Linear Image Mosaic Formation

W. D. Cotton (NRAO), November 25, 2024

***Abstract*—Details of a set of python scripts combining collections of pointing images into mosaics are discussed. These scripts should cover a fairly wide range of cases.**

***Index Terms*—Linear Image Mosaic**

I. Introduction

WIDE area radio interferometric sky surveys usually cover an area substantially larger that that viewed with good sensitivity in the area covered by a single antenna pointing. For such surveys, the survey area is covered by a pattern of overlapping pointings designed to give relatively constant sensitivity. This technique is usually referred to as forming a mosaic.

One approach to combining the overlapping data-sets is by means of a "linear mosaic" in which the individual data-sets are imaged and then combined into one or more "mosaics" covering the survey area. This memo describes a technique for linear image mosaic formation in the Obit package [1][1] .

II. Optimal Linear Mosaic Combination

The optimal weighting of overlapping images is weighting by the inverse variance of the noise. This weighting is achieved using weighting inversely proportional to the antenna gain squared. Noting that each pointing image has had the sky multiplied by one power of the gain, [2] gives the combination as:

$$M(x,y) \;=\; \frac{\sum_{i=1}^{n} P_i(x,y) I'_i(x,y)}{\sum_{i=1}^{n} P_i^2(x,y)}, \qquad (1)$$

where $M(x,y)$ is the output mosaic as a function of sky coordinates, n is the number of pointing images, P_i is the antenna power gain pattern on the mosaic grid for pointing i and I' is the pointing image evaluated on the grid of the mosaic. The result of this process, $M(x,y)$, will have the primary beam correction applied.

Pointing images will have one or more image planes. These may be either a "spectral line" cube or multiple sub-bands in a "continuum" image. These need to be individually combined into a mosaic with a similar structure.

In order to ensure a constant spatial resolution in the resultant mosaic image(s), the input pointing images can be convolved to a common resolution. Any known errors in the coordinates of the pointing images can be corrected prior to combination.

National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

[1]http://www.cv.nrao.edu/∼bcotton/Obit.html

III. Obit Implementation

The following sections describe how to create mosaics in Obit. The pointing FITS images, possibly gzip compressed, together with a template file used to define many of the aspects of the resultant mosaic files, are placed in the FITS data directory defined in script Mosaic.py described below. Executing the mosaicing scripts will result in the output FITS mosaic(s) in the FITS data directory. Intermediate data products are stored in AIPS format and are deleted upon script completion. Using multithreading, enabled in script Mosaic.py, and putting the AIPS directories on RAM disk will enhance performance.

A. Linear Mosaic

The linear mosaicing scheme described in section II reads through a list of pointing images and accumulates each pointing image, interpolated to the grid of the mosaic times the antenna gain function and the square of the antenna antenna gain function. These are the $\sum_{i=1}^{n} P_i(x,y) I'_i(x,y)$ and $\sum_{i=1}^{n} P_i^2(x,y)$ terms in Eq. 1. The input pointing images and the resultant mosaic image are in FITS format files in a common directory. The pointing images may be gzip compressed to minimize storage. Pointing cubes may be supplied either for multi-subband continuum images or spectral line cubes.

This is implemented in a set of python scripts to be executed from ObitTalk. These files are kept in $OBIT/share/scripts and should be copied to your working directory to be edited and renamed for the details of your project. These files are:

- **Targ.py** This file defines the desired mosaic including the name, center position, size and pixel spacing, Stokes parameter or transition name and whether or not the output is to be in Galactic, as opposed to Equatorial coordinates. The parameters which can be set are:
  – *stktrans:* The Stokes type or line transition name, input pointing file names are <file_root>.<stktrans>.fits where <file_root> is defined in PointingList.py. Files may be gzipped with suffix ".gz".
  – *size:* Mosaic image size in pixels.
  – *cells:* The pixel cell size in arcseconds.
  – *galactic:* True if the coordinates are to be converted from Equatorial to Galactic.
  – *targets:* A list of tuples defining the desired mosaic images. Each entry is of the form ('Name',[RA/GLong,Dec/GLat], [major,minor,PA]) where 'Name is the root of the mosaic FITS file name, full name <name>.<stktrans>_Mosaic.fits, [RA/GLong,Dec/GLat] is the center position in Equatorial or Galactic coordinates in degrees and

[major,minor,PA] is the beam size to which each pointing image resolution is to be convolved to (arcseconds, arcseconds, degrees). If no convolution is desired, the beam should be replaced by None. Mosaic FITS files will be written in the directory *fitsdir* set in Mosaic.py.

An annotated sample is given in Figure III-A.

- **PointingList.py** This file lists the pointing images to be considered for incorporation into each mosaic. This gives the root names, the J2000 pointing positions, a weight scaling factor and, potentially, astrometric corrections to be applied to the pointing image. The parameters which can be set are:
  - *data:* This is a list of tuples for each pointing image to be considered, ('Name','hh:mm:ss.s','dd:mm:ss.s', factor, off_ra, off_dec) where 'Name' is the file name root, the full name is <file_root>.<stktrans>.fits, <stktrans> is set in PointingList.py and the file resides in the directory *fitsdir* set in Mosaic.py. The pointing position in J2000 coordinates is given as 'hh:mm:ss.s' and 'dd:mm:ss.s' (note as strings), factor is a relative weighting scale factor, and off_ra, off_dec are pointing offsets in arcseconds to be applied to the image. These values are converted onto pixels (signed for RA) and added to the reference pixel of the input image:

```
d = acc.Desc.Dict;
ox=(cat[name][3]/3600.)/d['cdelt'][0];
oy=(cat[name][4]/3600.)/d['cdelt'][1];
d['crpix'][0] += ox;
d['crpix'][1] += oy
acc.Desc.Dict = d; acc.UpdateDesc(err)
```

  An annotated sample is given in Figure III-A.

- **Mosaic.py** This file gives other control parameters such as where the FITS and AIPS data are to be kept and the number of threads to use in processing. This is the top level script and calls the others as needed. The parameters which can usefully be set are:
  - *minWt:* Minimum summed weight for each pixel. Those with lower weights are blanked.
  - *doGPU:* Use a GPU for image interpolation if enabled. GPUs are VERY unforgiving when they run out of memory.
  - *minChan:* Minimum channel number (plane) to include, 1 rel.
  - *maxChan:* Maximum channel number (plane) to include.
  - *antSize:* Antenna diameter (m) used for primary beam correction. MeerKAT images processed in Obit will be recognized.
  - *datadisk:* disk for data (FITS) 0=CWD.
  - *fitsdir:* Directory (wrt datadisk or absolute) for input and output FITS images.
  - *accumdisk:* AIPS disk number to use for accumulators. Make this a RAM disk if possible for improved performance.
  - *maxd:* How far in degrees from the pointing center to consider the input images to overlap a given mosaic image.
  - *nThreads:* How many threads to use? This should not be more than the number of cores available.
  - *prtLv:* Level for diagnostic messages, 0-2.

  An annotated sample is given in Figure III-A.

- **mosaicBasic.py** This file has the software to generate the mosaics from the parameters specified in the previous files and may not need to be modified. However, if a customized beam pattern is needed it can be implemented here, see Section III-B. The output mosaic is defined by the specifications given in Targ.py and the FITS header of a template file, tmpl.fits, in the FITS data directory.
- **CheckPos.py** This file contains a routine to determine if a given pointing image overlaps with a given mosaic and may not need to be modified. It is given a parameter, *maxd*, set in Mosaic.py to specify the maximum distance from the pointing center to consider an overlap with the mosaic.

TABLE I
OBIT SUPPORTED PROJECTION CODES

Projection code	Type
-SIN	Sine
-TAN	Tangent
-ARC	Arc
-NCP	North Celestial Pole
-GLS	Global Sinusoid
-MER	Mercator
-AIT	Aitoff
-STG	Sterographic

Each run of the script will result in a log file of the processing as defined in Mosaic.py. For undetermined reasons, only one execution of Mosaic.py will work in a given execution of ObitTalk. The script is executed from ObitTalk:

```
>>> exec(open("Mosaic.py").read())
```

Other considerations:

1) No Prior Primary Beam Correction: The pointing images should not have had a primary beam gain correction applied. The mosaicing process will make this correction.

2) Pointing Image Coordinates: The pointing image FITS files should define the image coordinates with WCS keywords as used by Obit; i.e. CTYPEn, CRPIXn, CDELTn, CRVALn, CROTAn with a supported projection, see Table I.

3) Frequency/Velocity Axis: The third axis should be defined in frequency rather than velocity.

4) Template File: There should be a template FITS image file, tmpl.fits, in the FITS data directory with the pointing images whose header is used to populate the output mosaic header. The coordinate projection specified in this file should be one supported by Obit; these are given in Table I. The pixel contents of this file are not used.

Fig. 1. Targ.py

```
# Parameters defining output mosaics
stktrans = 'I'          # Stokes or line transition name
galactic = False        # Output in Galactic coordinates?
size    = [1200,1200]   # size of master accumulations/mosaic
cells   = 1.5           # cellsize (asec)
# triplets of mosaic name root, center position (deg, deg) and
#             beam (asec,asec,deg) or none
# Mosaic FITS files have names <mosaic name root>.<stktrans>_Mosaic.fits
# Example
targets =  [ \
             ('myMosaic_',[91.5, 19.0], [8.0,8.0,0.0]), \
]
#exec(open("Mosaic.py").read()) # to execute top level script
```

Fig. 2. PointingList.py

```
# Convert pointing list to catalog of pointings
# generates dict cat name:(ra, dec, factor, offx, offy)
# FITS file names <file root>.<stktrans>.fits (stktrans from Targ.py)
# FITS input can be gzipped
#  Pointing center positions are as 'hh:mm:ss.s", "dd:mm:ss"
#  factor is an addition weight scaling factor
#  offx, offy offsets in ra,dec (asec) (observed-unbiased)
# Example:
#           file root       RA(J2000)         Dec(J2000) factor off_ra off_dec
data = [ \
         ('06000+17576', '06:00:00.00000', '17:57:36.0000 ', 1.0, 0.0, 0.0), \
         ('06000+18243', '06:00:00.00000', '18:24:18.0000 ', 1.0, 0.0, 0.0), \
         ('06000+19178', '06:00:00.00000', '19:17:48.0000 ', 1.0, 0.0, 0.0), \
         ('06000+19446', '06:00:00.00000', '19:44:36.0000 ', 1.0, 0.0, 0.0), \
         ('06000+18510', '06:00:00.00000', '18:51:00.0000 ', 1.0, 0.0, 0.0), \
]
cat = {}
for d in data:
    cat[d[0]] = (ImageDesc.PHMS2RA(d[1]), ImageDesc.PDMS2Dec(d[2]), d[3], d[4],d[5])
```

B. Antenna Pattern

An important part of the mosaic formation is an accurate array primary beam (gain pattern). Unfortunately, this may be difficult to obtain. The antenna pattern can be accurately measured but in the presence of pointing errors, the effective gain pattern of the array will be broadened. Thus, the antenna pattern is usually what has to be used.

The default beam pattern used is a jinc function appropriate (above 1 GHz) for the antenna diameter given by antSize and centered at the celestial coordinate given in the pointing image header as keywords OBSRA and OBSDEC. MeerKAT images produced in Obit are recognized and a cosine squared beam is used. Pointing images produced by Obit task MFImage are recognized and the correct frequencies used to generate sub-band antenna patterns.

If a custom antenna beam is needed, it can be implemented in module mosaicBasic. A beam image with the same celestial grid as the associated pointing image and with the same channelization can be passed to routine PWeightImage as argument WtImage. This pattern should be normalized to 1 at the pointing position.

An "on-the-fly" (OTF) beam can be implemented in module mosaicBasic using optional arguments OTFRA and OTFDec in the call to PWeightImage for "Aussie" mode OTF mosaics. See [3] for an extended discussion of OTF imaging.

REFERENCES

[1] W. D. Cotton, "Obit: A Development Environment for Astronomical Algorithms," *PASP*, vol. 120, pp. 439–448, 2008.

[2] A. Brunthaler, K. M. Menten, S. A. Dzib, W. D. Cotton, F. Wyrowski, R. Dokara, Y. Gong, S. N. X. Medina, P. Müller, H. Nguyen, G. N. Ortiz-León, W. Reich, M. R. Rugel, J. S. Urquhart, B. Winkel, A. Y. Yang, H. Beuther, S. Billington, C. Carrasco-Gonzalez, T. Csengeri, C. Murugeshan, J. D. Pandian, and N. Roy, "A global view on star formation: The GLOSTAR Galactic plane survey. I. Overview and first results for the Galactic longitude range $28° < l < 36°$," *A&A*, vol. 651, p. A85, Jul. 2021.

[3] W. D. Cotton, "Simplified EVLA OTF Interferometry," *Obit Development Memo Series*, vol. 44, pp. 1–20, 2019. [Online]. Available: https://www.cv.nrao.edu/∼bcotton/ObitDoc/EVLAOTF.pdf

Fig. 3. Mosaic.py

```
#exec(open("Mosaic.py").read()) # to execute from ObitTalk
import os
exec(open("Targ.py").read())          # Target mosaic list
exec(open("PointingList.py").read())  # Pointing list, read to pointings
pointings = []
for p in data:
    pointings.append(p[0])
##############################################################################################
IStream  = 1         # Processing stream
strId    = ’S’+str(IStream)+stktrans   # Stream Id, unique, <= 5 char
restart  = False     # Restarting?
ch0      = 0         # restart first at channel ch0
minWt    = 0.35      # Minimum weight
doGPU    = False     # Use GPU?
minCh    = 1         # Minimum channel number
maxCh    = 16        # Maximum channel number
antSize  = 25.       # Antenna diameter (m)
datadisk  = 0        # disk for data (FITS) 0=CWD
fitsdir   = "./"     # FITS data directory
accumdisk = 1        # AIPS disk for accumulators  1=>RAM disk if setup
accumseq  = 10       # AIPS seq for accumulators
maxd      = 0.6      # How far from pointing center (deg) for overlap
nThreads  = 24       # How many threads
prtLv     = 1        # Diagnostic print level, 0, 1, 2

OSystem.PAllowThreads(nThreads) # enable threading
print ("Target=",targets[0][0],"restart=",restart,"ch0=",ch0,"minWt=",minWt)
OErr.PInit(err, taskLog =’Stream’+str(IStream)+’.log’)
OErr.PLog(err, OErr.Info, " Start processing "+strId);
exec(open("mosaicBasic.py").read())  # Do it
OErr.PLog(err, OErr.Info, "End processing "+strId);
OErr.printErr(err)
OSystem.Shutdown()
```