The GASKAP-HI Imaging Pipeline

Nickolas Pingel Yik Ki (Jackie) Ma

October 22, 2024

Abstract

This document describes how to use the GASKAP Imaging pipeline, from acquiring the calibrated visibilities, pre-processing, imaging, and the combination with single dish data to correct for missing short-spacings.

1 Introduction

This pipeline utilizes a combination of `WSClean`, the Common Astronomy Software Applications (`CASA`), `miriad`, and `python 3` to create deconvolved images from the Australian Square Kilometre Array Pathfinder telescope (ASKAP) for the neutral hydrogen HI Galactic ASKAP (GASKAP-HI) survey. These scripts essentially form a wrapper to process GASKAP-HI data efficiently distributed across a computing cluster. This pipeline further utilizes the joint deconvolution capabilities of `WSClean` to ensure the diffuse emission that extends across the boundaries of the formed ASKAP primary beams is accurately recovered. This document provides all of the necessary commands and steps to

- clone the pipeline repository;
- obtain the calibrated measurement sets for all 108 that make up a typical GASKAP-HI footprint;
- pre-process the measurement sets for imaging through optional binning and applying a continuum subtraction;
- imaging with the joint deconvolution capabilities of `WSClean`;
- combination of ASKAP and Parkes data to fill in missing short-spacings;
- and usage examples for utility scripts.

As stated above, this pipeline relies on several public third-party radio astronomy software packages. These packages are all available on the RSAA servers. Add the following lines to your `.bashrc` file:

```
## WSClean, chgcentre
export PATH=/home/nipingel/software/bin:$PATH

## miriad
source /pkg/linux/miriad/MIRRC.sh
PATH="${MIRBIN}:${PATH}"

## CASA 5.4
export PATH=/pkg/linux/casa-release-5.4.1-32.el7/bin:$PATH

## CASAcore
export PYTHONPATH=/home/nipingel/software/python-casacore-master/:$PYTHONPATH
```

It is assumed that one also uses an Anaconda distribution of python 3 and has `numpy` and `astropy` modules available. See the Linux installation instructions towards the bottom right of this page:https://www.anaconda.com/products/individual-d.

1.1 Cloning Pipeline Repository

The GASKAP-HI Imaging pipeline is hosted on GitHub: https://github.com/nipingel/GASKAP_Imaging. The source code can be obtained by cloning the repository:

```
-> git clone https://github.com/nipingel/GASKAP_Imaging.git
```

Due to the large number of path variables in the scripts that point to various aspects of the data (e.g., measurement sets, primary beam models), there is no make file or configuration to set up the scripts. The user is responsible for making these manual edits. The following sections discuss the exact variables to change to ensure the scripts know where the data are located.

2 Prepping the Measurement Sets for Imaging

2.1 Obtaining the Calibrated Measurement Sets

One must first obtain an OPAL account to access the calibrated visibilities. OPAL is a web-based application that is used to prepare and submit telescope applications to the Australia Telescope National Facility (ATNF). See Section 2 of the OPAL Users guide to register for an account: https://www.atnf.csiro.au/observers/docs/opal/guide.html.

Once your OPAL account is active, sign on the search form webpage of the CSIRO ASKAP Science Data Archive (CASDA): https://data.csiro.au/collections/domain/casdaObservation/search/. One generally needs to only enter the scheduling block ID (SBID) of interest (e.g., 14158) into the field labeled 'Observation scheduling block ID', check the 'unreleased' box, and hit the search button. When the data are located, select the tab labeled 'Visibilities' and select all available data. There should be 216 available for download. These consist of the 108 (36 beams×3 interleaves) and the equivalent 1 MHz-averaged measurement sets used to construct the continuum image. These are generally not used. However, they are much smaller than the measurement sets at full resolution and do not slow down the download rates too much. Once the data are selected, select the 'Download' button in the top right of the table and select the 'Download from public site' option from the drop-down menu. You will get an email with a link to download the data when it is ready, which may take a while.

We also note here that for 10-hour observation, the file size is about 5-6 TB, while CASDA in general seems to limit to 500 GB per download. You can request that your quota be increased such that the data can be downloaded all in one go.

When the data are ready for download, click the link in the email select 'Save links as a text file' link on the bottom right of the page. This should generate a file called `links.txt`. Transfer this file to the area on the Stromlo servers where you wish to download the data and begin the download with the command

```
cat links.txt | parallel -j 5 wget -c --content-disposition '{}'
```

You could parallelize the process by hacking the `links.txt` file by first removing all downloads to the averaged data (they are not needed) and spreading out the downloads between interleaves A, B, and C. That is, use your favorite text editor to copy/paste the links into three separate files: `links_A.txt`, `links_B.txt`, and `links_C.txt`. You can then run the above command in three separate terminal tabs. If downloading to your `/avatar/` area, open three separate interactive jobs on small memory nodes so as to not strain the head node:

```
qsub -I -X -l select=1:ncpus=1 -q smallmem
```

This download will take up to a full day. Another way to parallelize the download is the command

```
cat links.txt | parallel -j 6 wget -c --content-disposition '{}'
```

which will download 6 measurement sets at a time.

2.2 File Verification

A checksum file is an alphanumeric value that uniquely represents the contents of a file. You will see that each tarball measurement sets has an associated checksum file. You can use the new utility script, `checksum.py`, to ensure the tarballs are uncorrupted after being downloaded. First, copy the script over to your working directory:

```
cp /avatar/ykma/gaskap/38791/checksum.py ./
```

Change the settings within the script accordingly, and then execute it to compare the actual checksum with the expected value:

```
python checksum.py
```

The script will report any files with mismatching checksum, and write the download link to a new file `links_reDL.txt`.

2.3 Untar

We can untar them with the command

```
for tarFile in *.tar; do tar xvf $tarFile; done
```

Again, this will take some time. Perhaps create temporary directories, move the 36 measurement sets for an interleave into them, and run the tar command in the three separate terminal tabs to speed things up. Once finished, delete the tarballs using

```
rm *.tar
```

Be careful using that * symbol!

2.4 Phase Rotating to a Common Phase Centre

The PAF imaging mode of `WSClean` requires all input MSs be phase rotated to a common field phase centre. Fortunately, this is easily accomplished by using the packaged `chgcentre` task. The syntax is simple:

```
-> chgcentre name_of_MS_file.ms 00h00m00.0s 00d00m00.0s,
```

where the coordinates are J2000 RA and Declination, respectively. It is sufficient to set the common phase centre to the central coordinates from any one of the interleaves. The central coordinates of each interleave can be found on the GASKAP-HI Google spreadsheet. One can run a bash loop to set phase centre for all measurement sets in a single directory:

```
-> for msFile in *.ms; do chgcentre ${msFile} 01h44m55.0s -70d32m31s; done
```

The `chgcentre` task comes as a part of the `WSClean` package that uses `CASACore` to perform this phase rotation, and requires a table of Observatories to run properly. If one encounters the error of "Cannot read table of Observatories" during this step: create a new `$HOME/.casarc` file in your home directory if it is not already there and add:

```
# Add this line to your $HOME/.casarc file in Linux (e.g., Avatar server at Stromlo)
measures.directory: /usr/share/casacore/data

# Or (this can also work)
# measures.directory: /pkg/linux/casa-release-5.4.1-32.el7/data
# or in Mac (if CASA is installed on your personal Macbook).
# measures.directory: /usr/local/share/casacore/data
```

2.5 (Optional) Concatenating Multiple SBIDs

The control script, `contcat_All.pbs`, can be edited to concatenate associated measurement sets for the same beam for observations of the same field over multiple sessions. For example, pilot observations of a field centered on the Magellanic Stream were performed in two 12 hour pointings with the SBIDs 14158 and 14211. Assuming all 108 measurement sets (36 beams×3 interleaves) have been phased rotated to a common field center, edit the `fieldName`, `SBID_1`, `SBID_2`, and `baseDataPath` variables in `contcat_All.pbs` to reflect the location and general names of the measurement set files. To ease the formatting of the output file name strings, make a directory for the output concatenated files (e.g., `14158_14211`). Double check that the variables, `msName_1` and `msName_2` point to the location of the individual measurement sets for each SBID and the variable `concatName` is formatted such that it points to the created output directory and reflects that the output MSs are concatenated from two separate SBIDs. For example, the concatenated output MS for beam 00 from interleave A could be: `scienceData_SB14158_SB14211_GASKAP_M344-06_A.beam00_SL.ms`. Finally, submit to start 18 separate batch jobs, each handing the concatenation of 6 total measurement set files (2 beam×3 interleaves):

```
-> qsub concat_all.pbs
```

Note that, the `contcat_All.pbs` script, along with other useful `.pbs` scripts, can be found under the `pbsScripts` directory within the `GASKAP Imaging` package.

2.6 (Optional) Binning

Note that while this velocity binning was done to some earlier Pilot I data, it has since been decided that we will not do this for our current data (i.e., we use the native velocity resolution). Thus, this step should be skipped. Nonetheless, we maintain the below text here for bookkeeping.

Recall that the ideal radiometer equation states the noise in a given spectral channel is inversely proportional to the square root of the frequency width. It therefore often desirable to bin measurement sets along the spectral axis to increase the signal-to-noise. The script, `bin_All.pbs`, will bin each of the 108 MSs for a given GASKAP field by a user defined integer number of channels in a series of 18 batch jobs (each processing 6 total measurement set files (2 beams×3 interleaves). First, edit the line that changes to directory to ensure the script can find the corresponding `CASA` script; e.g.,

```
cd /avatar/nipingel/ASKAP/SMC/pros/GASKAP_Imaging/casaConfigScripts
```

Next, edit the `fieldName`, `SBID_1`, and `SBID_2`, and `baseDataPath` variables to again reflect the location and general names of the measurement set files. The remaining variable to set is the `width`. In general, our GASKAP observations are observed with a native spectral resolution of 1.156 kHz (∼0.24 km s^{-1} at the rest frequency of the HI line). It is usually desirable to set this value to 4 to decrease the spectral resolution to 0.98 km s^{-1}, which gives a good balance between spectral resolution and overall sensitivity. In some science cases, such as when the HI needs to be compared with molecular gas, this factor may be reduced to 2, or skipped altogether. The final paths to the measurement sets should be completely defined as such:

```
msName=${dataPath}/scienceData_SB${SBID_1}_${SBID_2}_${fieldName}_${inter}.beam${beamNum}_SL.ms
dataPath=${baseDataPath}\${SBID_1}_${SBID_2}
```

Submit the job with

```
-> qsub bin_All.pbs
```

2.7 UV-based Continuum Subtraction

Continuum sources in spectral data cubes complicate the deconvolution process and can ultimately cause the cleaning to diverge. The flux from continuum sources can be removed by first estimating the continuum emission by fitting polynomial models to the real and imaginary parts of the visibilities over

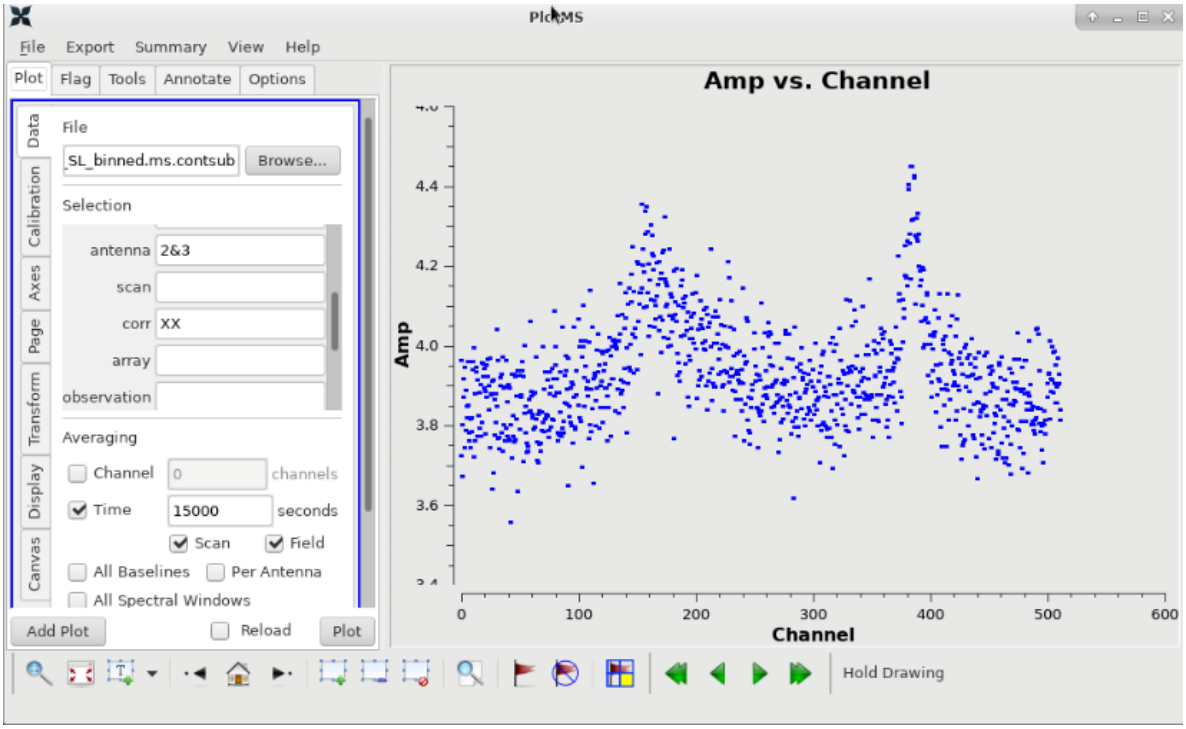

Figure 1: An average emission spectrum from a central beam (14A) that is useful for determining a range of emission-free channels.

a user defined range of emission-free spectral channels. These channels are determined by looking at the amplitude vs. channels for MSs for central beams (e.g., beam 14) using the `CASA` task, `plotms()`. An example spectrum is shown in Figure 1. Here, the visibilities from a pair of core antennas (2 and 3) have been selected along with the single XX polarisation. To further reduce clutter in the plot, amplitudes have been averaged over a time range of 15000 seconds, scan, and field. Finally, the scalar format has been selected (see lower left-hand corner). This can be achieved with the GUI or in CLI

```
-> casa
-> plotms(vis="scienceData.M355-07A.SB38509.M355-07A.beam14_SL.ms", xaxis="chan",
    antenna="2&3", correlation="XX", avgtime="1.5e4", avgscan=True, avgfield=True,
    scalar=True)
```

We see two clear peaks in the emission between channels ∼130 to 225 and 360 and 440. These peaks correspond to emission for the Magellanic Stream and Milky Way foreground, respectively. The spectra is also relatively flat. We can therefore fit a polynomial model of order 1 to the emission-free channel range: 0 to 100, 240 to 320, and 440 to 500. Check a few more measurement sets for outer beams (e.g., 0 and 35) to make sure the emission profiles are more or less similar.

The control script, `uvcontsub_All.pbs`, is used to submit a series of batch jobs to model and subtract the continuum in each MS file. Similarly, to the binning and concatenation, 18 total jobs will be submitted, each processing 6 total measurement sets: 2 beams×3 interleaves. [For organization purposes, it is recommended that you create a directory called BINNED to store your binned MSs.] In this script, again edit the `fieldName`, `SBID_1`, and `SBID_2`, and `baseDataPath` variables to again reflect the location and general names of the measurement set files. The remaining variables to set are `channelRange` and `order`. Based on our averaged emission profiles, we set `chanRange="0 100 240 320 440 500"` and `order=1`. Finally, double check that the `msName` variable gives the correct path to MS files, similar to the previous sections. The script is submitted with the command:

```
-> qsub uvcontsub_All.pbs
```

AM: fixme upload scirpt with only one SBID

3 Imaging

The primary script that is used for imaging is `mpiMSCLEAN_wsclean.pbs`. Similar to the scripts used to prep the measurement set files, there are several minor edits to make that tells the script where to find the data and several variables to set that are specific to the imaging. However, before submitting this script, we must first split out individual spectral channels from each of the 108 measurement set files. This is done to better handle the overall memory footprint and accelerate processing. Generally, chunks of 28 channels are split out and processed at a time. This seemingly arbitrary number is linked to the number of available cores (56) on a given Avatar large memory node. This ensure each imaging job for a given spectral channel is allocated 2 total cores.

3.1 `splitByChannel_All.pbs`

The script, `splitByChannel_All.pbs`, will split out a user defined number of channels. As pointed out above, we generally split out 28 channels at a given time, though one can choose any number of channels to split out. The useful variables to build a path to the MS files are again `fieldName`, `subDir`, `SBID_1`, and `SBID_2`, and `baseDataPath`. The remaining variables to set are `startChan` and the increment to `startChan` in the following line (usually set to `+ 28`); `startChan` is of course the beginning channel to split out from a measurement set and the increment dictates the total number of channels to split out. The channel range is *not* inclusive. For example, if `startChan=100` and the increment is 28, the last channel split out is 127. After double checking the variables `dataPath` and `msName` point to the location of the MS files, submit the script using

```
qsub splitByChannel_All.pbs
```

Some things to note:

- it is not recommended to image every available spectral channel since most of them contain no signal. However, it is useful in scientific analysis to have a few signal-free channels to assess the rms noise levels. Choose a range of channels that incorporates 10 to 15 emission free channels on either end of the spectral range. Using the time-averaged emission profile in Figure 1) as a guide, a good channel range would be 100 to 463 to capture the full spectral range of emission associated with the Magellanic System and Milky Way foreground emission while also being an divisible by a factor 28: (464-100)/28 = 13.
- for whatever reason, these jobs tend to die/stall. Watch the run time stats (R column when looking at `qstat -Jt`. If the run time stats are not increasing more or less in real time (e.g., the run time has not surpassed 10 minutes after ∼45 minutes have gone by, kill and restart the jobs.
- the number of output files is quite large (28 channels×36 beams× 3 interleaves = 3024!!). For better organization and making the paths to the MS files easier for the imaging stage, create a series of directories to store the split out measurements per channel. This can be accomplished in two bash for-loops in the directory where the split-out MS files live:

```
-> for i in {156..183}; do mkdir chan${i}; done
-> for i in {156..183}; do mv *_chan${i}.ms chan${i}; done
```

3.2 `mpiMSCLEAN_wsclean.pbs`

The script, `mpiMSCLEAN_wsclean.pbs`, is the main event. It launches a series of 28 individual batch jobs, each allocated 2 computing cores, to jointly deconvolve the emission associated with one of the split-out spectral channels. The most important variables to set in this script are: the command to change directory to the top level directory where the measurement set files are stored. That is, the directory above the 28 directories that each store the 108 measurement set file for each spectral channel. This command exists directly above the other important variable `chanNum`, which dictates which channel is being imaged in the batch job. If you had set `startChan` to 100 in `splitByChannel_All.pbs`, edit the line to read: `chanNum=$((100 + $job_num))`; `$job_num` is automatically set based on the assigned job ID, which range between 0 and 27, inclusive (see the PBS preamble line: `PBS -J 0-27`.

Unfortunately, we must share the cluster with other researchers. In regards to Avatar, we can check the status of each of the largemem nodes using the command `pbsnodes -a`. The output will show the status of each nodes. Look for nodes 29-40 to have `state=free`. You can then select this specific node for imaging by adding an additional resource allocation:

```
#PBS -l select=1:ncpus=4:mpiprocs=1:host=avatar40
```

Most of the imaging parameters do not need to be touched, as they have been tweaked to deal specifically with the extended emission targeted by GASKAP-HI.

3.2.1 Tracking Progress & Good Results

The progress of the submitted jobs can be tracked using the command `qstat -Jt`. The PBS command, `qstat`, is used to list the status of the job queue and the flags `-Jt` list the status of specifically batch jobs. Figure 2 shows an example output when imaging a chunk of 28 channels. The jobs are progressing as long as the `S` column lists `R` next to the job. If you need to kill these or any other batch jobs, note the job ID (first column; in this case 178494) and type

```
qdel 178494[]
```

One can check the explicit progress of a job by monitoring the output from the generated log files for each imaging batch job. For example, to monitor the log file `GASKAP_SB14158_14211_chan400.log`, type

```
-> less GASKAP_SB14158_14211_chan400.log
```

which allows a user to read the log file while it updates in real time. To go immediately to the end, hit Shift+F. Exit by hitting CTRL+C and then q.

The most important aspects of the jobs is to ensure the residual peak flux are gradually dropping and converging to a single value that is some multiple factor of the noise. This can be seen after the completion of an imaging job by parsing the log file. Open the log file in the `vim` editor: `vi GASKAP_SB14158_14211_chan400.log` and search for a occurrences of the string `iteration` by typing `:/iteration` and hitting ENTER. Cycle through occurrences by hitting the N key (you can cycle back by hitting SHIFT+N). Figure **??** shows a portion of the log where the peak fluxes for minor cycle iterations are displayed. For each minor cycle iteration, the peak flux drops gradually and begins to converge towards a single value. Once the minor cycle threshold — which is computed internally — is reached, a major cycle will begin where the model components are converted to visibilities and re-gridded. Generally, deconvolved images are obtained within 2 to 3 major cycles. The telltale sign of divergence happens when the peak residual flux begins to increase. If you notice this happening, kill the jobs and reduce the total number of iterations and/or the largest scale for the multi-scale clean.

A series of output FITS files will be created after the completion of an imaging job. The most important of these end of the suffixes: `*-dirty.fits`, `*-beam-I.fit`, and `*-image.fits`. Figure 3 shows an example of an image from a single spectral channel that has been adequately cleaned. Artifacts stemming from the sidelobe response are clearly visible around areas of bright emission in the dirty image shown in the top panel (with the suffix `*-dirty.fits`. These are effectively entirely mitigated in the cleaned image in the bottom panel (with the suffix `*-image.fits`). The larger-scale bowls of negative emission are mostly due to missing large scale emission that has been filtered out and will be corrected during combination with Parkes data (see Section 3.3).

3.2.2 ASKAP Primary Beam Models

The advantage of `WSClean` over similar imaging software packages, such as `CASA`, is the ability to provide user defined primary beam models for each individual beam using the PAF gridding mode. This is provided to `WSClean` through an a-term config file. For the sake of being concise, imagine ASKAP only formed 2 beams instead of 36 but kept the same interleaving scheme (2 beams×3 interleaves). A correctly formatted configuration file would then be:

```
aterms = [ paf ]
```

```
nipingel@avatar:/avatar/nipingel/ASKAP/SMC/pros/GASKAP_Imaging/pbsScripts>qstat -Jt
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
178494[0].avatar  wsc_1            nipingel          01:46:23 R largemem
178494[1].avatar  wsc_1            nipingel          01:46:41 R largemem
178494[2].avatar  wsc_1            nipingel          01:46:42 R largemem
178494[3].avatar  wsc_1            nipingel          01:46:28 R largemem
178494[4].avatar  wsc_1            nipingel          01:46:30 R largemem
178494[5].avatar  wsc_1            nipingel          01:46:31 R largemem
178494[6].avatar  wsc_1            nipingel          01:46:52 R largemem
178494[7].avatar  wsc_1            nipingel          01:46:30 R largemem
178494[8].avatar  wsc_1            nipingel          01:46:58 R largemem
178494[9].avatar  wsc_1            nipingel          01:46:29 R largemem
178494[10].avatar wsc_1            nipingel          01:46:24 R largemem
178494[11].avatar wsc_1            nipingel          01:46:10 R largemem
178494[12].avatar wsc_1            nipingel          01:46:15 R largemem
178494[13].avatar wsc_1            nipingel          01:46:47 R largemem
178494[14].avatar wsc_1            nipingel          01:46:40 R largemem
178494[15].avatar wsc_1            nipingel          01:46:34 R largemem
178494[16].avatar wsc_1            nipingel          01:46:17 R largemem
178494[17].avatar wsc_1            nipingel          01:46:35 R largemem
178494[18].avatar wsc_1            nipingel          01:46:48 R largemem
178494[19].avatar wsc_1            nipingel          01:46:47 R largemem
178494[20].avatar wsc_1            nipingel          01:45:39 R largemem
178494[21].avatar wsc_1            nipingel          01:46:17 R largemem
178494[22].avatar wsc_1            nipingel          01:46:26 R largemem
178494[23].avatar wsc_1            nipingel          01:46:09 R largemem
178494[24].avatar wsc_1            nipingel          01:45:59 R largemem
178494[25].avatar wsc_1            nipingel          01:46:21 R largemem
178494[26].avatar wsc_1            nipingel          01:46:27 R largemem
178494[27].avatar wsc_1            nipingel          01:46:27 R largemem
```

Figure 2: Status of the imaging jobs submitted for 28 individual channels. The jobs are progressing if the `S` column lists `R` next to the job.

```
paf.antenna_map = [00 01]
paf.beam_map = [00 01 00 01 00 01]
paf.beam_pointings = [00h23m16.0172s -74d25m50.8316s 01h26m55.5566s -68d35m15.1745s
    01h34m07.4035s -67d57m35.3812s 01h31m56.5462s -68d50m15.7817s 01h29m23.6643s
    -68d13m59.6301s 01h26m59.6257s -69d06m24.9018s]
paf.file_template =
    /avatar/nipingel/ASKAP/SMC/data/pilot_obs/ms_data/10941_10944/wsclean-test/beam_maps
    /holography_beams/ASKAP_BEAM$BEAM.fits
paf.reference_frequency = 1420.406e6
```

The `paf.beam_pointings` parameter requires the individual beam phase centers for each measurement set. Note that the order of measurement sets maps directly to these phase centers. This should not be an issue if one places the split-out channels into their own directories, as advised above. The individual phase centers can be obtained from the utility script, `utils/genPhaseCenterList.py` (see Section 4.1). An example configuration file that is formatted for all 36 beams is provided in `misc/beammap_template.config`.

3.2.3 Clean Mask

Cleaning masks are employed to tell `WSClean` where to look for strong components in the image plane for deconvolution. A clean mask is essentially a 2D image with the same pixel grid as an output image with pixel values of 1 or 0. A value of 1 signifies the pixel can be considered for cleaning, while a value of 0 means the pixel will be ignored in cleaning. Since we do not know the location of strong components a priori, it is easiest to construct a clean mask based on the response level of the sensitivity across the ASKAP field-of-view. This can be accomplished by selecting the beam response output FITS file of the form `*_beam-I.fits` from an imaging run. A mask can be constructed to include only pixels above the 85% response level with a few simple python commands:

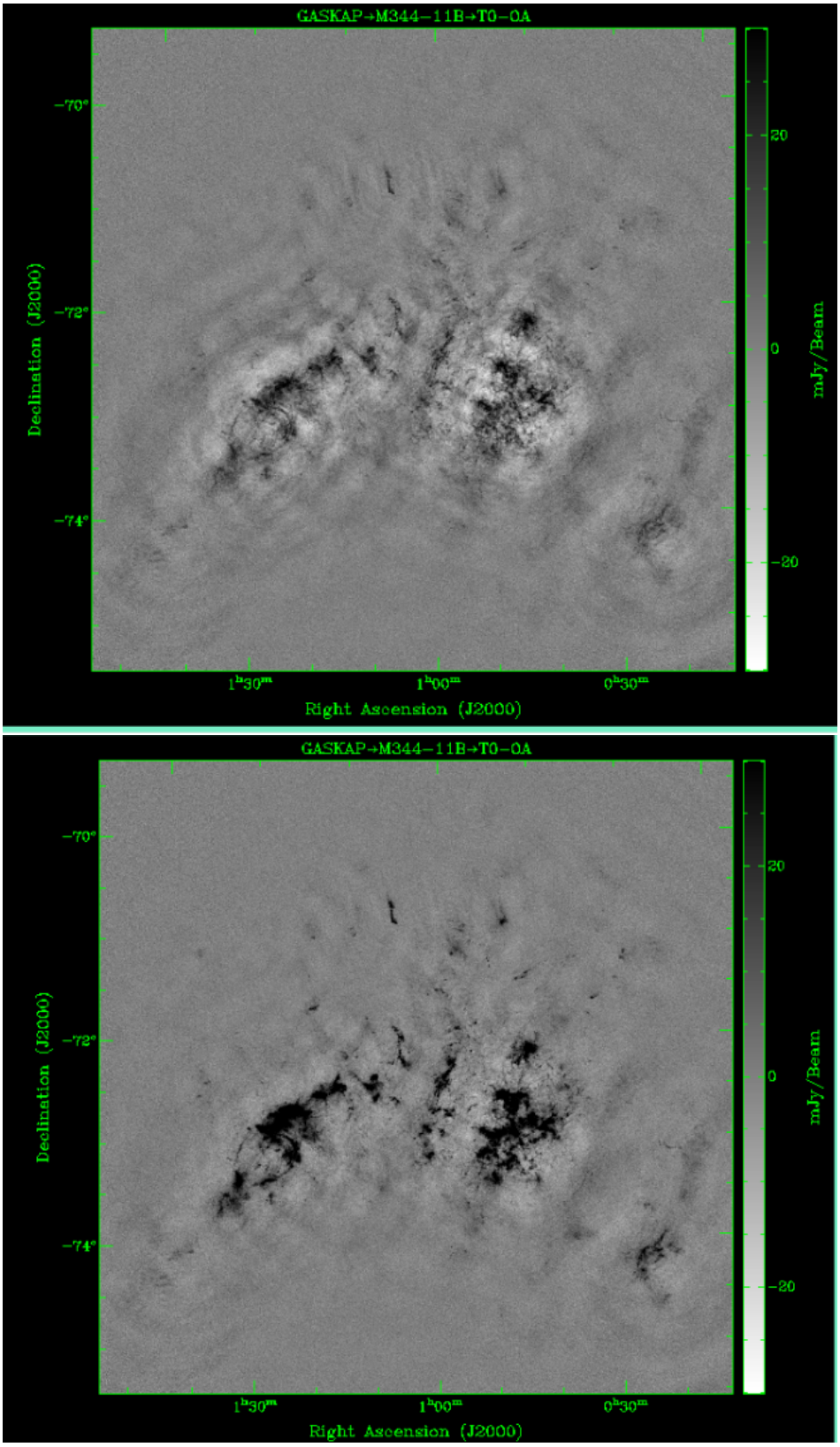

Figure 3: Top: dirty image of an imaging batch job a single spectral channel. Bottom: Resultant clean image. Note that the sidelobe response around areas of bright emission are effectively mitigated. The larger-scale bowls of negative emission are mostly due to missing large scale emission that has been filtered out and corrected during combination with Parkes data (see Section 3.3)

.

```
->ipython
->from astropy.io import fits
->import numpy as np
->hdu = fits.open('*_beam-I.fits')
->data = hdu[0].data
->mask_image = np.zeros(data.shape)
->inds = np.where(data > 0.85)
->mask_image[inds] = 1.0
->mask_image[0, 0, 4900:, :] = 0.0
->fits.writeto('mask.fits', mask_image, header = hdu[0].header)
->exit()
```

The third line from the bottom is to account for `WSClean` sometimes placing a pocket of high response (at the ∼50% level) towards the upper right-hand corner of the image. This step sets all rows past column 4900 to 0.0 to ensure that these erroneous pixels do not contribute to the mask. Note the first two redundant frequency and stokes axes that are included in the mask.

Convert the output FITS file into a `CASA` image format by using the task `importfits()` and provide the path in `mpiMSCLEAN_wsclean.pbs` using the `mask_path` variable.

3.2.4 Imaging Sub-regions

It is of course scientifically interesting to focus on various sub-regions of a full GASKAP field to e.g., tweak the uv-weighting or angular and spectral resolution to better match ancillary molecular gas or dust data sets. The scripts, `bin_subset.pbs`, `uvcontsub_subregion.pbs`, `splitByChannel_subregion.pbs` and `mpiMSCLEAN_wsclean_subregion.pbs` can accomplish all of the necessary prep and imaging of a subset of MS files and images. This is done by editing several bash arrays that such that only certain beams from certain interleaves are selected for processing. For example, say we wish to focus on a 1 deg region in the southwest bar of the SMC centered on RA=00h48m07.3s, Dec=−73d14m03.7s. We use the utility script, `return_beams.py`, to determine which beams lie within 1 deg of these coordinates. This script printed out 12 total beams: 07A, 08A, 13A, 14A, 06B, 07B, 13B, 14B, 07C, 13C, 14C, and 19C. The number is of course the beam number and the proceeding letter denotes the interleave (A, B, or C).

Now suppose we wish to only bin along the frequency axis of the MSs by a factor of 2 (instead of the usual 4) to obtain a spectral resolution of 0.5 km s^{-1}, which matches closely matches a corresponding CO data set. First create a separate directory called "SWBAR" and copy the calibrated MS files from CASDA for each beam/interleave into it. Then edit `bin_subset.pbs` the following way:

- change the number of batch jobs to match the number of total number of subset beams (e.g., `#PBS -J 0-11` in our example)
- edit the `beam_array` and `inter_array` bash array variables to store our subset beams

  `beam_array=( 07 08 13 14 06 07 13 14 07 13 14 19 )`

  `inter_array=( A A A A B B B B C C C C )`

  note the bash syntax: spaces following parentheses and lack of space between characters on either side of the =.
- edit the `SBID_1`, `SBID_2`, `fieldName`, `subDir`, and `msName` variables as above when imaging the full 108 beam data set to construct the path to our subset of beams. Set `width` to 2.
- submit just as before.

The same path construction variables in the scripts `uvcontsub_subregion.pbs` and `splitByChannel_subregion.pbs` can similarly be set. Task-specific variables, such as `chanRange` and `order` (see Section 2.7) should also be set. Finally, the subset of beams can be imaged with the same `mpiMSCLEAN_wsclean.pbs` script as is used for imaging all 108 MS files. To do so, simply place the split-out channels for the subset of beams in their own directories as advised in Section 3.2 and run with your ideal parameters.

3.3 Combining ASKAP and Parkes

The last step in data processing is to combine the individual images into an ASKAP-only cube and feather this together with Parkes data to correct for short-spacings. The utility script, `combineImages.py` can collate the individual deconvolved images into the ASKAP-only cube. The script inputs are:

```
->python combineImages.py -h
usage: combineImages.py [-h] -f FILEEXT -d FREQRES -o OUTFILE [-r RESTFREQ] [-l REFFRAME]

optional arguments:
  -h, --help          show this help message and exit
  -f FILEEXT, --fileExt FILEEXT
                      <required> file extension (e.g., image)
  -d FREQRES, --freqRes FREQRES
                      <required> frequency resolution in kHz
  -o OUTFILE, --outFile OUTFILE
                      <required> name of FITS cube
  -r RESTFREQ, --restFreq RESTFREQ
                      rest frequency in MHz (default is 1420.405752)
  -l REFFRAME, --refFrame REFFRAME
                      frqeuency reference frame (default is TOPO)
```

The cube is creating using a few `CASA` tasks, thus it is necessary to convert the input files (in this case those with the *-image.fits suffix) into `CASA` image format like so:

```
->casa --nologfile --log2term
->import glob
-> file_list = glob.glob('*-image.fits')
-> for f in file_list:
    importfits(fitsimage = f, imagename = f.replace('.fits', '.im'))
```

An ASKAP-only cube with the output name MG_Stream_askap_Jy_lsrk.fits with the command:

```
->casa --nologfile --log2term -c combineImages.py -f im -o MG_Stream_askap_Jy -d 1.157 -l
    LSRK
```

where 1.157 is the frequency resolution in kHz of single channel (and therefore, should be changed to, e.g., 4.628 when binning by 4 channels) and `_lsrk.fits` is appended to denote that the output spectral reference from has been shifted to the local kinematic standard of rest (LSRK) from topocentric. Note that the `glob` python package will not order the channels correctly if one is including channels below 100. Fix this by adding a leading '0' character to the the FITS file for these channels:

```
-> for i in {87..99}; do mv GASKAP_SB14158_SB14211_chan${i}-image.fits
    GASKAP_SB14158_SB14211_chan0${i}-image.fits; done
```

The ASKAP-only cube is actually the sky brightness distribution multiplied by the primary beam response. We must therefore divide out the primary beam response to ensure the flux values are not biased too low and account for the drop in sensitivity towards the edge of the PAF footprint. To do so, create a cube of the `*-beam-I.fits` files for each spectral plane following the same procedure as above (i.e., convert the FITS files to `CASA` image format and create a similar cube):

```
->casa --nologfile --log2term
->import glob
-> file_list = glob.glob('*-beam.fits')
-> for f in file_list:
    importfits(fitsimage = f, imagename = f.replace('.fits', '.pb'))
```

An ASKAP-only cube with the output name `MG_Stream_askap_Jy_lsrk.fits` with the command:

```
->casa --nologfile --log2term -c combineImages.py -f pb -o MG_Stream_askap_PB -d 1.157 -l
    LSRK
```

While running this to produce the primary beam cube (as well as the residual cube below), an error will occur during the `imreframe` step. This is normal, though because of this the resulting primary beam cube is not exported to .FITS format. Therefore, one needs to manually do this with task `exportfits`:

```
->casa --nologfile --log2term
->exportfits(imagename='MG_Stream_askap_PB.combImage',
    fitsimage='MG_Stream_askap_PB_lsrk.fits')
```

For whatever reason, the primary beam response sometimes contains pixels with values > 1.0. To fix this, read the cube into python and simply normalize it by the maximum value and create a new FITS cube

```
-> ipython
-> from astropy.io import fits
-> import numpy as np
-> hdu = fits.open('MG_Stream_askap_PB_lsrk.fits')
-> data = hdu[0].data
-> max_value = np.max(data)
-> norm_data = data/max_value
-> fits.writeto('MG_Stream_askap_PB_lsrk.fits', norm_data, header = hdu[0].header, overwrite
    = True)
```

Finally, while it is not necessary for the production of the final image cube, it is a good idea to produce a residual cube to ensure that sufficient CLEANing has been achieved. This can be done by first converting the residual `.fits` files into `CASA` image format

```
->casa --nologfile --log2term
->import glob
-> file_list = glob.glob('*-????-residual.fits')
-> for f in file_list:
   importfits(fitsimage = f, imagename = f.replace('.fits', '.res'))
```

This is then followed by using `combineImage.py` to form the residual cube:

```
->casa --nologfile --log2term -c combineImages.py -f res -o MG_Stream_askap_res -d 1.157 -l
    LSRK
```

Before combination, we must obtain data from the The Parkes Galactic All-Sky Survey (GASS; McClure-Griffiths et al. 2009; Kalberla & Haud 2015) that covers the region observed with ASKAP. These cubes can be requested from filling out the form on this web-page: https://www.astro.uni-bonn.de/hisurvey/gass/. Figure 4 shows an example of a filled in request form. The region is centered on the field center and extends 25° in RA and Dec and from -300 km s^{-1} to 300 km s^{-1} in the LSRK velocity frame. This coverage ensures complete spatial and spectral overlap with the GASKAP data. A link to download the cube will be emailed to your account when it is ready.

To proceed with the combination, move the GASS cube to the same directory as the ASKAP-only cube. The final preparations for the combination include reading the files into `miriad` format, editing the header of the GASS cube to reflect the true resolution of 960″, converting the brightness units of the GASS data into units of Jy/beam, and re-gridding the Parkes cube to be on the same spatial/spectral scales. This can be accomplished with the series of miriad commands:

```
-> miriad
-> tput fits
-> in=MG_Stream_askap_Jy_lsrk.fits
-> out=MG_Stream_askap_Jy_lsrk.mir
-> op=xyin
-> go
-> in=MG_Stream_askap_PB_lsrk.fits
```

```
-> out=MG_Stream_askap_PB_lsrk.mir
-> go
-> in=GASS_cube.fits
-> out=GASS_cube.mir
-> go
-> tput maths
-> exp=GASS_cube.mir*960*960/6.07e5
-> out=GASS_cube_Jy.mir
-> go
-> tput puthd
-> in=GASS_cube_Jy.mir/bunit
-> value='Jy/beam'
-> go
-> in=GASS_cube_Jy.mir/bmaj
-> value=960.0, arcsec
-> go
-> in=GASS_cube_Jy.mir/bmin
-> go
-> tput regrid
-> in=GASS_cube_Jy.mir
-> out=GASS_cube_Jy.regrid
-> tin=MG_Stream_askap_Jy_lsrk.mir
-> go
```

We are now ready to divide out the primary beam response and use the `miriad` task `IMMERGE` to linearly merge the ASKAP and Parkes data in the Fourier Plane to fill in the missing short-spacings. See Stanimirovic (2002) for details on this technique. The commands in a miriad environment are

```
-> tget maths
-> exp=<MG_Stream_askap_Jy_lsrk.mir>/<MG_Stream_askap_PB_lsrk.mir>
-> out=MG_Stream_askap_Jy_PBC_lsrk.mir
-> mask=<MG_Stream_askap_PB_lsrk.mir>.gt.0.05
-> go
-> tput immerge
-> in=MG_Stream_askap_Jy_PBC_lsrk.mir,GASS_cube_Jy.regrid
-> out=MG_Stream_askap_parkes_PBC_Jy.immerge
-> factor=1.0
-> options=notaper
-> go
```

The `PBC` denotes the primary beam correction has been applied. Explicitly setting the `factor` parameter to 1.0 applies a scaling factor to the the Parkes data such that we assume the flux scale is correct. We have found that this is a reasonable assumption. Letting `IMMERGE` solve for this factor itself results in a suggested scaling factor of 0.5, which is incorrect. This is likely due to an inconsistent internal definition between the XX and YY polarisations between `miriad` and `ASKAPsoft`. The `notaper` option avoids tapering the low-resolution image to match the residual primary beam response of the high-resolution image. `IMMERGE` will complain and crash if this option is not set because the ASKAP primary beam is not known to `miriad`. This step will take quite a bit of time. Do not fret (unless there is an error...)! The most common errors is a mismatch between the number of channels or some reference frame. The final cubes can be written out to FITS format using the `miriad` FITS task with `op=xyout`:

```
-> tget fits
-> in=MG_Stream_askap_parkes_PBC_Jy.immerge
-> out=MG_Stream_askap_parkes_PBC_Jy.fits
-> op=xyout
-> go
```

User information		
Your name	Nickolas Pingel	
Your email	Nickolas.Pingel@anu.edu.au	
Data options		
Data type	Cleaned data GASS III	
Spacial properties		
Coordinate system	Equatorial coordinates (RA, DEC, J2000)	
Center	RA [h m s]/ l [°]	01 44 55.379
	Dec [±° ' "]/ b [°]	70 32 31.07
Width	RA/l [°]	25
	Dec/b [°]	25
Degrees per Pixel [°]		0.08
Smoothing Kernel Radius FWHM [°]		0.125
Velocity properties		
Velocity Range (VLSR)	v_{min} [km/s]	-300
	v_{max} [km/s]	300
	Δv [km/s]	0.8
Submit query		

Figure 4: Example of a filled-in form to obtain a GASS cube with which to combine with an ASKAP-only cube. The region is centered on the field center and extends 25° in RA and Dec and from -300 km s^{-1} to 300 km s^{-1} in the LSRK velocity frame. This coverage ensures complete spatial and spectral overlap with the GASKAP data.

3.4 Final Mask and Extracting a sub-region

One may notice that an internal mask based on the primary beam response is NOT applied during imaging. This results in erroneous noise extending up to the edge of the image. Furthermore, the 5000×5000 can be cut down once the mask is applied. A utility script, `genPBMask.py`, is available to apply a mask based on the final combined cube. The inputs are:

```
-> python genPBMask.py -h
usage: genPBMask.py [-h] -i IMAGECUBE -p PBCUBE -t THRESHOLD -o OUTPUT

optional arguments:
  -h, --help          show this help message and exit
  -i IMAGECUBE, --imageCube IMAGECUBE
                      <required> full path to image cube
  -p PBIMAGE, --pbimage PBIMAGE
                      <required> full path to primary beam cube (MUST HAVE
                      SAME DIMENSIONS AS INPUT IMAGE CUBE)
  -t THRESHOLD, --threshold THRESHOLD
                      <required> minimum primary beam response level pixel
                      to include in mask
  -o OUTPUT, --output OUTPUT
                      <required> name of output file
```

The input image for the primary beam is assumed to be an average primary beam response measured across the spectral range of the image cube. This is quickly computed using a few python commands:

```
-> ipython
-> from astropy.io import fits
-> import numpy as np
-> hdu = fits.open('MG_Stream_askap_PB_lsrk.fits')
-> data = hdu[0].data
-> mean_data = np.mean(data, axis = 0)
-> fits.writeto('MG_Stream_askap_ave_PB.fits', mean_data, header = hdu[0].header)
```

A general PB-mask in `CASA` is at the 5% level. So let's apply this the our combined cube:

```
-> python genPBMask.py -i MG_Stream_askap_parkes_PBC_Jy.fits -p MG_Stream_askap_ave_PB.fits
    -t 0.05 -o MG_Stream_askap_parkes_PBC_Jy_mask
```

This command will output a masked cube. Sometimes, there are some peaks in the primary beam response that fall outside the PAF footprint and above the 0.05 mask level. These can be ignored as they are thrown away when the data are trimmed spatially. Trimming is necessary because the 5000×5000 imaged region is quite large relative to the PAF footprint. A sub-region can be extracted using the `miriad` task `IMSUB`. Determine the maximum and minimum range of x and y pixels that fully encapsulates the full image. Then follow the commands:

```
-> miriad
-> tput fits
-> in=MG_Stream_askap_parkes_PBC_Jy_mask.fits
-> out=MG_Stream_askap_parkes_PBC_Jy_mask.mir
-> op=xyin
-> go
-> tput imsub
-> in=MG_Stream_askap_parkes_PBC_Jy_mask.mir
-> out=MG_Stream_askap_parkes_PBC_Jy_mask.imsub
## x_min, y_min, x_max, y_max
-> region=box(380, 730, 4420, 4500)
-> go
-> tput maths
-> exp=<MG_Stream_askap_parkes_PBC_Jy_mask.imsub>*673
-> out=MG_Stream_askap_parkes_PBC_K_mask.imsub
```

```
-> go
-> tput puthd
-> in=MG_Stream_askap_parkes_PBC_K_mask.imsub/bunit
-> value='K'
-> go
-> tget fits
-> in = MG_Stream_askap_parkes_PBC_K_mask.imsub
-> out = MG_Stream_askap_parkes_PBC_K_mask_subregion.fits
-> op=xyout
-> go
```

We now have a final cube!

4 Other Utility Scripts

This section outlines the usage of several utility scripts that collate important metadata, such as the phase centers of individual beams, and final data manipulation steps.

4.1 genPhaseCenterList.py

The PAF mode of `WSClean` requires a configuration file that lists the corrsponding phase center for each input MS file. A list of beam phase centers can be generated using the utility script, `genPhaseCenterList.py`. The usage is summarized below:

```
usage: genPhaseCenterList.py [-h] -p PATH [-e PREFIX] -o OUTPUT -a INTER_A -b INTER_B -c
    INTER_C
optional arguments:
  -h, --help          show this help message and exit
  -p PATH, --path PATH <required> path to directory containing MS files
  -e SUFFIX, --suffix
                      <optional> file prefix for MS file (default *.ms)
  -o OUTPUT, --output OUTPUT
                      <required> name of output file to store beam phase
                      centers
  -a INTER_A, --inter_A INTER_A
                      <required> center coordinate for A interleave (J2000,
                      MUST be in 00h00m00.0s,00d00m00.0s format)
  -b INTER_B, --inter_B INTER_B
                      <required> pcenter coordinate for B interleave (J2000,
                      MUST be in 00h00m00.0s,00d00m00.0s format)
  -c INTER_C, --inter_C INTER_C
                      <required> center coordinate for C interleave (J2000,
                      MUST be in 00h00m00.0s,00d00m00.0s format)
```

The script generates a list of individual beam phase centers by applying the offsets stored in the MS files to the three user input J2000 coordinates (formatted exactly as 00h00m00.0s,00d00m00.0s, including no space between ',' character) for the individual interleave centers. For example, if all 108 MS files that have been continuum subtracted (with the file suffix `.contsub` are stored in a directory `/avatar/nipingel/ASKAP/SMC/data/ms_data/10941_10944/CONTSUB`, a script can be generated with the command:

```
-> python genPhaseCenterList.py -p ./ -s contsub -o SMC_phase_centers -a
    00h58m43.280s,-72d31m49.03s -b 01h04m26.220s,-72d14m31.59s -c 01h04m58.229s,-72d45m36.56s
```

You can retrieve the interleave positions for phase II from this spreadsheet: https://docs.google.com/spreadsheets/d/10P-6ylrk24T-x-sI7vkinkO6kwbeiDPOFtMOEyVAI_Q/edit?usp=sharing

The use of the `genPhaseCenterList.py` may require the installation of ASKAP's `aces-packages` (https://bitbucket.csiro.au/projects/ACES/repos/aces-packages/browse), which requires a valid CSIRO PUMA account that needs to be arranged by getting into contact with CSIRO staff.

You can reformat the output file SMC_phase_centers so that it is ready to be copied and pasted into your XXX_beammap.config file with (careful when copying these quotation mark):

```
cut -c 61- SSMC_phase_centers.txt | awk '{print}' ORS=' '> SMC_phase_centers_formatted.txt
```

4.2 return_beams.py

The utility script, `return_beams.py` can be used to determine which beams/interleaves are included in a subregion of the field of view. The syntax is:

```
usage: return_beams.py [-h] -f FILE_NAME -r RA -d DEC -b RADIUS

optional arguments:
  -h, --help          show this help message and exit
  -f FILE_NAME, --file_name FILE_NAME
                      <required> path to list of beam phase centers generated by
                      phaseCenterList.py
  -r RA, --ra RA      <required> RA in hms format
  -d DEC, --dec DEC   <required> declination in dms format
  -b RADIUS, --radius RADIUS
                      <required> radius (in deg) over which to determine inclusion of beams
```

This script determines the beams whose phase centers fall within the user provided radius (in deg). The beams and associated interleaves are printed to the terminal. An example command to find beams within 1.0 deg of $\alpha_{\rm J2000}$ =05h33m00s, $\delta_{\rm J2000}$=−69d51m28s is below:

```
python return_beams.py -f phaseCenters_GASKAP_M000+02_SBID33047.txt -r 05h33m00s --d '
    -69d51m28s' -b 1.0
14 A
15 A
20 A
21 A
8 B
14 B
15 B
19 B
20 B
14 C
15 C
19 C
20 C
26 C
```

Note that a quotations and a space in the string providing the Declination are required to avoid errors thrown by python's `argparse` module. This is because `argparse` uses leading negative signs to indicate user arguments. This style of input is required for all negative Declinations.

References

Kalberla P. M. W., Haud U., 2015, A&A, 578, A78

McClure-Griffiths N. M., et al., 2009, ApJS, 181, 398

Stanimirovic S., 2002, in Stanimirovic S., Altschuler D., Goldsmith P., Salter C., eds, Astronomical Society of the Pacific Conference Series Vol. 278, Single-Dish Radio Astronomy: Techniques and Applications. pp 375–396 (arXiv:astro-ph/0205329)