Greenhill Observatory Pipeline – UTGO_Pipeline

Astronomical image manipulation, reduction and photometry

Introduction:

In this document, I outline the code I have written to aid in image reduction and analysis for the Greenhill Observatory. For the moment, this is optimised for Harlingten 50cm data, but may be used on the 1.3m with some minor modification.

Installation:

This code is predominately written in Python, leaning very heavily on the modular pipeline building package 'Prose' (Garcia et al. 2021). It will also require the installation of SExtractor, SWarp and Astrometry.net, if full functionality is to be replicated on a private machine. To use the scripts, you will need to install the required dependencies. I recommend the user to utilise Anaconda or Miniconda in order to manage python environments. To start, create a new environment (let's call it 'red') by going to the terminal (or Anaconda Prompt on Windows) and typing:
conda create -n red python=3.9.16

Once activated, (red) should display on the far left of the terminal window. Download the project folder from Github (https://github.com/tjplunkett/UTGO_Pipeline) and then change into this directory, using: **cd [path2folder]**

Then, assuming PIP was installed successfully to this environment, type:
pip install -r requirements.txt

The dependencies should now be installed and you will be ready to use the scripts! If you do not want to install locally, a copy of this code and a pre-made environment will be set up on the Keck Machine.

Scripts

All scripts come with helper functions from the parser, so you can type:

python [scriptname] -h

This will give information on each scripts functionality and necessary parameters to pass to them.

Main Pipeline:

define_telescope.py:

This script needs to be run once before any other scripts. It defines Telescope objects for Prose to use, which contain information about the telescopes and fits header layouts. To run, type:

Usage: **python define_telescope.py**

This will save the Telescope object, so you don't have to run it ever again!

update_calibs.py

This script searches a night's folder for new calibration frames. If there are enough of each type (i.e => 15 for bias, >= 10 dark and >=5 for flats), they are copied to the 'Calibration' folder and are then used to produce master calibration frames using the 'make_master_calibs.py' script. Gets automatically called once daily by ReductionBot.py.

Usage: **python update_calibs.py [nightfolder] [verbose]**

autoReduce.py:

This script performs automatic reduction of a night's folder of images, using the closest prior calibration frames by date and scaling down the closest dark if none match the exact exposure time of the science image. It also creates nightly summaries for each target, with a .CSV file containing statistics on the observing session, a stacked image of the best 5 frames and calibrated catalogue from this stack (using SExtractor). Will be called once daily to reduce new data, but can also be placed in a crontab to periodically reduce data on the fly.

Usage: **python autoReduce.py [path2night] [on_the_fly] [verbose]**

[on_the_fly] – Do you want to run in 'on-the-fly' mode? (y/n)
[verbose] – Do you want to know what's happening? (y/n)

Note: The 'on-the-fly' mode is bare bones reduction, without building bad pixel maps, doing astrometry or creating the nightly summaries. This is useful for reduction whilst observing, as these processes tend to take a long time. If you want astrometry in this mode, then use the 'run_astrometry.py' script on the output science folder.

reduce_H50.py:

This script is used to perform a manual reduction of H50 data, given a folder containing science objects and calibration frames. The code does as follows:

- Finds all the frames (science and calibration) in specified directory and changes the 'TELESCOP' keyword value to 'Planewave 50cm' to be compatible with each other.

- It then iterates through all the science images in the folder, reading their size (i.e if they are subframes) and filters. For each unique subframe type, a new folder will be created labelled by the image dimensions. Then the calibration frames will be

trimmed (using functions defined in pytrimmer.py) to match the dimensions and saved to the new folders.

- Finally, for each subframe folder, the set of images with matching filters are found and reduced using the newly trimmed calibration frames (i.e for 1028x1028 images in r, the code uses all the darks & biases and only the r flats to reduce the images). The calibration includes exposure time-scaled bias, dark and flat corrections and cleaning of bad pixels (using a map created with the dark and flats). This will save the reduced images to subframe folders.

Usage: **python reduce_H50.py [path2folder] [depth]**

Note: Depth is the number of subfolders to look through (i.e 0 will just be in the specified folder). Therefore, if you have subfolders containing darks, flats and biases (i.e in a folder called 'CALIB' for instance), you could use depth=1.

Photometry:

pseudosex.py:

Performs source extraction on a set of images using either SExtractor or PhotUtils. These catalogues will then be calibrated using GAIA DR2 synthetic photometry, by finding the sigma clipped mean of the difference between the GAIA calibrated and H50 instrumental magnitudes.

Usage: **python pseudosex.py [path] [depth] [ap_size] [sex] [WCS_flag]**

[ap_size] – The radius of the aperture
[sex] – Do you want to use SExtractor or not? (y/n)
[WCS_flag] – Do the images have astrometry? (y/n)

calibrate_phot.py

Mainly defines the background methods to calibrate our photometry to GAIA DR2. However, this can be manually called on an individual catalogue if required (even non-H50 data, with minor modification).

Usage: **python calibrate_phot.py [path2cat] [filter] [sex]**

[filter] – The filter of observation to try search for corresponding calibrated magnitude in the GAIA DR2 catalogue.

[sex] – Is the catalogue a .cat file (from SExtractor)? (y/n)

aper_phot.py:

Perform aperture photometry on a set of images and (if desired) calibrate these measurements to GAIA DR2 photometry. This requires the target to be detectable on all frames for accurate centroiding, so may not be suitable for all transients.

Usage: **python aper_phot.py [path2images] [depth] [ID] [autoAperture] [verbose]**

[ID] – The target's ID/name

[autoAperture] – Do you want the code to find a good aperture size for you? (y/n)

[verbose] – Do you want know what's happening in the background?

diff_phot.py:

This script is used to perform differential photometry on a target star across multiple observations. It will output a light curve file in a CSV, along with images of the PSF model, the companion stars used and a summary of the photometry results in a folder called 'Phot_Outputs'. The script does as follows:

- Finds all the fits images in a specified folder (which should be calibrated, perhaps using reduce_H50.py).
- Performs source extraction on a reference image (just the first in the list found), giving the user a visual display to pick the correct star from its label.
- Registers all the images (i.e by finding the affine transform to account for any rotations/shifts).
- Then performs photometry on all the stars identified.

Then, the next step depends on how you want to choose companion stars. The built-in method from Prose is to use the Broeg et al. 2005 algorithm, which finds optimal stars by recursively building light curves for each star in the frame and evaluating which have the least variance across the observations. This does not consider differential refraction effects due to the star colours, so alternately you can query GAIA DR2 (use 'gaia_flag' = 'y') to find the 5 stars closest in colour and magnitude to the target. This requires internet (Broeg does not) and uses functions defined in the query_gaia.py.

Usage: **python diff_phot.py [path2calibratedfiles] [depth] [threshold]**

Note: The threshold flag dictates how many sources you will detect. Increase threshold, reduce number of objects detected.

forced_phot.py:

This script performs forced aperture photometry on a set of images, for the case when the target cannot be detected by Prose or Source Extractor (i.e a faint GRB perhaps!). The code does as follows:

- Finds all the science images in a folder.

- Allows the user to manually choose the desired target by either showing them a reference image (if 'vis' is 'y') and allowing them to zoom and double click on aperture centre, or by hand input of (x,y) coordinates.

- Iterates through each image in the list and registers it to the original reference (i.e finds the coordinate transformation).

- It applies this transformation to find the aperture centre on the target image (without performing any shifts on the actual data) and then uses 'PhotUtils' to perform simple aperture photometry with basic background subtraction.

- The counts are converted to electrons, then divided by exposure time and converted to instrumental magnitudes. The SNR is also estimated.

- Finally, the magnitudes are roughly calibrated using previously measured extinction coefficients and empirically measured zeropoints from pseudosex.py and calibrate_phot.py.

Usage: **python forced_phot.py [path2folder] [depth] [astr] [vis]**
[astr] – Has astrometry been done? (y/n)
[vis] – Do you want to see the reference image and click target? (y/n)

Utilities:

fix_header.py:

Bulk editing or creation of header keyword-value pairs ('cards'). Finds all the .fits file in a folder and sets the value to desired input in header.

Usage: **python fix_header.py [path2folder] [keyword] [value]**

pytrimmer.py:

This script can be used to trim individual fits images to specified dimensions (i.e a rectangle), retaining the associated WCS. It will then output this trimmed file to a new directory.

Usage: **python pytrimmer.py [path2fits] [x_min] [x_max] [y_min] [y_max] [path2output]**

nightly_summary.py:

This script defines the necessary functions to create nightly summaries, including measuring image statistics, stacking and source extraction/calibration. Currently not callable, but this may be changed in future.

make_master_calibs.py:

A script to create a master calibration image by combining multiple individual frames. The options are to use a sigma clipped average or a simple median in the combination process.

Warning: Whilst taking a sigma clipped average produces a higher SNR frame, it was found that twilight flats are not good for sigma clipping, as the thresholding changes so rapidly. This means stars can be missed in the rejection process and remain on the master frame. It is recommended to use the median instead for flats.

Usage: python make_master_calibs.py [path2calibs] [method]

stacker.py:

Stacks a set of images in a folder to increase the SNR of the observations. Can stack by the entire night (type ‘d’ or ‘D’) or hour per night (type ‘h’ or ‘H’), by a number of images (type an int) or just all images in the folder (type ‘A’ or ‘a’). You can specify the stacking procedure to use with the ‘method’ flag, either using SWarp (type ‘swarp’) or a default Prose method (median stack).

Usage: **python stacker.py [path2images] [depth] [method] [period]**

run_astrometry.py:

This script will perform astrometry on a directory of science images using a locally installed copy of Astrometry.net. It is designed to run on the Keck Machine, but you may be able to alter it to run on your local machine without too much effort (assuming you have the software installed).

Usage: **python run_astrometry.py [object_folder] [telescope]**

flipper.py:

This script will analyse and then flip images within a folder to match orientation. This is useful for when meridian flips need to be corrected, like with PySIS. Note, this is purely a transposition of the image axes, so does not interpolate pixels (which could lead to errors in flux measured).

Usage: **python flipper.py [path2fits] [verbose]**

prep4pysis.py:

This script is run on reduced science images to prepare them for input into the PySIS difference imaging pipeline. It does as follows:

- Adds the MJD time of the observation to the fits headers by reading 'OBS-DATE' keyword. Also, does some copying of keywords to new cards to be compliant with PySIS.

- Does a batch rename of the files to follow PySIS conventions (i.e of the form [Survey][Location][yr][ID][filter][ImNum], which for microlensing might be OB231060r0001). This will create a 13-character filename for your images.

- Call functions defined in flipper.py on these images, to match the orientation of the images around any meridian flips.

- Outputs these new files into a folder called 'All' which can then be used with PySIS.

Note: recommended to copy your reduced files from their original reduced folder, into the PySIS folder (in a sub-folder) and then run this code on that folder.

Usage: **python prep4pysis.py [path2images] [prefix] [verbose]**
[prefix] – The first 9 characters of the desired filenames. For instance, for the microlensing target OGLE-2023-BLG-1060, we would perhaps choose 'OB231060r' to indicate the target and filter used.

defringer.py:

This script defringes a set of images using a pre-made fringe map. This is only necessary for images taken with long exposure times in the SDSS i' filter on the H50. It is not optimised, and the fringe map could be improved in future.

Usage: **python defringer.py [path2fitsfolder] [fullpath2map]**