ThoughtWorks

ARTIP Documentation

(Automated Radio Telescope Imaging Pipeline)

Version 1.0

Updated on 29th Nov' 2017

Table of Contents

Getting Started **3**

Obtaining ARTIP 3

Prerequisites 3

Running ARTIP 4

Pipeline Output 4

Plotting Flagging Graphs 5

Pipeline Overview **6**

1. Pipeline Stages 7

1.1 Flux Calibration 7

1.2 Bandpass Calibration 8

1.3 Phase Calibration 9

1.4 Target Source Imaging 9

1.4.1. Continuum Imaging 9

1.4.2. Spectral Line Imaging 9

2. Flagging Algorithms 10

2.1 Angular Dispersion 10

2.2 Closure Phases 11

2.3 Window based MAD statistics 12

3. Pipeline Features 12

Configurations **13**

user_defined_flags.txt 14

casa.yml 14

pipeline.yml 15

calibration.yml 18

target_source.yml 22

imaging.yml 24

auto_flagging_config.yml 26

Pipeline stages - inputs and outputs **27**

Getting Started

Automated Radio Telescope Imaging Pipeline (ARTIP) is an end to end pipeline developed in Python using CASA and other standard libraries to perform data processing (i.e. from ingestion of raw visibility data to spectral line imaging).

Obtaining ARTIP

ARTIP releases are present at https://github.com/TWARTIP/artip/releases. Download and unzip the artip package to run the pipeline.

```
$ unzip artip.zip -d artip
```

Prerequisites

1. Install Anaconda and CASA
    a. Anaconda
        i. Install "Anaconda Python 2.7" version from https://www.continuum.io/downloads
        ii. Add conda to your PATH
    b. CASA (Common Astronomy Software Applications)
        i. Install "CASA Release 4.7.2" from https://casa.nrao.edu/download/distro
        ii. Add casa to your PATH

    Installation test

    Linux:

    ```
    $ casa --help
    $ conda --version
    ```

    OSX:

    ```
    $ casapy --help
    $ conda --version
    ```

2. Install pipeline dependencies

    ```
    $ cd artip
    $ ./setup.sh
    ```

    setup.sh will

    - Setup casa-pip for installing python modules CASA from PyPI

- Create "artip" conda environment from artip/environment.yml

Installation test

```
$ source <rc file path>
$ casa-pip -h
$ source activate artip
$ pyb --version
```

Running ARTIP

1. Default conf directory is present at <artip_path>/conf. You can either update it or create your own conf directory having same format.
2. Update the CASA path and model_path in <conf_dir_path>/casa.yml
3. Specify flags from the observation logs in "conf/user_defined_flags.txt". Flags follows format similar to CASA flagdata command with mode='list'. Below are the examples for the same :
    a. Flag antennas
    reason='**BAD_ANTENNA**' correlation='RR,LL' mode='manual' antenna='1,18' scan='1,7,2,4,6,3,5'

    b. Flag Baselines
    reason='**BAD_BASELINE**' correlation='RR,LL' mode='manual' antenna='11&19' scan='1,7,2,4,6,3,5'

    c. Flag Time
    reason='**BAD_ANTENNA_TIME**' correlation='LL' mode='manual' antenna='15' scan='1' timerange='2013/01/05/06:59:49~2013/01/05/07:00:00'

    reason='**BAD_BASELINE_TIME**' correlation='LL' mode='manual' antenna='7&8' scan='4' timerange='2013/01/05/06:59:49~2013/01/05/07:00:00'

4. Run pipeline through command line
    pyb run -P dataset="<ms_dataset_path>" -P conf="<conf_dir_path>"

    ```
    $ cd <artip_path>
    $ pyb run -P dataset="<ms_dataset_path>" -P conf="<conf_dir_path>" -P
    output="<output_dir>"
    ```

Pipeline Output

All the output artifacts like caltables, flag files, continuum and spectral line images are persisted in <output_path>/<ms_dataset_name> directory.

Plotting Flagging Graphs

Pipeline records antenna wise flags summary at different stages. After pipeline completion, user can generate flag summary plots using below scripts :

```
$ cd <artip_path>/flagged_data_summary
$ python generate_graph.py "../output/<ms_dataset_name>"
```

NOTE : Flags summary is recorded only when pipeline is run with "flag_summary: true" in <conf_dir_path>/pipeline.yml

Charts can be accessed on http://localhost:8000/chart.html

Pipeline Overview

Automated Radio Telescope Imaging Pipeline (ARTIP) is an end to end pipeline developed to perform data processing (i.e. from ingestion of raw visibility data to spectral line imaging) for the uGMRT and MeerKAT absorption line surveys. Having an automated pipeline brings Speed, Objectivity and Repeatability to the data reduction process.

ARTIP is written using standard python libraries and the CASA package to operate on measurement sets. It is modularized into 4 logical stages viz. Flux , Bandpass , Phase and Imaging allowing user to run each stage by configuring its properties. This helps users to validate the quality of output produced by each stage. Once the pipeline is tuned for one dataset, other datasets from the same observation can be run seamlessly in one go.

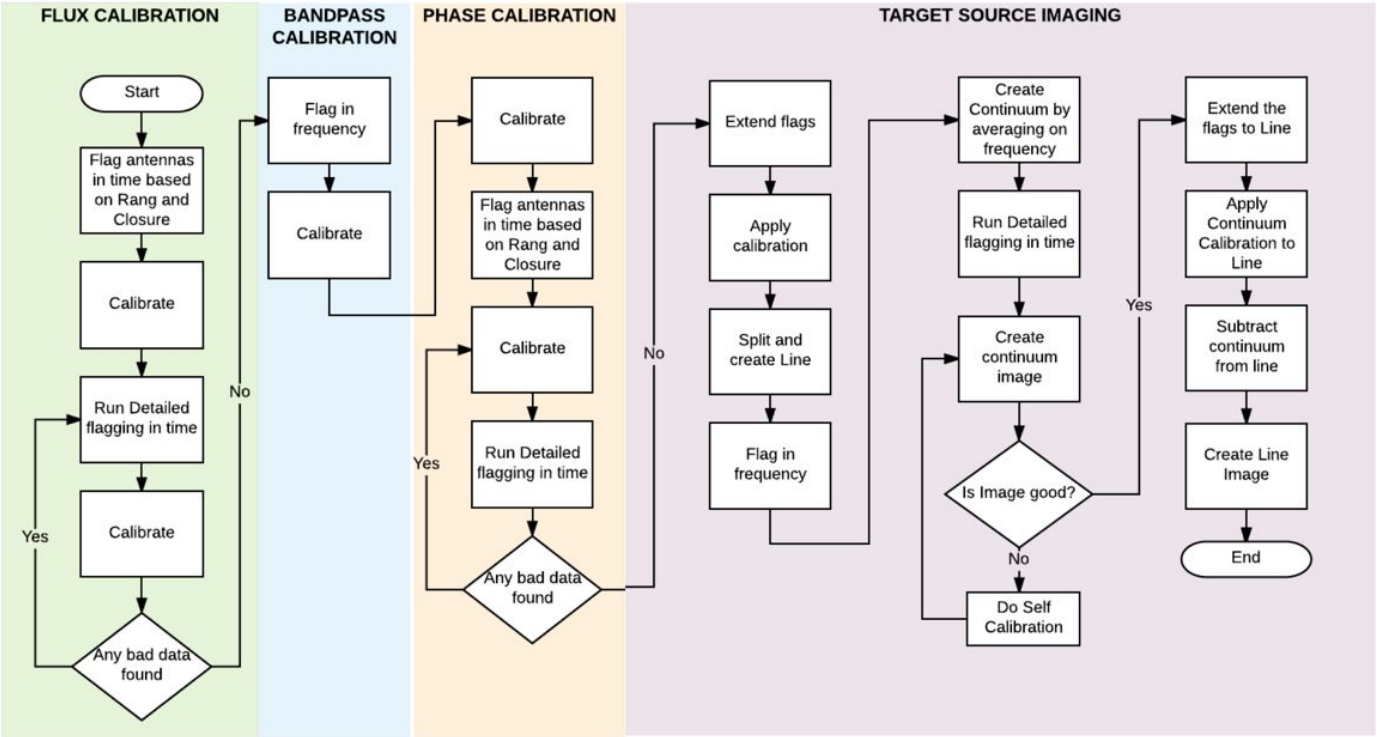
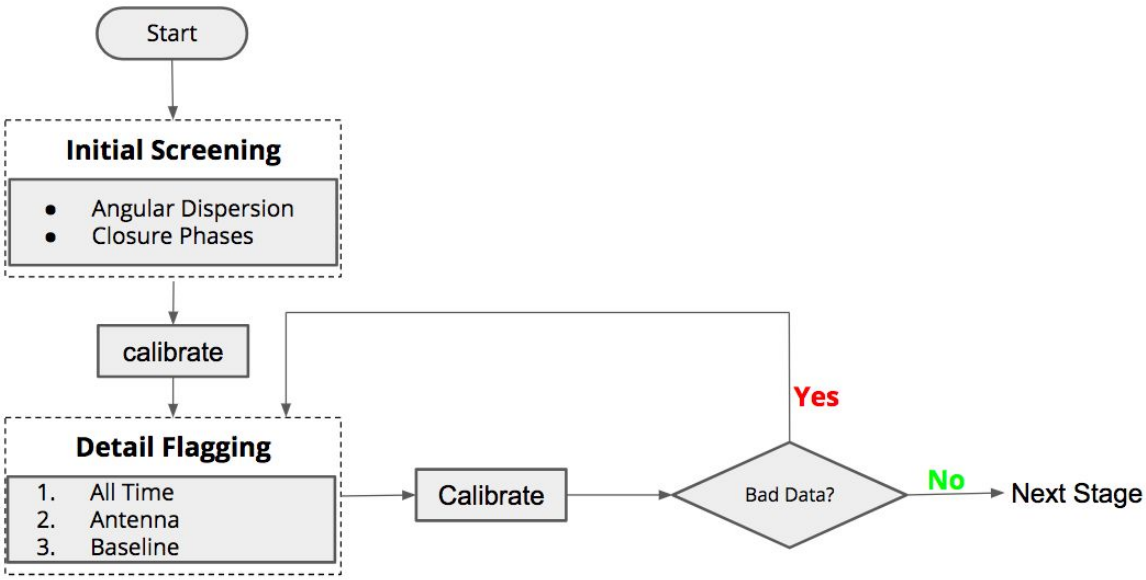


Fig. 1 Detailed flowchart of the pipeline stages and substages

1. Pipeline Stages

1.1 Flux Calibration

The main objective of flux calibration is to identify bad antennas, baselines, timeranges using directional and median absolute deviation (MAD) statistics. After excluding these bad data, the complex antenna gains are determined as a function of time.



Flux calibration

All the flagging algorithms of this stage are run on a **single channel of a reference spectral window (**which doesn't not have any RFI) **,** and the flags are extended across all the channels.

Flagging is done in two steps :

a). Initial Screening :

Initial screening uses combination of angular dispersion and closure phases (see section) to remove completely bad antennas for robustness.

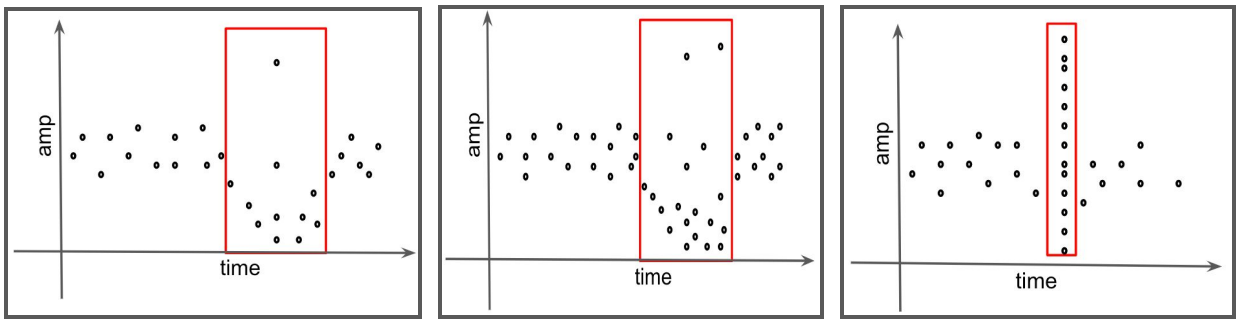
Motivation of initial screening is to remove big chunks of bad data before analysing on a granular level. Antennas which are bad in all the scans of flux calibrator are also flagged on other sources.

b). Detail Flagging

Before detailed flagging, complex gains are calculated and applied to correct the visibility data on flux calibrator. Antennas identified as good in initial screening might have bad data in time which is removed in detailed flagging. A sliding window with an overlap runs on time axis and identifies bad windows based on Median and MAD statistics. Both the size and overlap of the sliding window can be configured by the user based on the data quality. The algorithm is explained in detail in section 3.2.3

Detailed flagging also follows the strategy of removing big chunks of bad data before processing on a granular level by identifying bad data in three steps:

1). All antennas : Determines bad time across all antennas
2). Each antenna: Determines bad time for a particular antenna
3). Each baseline: Determines bad time for a particular baseline



***1).**Bad time across all antennas* ***2).**Bad time for a particular antenna* ***3).**Bad time for a particular baseline*

Each step is a continuous process of flagging and calibration which converges once all the windows in the data are good.

1.2 Bandpass Calibration

Unlike flux calibration, Bandpass calibration stage flags data in all frequencies/channels. Currently the pipeline uses Rflag and Tfcrop algorithms offered by CASA to identify the RFI.
For multiple spectral windows support, pipeline allows user to configure different flagging thresholds for each spectral window. After the flagging in frequency, bandpass calibration is done.

1.3 Phase Calibration

This stage starts by applying calibration tables from above stages on Phase calibrator. Unlike flux calibration which runs on single channel, phase calibration runs on channel averaged data due to low signal to noise ratio of the phase calibrators.

Phase calibration follows the similar procedure(Refer figure 2.1) as Flux calibration for identifying bad antennas, baselines and timeranges. After excluding these bad data, the complex antenna gains are determined as a function of time.

1.4 Target Source Imaging

Imaging is the last stage of pipeline which starts by applying calibration tables from above stages on Target source followed by generation of continuum and spectral line images.

1.4.1. Continuum Imaging

Visibility data for continuum imaging is generated by averaging target source data over channel.

Since imaging is the most time consuming step, pipeline allows user to first generate continuum image of reference spectral window. This helps in quality check of above stages.

Generation of reference spectral window continuum image starts with detail flagging(flags are extended to all spectral windows present in the visibility) followed by the process of self calibration. Self calibration uses Pixel and Threshold based mask. The number of iterations in self calibration is currently configured by the user, however it can be modified to auto converge based on RMS/Flux density in the image.

After verifying the quality of reference spectral window continuum image, pipeline generates continuum image for all spectral windows provided visibility data contains multiple SPWs. Generation of all spectral window continuum image follows the self-calibration procedure similar to reference spectral window continuum imaging.

1.4.2. Spectral Line Imaging

This step considers all channels from all spectral windows from the visibility data to generate line image.

Generation of line image starts with applying self calibration tables from continuum imaging on visibility data followed by continuum subtraction and line cube imaging.

The pipeline uses CASA's tclean task for both continuum and line imaging.

2. Flagging Algorithms

2.1 Angular Dispersion

Angular Dispersion from directional statistics is the method used in initial screening for determining completely bad antennas by calculating dispersion of phases. In this, all the antennas are analysed one by one and the value of R(angular dispersion) is calculated for all the baselines of the antenna under analysis using the following formula.

$$Y = \frac{\sum_{i=1}^{n} \sin_a}{n} \qquad X = \frac{\sum_{i=1}^{n} \cos_a}{n}$$

$$r = \sqrt{X^2 + Y^2}$$

$$\cos \bar{a} = \frac{X}{r} \qquad \sin \bar{a} = \frac{Y}{r} \qquad \theta_r = \arctan\left(\frac{\sin \bar{a}}{\cos \bar{a}}\right)$$

where X and Y are the rectangular coordinates of the mean angle, and r is the mean vector.

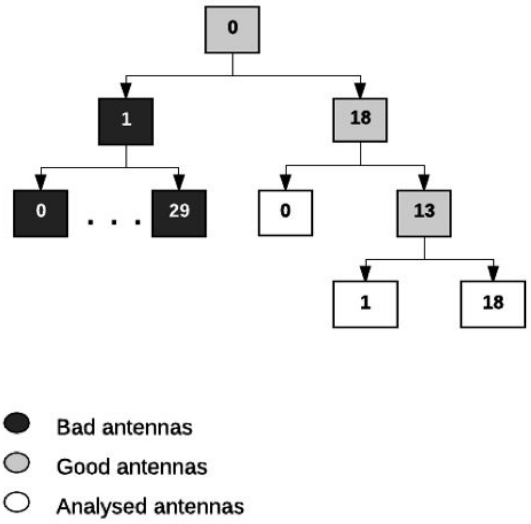
The value of R closer to 1 indicates less dispersion of phases whereas closer to 0 represents more dispersion. A threshold of the allowed dispersion is to be provided by the user. All the baselines with R value lesser than the specified threshold(R_threshold) are marked as bad. If the antenna under analysis has less than a certain percentage of good baselines (Percentage_of_good_baselines) which is configured by the user, the antenna is marked bad. In this way, a decision of good or bad can be taken for all the antennas.

The R values computed are as many as the number of baselines. As the number of antennas increase, the algorithm turns out to be less efficient. Hence the pipeline traverses the antennas in a tree by Depth First Traversal strategy.

Example:

Lets consider, threshold values to be
R_threshold: 0.3
Percentage_of_good_baselines: 70
Percentage_of_min_doubtful: 20

The traversal starts with the reference antenna (Antenna '0') which is known to be good from the observation logs. R values are calculated for all the baselines of the reference antenna. The antennas from baselines with R value lesser than 0.3 are marked as doubtful.



In the figure shown, 1 and 18 come out to be doubtful. Algorithm moves on to analyse these antennas. Since antenna 1 has less than 70% of good baselines, antenna 1 is marked bad, whereas antenna 18 has only 2 bad baselines (0,13), hence it is marked as good and the antennas which formed the bad baselines (0, 13) with it are analysed further in a similar manner. Doubtful antennas are always chosen from baselines of good antennas hence no antennas from antenna 1 baselines are marked doubtful. The antennas which are already analysed are skipped.
The traversal continues till all the leaf nodes of the tree have been visited before. In this way, the number of antennas to be traversed comes down by almost 60%.

To make algorithm more robust pipeline allows user to specify a percentage (Percentage_of_min_doubtful) ensuring that at least certain percentage of antennas are always analysed. The antennas are chosen based on lowest values of R.

For GMRT datasets with 30 antennas, we have kept it to 20% which comes to be 6 antennas to be doubted for each good antenna. The antennas which are already analysed are skipped. The traversal continues till all the leaf nodes of the tree have been visited before. In this way, the number of antennas to be traversed comes down by almost 60%.

2.2 Closure Phases

To make the method of determining bad antennas more robust, the pipeline combines results of angular dispersion with closure phases. Closure phases is a technique to find out bad antennas

based on the principle that the phase differences between baselines of any antenna triplet should be 0. But in algorithm a closure phase is considered good if it is within the range specified by user(Closure_threshold). The pipeline forms all possible triplets for the antenna under analysis and calculates closure phases for all times. A triplet is called bad if the percentage of good closure phases is lesser than the specified threshold (percentage_of_closures). Likewise, an antenna is called bad if the percentage of good triplets is lesser than the specified threshold (percentage_of_good_triplets).

Since the calculations are quite simple, this method uses brute force unlike angular dispersion which uses a tree traversal.

2.3 Window based MAD statistics

Median Absolute Deviation (MAD) statistics is used to determine if a time window is good or bad. Median and MAD are calculated for each window (local) as well as the whole scan (global). Window median and MAD are then compared against the global to determine if the window is good or bad.

A window is bad in either of the two conditions:

1. If the median is deviated
   Absolute (Local median - global median) > Sigma
   Where Sigma = global MAD * limit
2. If the amplitudes are scattered
   Local MAD > Global MAD

3. Pipeline Features

The salient features of the pipeline are

1. Pipeline supports multiple calibrators, target sources and spectral windows
2. Charts reporting flagging statistics at various stages of the pipeline are also generated.
3. Pipeline not only flags data but also maintains what data has been flagged along with the reasons.
4. Pipeline is divided into multiple stages and allows user to run each stage independently.
5. Pipeline provides continuous feedback through console logs. Logging is done at different levels which can be controlled by the user.
6. The statistical algorithms and stages of the pipeline can be controlled by the user through configurations.

Configurations

There are a number of parameters that need to be configured by the user for the pipeline to run on the desired dataset. These configurations are logically divided and organized in different files. All these files should be present (with the exact same name) in the “conf” folder passed to the pipeline from the command line. Following is the list of config files with a short description of what kind of configs it holds.

1. **user_defined_flags.txt:** This is a CASA formatted flag file for the user to specify flags that are known from the observation logs. The flags specified in this file are read by the pipeline and applied on the dataset.

2. **casa.yml:** This file contains CASA setup configs like path of CASA installation.

3. **pipeline.yml:** This is the first config file that the user needs to modify. This file contains stage toggles and configs that are globally used by the pipeline.

4. **calibration.yml:** This contains sub stage calibration toggles and configurations needed for flagging and calibration on all calibrators.

5. **target_source.yml:** This file contains sub stage toggles for target source and configs used for flagging on target source.

6. **imaging.yml:** This file is only for self calibration and imaging configs for both line and continuum

7. **autoflagging_config.yml:** This contains configs that are used for autoflagging algorithms (e.g. rflag, tfcrop) on bandpass calibrators and target sources.

user_defined_flags.txt

This file is a CASA formatted flag file. It is to specify the bad data which is known from the observation logs.

Example:
reason='BAD_ANTENNA' correlation='RR,LL' mode='manual' antenna='1,18' scan='1,7,2,4,6,3,5'

casa.yml

This file contains CASA executable details like installation path etc.

Section 1: casa

Depending on the OS you are running the pipeline on, set the ***path*** to the location of your CASAPY executable. Darwin is for MAC OS machines. Linux is for all linux OS machines.
model_path is the location of the flux calibrator model images.

pipeline.yml

Section 1: Stages

This section is for the main stages. Main stages are a list of tasks that the pipeline performs. This section is to toggle (enable/disable) these main stages. If the main stages are disabled, they are skipped from the execution flow, and the respective sub stages are not considered.
The type of toggles/configs in this section is boolean hence the values can only be **True/False.**

Key	Description
flag_known_bad_data	Apply pre known flags like quack and flags known from observation
flux_calibration	Analyses(single channel) and flags(all channels) the data on flux calibrator iteratively as a function of time followed by calibration. Also extends the flux calibrator flags on other sources if needed.
bandpass_calibration	Analyses and flags the data on flux calibrator as a function of frequency (mainly to detect and remove RFI) followed by bandpass calibration
phase_calibration	Applies flux calibration tables to the phase calibrator. Analyses(single channel) and flags(all channels) the data on phase calibrator iteratively as a function of time followed by calibration. Also extends the phase calibrator flags on the target sources if needed.
target_source	If any of the 3 main stage toggles under target_source are on, this stage would run. This stage will apply the calibration tables from above stages and create line source by splitting the target sources from the dataset
target_source-ref_continuum	This will generate a continuum visibility of the reference spw from the line source. It will create a base image followed by flagging in time and several iterations of self calibration. The flags from this continuum will then be extended to the line source.
target_source-all_spw_continuum	This stage starts with flagging in frequency on the line source. It will then generate a continuum visibility with all the spws from the line source followed by several iterations of self calibration.
target_source-all_spw_line	After the self calibration is done on all spw continuum, the cal tables will be applied to the line source and finally the line image will be generated.

Section 2: Global

This section contains configurations that are required throughout the entire pipeline.

Key	Description	Datatype	Example
polarizations	Contains a list of polarizations present in the dataset that need to be processed	List of strings	[‘RR’, ‘LL’]
flux_cal_fields	Contains a list of flux calibrator field ids as per the dataset (that need to be processed)	List of integers	[0,4]
bandpass_cal_fields	Contains a list of bandpass calibrator field ids as per the dataset (that need to be processed)	List of integers	[0,4]
phase_cal_fields	Contains a list of phase calibrator field ids as per the dataset (that need to be processed)	List of integers	[1,2]
target_src_field	Contains a list of target source field ids as per the dataset (that need to be processed)	List of integers	[3]
spw_range	Comma separated range of spectral windows present in the dataset (that need to be processed)	String	“0,1,2,3”
refant	Reference antenna to be considered for flagging algorithms and calibration	Integer	2
target_phase_src_map	This is a mapping of target source to phase calibrators.	Dictionary with Integer as key and List of Integers as value	{3:[1,2]}

Other configs

Key	Description	Possible values
code_profiling	Enable/disable the profiling of pipeline code.	True/False
flagging_percentage	Enable/disable display of percentage of flagged data at each stage	True/False
log_level	To set log level of the pipeline	10 (Debug) 20 (Info) 30 (Warning) 40 (Error) 50 (Critical)

calibration.yml

Section 1: Calibration stages

This section contains toggles for sub stages for each of the calibrators. The type of toggles/configs in this section is boolean hence the values can only be **True/False.**

Key	Sub key	Description
flux_calibration	flagging	Enables/disables all flagging algorithms from the flux calibrator. If this is disabled, only setjy and calibration is run on the flux calibrator
bandpass_calibration	run_auto_flagging	Enables/disables the auto flagging algorithms like rflag and tfcrop to detect RFI on bandpass calibrator. If this is set to false, only bandpass calibration and a redo of flux calibration is run on bandpass calibrator
phase_calibration	flagging	Enables/disables all flagging algorithms from the phase calibrator. If this is disabled, only phase calibration and fluxscale are run on the phase calibrator

Section 2: Calibration

This section contains configuration of flagging and calibration parameters required for each of the calibrators.

Section 2.1: Flux calibrator

This section contains parameters needed for all three flagging algorithms and calibration for the flux calibrator only.

Key	Description	Datatype	Example
calib_params	It contains a list of parameters needed for calibration. The list corresponds to [channel, width, minsnr, solint]. *channel* here is the reference channel used for calibration as well as flagging, *width* is the number of channels to average, *minsnr* is the signal to noise ratio used for calibration, and *solint* is the solution interval in seconds also used for calibration. The sequence of the params in this list should be maintained.	List of Integers and Floats	[100, 1, 2.0, 60]
phase_data_column	The Data column to be used for fetching phase data for flagging analysis	String	“corrected_phase”
angular_dispersion: r_threshold	This is the allowed dispersion in phases. Value closer to 0 being more strict i.e. no dispersion allowed.	Float	0.3
angular_dispersion: percentage_of_min_doubtful_antennas	Least percentage of antennas that needs to be analysed	Integer	20
angular_dispersion: percentage_of_good_antennas	Percentage of baselines that should be good for an antenna to be called as good	Integer	70
closure: threshold	Closure phase threshold in degrees above which an antenna triplet can be called as bad	Integer	17
closure: percentage_of_closures	Percentage of closure phases over time that should be good to call a triplet as a good triplet	Integer	70
closure: percentage_of_good_triplets	Percentage of triplets that should be good to call an antenna as good	Integer	70
detailed_flagging: time_sliding_window	Contains list of window configurations for **all-time** detailed flagging across all antennas. It is a list of bucket_size, overlap, mad_scale_factor in this order. Where *bucket_size* is the number of datapoints in each window, overlap is the number of datapoints to be	List of Integers	[1, 0, 3]

	slided for the next window, *mad_scale_factor* is the sigma.		
detailed_flagging: antenna_sliding_window	The parameters are same as above (time_sliding_window), except that this window spans across a single **antenna** at a time.	List of Integers	[10, 5, 3]
detailed_flagging: baseline_sliding_window	The parameters are same as above, except that this window spans across a single **baseline** at a time.	List of Integers	[10, 5, 3]
detail_flagging: amplitude_data_column	The amplitude data column to be used for fetching data for analysis of detailed flagging	String	“corrected_amplitude”

Section 2.2: Bandpass calibrator

This section contains parameters needed for calibrating the bandpass.

Key	Description	Datatype	Example
phase_calib_params	It contains a list of parameters needed for calibration of phases needed for bandpass calibration. The list corresponds to [minsnr, solint]. *minsnr* is the signal to noise ratio, and *solint* is the solution interval in seconds used for calibration of phases. The sequence of the params in this list should be maintained.	List of Integers and Floats	[2.0, 60]
bpcal_solint	This is the solution interval in seconds used for calibrating for the bandpass.	Integer	900

Section 2.3: Phase calibrator

This section contains parameters needed for all three flagging algorithms and calibration for the phase calibrator only.

Key	Description	Datatyp	Example

		e	
calib_params	It contains a list of parameters needed for calibration. The list corresponds to [channel, width, minsnr, solint]. *channel* here is the reference channel used for calibration as well as flagging, *width* is the number of channels to average, *minsnr* is the signal to noise ratio used for calibration, and *solint* is the solution interval in seconds also used for calibration. The sequence of the params in this list should be maintained.	List of Integers and Floats	[80, 40, 2.0, 60]
channels_to_avg	This is the channel/range of channels to be used for the phase calibration	String	“80~120”
phase_data_column	The phase data column to be used for fetching data for flagging analysis	String	“corrected_phase”
angular_dispersion	Same as that of section 2.1	-	-
closure	Same as that of section 2.1	-	-
detailed_flagging	Same as that of section 2.1	-	-

target_source.yml

Section 1: target_source_stages

This section contains toggles for sub stages for the target source. The type of toggles/configs in this section is boolean hence the values can only be **True/False.**

Key	Sub key	Description
calibrate		If set to True, will apply the calibration tables from flux, phase and bandpass calibrations.
ref_continuum	image	Enables/disables creation of continuum base image.
	flagging	Enables/disables detailed flagging to be run on continuum.
	selfcal	Enables/disables self calibration on the reference spw continuum
	extend_flags	Enables/disables extension of continuum flags on line source
all_spw	run_auto_flagging	Enables/disables the auto flagging algorithms like rflag and tfcrop to detect RFI on all spws on line source
	continuum - selfcal	Enables/disables self calibration on the all spw continuum
	line - apply_selfcal	Enables/disables application of self calibration tables from all spw continuum to line source
	line - image	Enables/disables creation of line image for all spws from the line source

Section 2: target_source

This section contains configurations needed for target source sub stages.

Key	Sub key	Description	Datatype	Example
ref_continuum	calib_params	It contains a list of parameters needed for flagging on the reference spw	List of Integers	[100, 1, - , -]

		continuum. The list corresponds to [channel, width, minsnr, solint]. *channel* here is the reference channel used for flagging, *width* is the number of channels to average, the other two params *minsnr* and *solint* are not applicable for target source, and hence should be filled with a *dash* (-).		
	channels_to_avg	Channels to average for creation of reference spw continuum	String	“80~100”
	channel_width	Number of output channels creating the reference continuum	Integer	41
	detailed_flagging	Same as that of section 2.1	-	-
spw_continuum	channels_to_avg	Channels to average for creating all spw continuum	String	“80~100”
	channel_width	Number of output channels for creating the all spw continuum.	Integer	41

imaging.yml

This file contains configurations specific to self calibration and imaging only for both continuum and line sources.

Section 1: imaging- default

This section contains imaging parameters that are common in continuum base image, continuum self calibrated images and line image. All the parameters in this section are directly passed to tclean. Refer to CASA's tclean documentation for description of the parameters.

Section 2: base_image

This section contains configs specific to continuum base image. It inherits all the configs from default section. User can override the default configs for base image in this section.

Section 3: cont_image

This section contains configs specific to continuum self calibrated images. It inherits all the configs from default section. User can override the default configs for self calibrated image in this section. It also contains configurations for self calibration. Refer to the table below for self_calibration params.

Key	Sub key	Description	Datatype	Example
minsnr		Used for self calibration of continuum	List of Integers	[100, 1, - , -]
masking	threshold	This is for threshold masking. Any points with value lesser than this will be excluded from the image mask	String	'0.0'
	mask_path	Custom mask path to be used for masking	String	
	bmask	This specifies the box coordinates for the box masking. The user should define the bottom left and top right corners appropriately	Comma separated String	'240,240,270,270'
	calmode	Self calibration runs for both 'p' and 'ap' mode. Loop count is the number of iterations of self		

		calibration.		

Section 3: line_image

This section contains configs specific to continuum self calibrated images. It inherits all the configs from default section. User can override the default configs for self calibrated image in this section. It also contains configurations for self calibration. Refer to the table below for self_calibration params.

auto_flagging_config.yml

This file contains configurations specific to auto flagging for both bandpass calibrator and line source.

Section 1: auto_flagging_algo

This section contains RFI flagging parameters that are common in bandpass calibrator and line source. Refer to CASA's flagdata (mode = rflag/tfcrop) documentation for description of the parameters.

Section 2: bandpass_calibrator: auto_flagging_algo

This section contains configs per spw. Each spw config inherits from the above auto_flagging_algo section and overrides parameters that differ from the above section.

Section 2: line: auto_flagging_algo

This section contains configs per spw. Each spw config inherits from the above auto_flagging_algo section and overrides parameters that differ from the above section.

Pipeline stages - inputs and outputs

Input location/Configurations: Various stages and parameters of the pipeline can be configured by providing a configuration folder from the command line through *"conf"* parameter. The configurations should be in the format as specified in the default conf.

Output location: The output location can be configured by providing an output folder from the command line through *"output"* parameter .
A folder (having dataset name) is created inside the *output* location so that each dataset will have its own output folder. All the output artifacts of the pipeline generated at various stages are persisted in this folder.

Stage	Sub stage	Input used	Output
Flag known bad data		conf/user_defined_flags.txt	Flags defined in input file are written on the given measurement set
Flux calibration	Flagging	Given measurement set	flux_calibrator_flags.txt
	Calibration	Given measurement set	intphase.gcal, amp.gcal
Bandpass calibration	Run auto flagging	Given measurement set	Calculates and writes RFI flags to the input measurement set
	Bandpass calibration	Given measurement set	bpphase.gcal, bandpass.bcal,
	Flux calibration	Given measurement set	intphase2.gcal, amp2.gcal
Phase calibration	Flagging	Given measurement set	phase_calibrator_flags.txt
	Calibration	Given measurement set, intphase2.gcal, amp2.gcal, bandpass.bcal	scanphase.gcal
	Flux scale	Given measurement set,	flux.gcal

		amp2.gcal	
Target source	Calibrate	Given measurement set, flux.gcal, bandpass.bcal, scanphase.gcal	Given measurement is corrected
	Line source	Given measurement set	line_{id*}/line_{id*}.ms
Target source-Reference spw continuum	Visibility	line_{id*}/line_{id*}.ms	continuum_ref_{id*}/continuum_ref_{id*}.ms
	Image	continuum_ref_{id*}/continuum_ref_{id*}.ms	continuum_ref_{id*}/cont_base_image
	Flagging	continuum_ref_{id*}/continuum_ref_{id*}.ms	flags_continuum.txt
	Self Calibration	continuum_ref_{id*}/continuum_ref_{id*}.ms	self_caled_ap_ref_{id*}, self_caled_p_ref_{id*}
	Extend flags	line_{id*}/line_{id*}.ms, flags_continuum.txt	Applies input flag file on input line ms.
Target source-All spw	Run auto flagging	line_{id*}/line_{id*}.ms	Calculates and writes RFI flags to the input measurement set
Target source-All spw Continuum	Visibility	line_{id*}/line_{id*}.ms	continuum_all_spw_{id*}.ms
	Self calibration	continuum_all_spw_{id*}.ms,	self_caled_ap_all_spw_{id*}, self_caled_p_all_spw_{id*}
Target source-All spw Line	Apply self calibration	line_{id*}/line_{id*}.ms, self_caled_ap_all_spw{id}/ap_selfcaltable_{lc^}.gcal, self_caled_p_all_spw{id}/p_selfcaltable_{lc^}.gcal	Applies input gcal tables on input line ms
	Image	line_{id*}/line_{id*}.ms	line_{id*}/line.image

Footnotes:

id* refers to the target source id.

lc^ stands for the self calibration loop count specified in the config.

Other outputs in the output directory:

casa.log:

This file is a CASA generated log file. It contains logs of all the casa tasks that are called from the pipeline with time stamps.

Note: This file isn't overridden on subsequent pipeline runs. The logs are appended to the file. If the user wishes to run the pipeline again, the log history needs to be maintained by the users themselves.

casa_output.txt

This file contains the CASA stdout dump. Whatever CASA prints on the console instead of log file goes in this output file.

Note: This file isn't overridden on subsequent pipeline runs. The stdout is appended to the file. If the user wishes to run the pipeline again, the output history needs to be maintained by the users themselves.