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

Owen Dugan

January 13, 2020

Abstract

SpaceX's Starlink satellite network promises world-wide high-speed internet access. With up to 42,000 satellites to be deployed [1], however, the Starlink satellite network will significantly degrade ground-based astronomical research and imaging due to trails (e.g., light reflections or emissions) from passing satellites. Removing satellite trail effects on night sky images is difficult 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. No known solution previously existed. Novel algorithms were developed and coded to accurately iden-tify and remove satellite trails and their effects on photometry. An in-ventive approach was developed that implements platesolving to identify stars within an image, and an algorithm to determine the radius of each star identified. Identified star brightnesses are replaced with median im-age brightness values. Satellite trails are identified by examining each possible line traversing the image, with recursive sizing, using area inter-polation, implemented for large images to reduce processing time. Area and/or cubic-spline interpolation is employed to optimize satellite trail modeling to within a tenth of a pixel. A Gaussian brightness profile is developed for the satellite trail to account for satellite trail effects across the entire image. The code returns original star brightnesses to the image and 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 maintaining image photometric accuracy. Additional novel solutions for preserving star brightness directly under the satellite trails are explored, as is application of deep learning for satellite trail identification. The novel satellite-trail-removal methods not only salvage astrophotography images that otherwise would have been ruined by satel-lite trails but also preserve these images for astronomical research, effec-tively increasing the productivity of astronomical equipment and reducing research costs.

1 Introduction

SpaceX plans to launch up to 42,000 satellites into orbit in support of Starlink, its high speed internet network which will provide high speed internet to any location on Earth[1]. Astronomers are deeply concerned about the impact of these satellites on astronomical observations and research [2]. Indeed, some have predicted that the Starlink network will be the "end of Astronomy" [3]. 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. [4]

The present research proposes to mitigate the effects of satellite trails by developing and implementing novel algorithms 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 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, it causes some objects in the image to appear artificially brighter than others.

Since satellite trails alter exactly what photometry seeks to measure, these trails can result in incorrect photometry measurements. Generally, if a satellite is observed in an image, the image must be thrown out. This is an extremely inefficient use of images, especially as the number of satellites increases, and is what the program discussed in this paper seeks to address.

3 Algorithm Overview

The algorithm is divided into segments which it performs sequentially. The first segment of the algorithm identifies and removes the stars in the image. It then feeds the resulting image with no stars into the second segment. This segment identifies the path of the satellite through the image. The third segment of the

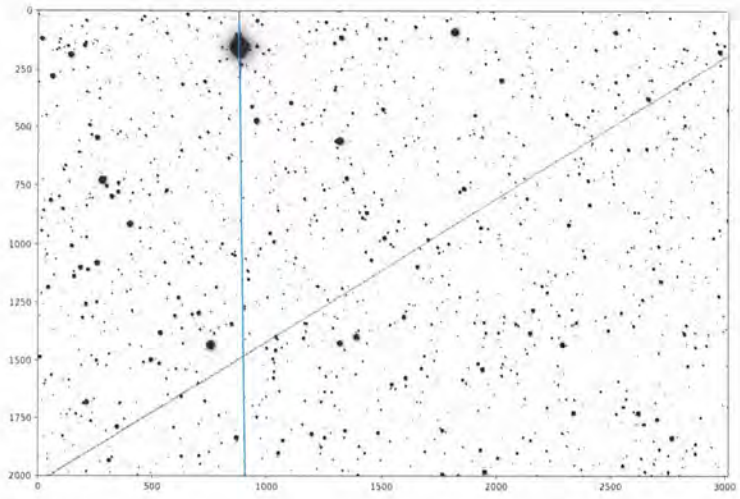

Figure 1: An incorrect fit of the satellite's trail when stars are not removed. The blue line indicates the "trail" the algorithm detected. The line running roughly diagonally through the image is the true satellite. Note the extremely bright star that the detected trail passes through. This star outshines the entire satellite trail, and so the algorithm identifies it as the trail. This image demonstrates the extent to which the star field outshines many satellite trails and how this can lead line detection algorithms astray.

algorithm takes the identified path of the satellite and uses the original image to determine the Gaussian that best fits the brightness of the satellite. Finally, the forth segment removes the trail from the original (with stars) image based on the identified path and brightness.

These four segments are described in more detail below:

3.1 Removing Stars and Outliers from the Image

3.1.1 Star Removal

The first segment of the algorithm locates and removes the stars in the image. This step is critical to the algorithm's success. Satellite trails tend to 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 along a line passing through several bright stars can often exceed the signal along the satellite trail. This makes it extremely difficult to detect the satellite instead of a line of stars, as shown in Figure 1.

The algorithm presented in this paper takes a novel approach to detecting satellite trails. Instead of attempting to detect the trail from behind the star field, the algorithm removes the star field and then detects the trail.

The star removal algorithm first identifies the stars in an image and second

iterates through each identified star and removes it. The code offers users three options for identifying the stars in an image:

1. The algorithm can plate-solve the image by sending it to the `Astrometry.net` server [5] and receiving a file from `Astrometry.net` containing the locations of all of the identified stars in the images.

2. The algorithm can plate-solve the image by running the `Astrometry.net` package locally. This is much faster than the previous approach, but it also requires downloading both the `Astrometry.net` package and all of the auxiliary packages needed for `Astrometry.net` to run.

3. The algorithm can take an already plate-solved image. In this case, the algorithm uses the `Astroquery` package to query the Gaia star catalogue for the locations of stars in the images.

The algorithm then iterates through each recognized star. It first determines the radius of the 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 8 equally spaced points a distance of r from the center of the identified star. These 8 points are defined by

$$(x_k, y_k) = \left(x_c + r\cos\left(\frac{k\pi}{4}\right), y_c + r\sin\left(\frac{k\pi}{4}\right)\right), \quad k = 0, 1, 2, \ldots, 7, \qquad (1)$$

where (x_c, y_c) is the center of the star. The algorithm finds the closest lattice point to each (x_k, y_k) and calculates the mean brightness of the pixels corresponding to these lattice points[1]. If this mean 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 mean brightness of the 8 points is less than or equal to the median brightness of the entire image, the algorithm determines that this must mark the edge of the star, and selects this value of r as the correct radius[2].

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 the entire image[3].

Note that in this algorithm (and in the algorithms discussed later), the median brightness of the image is used as an estimate of the background brightness of the image. This is a valid assumption, because the majority of pixels in most astronomical images are a part of the background.

[1] If any of the lattice points lies outside of the image, that point is not included in the average.

[2] According to the reasoning above, the star should have a radius that is one less than the selected radius. However (as discussed below) for this application, it is better to remove larger segments of an image than smaller ones.

[3] Again, if any lattice points that are less than $r + 1$ from (x_c, y_c) fall outside of the image, these points are discarded.

3.1.2 Outlier Removal

There are often hot pixels in the image which can outshine the satellite trail in the same way 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.

The first stage of the outlier removal process 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.

The issue with this approach to outlier removal is that the satellite trail is also very bright compared to the mean pixel brightness. This means that the satellite trail is often marked as an outlier and removed. This, of course, is not satisfactory, so the outlier candidates found in the first stage are then fed to a second stage.

The second stage 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 candidate, the second stage selects a 5 by 5 grid of pixels (with 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 is more than 3 standard deviations from the grid's mean, 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 pixels should be close to their surrounding pixels, but not necessarily close to the mean brightness of the entire image. For example, a pixel in a trail is likely to be much brighter than the mean brightness of the image without being abnormal. On the other hand, the same pixel can be only slightly different from its neighboring pixels before it is an outlier. The second stage of outlier detection uses a lower threshold to ensure that pixels drastically different from their neighbors are labeled as outliers. Finally, the true outliers are set to the median pixel brightness just like the stars.

3.2 Detection of Satellite Trails

3.2.1 Base Algorithm

Once the stars and outliers from the image have been removed, the image with no stars is fed into an 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 they may not be as bright as a few other objects in the 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. This observation is the basis for the following algorithm:

1. The algorithm iterates through each possible line through the image. It does so 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 and exit point is fed into step 2.

2. For each line corresponding to a combination of entrance and exit point, the mean and median brightness is computed. If the slope of the line is less than 1, one pixel steps in the x direction are taken along the line. If the slope is greater than 1, one pixel steps in the y direction are taken along the line. The brightnesses at each one pixel step are averaged to find the mean and median brightness along the line. These two values are added.

3. The line with the maximum sum of mean and median brightness is the one selected.

The median brightness term is an extra precaution for hot pixels and bright star fields that are by chance not removed. This is because 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.

3.2.2 Interpolation

When taking pixel steps along the line, one often does not arrive at a pixel, but instead at a point between pixels. At first, I linearly interpolated between pixels. However, the gaussian distribution of the satellite trail is not accurately approximated through linear interpolation. In particular, if the peak of the gaussian is between two pixels, linear interpolation will suggest that the peak is at one of the two pixels (see Figure 2). This can lead the identified line to be biased toward lattice points. This difference of half a pixel can greatly increase the error of the subsequent gaussian fit. To fix this issue the brightness of non-lattice points is determined using cubic interpolation instead. As shown in Figure 2, cubic interpolation much more accurately takes into account that a maximum may be in between two pixels.

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

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

that best fits the data. This polynomial is found using the `numpy.linalg.solve` function. The `__init__` function ends by storing all of these functions in a 2D

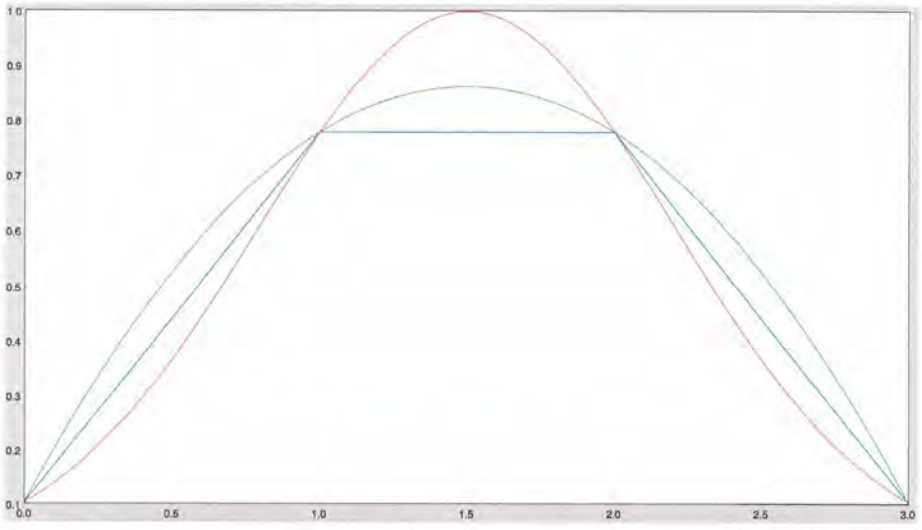

Figure 2: A gaussian (red) modeled through linear interpolation (blue) and cubic interpolation (green). Although neither approximation is perfect, the cubic interpolation does a much better job of approximating the shape of the gaussian. Although it still is not perfect, it is accurate in where the maximum occurs, which is most important for determining the location of the trail.

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 cubic function to interpolate using. It selects the function corresponding to the square of pixels that is most nearly centered on the desired point. It then determines the desired pixel's coordinates in the coordinate system of the square of pixels it corresponds to. Finally, substitutes these coordinates into the corresponding function and returns the result.

3.2.3 Recursive Scaling

Unfortunately, this algorithm is extremely slow when applied to images of more than 1000000 pixels. To see why, consider a 1000 by 1000 pixel image. This image is still much smaller than most images. For such an image, $6 \cdot 1000^2 = 6 \cdot 10^6$ lines must be examined. If we assume that on average 500 points are examined per line to compute the brightness of each line, in total the brightness of $500 \cdot 6 \cdot 10^6 = 3 \cdot 10^9$ points must be computed. Each point requires evaluating a 2-dimensional cubic. This can take several hours to run on a standard processor.

In order to combat this, my program utilizes recursive scaling. If the image is more than 90000 pixels, it scales down the image by a factor of 4 on each side. If the resultant image is still too large, it scales down the original image by 16 on each side. It continues in this way until the resultant image is less than 90000 pixels.

Then, the code finds the line using the algorithm described above in the small image. Next, it scales the endpoints so that they correspond to the second smallest image. It then checks all lines around the scaled-up line from the previous image. To do so, it lets the new lines to be tested cycle through all possible combinations of endpoints near the scaled up endpoints from the previous image. If the scale factor between the sides of the two images is x, these nearby endpoints are all edge pixels within x of the scaled up edge pixels from the previous image. Each possible line is compared against the other by the mean brightness (the median brightness is no longer needed to ensure that the trail is found). This process is repeated until the trail is identified for the final image. The final step, however, examines endpoints that are one tenth of a pixel apart. This allows for greater precision in the final trail which greatly improves the final image quality.

Sometimes, a bright star or pixel may be located near the corner of an image. Then, a line passing through the corner may appear promising, as that star or pixel overpowers the few other pixels on that line. To combat this, the code does not allow lines which are less than or equal to 40 pixel steps across.

3.3 Fitting of Satellite Trails

Once the path of a satellite has been detected in an image, the code must determine 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 along the length of the trail. The brightness as a function of distance along the trail can also vary, and may be modeled by a polynomial p. Thus, the photon count at a point can be expressed as

$$c(d_{\parallel}, d_{\perp}) = p(d_{\parallel})e^{-d_{\perp}^2/(2\sigma^2)}, \tag{2}$$

where $d_{\parallel}$ is the parallel distance along the trail, $d_{\perp}$ is the perpendicular distance to the trail, and σ is the standard deviation of the normal distribution. Thus, the satellite trail parameters that the code must determine are σ and the coefficients of p. The algorithm uses a second degree polynomial for p, which has 3 coefficients, so, in total, there are 4 parameters to determine.

To determine these parameters, the algorithm returns to the image with stars, and it first selects all pixels within a user-specified distance d_{max}(defaulting to 10 pixels) to the trail. It then subtracts the overall median image brightness from these pixels. This causes the background to have a brightness of approximately 0, meaning that the selected pixels can be more adequately fit to a gaussian. Next, the algorithm uses the `scipy.optimize.curve_fit` function [6] to fit $c(d_{\parallel}, d_{\perp})$ to the selected pixels. It imposes no constraints on the coefficients of p, but, it forces $0 \leq \sigma \leq d_{max}$.

Converting x, y coordinates to perpendicular distances and parallel distances is discussed in Section 3.4.

3.4 Removal of Satellite Trails

Given a line l, a standard deviation σ, and a brightness function $p(d_1)$, the code generates a satellite trail. It first determines the parallel and perpendicular components of each point in terms of the point's coordinates. If the line is of the form $y = mx + b$, the parallel component of the distance is

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

and the perpendicular component of the distance is

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

However, if the line is vertical, $m = \infty$, 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 parallel distance is given by

$$d_{\parallel} = y$$

and the perpendicular distance is given by

$$d_{\perp} = x - x_0.$$

These distances are then fed into the function c above to find the brightness of each pixel of a generated satellite trail. Finally, the generated satellite trail is subtracted from the original image to produce the trail-free image. This subtracts the effect of the satellite from each pixel in the original image.

3.5 Results

Figure 3 shows a side-by-side comparison of several images before and after satellite-trail removal.

4 Further Research

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

- The code to remove stars and outlier pixels could be upgraded to deal specifically with galaxies. While removing circular portions of an image works well for stars, which appear roughly circular, it is less optimal for galaxies. This is because galaxies are rarely face-on and circular, instead appearing as elliptical. Adding functionality to the star removal code which would specifically search for known galaxies in the image and remove them using ellipses could improve the removal of everything that is not a satellite trail, and hence could further improve the ability of the program to find the satellite trail.

(a) Image 1 with the satellite trail.

(b) Image 1 with the satellite trail removed.

(c) Image 2 with the satellite trail.

(d) Image 2 with the satellite trail removed.

Figure 3: A side-by-side comparison of images before and after satellite removal. These images were generously provided by Ryan Caputo, a student at my High School.

- Code to more carefully remove the effects of very large galaxies and nebulae could 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 galaxy or nebula fills most of an image, replacing the galaxy by the median could remove most of the trail. If this occurred, it may be difficult for the algorithm to find the satellite trail. More problematic, once the algorithm has found the trail, the background brightness of the nebula or galaxy may result in inaccurate fits of the satellite brightness. An algorithm which deals with such images by determining a brightness profile for the galaxy or nebula using either deep learning or a polynomial fit could greatly improve the code's functionality for these types of images.
- Additional methods for fine-tuning the location of the line could be explored.
- Functionality could be added which can handle satellite trails that do not pass fully through the image. Currently, the brightness profiles used do not accurately fit abrupt changes in brightness and so do not successfully remove trails that do not travel through the entire image.

References

[1] Bowler, Jacinta (2019, Nov. 20). That Starlink Problem Astronomers Were Worried About Is Totally Happening. Retrieved from https://www.sciencealert.com/starlink-is-being-an-absolute-nuisance-to-astronomers.

[2] Siegel, Ethan (2019, Nov. 20). This Is How Elon Musk Can Fix The Damage His Starlink Satellites Are Causing To Astronomy. Retrieved from https://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.

[3] Hall, Shannon (2019, Nov. 11). As SpaceX Launches 60 Starlink Satellites, Scientists See Threat to 'Astronomy Itself'. Retrieved from https://www.nytimes.com/2019/11/11/science/spacex-starlink-satellites.html.

[4] US astronomers speak on SpaceX Starlink satellites. (2019, June 12). Retrieved from https://earthsky.org/human-world/aas-statement-spacex-starlink-satellites.

[5] Dustin Lang, David W. Hogg, Keir Mierle, Michael Blanton, and Sam Roweis. ASTROMETRY.NET: BLIND ASTROMETRIC CALIBRATION OF ARBITRARY ASTRONOMICAL IMAGES. The Astronomical Journal,139(5):1782–1800, mar 2010.

[6] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, St´efan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Ant ̂onio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. Scipy 1.0–fundamental algorithms for scientific computing in Python, 2019.