Colibri Secondary Pipeline

Rachel A. Brown

May 2022

Contents

This document describes the secondary data reduction pipeline for the Colibri telescope array. It includes functions written by Mike Mazur to add .rcd file handling. There is also a function written by Emily Pass that was in her original version of the Main Pipeline. This pipeline is designed to run on the output from the Colibri Main Pipeline. Once candidate events have been identified, this pipeline matches the events to a set of pre-made occultation kernels.

1 Need to Run

1.1 File Structure

I use the same general file structure on both Pomegranate and the Colibri computers. All of the scripts in this document, and the Colibri pipelines, are designed to work within this file structure.

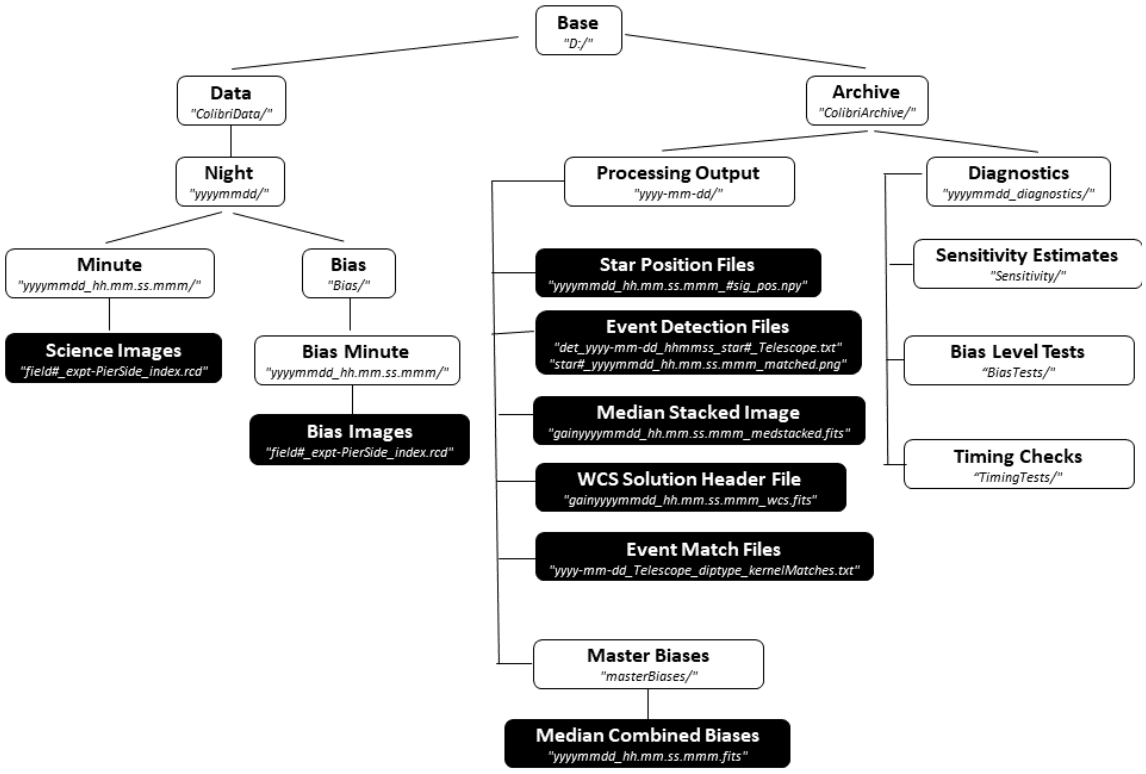

Figure 1: Tree showing the expected data file structure on each of the three Colibri telescope computers, and on Pomegranate. All scripts run on Colibri data work within this structure.

Colibri project files are stored in the *"Base"* directory. On the Elginfield computers this is the *"D:"* drive. On Pomegranate, this is */rbrown/Documents/Colibri/**telescope***, where ***telescope*** is either *"Red"*, *"Green"*, or *"Blue"*. Data for the project is divided in two categories: *data* (*"Base/ColibriData/"*) and *archive* (*"Base/ColibriArchive/"*). All the scripts below have a *base_path* variable that can be changed to the *base* directory for any file system. As long as the data and archive files are arranged with the same structure described below, the scripts should run on other systems by changing only this path.

The *data* folder contains the science and bias images and is further subdivided by observation *night* (ie *"20210804/"*) and *minute* (ie *"20210804_05.03.22.121/"*). Bias images are stored in a *bias* folder in the *night* directory, and are also subdivided by *bias minute*.

Archive files include any of the pipeline outputs or diagnostic script outputs. Pipeline outputs (star position files, detection files, median combined biases) are stored in an *output* folder, labelled by the date of processing (ie *"2022-06-16"*). Diagnostic outputs (sensitivity estimates, timestamp checks, bias subtracted images etc) are stored

in a *diagnostics* folder, labelled by the date of observation (ie *"20210804_diagnostics*). This is further subdivided by the diagnostics performed (ie sensitivity estimates, timing tests, median combined biases, bias subtracted single images). Some of these are further subdivided by *minute* to avoid mixing up diagnostics taken at multiple times throughout a single *night*.

1.2 Required Modules

- numpy
- astropy
- pandas
- matplotlib
- datetime
- sys
- os
- time
- argpparse
- VizieR_query.py
- getRAdec.py
- astrometrynet_funcs

2 Algorithm

The following is a basic overview of the steps the Colibri Secondary Pipeline takes to reduce the data. For a more detailed description of each function, see Section 3. A basic overview of the pipeline flow is shown in the below flowchart:

1. Parameters for running the pipeline are defined. This includes the name of the telescope, the gain level for the .rcd images (*high* or *low*), the solution polynomial order for astrometry.net, the date of observation, the date the main pipeline was run, and the base filepath for accessing data and saving observations.

2. A pre-made set of occultation kernels is loaded into the program. These should be made with KernelGeneratorGUI (see ColibriPipeline GitHub repository) and saved in a single text file. The corresponding kernel parameters should be saved in a separate file.

3. The correct filepaths to the input files are set up. A list of all the output detection .txt files from the main pipeline is made. A list of all the median combined images saved to the archive directory is obtained. An empty dictionary to hold coordinate transformations is created. Empty lists to hold results for both diffraction and geometric events are created.

4. Each detection file is processed.

    4.1 Data from the detection .txt file is read in. This includes the light curve (flux values over time), event frame number, star x coordinate, star y coordinate, event time, event type, star med, star std, and star number for identification.

    4.2 A coordinate transformation from image coordinates (X, Y) to ecliptic coordinates (RA, declination) is calculated using astronometry.net's web service. The transformation is passed to the function **getRADec-Single** in the script **getRADec.py** to calculate the current star's coordinates in RA and Dec.

    4.3 A query is sent to Gaia using **VizieR_query.py** to get a catalog of stars in the region around the current field of view.

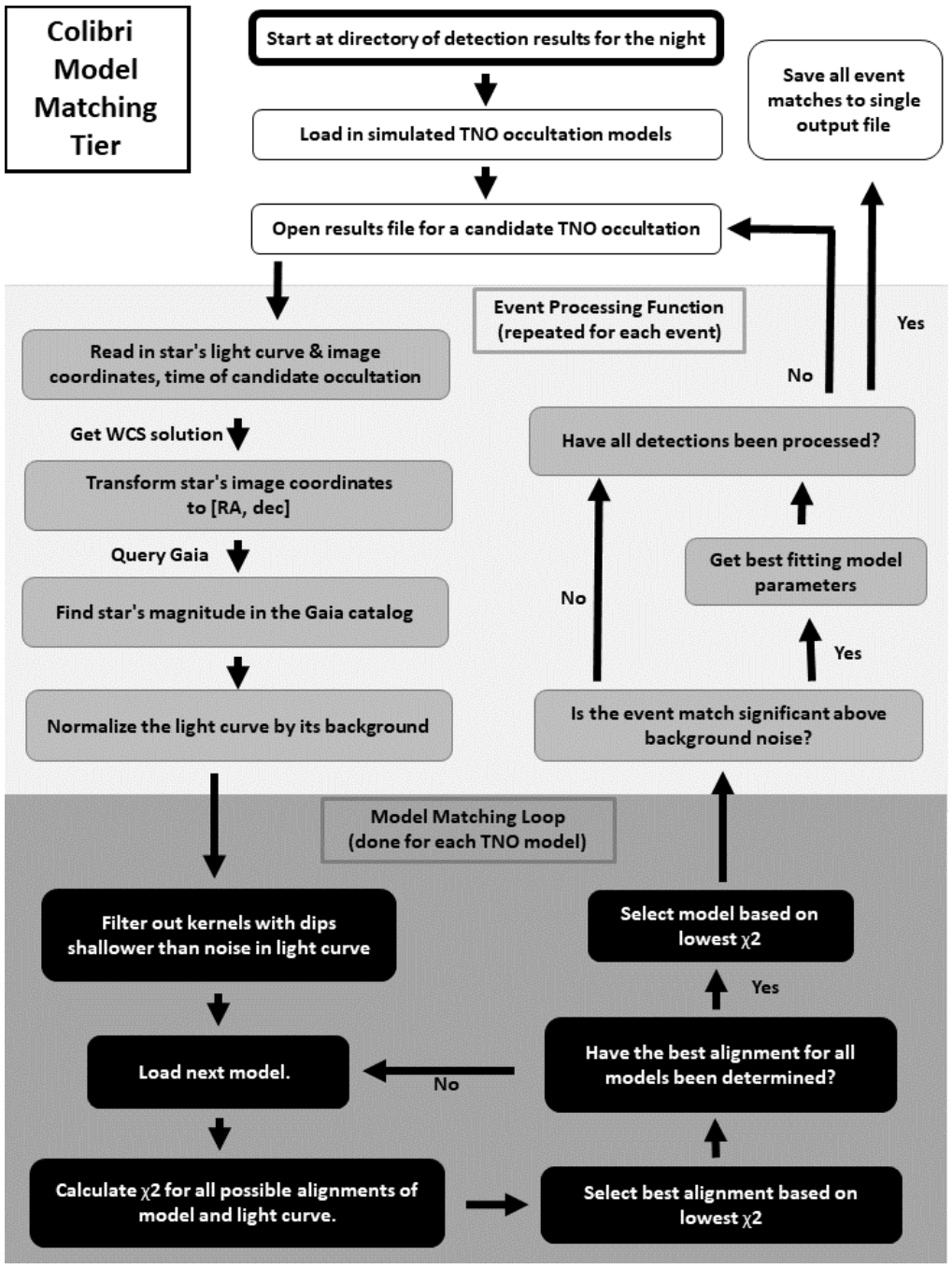

Figure 2: Flowchart for secondary pipeline

4.4 The current star is matched with a star in the Gaia catalog using. Note that on rare occasions this function is unable to find a match and will return a table of 'nan' values. This can be due to the star being fainter than the magnitude limit given by Gaia (for example two faint angularly close stars appearing as one brighter star in our images), or because the star is a missed detection on our end. These should not happen very frequently.

4.5 The event is matched with an occultation kernel from the set generated earlier. Before this is done, the kernels are checked to see if they contain dips that are within one σ of the light curve's mean. If the deepest trough of the kernel is shallower than our expected noise level, the kernel is removed from the set. The best match is determined by the function **kernelDetection** (Section 3.2) based on the kernel with the lowest χ^2. This returns the best matching kernel index, the χ^2 value for this match, the best starting

frame index for the match, and the parameters of the best matching model (TNO radius [m], stellar angular diameter [mas], impact parameter [m], shift adjustment [frames]). This information is appended to the results list, and is also saved in a plot during the matching process.

5. All of the results are written to a single text file.

3 Functions

3.1 diffMatch

- Originally written by Emily Pass
- Calculates the best alignment of a kernel and the light curve based on minimum χ^2.

Input:

1. kernel array
2. light curve array
3. Poisson noise levels

Returns:

1. statistical minimum for the best alignment
2. starting index for the best alignment

algorithm

1. The statistic for determining the best alignment as well as the starting index are set to a starting value of infinity. This is so at least one alignment will be returned.
2. Since the kernel is shorter than the light curve, the entire range of starting indices for the overlap is checked. For each one, a χ^2 value is calculated.
3. If the new χ^2 is lower than the previous, the best starting location is set to the current starting location.
4. Once all starting indices have been tested, the one with the lowest χ^2 is returned.

3.2 kernelMatch

- Modified from Emily Pass' original version
- Match a candidate occultation with an occultation model kernel

Input:

1. tuple of event data (fluxes, times, frame number)
2. set of pre-made kernels [array]
3. ID number of current star
4. directory to save results to [path object]
5. dataframe with star's coordinates and Gaia data

Returns:

1. index of best matched kernel
2. minimum χ^2 value
3. location of kernel alignment starting frame

4. parameters of best matched kernel

5. If no kernels matched which match criteria, -1 is returned

Algorithm:

1. Filter out any kernels which have dips shallower than 1 σ of the light curve.

2. Make light curve into lists. Calculate the normalized sigma referred to in Eq. 2 of Pass et al. 2018.

3. Calculate an array Poisson noise referred to in Eq. 7 of Pass et al. 2018 for each frame in the light curve.

4. Get light curve with dip and surrounding 10 frames removed to use as the 'background'.

5. Fit a line to this background, and divide the light curve by it to normalize and remove any linear trends.

6. The each kernel is aligned and the best fitting kernel is determined. First, the statistic for determining the best match is set to a starting value of infinity. This is so at least one model will be matched and returned. The following loop runs for each kernel in the set:

   6.1 The current kernel, along with the light curve and Poisson noise level is sent to **diffMatch** (Section 3.1) to be aligned. The χ^2 value from this alignment as well as the starting index for the overlap is returned.

   6.2 If the χ^2 for the current kernel is less than the current best-matched value, then the current kernel is provisionally set to be the 'active' kernel.

7. The criteria for a match as outlined in Eq. 2 of Pass et al. 2018 is tested with the best matched kernel.

8. If the match passes this criteria, the event is plotted with residuals and the parameters for the best fitting kernel.