Astronomy Will Not Trail Off: Novel Methods for Removing Satellite Trails From Celestial Images

Owen M. Dugan
Stanford Online High School
Redwood City, California
odugan@ohs.stanford.edu

Abstract

SpaceX's Starlink satellite network promises world-wide high-speed internet access. With up to 42,000 satellites to be deployed, however, the Starlink satellite network may significantly degrade ground-based astronomical research and imaging due to trails (e.g., light reflections or emissions) from passing satellites. The difficulty of removing the effects of satellite trails on night sky images is recognized because accurately identifying satellite trails is challenging and satellite trails effect not only the brightness measurements of stars they pass in front of but also the brightness measurements of stars in the vicinity of the satellite trails. Novel algorithms were developed and coded to accurately identify and remove satellite trails and reduce their effects on photometry. Platesolving is used to identify stars within an image, and an algorithm is used to determine the radius of each star identified. Identified star brightnesses are replaced with median nearby image brightness values. Satellite trails are identified by examining each possible line traversing the image, with recursive sizing using area interpolation implemented for large images to reduce processing time. Area and/or cubic interpolation is employed to optimize satellite trail modeling. The code returns to the original image with stars, and a Gaussian brightness profile is developed for the satellite trail to account for satellite trail effects across the entire image. The satellite trail is removed by applying the additive inverse of the fitted Gaussian to every pixel in the image. Significant reductions in the effects of satellite trails on images captured using Earth-based equipment are observed while improving image photometric accuracy. Additional novel solutions for preserving star brightnesses directly under the satellite trails are explored.

1. Introduction

SpaceX plans to launch up to 42,000 satellites into orbit in support of Starlink, a network which will provide high speed internet access to most locations on Earth, Bowler (2019). Astronomers are deeply concerned about the impact of these satellites on astronomical observations and research, Siegel (2019). Indeed, some have predicted that the Starlink network will be the "end of Astronomy," Hall (2019). The American Astronomical Society has engaged SpaceX in efforts to mitigate the effects of such satellites, but remains concerned:

> The American Astronomical Society notes with concern the impending deployment of very large constellations of satellites into Earth orbit. The number of such satellites is projected to grow into the tens of thousands over the next several years, creating the potential for substantial adverse impacts to ground- and space-based astronomy. These impacts could include significant disruption of optical and near-infrared observations by direct detection of satellites in reflected and emitted light. *EarthSky* (2019).

The present research proposes to mitigate the effects of satellite trails by developing and implementing novel algorithms in a Python program for detecting satellite trails in images and removing these trails.

2. The Effect of Satellite Trails on Photometry

If a satellite passes through the field of view of an imaging telescope, it can result in inaccurate astronomical measurements for that image. Particularly affected is image photometry. The satellite produces a line through the image with a brightness that is normally distributed as a function of distance to the center of the line. The trail is superimposed with all other objects in the image, causing the image to appear brighter than normal. However, because the brightness of the trail at a given pixel in the image varies with respect to the position of that pixel relative to the trail, some objects in the image appear artificially brighter than others.

Since satellite trails alter exactly what photometry seeks to measure, these trails can significantly impact photometry measurements. Generally, if a satellite is observed in an image, the

image must be discarded. This is an extremely inefficient use of images, especially as the number of satellites increases, and is what this paper seeks to address.

3. Program Overview

The program is divided into segments which are performed sequentially. The first segment identifies and removes stars and hot pixels in an image. With the stars and hot pixels removed, the second segment identifies the path of the satellite through the image. The third segment employs the identified path of the satellite and the original image (with stars) to determine a Gaussian function that best fits the brightness of the satellite trail. Finally, the fourth segment removes the satellite trail from the original image based on the identified path and Gaussian brightness function of the satellite trail.

These four segments are described in more detail below. Several Python packages are used: AstroPy, Robitaille (2013) and Price-Whelan (2018), and AstroPy coordinated packages such as Astroquery and Photutils, Bradley (2019), and packages from the SciPy ecosystem, Virtanen (2020), such as NumPy, van der Walt (2011), and Matplotlib, Hunter (2007).

3.1 Removing Stars and Outliers from image

3.1.1 Star Removal

The first segment of the program begins by locating and removing the stars in the image. This step is critical to success. Satellite trails may be extremely dim relative to the surrounding star field. Further, because the stars are randomly distributed, they often appear to form along lines. As a result of these two factors, the total signal brightness along a line passing through several bright stars can often exceed the total signal brightness along the satellite trail. This makes it difficult to detect a satellite trail instead of a line of stars, as shown in Figure 1.

Rather than attempting to detect a satellite trail from within the star field, the present approach removes the star field and then detects the trail.

A star removal algorithm first identifies the stars in an image and then iterates through each identified star and removes it. The program offers users three options for identifying the stars in an image:

1. Platesolve the image by sending it to the Astrometry.net server, Lang (2010), and receive a file from Astrometry.net containing the locations of all of the identified stars in the image.

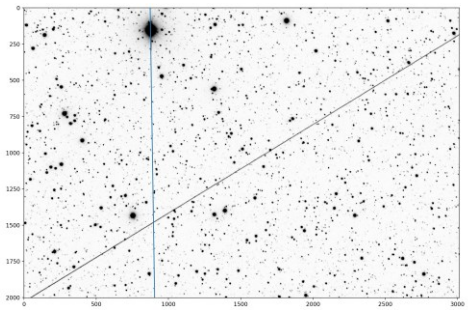


Figure 1: An incorrectly identified satellite trail when stars are not removed. The vertical line indicates the "trail" the program detected. The line running roughly diagonally through the image is the true satellite trail. Note the extremely bright star that the incorrectly identified trail passes through. This star outshines the entire true satellite trail, so the algorithm identifies it as part of the trail.

2. Platesolve the image by sending it to the Astrometry.net server and receive a file from Astrometry.net specifying the image's location and orientation in the sky. In this case, the program uses the Astroquery package to query the Gaia star catalogue for the locations of stars in the image. This produces many more stars than the previous method, and is the star removal algorithm's default method.

3. Accept an already platesolved image. In this case, the program once again uses the Astroquery package to query the Gaia star catalogue for the locations of stars in the image.

The star removal algorithm then iterates through each recognized star. It first determines the radius of each star in pixels. To do so, it begins with a radius r = 1 and continually increments the radius by 1. For each r, it selects 50 equally spaced points a distance of r from the center of the identified star. These 50 points are defined by:

$$(x_k, y_k) = \left(x_c + r\cos\left(\frac{k\pi}{25}\right), y_c + r\sin\left(\frac{k\pi}{25}\right)\right), \quad k = 0, 1, 2, \ldots, 50, \qquad (1)$$

where (x_c, y_c) is the center of the star. The algorithm finds the closest lattice point to each (x_k, y_k) coordinate pair and calculates the median brightness of the pixels corresponding to these lattice points. If this median brightness is greater than the median brightness of the entire image, the algorithm concludes that r is smaller than the radius of the star and continues to increment r. If on the other hand, the median brightness of the 50 points is less than or equal to the median brightness of the entire image, the algorithm determines that the current radius

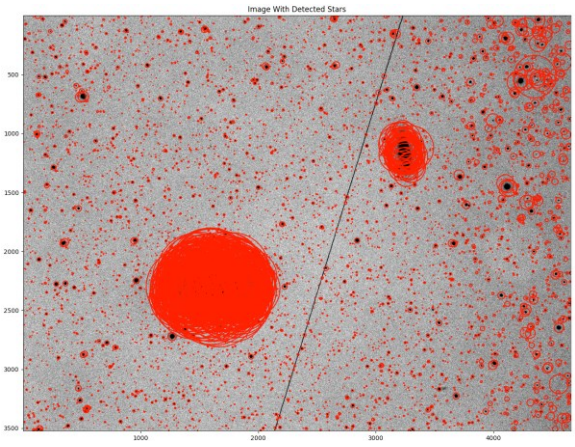


Figure 2(a). An example image with each star identified. Each identified star is surrounded by a circle marking the detected extent of the star. The two large covered regions are galaxies.

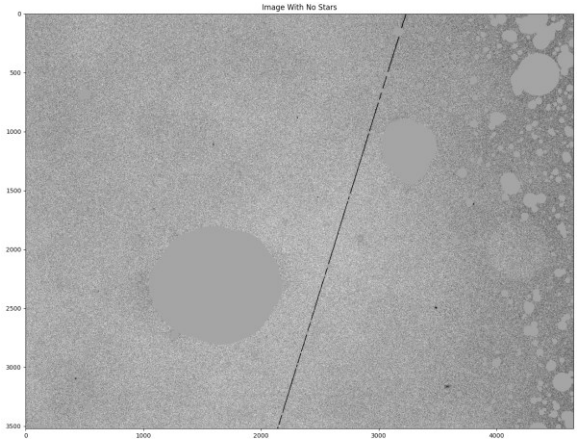


Figure 2(b). The image from Figure 2(a) with the detected stars removed.

marks the edge of the star, and records that the star has radius r. Figure 2(a) demonstrates an example of the algorithm's star fits.

Once the algorithm has determined the radius of a star, it sets the brightness of all pixels which are a distance of less than r+1 from (x_c, y_c) to the median brightness of nearby background pixels. This process is repeated for each star identified in the image. Figure 2(b) shows an example resulting image with the stars removed.

Note that the median brightness of the image is used as an estimate of the background brightness of the image. This is a reasonable assumption, because the majority of pixels in most astronomical images are a part of the background.

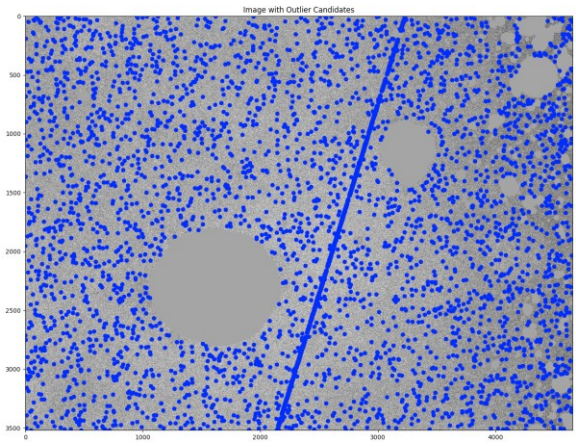


Figure 2(c). The image from Figure 2(a) with the potential outliers discovered in the first stage of outlier detection. Note that the entire satellite trail is marked as an outlier.

3.1.2 Outlier Removal

There are often hot pixels in the image which can outshine the satellite trail in the same manner as bright star fields. Thus, it is important to remove these outlier pixels as well as the stars. The outlier removal routine used in this program has two stages of outlier detection. (Note that the stars within the image have been removed prior to outlier identification and removal.)

The first stage of outlier detection finds outlier candidates by determining the number of standard deviations each pixel is from the mean pixel brightness. If this brightness difference is more than 4 standard deviations, the pixel is marked as an outlier candidate. Figure 2(c) shows the flagged pixels during the first stage of outlier detection.

An issue with this approach to outlier removal is that the satellite trail may also be very bright compared to the mean pixel brightness. Accordingly, the satellite trail is often marked as an outlier and removed if only the first outlier detection stage is employed. To remedy this, the outlier candidates found in the first stage are fed to a second stage.

The second stage of outlier detection determines which of the outlier candidates are truly outliers. It does so by comparing each candidate to only those pixels in the immediate vicinity of the candidate as opposed to all pixels. For each outlier candidate, the second stage selects a 5 by 5 grid of pixels (including the candidate at the center) and computes the average pixel brightness of this grid, as well as the grid's standard deviation. If the candidate's brightness is more than 3 standard deviations from the grid's mean pixel brightness, the candidate is determined to be a true outlier. The threshold for the second stage is lower than that of the previous stage because normal

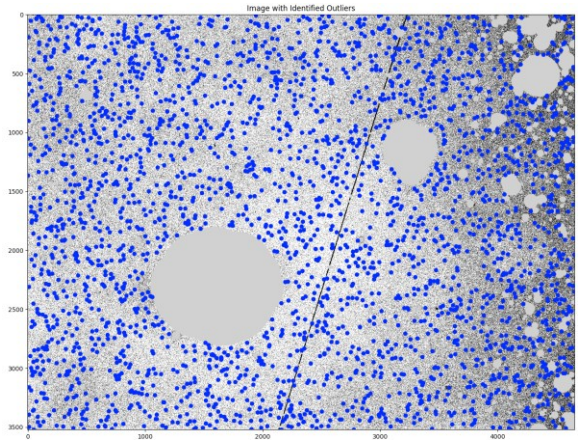


Figure 2(d). The image from Figure 2(a) with the confirmed outliers shown in blue. Note that the second stage of outlier detection no longer recognizes the trail as an outlier.

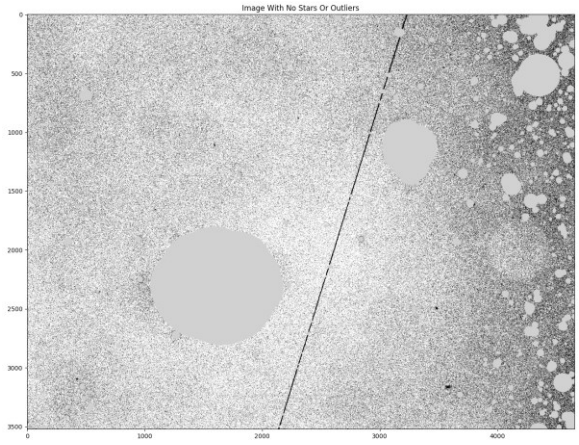


Figure 3: The image from Figure 2(a) with stars and outliers removed. Note that the satellite trail clearly stands out as the most pronounced feature of the image.

pixels should be close in brightness to their surrounding pixels, but not necessarily close to the mean brightness of the entire image. For example, a pixel in a satellite trail may be much brighter than the mean brightness of the image but not significantly brighter than neighboring pixels along the trail. On the other hand, if the same pixel is much brighter than its neighboring pixels, then it is likely an outlier. Figure 2(d) shows a sample image with outliers confirmed after the first and second outlier detection stages.

Finally, for outlier removal, the true outliers are set to the median nearby pixel brightness, in the same manner as the stars. Figure 3 shows the image with both stars and outliers removed.

3.2 Detection of Satellite Trails

3.2.1 Trail Detection Algorithm

In the second segment of the program, once the stars and outliers from the image have been removed, the image with no stars or outliers is fed into a trail detection algorithm, which detects the satellite trail in the image. This algorithm takes advantage of a unique characteristic of satellite trails: they typically traverse a large portion of an image. This property of satellite trails means that although a satellite trail may not be as bright as other objects in the image field, the mean brightness along the path of the satellite trail is likely to be greater than the mean brightness along any other line through the field, once stars and outlier pixels have been removed. This observation is the basis for the following three step trail detection algorithm:

1. Iterate through each possible line through the image by selecting an "entrance point" and an "exit point" for the line. Possible entrance and exit points can be specified as either each edge pixel or as evenly spaced edge points which fall between pixels. Each combination of entrance point and exit point is fed into step 2.

2. Compute the mean and median brightness for each line corresponding to a combination of entrance and exit points. If the slope of the line is less than or equal to 1, the program takes pixel-size steps in the x direction along the line. If the slope is greater than 1, the program takes pixel-size steps in the y direction along the line. The program then averages the brightnesses of pixel steps along the line to find the mean and median line brightness. Finally, the program sums these two values.

3. The program selects the line with the maximum sum of mean and median brightnesses as the satellite trail.

The median brightness term is an extra precaution for hot pixels and bright star fields that are by chance not removed. A line through two stars or hot pixels would have a very low median brightness, making median brightness a good differentiator between true satellite trails and combinations of bright areas.

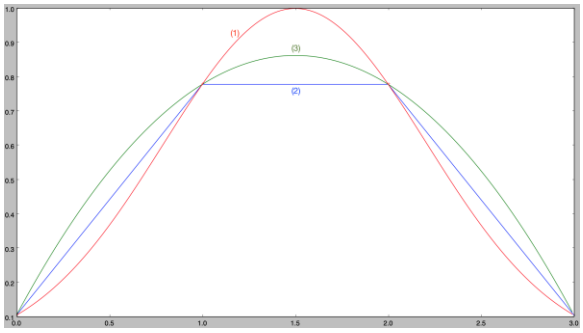


Figure 4. A Gaussian distribution (1) modeled through linear interpolation (2) and cubic interpolation (3). Cubic interpolation better approximates the shape and maximum location of the Gaussian distribution.

3.2.2 Interpolation

When taking pixel size steps along a line, one often does not arrive at a pixel, but instead at a point between pixels. Linear interpolation may be employed between pixels. However, the Gaussian distribution of a satellite trail is not accurately approximated through linear interpolation. In particular, if the peak of a Gaussian is between two pixels, linear interpolation will suggest that the peak is at one of the two pixels (see Figure 4). This can lead the identified line to be biased toward lattice (pixel) points. To fix this issue, the program more accurately determines the brightness of non-lattice points using cubic interpolation (Figure 4).

Cubic interpolation is performed as follows. The program contains a PiecewiseCubic class to hold all interpolation. This class has an __init__ function that iterates through all possible 4x4 squares of pixels. Each pixel is assigned coordinates (x, y) with $0 \leq x \leq 3$ and $0 \leq y \leq 3$. The pixels in each square are used to find the bivariate polynomial,

$$p(x,y) = c_1 + c_2x + c_3y + c_4x^2 + c_5y^2 + c_6xy + c_7x^3 + c_8y^3 + c_9xy^2 + c_{10}x^2y + c_{11}x^2y^2 + c_{12}x^3y + c_{13}xy^3 + c_{14}x^3y^2 + c_{15}x^2y^3 + c_{16}x^3y^3, \quad (2)$$

that exactly fits the brightness of the pixels in the square. This polynomial is found using the numpy.linalg.solve function. The __init__ function ends by storing all of these polynomials in a 2D list.

After the program creates a PiecewiseCubic, it can call that object's call function with points it wishes to interpolate. The __call__ function first determines which polynomial to employ. It selects the polynomial corresponding to the 4x4 square of pixels that is most nearly centered on the desired point. Finally, the desired point coordinates are substituted into the polynomial and the result is returned.

3.2.3 Recursive Scaling

The satellite trail detection algorithm described above is time consuming when applied to images of more than 1000000 pixels. Consider a 1000 by 1000 pixel image, which is still much smaller than most images. For such an image, $6 \cdot 1000^2 = 6 \cdot 10^6$ lines are examined. Assuming that on average the program examines 500 points per line, in total the brightness of $500 \cdot 6 \cdot 10^6 = 3 \cdot 10^9$ points are computed. Each point requires evaluating a 2-dimensional cubic. Examining all lines can take several hours on a standard processor.

To improve speed, the program employs recursive scaling. If an image includes more than 90000 pixels, the image is scaled down by a factor of 4 on each side. If the resultant image is still too large, the program scales down the original image by a factor of 16 on each side. The program continues in this manner, scaling down sequentially by powers of 4, until the resultant image is less than 90000 pixels.

Then, the program finds the satellite trail in the reduced-size image using the satellite trail detection algorithm described above. Next, the program scales the determined endpoints so that they correspond to the second smallest image. The program then checks all lines near the scaled-up trail from the previous image. The mean brightness of all possible lines are compared against each other (the median brightness is no longer needed to ensure that the trail is found). This rescaling process is repeated until the trail is identified in the original image (with stars removed).

Sometimes, a bright star or hot pixel may be located near the corner of an image. In such cases, a line passing through the corner may appear to be a satellite trail, as that star or hot pixel creates a high mean brightness for a line passing through it. To combat this, the program does not allow lines which are less than or equal to 40 pixel steps in length.

3.3 Fitting of Satellite Trails

In the third segment of the program, once a satellite trail has been detected in the starless image, the program returns to the original image with stars and determines several properties of the trail. As discussed in Section 2, the brightness of a satellite trail is normally distributed as a function of the perpendicular distance to the trail, with a standard deviation that does not change significantly along the length of the trail. The brightness as a function of

distance along the trail can vary and may be modeled by a polynomial $b(d_\parallel)$.

Thus, the photon count, $c(d_\parallel, d_\perp)$, at a point can be expressed as

$$c(d_\parallel, d_\perp) = b(d_\parallel)e^{-d_\perp^2/(2\sigma^2)}, \quad (3)$$

where $d_\parallel$ is the distance parallel to the trail, $d_\perp$ is the distance perpendicular to the trail, and σ is the standard deviation of the normal distribution. A satellite trail can also wobble around a central line of the trail. This wobble can be fit by a function f computed using a Fast Fourier Transform, and can be factored into the transformation from xy coordinates to $d_\parallel$ and $d_\perp$ (as described further in section 3.4.1).

To fit a Gaussian to the detected satellite trail, three key modules and four basic steps are employed. The modules are:

1. A curvilinear transformation module for trail wobble and slant.

2. A Fast Fourier Transform module to compute a fit for the trail wobble as a sum of sines and cosines.

3. A module with a set of functions to ignore pixels near identified stars and to interpolate the desired values of data near these ignored pixels.

Additionally, the piecewise cubic discussed above is used extensively in the third program segment.

The four basic steps are:

1. Compute and fit a function to the trail wobble;

2. Compute a more accurate value for the image background near the trail;

3. Compute and fit a polynomial to the trail brightness; and

4. Compute the standard deviation of the trail.

Each of these modules and steps is explored in greater detail in the following sections.

3.4 Key Modules

3.4.1 Curvilinear Transformations

A satellite trail can pass through an image in any orientation and can wobble around a central line of the trail. This means the light from a satellite trail must be approximated by a complex Gaussian-type distribution. To simplify this fit, the program first determines the orientation (see Section 3.2) and wobble (see Section 3.5.1) of the trail and then applies a curvilinear coordinate transformation to convert (x, y) pixel coordinates to the coordinates $d_\parallel$ and $d_\perp$. These new coordinates represent the distances from the y-intercept of the satellite trail to a point, in directions parallel to and perpendicular to the satellite trail, respectively. This coordinate transformation advantageously allows the path of the satellite trail to be described by the simple formula $d_\perp$ = 0, making subsequent Gaussian fitting significantly easier.

First assume that the satellite trail has no wobble and can be approximated by a straight line passing through the image. Given a line l of the form y = mx + b, the transformation to determine $d_\parallel$ and $d_\perp$ is

$$d_\parallel = \frac{x + my - mb}{\sqrt{1 + m^2}} \quad (4)$$

$$d_\perp = \frac{y - mx - b}{\sqrt{1 + m^2}}. \quad (5)$$

However, if the line is vertical, m = ∞, which means that the program cannot use these equations for vertical lines (without taking limits). If a vertical line is given by $x = x_0$, the coordinates are given by

$$d_\parallel = y \quad (6)$$

$$d_\perp = x - x_0. \quad (7)$$

Now, suppose that the satellite trail wobbles around a central line. Instead of having the satellite trail's location given by the equation $d_\perp$ = 0 as it would if the trail were a straight line, the trail is now determined by $d_\perp$ = f($d_\parallel$), for some function f that represents the trail wobble. This however defeats the purpose of having the satellite trail be along $d_\perp$ = 0. To remedy this, the program simply subtracts f($d_\parallel$) from the expression for $d_\perp$. For a non-vertical line,

$$d_\parallel = \frac{x + my - mb}{\sqrt{1 + m^2}} \quad (8)$$

$$d_\perp = \frac{y - mx - b}{\sqrt{1 + m^2}} - f(d_\parallel) \quad (9)$$
$$= \frac{y - mx - b}{\sqrt{1 + m^2}} - f\left(\frac{x + my - mb}{\sqrt{1 + m^2}}\right).$$

and for a vertical line,

$$d_{\parallel} = y \tag{10}$$

$$\begin{aligned} d_{\perp} &= x - x_0 - f(d_{\parallel}) \\ &= x - x_0 - f(y) \end{aligned} \tag{11}$$

This new curvilinear transformation satisfies the desired condition that the curve $d_{\perp} = 0$ represents the path of the trail.

The inverse transformation, which converts $d_{\parallel}$ and $d_{\perp}$ to x and y is given as follows:

$$x = \frac{d_{\parallel} - m\left(d_{\perp} + f(d_{\parallel})\right)}{\sqrt{1+m^2}} \tag{12}$$

$$y = \frac{m d_{\parallel} + d_{\perp} + f(d_{\parallel})}{\sqrt{1+m^2}} + b. \tag{13}$$

These transforms are stored in a module known as GenerateTrail.py.

3.4.2 Fourier Transform Curve Fitting

In the process of determining a 2D Gaussian fit for the trail, particularly while fitting wobble and trail brightness, the program must fit a function to data of an unknown form. Further, the trail wobble oscillates rapidly and unpredictably, making it unsuited for a polynomial fit. At the same time, because of the noise inherent in images, a standard Fourier transform, which fits data exactly with no smoothing, also is not appropriate for the task. As a result, the program employs a variation on a Fourier transform which strategically discards frequencies in order to smooth the fit without losing valuable data.

The program includes a modified Fourier transform module that first performs a Fast Fourier Transform (FFT) using NumPy's FFT module. Then, using the returned frequencies and amplitudes, the modified FFT module synthesizes a function which fits the given data exactly. Next, to remove negligible terms, the modified FFT module removes all terms involving a sine or cosine with amplitude less than a predetermined value (e.g., 0.0000001).

To smooth the exact fit, the modified FFT module weights each term, with amplitude A and frequency f, according to the equation

$$w = \frac{A}{f + 0.00001}. \tag{14}$$

The modified FFT module then removes a user selected number of terms with the smallest weights. The weighting balances two factors. First, terms with small amplitude are often either unimportant for the overall fit or only relevant when combined with a number of other small amplitude terms to create abrupt changes in the data. Neither of these contributions is beneficial to a smooth fit. Second, terms with high frequency are often used to exactly fit noise and so are undesirable for a smooth fit. The 0.00001 term in the denominator is simply to avoid division by zero errors which occur with the constant term in the synthesized fit, which is represented as a cosine function with frequency 0.

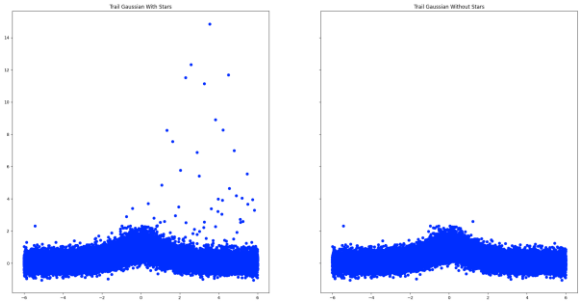
3.4.3 Functions for Ignoring Stars

During the several fitting procedures used to create a Gaussian brightness profile for the satellite trail, stars or hot pixels can interfere with the brightness fits because they often outshine the satellite trail. To remedy this, the program uses a function which detects which points are influenced by nearby stars or hot pixels.

This function uses the list of stars in the image found employing the Gaia catalogue. For each star and each point in question, the function determines if the point lies within a user specified minDist from the edge of the star. Similarly, the program determines which points lie within minDist of any detected hot pixels. If a point is too close to a star or a hot pixel, the function marks it false in a separate array. Further, because some of the trail fitting functions examine points outside of the image frame, this function also marks any points outside of the image frame as false as well. The function then returns the list of which points are affected by stars or hot pixels, or are outside the image frame.

For some purposes, such as fitting a Gaussian, affected pixels can simply be removed. However, a Fast Fourier Transform requires that its input be evenly spaced data. To remove the effects of stars or hot pixels while keeping data evenly spaced, the program uses a second function, interpolateSignal, to linearly interpolate between the unaffected values on both sides of a region with a star or hot pixel.

Figures 5a and 5b show the significant noise reduction that occurs as a result of detecting and ignoring points with stars or hot pixels.



Figures 5(a) (left) and 5(b) (right). Side-by-side comparisons of trail values before (left) and after (right) points near stars and hot pixels are ignored.

3.5 Trail Fitting

3.5.1 Fitting Trail Wobble

The code to fit the trail wobble examines equally spaced steps along the identified satellite trail. The number of steps is given by the number of pixels in the larger edge of the image. At each step, the code uses an algorithm, discussed below, to determine at what distance from the central line the center of the satellite trail lies. Once this distance has been determined, the code invokes the previously described function (Section 3.4.3) to remove data points influenced by stars and hot pixels and to interpolate between points. Finally, the modified FFT module (see Section 3.4.2) determines a fit for the wobble function $f(d_\parallel)$ of the trail.

The method for determining the location of a trail in each step is the essential part of the function to fit trail wobble. Two approaches were explored. In a first approach, the method examines points on the perpendicular to the trail. A discussed above, the program takes equally spaced steps along the trail. At each step, given by equally spaced values of $d_\parallel$, this method examines points with varying $d_\perp$. In particular, it examines the points given by

$$d_\perp = d_{step} i - d_{bound} \qquad 0 \le i \le \frac{2d_{bound}}{d_{step}}, \tag{15}$$

or, in other words, equally spaced points between $-d_{bound}$ and d_{bound}, separated by a distance d_{step}. The user may pick d_{bound} and d_{step}, but their default values are 5 and 0.1, respectively. The code then selects the brightest point as the location of the trail, and uses $(d_\parallel, d_\perp)$ as a data point for the curve fitting.

While this method works, the inherent noise in images makes the estimated position of the trail at each step noisy. As a result, the approach relies heavily on the modified FFT module's ability to smooth the noise appropriately.

A second approach for determining the location of a trail at each step appears more robust. Instead of taking the brightest pixel, the code first computes a Gaussian cross-correlation integral and then picks the point with the highest cross-correlation. This approach examines the same points as the previous approach. However, it examines them differently. It first computes a Gaussian with peak 1 and standard deviation set by the user (default set to 2),

$$g(x) = e^{-x^2/2\sigma^2}, \tag{16}$$

for x values given by

$$x = d_{step} j - x_{range} \qquad 0 \le j \le \frac{2x_{range}}{d_{step}}, \tag{17}$$

where x_{range}, chosen by the user (default set to 2), marks the range of x values at which the Gaussian is evaluated. Next, for each $d_\perp$, the program evaluates

$$corr(d_\perp) = \sum_{j=0}^{\frac{2x_{range}}{d_{step}}} [b(d_\parallel, d_\perp - x_{range} + d_{step} j) \cdot g(d_{step} j - x_{range})]. \tag{18}$$

This numerically approximates the cross-correlation integral of the trail brightness and $g(x)$. Finally, for each $d_\parallel$, the program selects the $d_\perp$ with the highest cross-correlation.

Figure 6 shows the computed wobble and wobble fit for a sample image.

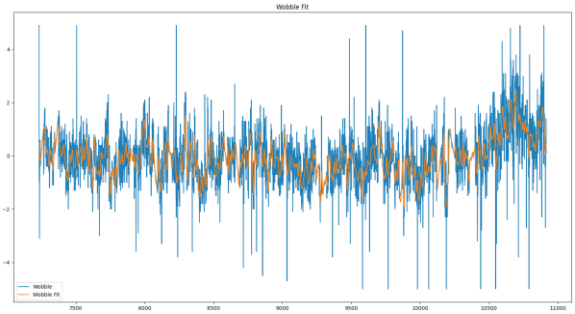


Figure 6. The determined wobble for the image in Figure 2a with its corresponding fit (shown in orange) computed using Fourier synthesis.

3.5.2 Background Approximation

The background approximation employed in previous steps is too inaccurate to be used to determine trail brightness. As a result, after fitting trail wobble, the program more accurately fits the image background. To do so, the program performs a curvilinear transformation on all pixels in the image, and then chooses the points with $10 \leq d_{\perp} \leq 20$. Next, the program removes the pixels which are too close to stars and hot pixels. The program then averages the brightness of all of these pixels to determine a more accurate value for the image background.

Figure 7 shows the computed background pixels and mean background for a sample image.

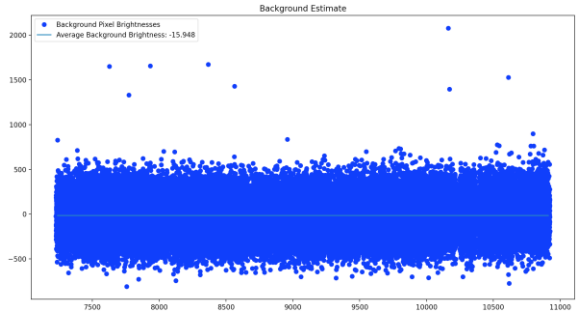


Figure 7: The determined background pixel brightnesses and overall background estimation (light blue line) for the image in Figure 2(a).

3.5.3 Brightness Fitting

Next, the program determines a function that approximates the peak brightness of the trail. To do so, it performs an inverse transformation on the curve defined by $d_{\perp} = 0$ to determine the points directly on the satellite trail. It then takes as many equally spaced steps along the trail as there are pixels in the larger edge of the image. Then, at each step, the program measures the brightness and stores the brightness and $d_{\parallel}$ as a data point. Finally, the program identifies and interpolates over points that are affected by stars or hot pixels, and fits a fifth degree polynomial, $b(d_{\parallel})$, to the brightness.

Figure 8 shows the computed trail brightness and brightness fit for a sample image.

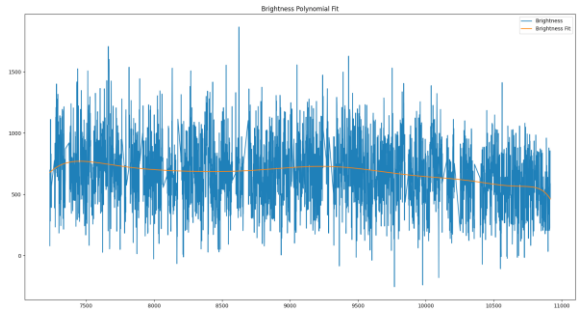


Figure 8. The determined trail brightness for the image in Figure 2(a) with its corresponding fit (shown in orange).

Another alternative being explored to reduce noise is use of a Gaussian cross-correlation approach similar to that for fitting the trail wobble.

3.5.4 Standard Deviation Fitting

After brightness fitting, the program determines the standard deviation of the satellite trail. To do so, the program performs a curvilinear transformation on all pixels in the image, and then chooses only the points with $d_{\perp} \leq 5$. Next, the program removes the pixels which are too close to stars and hot pixels. Then, the program divides each pixel brightness by $b(d_{\parallel})$, the fitted polynomial brightness calculated in Section 3.5.3. Finally, the program fits a Gaussian in $d_{\perp}$ with peak height of 1 to the pixels and determines the fitted value of σ.

Figure 9 shows the determined trail pixels and Gaussian fit for a sample image.

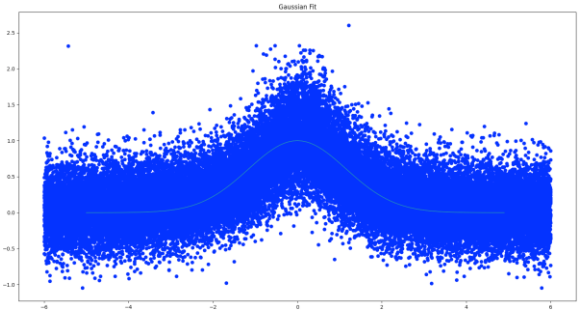


Figure 9: The Gaussian fit (shown in light blue) for the satellite trail from the image in Figure 2(a).

3.6 Removing Trail

Finally, in the fourth segment of the program, after fitting the trail wobble $f(d_{\parallel})$, the trail brightness $b(d_{\parallel})$, and the trail standard deviation σ, the program coordinate transforms each pixel in the image using wobble function $f(d_{\parallel})$. The program then computes

the brightness of the trail at each pixel according to the formula

$$c(d_{\parallel}, d_{\perp}) = b(d_{\parallel})e^{-\frac{d_{\perp}^2}{2\sigma^2}}. \quad (19)$$

Finally, the code subtracts the calculated brightness pixelwise from the image.

4 Results

4.1 Astrophotography Results

Figures 10(a) – 10(d) demonstrate a comparison of images before and after satellite-trail removal using the program described in this paper. Visually, the code is effective at removing signs of satellite trails in images.

Figure 10(a). A first image with a satellite trail. (Courtesy of Ryan Caputo).

Figure 10(b). The image of Figure 10(a) with the satellite trail removed using the program described in this paper.

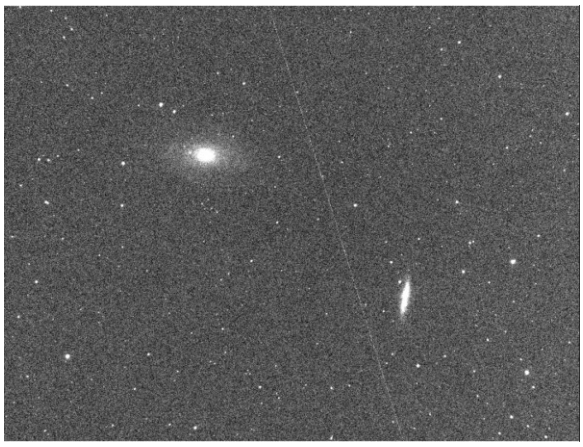


Figure 10(c). A second image with a satellite trail. (Courtesy of Ryan Caputo).

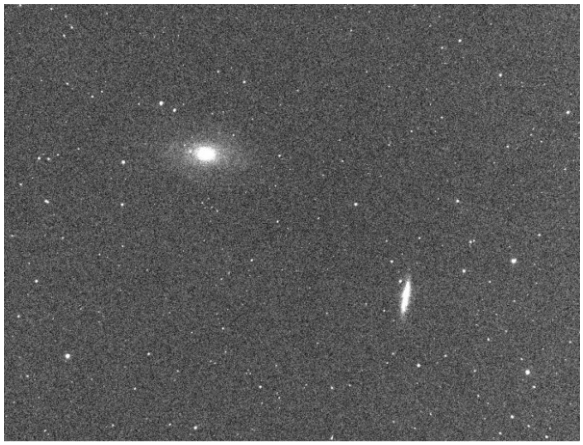


Figure 10(d). The image of Figure 10(c) with the satellite trail removed using the program described in this paper.

4.2 Photometry Results

Whether or not the program preserves image photometry is a key indicator of the program's effectiveness for scientific purposes. To test photometry, two images of the same region of the sky were examined: one image with a satellite trail, and one image without a trail (taken before the satellite traversed the star field). The satellite trail removal program described above was applied to the image with the satellite trail to generate a third image with the satellite trail removed. Next, the photometry of stars near the satellite trail in all three images was determined and compared using aperture photometry.

Although the current version of the code is still in development, the previous version, which did not fit trail wobble, proved very effective at improving image photometry (see Figure 11). Accounting for wobble should further improve the result below.

To compare the three images discussed above, the percent photometry error between the image with

the trail and the image without the trail was determined for nine stars. Similarly, the percent photometry error between the image with the trail removed and the image without the trail was determined. Figure 11 shows a comparison between the two photometry errors.

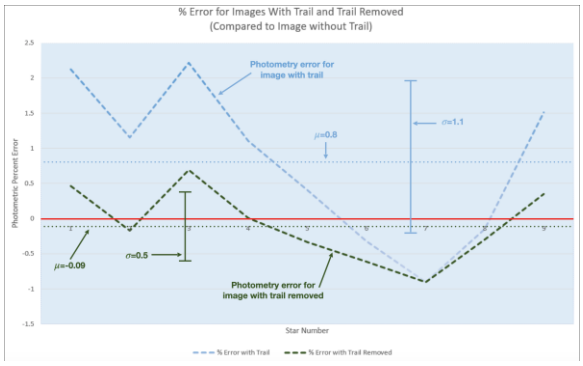


Figure 11: The percent photometry error for both an image with a satellite trail and the same image with the satellite trail removed. Note that the mean percent error is 9 times less for the image with the trail removed than for the image with the trail, and the standard deviation of the percent error is 2 times smaller for the image with the trail removed than for the image with the trail.

As shown in Figure 11, the mean percent error is 9 times less for the image with the trail removed than for the image with the trail. This demonstrates that the program significantly reduces photometric error caused by satellite trails. Similarly, the standard deviation of the percent error is 2 times smaller for the image with the trail removed than for the image with the trail. Although less direct, this demonstrates greater photometric precision for the image with the trail removed.

A more detailed test of the code's functionality is to determine the error in the photometry with no trail and whether the photometry of the image with the trail removed is within that expected error. This typically requires multiple images without a satellite trail to determine the intrinsic photometric error, and these images were not available at the time of testing.

5. Further Research

There are several avenues for further research which focus on improving different parts of the program:

• The program to fit the trail standard deviation and the program to remove the satellite trail could be modified to use an integrated Gaussian, instead of a standard Gaussian. This is a more realistic fit, because the satellite trail is Gaussian at the sub-pixel level. The use of integrated Gaussians is discussed in the documentation for Bradley (2019).

• The code to remove stars and outlier pixels may be upgraded to address galaxies. While removing circular portions of an image works well for stars, it is less optimal for galaxies. Galaxies are rarely face-on and circular; instead they appear elliptical. Adding functionality to the star removal code which specifically searches for known galaxies in the image and removes them using ellipses, could further improve the ability of the program to find satellite trails.

• Code to more carefully remove the effects of very large galaxies and nebulae may be implemented. Removing galaxies and nebulae by replacing the relevant pixels by the median brightness of the image is extremely effective for small deep-sky objects. However, if galaxies or nebulae fill most of an image, replacing these objects by the median brightness could remove most of the satellite trail. An algorithm which addresses such images by determining a brightness profile for each galaxy or nebula using either deep learning or a polynomial fit may greatly improve the code's functionality for these types of images.

• Functionality may be added which addresses satellite trails that do not pass fully through the image.

6. Conclusion

Algorithms have been implemented in Python code that automatically detect and remove satellite trails from night-sky images. In processed images, evidence of satellite trails is difficult to detect visually, and photometric accuracy is significantly improved. Notably, only a single image is required for satellite trail removal.

The program first preprocesses a given image by removing stars and hot pixels. Second, the program determines the location of the satellite trail in the image by examining each line in the image and by using recursive scaling to improve processing speed. Third, the program fits the satellite trail wobble, brightness, and standard deviation using Gaussian cross-correlations, curvilinear transformations, and Fourier Transforms. Fourth, the program computes the values of the fitted trail Gaussian for all pixels and subtracts these values from the image pixels.

Key aspects of the program include:

- Removing stars and hot pixels from the image to greatly improve satellite detectability.
- Testing all possible lines through the image for satellite trails.
- Using a piecewise bivariate cubic fit for interpolation instead of linear interpolation for more accurate Gaussian approximation.
- Recursively scaling images to greatly reduce processing time.
- Fitting satellite trail wobble using Gaussian cross-correlation to reduce fit noise.
- Using a modified Fast Fourier Transform for improved fits.
- Using curvilinear transformations to flatten the satellite trail.
- Ignoring points influenced by stars or hot pixels to improve trail fits.

Although the current version of the code, which adds satellite trail wobble fitting, is still in progress, a previous version has been shown to significantly improve image photometry. Accounting for satellite trail wobble should further enhance photometric results.

7. Acknowledgements

I gratefully acknowledge Kalee Tock for identifying the need for this research and encouraging me to work on it, and Ryan Caputo for generously providing images described in this paper.

I also gratefully acknowledge Dr. Jeffrey Hall, director of Lowell Observatory, for his helpful suggestions on photometry testing.

This research made use of Photutils, an Astropy package for detection and photometry of astronomical sources (Bradley et al. 2020).

8. References

Bradley, Larry, et. al. "astropy/photutils: v0.6 (Version v0.6)." (2019, January 7). *Zenodo*. http://doi.org/10.5281/zenodo.2533376.

Bowler, Jacinta. "That Starlink Problem Astronomers Were Worried About Is Totally Happening." (2019, Nov 20). *Science Alert*. www.sciencealert.com /starlink-is-being-an-absolute-nuisance-to-astronomers.

Hall, Shannon. "As SpaceX Launches 60 Starlink Satellites, Scientists See Threat to 'Astronomy Itself'." (2019, Nov. 11). *The New York Times*. www.nytimes.com/2019/11/11/science/spacex-starlink-satellites.html.

Hunter, John D. "Matplotlib: A 2D Graphics Environment." (2007). *Computing in Science & Engineering* 9, 90-95.

Lang, Dustin, et. al. Astrometry.net: "Blind Astrometric Calibration of Arbitrary Astronomical Images." (2010). *The Astronomical Journal*, 139(5): 1782–1800.

Price-Whelan, A. M., et. al. "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package." (2018). *The Astronomical Journal*.

Robitaille, Thomas, et al. "Astropy: A Community Python Package for Astronomy." (2013). *Astronomy & Astrophysics* 558.

Siegel, Ethan. "This Is How Elon Musk Can Fix The Damage His Starlink Satellites Are Causing To Astronomy." (2019, Nov. 20). *Forbes*. www.forbes.com/sites/startswithabang/2019/11/20/this-is-how-elon-musk-can-fix-the-damage-his-starlink-satellites-are-causing-to-astronomy/#842d0184ccce.

"US astronomers speak on SpaceX Starlink satellites." (2019, June 12). *EarthSky*. https://earthsky.org/human-world/aas-statement-spacex-starlink-satellites.

Van der Walt, Stéfan. "The NumPy Array: A Structure for Efficient Numerical Computation." (2011). *Computing in Science & Engineering*, 13, 22-30.

Virtanen, Pauli. "Scipy 1.0 – Fundamental Algorithms for Scientific Computing in Python," (2020). *Nature Methods* 17, 261-272.