Fire Opal Summer Project 2019 Code Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Email |
| 1 | Maureen Cohen | 23 August 2019 | mc53@hw.ac.uk |

Table of Contents

[1. fire\_opal.py 2](#_Toc17453170)

[A. convert\_to\_grey 2](#_Toc17453171)

[B. cloudy\_or\_clear 2](#_Toc17453172)

[C. line\_from\_two\_points 2](#_Toc17453173)

[D. process\_image 2](#_Toc17453174)

[E. process\_list 4](#_Toc17453175)

[2. fire\_opal\_postprocess.py 5](#_Toc17453176)

[A. Classes and functions 5](#_Toc17453177)

[B. Data extraction 5](#_Toc17453178)

[C. Identification of satellite trail over multiple images 5](#_Toc17453179)

[D. Determination of direction of satellite trail 6](#_Toc17453180)

[E. Formatting of data as IOD and saving in txt file 6](#_Toc17453181)

[F. Orbit determination using outside software 7](#_Toc17453182)

[3. Useful links 8](#_Toc17453183)

**Bold text** discusses input parameters and thresholds set in fire\_opal\_settings.py

Green text discusses potential future improvements to the code.

Red text describes issues that could lead to errors in some cases and might be difficult to identify simply from reading the code.

# 1. fire\_opal.py

This is the main Fire Opal program. It batch processes images taken by a Fire Opal camera, finds satellite streaks, extracts data, and writes it to a .txt file.

## A. convert\_to\_grey

RGB to greyscale image conversion. The function sums the three RGB values and divides by 3\*255.00 to create grey values. Output values are between 0 and 1.

## B. cloudy\_or\_clear

Sorting of images into cloudy (discarded) and clear (passed on for further processing).

Steps:

* *Use a Gaussian filter (σ given by the variable* ***cl\_sigma****, set at 10) to create a blurred version of the image.*
* *Subtract blurred version from original image*
* *Select a 500x500 subsection of the image*
* *Count the number of pixels in this subsection with brightness <* ***cl\_background\_thresh.*** *This is considered the portion of the image that is "background."*
* *Ignoring the background, calculate the percentage of meaningful pixels brighter than* ***cl\_lower\_thresh***
* *If this percentage is greater than 0, the image is considered clear*

**cl\_background thresh:** We ignore the background pixels to get a more meaningful value for the percentage -- otherwise it may be so small it gets rounded to 0.

**cl\_lower\_thresh:** Chosen based on comparison of histograms showing pixel brightness distribution for cloudy and clear nights. Visual inspection shows that cloudy nights do not have pixels brighter than this threshold.

**Changing thresholds**: Thresholds could be fine-tuned by trial and error. The specific values come from the pixel brightness distribution curves for cloudy and clear images.

## C. line\_from\_two\_points

Calculation of line slope and intercept from two (x, y) inputs. These values can be used in post-processing to identify a satellite trail across multiple images.

 , 

**Future development**: This function could return the Hesse normal form of the line fit instead. If necessary to improve accuracy, the normal form could ultimately be used in the grouping step in post-processing instead of the current method of consecutive image grouping.

## D. process\_image

Processing of images and extraction of data into a .txt file.

Steps:

* *Use rawpy to convert RAW image data into RGB*
* *Convert to grey using convert\_to\_grey function*
* *Check if cloudy or clear using cloudy\_or\_clear, discard cloudy, continue with clear*
* *Scale 0-1 values up to 0-255. This is necessary for compatibility with OpenCV's data type (unsigned integer 8-bit)*

Important note about OpenCV: While Python indexes images as (rows, columns) the OpenCV library indexes them as (columns, rows). This means that if you want to manipulate a specific pixel in e.g. numpy, you would access image[125, 250] to get the value at y = 125, x = 250. If you want to e.g. draw a red dot on this pixel using OpenCV, you must draw it at image[250, 125].

* *Use OpenCV's Canny edge detector to find edges.*

The Canny edge detector identifies edges based on the image gradient. The function uses the Sobel gradient calculated within an **apertureSize** or kernel. It needs a lower and an upper threshold, **definitely\_not\_an\_edge** and **definitely\_an\_edge**. The Canny function classifies pixels with a gradient lower than **definitely\_not\_an\_edge** as not edges and sets them to black. It classifies pixels with a gradient higher than **definitely\_an\_edge** as edges and sets them to white. If a pixel's gradient is between these two, it is classified based on whether the neighbouring pixels are edges or not. The result is a binary black-and-white image. This pre-processing step is very helpful in getting a useful result from the Hough transform.

**Changing thresholds:** The thresholds have been set by trial and error, but could be calculated specifically for each image based on expected ranges of gradients. OpenCV's Sobel gradient function should be used when finding values such as the average gradient, standard deviation, etc.

* *Use OpenCV's HoughLinesP function to find lines*

The first argument is the binary input image. Second and third arguments are the resolution of the r and θ parameters. The **line\_votes** argument specifies the minimum number of pixels (length X width) a line must have to be detected. The **minLineLength** is the minimum length of a line and **maxLineGap** is the maximum distance allowed between segments in order for them to be considered the same line.

The output takes the form [x1 y1 x2 y2], the endpoints of one detected line.

* *HoughLinesP returns the endpoints of all lines it has found as two sets of (x, y) coordinates. Average these results to get a streak.*

Because the satellite streaks are relatively wide (e.g. 20-50 pixels), the HoughLinesP function can't tell which of the end pixels are the "real" endpoints, and returns multiple overlaid lines corresponding to the same streak. We average these values to get one set of endpoints. The code can currently find one streak per image.

**Future development**: Create code that can find two or more streaks. This might be done using the itertools or more\_itertools packages to iterate over the list of endpoints and sort them into groups based on how close to each other they are.

* *Draw a box around the satellite streak and save this subsection of the image as .png file.*
* *Upload .png file to Astrometry.net, which returns a WCS file containing the astrometric conversion data.*

An account on nova.astrometry.net and an API key are needed for this step.

* *Use Astropy's WCS package and the WCS file to convert the (x, y) endpoints into right ascension and declination in degrees.*
* *Calculate the slope and intercept of a line fit to the streak.*
* *Collect data: filename, two (x, y) endpoints, two (ra, dec) endpoints, slope, intercept, timestamp, and two endpoint times.*

Each endpoint is associated with a time. By default, endpointa\_time is associated with (x1, y1) and endpointb\_time is associated with (x2, y2). This assumes all trails are moving in the "forward" direction (away from the origin); this will be checked and revised if necessary in post-processing. The endpointa\_time is taken from the timestamp of the image. Since the shutter speed is known to be 5 seconds, the endpointb\_time is simply 5 seconds later. These values can be modified if more accurate information about the camera timing and shutter speed is obtained.

* *Write all data to a .txt file*

## E. process\_list

Processing of a list of images and maintenance of a processing record. This function creates a processing record to keep track of which filenames have already been processed. It runs through a list of images from a directory, skips images that have already been processed, and passes ones that have not to process\_image.

The processing record also keeps track of the result of process\_image for each input image. Successful streak detections are recorded as clear\_streak. Cloudy images are recorded as cloudy and clear images without a streak are recorded as clear\_streakless. If an error occurs during processing, the program writes the filename and ERROR to the processing record.

# 2. fire\_opal\_postprocess.py

The input to the post-processing is the .txt file output by fire\_opal.py. The script first defines the data objects and functions needed for post-processing, then carries out a series of steps to extract and clean up the data, identify unique satellite trails over multiple images, discard satellites for which there is insufficient data for orbit determination, determine the direction each satellite is moving, and write the data into an individual IOD-formatted .txt file for each individual satellite in the batch.

## A. Classes and functions

The post-processing uses three classes of data objects to organize and manipulate the satellite streak data:

1. StreakyImage: Represents an image in which a streak has been detected. A StreakyImage data object stores the filename of the image, the timestamp and serial number of the image, and a Streak object or a list of Streak objects.

**Future development**: If fire\_opal.py is scaled up to detect multiple streaks in the same image, these can be stored as Streak objects in a list associated with a StreakyImage object. The code could append streaks in the .txt file that have the same filename to a list and assign this list to one StreakyImage.

1. Streak: Represents an individual streak in an individual image. A Streak object contains a large amount of data about a streak and the image it is found in, not all of which is used in the present code, but which could be used in variations or future developments of it.
2. OrbitalPoint: Represents a point used in an orbit determination calculation. An OrbitalPoint consists of the filename of the image containing it, a right ascension, a declination, and a time.

The function d2HMS takes as input a right ascension and declination coordinate pair in degrees and returns the same coordinate pair in hours, minutes, seconds in IOD format. It is used to format the data as needed for the orbit determination package.

## B. Data extraction

In this section, the data is extracted from the input .txt file, duplicate lines are removed, strings are converted into ints or floats as needed, and all data is assigned to a data object class for further processing. The code expects the input file to have \n line separators and data values separated by commas. If a different separator (such as a white space) is used to separate values, the split() function must be edited to reflect this.

## C. Identification of satellite trail over multiple images

Orbit determination requires at least three endpoints. Since one image contains maximum two endpoints, we need at least two images of the same satellite trail to determine its orbit. This section of the code identifies the same satellite in multiple images. It groups the streaks representing the same satellite in a list, then appends the list to a master list. The current version of the code groups images using the image serial number. It assumes that images containing streaks will typically be separated by at least one, but usually many, images without streaks. Batch processing so far supports this assumption, outputting streaks\_data.txt files with a serial number structure like e.g. [178, 179, 180, 207, 218, 235, 243, 244, 256, 257, 275, 290]. For this example, the grouping function would return a list unique\_satellites that looks like [[178, 179, 180], [243, 244], [256, 257]].

Important note about serial number grouping: If images 597, 598, and 599 contain Satellite A and images 600 and 601 contain Satellite B, the code will group them together as if they were the same satellite. Cases like this are probably infrequent but would lead to errors in orbit determination.

Important note about grouping in general: There are multiple ways to group streaks together. The Python packages itertools and more\_itertools provide a number of pre-made options. A previous version of the code grouped streaks that have the same slope, but this led to problems with rounding - some satellites could only be identified if the slope was rounded to the 2nd decimal places, others only if it was rounded to the 3rd. Grouping by serial number resulted in fewer errors.

**Future development**: The Hesse normal form of a line could be used to characterize streaks for greater accuracy than the slope/intercept. The more\_itertools library includes a function called split\_before that could split up the list of normal vectors or slopes when it encounters a sudden change in the value.

## D. Determination of direction of satellite trail

The code takes one group of image serial numbers (which should all be for images of the same satellite) at a time and creates a list of the StreakyImage objects matching those serial numbers. This list is consecutively ordered in time.

The HoughLinesP from which we obtained the endpoints of the streak returns the data in the form [x1 y1 x2 y2]. The x1 coordinate is the one closest to the origin (top left-hand corner). The code compares the value of the x1 coordinate in the first and second image of the group. If the x1 coordinate of the second image is greater than that of the first, the trail is moving "forward," and endpointa\_time remains assigned to the (x1, y1) spatial coordinates (i.e. the code simply leaves the data as is). If the x1 coordinate of the first image is greater than that of the second, the trail is moving "backward," and the endpointb\_time and endpointa\_time values are swapped.

If the x1 coordinates in both images happen to be equal, the code performs the same process using the y1 coordinate instead, with downwards motion (away from the origin) being considered "forward" and upward motion "backward."

Once the times are correctly assigned to endpoints, the data is stored in an OrbitalPoint object and the collected list of OrbitalPoints for one satellite is appended to a master list of detected satellites.

## E. Formatting of data as IOD and saving in txt file

This section creates a separate .txt file for each unique identified satellite. The files are numbered satellite1, satellite2, satellite3,... etc. Each OrbitalPoint associated with a unique satellite is written to that satellite's file as a separate line of data. The data is written in IOD format.

## F. Orbit determination using outside software

An IOD formatted .txt file can then be fed into an external orbit determination software, which will return the orbital elements for the satellite.

# 3. Useful links

[IOD format description](http://www.satobs.org/position/IODformat.html)

[itertools documentation (groupby function)](https://docs.python.org/2/library/itertools.html)

[more\_itertools documentation (consecutive\_groups and split\_after functions)](https://more-itertools.readthedocs.io/en/stable/)

<nova.astrometry.net>

[opencv feature detection documentation (Canny and HoughLinesP functions)](https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html)