odi-tools Documentation

Release 0.0.1

Owen Boberg, Bill Janesh

May 15, 2018

Contents

odi-tools is a suite of tools written in Pyraf, Astropy, Scipy, and Numpy to process individual QuickReduced images into single stacked images using a set of "best practices" for ODI data.

In this documentation we will provide descriptions of how to obtain and run the software, as well as a brief tutorial on using QuickReduced images.

CHAPTER 1

Getting started with odi-tools

1.1 Installation

To use this code simply fork this repository and clone it onto your local machine:

```
$ git clone https://github.com/bjanesh/odi-tools.git
$ cd odi-tools
```

Optionally add this folder to your `$PATH` so the odi-scripts maybe used in any current working directory.

To run the scripts you will need to install a number of dependencies:

```
$ pip install numpy scipy astropy photutils pyraf tqdm matplotlib pandas
```

It is possible to install these packages without root access by using the `--user` option:

```
$ pip install --user package-name
```

As noted on the astropy website, it might also be beneficial to use the `--no-deps` option when installing astropy to stop `pip` from automatically upgrading any of your previously installed packages, such as `numpy`:

```
$ pip install --no-deps astropy
```

1.2 QuickReduce, ODI-PPA, and odi-tools

QuickReduce is a set of pure python tools to reduce data from ODI. QuickReduce was created by Ralf Kotulla (UW Milwaukee, UW Madison) for the WIYN Observatory. The source code for QuickReduce is now on github. Documenation for the QuickReduce pipeline is available at this link.

The ODI-PPA is the online portal used to access, sort, and run QuickReduce on your ODI data. Information for gaining access and using the portal can be found at these help pages.

odi-tools is designed to work on images that have been processed using QuickReduce and downloaded from the ODI-PPA.

1.2.1 Running QuickReduce

After you become familiar with the layout and operation of the ODI-PPA, you can use these following steps to process your data. The options we list here are what we have determined to be the best practices for running QuickReduce. It is important to remember that these options can be data dependent. We will update these options should the change.

- Add the images you wish to reduce to a collection in the ODI-PPA and from the collection action menu, choose QuickReduce. See the PPA help pages for information about creating collections.
- Run QuickReduce with the following options. The `[X]` denotes that the option is selected.
  - `[X]` WCS
  - `[X]` Photometry
  - `[X]` Fringe (i- and z-band only)
  - `[X]` Persistency
  - `[X]` Nonlinearity
  - `[X]` Pupil Ghost (calibrations)
  - `[ ]` Pupil Ghost (science)
  - `[X]` Use Bad Pixel Masks
  - `[X]` Cosmic Ray Removal, 3 iteration(s)
- You should give this job a meaningful name and check the `Email me when when finished` box.
- When you receive the email letting you know your job is complete, download the QuickReduce data to your hard drive using the `Download Results` button on the QuickReduce job page. You don't need to select any of the optional boxes, just name the job and click submit. Eventually a `wget` command will pop up. Copy it to your clipboard, navigate to the folder you want the data to go into, then paste the wget command in your command line.
- At this point you are ready to start running `odi-tools`. See the *Basic usage* documentation for information on starting this process. See *An Introduction to working with Quick Reduced Images* for a brief tutorial on getting to know a QuickReduced image.

1.3 An Introduction to working with Quick Reduced Images

1.3.1 Navigating a QR image

Opening the fits file

`odi-tools` utilizes the `fits.io` module in the `astropy` package to open the multi-extension QR fits files:

```
>>> from astropy.io import fits
>>> img = '20140406T214040.2_GCPair-F1_odi_g.6183.fits'
>>> hdulist = fits.open(img)
```

Accessing the extensions

hdulist is now an astropy HDUlist where each element cooresponds to an extension of the fits file.

```
>>> print hdulist.info()
Filename: 20140406T214040.2_GCPair-F1_odi_g.6183.fits
No.    Name         Type      Cards   Dimensions     Format
0    PRIMARY     PrimaryHDU     294   ()
1    OTA33.SCI   ImageHDU       285   (4096, 4096)   float32
2    OTA34.SCI   ImageHDU       285   (4096, 4096)   float32
3    OTA44.SCI   ImageHDU       285   (4096, 4096)   float32
4    OTA43.SCI   ImageHDU       285   (4096, 4096)   float32
5    OTA42.SCI   ImageHDU       285   (4096, 4096)   float32
6    OTA32.SCI   ImageHDU       285   (4096, 4096)   float32
7    OTA22.SCI   ImageHDU       285   (4096, 4096)   float32
8    OTA23.SCI   ImageHDU       285   (4096, 4096)   float32
9    OTA24.SCI   ImageHDU       321   (4096, 4096)   float32
10   CAT.2MASS   BinTableHDU     15   680R x 2C    [D, D]
11   CAT.ODI     BinTableHDU    138   562R x 33C   [D, D, E, E, E, E, E, I, I, E, E,␣
↪E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E]
12   CAT.ODI+2MASS  BinTableHDU  31   165R x 5C    [D, D, D, D, E]
13   CAT.PHOTCALIB  BinTableHDU 172   251R x 36C   [D, D, D, D, E, E, E, E, E, E, E,␣
↪E, E, E, D, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E]
14   SKYLEVEL    BinTableHDU     50   1663R x 7C   [D, D, D, D, D, D, I]
15   ASSOCIATIONS  BinTableHDU   17   16R x 3C     [25A, 375A, 100A]
```

This particular image is from podi meaning there are a total of 9 OTAs in the focal plane. In the hdulist the extensions for each of the OTAs are given in rows 1-9 and have names with the following convention OTAxy.SCI. For a 5x6 ODI image, there would be 30 of these extensions.

The hdulist allows us to access each of the extensions by the its name give in the Name column, or by its number. If, for example, we wanted to only look at OTA33 we can do the following:

```
>>> hdu_ota = hdulist['OTA33.SCI']
# or using the number
>>> hdu_ota = hdulist[1]
```

Now that we have isolated this single OTA we can pick out its individual header and data:

```
>>> ota_header = hdu_ota.header
>>> ota_data = hdu_ota.data
```

ota_header and ota_data are now easily accessible to use in other scripts and modules. It is also easy to access header keywords:

```
>>> print ota_header['CRVAL1']
198.750082573
>>> print ota_header['CRVAL2']
18.4113084341
```

It is also convenient to pass individual OTAs to pyraf tasks based on the name of the extension. If we wanted to run daofind on OTA33 we could do the following:

```
>>> from pyraf import iraf
>>> img = '20140406T214040.2_GCPair-F1_odi_g.6183.fits'
>>> iraf.unlearn(iraf.apphot.daofind)
>>> iraf.datapars.setParam('fwhmpsf',fwhm,check=1)
>>> iraf.datapars.setParam('datamin',-900,check=1)
```

(continues on next page)

(continued from previous page)

```
>>> iraf.datapars.setParam('datamax',60000,check=1)
>>> iraf.datapars.setParam('sigma',25.,check=1)
>>> iraf.findpars.setParam('threshold',2.5)
>>> iraf.apphot.daofind.setParam('output',output.txt)
>>> iraf.apphot.daofind(image=img+'['+'OTA33.SCI'+']', verbose="no", verify='no')
```

The odi-tools scheme

The modules in `odi-tools` are designed to work over lists of images while stepping through each of the OTA extensions in a given image. This will discussed in further detail in other parts of the documentation. An pseudo code of this scheme would be:

```
>>> imglist = ['img1.fits','img2.fits','img3.fits']
>>> ota_dictionary = {1:'OTA33.SCI',2: 'OTA34.SCI',3 :'OTA44.SCI',
...                   4:'OTA43.SCI',5:'OTA42.SCI', 6:'OTA32.SCI',
...                   7:'OTA22.SCI' ,8:'OTA23.SCI',9:'OTA24.SCI'}
>>> for img in imglist:
...     for key in ota_dictionary:
...         ota = ota_dictionary[key]
...         perform tasks on img[ota]
```

The other extensions

In addition to the extensions for each OTA, the `hdulist` also contains extensions linking to fits tables with useful information. They are `CAT.2MASS`, `CAT.ODI`, `CAT.ODI+2MASS`, `CAT.PHOTCATLIB`, `SKYLEVEL`, `ASSOCIATIONS`. The header and data in each of these tables are easily accessed.

```
>>> photcat_data = hdulist['CAT.PHOTCATLIB'].data
>>> photcat_header = hdulist['CAT.PHOTCATLIB'].header
```

Some of the information in these tables is used by `odi-tools` during the image processing.

1.4 Basic usage

All you need to do to get started is download your QR-ed data from the ODI-PPA using the wget download command, then follow these steps. An explination of running quick reduce from ODI-PPA will be given in other sections of the documentation:

1.4.1 Preparing your data

1. move all individual `.fz` files into the top level folder: `mv calibrated/**/*.fz .`
2. unpack the compressed fits files using funpack
3. you need to rename your files to match the appropriate dither pointing identification. for example, QR files are named by the pattern `OBSID_OBJECT_FILTER.JOBID.fits`. The final digit of the OBSID e.g. `20151008T195949.1` needs to match the number sequence of the dithers 1-9. Your data may not match this pattern due to restarted observations, multiple night observations, etc.

1.4.2 Running the code (a broad overview)

1. copy `example_config.yaml` to your data directory as `config.yaml` and edit the file to match your preferences/data. Make sure that the number for each image matches the correct number in the dither sequence!
2. run `odi_process.py` in the folder containing the unpacked/renamed fits images. This will (optionally) illumination correct the images, fix their WCS, reproject them to a common pixel scale, and perform background subtraction on them.
3. this will take a while, so make sure nothing bad happened
4. run `odi_scalestack_process.py` in the folder containing the unpacked/renamed fits images. This will detect bright stellar sources in the images and use them to calculate a scaling factor relative to the image in the sequence with the lowest airmass, then apply the scale, stack the images, then add in a common background value.
5. finished! check your images to make sure everything went okay.

1.4.3 Example configuration file

Here are the contents of `example_config.yaml` available on the odi-tools GitHub repo

```
# odi-tools configuration file
basic:
  object: M13                           # the name of your object
  filters: [odi_g, odi_r, odi_i]        # correct filter strings required
  instrument: 5odi                      # podi | 5odi | mosaic; script will
                                        # verify using image header info

processing:                             # optional steps performed in odi_process.py
  illumination_correction: yes          # if yes, set dark sky flat source below
  dark_sky_flat_source: object          # object | master
  wcs_correction: yes
  reproject: yes
  scale_images: yes
  stack_images: yes

# list the images you want to process
# be sure to associate the filename with the correct dither pointing!
# OBSID and image header are NOT always an accurate reflection of the absolute dither␣
↪position
# so you must use your notes / observing log to define them here
# sections must be named according to the filter names

odi_g:
  1: 20130510T002928.1_m13-9_odi_g.5869.fits
  2: 20130510T002928.2_m13-9_odi_g.5869.fits
  3: 20130510T002928.3_m13-9_odi_g.5869.fits
  4: 20130510T002928.4_m13-9_odi_g.5869.fits
  5: 20130510T002928.5_m13-9_odi_g.5869.fits
  6: 20130510T002928.6_m13-9_odi_g.5869.fits
  7: 20130510T002928.7_m13-9_odi_g.5869.fits
  8: 20130510T002928.8_m13-9_odi_g.5869.fits
  9: 20130510T002928.9_m13-9_odi_g.5869.fits

odi_r:
  1: 20130510T002928.1_m13-9_odi_r.5869.fits
```

(continues on next page)

(continued from previous page)

```
  2: 20130510T002928.2_m13-9_odi_r.5869.fits
  3: 20130510T002928.3_m13-9_odi_r.5869.fits
  4: 20130510T002928.4_m13-9_odi_r.5869.fits
  5: 20130510T002928.5_m13-9_odi_r.5869.fits
  6: 20130510T002928.6_m13-9_odi_r.5869.fits
  7: 20130510T002928.7_m13-9_odi_r.5869.fits
  8: 20130510T002928.8_m13-9_odi_r.5869.fits
  9: 20130510T002928.9_m13-9_odi_r.5869.fits

odi_i:
  1: 20130510T002928.1_m13-9_odi_i.5869.fits
  2: 20130510T002928.2_m13-9_odi_i.5869.fits
  3: 20130510T002928.3_m13-9_odi_i.5869.fits
  4: 20130510T002928.4_m13-9_odi_i.5869.fits
  5: 20130510T002928.5_m13-9_odi_i.5869.fits
  6: 20130510T002928.6_m13-9_odi_i.5869.fits
  7: 20130510T002928.7_m13-9_odi_i.5869.fits
  8: 20130510T002928.8_m13-9_odi_i.5869.fits
  9: 20130510T002928.9_m13-9_odi_i.5869.fits
```

CHAPTER 2

Individual modules and tasks

2.1 Modules

2.1.1 Python configuration to run odi-tools

The function `odi_config.py` imports all of the modules needed to run `odi-tools` as well as sets up the needed dictionaries and directories. It is imported in the following way in the main `odi-tool` scripts (`odi_process`, `odi_scalestack_process`, `odi_phot_process`).

```
>>> import odi_config as odi
```

Once it is imported, the dictionaries are directories created in `odi_config.py` can be referenced throughout the `odi-tools` pipeline in the following manner

```
>>> example_dict = odi.dictionary
>>> example_directory = odi.directory
```

Similarly, we can also reference all of the modules and functions imported in `odi_config.py`.

```
>>> gaps = odi.get_gaps(img, ota)
```

The OTA dictionaries

There are two dictionaries defined by this function that correspond to different versions of ODI.

1. `podi_dictionary`
2. `odi5_dictionary`

As an example, here are the contents of `podi_dictionary`:

```
podi_dictionary = {1: 'OTA33.SCI',
                   2: 'OTA34.SCI',
                   3: 'OTA44.SCI',
                   4: 'OTA43.SCI',
                   5: 'OTA42.SCI',
                   6: 'OTA32.SCI',
                   7: 'OTA22.SCI',
                   8: 'OTA23.SCI',
                   9: 'OTA24.SCI'
                   }
```

For those that are not familiar with Python, a dictionary is made up of pairs of `keys` and `values`. The `keys` are to the left of the colon, and the `values` are to the right. In our case, the `keys` are the numbers 1-9, and the values are the names of the different OTAs, (e.g. `OTA33.SCI`). The dictionaries provide a clean way to work through a multi-extension fits image like those produced by ODI. The simpled code example below provides an illustration of how this is done in `odi-tools`.

```
>>> images = ['img1.fits','img2.fits']
>>> for img in images:
>>>     for key in podi_dictionary:
>>>         print img, key
```

This would produce the following output:

```
'img1.fits' 'OTA33.SCI'
'img1.fits' 'OTA34.SCI'
'img1.fits' 'OTA44.SCI'
'img1.fits' 'OTA43.SCI'
'img1.fits' 'OTA42.SCI'
'img1.fits' 'OTA32.SCI'
'img1.fits' 'OTA22.SCI'
'img1.fits' 'OTA23.SCI'
'img1.fits' 'OTA24.SCI'
'img2.fits' 'OTA33.SCI'
'img2.fits' 'OTA34.SCI'
'img2.fits' 'OTA44.SCI'
'img2.fits' 'OTA43.SCI'
'img2.fits' 'OTA42.SCI'
'img2.fits' 'OTA32.SCI'
'img2.fits' 'OTA22.SCI'
'img2.fits' 'OTA23.SCI'
'img2.fits' 'OTA24.SCI'
```

Although this is a simple example it illustrates the overall workflow of `odi-tools`.

The `odi5_dictionary` works the same way, but simply has more OTAs. The correct dictionary is selected by `odi_helpers.instrument()`. If the instrument used was `5odi`, `odi5_dictionary` is used for `odi-tools`, if it was `podi`, `podi_dictionary` is used.

Processing directories

`odi_config.py` also sets up a number of directories to hold the intermediate data products during the data processing. Here is an example of how one of those directories is created

```
>>> bpmdirectory = 'bpmasks'
>>> if not os.path.exists(bpmdirectory):
```

(continues on next page)

(continued from previous page)

```
>>>     print 'Creating directory for bad pixel masks...'
>>>     os.makedirs(bpmdirectory)
```

```
>>> bppath = bpmdirectory+'/'
```

The directory in this case is given the name `bpmasks`. Then, we check if the directory already exists. If is does not, the directory is created. Once this directory is created it can be accessed by other `odi-tools` modules and scripts using the following

```
>>> odi.bppath
```

Here is a full list of the directories created

- `bpmasks` - directory for bad pixel masks
- `illcor` - directory for illumination corrected ota images
- `reproj` - directory for reprojected ota images
- `bgsub` - directory for background subtracted ota images
- `scaled` - directory for scaled ota images
- `otastack` - directory for stacked ota images
- `skyflat` - directory for sky flats
- `coords` - directory for coordinate files
- `match` - directory for match files
- `sdssoffline`- directory for sdss catalogs
- `twomassoffline` - directory for 2mass catalogs
- `gaiaoffline` - directory for gaia catalogs
- `sources` - directory for detected sources

2.1.2 Reading configuration files

2.1.3 ODI helper functions

These are simple functions used throughout the `odi-tools` pipeline.

2.1.4 Calculating background statistics

2.1.5 Source catalogs

These functions retrieve and parse the SDSS and Gaia DR1 catalogs to be used when fixing the WCS solutions of individual OTAs.

2.1.6 Measuring Stellar FWHM

There are a number of steps in `odi-tools` that require having a measurement of the stellar fwhm of sources on individual OTAs or on a fully stacked image. In order to get these measurements we use the `pyraf` task `rimexam` on a list of known x and y positions for SDSS sources on a given field. Here is how the parameters are set for `rimexam`:

```
iraf.tv.rimexam.setParam('radius',radius)
iraf.tv.rimexam.setParam('buffer',buff)
iraf.tv.rimexam.setParam('width',width)
iraf.tv.rimexam.setParam('rplot',20.)
iraf.tv.rimexam.setParam('center','yes')
iraf.tv.rimexam.setParam('fittype','gaussian')
iraf.tv.rimexam.setParam('iterati',1)
```

2.1.7 Creating an OTA bad pixel mask

2.1.8 Create OTA gaps bad pixel masks

2.1.9 Improving WCS solutions

These are the functions that improve the WCS solutions of otas based on source catalogs with known Ra and Dec positions.

2.1.10 Find sources for OTA scaling

These functions locate the bright sources on OTAs, runs phot on these sources, and calculate the scaling factor needed to be applied to each OTA based on a reference image.

2.1.11 Full Calibrate

2.1.12 Full Phot

CHAPTER 3

Indices and tables

- genindex
- modindex
- search