CLI-Combine

Functional Specification
Elias Rubin
Last updated: July 15, 2016

Overview

Cli-Combine is a command-line tool to combine images from ALMA's 7 meter and 12 meter telescopes. The aim is to make it easy for astronomers to view various combinations of their images by reducing the combination process to filling out a single configuration file and running a few shell commands.

Non-goals

Cli-Combine will not aim to do continuum subtraction as I believe this step still requires line identification and some careful manual reduction.

It will also not do masking for the combined data. Implementing a good automasking routine could be a good project for another intern or me later (the automasking routine on the M100 docs does not seem to work very well.)

Scenarios

Scenario 1: Jim
Jim is an astronomer who is mainly focused on his science goals. He wants to create a multi-channel image of NGC 1068 to look at the H-alpha line in the center of the galaxy. He has continuum subtracted data from the 7 meter and 12 meter arrays. Jim has never done image combination before. He types in the terminal

```
$ combine --help
```

and gets a help menu describing the process of creating a new project, filling out a configuration file, and creating his combined images. Jim follows the instructions on the help menu and creates a new project by typing

```
$ combine --new NGC_1068
```

Jim fills out the configuration file `NGC_1068_combine.json` in his favorite text editor with the information required for his science goal. In his first pass, he leaves all of the technical parameters as `auto`. He then creates his combined images by typing

```
$ combine --combine NGC_1068
```

Jim decides to view his uv-combined data first, by typing

```
$ casaviewer NGC_1068_uv_combined.image
```

Although he is happy with the uv-combined image, he also wants to look at the feathered image. He types

```
$ casaviewer NGC_1068_feather_combined.image
```

Jim likes the feathered image and exports it as a FITS file from casa. He puts the image on his personal website so he can show all his friends how cool he is.

Scenario 2: Sonja
Sonja is interested in getting into the nitty gritty of her image combination to make the best images possible. She wants to combine data from the center of the Milky Way. She starts a new project by typing

```
$ combine --new SgrAstar
```

She fills out the configuration file `SgrAstar_combine.json` in her favorite text editor. She starts by filling out the scientific parameters set to examine the HCN line. She leaves the technical parameters as `auto` and runs

```
$ combine --combine SgrAstar
```

Although the output images look ok, Sonja thinks that they are a little cloudy. An experienced astronomer, she realizes that the uv cleaning was not performed deeply enough. She returns to `SgrAstar_combine.json` and changes the `threshold` parameter from `auto` to `0.015Jy`. She then runs

```
$ combine --combine SgrAstar
```

again and then opens `SgrAstar_uv_combined_v2.image`. She is satisfied with the lack of cloudiness. Sonja now wants to look at the zeroeth and first moments of SgrAstar. She returns to `SgrAstar_combine.json` and changes the `moments` parameter from `none` to `[0, 1]`. She then runs

```
$ combine --combine SgrAstar
```

once more and now has moment information. Sonja now wants to re-run the process for the HCO+ line. She copies the config file she used before by typing

```
$ cp SgrAstar_combine.json SgrAstar_HCOp_combine.json
```

and changes the `restfreq` parameter. Then she runs

```
$ combine --combine SgrAstar_HCOp
```

and generates new combined images in the HCO+ line.

Interaction

From reading the scenarios you probably have a basic idea of how users can interact with `combine`. We'll go through each of the options in detail later, but we'll begin with a quick summary.

`combine --help` displays a list of commands and what they do as well as a description of the config file.

`combine --example` creates a populated example config file.

`combine --new` creates a new empty config file.

`combine --combine` takes a config file as an argument and produces combined image results based on the parameters specified.

Adding the `-v` option enables verbose logging to `stderr` which can be redirected to a file. The `--overwrite` flag will overwrite old combined images and create a new one with the same filename. Otherwise, new images will have a version number.

Command-by-Command Specification

–help

The `--help` command should print a help screen to the terminal. This screen should have

- A basic list of all commands and options and their usage.
- An example workflow for the program.
- Descriptions of each parameter in the config file.

Ideally, `$ combine --help` should produce the same output as `$ man combine`.

Technical note: perhaps the help screen should be piped to `less` and displayed that way.

Design note: Would it be better to have the descriptions of the config parameters inside the config file?

–example

The `--example` command should create an `example.json` file that has a filled-out configuration file. For reference, we might as well use the configuration information that we used to do the manual image combination. This way will also help us compare the output of the manual combination to the combination program's output (they should be the same!)

–new

The `--new` command takes one argument - the name of the project - and generates a new, empty configuration file.
The configuration file should contain **at least** all of the following (note: this may not yet be a complete list of parameters)

Technical note: I have included type annotations (denoted by `::`) for each of the parameters. Although JSON is untyped, the program should fail and print a descriptive error message in the `--combine` command if any of the parameters are of the wrong type at runtime when the JSON is parsed. Where an option type (delimited by `|`) is specified, the program should try the types in the order listed.

- `twelve_meter_filename :: str` – no default
- `seven_meter_filename :: str` – no default
- `output_filename :: str` default `auto`
- `weightings :: tuple<float> | list<float> | dict<string, float>` default `auto` or `(1.0,1.0)`
- `mode :: str` – no default
- `imagermode :: str` – no default
- `spw :: str` – default `auto`
- `field :: str` – default `auto`
- `outframe :: str` – default `auto` or `lsrk`
- `imsize :: int | list<int>` – default `auto`
- `cell :: int | float | str` – default `auto`
- `phasecenter :: str` – no default
- `robust :: float` – default `auto` or 0.5
- `restfreq :: float | str` if float default units are GHz
- `start :: float | str` if float default units are km/s
- `width :: float | str` if float default units are km/s
- `nchan :: int`
- `thresh :: float | str` if float default units are jy/beam. If `auto` the combine task will attempt to determine a threshold.
- `produce_feather :: boolean` default `true`
- `produce_uv :: boolean` default `true`
- `moments :: list<int> | tuple<int>` default `auto`

Design note: Should parameters with default values or behavior be left as `auto` or should default values be specified explicitly? Leaving them as `auto` would seem to allow us to change the implementation without changing the `--new` command but perhaps making any changes explicit to users would be a good thing.

–combine

The `--combine` command is the meat of the program. When used correctly, the `--combine` command should chug along silently (unless logging is enabled with `-v`) and perform the image combination. The user should be able to go from just the separate 12 meter and 7 meter data files and a completed configuration file to outputted combined files in a single step by running

```
$ combine --combine [config_file_name]
```

Functionally, this means (assuming both `produce_uv` and `produce_feather`):

- parsing and verifying the input file
- performing concatenation unless a combined `.ms` file is present
- cleaning the concatenated ms in the uv plane
- imaging the concatenated ms as specified in the config file
- cleaning the individual ms separately
- feathering the cleaned individual ms together

Failures

`combine` should fail as early and as quickly as possible. Errors in parsing the config file should be accompianied by a descriptive message. For example:

```
$ combine --combine SgrAstar

Error: you wrote 'start: 200kn/s' in 'SgrAstar_combine.json'
We had  trouble parsing 'kn/s'.  Did you mean 'km/s'?
```

Errors in the other stages are probably going to be generated by CASA. If we can include helpful messages with those errors, great, but I wouldn't bet on it.

Technical note: failing quickly means that all parsing should be done **before** loading CASA. That way, we can allow users to quickly iterate on constructing their config files in case they struggle with creating an input for some reason.