École Polytechnique Fédérale de Lausanne

Rotation estimation of detected satellites

by Noah Kaltenrieder (301368)

Bachelor Thesis

Mathieu Salzmann
Thesis Supervisor

June 2021

Acknowledgments

I really would like to thank my supervisor Prof. Mathieu Salzmann who helped me a lot and all the people that participated in a way to this project, namely Dr. Cameron Lemon, Prof. Jean-Paul Kneib, Prof. Frédéric Courbin, and the SSA Team. I also want to express my gratitude to Yann Bouquet who helped me with pleasure when I had questions. Finally, I also would like to especially thank Alexandre Di Piazza and Marcellin Feasson with whom I have collaborated through all the semester in different ways.
Thanks a lot to Prof. Mathias Payer for the thesis template that I used to write this report [1]

Lausanne, June 2021 Noah Kaltenrieder (301368)

Abstract

We present an approach to extract the intensity along the satellite streaks that can be found in images that were obtained by OMEGACAM on the VLT Survey Telescope. In this project, we only analyzed the long tracks that were created by high velocity objects, to try to get an estimation about their rotation. We used Fourier transform to get the periodicity of the intensity along the tracks. Due to the lack of real data about rotation, we first had to create fake streaks with sinusoid intensity and different angles. We manually got the tracks to run the code on them, so ideally this project should be integrated with DetectSat [2] so that all the process can be done automatically. We found very great result with it and run the pipeline on real tracks. By making assumptions about the satellite, like their altitude, we get possible results on real streaks, but since we don't have access to the real values, we can't say if it is close to reality or not.

Contents

Chapter 1

Introduction

This project is about estimating the rotation of spatial objects detected by the OMEGA-CAM on the VLT Survey Telescope. In these times, we have a lot of flying objects, it can be debris, useful satellites or dead satellites, around the Earth and it becomes more and more necessary to have a clear view of all these objects. The project of Yann Bouquet "DetectSat"[2] was focused on detection of the satellite tracks on fits files, but with both, small (low velocity) and long (high velocity) streaks. For this project we had to focus on the long tracks, so that the variation of intensity along the line could be better detected. The rotation estimation can be used to determine which kind of object the streak is and, if it is a satellite, it may be used to better find out which one it is. Ideally, this should be run on the lines detected by DetectSat so all the process could be done with only the fits file as input.

The main challenge is that we don't have access to data about rotation of satellites, so we had to first create our own streaks with sinusoid intensity. This lack of resources is very restrictive, since even if we get a result, we cannot know the precision of this method and we cannot be sure that it is right or completely wrong. This is why we had to begin with creating lines that seem realistic and try with random angles.

The second part of this project was to retrieve the periodicity of the sinusoidal function along the streaks that we have previously generated, and finally try this on real tracks. Since this project does not recognize the lines, we had to find the coordinates of the streaks by hand and give them as input to the pipeline. For this part, there are many

factors that can decrease the precision of the pipeline, e.g., if the line crosses a really bright star, it gives a peak of intensity and can lead to wrong result, or if the brightness of the streak is too small compared to the brightness of the background of the sky on the image.

Chapter 2

Inputs

In this project, we used fits images that were generated by the OMEGACAM on the VLT Survey Telescope, and we used a script to create a mosaic of 32 images, separated by NaN values, that are $4'000 \times 2'000$ pixels. So the image with all the mosaic's dimension is $16'000 \times 16'000$ pixels. A mosaic is approximately 1.5-2 GB of data.
We principally used the fits file "*OMEGA.2020-02-22T02:11:36.295_fullfield.fits*", corresponding to the captured image February 22, 2020, at 02:11:36, and with the pixels scaled using **ZScale** to be able to see the streaks as wanted.

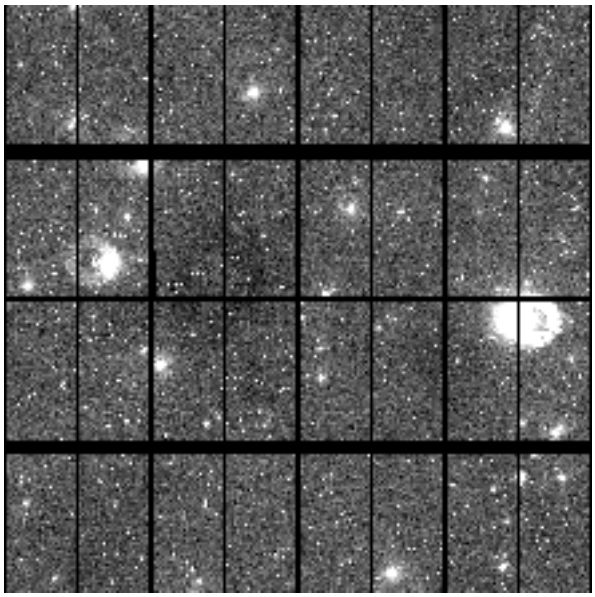

Figure 2.1 – The main FITS image that we used

The pipeline is being run on the entire file, with the coordinates that are relative to the entire file : the origin is at the bottom left of the fits file. The first part of this project, which is about line generation, is completely based on this file Figure 2.1. We chose this file because we can clearly see a really interesting track that goes through the image :

Figure 2.2 – The interesting streak

This track on Figure 2.2, the one that goes through four blocks, was clearly visible and the variation of intensity can also be notified, so it was a great way to test the pipeline with a streak that is not too difficult. Thanks to Marcellin, he could identify this and it turned out that this streak is actually a debris of the European carrier rocket Ariane 2 ! We know that it is at a geostationary altitude, so $\sim 36'000\ [km]$ and has a speed of $\sim 3'065\ [\frac{m}{s}]$. We will use this information later to analyze the coherence of the result obtained by the pipeline.

Chapter 3

Line Generation

When we started this project, we had no data about rotation of satellites and their corresponding tracks on a file, so we had to first generate streaks on the image myself, to be able to test the pipeline. First, to generate a line we had to choose an existing star in the file and cut it out to have a PSF to convolve my line with. The PSF : "Point Spread Function" is like the quantity of blurring of a point object. Here, we had to use it so that my generated line seems more realistic in the image.

Figure 3.1 – The line without using the PSF

Figure 3.2 – The line using the PSF

We will now explain the method we have used to achieve this result. First, we have to choose a star by hand and cut it out to use it as a PSF. The dimension of the star cut out is 30px, so we initialized an array of height's dimension of 30px and an arbitrary width,

with zeros everywhere, except at the center of the array, the center line of one pixel of height at the center is set with sinusoid values. Once, we have these two pieces, we can convolve the two arrays together to make the final line with an appropriate PSF.

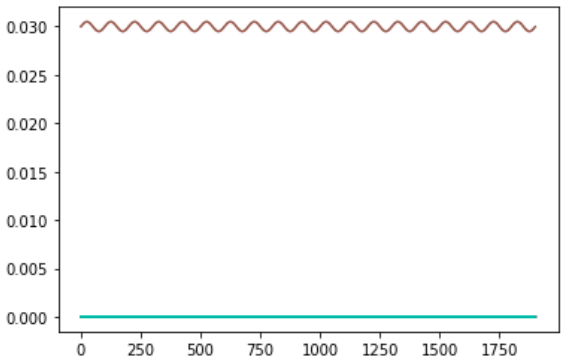

Figure 3.3 – Intensity of each row of the line's array

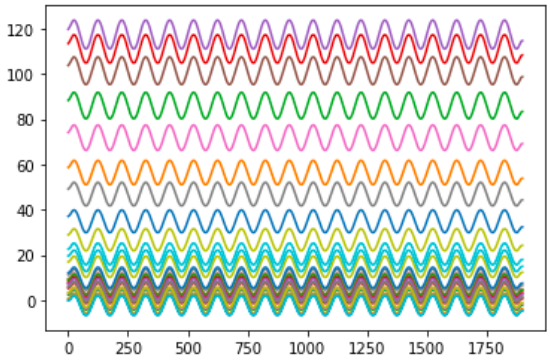

Figure 3.4 – Intensity of each row of the line's array after the convolution

We finally rotated the final convolved line by a random angle and add it to the image by summing up the values with the image's array.

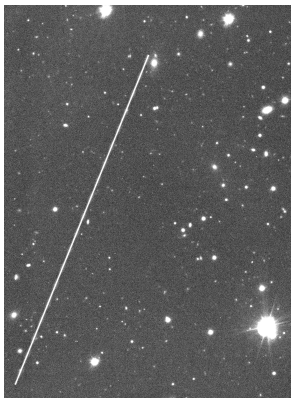

Figure 3.5 – The generated line on the image

The final result of the line in the image is pretty satisfying and coherent, as we can see on Figure 3.5.
The inconvenience of this method is that we had to choose a star by hand and it will work great for this specific file, if we want to make this code works with another image we have to choose a star from the new file to have a coherent PSF to integrate the line better on it.

Chapter 4

Find the intensity along the track

Once the generation part was done, we started the main part of the project that was finding back the periodicity of the previously generated sinusoid intensity streak and finally apply the final pipeline on real tracks.

The method that we used was very simple : We just sum up all the values on the y coordinate for all x coordinate, so it returns an array with a height of one and with a width as long as the streak. Here, the width was found manually, since we had to enter the start and the end coordinates of the track as input for the pipeline, it was trivial to recover the width of the line, even when the line was not horizontal, by using trigonometry.

$$width\_of\_the\_line = \frac{end\_line\_x - start\_line\_x}{cos(angle\_line\_in\_radian)}$$

Once we have this array with the intensity of each x point, we can consider it as a signal and use the fast Fourier transform (FFT) algorithm [3] on the array to calculate the discrete Fourier transform (DFT) that can be used to find the frequency of the signal. We also used the inverse FFT to try to remove a bit the noise that we get from the image; indeed the line is not isolated from the background of other stars or other streaks that can add more or less noise to the signal. We also choose to use a threshold for the number of periods that can be in a track to remove a lot of noise too. We have set it arbitrarily to 100 repetitions, because the width of a streak is limited to the width of one image of the mosaic, which is $2'000$px. It would mean that we would see a

repetition every 20px, so the frequency of the signal would be 0.05 [px^{-1}] and, assuming a geostationary object, this would imply 4.18 rotation per second and it is very unlikely that an object has such a high rotation rate.

Finally, we used the "**curve_fit**" function from the scipy library that use the least square optimization to find the frequency of the original sinusoid function. But to use this function, we have to make a good first guess of the parameters, so that they will converge fast into the right value. A good first guess for the frequency, which is the only parameter of the sinusoid function that really interests us, is to divide the number of cycles that we found with the Fourier analysis part, by the width of the streak.
Using this pipeline, we usually find good estimation of the wave but it can vary a lot when there are a lot of noises.

We cannot use the result directly found with the pipeline explained just above, because it expresses the frequency by pixel, to be able to compare it with a reference value as we would like, we have to convert it to spatial scale e.g., for the frequency $[s^{-1}]$. To do this we had to make some assumptions :

1. The flying object is geostationary
    - So the altitude would be $\sim 36'000'000\ [m]$
    - With a speed of $3'065\ [\frac{m}{s}]$
2. The fits file has a scale of $0.21\ [\frac{arcsec}{pixel}]$

With these elements, we can find the rotation rate easily :

$$one\_cycle\_in\_pixel = \frac{1}{frequency\_found}\ [pixel]$$

$$arcsec\_by\_cycle = 0.21 \times one\_cycle\_in\_pixel\ [arcsec]$$

$$distance\_one\_cycle = 36'000'000 \times (\frac{\pi}{180} \times \frac{arcsec\_by\_cycle}{3'600})\ [m]$$

$$rotation\_rate = \frac{3'065}{distance\_one\_cycle}\ [s^{-1}]$$

This rotation rate, which is the number of rotation of the object in one second, can be used to compare what we have found with the existing information about the objects that we will detect.

4.1 Result on Simulated data

Running the pipeline on a line that we created previously gives pretty satisfying results. Here we can see the result with a basic case with an horizontal line but that does cross a star or bright object.

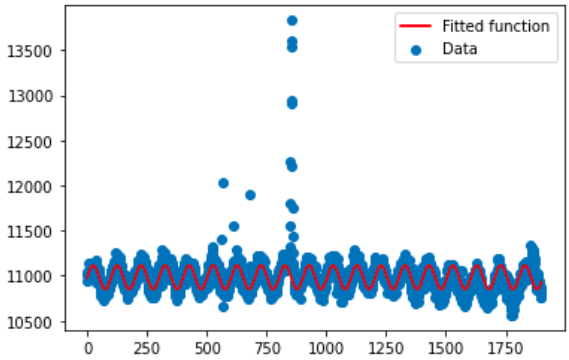

Figure 4.1 – Total intensity by x coordinate on the line

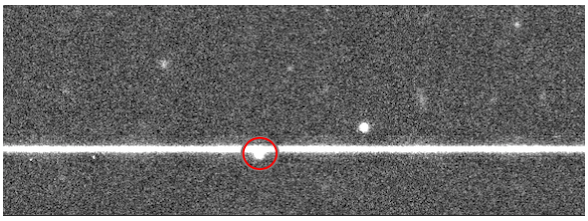

Figure 4.2 – The star that is crossed by the line

As we can see on Figure 4.1, the fitted function seems really close to the data even if there is a peak of intensity at some point, because of the star crossed. It is a good sign that our pipeline can resist, even if it is a small amount of noise here.

Chapter 5

Cast the frequency

We cannot use the result directly found with the pipeline explained just above, because it expresses the frequency by pixel, to be able to compare it with a reference value as we would like, we have to convert it to spatial scale e.g., for the frequency $[s^{-1}]$.

Chapter 6

Final Result

Running the pipeline on a line that we created previously, takes approximately less than 10 seconds, by block of the mosaic, using a 1,8 GHz Intel Core i5 dual core on a MacBook Air 2017.

The related work section covers closely related work. Here you can highlight the related work, how it solved the problem, and why it solved a different problem. Do not play down the importance of related work, all of these systems have been published and evaluated! Say what is different and how you overcome some of the weaknesses of related work by discussing the trade-offs. Stay positive!

This section is usually 3-5 pages.

Chapter 7

Conclusion

In the conclusion you repeat the main result and finalize the discussion of your project. Mention the core results and why as well as how your system advances the status quo.

Bibliography

[1] Mathias Payer. *Template for EPFL (BSc, MSc, or doctoral) theses and semester projects*. 2020. URL: https://github.com/HexHive/thesis_template (visited on 2021).

[2] Yann Bouquet. *Detectsat Repository*. 2020. URL: https://github.com/YBouquet/detectsat (visited on 2021).

[3] W.T. Cochran, J.W. Cooley, D.L. Favin, H.D. Helms, R.A. Kaenel, W.W. Lang, G.C. Maling, D.E. Nelson, C.M. Rader, and P.D. Welch. "What is the fast Fourier transform?" In: *Proceedings of the IEEE* 55.10 (1967), pp. 1664–1674. DOI: 10.1109/PROC.1967.5957.