Documentation for the Radboud PIpeline for the Calibration of high Angular Resolution Data (`rPICARD`)

Michael Janssen

January 26, 2021

Contents

1 Preface

This documentation is a manual for the pipeline. It is not meant be a guide that explains the methods and intricacies of radio astronomy. If the reader needs more background information, I recommend two references: ‘Interferometry and Synthesis in Radio Astronomy’ by A. Richard Thompson, James M. Moran, and George W. Swenson Jr. (the third edition is available online for free) and ‘Synthesis Imaging in Radio Astronomy II’, edited by G. B. Taylor, C. L. Carilli, and R. A. Perley (volume 180 of Astronomical Society of the Pacific Conference Series. 1999). Additionally, the https://ui.adsabs.harvard.edu/abs/2019A%26A...626A..75J paper can be consulted for a formal documentation of the pipeline, explaining calibration choice made in more detail and providing more information about the basic principles of VLBI data reduction.

This pipeline makes use of the CASA framework for radio astronomy data processing. The purpose of the pipeline is to calibrate very long baseline interferometry (VLBI) data from many different arrays. This became possible within CASA, thanks to a project initiated by BlackHoleCam. This project led to the addition of new data reduction tasks to the CASA software package, which are crucial for the processing of VLBI data.[1] These tasks were developed by Ilse van Bemmel, Mark Kettenis, and Des Small at JIVE.

[1]The most important new task is `fringefit()`, which is available since CASA 5.3.

2 The Basic Code Philosophy

1. No parameter is hard-coded. The user can fine-tune every knob to achieve an optimal calibration.

2. All parameters have sensible or self-tuning default values. It will suffice to set only a few basic parameters (path to the data and the names of the observed science target and calibrators) to achieve a good calibration completely hands-free.

3. Every step of the pipeline will produce diagnostic plots by default, which show if the calibration worked as intended. The plots plots are saved in special directories – it will be evident when the pipeline was run and which set of input parameters were used.

4. Every step of the pipeline can be re-run separately and without any effort for optimization (e.g., after trying different input parameters or after manually editing bad data).

Following these points, experienced users can easily customize the pipeline to their needs, while inexperienced users will still get good results when running the pipeline 'blindly' with the default settings.[2] Thanks to the many diagnostics of the pipeline, users can easily understand how and if the calibration steps worked. If the diagnostics show that a certain step did not work properly, then it can be re-run indefinite times without effort, allowing the user to optimize that step and fine-tune the calibration procedure.

`rPICARD` is a highly modular code, with a standardized wrapper for each calibration task. The wrapper automatically takes care of potential smoothing and plotting of calibration solutions as set by the user via a set of standardized input parameters, on-the-fly calibration, and the passing of any required complementary metadata from the Measurement Set (e.g., frequency, antenna, and scan information), in single objects. This makes it straightforward to adjust any calibration strategy or to add new functionalities, see section 20.

[2]If there are no severe issues with the dataset that need to be dealt with in a special way, the pipeline should produce perfectly calibrated data when using the default settings – thanks to self-tuning parameters and the automatic flagging routines.

3 Code Overview

The code is organized into the main script `picard/main_picard.py`, which handles the sequential calls to the different CASA calibration tasks, and several modules in the `picard/pipe_modules` folder: For phase calibration, amplitude calibration, polarization calibration, diagnostics, opacity corrections, automatic flagging, auxiliary functionalities, and generic functionalities around the CASA calibration framework. Additional scripts from JIVE for the amplitude calibration can be found in `picard/pipe_modules/JIVE_scripts`.

`picard/picard` is a wrapper around `picard/main_picard.py`, handling a proper call to *mpicasa -c*. `setup.py` must be used to link the pipeline to your CASA installation (saving the PATH to your CASA binary in a `your_casapath.txt` file). The setup script will also establish the mpi environment for your machine and use the `input_template` folder (section 7) to write a first set of default input parameters to `picard/input`. For the docker images, `setup.py` was already executed automatically.

By default, `setup.py` will search your whole file system for a CASA installation. Passing *-d <path>* as command line argument will narrow down the search to the directory <`path`> (useful if CASA is installed on a network disk or if the local file system is very big).

`picard/interactive_utils.py` is a module that can be loaded into interactive CASA sessions independently from the rest of the pipeline (if added to your PYTHONPATH). See section 10.

`picard/scripts/calibration` contains example scripts where `rPICARD` is run to calibrate data and `picard/scripts/imaging` contains scripts where `rPICARD` is run to image calibrated data.

4 MPI Scalability

The pipeline makes use of the built-in MPI scalability of CASA. Some tasks (like `applycal()` or `flagdata()` for example) are internally parallelized and will run faster if more CPU power is available.

Fringe-fitting is the most time-consuming task. It is done on a per-scan basis; for speed-up each scan will be fringe-fitted separately as a separate MPI job.

The `setup.py` script will prepare a default `picard/input/mpi_host_file`, filling it with the maximum number of cores available on the host machine. By default, the call to *mpicasa* in `picard/picard` reads from this mpi host file via *-hostfile ${pipedir}input/mpi_host_file*. However, if a custom `mpi_host_file` exists in a local input folder (section 7) that the code is told to use via the *--input* or *-p* command line arguments (section 8), the computing resources in that host file are used instead.

A `mpi_host_file` simply contains the number of cores that are to be used per machine. I.e., for a single machine, a single line is given with the name of the machine and the number of cores ('slots') that CASA is allowed to use. For an MPI-able computing cluster, multiple computing nodes/machines can be used by specifying the name of each node/machine together with the number of available cores on a separate line. Another option is to use the *-n* command line option (section 8) to specify the number of cores that are to be used.

4.1 Memory Safeguard

For datasets with many visibilities per scan (many baselines, long scans, large bandwidth), it can happen that the parallelized `applycal()` and `fringefit()` steps (section 6) cause memory starvation if the number of used CPU cores times the amount of memory needed to calibrate a single scan[3] is larger than the system's total amount of available memory. If `rPICARD` is run on a system with many cores but not a lot of RAM, the `mpi_memory_safety` parameter in `picard/input/constants.inp` can be set to limit the number of scans that are simultaneously calibrated by `applycal()` and to wait until enough memory is available before submitting a `fringefit()` job to an MPI server. The overhead of this memory monitoring and resource scheduling will decrease the performance a little bit.

[3]Different scans can contain different numbers of visibilities.

5 Quick Start Guide

This section provides instructions on how to run the pipeline on the provided example EVN fits-idi file in the `testing` folder. The purpose of doing this test is twofold:

- Test if the pipeline works as intended on your machine.
- Follow the typical actions that need to be taken when reducing a dataset with the pipeline:

1. Pick a location on your file system where you want to work with your data (`testing/`).
2. Prepare a set of input parameters (section 7), starting from the `picard/input` file in the pipeline folder (assuming *$ ./picard/setup.py* was done already). It is recommended to have the input folder always together with the data itself. So, copy the `picard/input/*.inp` files to a new input folder in your workdir (`testing/input/`). There, edit the `observation.inp` file, setting the absolute path for the working directory. For example, `workdir = ∼/Software/picard/testing` would be a typical input. Or use `workdir = $pwd` to set the working directory to `</path/to/input/folder>/../` (which is specified with the -- *input* command line argument, see below).
3. Your uv-data must be present in the working directory either as fits-idi files (or links to these files) or as measurement set. In `testing/`, a single example EVN fits-idi file is present.
4. Also, all useful metadata should be put in the workdir (or links to these files), see section 14. An ANTAB table (section 12) must be present for the amplitude calibration. In `testing/`, there is an ANTAB table `example.antab`, a uvflag file `example.flag`, a source model file `3C84.smodel` (section 17), and a file with receiver temperature information `example.trx` (section 16).[4]
5. The fits-idi file in `testing/` is an unknown dataset, so we first want to know what is in there before setting any other input parameters. So, we run the pipeline in 'inspection mode' (just loading the data and writing a listobs file, see section 8): *$ picard --input testing/input -l e*. For the test case you should see something similar to listing 1 below.
6. Inspect the diagnostics folder (section 13) from the previous action, which was created in the workdir and read the `list.obs` file that was created to get an overview of the observation: Pick the best reference antenna and note down science targets and calibrators.
7. Edit `testing/input/observation.inp`, defining science targets and calibrators and `testing/input/array.inp`, defining `array_type` and `refant`. For the test case, set `array_type = EVN ; refant = ON ; science_target = None` and all `calibrators = 3C84`.

[4]No opacity correction is done for EVN data. The Trx file is just a dummy example here.

8. Make a dry run (section 9), while also making an initial flag version backup (section 19) and without creating a new diagnostics folder this time (section 8): *$ picard --input testing/input -q -r*. For the test case you should see something similar to the listing 2 below.

9. For a real dataset, you could now perform the the complete calibration: *$ picard --input <path-to-input-files>*. If you know your data beforehand and the pipeline has been verified to work on your system, you can skip actions 5,6, and 8.

Listing 1: Typical terminal output for action 5 when using the example data in `testing/`

```
=========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
=========================================


=========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
=========================================


=========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
=========================================


=========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
=========================================

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications


-----------------------------------------------------------------------
        ......................................................
```

```
                    Thank you for using rPICARD v0.2.0-9-g8758686


            This program will automatically calibrate your VLBI datset
                                 - Make it so! -
            ******************************************************




Got --input option:
Reading input from
../testing/input/



Reading input...
  Found 174 parameters
Done



Changing directories from
/home/michael/Software/Bitbucket_repos/Picard
to
/home/michael/Software/Bitbucket_repos/testing/input/../



Writing this run's diagnostics to
/home/michael/Software/Bitbucket_repos/testing/diagnostics_2018-06-28_13-11-25

Warning: No valid WX table found.
Any opacity attenuation calibration will fail.



Loading the data...
  Found
    [/home/michael/Software/Bitbucket_repos/testing/input/../example_EVN.IDI1]
Done



Saving initial flag version to
  VLBI.ms.flagversions/flags.init_flagversion   ...
Done



Writing listobs file to diagnostics_2018-06-28_13-11-25/list.obs...
Done



Got -l e option:
Exiting now that I have written a listobs file.



Changing directories from
/home/michael/Software/Bitbucket_repos/testing
to
/home/michael/Software/Bitbucket_repos/Picard



---------------------------------------------------------------------------
                    .........................................

                                   - FINISHED -
```

```
                         Now: Tea. Earl Grey. Hot.
                  Waiting for the MPI environment wrap-up...
                  ****************************************
```

Listing 2: Typical terminal output for action 8 when using the example data in `testing/`

```

========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
========================================




========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
========================================

========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
========================================


========================================
The start-up time of CASA may vary
depending on whether the shared libraries
are cached or not.
========================================

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

IPython 5.1.0 -- An enhanced Interactive Python.

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications

CASA 5.3.0-122   -- Common Astronomy Software Applications


----------------------------------------------------------------------------
         .........................................................

                  Thank you for using rPICARD v0.2.0-9-g8758686


           This program will automatically calibrate your VLBI datset
                              - Make it so! -
           ******************************************************

```

```
Got --input option:
Reading input from
../testing/input/


Reading input...
  Found 174 parameters
Done


Changing directories from
/home/michael/Software/Bitbucket_repos/Picard
to
/home/michael/Software/Bitbucket_repos/testing/input/../


Writing this run's diagnostics to
/home/michael/Software/Bitbucket_repos/testing/diagnostics_2018-06-28_13-15-19

Got -r option:
Will enfore to (re)store an initial flag version of the data.


Attaching Tsys values from
  /home/michael/Software/Bitbucket_repos/testing/input/../example.antab
to the (first) fits-idi file:
  /home/michael/Software/Bitbucket_repos/testing/input/../example_EVN.IDI1 ...
Done


Warning: No valid WX table found.
Any opacity attenuation calibration will fail.


Loading the data...
  The measurement set VLBI.ms already exists.
    I assume you want to work with the same measurement set again,
    but probably with a different calibration strategy.
    Therefore, I will not load the fits-idi files again and keep the old MS.
Done


Restoring flags to initial version from
  VLBI.ms.flagversions/flags.init_flagversion...
Done


Writing listobs file to diagnostics_2018-06-28_13-15-19/list.obs...
Done


 -- Determining metadata collection from scratch --

Initializing metadata collection...
  Warning: There is a difference between all sources available
  and the ones specified as calibrators and science targets:
    set(['TXCAM', '3C273'])
  This either means you are not using all available sources
  or you specified sources which were not observed.
  Maybe due to a typo?
  In the former case the sources not specified will not be properly calibrated.
  In the latter case the code should exit with a ValueError now.
Done

  Using the following scans of the following sources for
  diagnostic plots, flagging, and maybe for trial and error fringe-fitting:
    [[('3C84', '1')],
    []]

Preparing DPFU and gain curve conversion file based on the ANTAB table
  /home/michael/Software/Bitbucket_repos/testing/input/../example.antab ...
Done


The pipeline will execute the following steps for the
EVN array in the given order:
   a : load models of observed sources (if present)
   b : use online flags from idi files (if present)
   c : use flags from metadata (if present)
   d : flag based on outlier detection from auto-correlations vs time
   e : flag based on outlier detection from auto-correlations vs frequency
   f : flag edge channels
   0 : task_tsys
   1 : task_scalar_bandpass
   2 : task_gaincurve
   3 : task_fringefit_solint_cal
   4 : task_fringefit_single
   5 : task_fringefit_multi_cal
   6 : task_complex_bandpass
   7 : task_fringefit_solint_sci
   8 : task_fringefit_multi_sci
   9 : task_rldelay
  10 : task_rlphase
  11 : task_dterms
   g : clear the calibrated data column of the MS from previous applycal runs
   h : apply all existing tables from all_calibration_steps
   i : print overview of flagged data (can be slow)
   j : make diagnostic plots of calibrated visibilities for selected baselines
   k : average and export the calibrated data
Can use quickmode [-q] to execute only a subset of these steps.

Got -q argument without a list specified:
Exiting now that I have shown the list of steps.

Changing directories from
/home/michael/Software/Bitbucket_repos/testing
to
/home/michael/Software/Bitbucket_repos/Picard



-----------------------------------------------------------------------------
                  ..............................................

                              - FINISHED -
                        Now: Tea. Earl Grey. Hot.
                   Waiting for the MPI environment wrap-up...
                   ****************************************
```

5.1 Usage Examples

This subsection gives some examples of things you can do with the pipeline. Here, I assume that you have followed the above steps and that you are in a working directory, where the data, metadata, and an input folder are present. For an explanation of the command line arguments, see section 8.

- Only load the data and create an initial flag version:
  $ picard -prq

- Make plots of uncalibrated and unflagged data, without creating a new diagnostics folder:
  $ picard -pdrq h,i,k

- Print overview of flagged data:
  $ picard -pq j

- Run flagging steps to see how much data they flag:
  $ picard -prq b,c,d,e,f,g,i

- Undo all flags, calibrate step 0, and make plots:
  $ picard -prq 0,h,i,k

- Full run of pipeline:
  $ picard -p

- Reload metadata (necessary after changing the source selection in the `input/observation.inp` input file for example) without creating a new diagnostics folder:
  $ picard -pmdq

- Determine fringe solints again (assuming it is step 4 here, this can be different for different array types set in the `input/array.inp` input file):
  $ picard -pfq 4

- Full run of pipeline excluding step 5 (if the calibration table still exists from an earlier run it will be applied nonetheless; in that case it may be useful to first delete the calibration table):
  $ picard -prq x,0∼4,6∼99

- Interactive imaging of a VLBI.ms.avg dataset created by `rPICARD` in the current working directory. The source is 3C279:
  $ mpicasa -n 3 <path-to/bin/casa>
  import interactive_utils
  interactive_utils.imager('3C279')

- Interactive imaging of a EVN.uvfits file in the current working directory. Perform an additional correction for the parallactic angle and allow the user to turn off interactive imaging once a final set of CLEAN boxes are in place. The source is 3C273:
  $ mpicasa -n 3 <path-to/bin/casa>
  import interactive_utils
  interactive_utils.imager('3C273', 'EVN.uvfits', parang=True, goautomated=True)

6 Overview of the Default Pipeline Steps

Below, a brief overview of the basic pipeline procedures is given. For a more detailed documentation, see section 24. Actions which will potentially flag data are underlined.

- **Preparation procedures**. These are executed every time the pipeline is run (except if the *-h* command line argument (section 8) is used.
  - Start the MPI client.
  - Read all input parameters from `picard/input/*.inp` (unless overwritten) and store them in the `inp_params` object, see section 7.
  - Change directory to the workdir specified in `picard/input/observation.inp`.
  - Make a diagnostics folder with the current timestamp, see section 13. Can be turned off with the *-d* command line argument, see section 8.
  - Write out the used input parameters in the diagnostics folder.
  - Grab an ANTAB table in the workdir folder (searched for by file extensions, see section 14). This step can be skipped for phase-only calibration, see the example in section 9.
  - Make a VLA-type gain curve table from the gain block of the ANTAB table. Skipped if no ANTAB table is present.
  - Get all fits-idi files that are to be loaded from all files with certain extensions (section 14) in the workdir folder. Use the tsys block in the ANTAB table to attach a `SYSTEM_TEMPERATURE` table to the first fits-idi file and load in all files to get a single measurement set[5] if there is enough free space. Exits otherwise. This step is skipped if a measurement set is already present.
  - Make an initial backup flagversion. Can fall back to that version if the pipeline is run again (section 19).
  - Save the output of `listobs()` to a file in the diagnostics folder. Exit here if *-l e* was passed as command line argument.
  - Determine a set of MS metadata (either from scratch or by reading a previously determined one from disk). This set contains info about the correlations, antennas, scans, and frequency setup. Also, a few selected scans are determined for every source. These are as uniformly distributed over the observations while scans with the most antennas present are picked. These scans are useful for diagnostics.
  - Print a quick overview of all steps and exit here in the case of a dry run (*-q* command line option without a list specified, see section 8).

[5]Will flag visibilities which have a weight of zero.

- **Run steps part 1.** These are executed before the actual calibration tasks. Quickmode (-q) can be used to perform only a subset of these steps. More information on the underlined flagging steps can be found in section 18.

  (a) Look for model files of the observed sources (searched for by file extensions, see section 14) and load them into the `MODEL_DATA` column of the measurement set as described in section 17.

  (b) Apply flag tables attached to the fits-idi files.

  (c) Look for files with certain flag extensions (section 14) in the workdir folder and apply the flags stored in these files (if necessary convert uvflag to casa flag format).

  (d) Automatic flagging algorithm based on outliers in the autocorrelation vs time data.

  (e) Automatic flagging algorithm based on outliers in the autocorrelation vs frequency data.

  (f) Flag edge channels.

  (g) Flag start and end segments of scans (quacking).

- **Calibration tasks.** Perform incremental on-the-fly calibration (for every task, the calibration solutions from all previous steps are applied on the fly where applicable), potentially smooth the solutions, and save plots of the calibration results to the diagnostics folder for every step. Quickmode (-q) can be used to perform only a subset of these tasks. The steps below are examples for a polarization continuum VLBA dataset. For other observations the steps may be different (see `picard/main_picard.py` and `picard/input/observation.inp`).

  (0) Generate an `accor()` calibration table. Corrects amplitude errors introduced by faulty sampler thresholds. The corrections are estimated by scaling the autocorrelations to unity. Not required for EVN data which has this correction already applied.

  (1) Generate a scalar `bandpass()` calibration table. It uses the autocorrelations to calibrate the amplitude frequency response of every station.

  (2) Generate a tsys calibration table. If necessary, correct for the opacity-induced source attenuation, using all available trec files for better fits (section 14 and section 16).

  (3) Generate a gc calibration table. The system temperature and gain curve information are used to multiply the correlation coefficients in the raw data with system equivalent flux densities (SEFDs) to have the visibilities scaled to units of Jansky. See section 12.

(4) Generate a single-band `fringefit()` calibration table to correct for instrumental effects. The instrumental phase and delay calibration is done by fringe-fitting every scan of the sources set via `calibrators_instrphase` in `picard/input/observation.inp` (unless an explicit list of scans is given in `picard/input/array_finetune.inp`) and then interpolating across scans between all solutions which have a high enough SNR (the threshold is defined in `picard/input/array_finetune.inp`). By default, scans will be fringe-fitter over their entire duration in this step. The basic principle of fringe-fitting is outlined in section 11.

(5) Determine optimal `fringefit()` solution intervals for every scan on a calibrator source (either from scratch or by reading in previously determined solints from disk). This is done by looking at the distributions of SNR vs solution interval on every baseline to the reference station and taking the shortest solint within the coherence time that still yields to detections on all baselines.

(6) Generate a multi-band `fringefit()` calibration table for the calibrator sources. Write out the same SNR-weighted average of RPC+LCP rates for both polarizations and smooth the multi-band delays in time (per scan and per antenna). Optimal solution intervals obtained from the previous step are used here.

(7) Generate a complex `bandpass()` calibration table. Corrects for the phase response over the bandpass for every antenna. (An amplitude correction is also possible, but not recommended, because this is already done by the scalar bandpass. See `picard/input/array_finetune.inp` for details.)

(8) Generate a RL delay calibration table using `gaincal()`. Calibrates the delay differences between the RCP and LCP receivers. Necessary for polarization experiments.

(9) Generate a RL phase calibration table using `polcal()`. Calibrates the RL phase for polarization experiments. The absolute RL phase establishes the EVPA of the source.

(10) Generate a D-terms calibration table using `polcal()`. Corrects for the instrumental leakage between the RCP and LCP feeds of every station.

(11) Generate a multi-band `fringefit()` calibration table for the science targets on scan-based solution interval with open search windows to take out the bulk delay and rate offsets of each scan.

(12) Determine optimal `fringefit()` solution intervals for all scans on science targets, with the same method as outlined for the calibrator sources above.

(13) Generate another multi-band `fringefit()` calibration table for the science targets. As opposed to the previous fringe-fit step, where the bulk delay and rate offsets are taken out, this step solves for residual intra-scan atmospheric phase distortions on the short optimal solution intervals determined by the previous step. Typically,

narrow search windows are used here. Same post-processing as for the calibrator sources multi-band `fringefit()` step (see above).

- **Run steps part 2.** These are executed after the calibration tasks. Quickmode (-*q*) can be used to perform only a subset of these steps.

  (h) Clear all existing calibrated data from previous `applycal()` runs.

  (i) Run `applycal()`, using the tables generated by all calibration steps. Flags all uncalibrated data (e.g., if no fringes were found in a scan).

  (j) Write the amount of flagged data to a file in the diagnostics folder.

  (k) Save plots for a set of calibrated visibilities in the diagnostics folder.

  (l) Take the calibrated data and average it into a new MS. Then, create uvfits files per source from the calibrated and averaged data.

7 Input Parameters

By default, input parameters are read from all `*.inp` files in the `picard/input` folder. They are stored in a single object `inp_params`, which is passed around in the code. A small description is given for each input parameter in the input files. One should definitely set the parameters in `observation.inp` each time the pipeline is run for a new VLBI experiment. For different arrays, different frequencies, or different calibration strategies, `array.inp` should be edited. Usually `constants.inp` can be left as the default. For a-typical calibration strategies or severe issues in the dataset which require non-standard calibration methods, the `array_finetune.inp` parameters can be adjusted. New input values can easily be added as `name_of_parameter = value_of_parameter`.

Multiple values can be separated by a semicolon, so that `name_of_parameter` becomes an array within the code. If not given as string, numbers are recognized as floats if they contain a decimal point and as integers otherwise. Booleans and strings are trivially recognized. `value_of_parameter` can then easily be accessed as `inp_params.name_of_parameter` in the code. For readability, lines can be continued with a backslash (\) character before a new line.

With the *--input* or *-p* command line arguments, a set of input parameters can be read from any directory, instead of the default `picard/input` folder (section 8).

The idea behind the input files is that users can keep and modify their own set of inputs. Therefore, `picard/input` has been added to `.gitignore`. However, an up to date developer's copy of the input files are kept as template in the repository's `input_template` folder. If these files are updated, then users will have to adjust their own input files accordingly, as described in section 21.

Additionally, `rPICARD` determines a set of ancillary or 'internal' metadata derived from the MS data itself. This data is stored in a single object for quick access from the different calibration modules. The stored information is about the polarization, frequency setup, data integration time, stations, scans, and observed sources.

7.1 Parameter Examples

Here, more detailed information is given about some of the input parameters mentioned in https://ui.adsabs.harvard.edu/abs/2019A%26A...626A..75J.

7.1.1 Parameters set by setup.py

The code uses the following prioritized reference station lists for the phase calibration by default:

1. **EHT**: ALMA, LMT, APEX, SMT, IRAM30m, SPT
2. **GMVA**: ALMA, Effelsberg, IRAM30m, Los Alamos, Fort Davis, Pie Town, Kitt Peak
3. **VLBA**: Los Alamos, Fort Davis, Pie Town, Kitt Peak

4. **EVN**: Effelsberg, Yebes, Medicina, Noto

The user could alter these lists based on the actual performance of the stations in an observations. The most important effects that need to be taken into account are weather and technical issues. These will become evident from the diagnostic output of the pipeline; most importantly from plots of system temperatures.

The following default search ranges are used when looking for fringes to calibrate atmospherically induced in-scan phase fluctuations:

1. **EHT**: 2 s – 30 s
2. **GMVA**: 2 s – 180 s
3. **VLBA**: 30 s – 400 s
4. **EVN**: 60 s – 400 s

Depending on the overall weather conditions, exact observing frequency, and source brightness, the user could adjust these search ranges. The best indicator is the number of fringe-non detections within the search windows, which are reported in the diagnostic plots for the fringe-fit solution interval searches. It is possible to specify different search ranges for calibrator sources and science targets. In the mm regime, `rPICARD` will try a few longer solution intervals to get detections for stations in scans outside of the ranges given above. For the EHT, 60 s and 120 s are tried. For the GMVA, 240 s and 300 s are tried.

For both the list of reference stations and phase calibration solution intervals, new lists of solution intervals for other arrays can easily be added.

7.1.2 Imaging Parameters

Table 1 gives a brief overview of the important parameters for `rPICARD` imager.

Table 1: Description of the most important `rPICARD` imager parameters and how they are updated in each imaging plus self-calibration iteration. For more information about the *tclean* parameter, see `https://casa.nrao.edu/casadocs-devel/stable/global-task-list/task_tclean/about`.

Parameter Name	Default	Description
cellsize	None	Size of an image pixel. If not set, CASA will determine it based on the longest baseline.
imsize	None	Size of the image in pixels. If not set, CASA will determine it based on the field of view.
niter0	500	Initial value for the maximum number of CLEAN iterations.

cleaniterations	‘shallow’	Parameter to update the *tclean niter* parameter, starting from the *niter0* parameter. If set to ‘constant’, niter will not be updated and for ‘shallow’, *niter* will be increased by *niter0* after each iteration.
gain	0.1	The CLEAN gain (fraction of source flux subtracted frm residual image).
multiscale	[0,2,6]	Pixel sizes for multi-scale deconvolution.
nterms	1	Number of Taylor coefficients in the spectral model for multi-frequency deconvolution.
robust	0.5	Briggs weighting robustness parameter (Briggs, D. S. 1995, 27, 1444).
threshold	‘auto’	Stop cleaning when the residual peak flux reaches this value. The default (‘auto’) uses the point source sensitivity determined from the data by the CASA *imager* tool.
nsigma	3.0	CLEAN stopping criterion based on the median absolute deviation.
startmod_sc	0	Do an initial phase self-calibration to a point source to align phases. The timescale can be set in seconds. The default (0) is to use the data integration time $t_{\rm int}$.
timeavg	False	Can average the data in time after *startmod_sc*.
phase_only_selfcal	[300s,10s]	A list of phase-only self-calibration steps to be done before the amplitude self-calibration and after *startmod_sc* and *timeavg*.
startsolint	10 hours	The starting solution interval for the amplitude calibration.
solint_denominator	2	The factor by which the amplitude self-calibration timescale $T_{\rm amp}$ is lowered after each iteration.
N_sciter	0	Maximum number of self-calibration iterations. The default (0) is to stop when $T_{\rm amp}$ reaches $t_{\rm int}$. If set to -1, only the phases will be self-calibrated. If set to -2, a single image without self-calibration will be made.
minsnr_sc_phase	3.0	The SNR cut for phase self-calibration solutions.
minsnr_sc_amp	5.0	The SNR cut for amplitude self-calibration solutions.

flag_last_sc	True	Flag data for which no self-calibration solutions are obtained in the last iteration.
amp_selfcal_ants	False	Can define a subset of station for which amplitude self-calibration is to be done. The default is to obtain solutions for all stations.
station_constraints	None	Can constrain the range for amplitude self-calibration gains.
station_weights	None	Can modify the data weighting of specific antennas.
uvrange_sc	None	Can constrain the u-v range for which amplitude self-calibration solutions are obtained.
uvzero_mod	None	Can anchor the large scale flux in the model visibilities to a specified zero-spacing source flux density up to a maximum u-v range.
mask	None	Can supply files with CLEAN box masks for the source that are to be used for different iterations.
interactive	True	Build CLEAN boxes interactively in *tclean*.
goautomated	False	Option to set *interactive* to False at runtime for a final set of CLEAN boxes.
usemask	'auto-multithresh'	Enable the *tclean* CLEAN auto-boxing capabilities (see text).

8 Command Line Arguments

The available command line arguments for `picard` are
[-p,--input <new-input-folder>] [-f] [-m] [-l (e)] [-q (<list>)] [-r (a)] [-d] [-s] [-n <#cores>] [-i] [-h]. They are all optional.

- With *--input*, the pipeline will look for *.inp files and a `mpi_host_file` in `new-input-folder` instead of the default `picard/input/` folder (section 7). If the shorter *-p* is used instead, the code will look for an `input/` or `input_template/` folder in the current working directory to use as `new-input-folder`. With this, every dataset can have a custom set of input parameters (as it should be), while using the appropriate computational resources.

- *-f* overwrites the `inp_params.fringe_params_load` parameter, forcing it to False: If there is an already stored set of optimal fringe-fitting parameters, then it will not be loaded and instead be determined again from scratch. The fringe-fitting parameters contain the information for each scan, which refant to pick and the optimal fringe-fit solution interval.

- *-m* overwrites the `inp_params.ms_metadata_load` parameter, forcing it to False: If there is an already stored set of metadata from the measurement set, then it will not be loaded and instead be determined again from scratch. Typically, this should be used when the science targets or calibrators in `picard/input/observation.inp` or `num_selected_scans` in `picard/input/constants.inp` are altered.

- *-l (e)* overwrites the default behaviour, where the `listobs()` task is not run again if a listobs file already exists. With *-l*, the pipeline will enforce to run `listobs()` again instead of quickly copying over an old file. Normally, running `listobs()` takes some time and should not be done multiple times unless the MS changed internally (after running `split()` for example). If not only *-l* is passed as command line argument, but *-l e*, the pipeline will exit after writing the listobs file ('inspection only'). This is useful if you do not know the names of the sources in the fits-idi files (section 5).

- *-q*, the 'quickmode', is explained in section 9.

- *-r*, an extra option for the handing of flag versions for the quickmode, is explained in section 19.

- *-d* overwrites the default behaviour, where all diagnostics (section 13) are stored in a new folder with the current datetime attached to its name for this run. With *-d*, the diagnostics folder that was last modified (typically, this should be the one from the previous run) will be used. This is useful when you are running the pipeline step by step using *-q*: combined with *-d*, all diagnostics will still go in the same directory then. If the same step is executed again in this mode, the diagnostics from the previous run will be overwritten.

- If *-s* is given, all calibration tables that were generated from previous runs and that would have been applied for the `array_type`[6] of this run are deleted before any new calibration tasks are executed (section 6).

- With *-n*, the pipeline will not use the `mpi_host_file` file to establish the MPI environment (section 4). Instead, *<#cores>* (must be some integer number > 1) CPU cores will be used for this run.

- *-i* activates the interactive/supervised mode. The pipeline will operate at the user's pace, waiting for a keypress for every step – either to execute the step or to exit at this stage. This is useful if the user wants to examine the results from every step before advancing. The module presented in section 10 offers a suite of functions that can be used to post-process calibration solutions from several pipeline steps in an interactive CASA shell before continuing to the next step.

- *-h* (or *--h*, *-help*, *--help*): Print help message and exit.

The order of the command line arguments is irrelevant, for example:
$ picard -m -q 1,3∼5,b,e -r a --input /data/VLBI/VLBA/M87/2008/input/ -f
and
$ picard --input /data/VLBI/VLBA/M87/2008/input/ -f -m -r a -q e,1,b,3∼5
are equivalent. The only constraint is that the list after *-q* must consist of comma separated characters without spaces (section 9).

Single-hyphen arguments can be combined as usual: *-fmdq ...* will be expanded into *-f -m -d -q ...* for example.

[6]The `array_type` is specified in `picard/input/array.inp` (section 7) and defines which calibration steps are executed according to the `array_specific_steps` defined in `picard/main_picard.py`.

9 Quickmode

Quickmode (*-q*) can be used to re-run certain steps of the pipeline.

Pass *-q <list>* to *picard* (see also section 8). For example:
$ picard -q 1~4,6,b,c,e,d

No spaces are allowed in *<list>* and it must consist of comma separated values. A ~ can be used to indicate a range of numbers; the example above is interpreted as 1,2,3,4,6,b,c,e,d. If *x* is given, it is interpreted as a,b,c,d,e,f,g,h,i,j,k,l,m,n... (for the lazy).

Information about which steps correspond to which number and letter are printed for every run of the pipeline. If *-q* is used without a list specified: *$ picard -q*,
the pipeline will exit after the information about steps is printed and not perform any calibration (dry run). Only the preparation procedures (section 6) are executed. A dry run is meant for testing and to quickly see which steps are executed in which order.

Rationale:
For multiple runs of the pipeline on the same dataset, one can quickly redo certain steps after adjusting calibration parameters in `picard/input/array.inp` or `picard/input/array_finetunes.inp` (section 7 and section 20). Or, a subset of steps can be executed, then the user can manually edit the calibration tables (flag delay outliers in the single-band `fringefit()` table for example, see section 10) or flag visibilities and then run the remaining steps (see also the *-i* command line option, section 8). Note that, if you are re-running single steps, all other previous existing calibration tables will still be applied on-the-fly and they will still all be applied with `applycal()` as described in section 6. If this is not what you want, then you can either use the *-s* command line option (section 8) or to move the old calibration tables to a different place.

Another pathway would be to run the pipeline in the standard mode, then image the exported data and save source models to disk (with CASA `tclean()` for example, see section 10). These can then be read in by a subsequent run of the pipeline (section 17). Then, one can for example redo the fringe-fitting step with the model using
$ picard -q <fringe-fit step number> -r a[7]

Or, quickmode can be used to exclude the amplitude calibrations steps, when no ANTAB table (section 12) is present[8], or to first do the phase calibration steps and the amplitude calibration steps at the end (see also section 20 on how to do that in a more systematic way).

Also, quickmode can be used to plot uncalibrated visibility data. In the default pipeline run (section 6), one of the last steps (after the calibration tables have been applied) is to plot visibility data. If you want to plot uncalibrated data, quickmode can be used to run the `clearcal()` and plotting step without `applycal()`:
$ picard -q <clearcal step number>,<plotting step number> -r

[7]See section 19 for the meaning of *-r a*.
[8]In that case it is necessary to set `pass_missing_antab = True` in `picard/input/constants.inp`.

10 Interactivity Capabilities

`picard/interactive_utils.py` contains functions that are independent from the rest of the pipeline. The functions can be imported in an interactive CASA shell if the module is added to your PYTHONPATH.

The following functionalities are currently implemented:

- An interactive flagging GUI based on the CASA `plotcal()` task for a quick and easy post-processing of calibration solutions for:
  - Single-band `fringefit()` calibration tables.
  - Multi-band `fringefit()` calibration tables.
  - `bandpass()` calibration tables.
- An imager that uses CASA's multi-scale, multi-frequency `tclean()` image reconstruction algorithm together with self-calibration loops to make images of the observed sources, creating FITS files. It requires an interactive mpicasa session, which can be started with
  $ mpicasa -n <number_of_cores> path_to_casa/casa [casa_options]
- A function to concatenate UVFITS files (useful when comparing CASA data with products from the EHT HOPS pipeline).

11 Fringe-fitting

Correlators employ sophisticated geometric models and make use of clock searches when they let the signals from a pair of two antennas interfere – i.e. cross multiplying their signals to form baseline-based visibilities. However, the correlator models are never perfect – in almost every case residual delays (phase slopes versus frequency) and rates (phase slopes versus time) will still be present in the data. These have to be taken out in post-processing, using a technique that is called 'fringe-fitting' (otherwise there will be large coherence losses when averaging in time and frequency).

The CASA `fringefit()` task will FFT the visibilities versus frequency and versus time and find the baseline-based residual delays and rates from the FFT peaks. These are globalized (going from baseline quantities to station solutions) with a least squares approach and referenced to a common reference antenna.

Typically, the first step is to align the phases across the separate frequency windows (IFs or spectral windows) by fringe-fitting each window individually, referenced to the same frequency.[9] The signal must be strong in each window, so a bright source should be used. Rates are zeroed and the phase plus delay solutions will be applied to all scans. This procedure is called 'instrumental phase and delay calibration'. In the code, the method for the instrumental phase and delay calibration can be specified in `picard/input/array_finetune.inp` (section 7), as explained in the comments for the different parameters. One can either select specific scans or fringe-fit all scans on the instrumental phase calibrator specified in `picard/input/observation.inp`. Next, one can either fringe-fit over scan durations, give a specific solution interval, or use the estimated optimal solution interval for each scan (see subsection 11.2 below). For the two latter cases, one should make use of the smoothing options in `picard/input/array_finetune.inp` to obtain single solutions per scan to correct for this instrumental effect. It is recommended to integrate over scan durations. Lastly, one can set which solutions are to be used. First an SNR cut is made (can be set by the user, the default is 10). From the remaining solutions one can either select only a single solution (with the highest SNR) per station and spectral window that is to be applied to the whole dataset. Or interpolate across scans between all solutions that made the initial SNR cut. This could be used if there are drifts in the electronics during very long observing sessions or when recording equipment was restarted during a run for example.

Once the phases are aligned (coherent) across the whole frequency band, `fringefit()` can run over the whole band for a higher signal to noise ratio – so the weaker sources can be fringe-fitted as well. For high frequencies, a source is not detected if you cannot find fringes on it. At lower frequencies it is possible to transfer phases from a bright nearby calibrator, so that weaker sources can be detected (at lower frequencies the troposphere does not distort the phases that much, so that solutions derived from another part of the sky can be used). Phase-referencing with this pipeline is explained in section 15.

[9] At high frequency observations it may be necessary to take out the fast phase rotations beforehand. In CASA it is possible to do that with `fringefit()` using `combine='spw'` as the FFT will be done over the whole band, which is very sensitive.

11.1 Picking a Reference Station

The `refant` and `refant_minvaliddata` parameters in `picard/input/array.inp` and `picard/input/array_finetune.inp` respectively determine which reference station will be used for which scan. The `refant` parameter gives a prioritized list of antennas as options for the reference stations. For each scan, the first antenna in the `refant` list which has a fraction of valid data (number of unflagged visibilities over the number of flagged visibilities) larger than `refant_minvaliddata`, unless less than 15 % of additional valid data was gained with respect to a station higher up in the `refant` priority list. If none of the `refant` antennas makes this cut-off, the one antenna with the most valid data from that list is picked as refant.

For each scan, fringes are referenced to the refant of that scan. In the end, all fringe solutions will be re-referenced to a common refant with the CASA *rerefant* task. For polarization experiments, low values for `refant_minvaliddata` should be used to avoid frequent re-referencing. Also, if there is a single central and sensitive station is present in the array, this station should be first in the `refant` list and `refant_minvaliddata` could be set to a low number to ensure that this best station is picked most of the time.

Within scans, `rPICARD` has the option to perform an exhaustive fringe search. The correlated signal power is usually a strong function of baseline length for resolved sources – the shorter the baseline length, the higher the SNR. For global 3 mm observations for example, picking Effelsberg as reference station may yield detections to all other European stations and to the GBT across the Atlantic, but not to each individual VLBA station. However, fringes may be present between the GBT and the VLBA. So both the European and the North American stations can be calibrated by performing two fringe-fits – one with Effelsberg as reference and one with GBT as reference, where all fringe solutions to the GBT are re-referenced to Effelsberg using the phase, delay, and rate relation between the two stations from the Effelsberg-GBT fringe detection in the first fringe-fit run. A generalized version of this process is implemented in `rPICARD`. The algorithm will go through each scan s in the observation and do the following:

1. Fringe-fit the data using the primary refant P_s. This refant is determined based on the amount of valid data in each scan, see above. Store all fringes with insufficient SNR in a list $N = \{\text{non-detections}\}$ and make a static copy N_P of that list to keep track of the non-detections to the primary reference station.

2. Compute the list of secondary reference stations S_s: If we denote all possible reference stations, set by the `refant` parameter in `picard/input/array.inp`, as R, the secondary refants for a particular scan are $S_s = \{R\} - \{P_s\}$.

3. If no non-detections are found in step 1, or when no detections are found between P_s and any station in S_s, the algorithm will go to step 6.

4. For each station s_s in S_s:

   (a) Fringe-fit the data using s_s as reference station and gather all stations, which can be connected to s_s via fringe detections, in a list called f_s.

   (b) Compute the connections to other reference stations as $c_1 = R \cap f_s$ and to stations not yet connected as $c_2 = N \cap f_s$.

   (c) If

$$|c_1| \geq 2 \tag{1}$$

   or

$$|c_1| \geq 1 \wedge |c_2| \geq 1 \; , \tag{2}$$

   s_s is added to a list U of useful secondary reference stations as $U(s_s) \equiv [U_1(s_s), U_2(s_s)] = [c_1, c_2]$. The entries of U are referred to as 'sub-clusters' in the code.

   (d) Update the list of non-detections: $N \to \{N\} - \{c_2\}$. Go to step 5 when N is empty.

5. Connect all sub-clusters:

   (a) The c_1 entries in each U determine which s_s can be connected. All entries are discarded for which no path – via any combination of multiple c_1 connections – can be established to P_s. If that leaves U empty, the algorithm will go to step 6.

   (b) For each station n_p in N_P – if it is in any c_2 of U – the shortest path through c_1 connections, along the prioritized list of reference stations, is stored as

$$E_s = \{[n_p, P_s \leftarrow x_1 \leftarrow x_2 \leftarrow ... \leftarrow x_M] \mid n_p \in N_P \cap U_2 \text{ and } x_i \in U_1\} \; . \tag{3}$$

   These E_s paths determine the exhaustive fringe search for each scan s employed by `rPICARD`. In the notation above, fringes for n_p are obtained from a x_M reference station, which can be connected to P_s via a chain of other secondary reference stations $x_{j \neq M}$.

6. The FFT delay and rate solutions and their SNR of all fringe-fit steps in this exhaustive fringe search process are stored as diagnostic output in csv ASCII format by the pipeline (section 13 and section 14). This information is usful to get a quick overview of all detections and non-detections in the dataset and to identify gross residual post-correlation delay and rate errors.

In the above process, each fringe-fit is performed over full scan durations and only FFTs are computed to determined detections, the least-squares is skipped. SNR cuts for non-detections are based on 1.2×`fringe_minSNR_mb_long_sci` for science targets and `fringe_minSNR_sb_instrumental` for calibrator sources from `picard/input/array_finetune.inp` (subsection 11.3). The SNR simply set limits for scan-based SNR thresholds below which an exhaustive fringe search needs to be employed – no data will be flagged in this step. `rPICARD` does the reference station search and exhaustive fringe search before any other phase calibration steps. In all subsequent fringe-fit runs, all fringe solutions are re-referenced according to the E_s paths. That is, for every fringe-fit of a scan s, the phase, delay, and rate solutions are re-referenced for every calibration table entry of n_p in E_s. Along the re-referencing chain, all flags are accumulated – if a single fringe along the chain fails, a flag will be written for the final re-referenced fringe as well. It should be noted that fringe solutions have uncertainties S due to the thermal noise in the system. The total uncertainty along a re-referencing chain grows as

$$S_{\text{total}} = \sqrt{\sum_{i=1}^{M} S_i^2} \, , \tag{4}$$

where each S_i represents the fringe solutions uncertainty of a specific station x_i along the chain. Considering current VLBI arrays, maybe one or two secondary reference stations x_M are typically chosen for an exhaustive fringe search and they will have direct connections to P_s in most cases. The `fringe_exhaustive_refant_search_depth` parameter in `picard/input/array_finetune.inp` can be set to control the maximum depth of the exhaustive fringe search or to disable it completely.

In the future, a FFT baseline stacking functionality will be added to the `fringefit` task, which will achieve similar results as the exhaustive fringe search.

11.2 Finding the Optimal Solution Intervals

Optimal fringe-fit solution intervals should be long enough to have enough SNR for a detection and short enough to capture atmospheric phase fluctuations. This is done first in the `task_fringefit_solint_cal` step for the calibrator sources and for the science targets in a later `task_fringefit_solint_sci` step, where solutions derived from calibrator scans for instrumental effects like a phase bandpass and delay plus phase offsets between spectral windows can be applied on-the-fly.

The method used is to just run the FFT (not the globalization) for many different solints for every scan and all baselines. First, a minimum detection SNR threshold is set in `picard/input/array_finetune.inp` (the default is 5.5). For all baselines that make it above this threshold, the smallest solint that yields an SNR above the minimum is stored. From that collection the longest solint is picked – this is the minimum solint to have all possible detections on the different baselines. If `fft_solint_estimation=sqrt` is set in `picard/input/array_finetune.inp`, the smallest solint that yields detections is taken as starting from which the expected coherent increase in SNR (by the square root of the solution interval) is compared to the actual increase in SNR from the FFT for each baseline. Once the actual SNR drops below the expected parabolic increase, the solint corresponding to that breaking point is taken as the coherence time on that baseline for that scan. Finally, the smallest number from the collection of all coherence times on each baselines per scan is taken as the optimal solution interval. Currently, a simpler and more robust method is

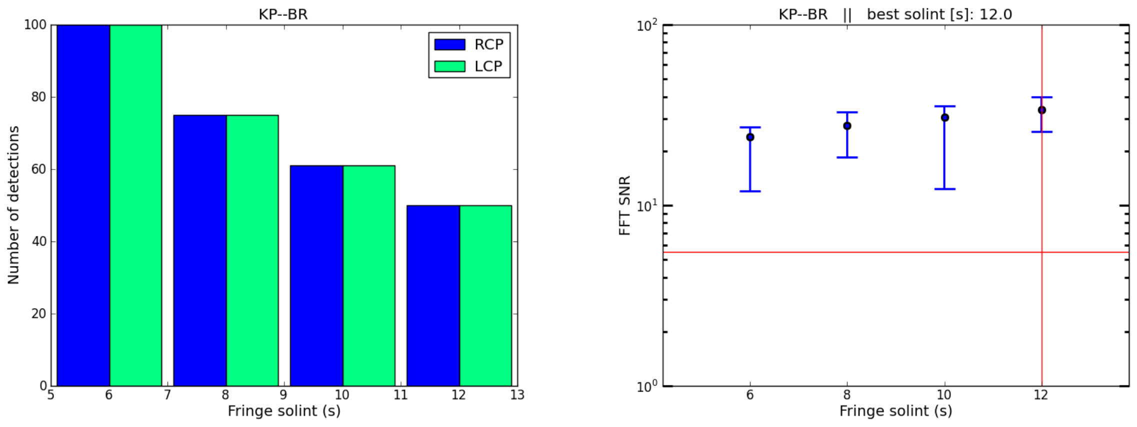


Figure 1: Example of diagnostic plots from a fringe-fit solution interval optimization procedure. *Left:* The number of detections within a specific scan for the KP–BR baseline for different solution intervals. It can be seen how the total number of detections goes down, as less solutions can be found within a fixed scan duration for longer segmentation times. *Right:* The change in FFT SNR for different solution intervals. The minimum, median, and maximum SNR from all solution segments for each specific integration time are plotted for the polarization with the smaller overall SNR. The horizontal red line marks a detection threshold of SNR = 5.5 and the vertical red line marks the minimum solution interval to have detections on all possible baselines, which was driven by a different baseline with a lower SNR.

implemented as default (`fft_solint_estimation=1.01`), where baselines are ignored if their SNRs do not increase with solint after a few iterations and the stopping criterion is set by SNR > threshold. The determined optimal scan-based solints will be used for all subsequent global fringe-fitting steps. There is the additional option to fringe-fit certain scans again with different (typically longer) solints, if there are non-detections on certain baselines – see the `fringe_solint_mb_reiterate` parameter in `picard/input/array.inp`.

All related data and plots from this optimization procedure are saved in the diagnostics (section 13) folder. An example is shown in Figure 1.

Dictionaries of the determined solution intervals and reference stations (subsection 11.1) are stored on disk. For re-runs of fringe-fitting steps, these parameters are loaded and used again, unless the *-f* command line option is set (section 8).

11.3 The Individual Fringe-fitting steps

Table 2 gives an overview of the different fringe-fitting steps implemented in `rPICARD` together with their main input parameter options, which are in `picard/input/array.inp` and `picard/input/array_finetune.inp`.

Science targets are first fringe-fitted on very long timescales to maximize the chance of getting a detection with the `phase_calibration.task_fringefit_multi_sci_long` step. This is not done for the brighter calibrators to save computational time. Each solution interval search following the method introduced in subsection 11.2 is done with the appropriate FFT windows. For example, narrow fringe search windows are used for the calibration of intra-scan atmospheric residual effects after the `phase_calibration.task_fringefit_multi_sci_long` step for the science targets. If the `fringe_SNR_cutoff_float_Perr` parameter in `picard/input/constants.inp` is set to True, the min SNR fringe detection threshold is lowered based on a reduction of the probability of false detection if narrow search windows are used.

Table 2: Overview fringe-fitting steps and main input parameters in picard/input/array.inp and picard/input/array_finetune.inp.

Function[a]	search windows[b]	solint[c]	min SNR[d]	step description
solint_cal	initial	NA	mb_shortFFT	Solint estimation following the subsection 11.2 method for calibrators.
multi_cal_coher[e]	initial	mb_coher	mb_coher	'Coherence' calibration for the calibrators on subsection 11.2 timescales, where intra-scan phases and rates are solved before doing the instrumental phase and delay calibration. This is done only at high frequencies.
single	initial	sb_instrumental	sb_instrumental	Instrumental phase and delay calibration.
multi_cal_short[e]	initial	mb_short	mb_short_cal	Multi-band fringe-fit on subsection 11.2 timescales, solving for phases, rates, and delays for calibrator sources. No long integration like multi_solint_sci is done for the calibrators to save computational time – the calibrator sources should be bright enough that this is not necessary.
multi_sci_long	initial	mb_long	mb_long_sci	Multi-band fringe-fit on long timescales for the science targets. This will determine if a source can be detected or not.
multi_solint_sci	mb_sci_short	NA	mb_shortFFT	Solint estimation following the subsection 11.2 method for science targets.
multi_sci_short[e]	mb_sci_short	mb_short	mb_short_sci	Residual multi-band fringe-fit on subsection 11.2 timescales, solving for residual (after multi_sci_long) intra-scan phase, rate, and delay variations for science targets. Typically, narrow search windows are used for the residuals.

[a] All function names have a phase_calibration.task_fringefit_ prefix.
[b] All FFT search window input parameters have a fringe_*_window_ prefix.
[c] All fringe-fit solution interval parameters have a fringe_solint_ prefix.
[d] All fringe-fit SNR threshold parameters have a fringe_minSNR_ prefix.
[e] If for some stations no fringes are found within the solint ranges for these steps, these stations are fringe-fitted again on longer timescales, given by the fringe_solint_mb_reiterate parameter in picard/input/array.inp.

12 ANTAB Tables: A-Priori Information for the Amplitude Calibration

You should get ANTAB tables from the observatories. They contain information about the DPFU, gain curve, and Tsys measurements from every station during the run. In the list below, specific information about ANTAB tables for different arrays is provided. You should put the ANTAB table (or a link to it) in your working directory (section 14).

1. **EHT**: You should have received an ANTAB table with you data. If not, then please contact me. If you want to calibrate the sidebands tgether, you may want to edit the INDEX rows of the ANTAB table.
2. **GMVA**: You should have received an ANTAB tables stations with you data. If not, then you could contact Thomas Krichbaum. Otherwise, you may find your metadata here.
3. **VLBA**: `SYSTEM_TEMPERATURE` tables should already be attached to your fits-idi files. Also a `GAIN_CURVE` extension should be attached as well. If not, you will still need an ANTAB table with DPFU and gain curve information per station. In that case, you should compile your own ANTAB table using entries from the `vlba_gains.key` file. Instructions on where to find that file on the public NRAO ftp server are outlined here. You will need to select DPFU and POLY entries from every station for the right frequency range and observing time. Moreover, you will have to adjust the format a little bit (switch <stationcode> with GAIN, to get the format outlined below).
4. **EVN**: You should have received an ANTAB table with your data. If not, you will find one on the EVN archive under your experiment code.

A detailed description of the ANTAB format can be found here. In the simplest form, an ANTAB table looks like this:

1) First, the *gain group* – a single line for each station giving the DPFU and gain curve:

```
GAIN <stationcode> <mount-type> DPFU = <rcp>, <lcp> POLY = <a0>, <a1>, <a2>, <a3>, ... /
```

Here, <mount-type> is typically ALTAZ, <rcp> and <lcp> are the DPFU values for the RCP and LCP receiver, and <a_i> are the polynomial coefficients of the gain curve.

2.1) Second, the *Tsys group* – blocks of system temperature with columns per IF (spw) and rows for the time spacing. For each station a block starts with

```
TSYS <stationcode> INDEX = 'i1', 'i2', 'i3', ... /
```

Where, the 'i_j' assign columns to RCP, LCP and IFs (spws).

2.2) Next, the Tsys values for the station block are given; one line for every timestamp:

```
<UT-day> <UT-time> <T1> <T2> <T3> ...
```

With <UT-day> in the ddd format, <UT-time> in a hh:mm.mm format, and $<T_j>$ the T_{sys} values in Kelvin assigned by i_j to a receiver and IF (spw).

2.3) Lastly, a station block is terminated with a / and the next one can start:

```
/
TSYS <next stationcode> INDEX = ... /
...
```

The ANTAB table formatting requirements are more strict for CASA than for AIPS. Every entry must be on a single line with keys and keyvalues written in the order shown above. Example:

```
GAIN AA ELEV DPFU = 1.0 POLY = 1.0 /
GAIN AP ELEV DPFU = 0.02478, 0.02521 POLY = 0.95515, 0.0022795, −2.8953E−05 /
GAIN AZ ALTAZ DPFU =  0.01683, 0.01681 POLY = 0.727089, 0.947364E−02, −0.822152E−04 /
GAIN PV ELEV DPFU = 0.127 POLY = 0.36029, 0.026008, −0.00026431 /
TSYS PV INDEX = 'R2', 'L2', 'R1', 'L1' /
141 03:00.317 131.89 142.49 132.70 142.70
141 03:01.867 132.07 142.76 132.77 142.76
!... and so on (exclamation mark = comment char)
/
TSYS AZ INDEX = 'L1', 'L2', 'R1', 'R2' /
141 03:01.456 122.24 291.24 117.59 332.75
141 03:02.956 122.18 292.14 117.66 331.82
!... and so on
/
!... idem for the other station blocks
```

The amplitude calibration scales visibilities formed from voltages in the correlator to a physical scale in Jansky (Jy). The system equivalent flux density of a single telescope is given as

$$\mathrm{SEFD} = \frac{\mathrm{Tsys} \cdot \exp(\tau)}{\mathrm{DPFU} \cdot \mathrm{gc}} \,. \tag{5}$$

Where Tsys is the system noise temperature in Kelvin (function of time), τ is the opacity (section 16), DPFU is the telescope gain in Kelvin per Jansky (the DPFU depends on the effective geometric area of the dish; it is mostly constant – differences may occur between day and night time), and gc is the normalized gain curve, which describes the variation of the antenna gain with elevation.

If a calibrated correlation coefficient (visibility amplitude) on a baseline between stations 1 and 2 is given by $r_{1,2}$, then the correlated flux density $S_{1,2}$ in Jy will be

$$S_{1,2} = r_{1,2}\sqrt{\mathrm{SEFD}_1 \cdot \mathrm{SEFD}_2}\,, \tag{6}$$

where $\mathrm{SEFD_i}$ is the system equivalent flux density of station i given by Equation 5.

13 Diagnostics

By default, the code produces logs and many diagnostic plots from the different calibration steps, unless these options are turned off in `picard/input/array_finetune.inp` (section 7). The diagnostics are stored in a folder in `inp_params.workdir` that also has the current datetime (UTC) attached to its name.[10] This and the fact that also the input parameters are logged, make it easy to run the pipeline several times on the same dataset with different input parameters and to compare the results.

For every run of the pipeline, `rPICARD` writes the usual `casa.log` file with the datetime attached. Moreover, (because of the talkative nature of mpicasa), stderr is redirected to a `mpi_and_err.out` file with the current datetime attached as well. If the pipeline finishes properly, both `casa.log` and `mpi_and_err.out` are moved over to the diagnostics folder of the run. If for some reason the pipeline crashes (not terminating with a - *FINISHED* - print to the terminal), both files are kept in the directory where `rPICARD` was called from. They can be examined to find the reason for the crash.

Moreover, the command line call with the used command line arguments (section 8) are saved to a file in the diagnostics folder (as specified in `picard/input/constants.inp`, see section 7) for every run. If the *-d* command line argument is used (writing incrementally to the same diagnostics folder), all incremental calls are appended to the same file.

13.1 Jplotter

By default, the code will produce plots of (calibrated) visibilities as one of the last steps (section 6). The recommendation is to use Harro Verkouter's jplotter program (See `README.md`) as it is faster and produces plots with a better layout than the alternative standard CASA `plotms()` tool. Moreover, jplotter does not require X-forwarding when saving plots directly to disk (it is unclear why this seems to be the case for `plotms()`...) and it has a simple and intuitive syntax exemplified in the `make_jplotter_plot()` function in `picard/pipe_modules/diagnostics.py`.

The jplotter figures made by `rPICARD` will show amplitudes and phases as a function of time (averaged over frequency) and as a function of frequency (averaged over scan durations) for each baseline. Different correlations (RR, LL and if specified also RL, LR) will be depicted with different colors. Unflagged datapoints are shown as dots. Fully flagged datapoints are plotted as crosses.

[10] Unless *-d* is used, see section 8.

14 Special Filenames

For the default input parameters, the pipeline will recursively search the workdir specified in `picard/input/observation.inp` (section 7) for files with certain extensions.[11] These extensions are defined in `picard/input/constants.inp`:

1. `flagfile_extensions`: All files with these extensions are used as metadata flag files.
2. `antab_extensions`: A file which has one of these extensions is used as ANTAB metadata, see section 12.
3. `fitsidi_extensions`: All files with these extensions are used as fits-idi input files.
4. `modelfile_extensions`: All files with these extensions are used as models for the observed sources, see section 17.
5. `trecfile_extensions`: All files with these extensions are used to look for T_{rx} (receiver temperature) info per station, see section 16.
6. `weatherfile_extensions`: A file which has one of these extensions is used as weather metadata, needed for section 16. Such an external file should only be necessary for data from the GMVA.
7. `antenna_mount_corrections_file`: If this file exists, it will be used to overwrite antenna's mount types in the MS. This is only necessary if erroneous mount types are written in the fits-idi input files.

The other special filenames used by the pipeline in the working directory are:

1. `ms_name` in `picard/input/observation.inp`: Name of the measurement set.
2. `gc_dpfu_fromidi_file` in `picard/input/constants.inp`: Name of the file that will be created to convert DPFUs and gain curves from a GAIN_CURVE FITS-IDI extension into an ANTAB ASCII file.
3. `gc_conversion_file` in `picard/input/constants.inp`: Name of the file that will be created to make a VLA-type gain curve from the ANTAB data.
4. `store_optimal_fringe_params` in `picard/input/constants.inp`: Files (with additional .sci and .cal extensions) that will be created to store fringe-fit parameters.
5. `store_scan_refants` in `picard/input/constants.inp`: Files (with additional .sci and .cal extensions) that will be created to store dictionaries with the chosen reference station for each scan.

[11] Links will also be followed.

6. `diag_fringe_overview` in `picard/input/constants.inp`: Files (with additional .sci and .cal extensions) that will show an overview of FFT-fringes for all scans across all baselines to all reference stations used (including secondary ones, subsection 11.1).
7. `store_ms_metadata` in `picard/input/observation.inp`: Name of the file that will be created to store internal (ancillary) metadata.
8. `diagdir` in `picard/input/observation.inp`: Name of the diagnostics folder that will be created with the UTC time attached to its name.
9. `diag_*` in `picard/input/observation.inp`: Name of several files that will be created in the diagnostics folder.
10. `calib_*` in `picard/input/array_finetune.inp`: These parameters contain names for the calibration tables that will be created. For phase-referencing, tables with a .phaseref extension will also be created.

15 Phase-Referencing

Phase-referencing is enabled if `calibrators_phaseref` in `picard/input/observation.inp` (section 7) is set. For every `science_target`, there must be a corresponding `calirbators_phaseref` source specified. An extra (.phaseref) table from the phase-referencing calibrators is created where all flagged(failed) solutions are removed and all solutions are smoothed my a median filter on a per-scan, per-antenna, per-spw basis. The fringe solutions from this phaseref table will be transferred to the science targets. If `phaseref_ff_science = True` in `picard/input/observation.inp`, a fringe-fit is also done on the science targets after the solutions from the phase-referencing sources have been applied. This can be done if the science targets are strong enough to be fringe-fitted for residual phases/rates/delays. Else, set `phaseref_ff_science = False`.

16 Opacity Correction

At high observing frequencies (> 15 GHz), it is necessary to correct for the attenuation of the source caused by the atmospheric opacity τ. This correction can be applied by raising the system temperature: $T_{\text{sys}} \to T_{\text{sys}} \exp(\tau)$. If the system temperature is measured with a chopper, then this correction is already applied and no further post-processing is necessary. This is the case for data from the EHT array for example. For other arrays, like the VLBA and GMVA, it is necessary to perform this correction.

The pipeline can solve for the opacity by solving the (approximate) T_{sys} equation for τ:

$$T_{\text{sys}} = T_{\text{rx}} + (1 - e^{-\tau})T_{\text{atm}} \Leftrightarrow \tau = -\log\left(1 - \frac{T_{\text{sys}} - T_{\text{rx}}}{T_{\text{atm}}}\right). \tag{7}$$

Here, T_{atm} is the mean atmospheric temperature and T_{rx} is the receiver temperature. The equation should be evaluated separately for RCP and LCP receivers.

For Equation 7, T_{atm} must be estimated from the available weather (WX) data in the `WEATHER` subtable of the measurement set. Users can easily add their favourite weather models in the `opacity_correction.py` module of the pipeline. Currently available are the accurate and recommended Pardo et al. model and the simple weather model from Altshuler, Falcone, and Wulfsberg (1968): ‘Atmospheric effects on propagation at millimeter wavelengths’[12].

The other unknown is the receiver temperature T_{rx}. By default, the code will estimate T_{rx} from an extrapolation of a T_{sys} vs airmass fit to zero airmass. The `opac_corr_airm_max` parameter in `picard/input/array_finetune.inp` can be set to exclude low elevation (high airmass) data from the fit. The fit may not be reliable if the atmospheric conditions change significantly over the course of the observations. By default, plots of the fits are made which can be inspected. Also, the pipeline will print warnings if there seems to be problems with the T_{rx} fits.

There are two possible failure modes, both of which are typically caused by faulty estimates of T_{rx}:

1) $T_{\text{rx}} < \min(T_{\text{sys}})$.[13] This can occur when the weather was bad during high elevation observations. If this happens, the code will try to guess a new value for T_{rx}, based on the $\min(T_{\text{sys}})$ measurement while assuming that $\tau = 0.05$ and $T_{\text{amb}} = 273.15$ K locally.

2) $(T_{\text{sys}} - T_{\text{rx}})/T_{\text{atm}} > 1$. This can occur when the fit underestimated T_{rx} or when T_{atm} is erroneously small. This typically takes place when T_{sys} is large – i.e. at low elevations with a high opacity. If this happens, the code will assume $\tau = 1$ instead of using Equation 7.

Additionally, whenever τ exceeds 1.0 at a certain time for a certain source, the code will use the previously determined value for the opacity.

The T_{rx} values determined from fits as described above can be overwritten by values manually entered in a `trec` file (section 14). Obviously, this should be done when observatories provided accurate T_{rx} values. Otherwise, the pipeline will print warnings if problems with

[12]Here, T_{atm} is estimated from the ambient temperature T_{amb}, which is measured by the local telescope weather station, as: $T_{\text{atm}} = 1.12\,T_{\text{amb}} - 50$ K.

[13]T_{sys} below 2.73 K are regarded as invalid and will be ignored.

the opacity correction are encountered. In that case, the user should definitely inspect the diagnostic plots for the T_{rx} fits. For problematic fits, a better receiver temperature estimate can generally be obtained by looking at the fits and guessing what T_{rx} should be.

Once, a `trec` file has been written with improved estimates, quickmode (section 9) can be used to re-run `task_tsys_add_exptau()`.

The `trec` files should be ASCII and contain lines with the following format:
<station code> <receiver> <T_{rx} value in K>

For example:

```
BR RCP 113.5
BR LCP 110.5
GB RCP 65.5
KP LCP 111.5
KP RCP 97.8
```

17 Source Models

CASA will store source models in a `MODEL_DATA` data column of the MS. For many calibration solvers, the data is divided by the model. Having a reliable source model therefore helps with the calibration.

By default, the pipeline will look for files with modelfile_extensions in the workdir (section 14). For every <source> in the data, the pipeline will search for <source>.<modelfile_extension> files and use them as source model for that source.

An optimal calibration strategy would be to run the pipeline, make first images from the products (using CASA's `tclean()` task for example, see section 10), save the resulting source models (modelfiles are automatically generated by `tclean()`), and run the pipeline again. This second run will then automatically make use of the determined source models for an optimized calibration (e.g., `fringefit()` or `polcal()`). Then, new images can be made with the improved calibration.

18 Flagging Algorithms

By default the code will first look through any available metadata from flagging information. Then, the edges of the bandpasses will be flagged as specified in `picard/input/flagging.inp` (section 7). See also section 6 and section 19.

Additionally, two experimental flagging algorithms are implemented, which worked satisfactorily for the EHT array. However, they have not been tested thoroughly for any other array. It is noteworthy that the options in `picard/input/flagging.inp` allow for flagging dry runs, where no written flags are actually applied. Every flagging step will write the flags to files specified in `picard/input/flagging.inp`.

`fg_autocorr_vs_freq()` looks for outliers in frequency space (channel outliers in station's bandpasses). It will flag outliers based on the difference in amplitude values between individual (or groups of) channels compared to the median derivative in the autocorrelation frequency spectrum.

`fg_autocorr_vs_time()` looks at the station based frequency-averaged autocorrelation spectrum as a function of time and flags around integrations whose amplitude differs by too much from the median amplitude per scan.

Both algorithms are described in more detail in the function's docstrings in `picard/pipe_modules/flagging_algorithm.py`.

19 Flag Versions

By default, the pipeline will save an initial blank flag version when run for the first time. Following section 6, there are multiple steps (marked by being underlined) which will write flags. Moreover, when the calibration is applied with `applycal()` at the end of the pipeline, all uncalibrated data will be flagged, while a backup is saved.

In quickmode (*-q*) (section 9), the current flag status is used by default. If the pipeline is run not in quickmode, the flags are saved/reverted to the initial blank version by default.

These default approaches are overwritten with the *-r* command line option. Using just *-r*, flags are saved/reverted to the initial version even in quickmode. Using *-r a*, flags are reverted to the latest flag version from `applycal()`, i.e. the flags before `applycal()` flagged any data. Ergo, this version should contain flags from the explicit flagging steps of the pipeline exclusively.

Whenever -*q* is used to redo certain calibration tasks[14] of the pipeline, it is recommended to use *-r a*, to have the flags from the flagging algorithms applied, while ignoring flags from previous `applycal()` runs.

[14] Every time the pipeline is run, all steps are printed to the terminal. The steps which are enumerated by numbers (and not letters) are the tasks. This convention is also adopted for the list in section 6. Moreover, all tasks follow the naming convention `task_*` when printed to the terminal.

20 Change of Calibration Strategy

A new calibration strategy (for a different telescope for example) can easily be implemented by adding a new block for `inp_params.array_type` in the main file of the pipeline, ideally with a customized `picard/input/array_finetune.inp` file (section 7). Of course, it is also possible to change the order of calibration steps in `picard/main_picard.py`. For example, the amplitude calibration steps could be done after fringe-fitting. Generally – for any calibration task – all calibration tables (if written successfully) from the previous steps will be applied on-the-fly where applicable.[15] A new calibration task can be defined in any module and if it obeys to the convention `task_taskname()`↔`calib_taskname` outlined in `picard/input/array_finetune.inp`, it can be wrapped in `calibration.go_calibrate()` for on-the-fly calibration, potential smoothing, and generation of diagnostic plots.

Example:
Let's say you have developed a method to refine the bandpass calibration, which you want to apply after the standard bandpass calibration. Then you should do the following:

1. Add a custom module, let's say `my_functions.py` in the `pipe_modules` folder. Add any required imports to that module.

2. Put your code in a function in that module, obeying the naming convention `task_taskname()` – let's say we call it `task_super_bandpass()`. Make sure that the function returns `True` after successful execution. The task must take `_inp_params, _ms_metadata, caltable, _fly_calib_tables, _fly_calib_interp, _fly_calib_gainfd, _mpi_client` as non-default arguments and any additional number of default arguments to overwrite parameters if necessary. Any user-defined input parameter must be defined in a `picard/input/*.inp` file – let's say you add a `normalize_super_bandpass = True` parameter to a `picard/input/my_inputs.inp` file, which you can then retrieve as `_inp_params.normalize_super_bandpass` in `task_super_bandpass()`. `_ms_metadata` can be used to retrieve any kind of required meta data (see the `diagnostics.py` module). `_fly_calib_tables` and `_fly_calib_interp` are the calibration tables from all previous calibration steps (see 3. below), that should be applied on-the-fly (which is trivial for CASA tasks). `caltable` is the name of the calibration table that will be written by `task_super_bandpass()` (see 4. below).

3. Import `my_functions.py` in `main_picard.py` and add `my_functions.task_super_bandpass` to `all_calibration_steps` for the arrays for which this calibration is to be done. Remember that the order within `all_calibration_steps` matters – the steps are executed in sequence while applying the calibration tables from previous steps on-the-fly.

[15]For some tasks like `accor()`, on-the-fly calibration is ignored. Also, the multi-band fringe-fitting is split between science targets and calibrators.

4. Add the mandatory input parameters to the "Block of calibration parameters for each calibration step" in `picard/input/array_finetune.inp`. You have to add it as `calib_super_bandpass = ...` to define the name of the output calibration table, together with smoothing, diagnostics, and interpolation options for the calibration solutions.

21 Code Updates

To always have the latest version of the code, running *git pull* should suffice.

As described under 'Versioning' in the `README.md` file, no additional action is required when pulling changes that go along with minor updates or patches.

For major updates, either `README.md` changed since a new recommended CASA version became available (run *./setup.py* again in this case), or the files in `input_template` (section 7) are updated (then *git diff* should be examined – you will have to adapt your local input files to the changes).

22 Frequently Asked Questions

1. Why are you doing a scalar bandpass?
   The scalar bandpass is used to accurately correct for the amplitude bandpass. Corrections from the complex bandpass are limited by the SNR of the cross-correlations and therefore, data from all scans of the specified calibrator sources are aggregated for the calibration task. If a scalar bandpass correction has been performed, the complex bandpass is used to correct only the phases. The scalar bandpass calibration should be skipped in the presence of strong RFI that has not been flagged. Additional configuration options are available with the `solvemode_scalar_bandpass` parameter in `picard/input/array_finetune.inp`, e.g., one can choose to not obtain solutions scan or to skip the scalar bandpass entirely.

2. What is the difference between the `VLBAhi` and `VLBAlo` options for the `array_type` input parameter?
   When `VLBAhi` is selected, an additional opacity correction is performed (section 16). For `VLBAlo`, the natively measured system temperatures are applied directly. I recommend to use `VLBAhi` for observing frequencies above 15 GHz, and `VLBAlo` below.

3. What is the difference between the `input` and `input_template` folders?
   This is described in section 3 and section 7.

23 Known Issues

1. Issue: CASA aborts with the message that a ~/.matplotlib path does not exist or that a ~/.matplotlib/tex.cache file does already exist.
   Solution: Start CASA or `rPICARD` again.

2. Issue: ‘UnboundLocalError’ when initializing the metadata collection.
   Possible solution: Verify that all sources set in the `observation.inp` input file are actually present in the data, especially when working with a subset of the data created with the CASA `split()` task.

3. Issue: ‘IOError: Too many open files’.
   Context: In the CASA MPI implementation, the MS is split into several sub-MSs. The number of sub-MSs created are limited by the number N of files that can be opened at the same time (c.f. N = $ ulimit -n) when the data is loaded. If N changes (data copied to different system or someone changed the system settings), the above IOError can occur.
   Solution: Raise N to the number used when the data was loaded.

4. Issue: ‘fatal: Not a git repository: ...’.
   Solution: Use git pull instead of downloading a tarball of the repository.

5. Issue: ‘no array in row x of column MODEL_DATA’.
   Context: This can happen when a source model is used. The CASA MPI implementation seems to handle source models incorrectly in some cases.
   Solution: Load the data again after setting `MS_partitioning = None` in `picard/input/constants.inp`.

6. Issue: `rPICARD` aborts with a message about ‘ORTE was unable to reliably start one or more daemons...’
   Solution: Verify that your `input/mpi_host_file` is correct and that all servers can be reached.

7. Issue: ‘Exception: MPI is not enabled’.
   Solution: Run `rPICARD` with *-n N_cores*, where N_cores ≥ 2.

8. The pipeline starts in a home folder of a different machine and may complain that no input files can be found.
   Solution: 1) Verify your `mpi_host_file` (section 4). Is the current machine that you want to use in there?
   Or 2): Run `rPICARD` with *-n N_cores*, where N_cores ≥ 2.

9. Issue when using docker or singularity: ‘All nodes which are allocated for this job are already filled.’
   Solution: Run `rPICARD` with *-n N_cores*, where N_cores ≥ 2.

10. Issue: Docker complains about permissions or other obscure errors.
    Solution: Use root-less docker without root/sudo. It is available starting with docker version 19.03.

11. Issue when using docker or singularity: 'KeyError: 'getpwuid(): uid not found'
    Context: This can happen when there is a mismatch between user information on the local system and docker.
    Solution #1: Use singularity instead of docker.
    Solution #2: Also pass *–env HOME=/data –user $(id -u) -v /etc/passwd:/etc/passwd* to *docker run* (see `README`).
    If that does not work (can happen for macOS):
    Solution #3: It could be that the user information is not actually stored in the `/etc/passwd` file on your local system. For macOS, user information can be retrieved with the *dscl* or *nidump* commands and a custom `passwd` file can be created with the information about the user. That file can then be shared with docker by using *-v passwd:/etc/passwd* instead of *-v /etc/passwd:/etc/passwd* for *docker run*.

12. Issue when using singularity for `rPICARD` or for the generation of diagnostic plots with jplotter: 'Unknown image format/type'.
    Solution: Run the plotting command again. If that does not help, upgrade your singularity installation to version 3.x.x.

13. Issue: When processing data with docker or singularity containers, software from the host system, as specified in the PATH, are used instead of the software from inside the containers.
    Solution: Set the environment variable *PYTHONNOUSERSITE=1*.

14. Issue: Trying to plot the solutions in a calibration table does not work and this message is printed: 'Note: Either your CalTable pre-dates name-based selection, or does not (yet) support selection, or the MS associated with this cal table does not exist. All antennas, fields, spws are being selected for plotting.'
    Context: Calibration tables are directly associated with the MS that they were obtained from.
    Solution: You will need to retain the original data structure on your file system. By default this is a working directory where the MS is present and the calibration table resides inside a `calibration_tables/` subfolder.

15. Issue: Antenna names are 'unknown' in an exported UVFITS file.
    Context: Sometimes the CASA exportuvfits() code messes up station names.
    Solution: Run the export data step again.

16. The flux density calibration gains and/or system temperatures from an ANTAB table are not applied [to some antennas].
    Context: The formatting restrictions for ANTAB tables are more strict for CASA

compared to AIPS.
Solution: Make sure that the ANTAB format follows exactly the file format exemplified in section 12.

17. Issue: 'TypeError: nan_to_num() takes exactly 1 argument (2 given)', 'Invalid rgb arg', or a raised AttributeError about a colormap when `rPICARD` is generating diagnostic plots.
Solution: Upgrade to the CASA version given in the README file.

18. Issue: Python OS Errors occur and no new terminals can be opened.
Context: When *mpicasa* is killed, zombie /tmp/CASA_MPIServer processes can be left over.
Solution: Verify that no instance of `rPICARD` is running and kill all zombies with $ *pkill -f /tmp/CASA_MPIServer*.

19. Issue: The imager of `picard/interactive_utils.py` (section 10) crashes with errors related to the `tclean` task (e.g., 'TypeError: Exception from task_tclean').
Context: There may be compatibility issues with newer versions of CASA.
Solution: Try again with an older CASA version, for example CASA 5.6.

23.1 Subarrays

Subarrays are not yet fully supported in CASA. If scans overlap in time, the calibration solutions from one scan can influence the other. In this case, the recommendation is to split out the antennas from the different subarrays into new measurement sets and process these individually. If necessary, a join phase calibration can be done beforehand on overlapping scans for calibration purposes. The phae calibration table should be applied to the data and the corrected data should then be split out.

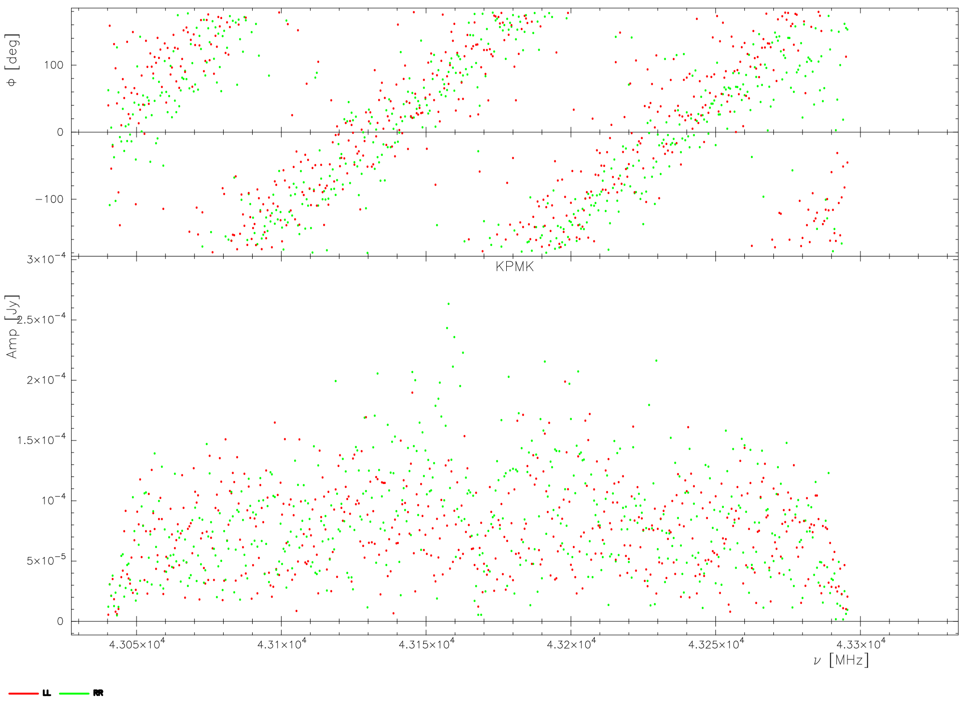
24 Publications – for Detailed Information

- Paper describing the pipeline in detail:
  https://ui.adsabs.harvard.edu/abs/2019A%26A...626A..75J.
  The DOI is https://doi.org/10.1051/0004-6361/201935181. This paper contains plots which illustrate the secondary reference antenna selection for the exhaustive fringe search (subsection 11.1), the fringe-fit solution interval optimization search (subsection 11.2), and the opacity correction (section 16) for example. Please cite this paper when you are using the pipeline. The software is also on ASCL: https://ascl.net/1905.015.

25 Comparisons with AIPS-based Calibration

Appendix A of https://ui.adsabs.harvard.edu/abs/2019A%26A...626A..75J shows some comparisons between an `rPICARD`-based calibration and a calibration using standard AIPS calibration methods. Here, some additional information regarding this comparison is given.

Figure 2 shows examples of uncalibrated data from the VLBA BW0106 M87 experiment. Uncorrected post-correlation delays are clearly visible. Figure 3 shows the same data after calibration by `rPICARD`. Edge channels are flagged, delays are taken out, bandpasses are flattened, and the phase scatter has been reduced as atmospheric phase variations have been calibrated. There is still some phase scatter present in the Owens Valley – St. Croix baseline due to the SNR-limited timescales on which intra-scan phase variations can be corrected for. The low SNR is also reflected in the amplitudes which are affected by decoherence from the scan-average. The data corresponding to Figure 3 after the standard AIPS calibration is shown in Figures Figure 4 and Figure 5, separately for the RR and LL. The AIPS calibrated data is in good agreement with the `rPICARD` calibrate data.



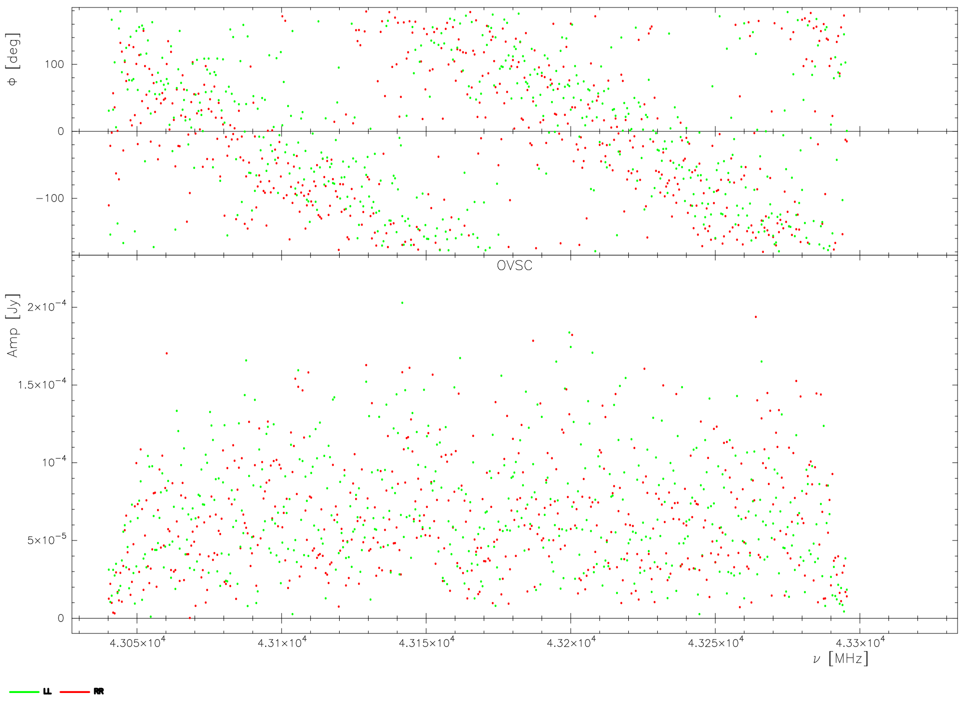
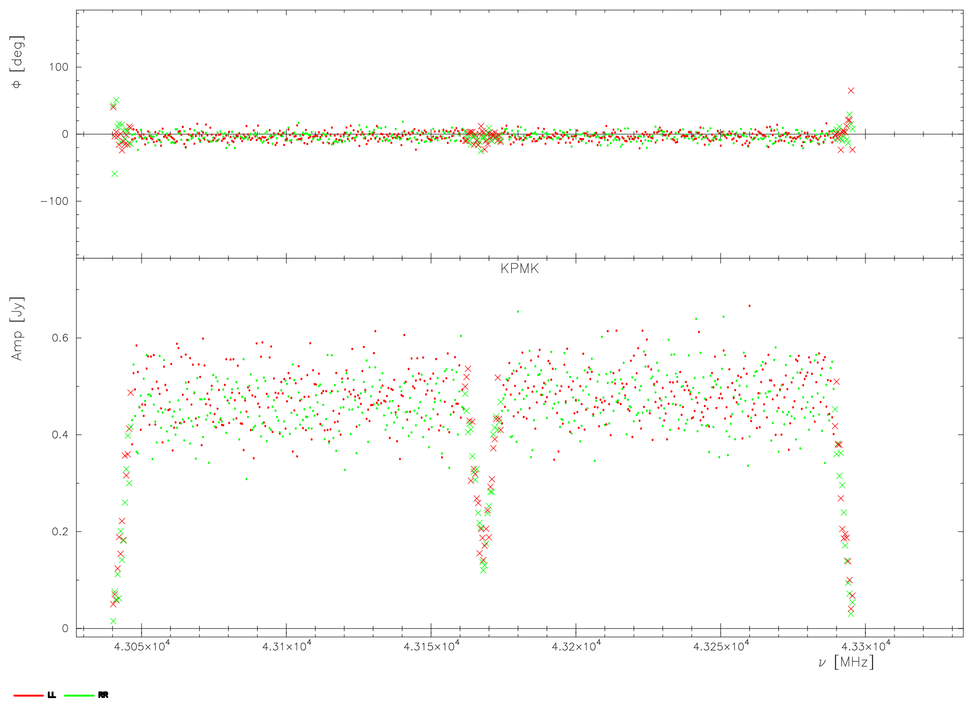


Figure 2: Unprocessed raw data of M87. Scan-averaged phases and amplitudes are shown for the two spectral windows in the dataset as a function of frequency for the parallel-hand correlations (color-coded) of two baselines (Kitt Peak – Mauna Kea in the top panel and Owens Valley – St. Croix in the bottom panel). The plots were made with jplotter as part of the diagnostics automatically generated by `rPICARD`.



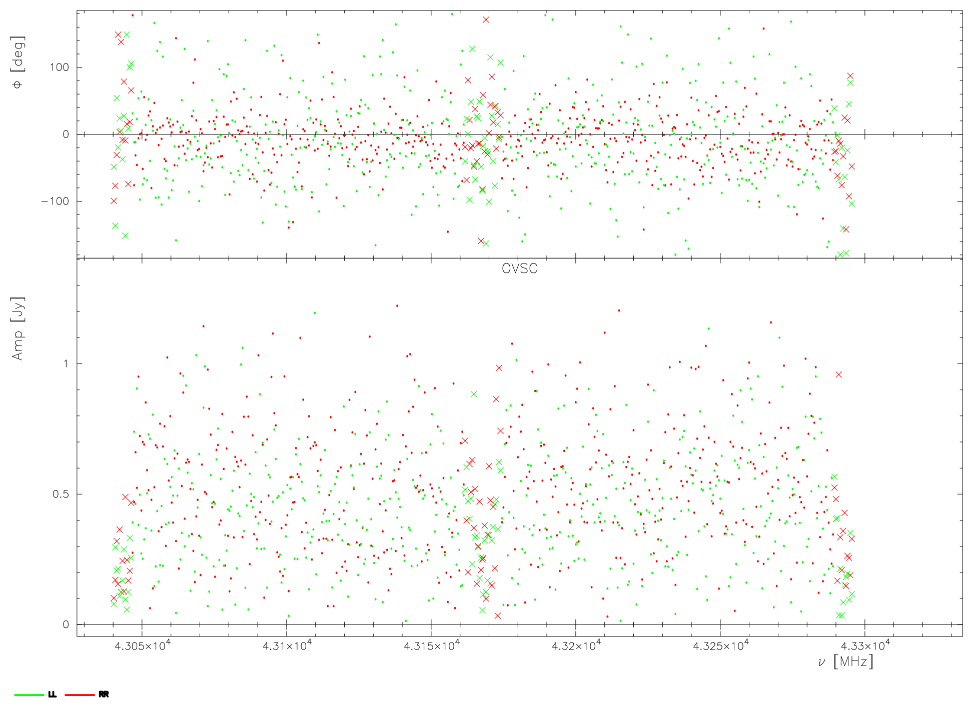
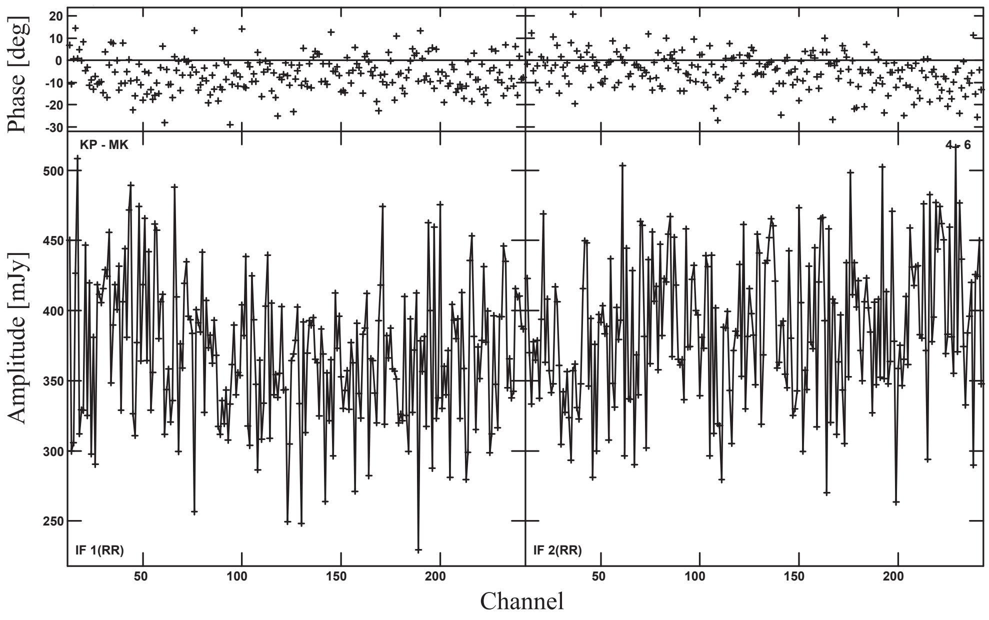


Figure 3: `rPICARD`'s pipeline processed data of M87. Scan-averaged phases and amplitudes are shown for the two spectral windows in the dataset as a function of frequency for the parallel-hand correlations (color-coded) of two baselines (Kitt Peak – Mauna Kea in the top panel and Owens Valley – St. Croix in the bottom panel). Crosses indicate flagged data. The plots were made with jplotter as part of the diagnostics automatically generated by `rPICARD`.



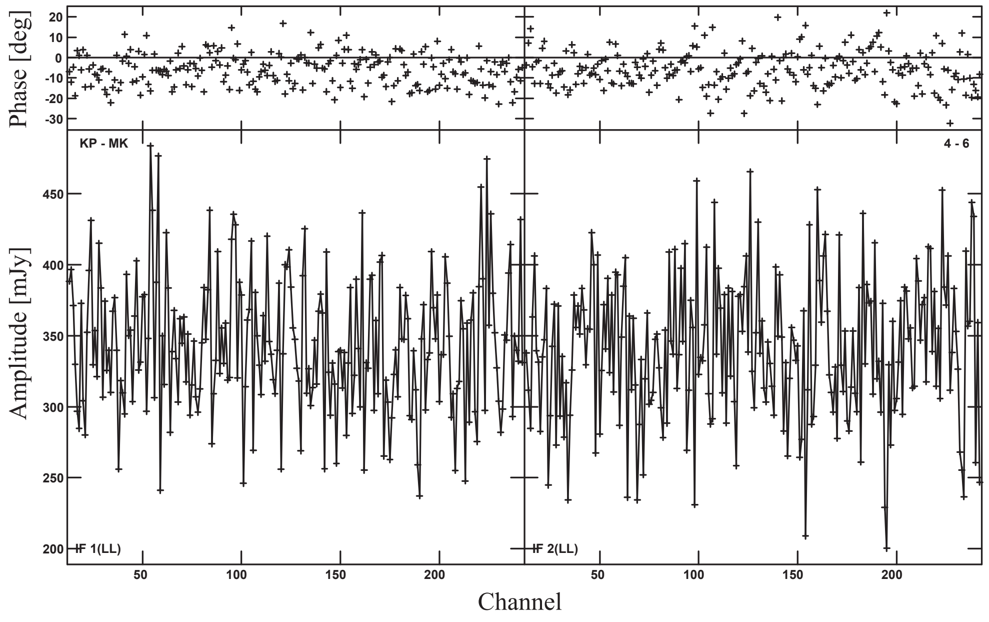
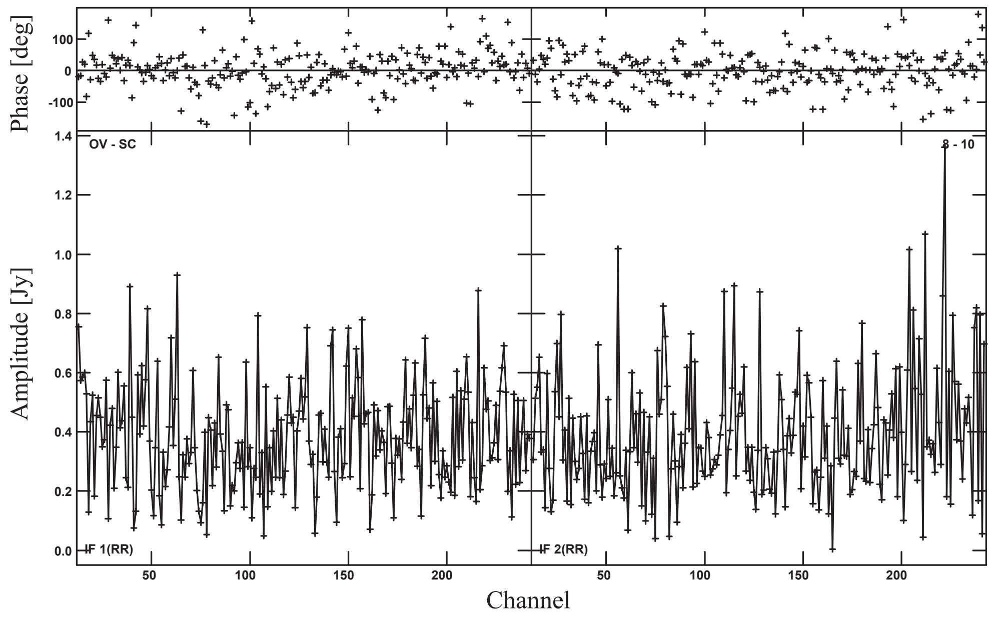


Figure 4: AIPS processed data of M87 corresponding to the data shown in the top panel (KP-MK baseline) of Fig. 3. The AIPS data is shown separately for the RR and LL correlations in the top and bottom panels respectively as a function of frequency channel here.



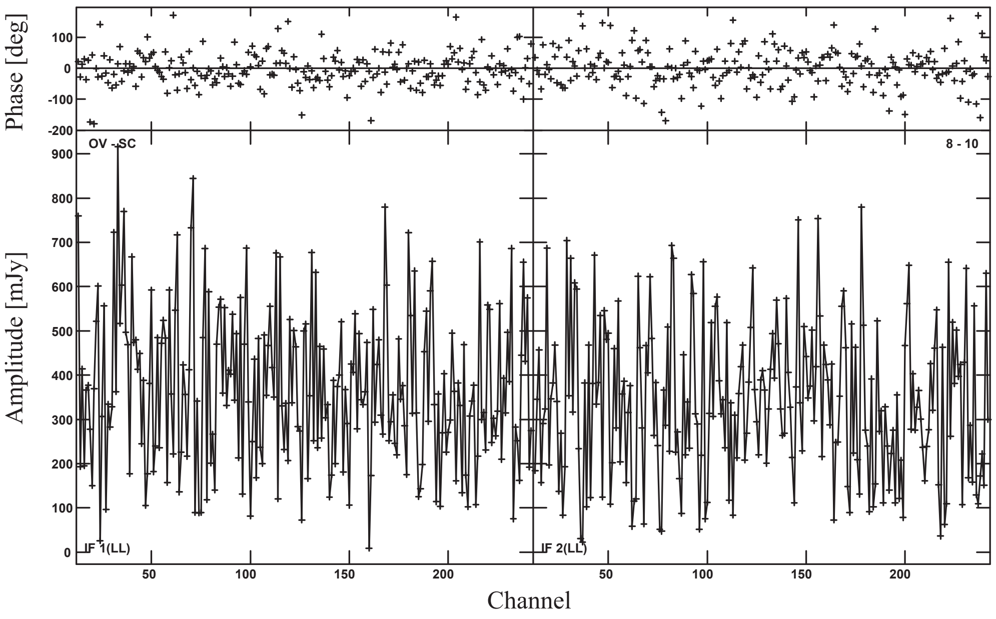


Figure 5: AIPS processed data of M87 corresponding to the data shown in the bottom panel (OV-SC baseline) of Fig. 3. The AIPS data is shown separately for the RR and LL correlations in the top and bottom panels respectively as a function of frequency channel here.