Colibri Diagnostic Scripts

Rachel A. Brown

May 2022

This document describes the diagnostic scripts used to examine Colibri data. The following scripts can all be found in RachelPipeline on the projects GitHub, or in the rbrown/Documents/Colibri/Scripts folder on Pomegranate. ***include example plots, table outputs?***

Contents

1 astrometry.py

This is used to analyze plate solutions from astrometry.net. This script does not query astrometry.net directly like **astrometry_funcs.py** (Section 2). You will need to upload the image to the astrometry.net website and download the output files manually from there.

1.1 Need to Run

•

2 astrometry_funcs.py

3 bias_subtract.py

4 getRADec.py

5 lightcurve_looker.py

6 lightcurve_maker.py

7 lightcurve_maker_rms.py

8 read_npy.py

9 sensitivity.py

This script is used to make plots of Gaia G-band magnitude vs Colibri SNR for detected objects.

9.1 Need to Run

- Directory containing .rcd or .fits images ordered by time and labelled sequentially. Expected file structure is '***base_path**/ColibriData/ **Telescope**Data/**yyyymmdd**/**yyyymmdd_hh.mm.ss.mmm**/*', although this can be adjusted by changing the *base_path* and *data_path* variables. Output files from the script are saved in a directory that the script creates called '***base_path**/ColibriArchive/Sensitivity/**Telescope**/**yyyy-mm-dd**/*'.

- Bias images in .fits or .rcd format (preferably multiple because *'lightcurve_maker.py'* takes a median of 10. These are expected to be in '***base_path**/ColibriData/ **Telescope**Data/**yyyymmdd**/Bias/*'

- *as of Feb 7 2022 you also need an output file from astrometry.net. It is the output file labelled as emph'new-image' when you use the web version of astrometry.net. The filename for this should contain the original image index number that astrometry was performed on, *'new-image'*, and the order of the solution (ex *'3rd'*, *'2nd'*)

- The other scripts listed in Section 12.4 in the same folder as sensitivity.py.

9.2 Algorithm

1. The first part of the script (marked *'Set up'*) sets up the variables and filepaths that will be used later on. Colibri data is generally first organized by telescope, then by observation night, then by observation minute.

    The first block of code the user inputs the observation date, observation minute, index of the image used in astrometry.net, polynomial order of the astrometry.net solution, radius of photometry aperture [px], gain level, telescope name, field name, and sep.extract detection level. These options are all available so different combinations can be easily changed tested without having to go into the many different functions and change them manually.

    The next few blocks of code set up the directory structure that the script will use to find data files and save output files. It will also make output directories if needed.

2. Next object detection and photometry is done on the data using functions in *lightcurve_maker.py* (Section 6). This will also save a median combined master bias image (in a folder labelled '***Gain**_masterBiases/*). A median-combined, bias-subtracted image used for object detection is also saved, labelled ***Gain**_medstacked.fits*. A list of detected object coordinates from sep.extract is saved in a numpy array file labelled ***fieldname_date_time_gain_threshold**sig* For each detected object a .txt file containing a table of fluxes from each image is saved in a folder labelled ***Gain_threshold**sig_lightcurves*. Plots of these lightcurves are made using *lightcurve_looker.py* (Section 5) and saved in the same folder as the lightcurve.txt files.

   This step takes the longest (can be several minutes depending on the number of images and detections). If I've already performed photometry with the desired parameters I'll comment the above two calls before running the script to save time.

3. The image is uploaded to astrometry.net to obtain a WCS solution. Previously this was done manually by uploading a subtracted image to the astrometry.net web service, then downloading the new image file. This can now be done automatically using a function in **astrometry_funcs.py**.

   The filepath to the median combined image saved by the previous step is obtained. The path to the file which will contain the WCS solution is also defined.

   A check is performed to see if this file already exists (it can take a long time to get a new solution every time, so if one has already been calculated for this minute it's faster to use that one). If the file already exists, it is opened and the transformation is taken from the header.

   If the file doesn't already exist, the function **getSolution** from **astrometry_funcs.py** is called. This calculates the new transformation, saves it in a file labelled ***date_time**_wcs.fits*, and returns the solution to the main script.

4. A transformation of object coordinates from (X, Y) [px] to (RA, Dec) [deg] is performed. This is done using a plate solution from astrometry.net and a list of star coordinates from the above .npy file.

   The first block of code sets up the filenames for this step.

   The next block performs the coordinate transformation on the object coordinates using functions in *getRADec.py* (Section 4), and makes the output into a pandas dataframe for ease of use (*coords_df*). A table listing object coordinates in (X, Y) and (RA, dec) is saved to a .txt file labelled *XY_RD**_gain_threshold**sig_date_time.txt*. In this step, a file is saved with star coordinates in a format that can be read by DS9 using a function in *read_npy.py* (Section 8). This file is labelled ***fieldname_date_time_gain_threshold**sig_pos_ds9.npy*. To use this file, open a .fits image in DS9, go to *Regions*, *Open Regions* and select the file. Objects are marked using circles over the image.

5. A query is made to the Gaia catalog to get all star data (coordinates and magntiudes) for stars within a search radius from the field centre that are also within a magnitude limit. This is done using a function in *VizieRQuery.py* (Section 13). Stars are returned in a dataframe called *gaia* that has RA, dec, G-band magnitude, and (B-R) colour.

6. The airmass, altitude, and azimth of the observed field is calculated. The first star detection .txt file is read in to get the header time of that event, which will be used to determine the telescope's pointing. This header time is passed along with the field centre coordinates to the function **getAirmass** (Section 9.3.1) to get the airmass, altitude, and azimuth of the observed field.

7. For each detected object the median flux, flux standard deviation, and SNR (median/stddev) is calculated and returned in a dataframe containing the object coordinates and these values. This is done using a function in *snplots.py* (Section 11). A table containing this data is also saved in a file labelled *stars_snr.txt* found in the lightcurve folder.

8. There are now three tables (pandas dataframes) that need to be matched: 1. object coordinates, 2. Gaia star data, 3. object flux data. This is done in two steps. First, a dataframe containing object (X, Y, RA, dec, Gmag) called *rd_mag* is created using *match_RADec* (Section 9.3.2). This is then matched with object flux data using *match_XY* (Section 9.3.3) to create a called *final* with (X, Y, RA, dec, GMag, median flux, flux standard deviation, SNR). *note that this whole algorithm could be simplified by one step by getting the RA dec transformation directly from the object data table. This requires some changes to the structure of the code.

   If the image has any region with artefacts (for example the stripes on the edge of the image) detections in these regions can be filtered out of the final dataframe.

   This final dataframe is saved to a .txt file labelled *starTable**_minuteDirectory_gain_solutionOrder_Telescope_detection***

9. A plot showing instrumental magnitude (-2.5(log(median))) vs Gaia magnitude is displayed and saved to *magvmed_**gain_minuteDirectory_detectionThreshold**.png*.

   A plot showing Signal/Noise (median/stddev) vs Gaia magnitude is displayed and saved to *magvSNR_**gain_minuteDirectory***

10. Prints out how many objects were detected, matched with Gaia stars, and have a SNR greater than or equal to 10.

11. Makes a scatter plot of right ascension and declination differences between Colibri data and Gaia data (Section 9.3.4).

9.3 Functions

9.3.1 getAirmass

- calculate airmass of the given field at the given time.

input:

1. header time in *isot* format [string]

2. RA of field [degrees]

3. dec of field [degrees]

returns:

1. airmass

2. altitude [degrees]

3. azimuth [degrees]

Algorithm:

1. The latitude and longitude of the site are defined.

2. The Local Sideral Time (LST) of the observations are calculated using the method **sidereal_time** from the *Time* package.

3. The Hour Angle of observations is calculated by subtracting the field RA from the LST.

4. The altitude of the field is calculated using the following equation:

$$alt = arcsin(sin(\delta) \times sin(lat) + cos(\delta) \times cos(lat) \times cos(HA)) \tag{1}$$

   Note that the **numpy** trig functions take angles in radians, not degrees so conversions are necessary.

5. The azimuth of the field is calculated. First the quantity A is calculated with the following formula,

$$A = acos\left(\frac{sin(\delta) - sin(alt)sin(lat)}{cos(alt)cos(lat)}\right) \tag{2}$$

   If $sin(HA)$ is negative, then the azimuth is equal to A. Otherwise, the azimuth is $360.0° - A$.

6. The Zenith Angle is calculated by subtracting the altitude from $90°$.

7. The airmass is calculated using the following formula:

$$airmass = \frac{1}{cos(ZA)} \tag{3}$$

9.3.2 match_RADec

- matches list of Colibri detections with Gaia catalog by RA/dec to get magnitudes

- Adapted from Mike Mazur's matching script

- If multiple matches are found within the search box, the match with the brightest magnitude is selected. There are two lines that can be uncommented to select based on closest distance instead.

input:

1. pandas Dataframe of detected star data x, y, RA, dec

2. dataframe of Gaia data ra [deg], dec [deg], magnitude, colour

3. search box half side length [deg] (ex. SR = 4, search rectangle is 8 x 8)

returns:

1. original star data frame with gaia magnitude columns added where a match was found X, Y, ra, dec, Gmag

9.3.3 match_XY

- matches list of Colibri detections with Gaia magnitudes to list of Colibri detections with flux info based on X, Y coords

- Adapted from Mike Mazur's matching script

- If multiple matches are found within the search box, the match with the highest SNR is selected (in theory this shouldn't be needed, because the X and Y coordinates from the two dataframes should match exactly as they come from the same source).

input:

1. pandas dataframe of detection data with Gaia magnitudes x, y, RA, dec, G magnitude

2. dataframe of star data with SNR x, y, med, std, snr

3. search box half side length [px] (ex. SR = 4, search rectangle is 8 x 8)

returns:

1. original detection data frame with flux data columns added where a match was found X, Y, ra, dec, GMAG, med, std, SNR

9.3.4 RAdec_diffPlot

- makes scatter plot of difference between Colibri star coordinates and Gaia star coordinates

- Originally from astrometry.py (Section 1)

input:

1. pandas dataframe of star data including an RA difference column ('*ra_diff*') and dec difference column ('*dec_diff*'). Columns should be Colibri - Gaia to match the plot labels

returns:

1. Saves scatter plot of dec difference vs ra difference, with the 2 sigma clipped mean and standard deviation for both.

9.4   Required Modules

- numpy

- matplotlib.pyplot

- pandas

- datetime

- astropy, astropy.stats

- pathlib

- getRAdec.py (Section 4)

- snplots.py (Section 11)

- lightcurve_maker.py (Section 6)

- lightcurve_looker.py (Section 5)

- read_npy.py (Section 8)

- VizieR_query.py (Section 13)

10   sensitivity_rms.py

11   snplots.py

12   timingplots.py

12.1   Need to Run

- Directory of observations for a single night (*night directory*). This should contain multiple subdirectories (*minute directories*), which in turn contain images for that minute. At 40 fps, should be around 2400 images in the directory. Images should be in .rcd format.

12.2   Algorithm

1. Set up input and output directories. The input directory (*night directory*) is given as an argument in the command line when running the script. Note that the output directory (where script results will be saved) is created relative to this. Printouts from the script will be saved in a .txt file in the output directory, along with any plots produced.

2. Each *minute directory* is looped through, checked for anomalies, and a plot of timestamps is produced.

   (a) A list of all images in the *minute directory* is created.

   (b) If this list is empty, a line is written to the output file saying that the directory is empty. It moves on to the next *minute directory*.

   (c) Each image is looped through.

      i. The image data is read using **readxbytes** (Section 12.3.1). The timestamp of the image is taken from its header. Note that the starting byte may need to be changed in future if the format of the image header is changed by FLI.

      ii. The hour and minute of observation are taken from the timestamp. This is done using a **try** and **except**, because if the file is corrupted in some way it will cause an error. In this case, we'd like to note the corrupted file and carry on with the next image.

iii. A check is performed to see if the timestamp hour is greater than 23:59. This was happening sometimes at one point in testing (hour was showing as '29:00' which of course is nonsense). In this case, the correct hour is taken from the *minute directory* name. If the hour has rolled over during the minute, 1 will need to be added to the hour. Also, if the observing computer is in local time, then the conversion from UTC (the cameras are all in UTC) to local time needs to be made.
The bad hour is replaced by the corrected hour once this has been done.

iv. The frame number and timestamp of the image are added to the lists.

(d) All timestamps are converted to Unix time so they can be compared. The list of times is then converted to a list of times elapsed since the beginning of the minute of observations.

(e) A check is performed to test if all the image time stamps are in order. This is done because during testing we had a few incidents of "time travel", where an image would have a later timestamp than the subsequent image. We believe this issue was resolved by placing the GPS unit outside the dome. If time travel is discovered, a plot showing the event and the surrounding 100 frames is saved.

(f) A plot showing the time elapsed since the beginning of the minute as a function of frame number is saved.

12.3   Functions

12.3.1   readxbytes

- Written by Mike Mazur.
- Read in the specified number of bytes (for reading .rcd images).
- Always used after the file has been opened, and the **seek** function has been used to read until the starting byte.

Input:

1. number of bytes [int]

Returns:

1. data in the specified number of bytes[array]

12.4   Required Modules

- sys
- matplotlib.pyplot
- astropy.time
- pathlib

13   VizieRquery.py