STScI | SPACE TELESCOPE SCIENCE INSTITUTE

Instrument Science Report ACS 2016-01

Satellite Detection in Advanced Camera for Surveys/Wide Field Channel Images

David Borncamp & Pey Lian Lim

January 6, 2016

ABSTRACT

This document explains the process by which satellite trails can be found within individual chips of an Advanced Camera for Surveys (ACS) Wide Field Channel (WFC) image. Since satellites are transient and sporadic events, we used the Hubble Frontier Fields (HFF) dataset which is manually checked for satellite trails has been used as a truth set to verify that the method in this document does a complete job without a high false positive rate. This document also details the process of producing a mask that will update data quality information to inform users where the trail traverses the image and properly account for the affected pixels. Along with this document, the Python source code used to detect and mask satellite trails will be released to users with as a stand-alone product within the STSDAS acstools package [1].

1. Introduction

Satellite trails are an occasional occurring feature in astronomical images, but where they occur they have the ability to compromise the scientific integrity of the image, making their detection important for good scientific analysis. Even though the Hubble Space Telescope (HST) is in space, it is still in low Earth orbit, allowing satellites to cross its line of sight. The large field of view of the Advanced Camera for Surveys (ACS) Wide Field Channel (WFC) unfortunately permits these events to be captured with relative ease. Smaller programs may not have the resources to manually examine each image to determine if a satellite is present, so this necessitates a software program that can detect if a satellite is indeed present and create a mask to ensure the scientific integrity of the image is not lost. This document describes an algorithm to

[1] http://acstools.readthedocs.io/en/latest/

both detect and mask satellite trails within ACS/WFC images. This algorithm is made available to users with no guarantee that it will correctly identify and mask all satellite trails in every image. This software was developed for use with ACS/WFC images and is provided with no guarantee that it will work on instruments other then ACS/WFC, however there is no reason that it should not provide correct results given the correct input parameters are used.

AstroDrizzle, the image combination software provided by STScI, easily flags satellite trails if there is more than one exposure (Gonzaga et al. 2012) in a visit. However, there are a significant number of HST programs that only have one exposure per field. In this case, users cannot rely on image combination to identify satellites. The solution for those cases is to use line detection to detect if a line (i.e., what a satellite trail looks like) has crossed the image. Performing line detection on these frames can be a hard task as the lines will not always appear straight due to the fact that WFC images suffer from extreme distortion, as high as 11% across its field of view (Avila et al. 2015) and the viewing geometry of both the satellite and the moving telescope can curve the trail. These effects can produce a curved rather than a straight line, making both detection and masking of a trail difficult. Here we describe an algorithm to do exactly that; in Section 2, we discuss requirements for the program, called `satdet`, in Section 3, we discuss the algorithm and in Section 4, we discuss the results from a study of the Frontier Fields data.

This software is released to the community with limited support and no warranty or guarantee that it will correctly detect or mask all satellite trails within their images. The associated software[1] is distributed by our Science Software Branch (SSB) via the acstools package.

2. Requirements for Software and Detection

2.1 Program

The Mikulski Archive for Space Telescopes (MAST) provides ACS/WFC users with a variety of data products, however HST users will most likely be interacting with calibrated, flat-fielded individual exposures that still contain Data Quality (DQ) arrays. This means that satellite detection should occur on those FLT/FLC images. In addition, the software should be easily implemented by users hence it should be written in Python and use only packages contained within the Ureka environment (http://ssb.stsci.edu/ureka/).

2.2 Detection

The detection algorithm operates on the assumption that a satellite will cross from one edge of the detector to another. While it is possible astronomical sources within an image mimic satellite trails in this respect, the program should not get confused by these issues. Possible edge-

spanning features that should be ignored by this algorithm include extended astronomical sources such as galaxies and nebulae, as well as CCD saturation trails, stellar diffraction spikes, grazing-incidence cosmic rays, and scattered-light anomalies such as glint. See HLA ISR 08-01 for examples of other anomalies

2.3 Masking

As a satellite crosses the detector's field of view, it can create a trail that cannot be effectively modeled and applied to every satellite as some satellites may be resolved as they pass over the field of view. The `satdet` module should create a mask that extends to include the wings of the profile and update the data quality array to the value of 16384 (flag for satellite trails; Gonzaga et al. 2015) where the trail occurs.

3. Algorithm

The `acstools.satdet` module is divided into two main parts: 1) detecting if a satellite trail is present in the image (described in Section 3.1); and 2) creating a mask for the data quality array (described in Section 3.2). Initial limitations on ACS/WFC data are discussed in Section 3.3 but it should be noted that the detection algorithm would return an approximation of where the trail occurs, the masking algorithm will make a much better determination of the satellites path.

3.1 How Detection Works

Satellites can have any brightness or even pulsate within an image. To correct for this the program starts by rescaling the image to make it easier to identify fainter or varying satellite trails while still preserving the lines of more intense trails. The user is able to set the limits of the rescaling, but the defaults use a lower bound of 4.5% and an upper bound of 93%. These bounds have been empirically found to give the highest success rate for ACS/WFC images, however the ability to change these values has been included so that this may work with any detector or in images with faint trails. This step uses the Python package `skimage` (Van der Walt et al. 2014) `exposure.rescale_intensity` module to rescale the image. The results from a relatively faint trail are seen in Figure 1.

After the image is rescaled, an edge image is made using a Canny algorithm (Canny, 1986). This program uses the implementation within the `skimage` package. This edge image creates an easy way to find boundaries in any given image by detecting discontinuities in the intensities from pixel to pixel which define the edge of a feature in the image. The Canny edge detection algorithm is well suited for satellite retrieval for several reasons: it will produce thin, continuous rising and falling edges; it will smooth noise in the image using a Gaussian filter (the user is able to specify the width of the filter using the sigma keyword); and due to the hysteresis

linking it is less likely to get confused by trails whose intensity vary as it crosses the image. The hysteresis linking allows for a threshold value to be set by the user to select how edges are linked. These features control the amount of detail that is included in the edge image and helps to link varying trails. The default values of .1 and .5 have been found to be the most effective for ACS/WFC images, but the ability to change the values has been retained if users find these values to be incorrect for their images. The results of this edge detection are seen in Figure 1.

Once the edge detection is complete the edge image is then cleaned up by removing small objects whose perimeter is less than 75 connected pixels, counted using all eight of a pixel's neighbors to determine the size of the perimeter. This significantly refines the edge image (seen in Figure 1) and allows line detection to be much more accurate as it does not try to link up erroneous edges.

When a good edge image is created, a Probabilistic Hough Transform (Galamhos et al, 1999) is performed to look for straight line segments in the image. The Probabilistic Hough Transform is more efficient than the standard Hough transform because it only uses a sub-sample of points to detect a line. This significantly cuts the number of places to look for a line while preserving accuracy for well-defined lines such as satellite trails. This also allows for fitting of smaller line segments, which also increases the likelihood of correctly identifying a trail. Since the satellite trail is likely to be straight to within 1 degree across hundreds of pixels even accounting for ACS/WFC geometric distortion, we are able to fit a straight line to this segment. This allows us to more easily find and fit curved and varying lines in an image. An example of the line segments returned is shown in Figure 2. After line segments are found, the angle of the segments is found and rounded to nearest 5 degrees. Then, any segment whose angle is not the same as the mode of all of the angles is removed from the possible satellite track. This is because cosmic rays and diffraction spikes may be fitted with their own line segment and we do not want them to be confused and included within the satellite.

Using the end points of the segments and the filtered list of angle for the segments, the program will determine if a satellite traversed the image or not. The satellite must be within the user definable input variable 'buf' of 2 edges of the image and be at the correct angle for the type of crossing (e.g. between 0° and 90° if crossing from top to left). Assuming the trail is seen to go across the image and is at the correct angle for its crossing, a list of the segment endpoints is returned. If the trail is not seen to traverse the image an empty list is returned. The segmented endpoints are seen over plotted on the edge image in Figure 2.

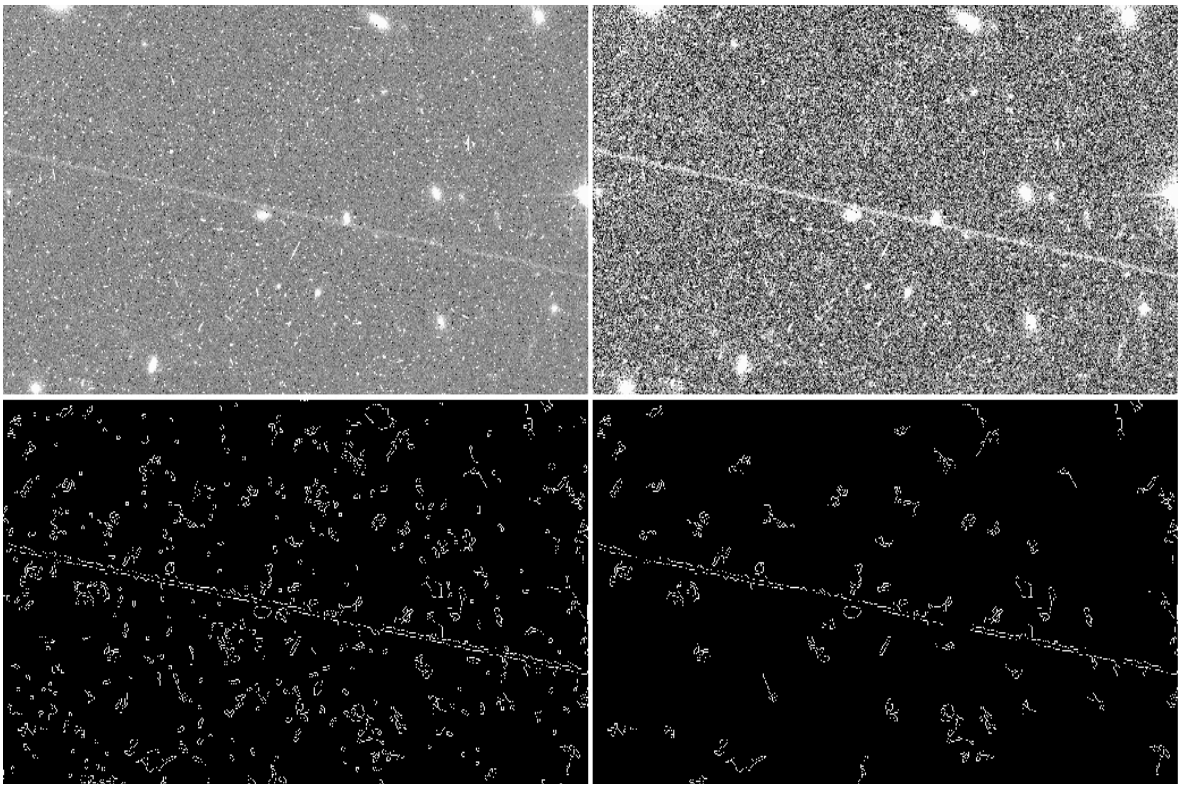

Figure 1. Satellite trail in jc8n81r7q_flc.fits. Top Left: Original FLC image. Top Right: Rescaled image of the same location. The satellite trail in the rescaled image is much clearer which makes edge detection much easier. Bottom Left: Edge detection on rescaled image. Bottom Right: Cleaned up edge image with small objects removed. The trail of the satellite is preserved and complete in the cleaned up edge image.

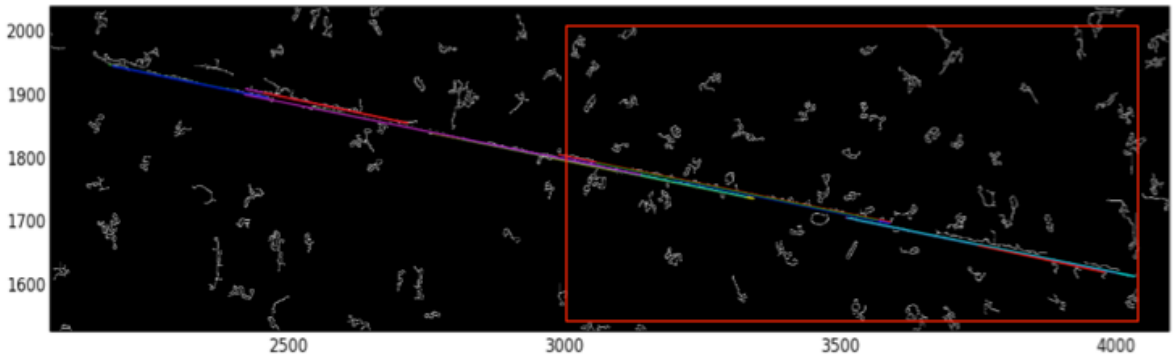

Figure 2. Segmented edge detection over plotted for the same image as in **Figure 1**. This is a zoomed out section of **Figure 1** so that the segmentation can be easily seen. X and Y units are pixels of one chip of the input image. Red box indicates the zoomed in region in for **Figure 1**.

3.2 How Masking Works

Masking a satellite trail presents an interesting problem, as it is similar to fitting a linear profile to a feature that can occur at any orientation within an image and be curved in the image.

It is analogous to extracting long slit spectra on a distorted camera whose dispersion can occur at any angle in the image.

The masking algorithm takes the endpoints returned by the detection algorithm and works out a rough angle at which the trail is crossing the image. The image is then rotated so the trail is horizontal. Using the first endpoint returned by the detection algorithm as a center point, a sub region of the dimensions set by the keywords *subwidth* and *sublen* is taken. The sub region is then flattened and a median value for all pixels in each row is taken for every column. This allows the program to filter out bad points and sources within the image while still preserving the parts of the satellite trail that are constantly above the background level in the row. A bi-weight mean is then computed for the flattened sub region and everything above a user defined sigma value is flagged as being part of the satellite trail. In some cases, a smaller region may lead to a better fit of an intense, curved trail but users should use the *subwidth* and *sublen* parameters with caution as too narrow a sub region will allow background stars to be included as part of the trail and will make the mask inaccurate. The default parameters for these values have also been empirically found to be the most accurate on a small sample of ACS/WFC images. Since not all satellites can be fitted with a Gaussian profile (some satellites may be resolved) a user definable padding value is added to both edges of each detection to ensure that the wings of the satellite are contained within the mask. The user can set this padding value by changing the *pad* keyword. In general, this should be kept rather large to avoid the trail contaminating the science measurements.

Once this process is completed for the initial starting point, it is repeated for each sub region, moving to the right (positive column numbers) along the satellite trail, updating the center point of the sub region as it goes to account for possible curvature of trails. Once the right edge of the image is encountered, the program then goes back to the original starting point and works left to ensure the entire trail is masked.

3.3 Discussion of the Software and Current Limitations

The software will detect many faint trails in an ACS/WFC image by using the default parameters. However, it can fail at finding diffuse, faint trails. Even after rescaling an image, it can be very hard to detect edges that are only 1σ above the background as seen in Figure 3. While the trail is there, the program using the default values does not consider the edges of the trail to be linked because they are too spread apart. This particular situation appears to be an outlier, but further refinements to the algorithm could allow for correct detection of these types of trails.

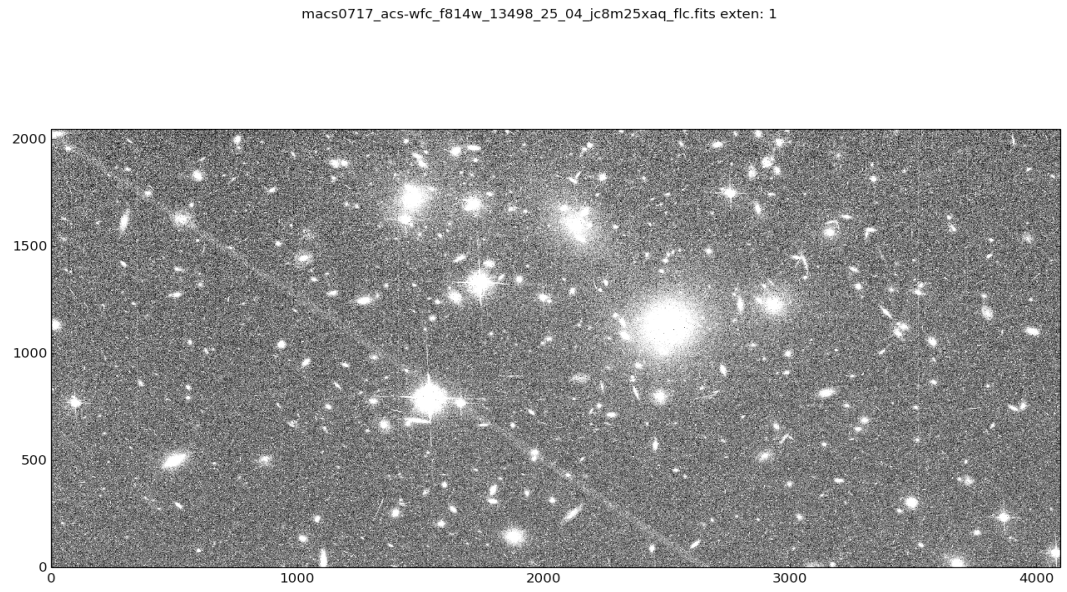

Figure 3. Rescaled image of a faint, very diffuse satellite trail in a crowded field of resolved sources. These kinds of trails can be extracted if the correct parameters are used, but the default parameters will fail at discovering the satellite trail due to the low signal to noise ratio of the satellite. The most effective way of altering the parameters to detect these trails is to change the rescaling parameters.

In testing this algorithm, it has been found that many false positives are a result of long, interspersed cosmic rays traveling at the same angle while crossing the corners of the chip as seen in Figure 4. This is unavoidable to some extent as this program was designed with finding satellite trails whose brightness varies as they cross the detector in mind. However, the false positive rate due to this type detection can be reduced by defining a more aggressive lower bound on the length of a line segment.

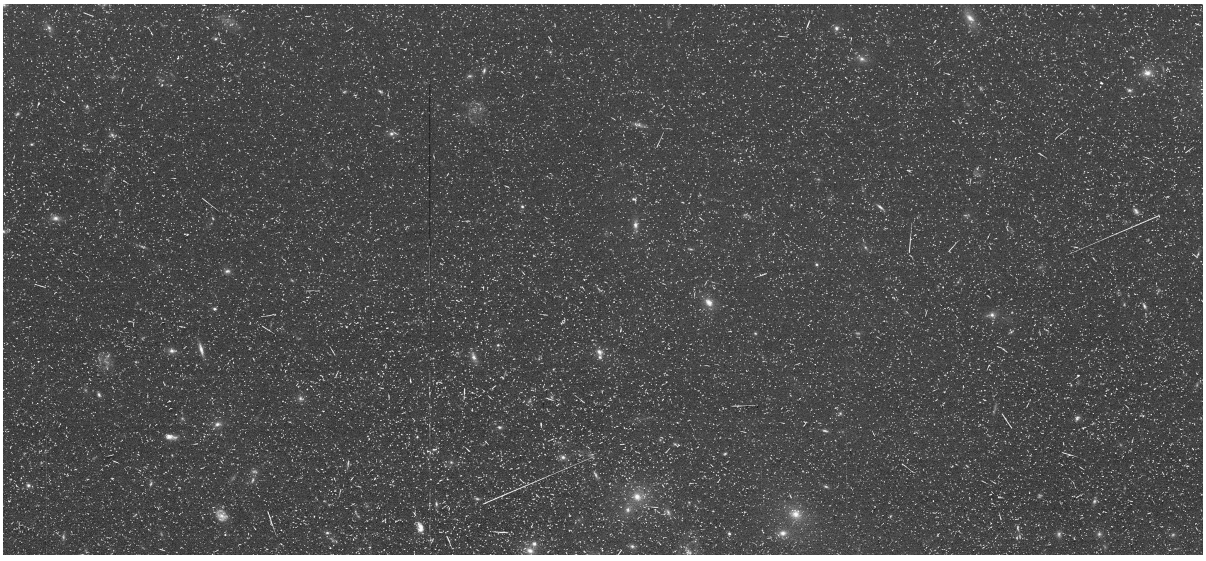

Figure 4. Example of image with false positive satellite detected. There are two long cosmic rays that strike the detector at similar angles close to the edges of the image. This type of false positive accounts for the majority of false positives the program detects.

4. Analysis of Results

The first test of this algorithm is to see if it will flag real astronomical sources in an image as satellite trails. Several datasets including the Orion Nebula, saturated sources in 47 Tuc and extended galactic sources in the Frontier Fields (HST program ID 14037 PI: Lotz) datasets have shown that the algorithm only correctly flags distinct lines that cross the images and ignores real astronomical features. This is because many astronomical features will not have a sharp cutoff similar to that of a satellite trail.

The Frontier Fields program reviews every image by eye to determine if there is anything within the datasets that is not an astronomical object. This includes finding and masking satellite trails by hand. Because every image is individually vetted by eye, it provides a large "truth set" by which to test this algorithm. We ran many of the ACS/WFC image in the HFF program through our software and compared the results to the truth set. The results of this test are presented in Table 1, which shows the completeness and false positive rate for satellite detection using this algorithm. The detection algorithm has a 95.56% completeness rate and ~1.63% false positive rate within the Frontier Fields dataset. Many of the false positives are due to cosmic rays that cross the corner of a chip or that hit the edge of the chip at the same angle.

Table 1. Table showing completeness and false positive rate of satellite detection algorithm compared to the HFF truth set using only the default values. The algorithm must be run twice on each image, once for each chip, a detection in one chip counts as a detection in the entire image.

Dataset	# ACS Chips in Dataset	# Manually Flagged	# Flaggged by Software	False Positive	# Not Flagged by Software With Satellite Trail
abell2744	138	6	7	1	0
abell2744 – par	177	8	11	3	0
macs0416	180	9	11	5	1*
macs0416 – par	140	6	7	1	0
macs0717	161	7	9	3	1*
Total	**796**	**36**	**45**	**13**	**1**
% of Total	**100%**	**4.52%**	**5.65%**	**1.63%**	**4.44%**

* See **Figure 3** for incorrectly flagged image explanation.

5. Conclusions

Satellite detection can be done in many different ways, but the method presented here is just one way that works for individual images on ACS/WFC. The method described here has fulfilled the requirements by detecting and masking faint and bright satellites in distorted space on an individual chip level and creates a mask to notify users that these pixels may be compromised. While this program has shown to be effective, it is up to the user to check that the masks and verify that they correspond to the actual satellite trails. We leave it up to the user to decide if they wish to incorporate or exclude them in subsequent processing based on their validity.

We have presented an algorithm, along with a software implementation, that detects and masks satellite trails marring ACS/WFC images. Though but one of many possible algorithms for the task, our code exhibits high completeness and low false-positive rate when validated against HFF imagery. Nonetheless, we strongly encourage users of this software to fine-tune the adjustable parameters in order to achieve trail-masking of the highest fidelity for their particular data.

An example of how to run this software can be found along with the API documentation here: http://ssb.stsci.edu/doc/stsci_python_x/acstools.doc/html/satdet.html#examples

Acknowledgements

We would like to thank the ACS team for reviewing the content of the ISR and more extensive testing of the algorithms. We would also like to thank the Frontier Fields team for manually vetting all of the images taken to make the testing of this program possible. The detection part of the program was created for the class COSC 602 "Image Processing and Pattern Recognition" at Towson University taken by David Borncamp Fall 2014.

References

Avila, R., et al. 2015, “ACS Instrument Handbook”, Version 14.0 (Baltimore: STScI)

Canny, J., A Computational Approach to Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986

C. Galamhos, J. Matas and J. Kittler, “Progressive probabilistic Hough transform for line detection”, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999.

Gonzaga, S., et al., 2013, ACS Data Handbook, Version 7.1 (Baltimore: STScI).

Gonzaga, S., Hack, W., Fruchter, A., Mack, J., eds. 2012, The DrizzlePac Handbook. (Baltimore: STScI)

M. Stankiewicz, S. Gonzaga, B. Whitmore, HLA Instrument Science Report 08-01 (Baltimore: STScI)

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) http://dx.doi.org/10.7717/peerj.453