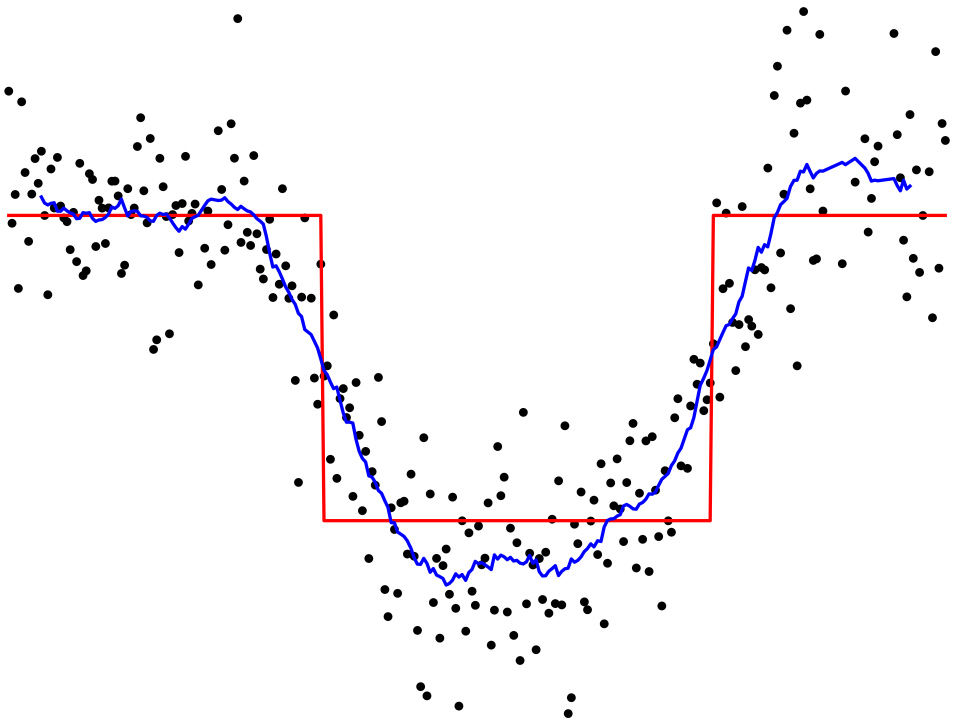
Kantonsschule Im Lee Winterthur

Matura Paper Fall Semester 2022

The Developing and Testing of a Software to Help the Search for Exoplanets

Josia John, 4a
Supervisor: Thomas Graf



Winterthur, 2023-01-09

Contents

1 Abstract **3**

2 Introduction **3**

3 Motivation **3**

4 About Exoplanets **4**
4.1 What are Exoplanets? . . . 4
4.2 How can Exoplanets be Observed . . . 5
4.3 Exoplanets in this Paper . . . 6

5 Use Cases **6**
5.1 Finding Exoplanets . . . 6
5.2 Other Kinds of Variable Stars . . . 7

6 Working Principle of the Software **7**

7 Overview of the Software **8**
7.1 Data Calibration . . . 8
7.2 Star Detection . . . 8
7.3 Alignment . . . 9
7.3.1 Use of Deep Sky Stacker . . . 9
7.3.2 Implement Algorithm . . . 10
7.4 Brightness Calculation . . . 12
7.5 Data Cleansing . . . 14
7.6 Light Curve Generation . . . 14
7.7 Data Analysis . . . 15
7.8 Output . . . 16

8 Implementation Details **17**

9 Using the Software **18**

10 Data **19**

11 Results **19**
11.1 Comparison with AstroImageJ, a Professional Photometry Software . . . 19
11.1.1 Processing Time . . . 20
11.1.2 Results . . . 20
11.1.3 Features . . . 20
11.1.4 Conclusion . . . 20
11.2 More Examples . . . 21
11.2.1 First Dataset, Matt Baker . . . 21
11.2.2 Gornergrat Observatory . . . 21
11.2.3 Siding Spring Observatory . . . 23

A Thanks **24**

B Glossary **24**
B.1 Sigma-Clipping Method . . . 24
B.2 Binary Search . . . 24

C References **25**

D Code **26**
D.1 run.py . . . 26
D.2 generateCatalogs.py . . . 26
D.3 mergeCatalogs.py . . . 27
D.4 generateField.py . . . 31
D.5 getBrightnessOfOneStarInField.py . . . 32
D.6 generateBrightnessOfAllStarsInAllImages.py . . . 33
D.7 generateLightCurve.py . . . 35
D.8 analyzeLightCurves.py . . . 36
D.9 output.py . . . 37
D.10 Auxiliary Scripts . . . 38
D.10.1 main.py . . . 38
D.10.2 getFilelist.py . . . 39
D.10.3 readImage.py . . . 39
D.10.4 myAlgorithms.py . . . 40
D.10.5 getTimeOfObservation.py . . . 40

1 Abstract

This paper describes the ideas and algorithms which are implemented in the program ExoScanner. ExoScanner makes the search for exoplanets accessible to amateur astronomers with medium to high-end equipment. ExoScanner's key feature is its ability to automatically check a whole frame for exoplanets while keeping the running-time low. Other programs are only able to analyze one star at a time. This significant reduction in effort needed to analyze an image-sequence for exoplanet-transits could motivate many amateurs to run the program on their own data, possibly revealing a bunch of new exoplanets.

2 Introduction

It does not matter whether we are searching for an alternative for earth, trying to become an interstellar species, or searching for alien life. Fact is that we are interested in exoplanets. Even so, did we only just verify the first exoplanets in 1992, 30 years ago. In other words, our parents had to grow up without knowing any exoplanets. How boring. In the last 30 years there have been enormous improvements and developments, to the point where amateurs can observe some known exoplanets without even owning a telescope. All that is needed is a good digital camera with an appropriate lens and a tracking device. The lens should have a focal length of 200 millimeters or more and a focal ratio of less than f/5. A tracker can be built at home for a few bucks.

Exoplanets are a very new topic, and we have not yet detected that many. That is why I decided to write a software which should make the search for new exoplanets using the transit-method easier. The main idea behind my software is that it shall analyze all stars in a frame at once instead of manually analyzing one star after the other. This allows for the process of finding exoplanets to be automated even further. The required time to analyze all the stars in a frame is decreased drastically.

3 Motivation

With my interest and experience in astrophotography came the idea to observe an exoplanet-transit. As usual, when doing something new, I did some research. After some time I knew a lot about how exoplanets can be observed and what equipment might be necessary. The only thing that was left, was a good software to analyze the captured data. A very popular program for the analysis of exoplanet-transits is AstroImageJ. I tried it and quickly realized that it is a very advanced program with many features. So I figured that there has to be a feature which analyzes all my images, searches for exoplanet-transits and tells me the results. But it turned out that the search for this exoplanet-transit had to be done manually. I had to select a target star, which then was analyzed by the software. From there, the program behaved as expected. But I was quite surprised to see that, if I planned to find an exoplanet myself, I would have to click on the stars manually, and then, for each star, let the program run for about three minutes. That is a very repetitive and time-intensive process. When trying to do it for a few stars, it did not take long for me to loose track of which stars I have already analyzed. It means that no amateurs would analyze their captured data with the hope of discovering an exoplanet, because it takes so much time and effort. I thought to myself that it should not be too hard to automate that process to make the search for new exoplanets less time-intensive and more accessible to amateurs. Then, I moved on with my life.

Some months later, I got the task of writing my Matura paper in high-school. While thinking about some interesting topics, I remembered the idea of automizing the search for exoplanets. It was the perfect project idea. It was a quite challenging project for sure, but not impossible. From there it did not take me long to start developing a program with the vision of allowing many amateur astronomers all over the world to install my program and test the data they have lying around for anomalies, possibly revealing many exoplanets.

The thing amateur astronomers most often do is astrophotography. When doing astrophotography, many images are recorded which are then later merged into one final image. The durations of these recorded image sequences often last multiple hours. That means, that many amateur astronomers already have a lot of data that could reveal exoplanets. On the other hand, professional telescopes rarely look at one target for multiple hours, because time on their big telescopes is just too expensive.

There were two options, writing the whole software from scratch, or improving AstroImageJ. I chose the first option because it allowed me to improve the runtime of the whole process with some nice insights and ideas. The other reason was that I wanted to deeply understand all the steps required to do such a photometric analysis.

This project was perfect as it combined my interest in both astronomy and informatics. My first idea was to do astrophotography, but I quickly realized how risky that would be. Everyone has experienced the weather being bad at the worst times possible. I also think that writing a program which is able to automate the search for exoplanets is a lot more interesting than just taking a few pictures of the night sky.

4 About Exoplanets

4.1 What are Exoplanets?

Exoplanet is an abbreviation for extrasolar planet. That means it is a planet which is not orbiting our star, the sun, but orbiting some other star. The International Astronomical Union, or IAU for short, has defined exoplanets as follows [4], [2]:

1. The object should orbit a star (or a brown dwarf or stellar remnants)
2. The mass ratio between the exoplanet and it is central object should be below the L4/L5 instability. This means that $\frac{M}{M_{\text{central}}} < \frac{2}{25+\sqrt{621}}$ should hold.
3. The minimum size and mass of the object for it to be considered an exoplanet are the same as in our solar system.
4. The object's mass must be small enough such that thermonuclear fusion of deuterium will not take place.

While 1 and 3 both intuitively make sense, 2 and 4 might not be obvious. That is why I will shortly explain their rationale.

2: The mass ratio being smaller than $\frac{2}{25+\sqrt{621}} \approx \frac{1}{25}$ means that the planet has to be at least 25 times lighter than its host star. This will exclude two similarly sized objects orbiting each other.

4: Thermonuclear Fusion of deuterium is a process which takes place in stars. Because we do not want to include stars in our definition of exoplanets, we define that this process should not take place. The case of two stars orbiting each other is excluded with this restriction.

4.2 How can Exoplanets be Observed

Direct Imaging

The first thought one might have when thinking about detecting something with the help of a telescope and a camera is to just photograph it. The problem is that exoplanets are small and very faint because they can only reflect the light from their star. That makes capturing the planet directly almost impossible. An example for the direct imaging of an exoplanet is the system of the star HR 8799. The Keck and Gemini telescopes, both located in Hawaii, have discovered this system in 2008 [5]. The system is around 133 light-years away. An image of the system can be seen in figure 1. In this paper, this method will not be discussed any further because a telescope with a huge diameter would be needed to succeed with this method. ExoScanner is designed for to work with telescopes that fit in backyards.

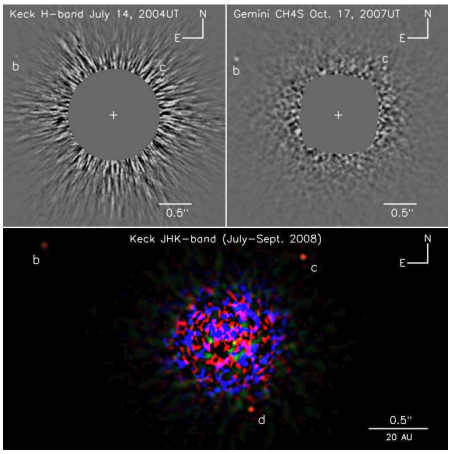


Figure 1: Direct images of the system HR 8799, Figure 1 from [5]

Indirect Imaging

Currently, the most common method to find and observe exoplanets is the transit method. If the exoplanet's orbital plane lies in a way that the planet will cross between its host star and earth, the brightness of the host star will decrease minimally when the planet is in between the observer and the star. This dip can be observed with telescope and camera. By analyzing the data, a light curve can be generated. Such a light curve is shown in figure 2. This happens once every orbit. Most planets that have been found that way, have an orbit time of a few days. The lengths of these transits are often about two hours. That is the case because it is quite hard to observe a dip in brightness which lasts for longer than one night. Also, the dip has to be detected multiple times in order for the exoplanet to be verified. That is of course a lot easier if the transit happens every few days rather than once a year.

To get an idea of the properties of such planets, let us look at the example WASP-140 b. It is 384 light-years away from us. WASP-140 b is an exoplanet orbiting the star WASP-140 once every 2.2 days. It is, like Jupiter, a gas giant, and has the mass of 2.44 Jupiters, or 775.7 earths. That is one reason why it is definitely not habitable. Another reason is that it is very close to its star. To get a better idea, its orbit can be compared to some orbits of our solar system. Figure 3 shows the star WASP-140 in the center and the planet WASP-140 b orbiting it (in white). It also shows our planets orbits. Like that, the different orbits can be compared easily. In green it shows the habitable zone. As we can see, the exoplanet is far away from it.

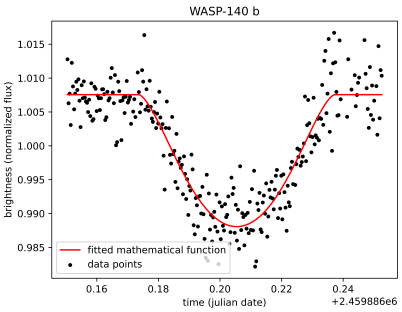


Figure 2: transit light curve WASP-140 b

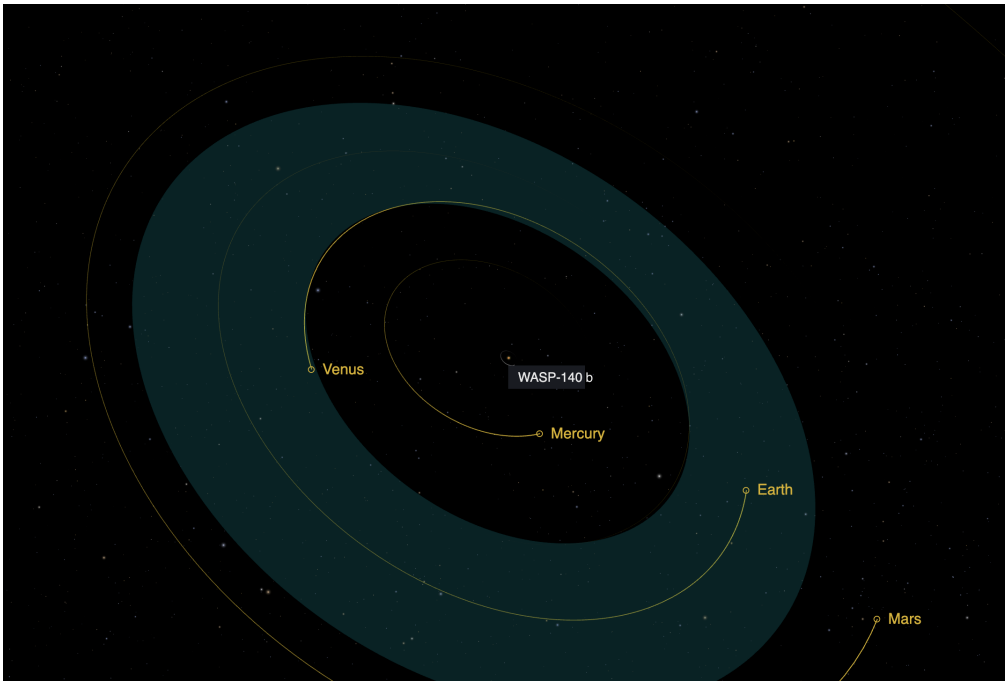


Figure 3: NASAs *Eyes on Exoplanets*: WASP-140 b
(https://exoplanets.nasa.gov/exoplanet-catalog/3422/wasp-140-b/, visited on 2022-12-28)
comparing WASP-140 b with our planets

Thanks to enormous advancements in technology, it is now even possible for amateurs to detect these transits from their backyards. Because of the relatively low requirements, and thus the high accessibility, this is the method I will be writing my software for.

4.3 Exoplanets in this Paper

My goal is to write a software that can find candidates for host stars of exoplanets using the transit method. Because I am only searching for candidates, the exact definition of exoplanets is not that important. The scientific analysis of these candidates should then be done with software explicitly designed for analyzing exoplanet transits with high accuracy.

5 Use Cases

5.1 Finding Exoplanets

As with every software, there are certain use cases for which it was developed. With my software, I had in mind to make the search for new exoplanets more accessible to ambitious amateur astronomers. That is done by making it easier and faster to analyze the data that was gathered overnight. My software goes through all the data and analyzes it with the goal of detecting an exoplanet transit. If it finds such a transit, it will output the star on which the transit was detected, and some basic information about the transit, such as the depth of the dip or the length of the transit.

When detecting an exoplanet transit, we have to record the same piece of sky for at least 3 hours. Otherwise, we can not record the whole transit. If we now try to find an exoplanet, 3 hours will probably not be enough. That is because we do not know when exactly a transit will happen. Optimally, we just observe the same patch of sky for the whole night. If we do that, we will end up taking about one image per minute for around 6 hours straight. No one wants to stay up for 6 hours looking after a telescope and pressing the shutter of a camera once a minute. That is why I strongly recommend an, at least partly, automated setup. If we do that, we can just program a computer to take the images for us. An example for such a setup would be the following:

- a 200 mm reflector telescope (300 USD)
- a motorized equatorial mount (1000 USD)
- a cooled astro-camera (1000 USD)
- a guide-scope and guide-camera (200 USD)
- any computer running Windows 10 and some cables (0 USD, are present in most households)

That sums up to around 2500 USD. While it may be possible to use a DSLR instead of a cooled astro-camera, I recommend a cooled astro-camera because they are very optimized for astronomical observations and produce better data than a DSLR.

In conclusion, I would recommend my software to everyone who wants to find exoplanets using the transit method and has appropriate equipment.

5.2 Other Kinds of Variable Stars

It turns out that, with the methods and ideas used, the software should be able to detect any kinds of variable stars. Only the very last step in my software would have to be modified to return findings on variable stars in general. Variable stars are stars which change in brightness over time. It is a very broad topic, so I will not go into detail. But it is also a topic in which there is a lot of research being done.

6 Working Principle of the Software

The input is a sequence of images of one spot in the sky. Then the brightness of all stars in the sequence will be compared from image to image. This will result in a so-called light curve. There will be one such light curve per star. A light curve shows how the brightness of a star changes over time. To make up for slight clouds, minor sky glow, and other sources of noise that affect the whole frame, we calculate the brightness of a star relative to other stars. Then the resulting light curves will be analyzed to check whether there is a typical dip in the brightness of a star which might indicate an exoplanet transit. In order to implement ExoScanner, I had to solve the following sub-problems:

1. Data-Calibration: Calibrating the data to reduce the noise level.
2. Star Detection: Detecting all stars present in a frame.
3. Alignment: Identifying the same star in two different images.

4. Brightness Calculation: Calculating the absolute brightness of a star.
5. Data Cleansing: Removing bad stars and bad images.
6. Light Curve Generation: Comparing different stars and creating light curves based on relative brightness.
7. Data Analysis: Analyze the light curves and check for possible exoplanet transits.
8. Output: Presenting the results to the user.

7 Overview of the Software

7.1 Data Calibration

In photometry, the concept of calibration is very important. The idea is to subtract sources of noise which are constant over time. To get rid of that noise we use so-called calibration-frames. There are three types of calibration frames: Flat-Frames, Dark-Frames, and Bias-Frames. To get a better idea of what each type of calibration-frame is responsible for, I will list examples below:

- Flat-Frames are taken by having the same amount of light going through the light-train from every angle. Dividing through the resulting frame will get rid of the following artifacts:
  - Vignetting: The border of the image might be darker than the center due to imperfections in the optical equipment.
  - Dust specs: If there is a piece of dust anywhere on the equipment, it is obvious that this will cause certain parts of the image to appear slightly darker.
- Dark-Frames are taken at the same exposure settings as the light-frames. (Light-frames are the actual images of the stars.) No light should be captured in the dark-frames. They will record thermal noise.
- Bias-Frames are taken at the same ISO-value and the shortest possible shutter-time. They will record the read-out noise of the camera.

In this paper, I will not go into detail. For more information, papers like Armin Rests "Calibration of a CCD Camera and Correction of its Images" [8] should be consulted.

Because I did not really have data with these calibration frames, though, I chose to not yet implement the feature of calibration frames. This will, however, be one of the first things that will be implemented when the software is released.

7.2 Star Detection

When starting this project, I first focused on brightness calculation and light curve generation, so my first approach was to just find the stars manually. I opened a frame in the image processing software GIMP to bring the mouse over the stars I wanted and copied the coordinates. But this gets very annoying if more than just a few stars are required. Also, it is not an elegant solution. But for the moment it was good enough.

When finding stars manually got annoying, I had to come up with something better. I sat down, took pen and paper and started thinking. All my ideas had some flaws and so I always had to come up with new ideas to fix these flaws. I will list some of the ideas and insights I have had. Right below them, I will mention the problems the ideas brought with them.

- Stars are bright, so the brightest few pixels will probably be stars.
  - How bright does a pixel have to be in order to be a star? What should that lower threshold be set to?
  - Hot pixels would also be detected as stars.
- Thinking about the threshold, I realized that the background is essentially just a huge sea of noise, while the stars will have a much higher pixel-value. So by applying the sigma-clipping method (Explained in B.1) to find a threshold, one could probably determine which pixels belong to stars. We will call these pixels above the threshold star-pixels.
  - How would it be determined which pixels belong to which star?
  - Again, hot pixels will be a problem.
- Given that we have all the pixels which are significantly brighter than the background-level, these pixels can just be assumed to be stars. By then checking which star-pixels are connected to each other, they could be grouped into single stars. By then taking the weighed average, the position can also be determined.
  - Two stars which are close to each other might be merged into one star.
  - And once more, hot pixels are still a problem.

Because the problem quickly turned out to be much more complicated than I first anticipated, I did some research and found a paper [9] written by Peter B. Stetson in 1987. It describes an algorithm to detect stars. It is optimized to perform well in crowded fields. That means it is able to handle stars which are close together. This is especially useful when analyzing star-clusters. Even though that feature is not really important in this paper, because my algorithms do not handle crowded fields that well, I will use his algorithm because it is implemented in the python-package astropy. It is able to detect all the stars in one frame of about 4000×5000 pixels in about 2 seconds. The algorithm takes an image as input and outputs the coordinates of all the stars it found.

This algorithm also computes the brightness of the star. But after a bit of testing, I had to realize that these calculations were not nearly accurate enough to detect a slight change in brightness like an exoplanet-transit would cause one. That is why I had to come up with my own brightness calculation.

7.3 Alignment

7.3.1 Use of Deep Sky Stacker

For my hobby, astrophotography, I regularly used the software Deep Sky Stacker for stacking the images and producing a less noisy image. Because stacking the images requires the images to be aligned, Deep Sky Stacker was the first thing I thought of when tackling this sub-problem. It turned out, that it is not too hard to get the parameters for the alignment. One can just copy them out of a table. So the first solution was to first let Deep Sky Stacker do the alignment for me and then extract, for each frame, the amounts, by which the image has to be moved in both x and y direction, and the angle by which it has to be rotated. From there, it was just a bit of simple trigonometry to transform coordinates in one image to coordinates in another image. But because the process of first running Deep Sky Stacker every time when some data should be analyzed is very tedious, I had to come up with a better solution.

7.3.2 Implement Algorithm

The documentation of Deep Sky Stacker lists a paper [11] written in 1995 describing an algorithm to match catalogs of astronomical objects. In other words, it is able to take multiple lists of coordinates of stars, one list corresponding to one image, and figure out which of those stars correspond to each other in the different images. It was written for the FOCAS (Faint Object Classification and Analysis System) project. As it names suggests, this project aims to automatically catalog dark objects in our night sky. Because I could not find an implementation, I had to implement the algorithm myself. The paper is based on two older papers written independently by Groth (1986) [3] and Stetson (1989) [10]. When reading through the paper [11], I really enjoyed their ideas. That is why I want to briefly describe the algorithm in the following. For more details the original paper should be consulted.

Problem The algorithm is designed to match two catalogs of astronomical objects to each other. It should work if there was a translation, rotation, rescaling, or reflection applied. This is a so-called pattern-matching problem.

Finding Triangles The approach of the algorithm is very similar to the approach humans take when they try to identify stars in two different images. Intuitively, one looks for obvious shapes and tries to locate them in the second picture. What the algorithm does is to first select the 20 brightest stars from both images. We assume that most of these 20 stars are the same in both images. Next, the algorithm generates all the possible triangles using the 20 brightest stars as vertices. We do this for both catalogs. This will result in two sets of $\binom{20}{3} = 1140$ triangles each. We will call these sets T and T' for the rest of this section. All of the triangles from T and T' will now be put into a two-dimensional *triangle space* (x, y), where

$$x := \frac{b}{a}, \; y := \frac{c}{a}$$

with a, b, c being the lengths of the sides of the triangles in decreasing order. The great property of the *triangle space* is, that it gives any two triangles with the same shape the same coordinates, regardless of their size, rotation, position, or reflection. For an illustration and examples of this *triangle space* see figure 4. It can be seen that all triangles will end up within the marked triangle: Us sorting the sides by their lengths results in $y = \frac{c}{a} \leq \frac{b}{a} = x$. And the fact, that the sum of the shorter two sides has to be greater than the length of the longest side, results in $\frac{b}{a} + \frac{c}{a} = x + y \geq 1 \Leftrightarrow y \geq 1 - x$.

If the shortest edge is very short, and thus one angle is very small, the triangle is subceptible to be mismatched. To prevent these mismatches, the original authors did not include any triangles where the sides a and b were of similar length. More precisely, they removed all triangles where $\frac{b}{a} = x > 0.9$. In figure 4 that is the area which is dotted in red. When reading through their paper, that restriction stuck out to me and I tried to understand it. My first thought was why they would not restrict the ratio between a and c. That would make more sense because the problematic triangles are the ones with a very short shortest edge, not the ones with sides of similar length. After some confusion and a lot of thinking, I came to the conclusion that the authors of the original paper made a mistake. With that mistake they would cut out some close to equilateral triangles which are some of the easiest ones to match. They will not produce problems with mismatches. I fixed that in my implementation of the algorithm by removing all triangles with $\frac{c}{a} = y < 0.1$ instead of removing the triangles with $\frac{b}{a} = x > 0.9$. In figure 4, this is the area which is hatched in blue. While this is only a small improvement, it is a fixed bug nevertheless.

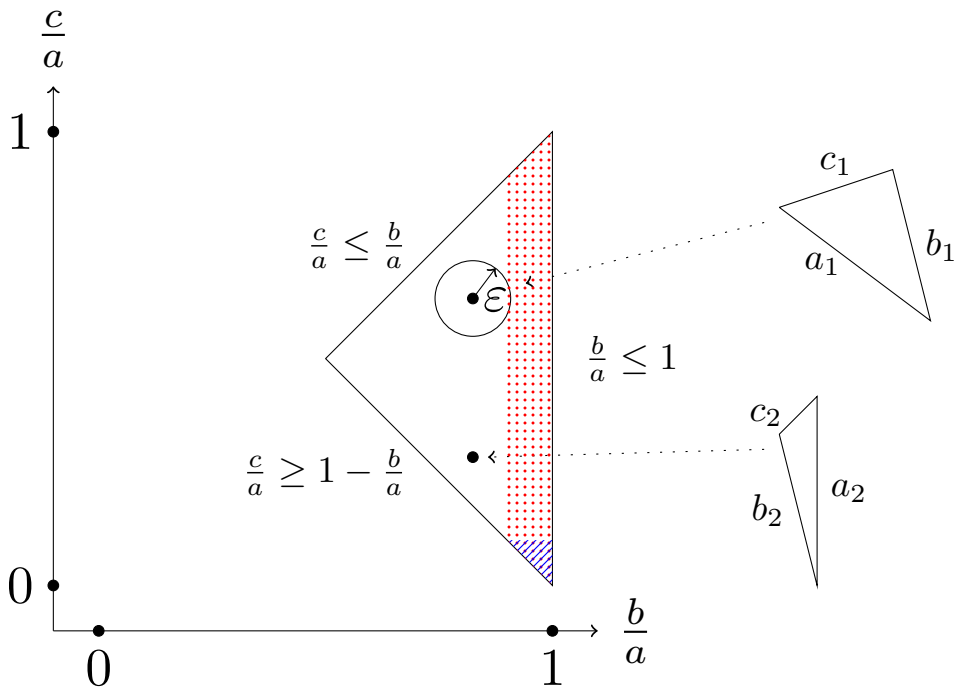


Figure 4: Graphic examples for the triangle-space.
The first triangle has $x = \frac{b_1}{a_1} \approx 0.83$ and $y = \frac{c_1}{a_1} \approx 0.63$.
The second triangle has $x = \frac{b_2}{a_2} \approx 0.8$ and $y = \frac{c_2}{a_2} \approx 0.28$.

Matching Triangles Our next aim is to match the triangles from T and T', as defined in Finding Triangles. This means that we will identify, for a given triangle $T_i \in T$, which triangle $T'_j \in T'$ has the same shape and are thus is probably made of the same objects. Intuitively, we check the Euclidean distance between the two triangles T_i, T'_j in *triangle space*. If it is smaller than a given ε, we hope that T'_j is made of the same objects as T_i and call T'_j *matched* to T_i. We empirically set ε to 0.02. Note that it is possible for multiple triangles from T' to be matched to one triangle from T. This ambiguity will be solved in the later steps, see Matching Stars and Rejecting Mismatches and Computing Transformation.

The trivial implementation approach would be to go through all triangles $T_i \in T$ and compare them to the 1140 triangles in T'. But like that, we would end up with over 1,000,000 comparisons. Because we want our program to run in reasonable time, we have to come up with some optimization: we first sort the triangles in T' by their x-coordinate in the *triangle space*. When now searching for matches for a selected triangle $T_i \in T$, we only have to look at a range of triangles $T'_j \in T'$ with $j \in [l, r]$; l, r to be defined.

Let x_i and x'_j be the x-coordinates of T_i and T'_j in the *triangle space*. We define l such that $x_i - x'_l \leq \varepsilon$ and $x_i - x'_{l-1} > \varepsilon$. Similarly, we define r such that $x'_r - x_i \leq \varepsilon$ and $x'_{r+1} - x_i > \varepsilon$. Only the triangles within this range could have a distance of $\leq \varepsilon$. It is obvious that there will not be that many triangles in the range and thus our program will run a lot faster with this optimization. The only question is how to find $[l, r]$ fast. However, this can easily be solved with a method called binary search. An explanation of binary search can be found in B.2 Binary Search.

Matching Stars After matching the triangles, we have to match the individual stars as well. A *match* consists of two triangles $T_i \in T$ and $T'_j \in T'$. Let us call the stars which T_i is made of $\{\alpha, \beta, \gamma\}$, while the stars T'_j is made of will be called $\{\alpha', \beta', \gamma'\}$. We do not know which of those do actually correspond to each other, we just know that they are the same set. But because we have many triangles, we are able to figure out which stars correspond to each other.

We create a two-dimensional array v which is used to "vote" on which stars are the same. The first dimension is the 20 brightest stars in the first frame, the second dimension is the 20 brightest stars in the second frame. Then, for all *matches*, we increase the value in v in the following nine positions by one:

$$\begin{array}{ccc} v_{\alpha,\alpha'}, & v_{\alpha,\beta'}, & v_{\alpha,\gamma'}, \\ v_{\beta,\alpha'}, & v_{\beta,\beta'}, & v_{\beta,\gamma'}, \\ v_{\gamma,\alpha'}, & v_{\gamma,\beta'}, & v_{\gamma,\gamma'} \end{array}$$

After doing that for all matched triangles, the entries with the highest counts indicate a possible match of stars. Because we will remove mismatches later on (see Rejecting Mismatches and Computing Transformation) and because this algorithm should be as lightweight as possible, we will just take the 20 highest scores and assume them to be correctly matched stars.

Rejecting Mismatches and Computing Transformation We will calculate the coordinate-transformation between the two images iteratively. Like that we can get a better accuracy. In each iteration, we will reject mismatches. In the first iteration, we will only use the 6 matches with the most votes because these are the matches we are the most sure about. Otherwise, the chances are big, that there would be too many mismatches messing up our coordinate-transformation. The coordinate-transformation between the two catalogs will now be estimated using the least-squares method. After that, we have to figure out which matches are wrong. Because a mismatch will be far off its matched star, while correct matches will be very close to their matched stars, we can set sigma to be the 60th percentile of the squared errors. Every match with an error of more than two sigmas will be rejected. After this rejection phase, the next iteration will begin. This method is called iterative sigma-clipping. For more detail, see B.1 Sigma-Clipping Method. As soon as there are no matches being rejected, we return the coordinate-transformation.

The remaining stars, so all the stars which were not included in the 20 brightest, will be matched by position after applying the coordinate-transformation. That means that the closest star will be matched, given it is not farther away than 3 pixels.

7.4 Brightness Calculation

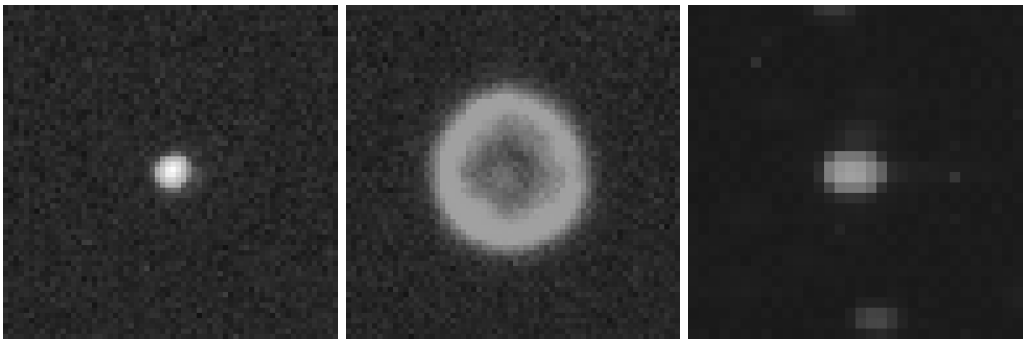
One could argue that this is the most important part of the software. It is extremely important that we are able to measure the brightness of a star with high precision. The dips in brightness we are talking about when observing exoplanet transits are extremely small. The deepest dips only change the brightness of the star about 1 to 2 percent. This means, that optimally, we can determine the brightness of a star with less than 1 percent error. While the precision of the data points also depends on the quality of the equipment, the best raw data does not help a bit, if the software processing it, introduces a bunch of noise.

When calculating the brightness of a star, there are many factors, which we have to account for. In the following, I will list some of them:

- The stars are so far away, that reasonable equipment can not resolve them. That means that, in theory, they should appear on just one pixel on the camera. But they do not.

They will look more like a circle. That is because the light will be distorted by the atmosphere. Some distortion also comes from the not quite perfect optical properties of the telescope. The resulting shape can be described by a point-spread-function. It describes how one point of light is distorted and thus how it will appear on the camera-sensor. The normal-distribution is a fairly good first approximation for the point-spread-function of stars captured through the atmosphere with a telescope. While this approximation works quite well on well focused, round stars (figure 5a), it has troubles as soon as the stars are elongated (figure 5c). That might happen because of breezes hitting the telescope and thus slightly moving it. Another common source are tracking errors in the mount. What causes problems too, is if the stars are not well focused (figure 5b). That can easily be prevented by carefully setting up the telescope. A fix for these problems would be custom point-spread-functions.

- The image will have a certain background-level due to things like light pollution. It is beneficial to remove this background-level as it does not stay constant over time and would thus mess up our light curves.
- There will be certain situations which will dim all the stars in the frame equally. For example, the amount of atmosphere that the light of the star has to pass through. To get rid of that effect, one compares the brightness of the observed stars to other stars which are nearby and have a similar brightness. To prevent the introduction of more noise into the already noisy data of the observed star one takes many stars to compare it with. The comparison will be done with division. That is due to the fact that the atmosphere "swallows" a fraction of the light and not an absolute value. This factor will be accounted for in section 7.6 Light Curve Generation.



(a) perfect star (b) poorly focused star (c) elongated star

Figure 5: Problems a point-spread-function accounts for

I improved the precision of my brightness calculation with a constant cycle of writing new code, testing the new program, coming up with improvements, and then repeating. My final algorithm for calculating the brightness is described below:

The first observation is that it is not required to analyze one whole image to get the brightness of a star. It is enough to just analyze a small box of pixels around the target star. So the first thing we do is to construct boxes of 15×15 pixels around the stars.

These boxes are then passed on to the next function, one at a time. As mentioned above, the background level should be removed. When estimating the background level of a whole frame, the median is quite a good approximation, because stars are only present on very few pixels. In

the case of a small box, though, there are, relatively speaking, a lot more pixels which captured starlight, than in a whole frame. That is why the median does not work that well anymore. But if we think about it, the median is just the 50th percentile. So what we do now, is to just take the 25th percentile and assume it to be the background level of the box. While this is a purely empirical approach, it works surprisingly well.

The next step is to calculate the brightness of the star. As mentioned, a fitted normal-distribution is a fairly good approximation for the point-spread-function. When testing the method of fitting a normal-distribution on the box, I noticed that it was rather slow. Because this is a part of the software which is executed very often, I had to do some optimizations. So instead of fitting a normal-distribution using the least-squares method, I calculated the median μ and the standard deviation σ of the observed data. With this simplification, the normal-distribution can now be quickly approximated.

But how do we get from a fitted normal-distribution to a number which tells us the absolute brightness of a star? One idea was to integrate the normal-distribution. But because that integral is the exact same as the sum of the pixels, that was the easier option. Because it does not make sense to sum up pixels which only add in noise to the data, I decided to sum up the pixels which are closer to the median μ than 2.5 times the standard deviation σ. As discussed in 11.1, we can see that the result of this approach is not far off from more involved algorithms.

7.5 Data Cleansing

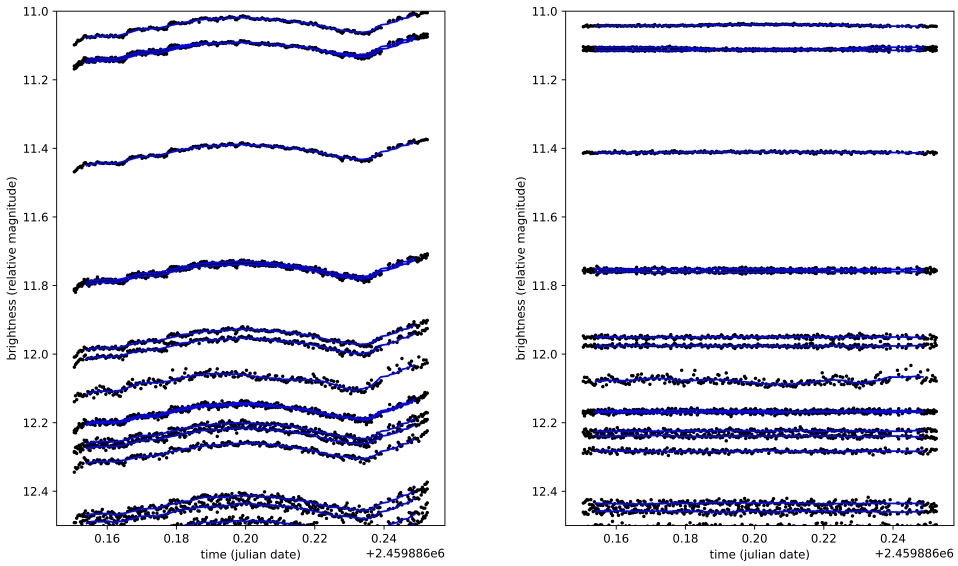
Now that we calculated the brightness of all stars in all images, we have to clean up the acquired data. The input is a huge table, with one axis being the star-number and the other being the frame-number. The entries will be the brightness of a certain star in a certain frame. But there might be gaps in this table. Because such gaps would mess up algorithms further down the line, we have to remove them. For any gap we have two choices. We can either remove the star from our table, or remove the image from our table. The question is, when it is better to do what. Intuitively, if there are many stars missing in one frame, it is likely that there is a problem with the frame. Examples would be clouds or slight tracking problems. In that case, we would want to remove the image. On the other hand, it is possible that some very faint stars are not being picked up in some images because they are just too faint. Here, we would want to remove the star.

For any gap, we have a position. The position is composed of a frame number and a star-number. Now we calculate two numbers. We calculate how many stars are not detected in the current image. We will call this number the *image score*. Then we calculate in how many images the current star is missing. This number will be called the *star score*. These two numbers will then both be converted into percentages. Note that a lower score is better. Now, if the *star score* $\times 2$ is more than the *image score*, the star will be removed. Otherwise, the image will be removed. The factor $\times 2$ is there because we value images more than stars. It is better to loose a faint star rather than to loose a data point for every star.

7.6 Light Curve Generation

Now that we can calculate the brightness of a star, we can start thinking about the change of the brightness of this star over time. As already mentioned earlier, there might be some sources of noise which affect all the stars equally. We compare the stars in question with other so-called reference stars. Like that, we can get rid of noise sources like light pollution or the thickness of the atmosphere. We calculate the brightness as a ratio between a star and its reference stars. The importance of this comparison with reference stars can be seen in figure 6. Because we do

not know which stars host an exoplanet which can be observed with the transit method, we have to take enough reference stars, such that if there was a transit in one of the reference stars, the whole reference would not shift too much.



(a) light curves of the brightest few stars in the frame, without using reference stars

(b) light curves of the brightest few stars in the frame, using reference stars

Figure 6: importance of reference stars

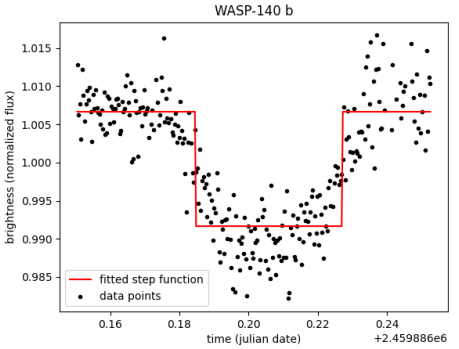
A possible improvement could be to search for reference stars which are known to be of constant brightness. But that would be quite hard because we would have to identify the stars in the frame and compare them with big databases. For this project, I decided to stick with the method of choosing enough reference stars that the effect of a transit in one of them would be small enough.

7.7 Data Analysis

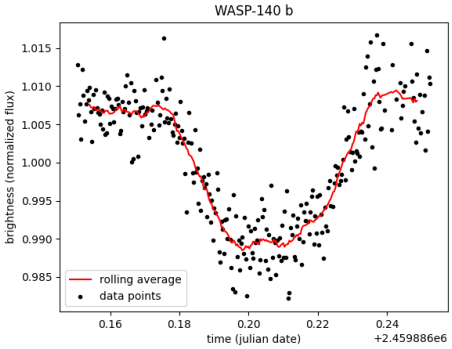
Once all the light curves are generated, they also have to be analyzed. That means that it should be checked whether the curve contains a dip in brightness which might be caused by a transiting exoplanet. While there are some mathematical models describing the dip caused by an exoplanet-transit, it was too involved for my software. An easier idea is to just idealize an exoplanet-transit as a reduction in brightness which starts in an instant and then also ends just as abruptly. Such a dip could then be described by a step-function. It starts off being at the normal brightness of the star. As soon as the exoplanet is in front of the star, it will drop to a lower brightness. Then, when the transit is over, the brightness goes back to normal. When comparing the rolling average of the data (Figure 7b) or the mathematical fit, generated with the model-fit feature in the Exoplanet Transit Database [7], (Figure 7c) with the fitted step function (Figure 7a) it can be seen that the step function is a reasonable approximation.

The described step-function is fitted by brute-force trying out all positions to start and end a transit. Then it will set the normal brightness value and the reduced brightness value to the median of all the corresponding data. These curves are then rated by calculating the average squared-error between them and the actual data points. The best of these options will then be returned.

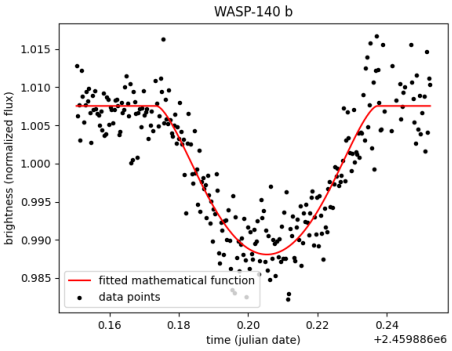
It is a rather slow approach as it calculates the quality of $\frac{n^2}{2}$ curves. But because there was no time to come up with a better algorithm and because it was sufficiently fast after parallelizing the process, I decided to just keep it.



(a) Transit of WASP-140 b with fitted step-function



(b) Transit of WASP-140 b with rolling average



(c) Transit of WASP-140 b with fitted mathematical curve

Figure 7: Comparison of fitted step-function, fitted mathematical-function, and rolling average.

7.8 Output

Now that we fitted a step-curve over all light curves, we have to determine which of the light curves are actually exoplanet-transits. There is essentially two pieces of information we have about any light curve. We know how deep the dip is according to our fitted step function, and we know how well the step function describes the light curve. The software looks at those two parameters and calculates a score for each star by dividing the depth of the dip by the average square error of the fitted curve. If a star has a very high score, it is likely to be an exoplanet.

Now all these candidates will be sorted by their score. By default, there will be graphs generated for the top 10 candidates. They will be written as files into a folder. The graphs will contain the data points, a rolling average and the fitted step-curve. Each graph also has a short description. It denotes the score of a candidate, the depth of the dip and the coordinates of the star in the first frame.

This allows the user to just quickly look at the top few candidates and check whether they might be a transit.

Figure 8 shows how the output might look like. One can quickly look over them and check manually whether a transit might have occurred.

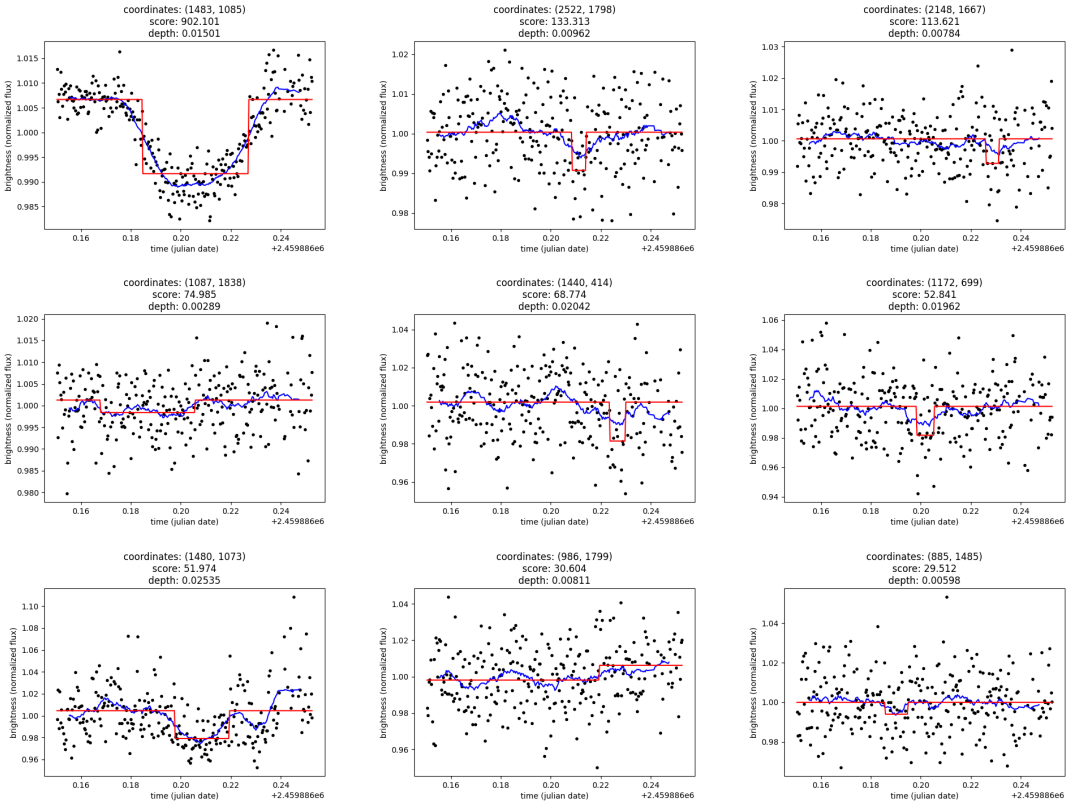


Figure 8: example for an output ExoScanner might generate

8 Implementation Details

Now that we have solved all of these sub-problems, we have to find a way of combining them into one software. In this section, I will discuss which sub-problems correspond to which files. I will also discuss how the functions in the files interact with each other.

When running ExoScanner main.py is called. It parses the input parameters and passes them on to run.py. That is the file which combines all the other files and runs the functions in the right order. It starts by finding all the image files in the given folder. This is done in getFilelist.py. It is also obvious that there has to be some function reading the files. A file is read

and a two-dimensional array of pixel values is returned. This is what readImage.py is responsible for. Note that currently only FITS files are supported. While there is a feature for canon raw files within readImage.py, other features of the software currently do not allow canon raw files. Alongside, there is a file called getTimeOfObservation.py which reads the header of an image file and returns the time and date it was taken on.

myAlgorithms.py is just implementing some algorithms which are used throughout the program. An example is an implementation of the rolling average.

Once all the files have been found, we can start detecting stars in the images. That is done in generateCatalogs.py. It takes the filenames as an input and returns a list of tables. Each table contains all the stars found in one image. The generation of the catalogs is parallelized, speeding it up quite a bit. The actual finding of the stars is done by the algorithm DAOPHOT, implemented in the photutils python library.

Now that the stars have been found, it is time to figure out which stars correspond to which in the different images. That is done by mergeCatalogs.py. Because there is no implementation of this algorithm, I implemented it myself. It takes the list of tables generated by generateCatalogs.py as input and returns a list of lists. The jth item of the ith list tells us where to find the jth star in the catalog of the ith image.

Using this information, and all the catalogs generated before, we can start calculating the brightness of all the stars. There are two files generateField.py and getBrightnessOfOneStarInField.py which, when combined, are able to find the brightness of a star, given an image and its position in the image. This process is repeated for all stars in all images in generateBrightnessOfAllStarsInAllImages.py, resulting in a table. This file is also responsible for cleaning up the data in the table afterwards. To improve the runtime, I also parallelized this process.

Then we have to generate the light curves. This is done in generateLightCurve.py. It just compares the calculated brightness values to give a light curve for every star.

Finally, we need to analyze all of the light curves and check whether they look like a typical transit-curve. This is done in the file analyzeLightCurves.py. It takes a light curve as input and returns a fitted step curve, along with a score, telling us about the probability of the curve describing a transit. While the current algorithm is rather slow, parallelizing the process made it run fast enough. Then, the output is generated by output.py.

9 Using the Software

Because writing a fancy user-interface is a big hustle and because it will not make the program work better, I opted for writing ExoScanner as a CLI-tool. That means that it will only run as a command through the terminal on the computer and that there is no user interface. When running the program, a path to a folder containing all the image-files has to be provided. The folder should not contain anything other than image-files. The format of these files has to be FITS. It is important, that, if the files are sorted by name, the files should also sorted by observation time. The first file should have a reasonable quality. If it does not, some stars may be lost.

The program is published on GitHub under https://github.com/josia-john/ExoScanner. I hope to be able to maintain the project and release some updates in the future. ExoScanner requires python to be installed. Then, the program can be easily installed using pip:

```
pip install https://github.com/josia-john/ExoScanner/archive/main.zip
```

After that, ExoScanner can be run using the following command:

```
python -m ExoScanner <path>
```

with `<path>` being the path to the folder with all the images.

Behind this link, there is a sample dataset, with which the software can be tested locally: https://drive.google.com/drive/folders/13AZ1xhNR8qf8G5aFWLcC5ObRZvBCNXpz?usp=sharing There is no guarantee for how long the link will still work.

10 Data

Because data is a very big part of this project, I will quickly list the most important data sources I have worked with.

- Matt Baker: transit of HAT-P-3 b
  - Equipment:
    Telescope: 0.09 meter diameter refractor
    Camera: ZWO ASI 1600MM (cooled astro-camera) Pro Camera
    Mount: Skywatcher HEQ5 Pro
  - Acquisition: 113 exposures, 90 seconds each
  - Size: 4656×3520 pixels, 3.7 GB
- Stellarium Gornergrat: WASP-52 b
  - Telescope: 0.6 meter diameter reflector
  - Acquisition: 100 exposures, 90 seconds each
  - Size: 4096×4096 pixels 3.3 GB
- Siding Spring Observatory (published by Las Cumbres Observatory [6]): WASP-140 b
  - Telescope: 0.4 meter diameter reflector
  - Acquisition: 294 exposures, 17 seconds each
  - Size: 3136×2112 pixels, 3.9 GB

I wanted to capture some data with my 0.2 meter diameter reflector, but sadly the weather was not allowing it. That is one reason I chose to write this program rather than doing astrophotography.

11 Results

Now, that the algorithms have been described, the big question is how the software actually performs.

11.1 Comparison with AstroImageJ, a Professional Photometry Software

As mentioned in the very beginning of this paper, AstroImageJ is one of the leading softwares in photometry. So why not compare its results with the ones my program generated. I gave both my program and AstroImageJ the same dataset. Otherwise, the comparison would make no sense. The used dataset is the dataset from the Siding Spring Observatory of the transit WASP-140 b.

11.1.1 Processing Time

When running both AstroImageJ and my own software, I compared the computation-times of both. On my machine (MacBook Pro, 16gb ram, intel i5 2.4 GHz 4 cores) both the programs ran in just over three minutes, so their computation-time is very similar.

11.1.2 Results

As we can see in figure 9, where both results are plotted, the two programs do very much agree with each other. The two generated curves lie exactly on top of each other, the dips have the same depth, and the dips start and end at the time. Looking very closely, one can see that my program generates output which is a tiny bit more noisy.

11.1.3 Features

While AstroImageJ was only able to compute the light curve for one star that had to be manually selected, my software computed a light curve for every star in the frame. But that is not all my program did. It also analyzed the light curves and returned them sorted by their score. The score says how likely they are to describe an exoplanet transit. With the chosen dataset, my program could identify the correct light curve without any problems.

To be fair, I have to mention that AstroImageJ has many features my software does not have. But for the described use case, those features are not really important.

11.1.4 Conclusion

Considering that I only had a very limited amount of time to write and improve my software, while AstroImageJ was developed over multiple years by multiple professors, I consider this result as a big success.

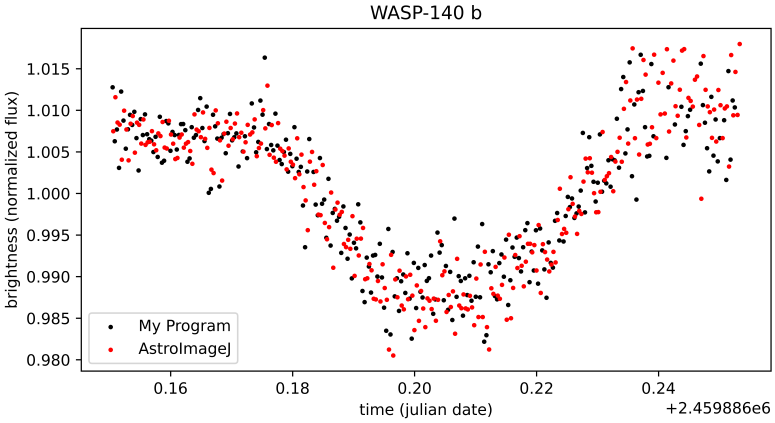


Figure 9: Comparison of the results returned by ExoScanner and AstroImageJ.

11.2 More Examples

11.2.1 First Dataset, Matt Baker

It is obvious that the first dataset I worked with will be shown first. When starting out with this project, I needed some data to be able to test parts of my software and optimize the used algorithms. This first data-set was provided by Matt Baker [1], or "mattsastro", his reddit-username. I saw his post about exoplanets and so I figured that I can just ask him whether he would let me use his raw-data.

As mentioned in section 10, he used a 0.09 meter diameter refractor telescope with a cooled astrophotography camera. Those were mounted on a motorized telescope mount. The results my program extracted from his data are shown in figure 10.

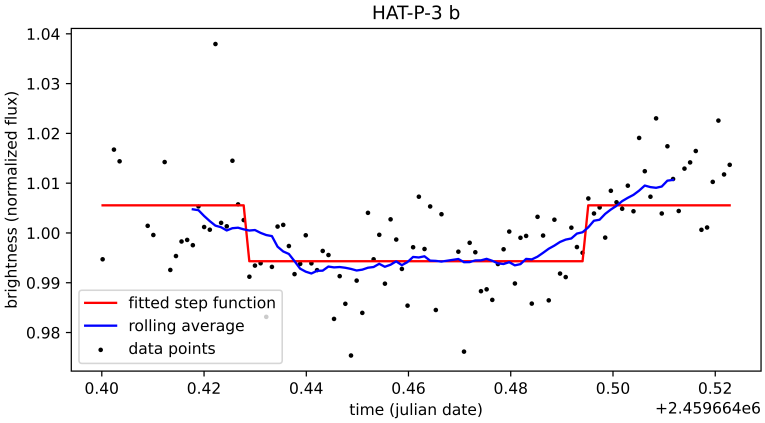


Figure 10: Matt Baker's data, analyzed with my program; HAT-P-3 b

The actual exoplanet-transit is listed as the 6th likeliest option. We can see that there is quite a lot of noise in his data. Yet, I am really happy to be able to report that my software can analyze amateur-data reasonably well. The software fitted the step-curve correctly and was able to give the transit a high score. While there were some mismatches, it is no problem to go through the first few results to check which light curves look like they might contain an exoplanet transit.

11.2.2 Gornergrat Observatory

After having worked with Matt's dataset for quite some time, I wanted to test ExoScanner on some more data. So I set out to find some more datasets. It turns out that there is a telescope in Zermatt, the Stellarium Gornergrat. Their instruments do not fulfill the latest requirements of science anymore. That is why they are making their instruments available to the public. Luckily for me, I am not the only one who came up with the idea of writing a Matura paper about exoplanets. Because of that, they already had a small archive of raw-data of exoplanet-transits. I asked them whether they could share it with me and they agreed. Their data was shot on a telescope 0.6 meters in diameter. That means that they can gather about 40 times more photons compared with Matt Baker's 0.09 meter telescope. Looking at the result in figure 11 we see that there is much less noise. The step-function describes the transit quite well.

But their data is not quite perfect. The correct curve is only listed as the second likeliest candidate, with some close follow-ups. In the following I want to further explain what caused

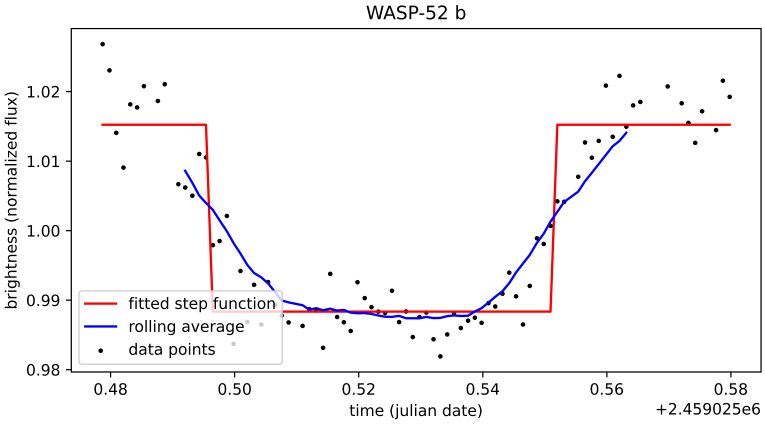
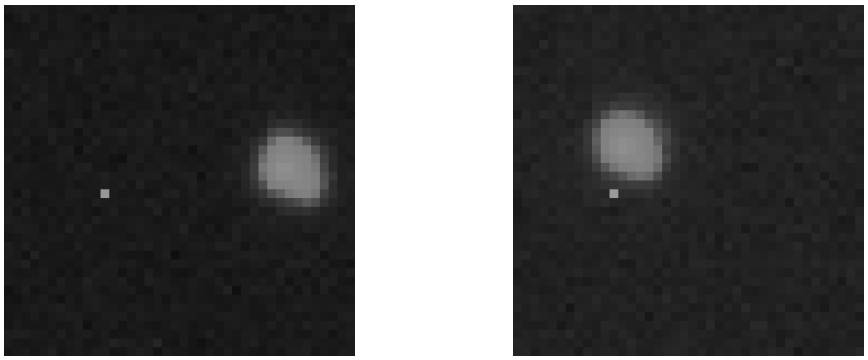


Figure 11: Data from the Gornergrat Observatory, analyzed with my program; WASP-52 b

these false positives. But I can already say that, with some more work on the software, it would not be too hard to reduce the amount of false positives significantly.

False Positives: Hot Pixels In the case of the data from the Gornergrat Observatory, the highest scoring star is a false-positive. In this case, the false positive is caused by a hot-pixel. That is a pixel with a value much higher than it should. These hot-pixels are small defects present in every camera. In figure 12a we can see a star rather close to a hot-pixel. If we compare that with figure 12b we see that the star further approached the hot-pixel. Now, the hot pixel is very close to the star, which makes the hot-pixel count towards the brightness of the star, letting my program think it is a lot brighter than it actually is. This drastic change is then reflected in figure 13, where the brightness goes up by a crazy amount in the last few data points.

As mentioned in the section about calibration frames, hot pixels can be removed by using calibration frames. Since the time did not yet allow for this feature to be implemented, false-positives caused by hot pixels may occur from time to time.



(a) The star is approaching the hot pixel, not yet problematic

(b) The star is too close to the hot pixel, hot pixel is counted towards the brightness

Figure 12: Problem of hot pixels shown in the raw data

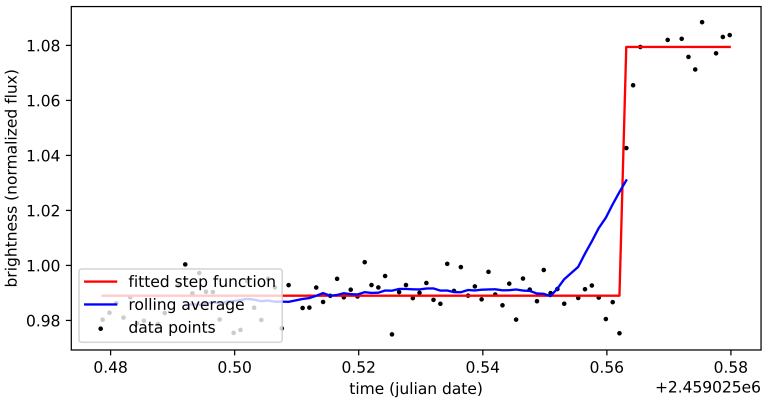


Figure 13: A false positive, caused by a hot pixel

11.2.3 Siding Spring Observatory

When discussing ExoScanner online, the LCO science archive [6] was mentioned. From there, I chose a dataset of WASP-140 b transiting. It was recorded by the Siding Spring Observatory. This is the dataset used in 11.1 to discuss ExoScanner's precision. ExoScanner had no troubles to identify the correct star when analyzing this data. When comparing the results using their dataset in figure 14 with the results using the data from the Gornergrat Observatory in figure 11, one notices that the Siding Spring Observatory has more samples. That means that they recorded more images. But that also means that their exposure time had to be shorter than the exposure time used at the Gornergrat Observatory. That is why their individual data points are a bit noisier. Another reason for their slightly noisier data is that their telescope is 0.4 meters in diameter, quite a bit smaller than the 0.6 meters in the Gornergrat Observatory. That causes them to detect only half of the amount of photons than what the Gornergrat Observatory can. What plays for the Siding Spring Observatory, though, is that they have a lot less light-pollution than the Gornergrat Observatory.

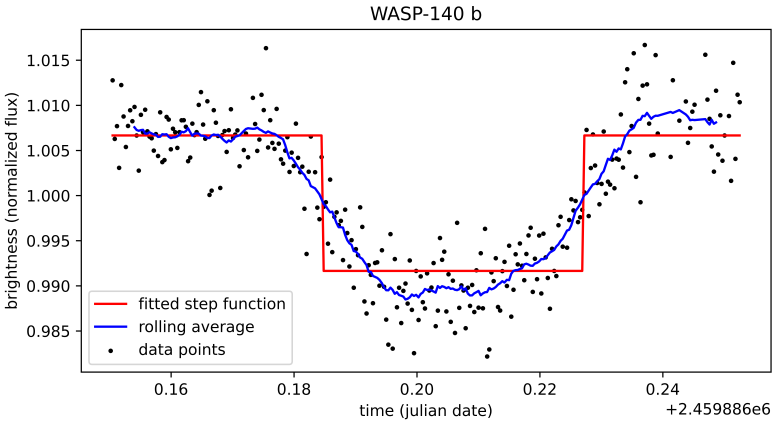


Figure 14: Data from Siding Spring Observatory, analyzed with my program; WASP-140 b

A Thanks

First, I want to thank Thomas Graf for supervising my project. He helped me in a lot of ways. Most of his advice was about the formality of this paper and I really appreciate the time and effort he put in to help me make this paper be a success.

Next, I want to thank Matt Baker [1], or "mattsastro", his reddit-username, for providing me with my first data-set going into this project. It was very helpful to have some data to test on. At the same time it was very inspiring to see that, with amateur equipment, it is possible to detect exoplanets.

Then the Stellarium Gornergrat should not be forgotten. They supplied me with the next batch of data. They even let me have access to their sorted exoplanet transit observation archive. It was very nice to be able to download the data from an intuitive cloud-service.

The third party providing data was the Las Cumbres Observatory. This work makes use of observations from the Las Cumbres Observatory global telescope network. I want to thank them for publishing some of their data.

And then, last but not least, I want to thank my family. My mother and sister for supporting me and listening to me talk about my constant stream of problems, not quite understanding my world, but being there nevertheless. My father for teaching me how to program back when I was a kid and for now helping me come up with creative ideas and helping me fix my problems with LaTeX.

B Glossary

B.1 Sigma-Clipping Method

Sigma-Clipping is a method of rejecting outliers. We select a σ, often the standard-deviation. Additionaly we select some constant a. The median of our dataset should be m. Now all the datapoints x for which $|x - m| > a \cdot \sigma$ are ignored.

B.2 Binary Search

Given an array a with length n such that $a_i = 0$ for any $i < x$ and $a_i = 1$ for any $i \geq x$ with some $x \in [0, n-1]$, binary search can be used to find x. If we check the array at some position a_m and the result is 1, we know that $x < m$. Similarly, if we get $a_m = 0$, we know that $x \geq m$.

In the beginning, we know that $l \leq x \leq r - 1$ with $l = 0$ and $r = n - 1$. If we now set $m = \lfloor \frac{l+r}{2} \rfloor$ and check a_m, depending on the result, we can set $l = m$ or $r = m - 1$. This means that the range x could be in has approximately halved in length. That is why this has to be done only $\log_2(n)$ times in order to find the exact x.

C References

[1] Matt Baker. *Matt's Astro*. 2021. URL: https://www.mattsastro.co.uk/ (visited on 2022-11-23).

[2] A. Lecavelier des Etangs and Jack J. Lissauer. "The IAU working definition of an exoplanet". In: *New Astronomy Reviews* 94 (June 2022), p. 101641. DOI: 10.1016/j.newar.2022.101641. URL: https://doi.org/10.1016%2Fj.newar.2022.101641.

[3] Edward J Groth. "A pattern-matching algorithm for two-dimensional coordinate lists". In: *The astronomical journal* 91 (1986), pp. 1244–1248.

[4] IAU. *Working Group on Extrasolar Planets - Defintion of a "Planet"*. 2003. URL: https://w.astro.berkeley.edu/~basri/defineplanet/IAU-WGExSP.htm (visited on 2022-11-23).

[5] Christian Marois et al. "Direct Imaging of Multiple Planets Orbiting the Star HR 8799". In: *Science* 322.5906 (Nov. 2008), pp. 1348–1352. DOI: 10.1126/science.1166585. URL: https://doi.org/10.1126%2Fscience.1166585.

[6] Las Cumbres Observatory. *LCO Archive*. 2020. URL: https://lco.global/documentation/archive-documentation/#overview (visited on 2022-12-09).

[7] Stanislav Poddaný, Luboš Brát, and Ondřej Pejcha. "Exoplanet Transit Database. Reduction and processing of the photometric data of exoplanet transits". In: *New Astronomy* 15.3 (Mar. 2010), pp. 297–301. DOI: 10.1016/j.newast.2009.09.001. URL: https://doi.org/10.1016%2Fj.newast.2009.09.001.

[8] Armin Rest. "Calibration of a CCD Camera and Correction of its Images". In: *Dissertations and Theses* (1996). DOI: 10.15760/etd.7062.

[9] Peter B. Stetson. "DAOPHOT: A Computer Program for Crowded-Field Stellar Photometry". In: *Publications of the ASP* 99 (Mar. 1987), p. 191. DOI: 10.1086/131977.

[10] Peter B. Stetson. "Image and Data Processing/Interstellar Dust". In: *V Advanced School of Astrophysics* (1989).

[11] Francisco G. Valdes et al. "FOCAS Automatic Catalog Matching Algorithms". In: *Publications of the ASP* 107 (Nov. 1995), p. 1119. DOI: 10.1086/133667.

D Code

D.1 run.py

```
# The function run() calls all the functions needed in the correct order and
# combines their returned results.

from getFilelist import getFilelist
from generateBrightnessOfAllStarsInAllImages import
↪ generateBrightnessOfAllStarsInAllImages
from generateBrightnessOfAllStarsInAllImages import cleanUpData
from generateCatalogs import generateCatalogs
from mergeCatalogs import mergeCatalogs
from analyzeLightCurves import analyzeLightCurves
from getTimeOfObservation import getTimeOfObservation
from generateLightCurves import generateLightCurves
from output import output

from astropy.time import Time


def run(pathToLights):
    files = getFilelist(pathToLights)   # get all files

    catalogs, files = generateCatalogs(files)   # get catalogs and ignore files with
    ↪ less than 20 stars

    brightness = generateBrightnessOfAllStarsInAllImages(files, catalogs,
    ↪ mergeCatalogs(catalogs))  # get brightness
    brightness, axis, stars = cleanUpData(brightness)   # remove bad images and bad
    ↪ stars

    lightCurves = generateLightCurves(brightness)   # get Lightcurves

    analysis = analyzeLightCurves(lightCurves)

    times = []  # get observation-times for the included images
    for i in axis:
        times.append(getTimeOfObservation(files[i]))
    times = Time(times, format='isot', scale='utc').jd

    for i in range(len(analysis)):  # add index and coordinates in the first image to
    ↪ the analysis of each star
        analysis[i]["index"] = i
        analysis[i]["coordinates"] =
        ↪ (round(catalogs[0]["xcentroid"][stars[i]]),round(catalogs[0]["ycentroid"][stars[i]]))

    output(lightCurves, times, analysis)    # generate output
```

D.2 generateCatalogs.py

```
# This file is responsible for generating a catalog for each image in the
# sequence, each catalog containing all the stars in the image. This is done
```

```
# using the algorithm "DAOPHOT / DAOFIND"

from astropy.stats import sigma_clipped_stats
from photutils.detection import DAOStarFinder
from readImage import readImage
import numpy as np
from multiprocessing import Pool


def generateCatalogForOneImage(file):
    data = readImage(file)
    border = int(((len(data) + len(data[0]))/2) * 0.1)
    mean, median, std = sigma_clipped_stats(data, sigma=3.0)
    box = np.ones([len(data), len(data[0])], dtype=bool)
    box[border:len(data)-border, border:len(data[0])-border] = False
    daofind = DAOStarFinder(fwhm=4.0, threshold=20*std)
    sources = daofind.find_stars(data-median, mask=box)

    return sources


def generateCatalogs(files):
    with Pool() as mp_pool:
        res = mp_pool.map(generateCatalogForOneImage, files)

    newRes=[]
    newFiles=[]

    for i in range(len(files)):
        if len(res[i]) < 20: continue
        newRes.append(res[i])
        newFiles.append(files[i])

    return newRes, newFiles
```

D.3 mergeCatalogs.py

```
# This file implements the algorithm described in
# "FOCAS Automatic Catalog Matching Algorithms".
# It finds which stars are the same in different catalogs.

import math
import numpy as np
from queue import PriorityQueue

from matplotlib import pyplot as plt


def convertToTriangleSpace(sides):
    sides.sort()
    return (sides[1]/sides[2], sides[0]/sides[2])


```

```
def selectBrightest(catalog, N):
    brightestElements = PriorityQueue()
    for i in range(len(catalog)):
        smallest = -1
        if (not brightestElements.empty()): smallest = brightestElements.get()
        if (brightestElements.qsize() < N-1 or smallest < (catalog[i]["flux"], i)):
            brightestElements.put((catalog[i]["flux"], i))
        if (brightestElements.qsize() < N and smallest != -1):
            brightestElements.put(smallest)

    res = []
    while not brightestElements.empty():
        res.append(brightestElements.get()[1])

    return res


def getDistances(points):
    distances = []
    for i in range(0, len(points)):
        distances.append([])
        for j in range(0, len(points)):
            distances[-1].append(math.sqrt((points[i][0]-points[j][0])**2 +
            ↪  (points[i][1]-points[j][1])**2))

    return distances


def findTriangles(catalog, N):
    selection = selectBrightest(catalog, N)
    points = []
    for i in selection:
        points.append((catalog[i]["xcentroid"], catalog[i]["ycentroid"]))
    distances = getDistances(points)

    triangles = []
    for i in range(0, N):
        for j in range(i+1, N):
            for k in range(j+1, N):
                if (convertToTriangleSpace([distances[i][j], distances[j][k],
                ↪  distances[k][i]])[1] < 0.1): continue     # ignore long triangles
                triangles.append((*convertToTriangleSpace([distances[i][j],
                ↪  distances[j][k], distances[k][i]]), i, j, k))

    return triangles


def binarySearchTriangles(triangles, value):
    l = 0
    r = len(triangles)-1

    while l != r:
        m = int((l+r)/2)
```

```
67          if triangles[m][0] < value:
68              l = m+1
69          else:
70              r = m
71
72      return l
73
74  def firstMatchingUsingTriangles(mergedCatalog, newCatalog, N, epsilon=0.02):
75      votes = [[0 for i in range(N)] for i in range(N)]
76      oldTriangles = findTriangles(mergedCatalog, N)
77      oldTriangles.sort()
78      oldSelection = selectBrightest(mergedCatalog, N)
79      triangles = findTriangles(newCatalog, N)
80      selection = selectBrightest(newCatalog, N)
81
82      # Optimize later with BS (if necessary...)
83      for i in range(0, len(triangles)):
84          for j in range(binarySearchTriangles(oldTriangles, triangles[i][0]-epsilon),
            ↪ binarySearchTriangles(oldTriangles, triangles[i][0]+epsilon)):
85              if (((triangles[i][0]-oldTriangles[j][0])**2) +
                ↪ ((triangles[i][1]-oldTriangles[j][1])**2)) < (epsilon**2):
86                  for x in range(2, 5):
87                      for y in range(2, 5):
88                          votes[triangles[i][x]][oldTriangles[j][y]] += 1
89
90
91      highestVotes = PriorityQueue()
92
93      for i in range(0, N):
94          for j in range(0, N):
95              smallest = -1
96              if (not highestVotes.empty()): smallest = highestVotes.get()
97              if (highestVotes.qsize() < N-1 or smallest < (votes[i][j], i, j)):
98                  highestVotes.put((votes[i][j], i, j))
99              if (highestVotes.qsize() < N and smallest != -1):
100                 highestVotes.put(smallest)
101
102     matching = []
103
104     while not highestVotes.empty():
105         tmp = highestVotes.get()
106         matching.append((selection[tmp[1]], oldSelection[tmp[2]]))
107
108     matching.reverse()
109     return matching
110
111
112 def getTranslation(mergedCatalog, newCatalog, N):
113     matching = firstMatchingUsingTriangles(mergedCatalog, newCatalog, N)
114
115     translation = np.array([None])
116
117     while True:
```

```
118         newX = []
119         newY = []
120         mergedX = []
121         mergedY = []
122 
123         for i in matching:
124             newX.append(newCatalog[i[0]]["xcentroid"])
125             newY.append(newCatalog[i[0]]["ycentroid"])
126             mergedX.append(mergedCatalog[i[1]]["xcentroid"])
127             mergedY.append(mergedCatalog[i[1]]["ycentroid"])
128             if (len(newX) == 6 and translation.any() == None): break                    #
                ↪  Only use first 6 in first iteration to make sure it's precise enough
129 
130         newCoordinates = np.vstack([newX, newY, np.ones(len(newX))]).T
131         mergedCoordinates = np.vstack([mergedX, mergedY]).T
132 
133         translation = np.linalg.lstsq(newCoordinates, mergedCoordinates, rcond=None)[0]
134 
135         distancesSquared = []
136         for i in matching:
137             transformedX, transformedY = [newCatalog[i[0]]["xcentroid"],
                ↪  newCatalog[i[0]]["ycentroid"], 1] @ translation
138             distancesSquared.append(((mergedCatalog[i[1]]["xcentroid"]-transformedX)**2
                ↪  + (mergedCatalog[i[1]]["ycentroid"]-transformedY)**2, *i))
139 
140         distancesSquared.sort()
141 
142         sigma = distancesSquared[int(0.6*len(distancesSquared))][0]
143 
144         newMatching = []
145 
146         for i in distancesSquared:
147             if (i[0] <= sigma*2):
148                 newMatching.append((i[1], i[2]))
149 
150         if (len(matching) == len(newMatching)):
151             break
152 
153         matching = newMatching
154 
155     return translation
156 
157 
158 def binarySearchCatalog(table, value, translation):
159     l = 0
160     r = len(table)-1
161 
162     while l != r:
163         m = int((l+r)/2)
164         if ([table[m]["xcentroid"], table[m]["ycentroid"], 1] @ translation)[0] < value:
165             l = m+1
166         else:
167             r = m
```

```
168
169         return l
170
171
172     def mergeNext(mergedCatalog, newCatalog, allowedError=3):
173         translation = getTranslation(mergedCatalog, newCatalog, 20)
174
175         newCatalog.sort("xcentroid")
176
177         transition = []
178         for i in range(len(mergedCatalog)):
179             bestDist = allowedError*allowedError+0.1
180             index = -1
181             for j in range(binarySearchCatalog(newCatalog,
                ↪  mergedCatalog[i]["xcentroid"]-allowedError, translation),
                ↪  binarySearchCatalog(newCatalog, mergedCatalog[i]["xcentroid"]+allowedError,
                ↪  translation)):
182                 mergedX, mergedY = [mergedCatalog[i]["xcentroid"],
                    ↪  mergedCatalog[i]["ycentroid"]]
183                 newX, newY = [newCatalog[j]["xcentroid"], newCatalog[j]["ycentroid"], 1] @
                    ↪  translation
184                 dist = (newX-mergedX)**2 + (newY-mergedY)**2
185                 if (dist < bestDist or bestDist == -1):
186                     bestDist = dist
187                     index = newCatalog[j]["id"]-1
188
189             bestDist = math.sqrt(bestDist)
190
191             transition.append(index)
192
193         # for i in range(len(newCatalog)):
194         #     if (matched[i]): continue
195         #     mergedCatalogWithNew.add_row(newCatalog[i])
196
197         return transition
198
199
200     def mergeCatalogs(catalogs):
201         transitions = []
202         for i in range(1, len(catalogs)):
203             transitions.append(mergeNext(catalogs[0], catalogs[i][:]))
204             catalogs[i].sort("id")
205
206         return transitions
207
```

D.4 generateField.py

```
1   # This file contains the function "generateField()" which will, given a target
2   # coordinate, radius, and image, return the small field around the target
3   # coordinate. The fields size is determined by radius. This is used when
4   # calculating the brightness of one star, as it would not make sense to look at
5   # the whole image when calculating the brightness of just one star.
```

```
6
7
8   def generateField(rgb, targetX, targetY, radius):
9       field = []
10      for i in range(round(targetY-radius), round(targetY+radius)):
11          field.append([])
12          for j in range(round(targetX-radius), round(targetX+radius)):
13              [x, y]=[j, i]
14              x=int(round(x))
15              y=int(round(y))
16              try:
17                  field[-1].append(rgb[y][x] ** 1)
18              except IndexError as error:
19                  print(error)
20                  print("star is probably not fully contained by all images!")
21                  field[-1].append(0 ** 1)
22
23      return field
```

D.5 getBrightnessOfOneStarInField.py

```
1   # calculates the brightness of one star in one image by taking a field of a few
2   # times a few pixels as input.
3
4   from cmath import sqrt
5   import numpy as np
6
7
8   def getMeanAndDeviation(starRegion, debug=False):
9       mean = [0, 0]
10      sum = 0
11      for i in range(0, len(starRegion)):
12          for j in range(0, len(starRegion[0])):
13              mean[0]+=i*starRegion[i][j]
14              mean[1]+=j*starRegion[i][j]
15              sum+=starRegion[i][j]
16
17      mean[0]/=sum
18      mean[1]/=sum
19
20      s = 0
21      for i in range(0, len(starRegion)):
22          for j in range(0, len(starRegion[0])):
23              distSquared=(i-mean[0])**2+(j-mean[1])**2
24              s+=distSquared*starRegion[i][j]
25      s/=sum
26      s = np.sqrt(s)
27
28      return mean,s
29
30
31  def subdivideMatrix(starRegion, amount):
32      newRegion = []
```

```
    for i in range(0, len(starRegion)*amount):
        newRegion.append([])
        for j in range(0, len(starRegion[0])*amount):
            newRegion[i].append(starRegion[int(i/amount)][int(j/amount)])

    return newRegion


def backgroundRemoval(starRegion):
    values = []
    for i in starRegion:
        for j in i:
            values.append(j)
    values.sort()
    background = values[int(len(values)/4)]
    for i in range(len(starRegion)):
        for j in range(len(starRegion[0])):
            starRegion[i][j]-=background

    return starRegion


def getBrightnessOfOneStarInField(starRegion, subdivide=3, debug=False):
    starRegion = backgroundRemoval(starRegion)

    starRegion = subdivideMatrix(starRegion, subdivide)

    # print(background)

    mean,s = getMeanAndDeviation(starRegion, debug)
    # print(s)

    res = 0

    for i in range(0, len(starRegion)):
        for j in range(0, len(starRegion[0])):
            if sqrt((i-mean[0])**2 + (j-mean[1])**2).real <= s*2.5:
                res += starRegion[i][j]
            # elif sqrt((i-mean[0])**2 + (j-mean[1])**2).real <= s*2.5+1.5:
            #     res += (1-((sqrt((i-mean[0])**2 + (j-mean[1])**2) - s)/1.5).real) *
            ↪  starRegion[i][j]


    return res
```

D.6 generateBrightnessOfAllStarsInAllImages.py

```
# This file contains functions for getting information about stars, like their
# brightness or their noise-level.
# The function generateBrightnessOfAllStarsInAllImages() calls another function
# to find the brightness of all stars in all images.
# The function cleanUpData() is responsible for data-cleansing.

```

```
7  from math import ceil, floor
8  from generateField import generateField
9  from getBrightnessOfOneStarInField import getBrightnessOfOneStarInField
10 from readImage import readImage
11 from myAlgorithms import rolling
12 import numpy as np
13 from multiprocessing import Pool
14
15 def cleanUpData(brightness):
16     brightness = np.array(brightness)
17     removeStars = []
18     removeImages = []
19     for star in range(0, len(brightness[0])):
20         for i in range(0, len(brightness)):
21             if brightness[i][star] == 0:
22                 percentageStar = np.count_nonzero(brightness[:,
                   ↪  star]==0)/len(brightness)
23                 percentageImage = np.count_nonzero(brightness[i]==0)/len(brightness[0])
24                 if percentageStar*2 > percentageImage or i == 0:
25                     removeStars.append(star)
26                 else:
27                     removeImages.append(i)
28
29     cleanData = []
30     usedImagesIndex = []
31     usedStarsIndex = []
32     for i in range(0, len(brightness)):
33         if i in removeImages:
34             continue
35         usedImagesIndex.append(i)
36         cleanData.append([])
37         for star in range(0, len(brightness[0])):
38             if star in removeStars:
39                 continue
40             if i == 0: usedStarsIndex.append(star)
41             cleanData[-1].append(brightness[i][star])
42
43     return cleanData, usedImagesIndex, usedStarsIndex
44
45
46
47
48 def getNoiseScoreOfStars(brightness):
49     score = []
50
51     for star in range(0, len(brightness[0])):
52         s = 0
53         curve = [brightness[i][star] for i in range(len(brightness))]
54         windowWidth = 20
55         rolled = rolling(curve, windowWidth)
56         for i in range(ceil(windowWidth/2)-1, len(brightness)-floor(windowWidth/2)):
57             s +=
                   ↪  (rolled[i-ceil(windowWidth/2)-1]-curve[i])**2/rolled[i-ceil(windowWidth/2)-1]**2
```

```
58
59          score.append((s/len(rolled))**0.5)
60
61      return score
62
63  def getBrightnessScoreOfStars(brightness):
64      score = []
65      for star in range(0, len(brightness[0])):
66          score.append(0)
67          for i in range(0, len(brightness)):
68              score[star] += brightness[i][star]
69
70      return score
71
72
73
74  def getBrightnessInOneStar(files, catalogs, transitions, i, radius=8):
75      rgb = readImage(files[i])
76
77      starsInImage = []
78
79      for currentStarIndex in range(0, len(catalogs[0])):
80          indexInCatalog = transitions[i-1][currentStarIndex]
81          if indexInCatalog == -1:
82              starsInImage.append(0)
83              continue
84
85          field = generateField(rgb, catalogs[i]["xcentroid"][indexInCatalog],
            ↪  catalogs[i]["ycentroid"][indexInCatalog], 8)
86          starsInImage.append(getBrightnessOfOneStarInField(field))
87
88      return starsInImage
89
90
91
92  def generateBrightnessOfAllStarsInAllImages(files, catalogs, transitions, debug=False,
    ↪  radius=8):
93      with Pool() as mp_pool:
94          brightness = mp_pool.starmap(getBrightnessInOneStar, [(files, catalogs,
            ↪  transitions, i, radius) for i in range(1, len(files))])
95
96      return brightness
```

D.7 generateLightCurve.py

```
1  # The function generateLightCurves() generates lightcurves using the calculated
2  # brightness values by comparing them.
3
4  from generateBrightnessOfAllStarsInAllImages import getBrightnessScoreOfStars
5  from generateBrightnessOfAllStarsInAllImages import getNoiseScoreOfStars
6
7  def generateLightCurves(brightness):
```

```
    brigthnessScore = getBrightnessScoreOfStars(brightness) # get information about all
    ↪ the stars
    noiseScore = getNoiseScoreOfStars(brightness)


    lightCurves = []
    for star in range(0, len(brightness[0])):
        factors = []

        for i in brightness:
            exoplanet = i[star]
            reference = sum(i[j] for j in range(len(i)) if
            ↪ abs(brigthnessScore[j]-brigthnessScore[star])/brigthnessScore[star] <
            ↪ 0.3 and noiseScore[j] < noiseScore[star]*1.2) - exoplanet
            if (reference == 0):
                if len(factors): factors.append(factors[-1])
                else: factors.append(1)
                continue
            factors.append(exoplanet/reference)

        multiplier = 1/(sum(factors)/len(factors))
        for i in range(len(factors)): factors[i]*=multiplier#*brigthnessScore[star]

        lightCurves.append(factors)

    return lightCurves
```

D.8 analyzeLightCurves.py

```
# This file contains the function "analyzeLightCurve()" which is responsible
# to fit a step-curve to the reduced data. It thin returns information about
# the fit like: score of the transit, depth, fit-quality, brightness in dip,
# normal brightness, start of transit, end of transit


import statistics
from multiprocessing import Pool

def fitStepCurve(l, r, curve):
    valuesIn = []
    valuesOut = []
    for i in range(0, l):
        valuesOut.append(curve[i])
    for i in range(l, r):
        valuesIn.append(curve[i])
    for i in range(r, len(curve)):
        valuesOut.append(curve[i])

    medianIn = statistics.median(valuesIn)
    medianOut = statistics.median(valuesOut)

    squares = 0

```

```
    for i in range(0, l):
        squares+=(curve[i]-medianOut) * (curve[i]-medianOut)
    for i in range(l, r):
        squares+=(curve[i]-medianIn) * (curve[i]-medianIn)
    for i in range(r, len(curve)):
        squares+=(curve[i]-medianOut) * (curve[i]-medianOut)

    return squares, medianIn, medianOut


def analyzeOneLightCurve(curve, minLength=10):
    bestSquare = -1
    mIn = 0
    mOut = 0
    l = -1
    r = -1
    for i in range(0, len(curve)-minLength):
        for j in range(i+minLength, len(curve)):
            if (i+1 + (len(curve)-j) < minLength): continue
            s, r1, r2 = fitStepCurve(i, j, curve)
            if (bestSquare == -1 or s < bestSquare):
                bestSquare = s
                mIn = r1
                mOut = r2
                l = i
                r = j

    bestSquare /= len(curve)
    if (bestSquare == 0): bestSquare = 0.0000000001

    d = {
        "score": (mOut-mIn) / bestSquare,
        "depth": mOut-mIn,
        "error": bestSquare,
        "dipFlux": mIn,
        "normalFlux": mOut,
        "startTime": l,
        "endTime": r
    }

    return d


def analyzeLightCurves(lightcurves):
    mp_pool = Pool()
    return mp_pool.map(analyzeOneLightCurve, lightcurves)
```

D.9 output.py

```
# The function output() generates the output.

import matplotlib.pyplot as plt
import myAlgorithms
```

```
import os

def output(lightcurves, times, analysis, count=10):
    count = min(count, len(analysis))
    analysis = sorted(analysis, key=lambda d: d['score'], reverse=True)

    try:
        os.makedirs("results/lightcurves")
    except:
        print("directory results/lightcurves already exists, continuing anyway...")

    for i in range(count):
        lc = lightcurves[analysis[i]["index"]]
        plt.scatter(times, lc, marker='.', color="black")

        WS = round(len(lc)/10) + 1
        rolling = myAlgorithms.rolling(lc, WS)
        plt.plot(times[int(WS/2):len(rolling)+int(WS/2)], rolling, color="blue")

        fit = ([analysis[i]["normalFlux"]]*analysis[i]["startTime"] +
        ↪  [analysis[i]["dipFlux"]]*(analysis[i]["endTime"]-analysis[i]["startTime"]) +
        ↪  [analysis[i]["normalFlux"]]*(len(times)-analysis[i]["endTime"]))
        plt.plot(times, fit, color="red")

        plt.title("candidate: " + str(analysis[i]["index"]))
        plt.title("coordinates: " + str(analysis[i]["coordinates"]) +
            "\nscore: " + str(round(analysis[i]["score"], 3)) +
            "\ndepth: " + str(round(analysis[i]["depth"], 5)))

        plt.subplots_adjust(top=0.8)

        plt.xlabel("time (julian date)")
        plt.ylabel("brightness (normalized flux)")

        plt.savefig("results/lightcurves/candidate-" + str(i) + ".png")

        plt.clf()
```

D.10 Auxiliary Scripts

D.10.1 main.py

```
# This is the main file of this program. It contains the main function which
# calls the run function with all the parameters it extracted from the program-
# call.

from run import run

import sys

def main():
    run(sys.argv[1])

```

```
12
13 if __name__ == "__main__":
14     main()
```

D.10.2 getFilelist.py

```
1  # This function returns all the files present in one folder. It filters out all
2  # files starting with "." (hidden files). It is planned to later implement the
3  # function to filter the selected files with a regex.
4
5  import os
6
7  def getFilelist(pathToFolder):
8      files = os.listdir(pathToFolder)
9      files.sort()
10     res = []
11
12     for i in files:
13         if(i[0]=="."): continue
14         res.append(os.path.join(pathToFolder, i))
15
16     return res
```

D.10.3 readImage.py

```
1  # This file is responsible for reading images from files. Theoretically cr2
2  # (canon raw) is supported. But the file "getTimeOfFile.py" breaks that
3  # support.
4  # The function bin() allows for the data to be binned. That is useful when the
5  # image is bayered (color-cameras).
6
7  import os
8  from astropy.io import fits
9  import rawpy
10 import numpy as np
11
12 def rebin(arr, new_shape):
13     """Rebin 2D array arr to shape new_shape by averaging."""
14     shape = (new_shape[0], arr.shape[0] // new_shape[0],
15              new_shape[1], arr.shape[1] // new_shape[1])
16     return arr.reshape(shape).mean(-1).mean(1)
17
18 def bin(rgb, size):
19     rgb = np.array(rgb)
20     newRGB = rebin(rgb, (int(len(rgb)/size), int(len(rgb[0])/size)))
21     return newRGB
22
23 def readImage(filename, binning = 1):
24     if (os.path.splitext(filename)[1].lower() == '.fits' or
       ↪  os.path.splitext(filename)[1].lower() == '.fit'):
25         with fits.open(filename) as hdu:
26             rgb = list(hdu[0].data)
```

```
            rgb.reverse()

    elif (os.path.splitext(filename)[1].lower() == '.cr2'):
        with rawpy.imread(filename) as raw:
            rgb = raw.postprocess(gamma=(1, 1), no_auto_bright=True, output_bps=16)

    else:
        raise ValueError("your file has a not supported type!")

    if binning != 1: rgb = bin(rgb, binning)
    return rgb
```

D.10.4 myAlgorithms.py

```
# This file implements some standard-algorithms which are used all over the
# program. Currently there is only a function which calculates the rolling
# average over data.

def rolling(numbers, window_size):
    i = 0
    moving_averages = []
    while i < len(numbers) - window_size:
        this_window = numbers[i : i + window_size]

        window_average = sum(this_window) / window_size
        moving_averages.append(window_average)
        i += 1

    return moving_averages
```

D.10.5 getTimeOfObservation.py

```
# This function returns the observation-time of a fits-image. It gets it out of
# the fits-header. If cr2 files (canon-raw) are used, the program will crash
# here because cr2 files don't have fits-headers.

import os
from astropy.io import fits

def getTimeOfObservation(filename):
    if (os.path.splitext(filename)[1].lower() == '.fits' or
    ↪  os.path.splitext(filename)[1].lower() == '.fit'):
        with fits.open(filename) as hdu:
            return hdu[0].header["DATE-OBS"]

    raise Exception("File type not supported!")
```