ThoughtWorks

ARTIP Documentation

(Automated Radio Telescope Imaging Pipeline)

Version 1.0

Updated on 26 Oct' 2017

Table of Contents

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).

Prerequisites

1. Anaconda
    a. Install "Anaconda Python 2.7" version from https://www.continuum.io/downloads
    b. Add conda to your PATH
2. CASA (Common Astronomy Software Applications)
    a. Install "CASA Release 4.7.2" from https://casa.nrao.edu/download/distro
    b. Add casa to your PATH
    c. Update path and model_path in artip/conf/casa.yml

Prerequisites test

Linux:

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

OSX:

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

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
```

Run ARTIP

1. Update config files present in "conf/" directory.
2. 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'

3. Run pipeline through command line
    pyb run -P dataset="<ms_dataset_path>" -P conf="<conf_dir_path>"
    $ cd <artip_path>
    $ pyb run -P dataset="~/Downloads/may14.ms" -P conf="conf"

Generate Flagging Charts

Flags are recorded only when pipeline is run with "flag_summary: true" in conf/pipeline.yml
To generate charts :

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

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

Pipeline Overview

ARTIP is built in python and uses CASA to operate on measurement sets. It also uses anaconda libraries[casa-data and casa-python from pkgw-forge channel] for accessing data from measurement set. The pipeline is modularized by breaking it down into logical stages, which helps user to run any stage in segregation by checking the quality and artifacts produced by the previous stages, or to change any configuration parameters depending on those results. The user also has a choice to run the entire pipeline in one go. The pipeline is also made highly configurable allowing users to change the properties or thresholds on any stage of the algorithm depending on the data quality. The stages of the pipeline are explained in detail in the following subsections.

1.1 Pipeline Stages

The pipeline has four main stages viz. Flux calibration, Bandpass calibration, Phase calibration and Target source imaging.

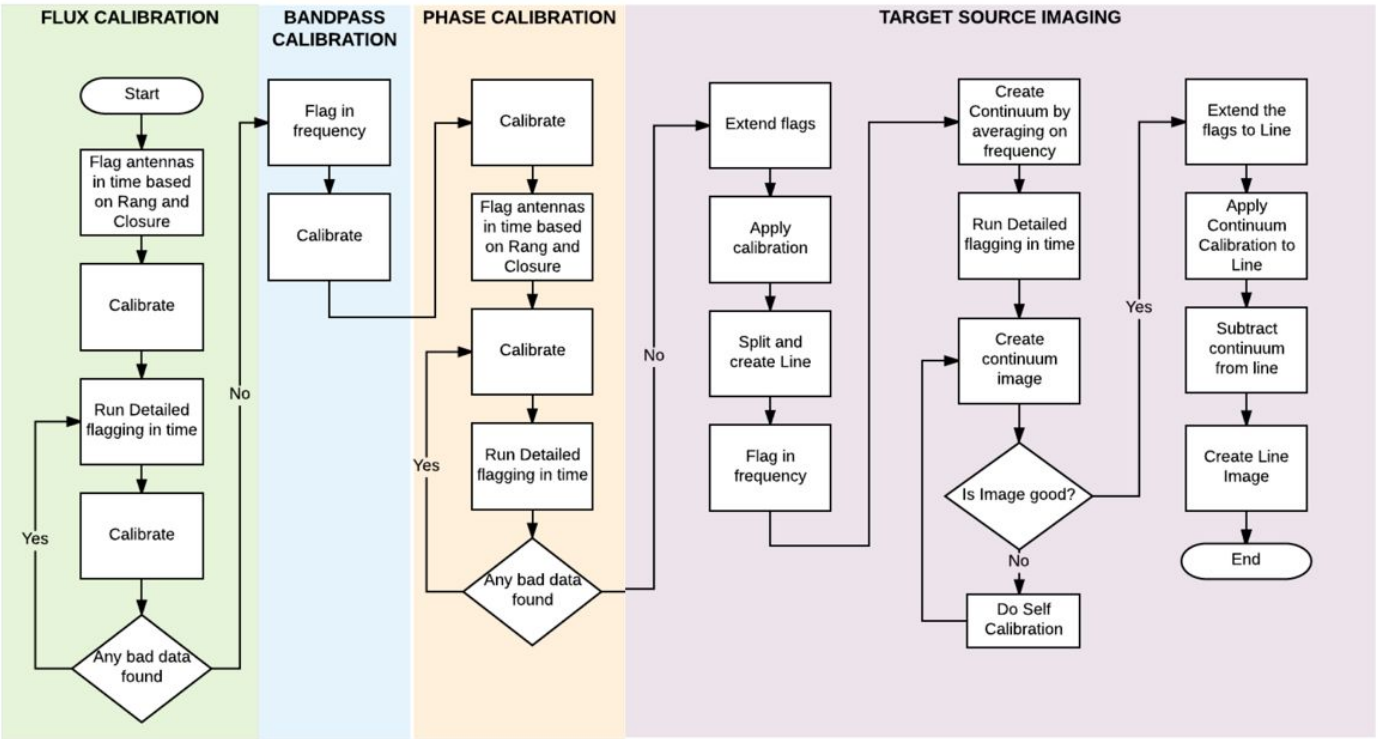
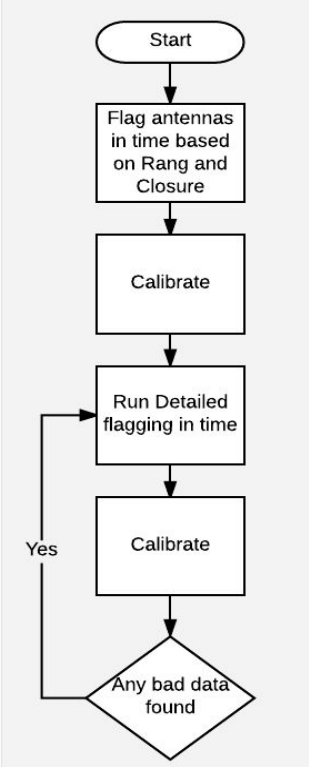


Fig. 1 Detailed flowchart of the pipeline stages and substages

1.1.1 Flux Calibration

The main objective of doing the flux calibration is to flag the bad data and get the gain solutions in flux to be applied on the target source.



The flagging is done in two steps. The first step is initial screening of antennas where we remove antennas that are completely bad. The motive here is to remove big chunks of bad data quickly without much processing and later analyse on a granular level. We use two algorithms for the initial screening viz. 1. Angular dispersion for calculating the dispersion of phases and 2. Closure phases. Both the algorithms are described in detail in section 2.2. The results of both the algorithms are combined to make the initial screening of antennas more robust.

Only the antennas that are found bad in both the above algorithms are removed. The flags are then extended to other sources. The antennas that are found bad in all the scans of a flux calibrator are removed from all the sources as a part of this extension. Flux calibration is done after the initial screening to improve the data quality before further processing.

As the first step removed completely bad antennas, what we are left with now are possible good antennas. These antennas might have data which is bad in time. Hence the second step is detailed flagging which works on calibrated amplitude data to remove bad data in time by dividing the data in time windows and analysing each window. Median Absolute Deviation (MAD) statistics are used to determine if a time window is good or bad. The algorithm is explained in detail with example in section 3.2.3

All the bad windows are determined and flagged in this step. The detailed flagging runs in a loop. After the windows are flagged, the data is calibrated and again the detailed flagging is done. As the data quality improves, the Median and MAD statistics improve and more bad data if present stands out clearly. The cycle continues until we keep finding bad data. The pipeline moves to the next stage once all the data is good.

The detailed flagging is done thrice. Once for the time, to find out if some time was bad across all the antennas and baselines, second for antennas and third time for baselines. The algorithm is run for antennas before baselines because if all the baselines of the antenna are bad for a particular time, flagging of the whole antenna window is quicker and more logical rather than nitpicking that in baselines.

Note: Both the above algorithms run on a **single channel,** and the flags are extended across all the channels.

1.1.2 Bandpass Calibration

After the flux calibration, the data is flagged in frequency. Currently the pipeline uses Rflag and Tfcrop algorithms offered by CASA to identify the RFI. After the flagging in frequency, bandpass calibration is done.

1.1.3 Phase Calibration

The third stage of the pipeline that is phase calibration runs on channel averaged data due to low signal to noise ratio of the phase calibrators being weak sources. Also the least doubt percentage threshold for initial flagging is kept lenient (i.e. low, 10% for GMRT dataset) as completely bad antennas are already identified and removed by extending flags from flux calibrator.

After running the above stages, statistics showed that around 12% of data was flagged from the entire dataset among which around 45% was flagged through initial screening i.e. angular dispersion and closure phases, 21% in detailed flagging, 12% in rflag and tfcrop and remaining 22% was flagged due to quack.

1.1.4 Target Source Imaging

Finally, the last step is to apply the flags and calibration solutions from previous step on the target source and generate images. The pipeline can generate both continuum and line images. It is designed such that one can create continuum image only for the reference spectral window first, do a quality check and then if it is as desired, one can continue creating continuum images for all spws and line cube.

For creating the continuum image, target source data is averaged by channels and detailed flagging is run in time on the channel averaged data with a single loop followed by self calibration. To improve self calibration, masking is done based on both flux density and shape. The number of iterations in self calibration is currently configured by the user, however it can be modified in future to auto converge based on RMS in the image. Once the self calibration is done, the flags and calibration tables from continuum are then applied to the line cube to generate the final line cube.

1.2. Flagging Algorithms

1.2.1 Angular Dispersion

Angular dispersion is the method from directional statistics which is being used to find the dispersion. In the method, the value of R(angular dispersion) is calculated for each baseline of the antenna being analysed by the following formula.

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

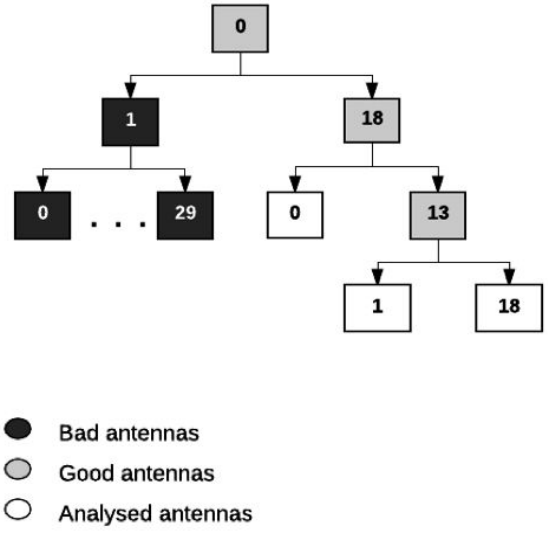
$$r = \sqrt{X^2 + Y^2}$$

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

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 hence a bad baseline. If the antenna under analysis has many such bad baselines, the antenna is marked as bad. The percentage of bad baselines required to call an antenna bad can be configured by the observer as per the data quality. 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 number of R values to be computed also increases. This leads to a lot of processing which can be easily optimized by analysing the antennas efficiently. The pipeline traverses the antennas in a tree by the Depth First Search strategy.



The traversal starts with the reference antenna which is known to be good from the observation log. R values are calculated for all the baselines of the reference antenna.
The antennas which form the baselines with R values below the specified threshold are marked as doubtful. In the figure shown, 18 and 1

come out to be doubtful. Now, antennas 18 and 1 are analysed. Since antenna 1 has more than 70% of the baselines bad, antenna 1 is marked bad, whereas antenna 18 has only 2 bad baselines, hence it is marked as good and the antennas which formed the bad baselines with 18 are analysed further. The 70% here is the percentage threshold of minimum number of good baselines that should be found in order to mark an antenna as good and it can be configured depending on data quality. Also, there is a provision to configure the least doubt percentage. Least doubt percentage is the percentage of antennas one would like to analyse for each good antenna found. For GMRT dataset 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%.

1.2.2 Closure Phases

Closure phases is also a technique to find out bad antennas based on the principle that the phase differences between baselines of any antenna triplet should be 0. If not, one or more antennas from the triplet might be bad. The pipeline forms all possible triplets for the antenna under analysis and computes the closure phases to take the decision. Brute force is done here unlike angular dispersion which uses a tree traversal since the calculations are quite simple and quick.

1.2.3 Window based MAD statistics

Median Absolute Deviation (MAD) statistic 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). Local 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

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]
output_path	Location/path to put in output artifacts like cal tables, images etc	String	“~/artip_outputs/”
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	List of	[100, 1,

		for flagging on the reference spw 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* (-).	Integers	- , -]
	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 is configurable by the user in the global config. A new folder with the name same as that of the dataset is created inside this *output* location and all the output artifacts of the pipeline at various stages are generated in this new 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, amp2.gcal	flux.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.