ska-sdp-prot-imaging-pipeline Documentation

Release 0.1.0-beta

See CONTRIBUTORS

Nov 23, 2022

CONTENTS

This project contains a prototype continuum imaging pipeline, which is based on the RASCIL continuum imaging pipeline (part of rascil_imager). It aims to provide a test-bed for integrating and testing accelerated processing functions from the Processing Function Library.

CHAPTER

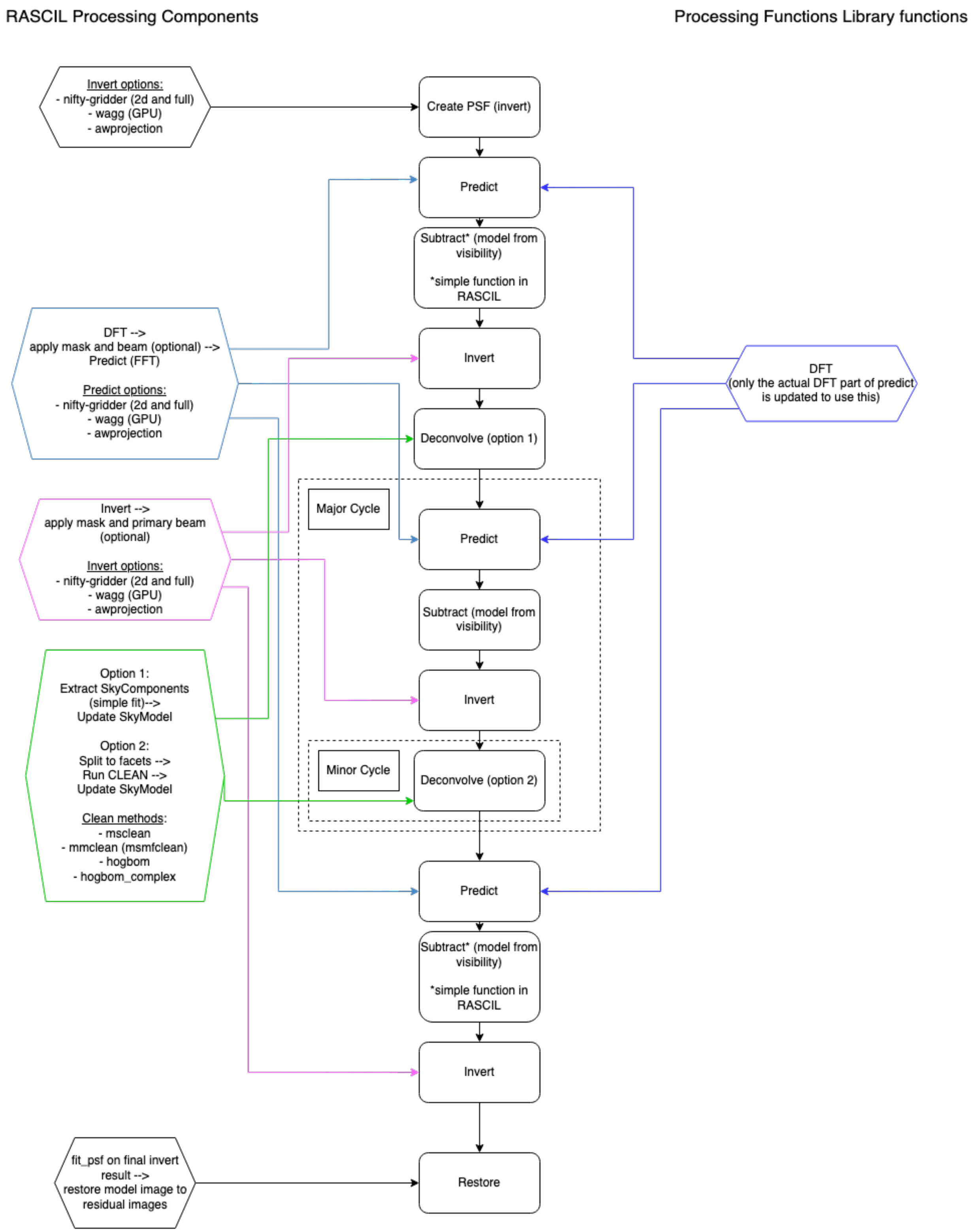
ONE

STRUCTURE OF THE PIPELINE

A discussion of the pipeline structure, including a diagram, can be found at the following Confluence page: Stand-alone imaging pipeline for processing function integrations. We keep updating the page and the diagram, and related pages, as more functions are integrated from the Processing Function Library.

The structure follows the one of the RASCIL Continuum Imaging Pipeline. This is the equivalent if the ICAL pipeline without the self-calibration part (i.e. `do_selfcal = False`).

The following diagram is taken from the Confluence page and it shows the state of the structure and the integration as of 12 May 2022. On the left, we list the relevant code information from RASCIL, while on the right the already integrated Processing Function Library equivalents are found.


RASCIL Processing Components
Processing Functions Library functions
Invert options:
- nifty-gridder (2d and full)
- wagg (GPU)
- awprojection
Create PSF (invert)
Predict
Subtract* (model from visibility)
*simple function in RASCIL
DFT -->
apply mask and beam (optional) -->
Predict (FFT)
Predict options:
- nifty-gridder (2d and full)
- wagg (GPU)
- awprojection
Invert
DFT
(only the actual DFT part of predict is updated to use this)
Deconvolve (option 1)
Major Cycle
Invert -->
apply mask and primary beam (optional)
Invert options:
- nifty-gridder (2d and full)
- wagg (GPU)
- awprojection
Predict
Subtract (model from visibility)
Invert
Option 1:
Extract SkyComponents (simple fit)-->
Update SkyModel
Option 2:
Split to facets -->
Run CLEAN -->
Update SkyModel
Clean methods:
- msclean
- mmclean (msmfclean)
- hogbom
- hogbom_complex
Minor Cycle
Deconvolve (option 2)
Predict
Subtract* (model from visibility)
*simple function in RASCIL
Invert
fit_psf on final invert result -->
restore model image to residual images
Restore

Currently, this structure is implemented using the *ContinuumImagingPipeline* class and tha main function in imaging_prototype.py.

CHAPTER

TWO

COMPONENTS USED FROM RASCIL

High-level functions are used from RASCIL, while data models are in ska-sdp-datamodels, which contains data models extracted from RASCIL. In addition, processing components (functions) are also taken from RASCIL, with the option of running the available ones from the Processing Function Library instead.

As we integrate more and more Processing Function Library functions the list below may change.

2.1 Data models

- **Visibility**:

  RASCIL's basic visibility model. It is an observation with one direction. We load data from a MeasurementSet into this object (or a list of them). It is based on the xarray.Dataset object.

- **Image**:

  In-memory representation of an image, with pixels as data variables, and the Astropy implementation of the World Coordinate System, which is stored in attribute format. It inherits from xarray.Dataset.

- **SkyModel**:

  Simple Python class. Has various methods, the most important ones are `component` and `image`. Former containing a list of SkyComponents, latter containing a model image (RASCIL Image object). At minimum, an image object is needed to initialize a SkyModel.

- **SkyComponent**:

  Represents a component source on the sky, with flux, direction, frequency, polarisation, etc.

2.2 Control functions and processing components

These functions coordinate other processing components and lower level functions. Where applicable, we list a set of arguments that can be controlled by the user when the prototype pipeline is executed, and another set that is either hard coded at the moment or the code simply uses some defaults defined by the functions that need them. These latter arguments may be promoted to user-defined ones if we see fit, however, at the moment we keep them as is for simplicity.

We note where in the *ContinuumImagingPipeline*, or in other prototype-pipeline functions each RASCIL function is used to give context to their usage.

Some of the functions below were migrated into a new package called ska-sdp-func-python during the autumn of 2022. The links point to this new repository.

- **create_visibility_from_ms**:

  Loads MeasurementSet (MS) data into one or more Visibilities.

  Used in `ContinuumImagingPipeline._load_bvis_from_ms`

User-controlled arguments:

- msname: name and path to MeasurementSet
- channels_in_data: how many frequency channels does the used data descriptor contain
- nchan_per_vis: how many channels we want to load into a Visibility;

Hard-coded arguments:

- selected_dds: which data descriptor to load from the MS; hard coded to [0]
- average_channels: whether or not to average the channels loaded into a Visiblity; hard coded to False

- **convert_visibility_to_stokesI:**

  Convert the BlockVisibility data to StokesI.

  Used in ContinuumImagingPipeline._load_bvis_from_ms

- advise_wide_field:

  Used in ContinuumImagingPipeline._init_model_images_list TODO

- **create_image_from_visibility:**

  Used to create the initial model image used for the SkyModel and as a model for the final FITS images. This function takes a lot of arguments in the form of kwargs, from which we hard-code nchan, the number of channels the output image should have. This is set to 1, because deconvolution (as is in RASCIL) needs images of a single channel.

  Used in ContinuumImagingPipeline._init_model_images_list

  User-controlled arguments:

  - npixel: how many pixels (on each side) we want our final images to contain;
  - cellsize: how big we want a cell to be in radians; default is calculated in function if not provided, defaults to being calculated.

  Hard-coded arguments:

  The function loads multiple arguments from kwargs if present, else it uses some defaults.

- **invert_visibility:**

  Invert a Visibility to make an (image, sum of weights) tuple.

  Used in ContinuumImagingPipeline._init_psf_list and ContinuumImagingPipeline.invert

  User-controlled arguments:

  - context: which imaging context, i.e. gridder/degridder to use

  Hard-coded arguments:

  invert_visibility takes various arguments, some if which maybe worth investigating and promoting to user-controlled ones

- **extract_direction_and_flux:**

  This function formats the data needed for the various DFT kernels / functions to run. It is wrapped with dft_visibility, which also allows for choosing between RASCIL's dft_kernel function or the Processing Function Library's dft_point_v00 function.

- **dft_kernel:**

  Choose and run a CPU or a GPU-based DFT kernel via RASCIL. It takes dft_compute_kernel argument to specify which kernel to use. It is wrapped with dft_visibility

  Hard-coded arguments:

  - `dft_compute_kernel`: determines which kernel to use; default is None, which reverts to `cpu_looped`

- **predict_visibility**:

  Predict Visibility from an Image.

  Used in `ContinuumImagingPipeline.predict`

  User-controlled arguments:

  - `context`: which imaging context, i.e. gridder/degridder to use

  Hard-coded arguments:

  > `predict_visibility` takes various arguments, some if which maybe worth investigating and promoting to user-controlled ones

- **imaging_subtract_vis**:

  Implemented directly in imaging_utils.py, but it is a copy of an inner function of RASCIL's subtract_list_rsexecute_workflow. It subtracts the model data from the input visibility data.

  Used in `ContinuumImagingPipeline.predict`

- **deconvolve_skymodel_list_rsexecute_workflow**:

  A high level function, called a 'workflow' in RASCIL. It controls the full process of deconvolution. A detailed breakdown and analysis of RASCIL's deconvolution can be found in Confluence: Deconvolution - detailed breakdown in RASCIL CIP

  This is wrapped with the deconvolution function, which also hard-codes some of the input arguments.

  deconvolution is used in `ContinuumImagingPipeline.deconvolve`

  User-controlled arguments:

  - `fit_skymodel`: whether to fit for the SkyComponents and update the SkyModel with the component list
  - `component_threshold`: sources with absolute flux > this level (Jy) are fitted and added to the SkyComponent lis; only used if `fit_skymodel == True`
  - `clean_threshold`: clean stopping threshold (Jy/beam); this translates to the `threshold` argument of the RASCIL workflow function.

  Hard-coded arguments:

  > - `component_method`: what method to use for extracting SkyComponents; hard-coded to `fit`
  > - `deconvolve_facets`: how many facets to break the image up before deconvolution; hard-coded to 1
  >
  > `deconvolve_skymodel_list_rsexecute_workflow` takes many more arguments in the form of `kwargs`. These will need investigating and potentially promoted to user-controlled arguments.

- **restore_skymodel_list_rsexecute_workflow**:

  High-level RASCIL workflow controlling the functionality of restoring the image. It takes various key-word arguments, none of which are controlled by the user at the moment. See function for a full list.

  Used in `ContinuumImagingPipeline.restore`

- **export_skymodel_to_hdf5**:

  Export a SkyModel or a list of it into HDF5 format. Used in export_results.

- **image_gather_channels**:

  Concatenates the Image objects along the frequency dimension. At the beginning, we created a model image per frequency channel; these are now merged into a single image to be exported into FITS files Used in export_results.

CHAPTER

THREE

PROCESSING FUNCTION LIBRARY INTEGRATION

3.1 Integration steps

TBC after Workshop with teams.

3.2 DFT

We have integrated the DFT (direct Fourier Transform) function. The logic to choose between this and the RASCIL version can be found in dft_visibility. One needs to specify the `dft_function` argument to determine which to use:

- `rascil`: to use the RASCIL version
- `proc_func`: to use the Processing Function Library version

Relevant unit tests have been added to test_processing_function_integration.py. These show that, using basic visibility data, the two versions produce the same results.

3.3 Future work

As new functions are added to the Processing Function Library, one can start integrating those with the pipeline. We encourage the writers of these functions to give it a go at the integration and test their code using the prototype pipeline.

CHAPTER

FOUR

RUNNING THE PROTOTYPE PIPELINE WITH DASK

Parallel run of the prototype imaging pipeline has been implemented using Dask.

Most of the functions are wrapped using a simple decorator, which decides whether to compute using dask.delayed or run the code in serial. This is controlled by the `use_dask` user-defined argument (see *Command Line Interface to run the pipeline*).

A few of the high-level functions directly imported from RASCIL use RASCIL's own class that wraps its objects with Dask. These are the so called "workflows", e.g. deconvolve_skymodel_list_rsexecute_workflow and restore_skymodel_list_rsexecute_workflow. They inherently use the rsexecute class, which we need to initialize in the main function of imaging_prototype.py.

CHAPTER

FIVE

COMMAND LINE INTERFACE TO RUN THE PIPELINE

The main function of the pipeline can be found in imaging_prototype.py.

Prototype imaging pipeline

```
usage: imaging_prototype.py [-h] --input_ms INPUT_MS [--output_dir OUTPUT_DIR]
                            --input_nchan INPUT_NCHAN
                            [--nchan_per_vis NCHAN_PER_VIS]
                            [--n_major N_MAJOR]
                            [--imaging_context IMAGING_CONTEXT]
                            [--imaging_npixel IMAGING_NPIXEL]
                            [--imaging_cellsize IMAGING_CELLSIZE]
                            [--component_threshold COMPONENT_THRESHOLD]
                            [--clean_threshold CLEAN_THRESHOLD]
                            [--processing_func PROCESSING_FUNC]
                            [--use_dask USE_DASK]
```

5.1 Named Arguments

--input_ms MeasurementSet to be read (including path to directory)

--output_dir Directory where output files should go.Default is the one where program is executed from.

Default: "."

--input_nchan Number of channels in a single data descriptor in the MS.Note: we don't allow specifying a list of data descriptor,instead we always load [0] from the MS.

--nchan_per_vis How many channels per Visibility to read in

Default: 1

--n_major How many major cycles to run

Default: 1

--imaging_context What gridding context to use: ng (nifty-gridder) | 2d | awprojection | wg (WAGG for GPU only)

Default: "ng"

--imaging_npixel Number of pixels in ra, dec: Should be a composite of 2, 3, 5

Default: 512

--imaging_cellsize Cellsize (radians). Default is to calculate.

--component_threshold Sources with absolute flux > this level (Jy) are fitted using SkyComponents

Default: 1.0

--clean_threshold Clean stopping threshold (Jy/beam), note that this is different from component_threshold

Default: 1.0

--processing_func Which processing functions to use: 'rascil': RASCIL-based processing components'proc_func': accelerated functions from the Processing Function Library

Default: "rascil"

--use_dask Whether to run the computation with Dask.delayed or not

Default: "False"

CHAPTER

SIX

CONTINUUM IMAGING PIPELINE PYTHON CLASS

class `src.imaging_prototype.`**`ContinuumImagingPipeline`**(*imaging_context*, *use_dask=False*)

Prototype Continuum Imaging Pipeline. Doesn't use calibration.

`_init_model_images_list`(*n_pixel*, *cell_size=None*)

Initialize model images from input visibility data

Note: According to RASCIL Continuum Imaging Pipeline,
each model image needs to have a single frequency channel. This will make sure that when we run invert with ng, the resulting Image list will have a channel each and, hence deconvolution won't break (which expects 1 chan / Image).

Parameters

- **`cell_size`** – image cell size [rad]; if None, it is calculated
- **`n_pixel`** – number of pixels on a side of the image

`_init_psf_list`()

Create Point Spread Functions (PSF)

`_init_skymodel_list`()

Initialize SkyModel for each model image (Note: we may need to allow it as input too, in the future)

`_load_bvis_from_ms`(*input_ms*, *channels_in_data*, *nchan_per_vis*)

Load MeasurementSet data into Visibility objects. (rascil.data_models.memory_data_models.Visibility)

Note: - Polarization of data is always converted to Stokes I after loading. - Based on rascil.apps.rascil_imager.get_vis_list and

rascil.workflows.rsexecute.visibility.visibility_rsexecute.create_visibility_from_ms_rsexecute

- these RASCIL functions take multiple arguments, most of which we hardcode here. However, in the future, we may need to allow for users to specify these.

Parameters

- **`input_ms`** – path to input MeasurementSet
- **`channels_in_data`** – how many frequency channels does the data set contain
- **`nchan_per_vis`** – how many frequency channels to load into a single Visibility

`deconvolve`(*fit_skymodel*, *component_threshold*, *clean_threshold*)

Run deconvolution.

Parameters

- `fit_skymodel` – True: fit the skymodel and extract sky components False: run CLEAN-based deconvolution
- `component_threshold` – Sources with absolute flux > this level (Jy) are fitted
- `clean_threshold` – Clean stopping threshold (Jy/beam)

Returns

updates self.skymodel_list in place

`invert`()

Run the invert step.

Returns

updates self.dirty_image_list in place

`predict`(*processing_func_source*)

Run the predict step, including subtracting the predicted model data from the input data.

Parameters

`processing_func_source` – which type of processing functions to use; Options: 'rascil' - use RASCIL

'proc_func' - Use the Processing Function Library

Returns

updates self.bvis_list in place

`restore`()

Restore images.

Uses RASCIL's restore_skymodel_pipeline_rsexecute_workflow function, which implicitly uses rsexecute to run with Dask.

INDEX