Colibri Main Pipeline

Rachel A. Brown

May 2022

Contents

This document describes the main data reduction pipeline for the Colibri telescope array. The main pipeline was originally written by Emily Pass in 2018. It has since been modified by Tristan Mills (adding .fits capability) and myself. It also includes functions written by Mike Mazur to add .rcd file handling (2021). A basic overview of the pipeline flow is shown in the below flowchart:

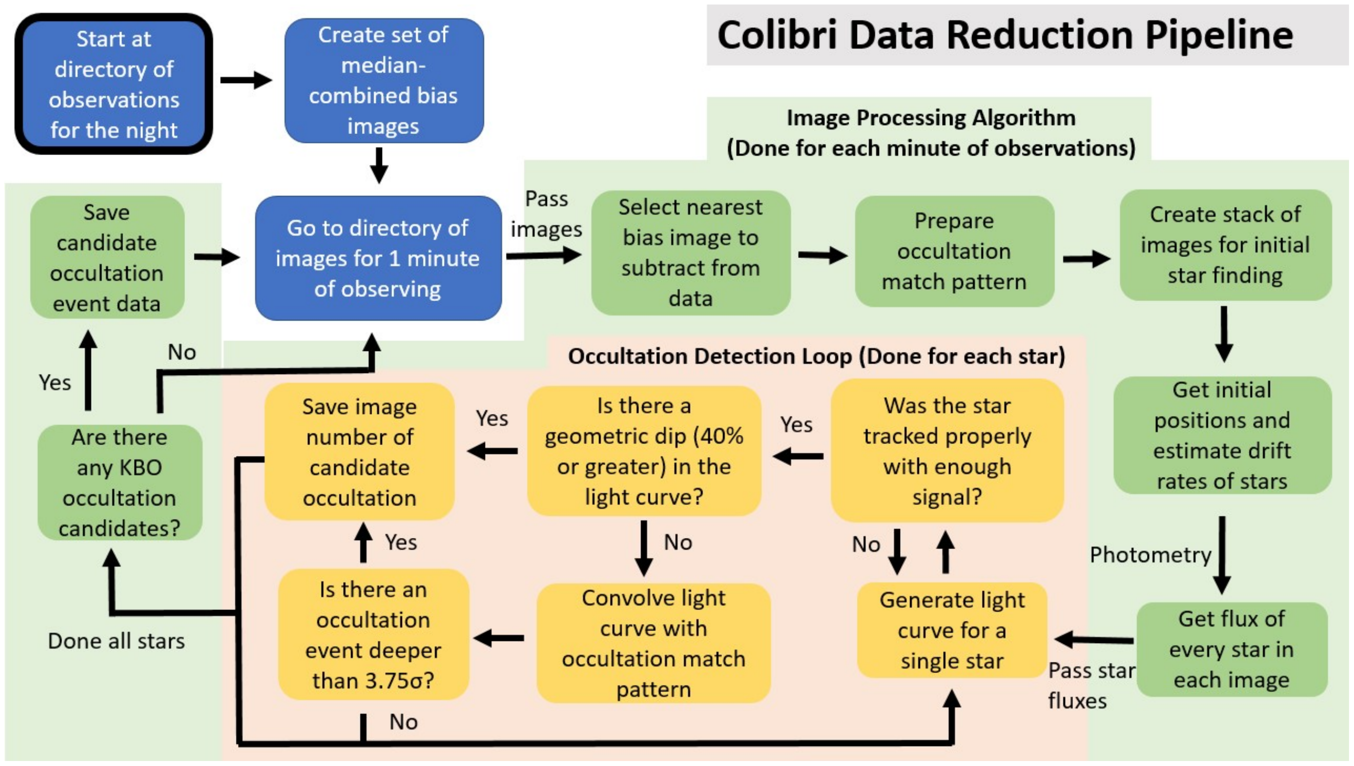
Colibri Data Reduction Pipeline
Start at directory of observations for the night
Create set of median-combined bias images
Go to directory of images for 1 minute of observing
Save candidate occultation event data
Pass images
Image Processing Algorithm (Done for each minute of observations)
Select nearest bias image to subtract from data
Prepare occultation match pattern
Create stack of images for initial star finding
Get initial positions and estimate drift rates of stars
Photometry
Get flux of every star in each image
Pass star fluxes
Occultation Detection Loop (Done for each star)
Generate light curve for a single star
Was the star tracked properly with enough signal?
Yes
No
Is there a geometric dip (40% or greater) in the light curve?
Yes
No
Save image number of candidate occultation
Convolve light curve with occultation match pattern
Is there an occultation event deeper than 3.75σ?
Yes
No
Done all stars
Are there any KBO occultation candidates?
Yes
No

Figure 1: Primary pipeline flow

1 Need to Run

1.1 File Structure

1.2 Required Modules

2 Algorithm

The following is a basic overview of the steps the Colibri Main Pipeline takes to reduce the data. For a more detailed description of each function, see Section 3.

1. Parameters for running the pipeline are defined.

   RCDFiles (boolean) is set to *True* if the pipeline should run on .rcd images. If set to *False* the pipeline will run on .fits images.

   runPar (boolean) is set to *True* to run the pipeline in parallel mode, which will process minute-long datasets on different cores.

   The name of the telescope, the gain (*high* or *low*) for .rcd files, the date of observations, and the base path to the observation and archived data is set here.

2. Filepaths to the directories where the data is stored are defined. Note that filepaths in the pipeline are managed using python's **pathlib** library. This is so the pipeline can be run on both Linux and Windows operating systems, and to make file structure navigating and accessing more straightforward.

3. Bias images are prepared using the **makeBiasSet** function (Section 3.14). This creates a set of median combined "master bias images" to be subtracted from the science images before they are run through the pipeline. During data collection, a set of 50 bias images are taken every 15 minutes to mediate the effects of changing bias levels throughout the night.

4. The Ricker wavelet ("Mexican Hat") kernel is prepared using **RickerWavelet1DKernel** module in **astropy.convolution**. The width of this kernel is determined by the exposure time of an image and the characteristic scale length of a KBO occultation. The kernel will later be convolved with individual star light curves to search for occultation patterns in the data (Section 3.4).

5. The main part of the pipeline is run for each minute-long data set. If running in parallel, the pipeline runs on minute-long directories simultaneously on multiple cores. Otherwise, the directories are run in sequence. If the pipeline is running on .fits files, there is a check in place to see if the conversion has taken place. If not, the .rcd files are converted to .fits files before calling **firstOccSearch**. The function **firstOccSeach** (Section 3.5) is really the main function of the pipeline which runs through all the steps necessary to process the data and search for occultation events. It runs on one minute-long data directory at a time.

   5.1 The filepath to a minute-long dataset (**minuteDir**) is passed to the **firstOccSearch** function (Section 3.5), along with the list of master biases, the generated kernel from Step 4, the exposure time and gain of the images.

   5.2 The folder in which results are saved is created if it doesn't exist already.

   5.3 The median combined bias image that is closest in time to the current minute is selected using the function **chooseBias** (Section 3.2).

   5.4 The detection threshold for star finding and the circular aperture radius for photometry are chosen.

   5.5 A list of image filepaths within the current **minuteDir** is made.

   5.6 The dimensions of the images and the number of images in the directory is determined using either **getSizeFITS** (Section 3.8) or **getSizeRCD** (Section 3.9), depending on the **RCDFiles** parameter set in Step 1. The current minute directory is skipped if there are too few images (set to 3x the kernel length).

   5.7 Star positional data is created from a median combined stack. Image stacking is performed using the **stackImages** function (Section 3.20). This image is passed to **initialFind** (Section 3.12) to perform object detection using the **sep.extract** module. The object positions and half-light radii are saved in a .npy file labeled ***minuteDirName_detectThresh****sig_pos.npy* in the results directory. If there are too few stars found, this step will be repeated until enough detections are made.

   5.8 The average drift rate of the star field over the minute is calculated. This is done by importing the first and last images in the set using either **importFramesFITS** (Section 3.10) or **importFramesRCD** (Section 3.11). Each of these images are passed to **refineCentroid** (Section 3.17) along with the initial star positions to refine the positions using **sep.winpos**. The average drift rate of stars over the minute-long data set is calculated using **averageDrift** (Section 3.1). If the average drift in either the x or y direction is greater than the threshold, drift will be accounted for when doing photometry.

   5.9 A time series for each star over the minute-long data set is created, accounting for star field drift if necessary. For each image in the series, the image file is imported using the function **importFramesFITS** (Section 3.10) or **importFramesRCD** (Section 3.11). This returns the image array and the header time. This information along with the previous image's star positions is passed to **timeEvolve** (Section 3.22) or **timeEvolveNoDrift** (Section 3.23) if the drift is significant. These functions perform aperture photometry using the **sep.sum_circle** module, including local background subtraction in an annulus. The output of this step is a multidimensional array with dimensions [frames, star number, X coord, Y coord, list of fluxes, list of unix times].

   5.10 The time series of each star is checked for candidate occultation events. This is done by sending each star's time series to the **dipDetection** function (Section 3.4). This function runs through the following steps:

      5.10.1 The light curve is checked for tracking failures, and a Signal-to-Noise Ratio threshold of 5 is set.

      5.10.2 First the light curve is checked for any dips greater than 40% of the light curve normalized by the median. If found, the time series and the image frame number of the event are returned to **firstOccSearch**.

5.10.3 If no events of this type are found, the light curve is convolved with the Ricker Wavelet kernel created in Step 4. A check is performed to make sure the time series isn't cut off at the beginning or end of the minute.

5.10.4 The convolved time series is checked for a dip due to diffraction (candidate occultation event). A significant dip is considered to be

$$(\textit{lightcurve minimum}) < (\textit{mean background level}) - 3.75\sigma \tag{1}$$

Where the background is taken to be the entire convolved lightcurve excluding a specified number of buffer frames at the beginning and end of the curve. σ is taken to be the standard deviation of the background. If a dip of this significance is found, the original light curve and the frame at which the dip is found are returned to **firstOccSearch**.

5.11 Light curves containing events are archived to be run through the secondary pipeline. A list of frame numbers where events were detected is created from the results of the dip detection step. A list of light curves containing events is also created. The number of frames on either side of the detected event to archive is calculated as well.

For each detected event a .txt file is created that saves the important data about the event. This file has a header with the name of the image where the event occured, the header date and time for the event, the star coordinates, telescope name, and star field identifier. It also contains a table with the list of image filepaths, the image time (seconds only), and the star flux.

5.12 Printout statements with the number of detected stars and percentage of events are printed to the screen.

6. Steps 1-5 are completed for each minute-long dataset until the entire night is processed. When the pipeline has finished, a directory labelled by the processing date in *ColibriArchive* will contain .npy files with star coordinates for each minute, any master bias images created, median combined images for star detection in each minute, and all the detection .txt files.

3 Functions

3.1 averageDrift

- Determines the median x and y drift rates (px/s) of all stars across a minute-long data set.
- Compares the star positions between two frames, and uses the time difference between these frames to determine the drift rate.

Input:

1. 3D array of [x,y] star positions - [# frames (should be 2), # stars, # position columns (should be 2 - X & Y)]
2. list of header times of each position

Returns:

1. median x drift rate [px/s]
2. median y drift rate [px/s]

Algorithm:

1. Convert header times to unix times (for simple subtraction).
2. Get number of pixels each star has drifted in x and y separately by subtracting the last frame positions from the first frame positions.
3. Get the time interval between the first and last frame.
4. Calculate a drift rate for each star by dividing the amount of drifted pixels by the time. Take the median in x and y.

3.2 chooseBias

- Choose the correct master bias by comparing time to the observation time

Input:

1. Filepath to the directory for the current minute-long dataset.
2. 2D numpy array of [master bias image datetimes, master bias image filepaths]

Returns:

1. Bias image array that is closest in time to observation

Algorithm:

1. Get a datetime object for the current directory using **getDateTime** (Section 3.7).
2. Make an array containing the differences between each master bias datetime and the current datetime. Select the array index with the smallest difference. This is the master bias image that is nearest in time to the time the current directory was created.
3. Load in the image data for the selected master bias image and return this image array.

3.3 clipCutStars

- When the photometry aperture is near the edge of the field of view sets flux to zero to prevent fadeout

Input:

1. List of X coordinates for each star in pixels
2. List of Y coordinates for each star in pixels
3. Length of image in x direction
4. Length of image in y direction

Returns:

1. Indices of stars to remove

Algorithm:

1. Set a threshold for the number of pixels within the edge of the image to remove stars
2. Make arrays of star x and y coordinates.
3. Get indices of stars which have their x coordinates outside the threshold on the right side of the image and on the left side. Get indices of stars which have their y coordinates outside the threshold on the top and the bottom.

3.4 dipDetection

- Checks the light curve of a star for large ('geometric') dips in flux, and for smaller ('diffraction') dips in flux.

Input:

1. Light curve of a star over the minute-long data set as an array of fluxes.
2. Ricker wavelet kernel as an array of normalized counts.
3. ID number of the star.

Returns:

1. Frame number of the detected event. If no event detected, returns an error code:

- *-1*: No events in the minute that passed detection thresholds.
- *-2*: Light curve was too short, star was not tracked properly, SNR below the threshold, or the event was too close to the beginning/end of the series.

2. Light curve of the event (**not** normalized). Returns empty list if no detection.
3. Keyword indicating the type of event detected.
   - *Geometric*: Event is below 40% of the normalized median flux level.
   - *Diffraction*: Event is below 3.75σ of the light curve convolved with the Ricker wavelet kernel.

   Returns an empty string if no event detected.

Algorithm:

1. Prune light curves. Any leading or trailing zeroes are removed from the array. If, after this is done, the light curve is empty, it returns an error code indicating this.
2. Checks are performed to test if the light curve is good enough to look for dip events. Sets the ideal number of images per minute (2400 at 40 Hz). Sets a minimum Signal to Noise threshold, where SNR is defined as the light curve median divided by the standard deviation. Sets a minimum number of array elements for the light curve.
   - If the lightcurve has fewer elements than the minimum number the star is rejected.
   - If the difference in the star's mean flux between the beginning and the end of the minute is greater than the standard deviation, this indicates that the star has not been tracked properly. The star is rejected.
   - If the star's SNR (median/standard deviation) is below the limit set above, the star is rejected.

   There is an option here to uncomment a line that will return the star's light curve without checking for dips. This was used for debugging.
3. The light curve is convolved with the Ricker wavelet kernel using the function **convolve_fft** from astropy. The index of the minimum value in the convolution is found. This value of this minimum is found.
4. The light curve is checked for large (*geometric*) dips. The threshold for a geometric dip (*geoDip*) is set Note that this will look for a dip that is 1.00 - threshold (ie if $geoDip = 0.6$, dips of 40% or greater will be detected). The light curve is normalized by its median value.
   If the value of the normalized light curve at the index of the convolution's minimum is less than the threshold, the dip event is returned.
5. If no geometric dip has been detected, the light curve is checked for smaller (*diffraction*) dips. If the location of the convolution's minimum is too close to the beginning or end of the light curve, the star is rejected. Otherwise, a background zone is created. This background zone is the entire light curve excluding a buffer at the beginning and end of the series. A dip detection threshold (currently set at 3.75) is set.
   The criteria for a diffraction dip is:

$$minimum(convolution) < mean(backgroundzone) - threshold \times \sigma \tag{2}$$

   where σ is the standard deviation of the background zone. If this criteria is satisfied, the dip event is returned. If not, the star is rejected.

3.5 FirstOccSearch

3.6 getBias

- Make a median combined bias image to use for bias subtraction.

Input:

1. Filepath to directory containing bias images [pathlib path]
2. Number of bias images to include in the median combine [int]

3. Gain ('high' or 'low') of .rcd or .fits images. [string: 'high' or 'low']

Returns:

1. Median combined bias image [2D array]

Algorithm:

For .fits files:

1. Create a variable with the name of the 'check file'. This is an empty file that indicates whether a .fits conversion has been done to each of the .rcd files in the directory.
2. If this file doesn't exist, the .fits to .rcd conversion script (**RCDtoFTS.py**) is called to do the conversion with the specified gain. If the file does exist, the files are already in .fits format and the script can proceed.
3. Make a list of all .fits files in the directory. Prepare an empty list to hold data for these images.
4. For each image in the list of files, read in the data and add the image array to the bias list.
5. Take the median of these images to create a new median combined image of the biases. Return this.

For .rcd files:

1. Make a list of all .rcd files in the directory.
2. Import each .rcd image in the list using the function **importFramesRCD** (Section 3.11). Pass it the list of bias images, the starting index (set to 0), the number of biases to import, and empty array of the same dimensions to serve as the 'bias' for these images, the gain value. These imported images are returned as a list.
3. Take the median of the images in the list to create a new median combined image of the biases. Return this.

3.7 getDateTime

- Get a datetime object from the name of a data directory using the python module **datetime**. Datetime objects can be manipulated and compared more easily than basic strings when doing math that involves dates and times.
- Each minute long dataset is contained in a directory labelled *'yyyymmdd_hh.mm.ss.uuu'*. This function takes this string and converts it into a datetime object *'yyyy-mm-dd'* and *'hh:mm:ss:uuu'*

Input:

1. Filepath to minute-long dataset directory. [path object]

Returns:

1. Date and time contained in the directory name [Datetime object].

Algorithm:

1. Separate the part of the directory name string that contains the time.
2. The date is taken using the global *obs_date* variable set at the beginning of the main script.
3. A **datetime** *time* object is created from the directory time.
4. This is combined with the directory date to create a **datetime** *datetime* object.

3.8 getSizeFITS

- Get the number of .fits images in a data directory (for minute-long data sets should be around 2400). Get the dimensions of .fits images.

Input:

1. List of image filepaths in the data directory [list of path objects]

Returns:

1. width of .fits image [px]
2. height of .fits image [px]
3. number of .fits images in directory [int]

Algorithm:

1. Get the number of images in the directory by finding the length of the list of image filepaths.
2. Open the first image header in the directory to get the X, Y axis dimensions.

3.9 getSizeRCD

- Get the number of .rcd images in a data directory (for minute-long data sets should be around 2400). Get the dimensions of .rcd images.
- written by Mike Mazur.

Input:

1. List of image filepaths in the data directory [list of path objects]

Returns:

1. width of .rcd image [px]
2. height of .rcd image [px]
3. number of .rcd images in directory [int]

Algorithm:

1. Get the number of images in the directory by finding the length of the list of image filepaths.
2. Open the first image header in the directory to get the X, Y axis dimensions.

3.10 importFramesFITS

- Reads in multiple images from .fits files in the directory

Input:

1. List of image filepaths in the data directory [list of path objects]
2. Starting frame number [int]
3. Number of frames to read in [int]
4. Bias image [2D array of counts]. Must be the same size as the images in the directory.

Returns:

1. List of image data arrays [list of floats]

2. Array of header times for these images [1d array of strings]

Algorithm:

1. Create empty lists to hold the image arrays and header times.
2. Make a list of filenames to read in. This is done by list comprehension; taking the input list values that are between the given starting index and the starting index plus the given number of frames to read.
3. Loop through each image filepath in the new list.

   3.1 Open the image using the **fits** module in **astropy**. Read in the image header. Read in the image array (counts) and subtract off the bias image. Get the timestamp of the image.

   3.2 Check if there is a time error and correct it. In some cases during testing, the header timestamp would read a time of "29:00h", which of course is an error. This section of the code was written to identify the issue and patch it so we can continue processing.
   The header time string is separated out to get the hour and minute of the image's timestamp. The minute the parent directory was created is taken from the directory filename. If the hour is greater than 23, we have an issue.
   The correct hour is taken from the parent directory name. If the image timestamp minute is less than the directory name minute, the hour must have rolled over during the minute of observations. In this case, 1 will needed to be added to the correct hour. Note that during the process of developing the pipeline, only Red computer was using UTC. At the time of writing this documentation, both Red and Green use UTC. Since all image timestamps are in UTC, any telescope computer that is using local time will need to add an additional 4 (5 if daylight savings) hours to the corrected time. These lines are commented out in the script.
   The incorrect hour is replaced by the corrected one in the timestamp string.

   3.3 The image is closed. The image array is appended to the list of images, and the timestamp string appended to the list of header times.

4. The list of images is made into a multidimensional array. This is for ease of operations later on in the pipeline.
5. The data is reshaped if necessary, and count values are made into floats.

3.11 importFramesRCD

- Reads in multiple images from .rcd files in the directory.
- Adapted from **importFramesFITS** (Section 3.10) by Mike Mazur.

Input:

1. List of image filepaths in the data directory [list of path objects]
2. Starting frame number [int]
3. Number of frames to read in [int]
4. Bias image [2D array of counts]. Must be the same size as the images in the directory.
5. Gain keyword [string - either *"high"* or *"low"*]

Returns:

1. List of image data arrays [list of floats]
2. Array of header times for these images [1d array of strings]

Algorithm:

1. Create empty lists to hold the image arrays and header times. Also set the width and height of the images (in px) manually.
2. Make a list of filenames to read in. This is done by list comprehension; taking the input list values that are between the given starting index and the starting index plus the given number of frames to read.
3. Loop through each image filepath in the new list.
   - 3.1 Open the image using the **fits** module in **astropy**. Read in the image header. Read in the image array (counts) and subtract off the bias image. Get the timestamp of the image.
   - 3.2 Check if there is a time error and correct it. In some cases during testing, the header timestamp would read a time of "29:00h", which of course is an error. This section of the code was written to identify the issue and patch it so we can continue processing.
     The header time string is separated out to get the hour and minute of the image's timestamp. The minute the parent directory was created is taken from the directory filename. If the hour is greater than 23, we have an issue.
     The correct hour is taken from the parent directory name. If the image timestamp minute is less than the directory name minute, the hour must have rolled over during the minute of observations. In this case, 1 will needed to be added to the correct hour. Note that during the process of developing the pipeline, only Red computer was using UTC. At the time of writing this documentation, both Red and Green use UTC. Since all image timestamps are in UTC, any telescope computer that is using local time will need to add an additional 4 (5 if daylight savings) hours to the corrected time. These lines are commented out in the script.
     The incorrect hour is replaced by the corrected one in the timestamp string.
   - 3.3 The image is closed. The image array is appended to the list of images, and the timestamp string appended to the list of header times.
4. The list of images is made into a multidimensional array. This is for ease of operations later on in the pipeline.
5. The data is reshaped if necessary, and count values are made into floats.

3.12 initialFind

- Locates stars in an image.

Input:

1. Flux data for an image [2D array]
2. Detection threshold coefficient for star finding

Returns:

1. Zip object for all objects containing X coordinate [px], Y coordinate [px], and half light radius [px].

Algorithm:

1. Extract the background for the image. This is done using functions in the **sep** module.

   First, a copy of the data is made. Then, a 2D array of the variable background across the image is extracted using the **sep** class **Background**.

   This background array is subtracted from the image array copy using the **sep.Background** method **subfrom**.

   The significance threshold for star detection is set. This is determined by multiplying the given detection threshold level with the global root-mean-square of the background array (using the **sep.Background** attribute **globalrms**.
2. Stars are identified in the background subtracted image. This is done using the **sep** funcition **extract**. The significance threshold calculated above is passed to the function as the pixel threshold value for detection. The object returned from this extraction is a structured array. See **sep.extract** documentation for more details.
3. The star's half-light radius is approximated as half the star's radius as returned by **sep.extract**.
4. A zip object containing the X, Y coordinates and half light radius for each star is generated.

3.13 nb_read_data

3.14 makeBiasSet

3.15 readRCD

3.16 readxbytes

3.17 refineCentroid

3.18 runParallel

3.19 split_images

3.20 stackImages

3.21 sumFlux

3.22 timeEvolve

3.23 timeEvolveNoDrift