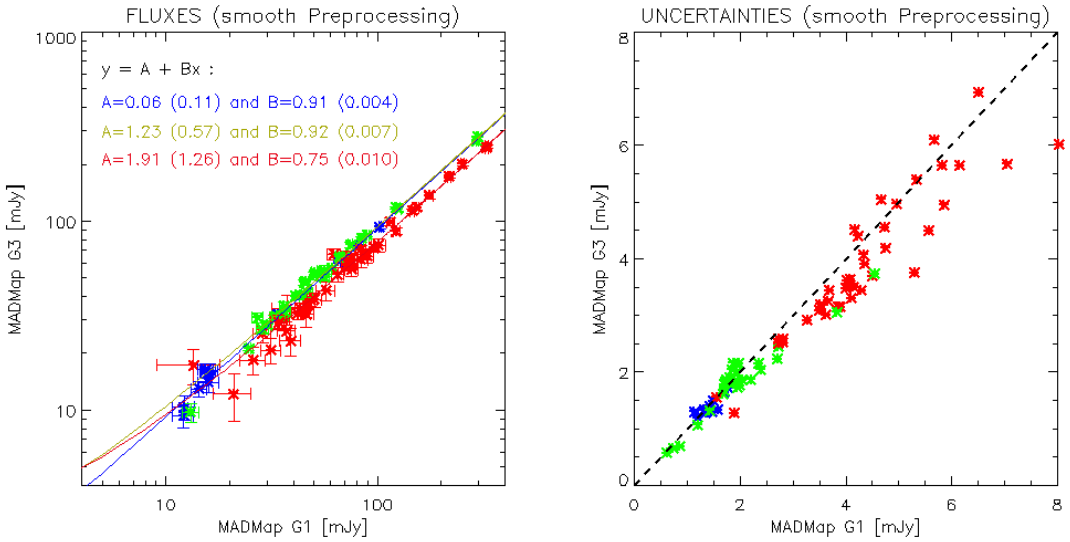

FIGURE 3 – Correlation between the photometry derived from maps obtained with MADMap calibration files version 1 and 3. The third generation of INVNTT files appears to change the photometry by 10-25%. Symbols and colors are coded as in figure 2.

4 Impact of pre-processing

4.1 On the photometry

The baseline removal in the data cube only affects large time scales, equivalently large spatial scale in the projected maps. So we expect the photometry of point sources, highest spatial frequencies in the image, to be unaffected by the different flavors of the pre-processing. Indeed, we derived maps for the three pre-processing methods enumerated in section 2.1, and we find that the photometry is consistent within 1% as shown in figure 4.

4.2 On the background

In general, the Photprojected maps exhibit a flat background, devoid of any large scale structures, as expected ; whereas MADmaps consistently present low-amplitude brightening/darkening at some somewhat medium spatial scales. Those darkening/brightening of the background are generally reduced when using the segmented or smoothed pre-processing compared to the default pre-processing (mostly because the modeled drifts get closer to the actual signal drifts). Figure 5 shows the evolution of the background for various pre-processing approaches. Segmenting per scan legs or subtracting a smoothed version of the median timeline improves considerably the flatness of the background. Now, figures 6 and 7 present histograms of the background levels extracted from various maps obtained with HPF/photproject and MADMap. The pixel-to-pixel dispersion is slightly larger for MADMap maps, which is expected considering the very tight filtering of the HPF/photproject approach. The smooth pre-processing gives better results than the default pre-processing. Note that the signal offset between the different

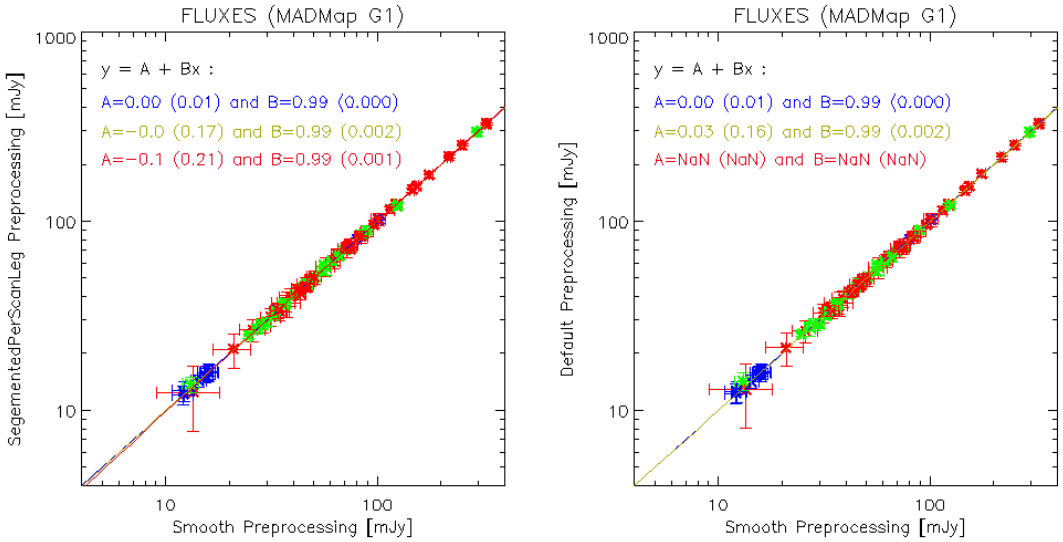

FIGURE 4 – Correlation between the photometry derived from maps obtained with MADMap and various pre-processing approaches. It appears that the choice of the pre-processing does not change the photometry. Symbols and colors are coded as in figure 2.

maps is a known feature; it means that the surface brightness of the maps is not absolutely calibrated.

5 Conclusions

The absolute photometric calibration of MADmap is consistent with the highpass-filtering/Photproject approach when using the first generation of calibration files. More work is required to fully validate/calibrate the third generation of calibration files (possibly using the naive map to calibrate the fluxes). The various pre-processing approaches tested in this report do not alter the photometry but give best satisfying results for the background when using a segmented or smooth pre-processing. Finally, MADmap overall point source photometric quality is noticeably worst than highpass-filtering/Photprojecting by ∼20-40% due to the stringent settings used in the HPF/photproject approach.

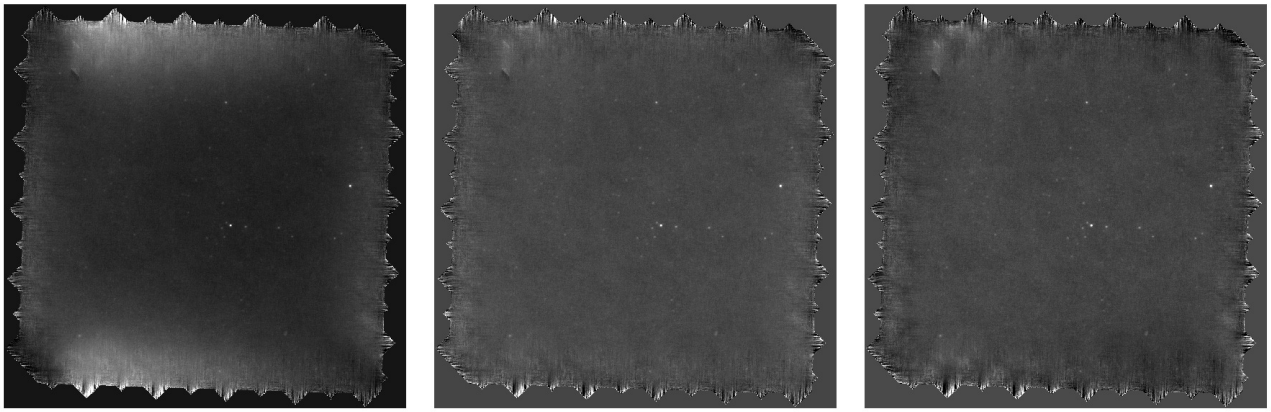

FIGURE 5 – Maps obtained at 100 μm from the Locuss program with MADmap using 3 different pre-processing approaches : default (left), segmented per scan legs (middle) and smooth pre-processing (right).

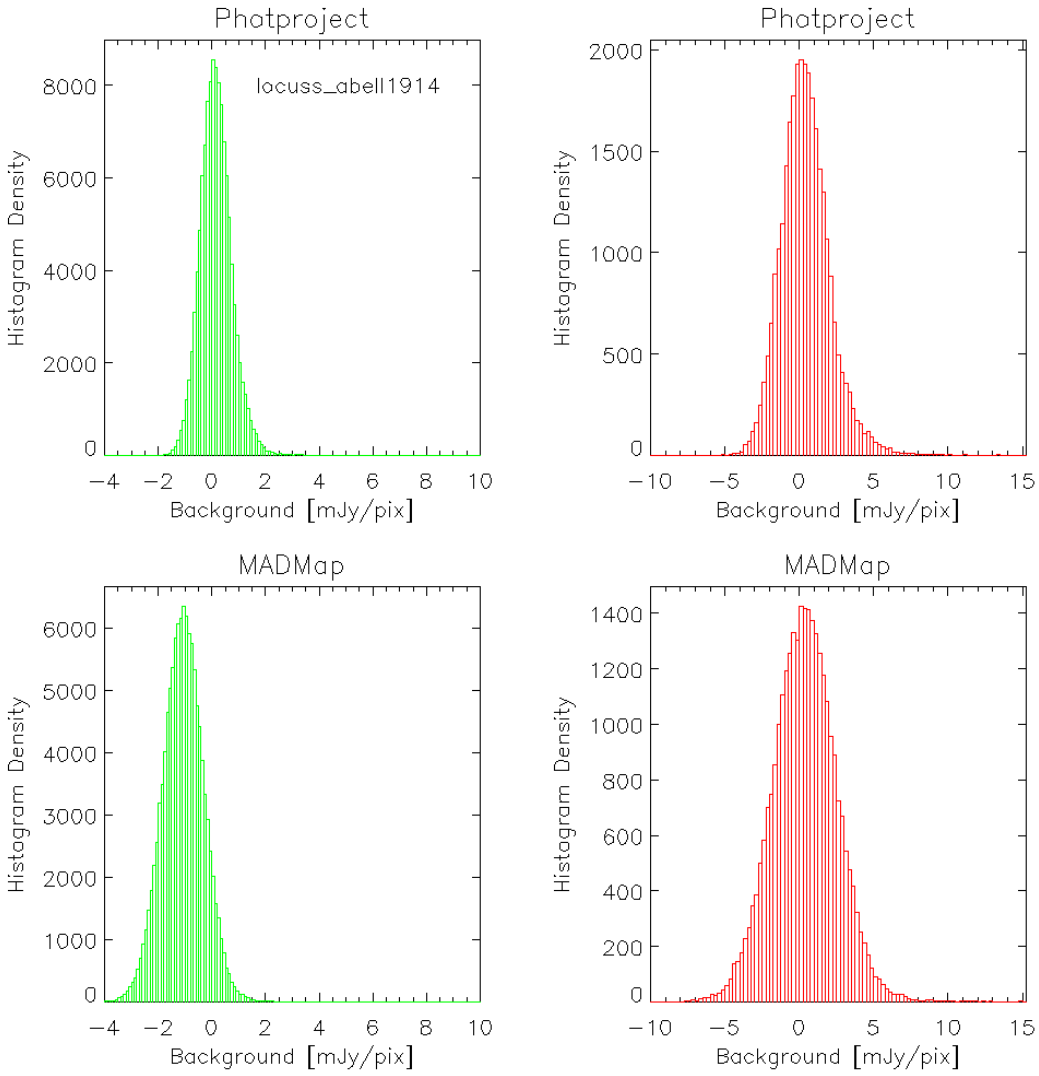

FIGURE 6 – Histograms of the background level for the LOCUSS field in the two bands (100 μm on the left and 160 μm on the right) for the HPF/photproject (top row) and MADMap (bottom row) mapmaking approaches. The MADmap backgrounds were obtained with the smoothed pre-processing and version 3 of the INVNTT files.

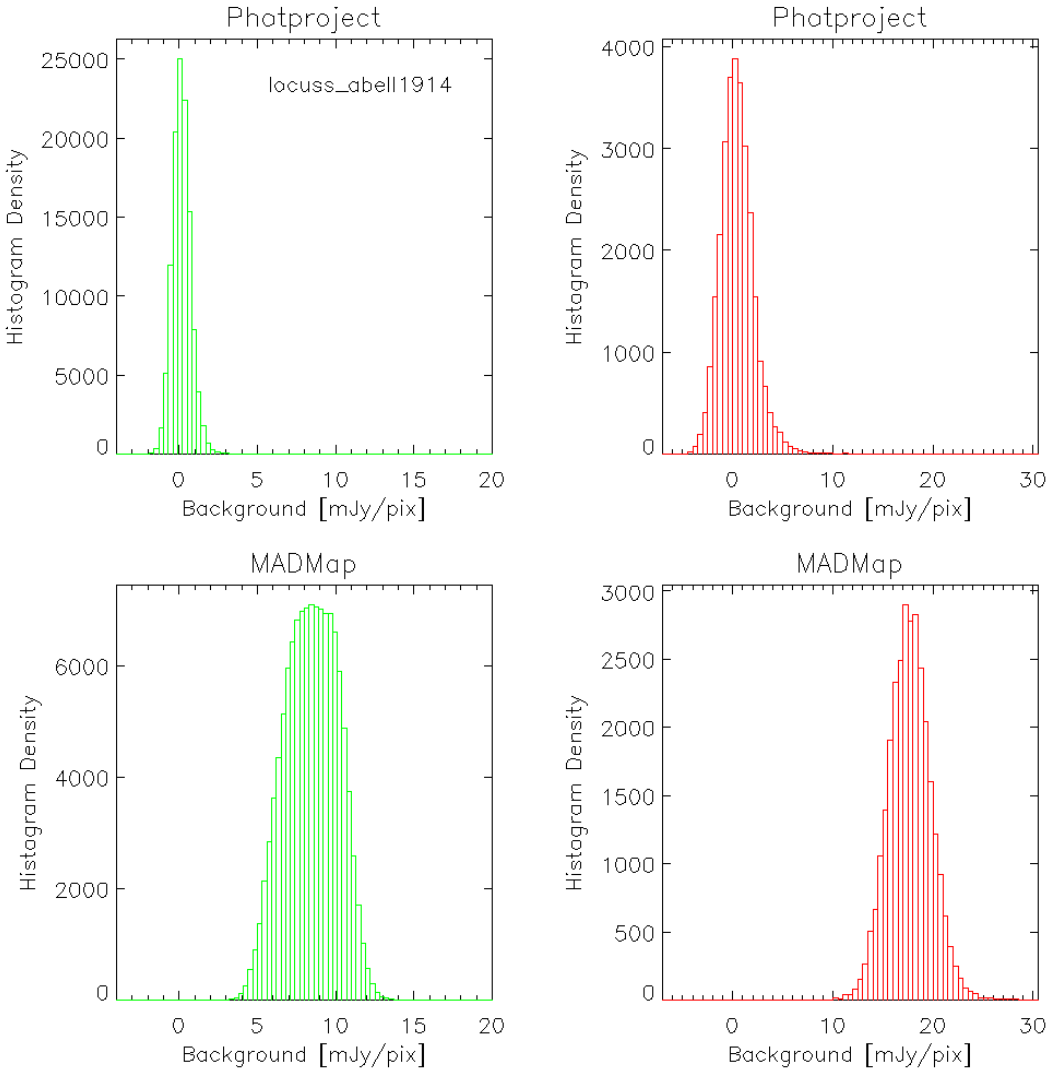

FIGURE 7 – As in figure 6, but with MADmap backgrounds obtained with the default pre-processing and version 3 of the INVNTT files.

A The Photometry

Table 1: Photproject, MADmap and Scanamorphos photometry comparison.

Source #	RA	Dec	Band	Photproject [mJy]	MADmap [mJy]	Scanamorphos [mJy]
Field : GOODS-S						
1	53.083	-27.690	BS	16.51 ± 1.26	14.11 ± 1.56	
			R	20.16 ± 3.80	12.63 ± 3.60	
2	53.146	-27.925	BS	69.01 ± 1.20	60.59 ± 1.30	
			R	249.8 ± 2.56	199.3 ± 3.77	
3	53.163	-27.899	BS	15.74 ± 0.89	13.10 ± 1.23	
			R	22.12 ± 2.78	24.11 ± 2.35	
4	53.185	-27.861	BS	82.86 ± 1.14	74.00 ± 1.41	
			R	91.47 ± 2.39	72.09 ± 3.27	
5	53.075	-27.850	BS	14.93 ± 1.20	15.95 ± 1.33	
			R	70.73 ± 2.83	65.88 ± 2.98	
6	53.117	-27.778	BS	16.19 ± 1.16	16.10 ± 1.32	
			R	33.37 ± 2.27	30.03 ± 3.28	
7	53.055	-27.711	BS	13.83 ± 1.16	9.593 ± 1.25	
			R	27.59 ± 2.45	20.19 ± 3.20	
8	53.125	-27.740	BS	100.7 ± 1.65	92.92 ± 1.62	
			R	203.7 ± 3.28	171.5 ± 3.27	
Field : locuss_abell1914						
1	216.70	37.8584	BL	44.56 ± 1.22	47.58 ± 1.82	
			R	59.05 ± 3.52	66.76 ± 4.65	
2	216.54	37.9564	BL	36.94 ± 1.38	31.59 ± 1.79	
			R	46.42 ± 3.75	22.98 ± 4.04	
3	216.48	37.9536	BL	118.5 ± 1.46	118.7 ± 1.98	
			R	146.6 ± 3.03	118.3 ± 3.56	
4	216.50	37.8971	BL	51.00 ± 1.30	53.63 ± 1.77	
			R	34.70 ± 3.74	35.61 ± 6.15	
5	216.65	37.9856	BL	26.89 ± 1.43	32.00 ± 1.83	
			R	51.90 ± 3.48	43.40 ± 5.18	
6	216.74	37.9894	BL	44.03 ± 1.32	42.70 ± 2.08	
			R	60.81 ± 3.87	56.98 ± 5.04	
7	216.53	37.8176	BL	36.60 ± 1.56	36.22 ± 1.92	
			R	43.29 ± 3.92	32.71 ± 4.48	
8	216.48	37.8294	BL	52.22 ± 1.50	51.65 ± 2.12	
			R	84.10 ± 3.49	67.73 ± 3.54	
9	216.45	37.9427	BL	28.84 ± 1.29	26.48 ± 1.92	
			R	41.21 ± 2.49	41.28 ± 3.34	
10	216.32	37.9875	BL	70.11 ± 1.59	65.35 ± 1.82	
			R	77.88 ± 4.65	64.81 ± 5.53	
11	216.28	37.8504	BL	297.9 ± 1.67	277.2 ± 1.77	
			R	324.8 ± 3.49	243.1 ± 3.82	
12	216.26	37.8976	BL	63.55 ± 2.18	53.26 ± 2.24	
			R	99.42 ± 5.44	76.36 ± 5.95	
13	216.39	37.7989	BL	79.88 ± 1.53	80.77 ± 1.73	
			R	125.2 ± 3.18	87.51 ± 3.72	
14	216.37	37.7878	BL	60.34 ± 1.60	57.22 ± 1.89	
			R	96.94 ± 4.14	98.57 ± 4.26	
15	216.31	37.7856	BL	72.05 ± 1.69	73.08 ± 1.91	
			R	73.39 ± 4.10	70.82 ± 4.04	
16	216.44	37.8000	BL	94.21 ± 1.40	84.61 ± 1.80	
			R	127.9 ± 3.96	112.8 ± 3.13	
17	216.47	37.8018	BL	295.8 ± 1.58	264.8 ± 1.90	
			R	277.5 ± 4.33	238.5 ± 5.08	
18	216.51	37.7862	BL	44.09 ± 1.34	46.83 ± 1.63	

Source #	RA	Dec	Band	Photproject [mJy]	MADmap [mJy]	Scanamorphos [mJy]
			R	59.55 ± 3.00	53.68 ± 4.36	
19	216.45	37.7094	BL	43.49 ± 1.53	39.34 ± 1.86	
			R	68.34 ± 4.24	59.74 ± 5.44	
20	216.49	37.6216	BL	42.37 ± 1.99	41.08 ± 2.48	
			R	62.54 ± 2.52	61.41 ± 7.06	
21	216.62	37.7901	BL	37.55 ± 1.40	32.05 ± 2.00	
			R	50.55 ± 3.50	38.29 ± 3.33	
22	216.70	37.6655	BL	61.58 ± 1.60	63.16 ± 2.33	
			R	69.04 ± 4.16	61.14 ± 5.29	
Field : HLS_abell383						
1	41.998	-3.5421	BL	31.33 ± 0.39	30.90 ± 0.69	
			R	51.49 ± 1.01	31.73 ± 1.52	
2	42.010	-3.5475	BL	30.58 ± 0.48	30.45 ± 0.57	
			R	41.04 ± 1.50	33.19 ± 1.66	
3	42.031	-3.5873	BL	41.83 ± 1.01	51.20 ± 2.38	
			R	71.94 ± 2.68	59.71 ± 5.89	
4	42.020	-3.5713	BL	19.96 ± 1.28	28.16 ± 1.06	
			R	39.31 ± 2.15	31.34 ± 3.04	
5	42.005	-3.5318	BL	23.84 ± 0.59	21.56 ± 0.56	
			R	42.60 ± 1.49	33.58 ± 1.70	
6	42.050	-3.4908	BL	137.5 ± 1.22	110.4 ± 3.04	
			R	169.5 ± 5.05	131.3 ± 6.64	
7	41.976	-3.5398	BL	9.992 ± 0.86	11.80 ± 1.17	
			R	25.58 ± 1.37	28.58 ± 2.90	

B Scripts

B.1 toLevel1.py

```
## This script processes the data up to level-1, that is right before the mapmaking proc
# We reduce GOODS-S data and build maps with Photproject and MADmap to compare the photc
# GOODS-S was chosen because sources are faint, hence no point source artefact with MADm
#         but enough sources to do many aperture photometry to have statistics.


from java.util import Date
print ""
print "Start Processing at :" , Date()
print ""


from herschel.ia.numeric.toolbox.basic import Sigclip
from herschel.pacs.spg.phot import IIndLevelDeglitchTask

verbose = 0
preprocess=0
secondOrderDeglitch=0

execfile("photMarkSegments.py")
#execfile("PhotGlobalDriftCorrectionTask.py")
execfile("photMarkScanLegs.py")
execfile("my_globalDriftCor.py")

fa=FitsArchive()

calTree=getCalTree()

field = ['GOODS-S', 'locuss_abell1914', 'HLS_Egami_abell383']
field = ['GOODS-S']
nField = len(field)

for iField in range(nField):
    print ''
    print "Reducing Field: "+ field[iField]
    if preprocess == 1:
        outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iField]+'
    if preprocess == 2:
        outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iField]+'
    elif preprocess == 3:
        outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iField]+'
```

```
else:
    outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iField]+'
datadir="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/pool/"

if field[iField] == 'GOODS-S':
    # data from GOODS   (BL/R)
    obsids=[1342189939,1342189940]  # this one has a dark trail in it....!
    obsids=[1342189941,1342189942]
elif field[iField] == 'HLS_Egami_abell383':
    # data from E. Egami program: Abell 383 on OD 242   (BL/R)
    obsids=[1342189151,1342189152]
elif field[iField] == 'locuss_abell1914':
    # data from Locuss program: Abell 1914  (BL/R)
    obsids=[1342188464,1342188465]
else:
    obsids=0

outprefix = ['scan','xscan']

skip=1000

channels = ["red", "blue"]
channels = ["blue"]
for iCh in range(len(channels)):
    channel=channels[iCh]
    print '        Reducing '+channel+' channel'

    # Processing starts here.

    first=True
    i=0
    for iobsid in obsids:
       print '               Reducing '+outprefix[i]
       obs = getObservation(iobsid,poolLocation=datadir)
       pp = obs.auxiliary.pointing
       if channels[iCh]=='blue':
          tmp_frames=obs.level0.refs["HPPAVGB"].product.refs[0].product
          # fix the 'Status' in case it is erroneous
          wpr = tmp_frames.getStatus("WPR")
          bands = tmp_frames.getStatus("BAND")
          idGreen = wpr.where(wpr == 0)
          idBlue = wpr.where(wpr == 1)
          if idGreen.length()>0:
            if bands[idGreen][0]=='BS':
               print ' WARNING for Green filter : WPR =0 was erroneously assigned BS
```

```
            bands[idGreen] = String('BL')
            band = 'BL'
        if idBlue.length() > 0:
          if bands[idBlue][0] == 'BL':
            print ' WARNING for Blue filter : WPR =1 was erroneously assigned BL
            bands[idBlue] = String('BS')
            band = 'BS'
        tmp_frames.setStatus("BAND",bands)
        #
        tmp_frames = obs.level0.refs["HPPAVGB"].product.refs[0].product
        band='BL'
      elif channels[iCh]=='red':
        tmp_frames=obs.level0.refs["HPPAVGR"].product.refs[0].product    #  to get
        band='R'
      oep = obs.auxiliary.orbitEphemeris
      horizons = None
      isSso = False
      if (obs.meta.containsKey("naifid")) :
        if (obs.meta["naifid"].value != 0):
          isSso  = True
      timeCorr = obs.auxiliary.timeCorrelation
      photHK=obs.level0.refs["HPPHK"].product.refs[0].product["HPPHKS"]
      tmp_frames = findBlocks(tmp_frames, calTree=calTree)
      tmp_frames = detectCalibrationBlock(tmp_frames)
      tmp_frames = removeCalBlocks(tmp_frames)
      tmp_frames = tmp_frames.select(Int1d.range(tmp_frames.signal.dimensions[2]-1-
      tmp_frames = photFlagBadPixels(tmp_frames, calTree=calTree)
      if (channel == 'blue'):
        blue_badpix=calTree.photometer.badPixelMask.blue
        blue_badpix[2,30]=1
        tmp_frames.setMask("BADPIXELS",blue_badpix)
#     tmp_frames = photFlagSaturation(tmp_frames, calTree=calTree, hkdata=photHK)
      tmp_frames = photConvDigit2Volts(tmp_frames, calTree=calTree)
      tmp_frames = convertChopper2Angle(tmp_frames, calTree=calTree)
      tmp_frames = photMMTDeglitching(tmp_frames, incr_fact=2,mmt_mode='multiply',
      if verbose:
          MMTMask = tmp_frames.getMask('MMT_Glitchmask')
          print 'MMT deglitching has flagged '+ str(MMTMask.where(MMTMask==True).le
      tmp_frames = photRespFlatfieldCorrection(tmp_frames, calTree = calTree)
      # frames = photDriftCorrection(frames, calTree=calTree)
      tmp_frames = photAddInstantPointing(tmp_frames,pp,orbitEphem = oep)
      tmp_frames = photAssignRaDec(tmp_frames, calTree=calTree)
      # Subtract pixel-to-pixel offsets
      if preprocess == 1:
          tmp_frames=photOffsetCorr(tmp_frames)
```

```
    # Define segments to apply segmented fit to the obsid
    segments = photMarkSegments(tmp_frames,binsize=300000)  # per entire numb
    #segments = photMarkScanLegs(tmp_frames)   # per scan leg
    nSeg = int( MAX( segments ) )
    if verbose:
        print "The data stream is segmented into "+str(nSeg)+" pieces"
    for iblk in range(1,nSeg+1):
       q = segments==iblk
       sel = q.where(q)
       new_frames = tmp_frames.select(sel)
       my_binsize=1000
       # make sure there are enough points in the segment to fit the model
       if sel.length() > my_binsize:
           photModuleDriftCorrection(new_frames,doPlot=0,q1=-100., q2=100., \
                                     datadir=outdir,outprefix=outprefix[i]+'segme
           photGlobalDriftCorrection(new_frames,model=1,verbose=verbose,doPlo
                                     datadir=outdir,outprefix=outprefix[i]+'segme
       tmp_frames["Signal"].data[:,:,sel.toInt1d()[0]:sel.toInt1d()[-1]+1] =
       del(new_frames)
       #
if preprocess == 2:
    tmp_frames=photOffsetCorr(tmp_frames)
    my_binsize=1000
    #photModuleDriftCorrection(tmp_frames,doPlot=0,q1=-100., q2=100., \
    #                           datadir=outdir,outprefix=outprefix[i]+'segm
    photGlobalDriftCorrection(tmp_frames,model=1,verbose=verbose,doPlot=0,bin
                              datadir=outdir,outprefix=outprefix[i],order=
if preprocess == 3:
    tmp_frames=photOffsetCorr(tmp_frames)
    binsize = 10000
    tmp_frames = my_globalDriftCor(tmp_frames, binsize)
System.gc()
if secondOrderDeglitch:
   photProject(tmp_frames, deglitch = True, slimindex=True)
   index = photProject.getValue("index")
   deg = IIndLevelDeglitchTask()
   s = Sigclip(10,7, outliers = "both")
   img = deg(index, tmp_frames, mask = True, map = False, algo = s)
   del(index, img)
if verbose:
    IIndMask = tmp_frames.getMask('2nd level glitchmask')
    print 'IInd level deglitching has flagged '+ str(IIndMask.where(IIndMask=
fa.save(outdir+"Frames_Level1_"+iobsid.toString()+"_"+channel+".fits",tmp_fra
if first:
   frames=tmp_frames.copy()
```

```
            first=False
         else:
            frames.join(tmp_frames)
         i=i+1
         System.gc()
      del(tmp_frames)
      print "done with pre processing"

      fa.save(outdir+'frames_'+band+'_'+(obsids[0]).toString()+'+1_preMapMaking.fits',

print ""
print "End processing at :" , Date()
print ""
```

B.2 makeMaps.py

```
# Build the maps from the same level-1 data cube
# MADmap first
# then Photproject

from java.util import Date
print ""
print "Start Processing at :" , Date()
print ""


fa=FitsArchive()
fa_SR = FitsArchive(reader=FitsArchive.STANDARD_READER)

calTree=getCalTree()

field = ['GOODS-S', 'locuss_abell1914', 'HLS_Egami_abell383']
field = ['GOODS-S']
nField = len(field)

invntt_version = 'old'
#invntt_version = 'old'from herschel.spire.ia.pipeline.scripts.POF5.POF5_tasks import *
madmap_process = 0
photproj_process = 1
mask_threshold = 1

for iField in range(nField):
    print ''
```

```
print 'Processing field: '+ field[iField]
if field[iField] == 'GOODS-S':
    # data from GOODS   (BS/R)
    obsids=[1342189939,1342189940]  # this one has a dark trail in it....!
    obsids=[1342189941,1342189942]
    bandBlue = 'BS'
    rastar_deg = Double1d([53.085035, 53.102421, 53.041072, 53.083373, 53.188267, 53
    decstar_deg = Double1d([-27.8739, -27.91286, -27.744713, -27.689888, -27.9108, -
elif field[iField] == 'HLS_Egami_abell383':
    # data from E. Egami program: Abell 383 on OD 242   (BL/R)
    obsids=[1342189151,1342189152]
    bandBlue = 'BL'
    rastar_deg = Double1d([41.998211, 42.010878, 42.032196, 42.020764, 42.005317, 42
    decstar_deg = Double1d([-3.5421264, -3.5473686, -3.5877629, -3.5717286, -3.53195
elif field[iField] == 'locuss_abell1914':
    # data from Locuss program: Abell 1914  (BL/R)
    obsids=[1342188464,1342188465]
    bandBlue = 'BL'
    rastar_deg = Double1d([216.70035, 216.54417, 216.48013, 216.50575, 216.65858, 21
    decstar_deg = Double1d([37.858461, 37.956703, 37.953817, 37.897559, 37.985951, 3
else:
    obsids=0
#
channels = ["red", "blue"]
channels = ["blue"]
for iCh in range(len(channels)):
    channel=channels[iCh]
    if channel == "red":
        band='R'
        outpixsz=6.4
        hpfwidth=30 # high pass filtering width
        #hpfwidth=50
        threshold = 0.0025
        distFilter = 35.0
    else:
        band=bandBlue
        outpixsz=3.2
        hpfwidth=20 # high pass filtering width for green filter in non-GOODS-S data
        #hpfwidth=30 # high pass filtering width for blue filter in GOODS-S data
        #hpfwidth=40
        threshold = 0.0008
        distFilter = 30.0
    #
    # The drift correction is now applied per obsid
    #frames=photOffsetCorr(frames)
```

```
        #photGlobalDriftCorrection(frames,model=1,verbose=verbose,doPlot=True,datadir=ou
        #
        if madmap_process:
            #outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFie
            outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFiel
            #outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFie
            fileframe = outdir+'frames_'+band+'_'+(obsids[0]).toString()+'+1_preMapMakin
            frames = fa.load(fileframe)
            print '    Restoring: '+fileframe
            print '    Processing channel: '+channel

            print '        Processing MADMap'
            frames.setStatus("OnTarget",Bool1d(frames.signal.dimensions[2],False))
            otf=frames.getStatus("OnTarget")
            bbid=frames.getStatus("BBID")
            q=(bbid==215131301)
            sel=q.where(q)
            otf[sel]=True
            frames.setStatus("OnTarget",otf)

            scale = 1.0
            crota2 = 0.0
            maxRelError   = 1.e-6
            maxIterations = 500

            tod = makeTodArray(frames,scale,crota2)

            naivemap = runMadMap(tod,calTree,maxRelError,maxIterations,True)
            fa.save(outdir+"naivemap_"+band+".fits",naivemap)

#            # use old Red invntt
#            print "MADmap with old invntt"
#            if channel=="red":
#               cprod = getCalProduct("Photometer","InvnttRed",1)
#            else:
#               if band=="BS":
#                  cprod = getCalProduct("Photometer","InvnttBS",1)
#               else:
#                  cprod = getCalProduct("Photometer","InvnttBL",1)
#            ct = getCalTree()
#            ct = setInvntt(cprod["Contents"].data,ct,band=band)

            if invntt_version == 'old':
                print "MADmap with old invntt"
                if channel=="red":
```

```
            cprod = getCalProduct("Photometer","InvnttRed",1)
          else:
            if band=="BS":
              cprod = getCalProduct("Photometer","InvnttBS",1)
            else:
              cprod = getCalProduct("Photometer","InvnttBL",1)
          #calTree = getCalTree()
          ct = setInvntt(cprod["Contents"].data,calTree,band=band)
          madmap_outfile = outdir+"madmap_"+band+"_G1invntt.fits"
      else:
          print "MADmap with new invntt"
          if channel == 'red':
              if invntt_version == 'v3_50':
                  invntt_filename = '/home/nbillot/iascripts/my_CalFiles/INVNTT/in
                  madmap_outfile = outdir+"madmap_"+band+"_G3invntt50.fits"
              else:
                  invntt_filename = '/home/nbillot/iascripts/my_CalFiles/INVNTT/in
                  madmap_outfile = outdir+"madmap_"+band+"_G3invntt2000.fits"
          else:
              invntt_filename = '/home/nbillot/iascripts/my_CalFiles/INVNTT/invntt
              madmap_outfile = outdir+"madmap_"+band+"_G3invntt100.fits"
          print 'Presumably using '+ invntt_filename
          rawdat = Double2d(fa_SR.load(invntt_filename)["PrimaryImage"].data)
          #calTree = getCalTree()
          ct=setInvntt(rawdat,calTree,band=band)
      # dummy MADmap run -- buggy first try
      dummy = runMadMap(tod,ct,1.e-2,maxIterations,False)
      #fa.save(outdir+"madmap_dummy_"+band+"_G1invntt.fits",dummy)
      madmap=runMadMap(tod,ct,maxRelError,maxIterations,False)
      #
      Display(madmap, title='MADMap, '+field[iField]+', '+channel)
      fa.save(madmap_outfile,madmap)
      del(tod,naivemap,madmap)
#
############################################################
# Photproject
# import for masked high-pass
#from herschel.pacs.spg import PhotReadMaskFromImageTask
#
if photproj_process:
      outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFiel
      #outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFie
      #outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFie
      #outdir ="/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field[iFie
      fileframe = outdir+'frames_'+band+'_'+(obsids[0]).toString()+'+1_preMapMakin
```

```
frames = fa.load(fileframe)
print '    Restoring: '+fileframe
print '    Processing channel: '+channel

print '        Processing Photproject'
if mask_threshold:
    # Need first pass to make the map from which the mask will be derived
    #frames = fa.load(fileframe)
    frames = highpassFilter(frames,hpfwidth)
    frames = frames.select(frames.getStatus("BBID") == 2151313011)
    map1 = photProject(frames, calTree=calTree,calibration=True,outputPixels
    #
    #Display(map1)
    fa.save(outdir+"map1_"+band+".fits",map1)
    #
    # Second pass to apply the highpass filter while providing a mask
    #threshold=STDDEV(map1.image[map1.image.where(ABS(map1.image) > 1e-6)])
    print '          Threshold : '+str(threshold)
    maskMap = map1
    frames = fa.load(fileframe)
    frames  = photReadMaskFromImage(frames, maskMap, extendedMasking=True,ma
    frames      = highpassFilter(frames,hpfwidth,maskname="HighpassMask")
    frames = frames.select(frames.getStatus("BBID") == 2151313011)
    map2        = photProject(frames, calibration=True,outputPixelsize=outpi
    #
    Display(map2, title='PhotProject + HPF, '+field[iField]+', '+channel)
    fa.save(outdir+"map2_"+band+".fits",map2)
    del(map1,map2)
    #
    # Check the 2D mask
    #temp = frames.copy()
    #temp["Signal"].data = Double3d(temp.mask["HighpassMask"].data)
    #mask2d        = photProject(temp, calibration=True,outputPixelsize=outp
    #del(temp)
    #Display(mask2d, title='HPF Mask, '+field[iField]+', '+channel)
    #fa.save(outdir+"mask2D_"+band+".fits",mask2d)
    #del(maskMap)
else:
    # Now define the ignore mask and fill it appropriately
    # The sense of the mask is
    #  False = no star
    #  True = Yes, star present
    #
    mask_restore = 0
    if mask_restore == 0:
```

```
    newmask = Bool3d(frames.dimensions[0],frames.dimensions[1],frames.di
    frames.addMaskType('STAR',"Mask bright stars")
    #
    deg2rad = Math.PI / 180.
    rastar = rastar_deg * deg2rad
    decstar = decstar_deg * deg2rad
    ra=frames["Ra"].data * deg2rad
    dec=frames["Dec"].data * deg2rad
    for istar in range(len(rastar)):
       print '     Masking star '+str(istar)+' out of '+str(len(rastar))
       #print istar, rastar[istar], decstar[istar]
       inside = COS(ra -rastar[istar]) * COS(dec) * COS(decstar[istar])
       inside = ARCCOS(inside) / deg2rad * 3600
       #inside2=inside.copy()
       #sel = inside2.where(inside2<distFilter)
       sel = inside.where(inside<distFilter)
       while sel.hasNext():
          newmask.setAt(sel.next(), True)
    #save("/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+field
    save(outdir+"map3_mask3D_"+band+".fits","newmask")
else:
    #newmask = fa.load(outdir+"map3_mask3D_"+band+".fits")
    restore("/herscheldata/pacs/nbillot/FluxCal/MADmap_Photproject/"+fie
    frames.addMaskType('STAR',"Mask bright stars")
#
# Now set the mask and delete temporary variable to release memory.
frames.setMask('STAR',newmask)
del(newmask)
System.gc()
# high-pass filter
frames = highpassFilter(frames,hpfwidth,maskname='STAR',interpolateMaske
#
# Check the 2D mask
temp = frames.copy()
temp["Signal"].data = Double3d(temp.mask["STAR"].data)
mask2d        = photProject(temp, calibration=True,outputPixelsize=outpi
del(temp)
Display(mask2d, title='HPF Mask, '+field[iField]+', '+channel)
fa.save(outdir+"map3_mask2D_"+band+".fits",mask2d)
del(mask2d)
#
# Redefine the STAR mask
# This will ensure that photProject will not interpret it as a bad pixel
newmask = Bool3d(frames.dimensions[0],frames.dimensions[1],frames.dimens
frames.setMask('STAR',newmask)
```

```
                del(newmask)
                #
                frames = frames.select(frames.getStatus("BBID") == 2151313011)
                map3 = photProject(frames, calibration=True,outputPixelsize=outpixsz,cal
                #
                Display(map3, title='PhotProject + HPF + maskPerPatch, '+field[iField]+'
                fa.save(outdir+"map3_"+band+".fits",map3)
                del(map3)
print ""
print "End processing at :" , Date()
print ""
```