Keck Cosmic Web Imager DRP

Release 1.0

Jul 16, 2021

Contents

KCWI_DRP is a the official data reduction pipeline (DRP) for the Keck Cosmic Web Imager. This DRP has been developed by the KCWI team at Caltech in collaboration with the W. M. Keck Observatory Scientific Software Group.

CHAPTER 1

Release 1.0

We are happy to announce the first official release of the KCWI pipeline. This pipeline is based entirely on the glorious IDL pipeline developed by the KCWI team at Caltech and it is the only pipeline supported by the W. M. Keck Observatory.

1.1 What this version provides

- Simplified installation via pip and conda environment
- Vacuum to air and heliocentric or barycentric correction (the algorithms used here are courtesy of Yuguang Chen at Caltech)
- Ability of using KOA file names or original file names
- Better provenance and traceability of DRP versions and execution steps in the headers
- Versatile sky subtraction modes including using external sky frames and ability of masking regions
- Formal support system via GitHub issues

The pipeline is available for use at WMKO. We are in the process of automating the execution and ingestion of the reduced data into KOA.

For older versions, see *Previous versions*.

CHAPTER 2

Users

Follow the documentation in this section to reduce KCWI data.

2.1 Installing KCWI_DRP

This document describes how to install KCWI_DRP, for both users and developers.

2.1.1 Installing Dependencies

We highly recommend that you use Anaconda for the majority of these installations.

Detailed installation instructions are presented below:

Installing with environment.yml

An environment.yml file is provided here which contains the majority of the required dependencies. To create the conda environment, download the environment file and run

```
conda env create -f environment.yml
conda activate kcwidrp
pip install kcwidrp
```

This creates an environment called kcwidrp that contains most of the required dependencies.

Installing Manually

This pipeline currently runs on python 3.7. Instructions for installing the other dependencies are below:

```
conda install bokeh
conda install -c conda-forge selenium geckodriver firefox phantomjs
conda install -c astropy ccdproc pyregion
conda install psutil
conda install requests
conda install pytest
conda install cython
conda install pandas
pip install ref_index
pip install keckdrpframework
pip install kcwidrp
```

Installing for Development

If you want to alter the pipeline, you can install it directly from source by skipping `pip install kcwidrp` during the requirements section above, and instead running:

```
git clone https://github.com/Keck-DataReductionPipelines/KCWI_DRP.git
cd KCWI_DRP
python setup.py develop
```

2.2 Quick Start

For users who don't want to know too much about KCWI_DRP, here is a quickstart guide.

The assumption is that you have a directory containing KCWI data, and that the names of the files are those assigned at the telescope, `kb*.fits`.

Give a quick look at the configuration parameters for the pipeline, contained in the `kcwidrp/config/kcwi.cfg`.

For a quick start, it is enough to decide if you want to see plots as the pipeline runs or not. This is controlled by two parameters: `enable_bokeh` and `plot_level`. For no plotting, disable Bokeh and set the plot level to 0.

Next, go to the data directory and run the startup script:

```
cd mydata
reduce_kcwi -f kb*.fits
```

The Keck DRP Framework will load and initialize the pipeline, ingest all the files, and then start processing in the order in which they appear on disk.

Two directories will be created: a `redux` directory with the results of the reduction, and a `logs` directory with separate logs for the framework itself and for the DRP.

2.3 Configuration Parameters

A number of reduction parameters can be changed using entries in the configuration file.

If you installed the pipeline with `pip`, the configuration file will not be easy to find, since it will be stored with installed `pip` packages. If you need to modify the configuration, we advise that you download a copy of the config file, saving it to some easy to remember location, and using the `-c` option to point to that file.

If you installed the package via `git`, the master copy of the configuration file is in the installation directory, in `kcwidrp/config/kcwi.cfg`. This file can be modified in place or copied in another directory (the `-c` option of the main reduction script is used to specify the configuration file).

The configuration file contains a number of parameters connected to the structure of the files and to the specifications of the instrument, e.g. the number of continuum bars. There is usually no reason to modify these parameters, and they are not described here.

The remaining parameters are used to control the processing algorithms and are described here.

2.3.1 Processing parameters

```
output_directory = "redux"
```

This parameter specifies the output directory for the data products. If this directory is missing, it will be created automatically.

```
bias_min_nframes = 7
flat_min_nframes = 6
dome_min_nframes = 3
twiflat_min_nframes = 1
dark_min_nframes = 3
```

These parameters control the minimum number of bias, internal/dome/twilight flats and darks that the DRP expects before producing a master calibration. The values shown here are synchronized with the calibration scripts that we use in the afternoon.

```
clobber = False
```

This parameters controls the behaviour of the DRP if one of the data product has already been generated: set `clobber = True` to overwrite existing products.

```
skipscat = False     # Skip subtracting scattered light?
```

This parameter disables the subtraction of scatteres light if set to `True`. In some case the subtraction of scattered light can produce unexpected results.

```
default_arc_lamp = 'ThAr'
```

KCWI has two calibration lamps, Thorium/Argon (ThAr) and Iron/Argon (FeAr). This parameter specifies which of the two lamps should be used by the DRP. The default is to use the ThAr lamp.

2.3.2 Plotting parameters

```
plot_pause = 1
saveintims = False
inter = 1
plot_width=1000
plot_height=600

# BOKEH SERVER
enable_bokeh = False
plot_level = 0
```

These parameters control the plotting features of the DRP. Plotting is performed using a combination of a Bokeh server running in the background and a browser front end.

To activate the plotting features, set `enable_bokeh = True`. When the DRP starts, it will check if there is an instance of the Bokeh server running or start one. A browser window will be opened automatically when needed.

The `plot_level` parameter controls the level of interactivity. Setting it 0 will disable interactive fetures: the DRP will produce plots when needed but it will not interact with the user. A higher level will increase both the verbosity and the interactivity of the plots. The highest level is 3 (CHECK). At this level, the user will be provided with a plot of every arc line, for example, with a graphic representation of the fitting used to determine the central position.

For general use, it is advisable to leave the plot level to 1.

The `plot_pause` parameter controls how long the DRP will pause between automatically generated plots (in seconds). Finally, the `saventims` parameter controls the generation of JPG diagnostics plots saved in the current directory.

The size of the plotting window can be specified using `plot_width` and `plot_height`.

2.3.3 Cosmic rays rejection parameters

```
CRR_MINEXPTIME = 60.0
CRR_PSSL = 0.0
CRR_GAIN = 1.0
CRR_READNOISE = 3.2
CRR_SIGCLIP = 4.5
CRR_SIGFRAC = 0.3
CRR_OBJLIM = 4.0
CRR_PSFFWHM = 2.5
CRR_FSMODE = "median"
CRR_PSFMODEL = "gauss"
CRR_SATLEVEL = 60000.0
CRR_VERBOSE = False
CRR_SEPMED = False
CRR_CLEANTYPE = "meanmask"
CRR_NITER = 4
```

These parameters are used to control the CRR algorithms. See the documentation in astroscrappy for details (PROVIDE LINK)

2.3.4 Wavelength correction parameters:

The `radial_velocity_correction` parameter controls what reference frame to use for radial velocity corrections. The options are `heliocentric`, `barycentric`, or `none`

The `air_to_vacuum` parameter controls if the pipeline should convert to vacuum wavelengths from air wavelengths.

2.4 Running the pipeline

The DRP is instantiated using a startup script offers contains several command line options and different execution modes.

2.4.1 Process files, file lists and entire directories

- To reduce all files in a directory in the order in which they appear and group them correctly according to the logical sequence needed by the pipeline:

```
reduce_kcwi -f kb*.fits -g
```

- To reduce only a subset of the files (for example, a single object out of an entire observing run):

```
reduce_kcwi -l input_files.lst
```

- To reduce a single file:

```
reduce_kcwi -f kb001.fits
```

For the "list" and "single file" cases, please note that the preliminary calibrations needed to reduce the file or the file list must be already on disk. For example, if the file you are trying to reduce is a science frame, the wavelength calibration, flat field and bias frames must already exist.

Reducing a single file is a good way to re-reduce a target for which the pipeline didn't do a good job. An example case would be the sky subtraction: if you realize that the sky subtraction is not correct, and you modify the sky subtraction using an external file, then you can rerun the pipeline just on the trouble file. In this case it is advisable to use the `clobber=True` option.

The following option can be used to modify the behaviour of the DRP when processing files in a directory:

`--groups` This options forces the DRP to group the files by image type and then processes them in the correct order so that master calibrations are produced before processing science images, regardless of the order in which the files appear on disk or are specified in the input list. This is also accessible with the `-g` flag.

2.4.2 Running individual steps

This is not a standard operating mode and is not supported but it is possible. The pipeline has a number of high level procedures that determine the correct order for data reduction.

If you know what you are doing, you can in principle create individual file lists for the steps.

Example:

Create a file containing bias frames and call it `bias.lst`. You can can run

```
reduce_kcwi -l bias.lst
```

Because the files are all and only bias, the pipeline will only proceed as far as generating the master bias.

For an even finer control, each file in the bias list could be run individually, and only when enough bias frames are present and reduced, the DRP will generate a master bias.

2.4.3 Monitor directories

The DRP has the ability of monitoring a specified directory. When files appear, they are ingested and processed. To start the DRP in this mode use:

```
reduce_kcwi -d /home/mydata -i kb*.fits -m
```

The `-i KB*.fits` (or `-i kb*.fits` if you are using OFNAME files) is the filter used to recognize the correct files. If it is not specified the pipeline will ingest all files appearing in a directory, and might fail if those files are not KCWI frames.

2.4.4 Other command line options

- `-c config_file.cfg` This options overrides the standard configuration file that is stored

in the installation directory in `kcwidrp/config/kcwi.cfg`.

- `-p proctable.proc` When the DRP runs, it keeps track of the files processed using

a processing table. Normally that table is called `kcwi.proc` and is stored in the current directory. This options is used to specify a different file if needed.

- `-t taper_fraction` TBD

2.5 Sky Subtraction

The pipeline performs automatic sky subtraction by identifying areas of the sky that do not contain objects. If this fails, it is possible to modify the sky subtraction algorithm in two ways: (1) by using a different frame as the sky frame and (2) by specifying areas of the target or sky frame to exclude from the calculation of the sky (because they are contaminated by an object, for example).

To specify the preferred method for sky subtraction, create a file in the data directory called `kcwi.sky`. For each KCWI frame for which you want to modify the sky subtraction algorithm, enter a row with this format:

```
raw_object_file.fits raw_sky_frame.fits <mask.fits>
```

If you want to specify an external sky frame, only use the first two columns and do not specify a mask file.

If you want to use a mask on the original file, make sure that the first two columns contain the *same* file name and add the mask file.

2.5.1 Building a mask file

To build a mask file based on `ds9` regions, use the script `kcwi_masksky_ds9.py` contained in the `scripts` directory.

This script uses the `_intf.fits` files which are generated as part of the first execution of the pipeline. You should run the pipeline first and verify the quality of the sky subtraction. If you are not satisfied with the sky subtraction, use the procedure described here and run the pipeline again.

To start, display the `_intf.fits` frame on `ds9` and create regions around areas of the frame that you would like to exclude. Save the regions to a file and run the kcwi_mask_sky command indicating the file name (`_intf.fits`) and the region file. This will produce a mask file that can be added to the `kcwi.sky` file.

2.5.2 Re-run the reduction of this file

Rather than re-run the entire pipeline, it is possible to run it only on the file that needs a better sky subtraction.

The instructions are listed in *Running the pipeline*. In summary, you will need to specify just one file, the one for which you want to improve the sky subtraction and make sure that `clobber` is set to True in the config file.

```
reduce_kcwi -f myfile.fits
```

2.6 Data Products

The following is an overview of the data products created by the pipeline.

Table 1: Data Products

File Extension	Units	Note
int	e-/px	reduced output intensity
intd	e-/px	dark current/scattered light removed
wavemap	A	wavelength value at pixel
slicemap	0-23	slice number at pixel
posmap	0-140	slice position (px) at pixel
intf	e-/px	instrumental variations removed
intk	e-/px	sky subtracted
obj	e-/px	un-sky subtracted image
sky	e-/px	sky model image
icube	e-/px	geometry corrected data cube
icubed	e-/px	DAR corrected data cube
icubew	e-/px	Wavelength corrected data cube
icubes	flux/px	Std star calibrated data cube

More detail can be found at this file, which describes the products from the old IDL pipeline.

2.7 Support

If you find a problem with the pipeline, or need support with installing or running it, please check the *Known Issues* page. If you cannot find a solution to your problem, submit an issue on GitHub. This not only ensures that we see your request, but also allows our users to review previous solutions to issues they might be having. Wherever possible, please make sure to include:

- Logs and system messages
- Information on the files being processed (date of observation, filenames, frame types, etc)
- Any steps needed to replicate the issue

We discourage direct email interaction with Staff Astronmers or with the KCWI team.

2.7.1 What can you expect?

For issues submitted during business hours, we will try to acknowledge your request within 24 hours. Your request will be evaluated and redirected to the relevant staff. Our resources might not always be sufficient to provide immediate resolution, but we will work with you to help as much as possible.

2.8 Previous versions

2.8.1 Version 0.1 (2019)

- First end-to-end version including all the reduction step of the IDL pipeline
- Three execution modes

  - Reduce all files in a directory in the order in which they appear
  - Reduce all files after grouping them by file type and in the correct order
  - Monitor a directory for new files and reduce them as they appear
- Multi-threading for CPU intensive tasks such as wavelength calibration
- Multi-processing for large datasets

CHAPTER 3

More information

3.1 The KCWI DRP: basic concepts

While most of the basic algorithms are inherited from the IDL KCWI pipeline (KDERP), the architecture of this pipeline is completely different.

Most of the underlying architecture is a consequence of using the Keck DRP Framework. This event-based framework implements a set of rules that connect events to processing code.

To illustrate this concept, we can use a simple event such as a new file on disk. The KCWI_DRP associates this event to a Python class called `ingest_file`. This means that the namespace contains a class or a function with this name. The processing code contained in the class or the function will then be applied to the file.

In more detail, if the code is a class, the `_perform` method of the class will be executed.

In our case, the `ingest_file` class looks at the header of the file and ingests it in its entirety. It then trigger further processing steps based on the image type.

3.1.1 Recipes and primitives

The original concept upon which this DRP is based is the separation of code in primitives and recipes. Primitives are simple pieces of code usually dedicated to a single operation. An example of a primitive would be `subtract_bias`, which takes a science frame and a master bias and performs the subtraction. Recipes are sequences of primitives, and they are usually associated with a specific image type. An example of a recipe is `process_science` which would contain calls to primitives such as `subtract_overscan`, `apply_flat_field` or `apply_wavelength_calibration` and so on.

The Keck DRP Framework doesn't natively implement the concept of recipes, but it allows to specify a *next* event: the *next* event is an event that should automatically be triggered when the current event is finished. Using this, and linking individual events via the *next* event, we can easily build sequences of events which, for all intent and purposes, are the same as recipes.

These recipes are specified in a pipeline definition file, which can be found in `kcwidrp/pipelines/ kcwi_pipeline.py". This file contains an ``event_table` which provides the link between events and processing code. For each entry, the third field is the *next* event.

3.2 Primitives/API

class `kcwidrp.primitives.ApplyFlat.`**`ApplyFlat`**(*action*, *context*)
Generic routine to apply flat fielding. Not yet implemented.

class `kcwidrp.primitives.CalcPrelimDisp.`**`CalcPrelimDisp`**(*action*, *context*)
Calculate dispersion based on configuration parameters.

The parameters of the grating equation are calculates as:

alpha = grating_angle - 13 - adjustment_ange (180 for BH, RH and 0 for all other gratings)

beta = camera_angle - alpha

dispersion = cos(beta)/rho/focal_length x (pixel_scale x binning) * 1.e4

class `kcwidrp.primitives.CorrectDar.`**`CorrectDar`**(*action*, *context*)
Correct for Differential Atmospheric Refraction

`kcwidrp.primitives.CorrectDar.`**`atm_disper`**(*w0*, *w1*, *airmass*, *temperature=10.0*, *pressure_pa=61100.0*, *humidity=50.0*, *co2=400.0*)
Calculate atmospheric dispersion at w1 relative to w0

Parameters

- **w0** (*float*) – reference wavelength (Angstroms)
- **w1** (*float*) – offset wavelength (Angstroms)
- **airmass** (*float*) – unitless airmass
- **temperature** (*float*) – atmospheric temperature (C)
- **pressure_pa** (*float*) – atmospheric pressure (Pa)
- **humidity** (*float*) – relative humidity (%)
- **co2** (*float*) – Carbon-Dioxide (mu-mole/mole)

class `kcwidrp.primitives.CorrectDefects.`**`CorrectDefects`**(*action*, *context*)
Remove known bad columns

class `kcwidrp.primitives.CorrectGain.`**`CorrectGain`**(*action*, *context*)
Convert raw data numbers to electrons

class `kcwidrp.primitives.CorrectIllumination.`**`CorrectIllumination`**(*action*, *context*)
Subtract master bias frame

class `kcwidrp.primitives.CreateUncertaintyImage.`**`CreateUncertaintyImage`**(*action*, *context*)
Generate a variance image based on Poisson noise plus readnoise

class `kcwidrp.primitives.CubeImage.`**`CubeImage`**(*action*, *context*)
Transform 2D images to 3D data cubes

class `kcwidrp.primitives.FluxCalibrate.`**`FluxCalibrate`**(*action*, *context*)

class `kcwidrp.primitives.GenerateMaps.`**`GenerateMaps`**(*action*, *context*)
Generate map images

class kcwidrp.primitives.MakeMasterDark.**MakeMasterDark**(*action*, *context*)
Stack dark frames into master dark

class kcwidrp.primitives.NandshuffSubtractSky.**NandshuffSubtractSky**(*action*, *context*)

class kcwidrp.primitives.ProcessArc.**ProcessArc**(*action*, *context*)

class kcwidrp.primitives.ProcessBias.**ProcessBias**(*action*, *context*)

class kcwidrp.primitives.ProcessContbars.**ProcessContbars**(*action*, *context*)

class kcwidrp.primitives.ProcessDark.**ProcessDark**(*action*, *context*)

class kcwidrp.primitives.ProcessFlat.**ProcessFlat**(*action*, *context*)

class kcwidrp.primitives.ProcessObject.**ProcessObject**(*action*, *context*)

class kcwidrp.primitives.RectifyImage.**RectifyImage**(*action*, *context*)
Ensure output image has a consistent orientation

class kcwidrp.primitives.SolveGeom.**SolveGeom**(*action*, *context*)
Solve the overall geometry of the IFU

class kcwidrp.primitives.StackFlats.**StackFlats**(*action*, *context*)
Stack flat images

class kcwidrp.primitives.SubtractBias.**SubtractBias**(*action*, *context*)
Subtract master bias frame

class kcwidrp.primitives.SubtractDark.**SubtractDark**(*action*, *context*)
Subtract master dark frame

class kcwidrp.primitives.SubtractSky.**SubtractSky**(*action*, *context*)

class kcwidrp.primitives.TrimOverscan.**TrimOverscan**(*action*, *context*)
Trim off overscan region

class kcwidrp.primitives.kcwi_file_primitives.**ingest_file**(*action*, *context*)
Constructor

delta_wave_out()
Return output delta lambda in Angstroms for the given grating

map_ccd()
Return CCD section variables useful for processing

Uses FITS keyword NVIDINP to determine how many amplifiers were used to read out the CCD. Then reads the corresponding BSECn, and DSECn keywords, where n is the amplifier number. The indices are converted to Python (0-biased, y axis first) indices and an array is constructed for each of the two useful sections of the CCD as follows:

Bsec[0][0] - First amp, y lower limit Bsec[0][1] - First amp, y upper limit Bsec[0][2] - First amp, x lower limit Bsec[0][3] - First amp, x upper limit Bsec[1][0] - Second amp, y lower limit etc.

Bsec is the full overscan region for the given amplifier and is used to calculate and perform the overscan subtraction.

Dsec is the full CCD region for the given amplifier and is used to trim the image after overscan subtraction has been performed.

Tsec accounts for trimming the image according to Dsec.

Amps are assumed to be organized as follows:

(0,ny) ——— (nx,ny)

3 | 4 |

1 | 2 |

(0,0) ——— (nx, 0)

self: instance of CcdPrimitive class (automatic)

list: (int) y0, y1, x0, x1 for bias section list: (int) y0, y1, x0, x1 for data section list: (int) y0, y1, x0, x1 for trimmed section list: (bool) y-direction, x-direction, True if forward, else False

resolution()

Return FWHM resolution in Angstroms for the given grating

kcwidrp.primitives.kcwi_file_primitives.**kcwi_fits_reader**(*file*)

A reader for KeckData objects. Currently this is a separate function, but should probably be registered as a reader similar to fits_ccddata_reader. Arguments: file – The filename (or pathlib.Path) of the FITS file to open.

CHAPTER 4

Developers

The DRP was developed in collaboration between:

- Don Neill, Matt Matuszewki, Chris Martin and the KCWI Team at Caltech
- Max Brodheim and Luca Rizzi at W. M. Keck Observatory

Maintenance and support of the DRP are provided as part of the Data Services Initiative (PI: John O'Meara), in collaboration with the National Aeronautics and Space Administration.

4.1 Updating documentation

The documenation is generated automatically by ReadTheDocs, and triggerd by pushing commits to the `documentation` branch.

Note that the entire documentation infrastructure only exists in the `documentation` branch: the `docs` directory is not and should not be checked into any branch other than the documentaiton branch.

Make sure you are in the `documentation` branch at all times when updating documentation. If you accidentally make changes to the code or to other parts of the repository while in the `documenation` branch, do not push your changes.

If you don't have the `documentation` branch in your repository, use:

```
git checkout --track origin/documentation
```

4.1.1 Updating text

The documentation is contained in `.rst` files in the `documentation` branch. The `docs` subdirectory is only present in this branch. New documents can be added to the source directory, and linked to the correct section (usually in `index.rst`).

Once the new document (or a modified document) is ready, follow this procedure:

```
git checkout documentation
git pull
git add new_document.rst
git commit -m "I have added a new document"
git push
git checkout master (or develop or any other working branch)
```

4.1.2 Keeping the documentation branch up to date

Because the documentation uses autodoc methods (such as `automodule`) to extract doc strings from the functions and classes, it is important to keep the `documenation` branch in sync with the master branch (or with any other active branch). To to this, follow this procedure:

```
git checkout documentation
git pull
git rebase master (or develop or other active branches)
git push --force
git checkout master (or develop or other working branches)
```

4.2 Development Procedure

The following document describes the procedure to follow for making changes to KCWI_DRP.

4.2.1 Managing Git

There are three persistent branches in the KCWI_DRP repo:

- `master` (default): This branch contains the current release version of the code, and provides the source used to build the package for `conda` and `pip`.
- `develop`: This branch contains new features and non-essential fixes. It should be stable at all times. A new release is triggered by this branch being merged into `master`. Any new code should be added to this branch.
- `documentation`: This branch contains the documentation. Any new features or changes should be recorded here. For instructions on updating the docs, see the Updating Documentation page.

When adding new code, please do your development in a branch off of `develop`:

```
git checkout develop
git pull
git checkout -b new_feature
```

While developing, it is important to stay up-to-date with any changes in the `develop` branch:

```
git checkout develop
git pull
git checkout new_feature
git merge --no-ff develop
```

Alternatively, you can do your own development in your own fork of the repo.

Pull Requests

Once your new feature is ready for merging into the develop branch, you need to issue a pull request. The pull request must contain the following:

1. A short but thorough explanation of the feature and its purpose
2. A pass from the automatic TravisCI tests
3. Evidence of successfully processing a night of KCWI data

Additionally, the code must meet the following:

- All code must be documented. This includes updated docstrings for altered code and new ones for new functions or classes
- Comments explaining any unusually complicated or unintuitive code blocks, in addition to the documentation above
- Remove all print statements. If you need to output information to terminal, use the logging interface
- At least two developers must approve of the request for it to be accepted

If your pull request meets all of the above, it will be merged into `develop` during the next developer meeting.

Merging `develop` into `master`

At the developer's discretion, `develop` will be merged into `master`, which will signify a new sub-release. The version number should be incremented in `develop`, and a summary of changes should be added to CHANGES.rst. Then, issue and accept a pull request from `develop` into `master`.

4.2.2 Building for Release

These instructions summarize the steps required to build a new release of kcwidrp for conda or pip. These steps should be followed every time `master` is updated.

Pip

In order to upload to pip, you will need access to a PyPI account with owndership status for the kcwidrp project. For access to the KeckDRPs account, ask Max or Luca.

After your pull request is merged into master, download the changes:

```
git checkout master
git pull
```

Then, after ensuring the `dist` directory is empty (ensure no previous versions exist; these will conflict with the upload to PyPI):

```
# Make sure you have the most recent version of twine installed
pip install twine --upgrade

# Construct the pip distribution
python setup.py sdist bdist_wheel

# Test the upload by uploading to TestPyPI
twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

(continues on next page)

(continued from previous page)

```
#If all went well with the test, upload to the permanent PyPI
twine upload dist/*
```

Conda

Eventually, these steps will be rendered obsolete by the use of conda-forge. In the meantime, the following instructions will build a conda package from the pip package. This should be run whenever a new pip version is created.

```
conda update conda
conda install conda-build anaconda-client

conda-build conda_build_files
conda build conda_build_files --output

anaconda login
anaconda upload PATH-FROM-OUTPUT
```

4.3 Known Issues

This page contains a list of known issues related to the pipeline that are not currently being worked on. Only issues which cannot be fixed with a new release will be added to this list. Wherever possible, a link to relevent GitHub issues will be provided.

4.3.1 Firefox/geckodriver cannot be found

Error message:

```
RuntimeError: Neither firefox and geckodriver nor a variant of chromium browser and
↪chromedriver are available on system PATH. You can install the former with 'conda
↪install -c conda-forge firefox geckodriver'.
```

This issue has been submitted by a small number of users. This error is thrown by our plotting library, bokeh, when it can't automatically find the path to a driver needed to generate plots. The following describes a one-off workaround.

These instructions assume you are using `conda` to manage your environment.

1. If you installed the pipeline with `pip`, uninstall with `pip uninstall kcwidrp`
2. Install the pipeline following the Installing for Development instructions on the *Installing KCWI_DRP* page.
3. Open `KCWI_DRP/kcwidrp/core/kcwi_plotting.py` in a text editor
4. Add the following import at the top of the file:

   ```
   from selenium import webdriver
   ```

5. Find your firefox installation by executing `which firefox` from the command line. Make note of the output. It should look something like `/PATH/TO/CONDA/envs/kcwidrp/bin/firefox`
6. Replace the function `save_plot` with the following:

```
def save_plot(fig, filename=None):
    if filename is None:
        fnam = os.path.join('plots', 'kcwi_drp_plot.png')
    else:
        fnam = os.path.join('plots', filename)

    options = webdriver.FirefoxOptions()

    options.add_argument("--headless")
    options.add_argument("--hide-scrollbars")
    options.add_argument("--force-device-scale-factor=1")
    options.add_argument("--force-color-profile=srgb")
    driver = webdriver.Firefox(firefox_binary="[YOUR/PATH/HERE]",
                                firefox_options=options)
    export_png(fig, filename=fnam, webdriver=driver)
    driver.close()
    logger.info(">>> Saving to %s" % fnam)
```

where `[YOUR/PATH/HERE]` is replaced by the path found in the previous step

7. Navigate to the `KCWI_DRP` directory, and run:

```
python setup.py install
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index