Photometry of Celestial Fireballs Using a Portable All-Sky Camera

by

Luke Galbraith Russell

Submitted in Partial Fulfillment of the
Requirements for the Degree

Bachelor of Arts

Supervised by
Dr. Jed Rembold

Department of Physics

Willamette University, College of Liberal Arts
Salem, Oregon

2018

Presentations and publications

Poster??

L. Russell, Photometry of Celestial Fireballs, Oral Presentation at Willamette University, SSRD (April 2018)

L. Russell, Photometry of Celestial Fireballs, Oral Presentation at Willamette University, Research Seminar I: Status Report Presentation (December 2017)

L. Russell, Photometric Analysis of Earthbound Fireballs, Oral Presentation at Willamette University, Advanced Techniques in Experimental Physics: Senior Proposal Presentation (March 2017)

Acknowledgments

I would like to thank the following for their own respective contributions:

- Dr. Jed Rembold for dealing with all my stupid questions regarding life and this thesis.
- Dr. Rick Watkins for showing me that space is pretty rad.
- Dr. Daniel Borrero for reminding me that `pile_of_illegible_code != satisfactory_response`
- Future Dr. Adam Newton Wright for his contributions on our future Nobel Prize-winning discoveries
- Kelsey Walker for having optimism so bright it would immediately oversaturate our all-sky camera's sensors.
- Mark Watney for making all my problems seem pretty trivial.

Abstract

When designing new satellites, incorporating damage mitigation techniques is currently difficult due to a lack of understanding about the near-Earth small asteroid population. In an effort to investigate this population, we have constructed a portable all-sky camera to continuously monitor the night sky. It has the capability to detect and then record meteors burning up in Earth's atmosphere. To begin to measure the number and size distribution of these small asteroids, this project focuses on writing software to automatically measure a meteor's brightness, calibrate it to a known astronomic luminosity and estimate its size.

Table of Contents

List of Tables

List of Figures

1 Introduction

Space, while mostly space, is not entirely empty. There are planets, stars, comets, and many other objects, but here we are concerned about meteors. Specifically, we are interested in the smallest meteors. While rocks the size of pebbles may not seem dangerous to those of us protected by Earth's atmosphere, they pose a great threat to equipment and life beyond our atmosphere. We desire to know how many of these objects are in orbit near Earth in order to estimate the necessary level of protection for our structures and to better estimate the lifespans of our structures.

Currently, one of the ways meteors can be detected entering our atmosphere is by all-sky cameras detecting the bright streaks they leave as they ablate in the atmosphere. This brightness is what gives them another name: fireballs. All-sky cameras are set up by NASA and other groups[JGD+11; TRML+07]. They are installed on roofs or mounts and hooked up to desktop computers in buildings. It is a big and inflexible setup. This large setup limits the amount of setups that currently exist, which in turn limits the amount of total sky coverage we can get. Our all-sky camera explores a new design: one in which its detection software is housed inside its small, basketball-sized chassis. It still depends on external power, but low power consumption makes it a strong candidate to run off battery power if needed. We aim to show that our tiny unit (named D6[1]) is capable of observing and analyzing fireballs just as reliably as the larger, more established networks .

We are using our all-sky camera to detect meteors and then implementing photometry to calculate the meteors' magnitudes, then extrapolating their masses from those magnitudes. There have been many other studies by larger all-sky networks to create a model for the population distribution. We have no reason to expect our data to deviate greatly from this trend. In fact, we are hoping to test the effectiveness of our all-sky camera setup by comparing its findings to the established model. Also, while the general trend is well known, there is no

[1]D6 is named after a droid in the Star Wars universe similar to R2-D2, but is green and owned by Wedge Antilles.

guarantee that it is as precise as it could be. We are curious about any possible small discrepancies within the model regarding the smaller size meteoroids, and believe that if any discrepancies are there, they would only become apparent if data was being collected much more frequently than the current rate.

The background section details the information needed to understand what fireballs are and why they are important. The size of a meteor is determined through a long chain of physical relations that begin at its apparent magnitude, and is explained in detail in the methods section. The actual results are then presented in the data section.

2 Background

With respect to the Sun, our planet is moving through space at 30 kilometers per second. Many other objects orbit the sun, and those orbits can overlap or nearly overlap Earth's. The vast majority of these objects are small and are difficult to detect. These objects are known as near-Earth objects. Near-Earth objects consist of many similar rocks, but they have important differences between them. Regardless, they are hazards of the solar system, so we use systems such as all-sky cameras to detect them and gather data about their population distribution.

2.1 Asteroids and Meteoroids

There are five terms that are often incorrectly used interchangeably to describe rocks from space. They are the following: asteroid, meteoroid, fireball, meteor, and meteorite. They are visually represented in Figure 2.1. It should be noted that a comet does not belong with these terms as only a small percentage of its composition is rock: a comet is a mass of ice and dust. Compositionally, asteroids and meteoroids are the same. They are composed of rock or metal and are traveling through space outside Earth's atmosphere. They are delineated by their size. Asteroids are much larger, such as the asteroid that caused the Chicxulub crater in the Yucatan peninsula [Bot07]. The accepted upper boundary on what constitutes a meteoroid but not an asteroid is around 10 meters. In order to be defined as a meteoroid, the piece of rock should be at least 100 μm[1][Ste96].

In terms of their origins, meteoroids are often fragments of asteroids that are separated by some sort of collision in space, or are small pieces of rock and dust that are left behind by comets. Near-Earth asteroids are most likely objects expelled from the main asteroid belt between Mars and Jupiter due to Yarkovsky thermal forces or secular resonances [Bot07]. The Yarkovsky thermal force is a net force that occurs from the heating up of one side of an asteroid or meteoroid from

[1]Anything smaller than this is nothing more than dust, which makes up the interplanetary dust cloud.

the sun. The hotter side dissipates more energy while cooling than the already cold side, and thus that departing energy results in a recoil force moving the asteroid in the direction opposite the hotter side [BVRN06]. Secular resonance is when an asteroid syncs its precession to a nearby planet. This disturbs the orbit of the asteroid, and causes it to slowly change its orbital eccentricity as a result [MM95].

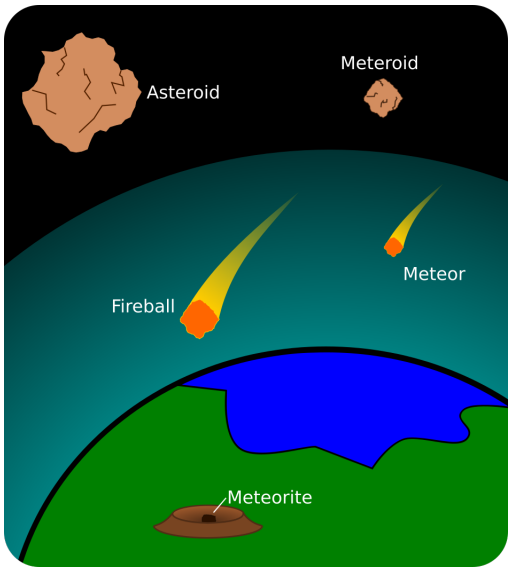

Figure 2.1: A visual representation of the different varieties of rock in the space near Earth (Photo courtesy of Dr. Jed Rembold).

2.2 Fireballs and Meteors

Asteroids and meteoroids are by definition only found outside the atmosphere of the planet. They only becomes meteors when they enter the atmosphere. Some literature even rechristens their status from near-Earth objects to inside-Earth objects, or IEOs for short [Bot07]. Fireballs are especially bright meteors, and are often chosen to be studied due to their comparatively easy detection. Technically, fireballs are only considered such if they are brighter than Venus, but the colloquial use is much less stringent [Har08].

Most of the time, meteors are completely destroyed as they travel through the atmosphere. This is done through ablation due to how they interact with molecules in the atmosphere. As the meteor hits the molecules in the atmosphere, ionization occurs. The point in the atmosphere at which light is emitted varies based on the velocity of the object. Slower objects need to be in a denser part of the atmosphere before ablation can occur. The average point at which this

happens is most often between 70 and 110 kilometers above sea level [HGB96]. During this process the meteor loses mass and energy in the form of light. The percentage of energy converted to life is known as the luminous efficiency and is an important term in predicting the mass of a meteor from its luminosity. If there is still mass remaining by the time it comes into contact with the Earth, it cools and becomes a meteorite. More often than not, however, meteors are destroyed tens of kilometers above sea level [HGB96].

Meteors tend to appear during two types of events. The first event is a meteor shower and can be reliably predicted. This is due to Earth going through parts of space where known debris are located. The debris are trails left behind by comets. The majority of these comet trails are due to water vapor drag. As a comet moves around the sun, different parts of its ice evaporate, freeing the small pieces of rock that were embedded inside it [Whi51]. A meteor shower can also be caused by a large amount of debris left behind when a comet is fragmented in space [Jen06]. Many meteor showers are well charted and known at this point. As of this writing there are 112 established meteor showers, with the potential of hundreds more [R R17]. Besides a meteor shower, the other event is a single isolated event, called a sporadic event. A sporadic event may have originally been part of a meteor shower or an asteroid belt before getting set off course due to extraneous forces.

2.3 Near-Earth Objects

While there are differences between the types of near-Earth objects, they all share one thing in common: they pose a hazard to any sort of structure or human sent out into space. Having more accurate and complete data on the population of these objects will give humanity a better chance to adequately defend their structures and shelters as well as more accurately estimate the lifespans of such structures. As space exploration develops, this will become more relevant, with potential landing spots for humans on the Moon and on Mars not having the luxury of the protective atmosphere of Earth. This can be seen just by looking up at the night sky; the moon is pockmarked with craters. This is not just exclusive to the moon, as craters can be seen on any rocky space object that does not have a thick atmosphere. These craters are the most obvious pieces of evidence of meteor strikes The moon also provides another strong example of the power of these strikes, as when a meteoroid hits the lunar surface, enough energy is released as to be visible from Earth [RR15]. Without the protective atmosphere, near-Earth objects could potentially wreak havoc on any structures. For example, the European Space Agency communication satellite Olympus was permanently decommissioned due to damage sustained during the Perseid meteor shower of 1993. The majority

of international and national organizations[2] concerned with planetary protection have all deemed large near-Earth objects to be credible threats to Earth and to any missions out of its atmosphere[Bot07].

Observational data from many different studies has resulted in a model for the population distribution of these objects. Specifically, there is a well-known linear relationship between the mass of an asteroid and the number that exist if plotted along log-log axes [RR15]. This is shown in Figure 2.2.

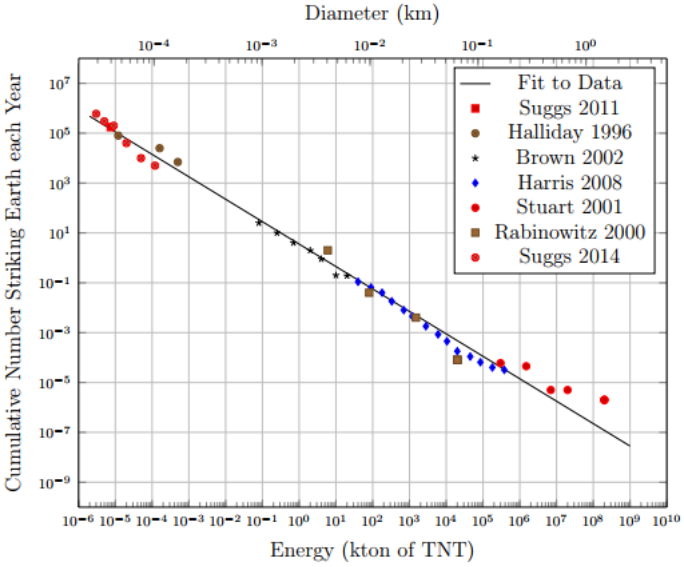

Figure 2.2: The population distribution of near-Earth objects is straightforward: small objects are much more likely than large objects. [RR15]

Initially this model seems reassuring. The larger objects, which would have the most potential to cause damage, are much less common than the smaller, less energetic objects. This can be seen in Equation 2.1, where m is equal to the mass and v is equal to the velocity of the object.

$$\text{Kinetic Energy} = \frac{mv^2}{2} \tag{2.1}$$

However, these small objects still pose a large threat to structures due to their extremely fast velocities. A one-millimeter meteoroid has similar strength to a 22-caliber bullet, while a one centimeter meteoroid can be as powerful as a cannonball. This is because velocity is the dominant term in equation 2.1 and meteoroid speeds can range from 11 to 70 kilometers per second [Har08]. Figure 2.3 shows the damage that a pebble-sized object can do to a thick slab of metal, and satellites without armor are much more fragile than that.

[2]These include, but are not limited to, the United Nations, the United States Congress, the European Council, the UK Parliament, the IAU, OECD, NASA, and ESA[Bot07]

Figure 2.3: Even small objects such as this ball bearing can exert tremendous amounts of force if they are going 50 kilometers per second [ESA17].

It should be noted that while this model appears consistent, and it may very well be, it is not guaranteed. With such a high population of smaller-sized objects, any variance could greatly change the likelihood of accurately accounting for potential damage. For instance, a satellite can only take so many impacts with meteoroids before becoming disabled. Even if there are only slightly more meteoroids than currently predicted, the lifespan of that satellite would be shortened, jeopardizing missions.

2.4 Detecting Fireballs

There are many current projects aimed at collecting optical data about meteors [JGD+11; TRML+07; HGB96]. Most make use of all-sky cameras. All-sky cameras are CCD cameras that are often utilized as video capture devices, recording either 25 or 30 frames per second depending on its video signal [MG05].

All-sky cameras are useful pieces of equipment for finding and analyzing objects in the sky. An all-sky camera is a camera that provides a view of up to 180° of the sky above, producing a circular image, as seen in Figure 2.4. This allows a clear view of the sky as a whole, allowing for easy identification of constellations. By locating constellations, stars can be identified and objects can have their relative location recorded.

In reality, this view is not quite 180°, but more often around 140° due to the fact that lenses wide enough to take in 180° end up just taking in the edges of the all-sky cameras chassis instead. Our own camera is only able to see 140° into the sky, as shown in Figure 2.5. Figure 2.5 also shows how much of the sky is covered. The amount of Earth being covered is defined by taking the edges of the all-sky camera view and treating those as zeniths. A zenith is the point in the sky

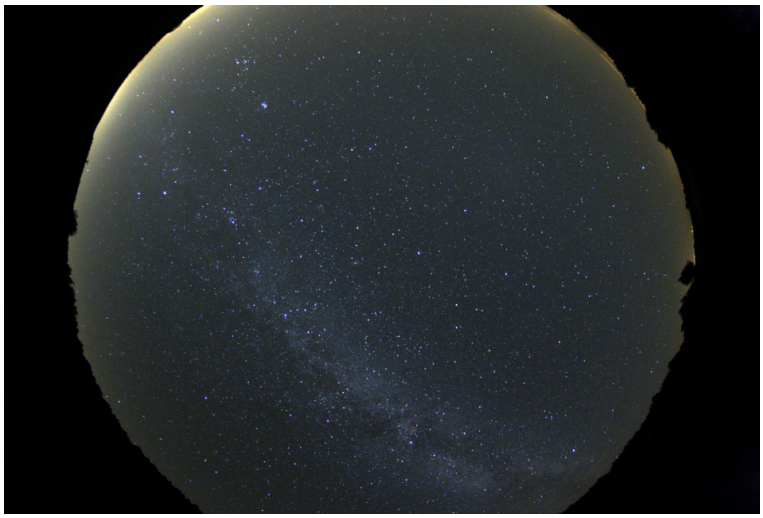

Figure 2.4: All-sky cameras feature a 360° view of the horizon, providing maximum sky coverage. This is an image from a high resolution color camera [Alc].

that is directly above a certain point on Earth. Those points are the edges of how much of the Earth it covers. Knowing how much of the Earth is covered is useful because those zeniths provide the points where another camera can be placed to overlap one half of the original camera's view of the sky. Overlapping is useful because multiple camera angles allow the triangulation of the object's position and velocity. The amount of sky coverage can be found through trigonometry, resulting in equation 2.2, where Ω is the steradian, $h + r$ is the distance from the center of the Earth to the edge of the area of sky coverage, and sr is the unit steradian.

$$\text{Sky Coverage} = \Omega(h + r)^2\text{sr} \tag{2.2}$$

Using an average height h of 75 kilometers, the area of the sky that is covered by an all-sky camera such as ours is around 119,000 square kilometers. This is equal to 118,000 square kilometers of land coverage. This is only 0.023 percent of the Earth that is covered, so there is a need for more all-sky cameras to ensure better coverage. Our all-sky camera has been in two locations, Salem and Baker City, Oregon. These two spots provide examples of how much land can be covered in Figure 2.6. Even for two locations in the same state, there is no overlap.

All-sky cameras are currently detecting fireballs in many places around the world. One notable example is NASA's Camera for All-Sky Meteor Surveillance (CAMS) Network. They have cameras located throughout many parts of the United States, forming a strong cohesive network. The CAMS network has been successful in correctly detecting all known meteor showers[3] during the month of

[3]The meteor showers were the theta Aurigids, chi Taurids, omicron Eridanids, and the sup-

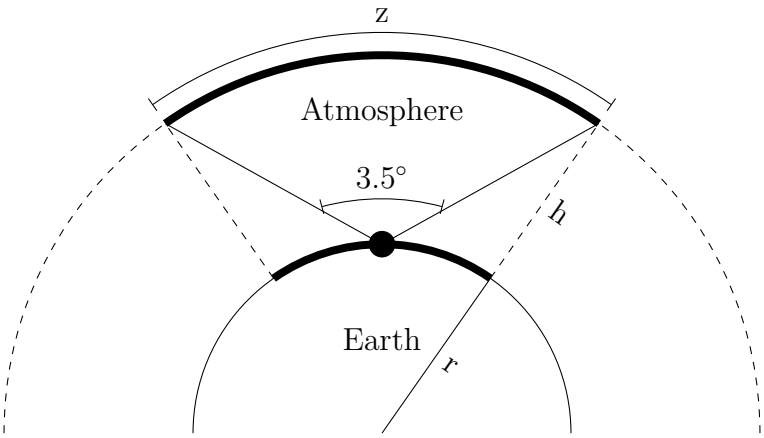

Figure 2.5: The all-sky camera can see distance z. The area of the Earth it covers are between the points that have the endpoints of z as their zeniths along height h. Note that the height of the atmosphere where ablation occurs is often less than 100 kilometers, while the radius of the Earth is 6,371 kilometers, so the radii are not to scale.

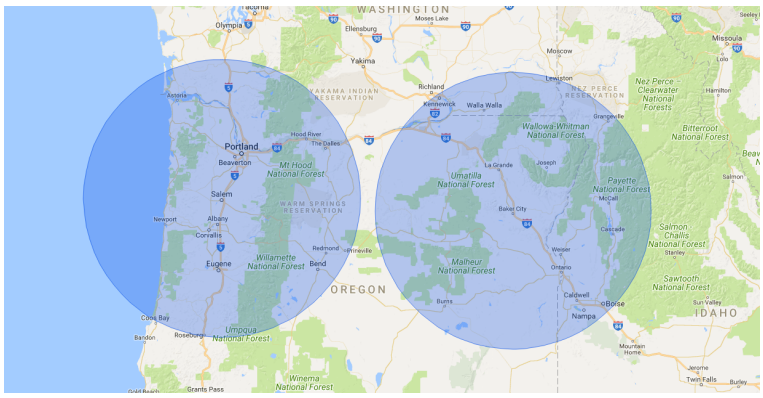

Figure 2.6: An all-sky camera can cover a circle with a radius of roughly 200 miles.

November in 2011[JGD+11].

NASA is not the only one to have such a network, with other organizations such as the SPanish Meteor Network (SPMN) also possessing well established systems of their own. They were able to successfully confirm expected meteor showers with the use of their high-resolution, all-sky cameras.[4] They even were able to find the nu Aurigids, that while previously identified, were not expected

posed iota November Aurigids. The latter was in fact shown to be merely an overlap between the theta Aurigids and chi Taurids with CAMS network data.

[4]The meteor showers were the Orionids, the Taurid stream related to comet 2P/Encke, and the delta Aurigids.

to be seen [TRML+07]. These organizations are able to collect relatively large amounts of data with all-sky camera networks compared to previous endeavors but they do not have enough cameras to come even close to covering the entire sky. Figure 2.7 shows the Sky Sentinel Camera Network and their coverage in the southwest United States. The network currently operates a network of around 65 cameras and hope to have over 100 in the future [Ban12].

The current model is to have these cameras connected to large structures or buildings with desktop computers where they can access the constant power and internet access that is needed for the detection and analysis. The necessary infrastructure that is currently needed lends these networks to be built in more urban areas, where light and air pollution become more problematic. Therefore this model also severely limits not only the quantity of skies covered, but also the quality.

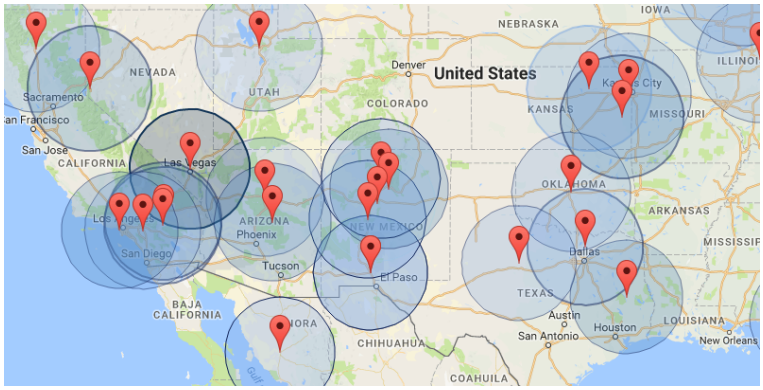

Figure 2.7: This network manages to cover a lot of area, but it may not have the best observational quality due to the cameras being centered in cities [Sky].

All-sky cameras themselves are not inherently complex. They are small and can be built relatively cheap, with builds being found online from Sky at Night Magazine and New Mexico State University, among others [Ban12]. Figure 2.8 shows the diminutive size of a camera used by the Sky Sentinel network, but also shows the power and video signal cords that have to connect to an external desktop for analysis.

Having a small, portable system has many potential benefits. Not only does a portable system increase ease of use and decrease equipment cost, it also vastly improves flexibility. For example, if nearby blooming trees block some view of the sky during the spring, a portable system could just be moved, while a permanent system would require the removal of the trees or the acceptance of a less than optimal observational status.

Software accompanies all-sky camera systems to perform meteor detections. This saves the user from having to watch hours of video looking for events that

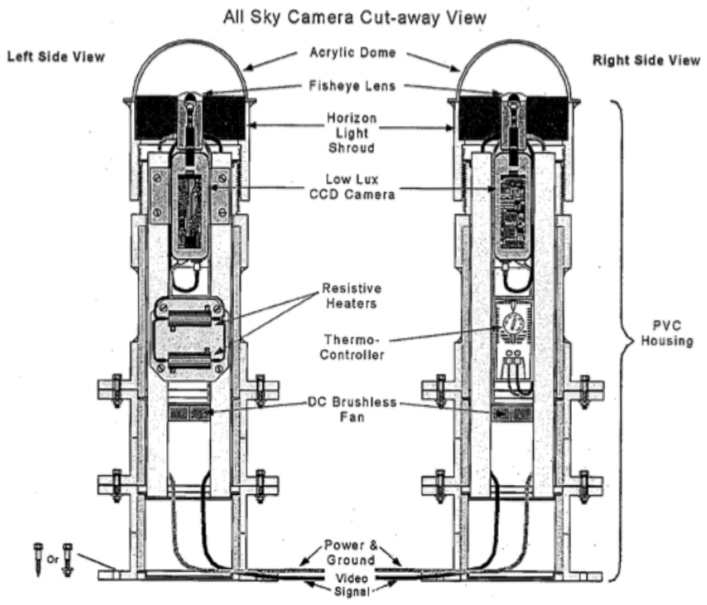

Figure 2.8: This schematic of an all-sky camera leaves out the attached structure to which the bottom cords must connect[Ban12].

may just be a few seconds long. Detection systems look for moving patches of light, and thus can produce numerous false positives from other environmental factors. These include light being reflected off moving clouds and planes flying through the field of vision [Har08]. Fortunately, these false positives can mostly be eliminated by visual inspection. The exact setup depends on the specific system, but meteor detection is either constantly processed by a computer checking the frames as they arrive, or the video is stored as a whole to be run through a detection program at a later date [MG05]. The software is full-fledged but bulky as a result. For example, METREC is a meteor detection software that is used by networks such as the Polish fireball network. It requires a Pentium processor [MG05]. The hardware requirements thus require a complete computer to accompany the camera.

2.5 Analyzing Fireballs

An all-sky camera can result in the acquisition of many pieces of useful information about a meteor. That being said, there are parameters a single all-sky camera cannot extract that a full network can, such as velocity and height. If there is a system of multiple cameras detecting the same meteor the orbit can be extrapolated as well [TRMW$^+$09]. The Canadian fireball network detected 259 fireballs in 1996, and was able to get values for height, velocity, magnitude and

mass for every fireball [HGB96]. A single all-sky camera would only be able to get magnitude, and would have to extract mass from that. For a full fireball network, having velocity means it can find the mass using equation 2.3, where m_d is the mass at a certain point in the flight, A is the cross sectional area at that same point in flight, ρ_a is the atmospheric density, and v is velocity.

$$\frac{m_d}{A} = \frac{\rho_a v^2}{2\dot{v}} \tag{2.3}$$

Not only does the velocity help determine mass in this situation, it also can be used to work backwards to find an estimate of the velocity before the affects of drag began slowing it down. This means that all-sky cameras can be used to gather data on the potential velocities of meteor showers in open space. The SPanish Meteor Network (SPMN) used velocity data to extrapolate such pre-atmosphere velocities with their five station setup in 2006 [TRML$^+$07].

Ultimately, all-sky cameras are most useful for measuring the photometric qualities of objects in the sky. They can detect fireballs and then provide the needed data to determine the fireballs' magnitudes, which is how bright they appear to the camera. This, however, can be done with just a single all-sky camera.

It should be noted that there are differing techniques when performing photometry. Different cameras are more sensitive to different wavelengths of light, and that can make comparing results from different cameras challenging. One solution to this is to apply a filter. A filter isolates light along a specific wavelength of the electromagnetic spectrum, as shown in Figure 2.9. For example, a visual filter is one that focuses on visible light, being centered on the wavelength for green light. The most commonly used values for star magnitudes are values determined after applying a visual spectrum filter. If a camera is more sensitive towards the infrared spectrum, an R filter would more likely fit its data, and can be applied instead [SMCS14]. It is possible to compare magnitude values with two different filters, but the process is somewhat convoluted and best to be avoided.

In some cases, it is useful to not apply a filter at all. While this can make it difficult to calibrate the camera to other known values, this guarantees that all light is being detected. This is crucial in highly sensitive detection situations where it would be ill-advised to sacrifice any of the small amount of potential data [RR15]. Even if no filter is applied, knowing the different bands allows the comparison of data to similar observations. For example, if the unfiltered camera sensitivity closely approximates the R band, R filter magnitudes are comparable.

The luminosity can be used to find the mass, but only by using approximations for other values such as density, which can be estimated due to the structural analysis of meteorite collections. As luminosity is the only needed data point, a single all-sky camera is just as viable for measuring the population distribution.

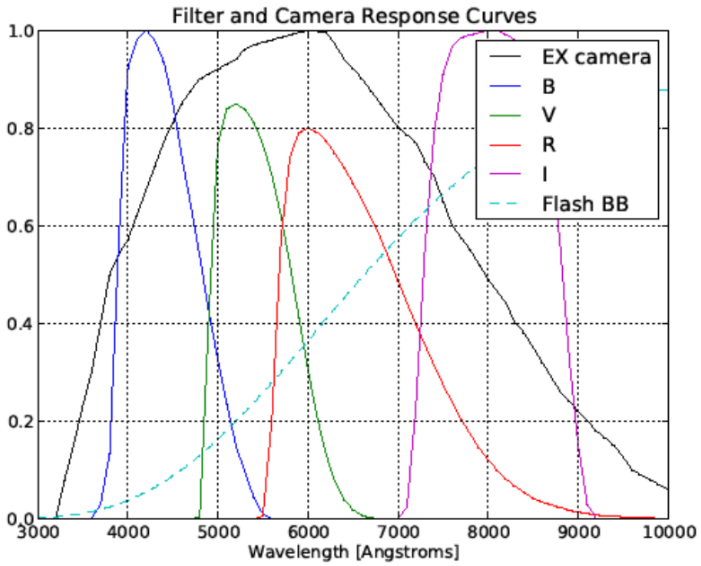

Figure 2.9: The visual filter, outlined in green, is shifted to the left of the R filter, which is towards the infrared part of the spectrum. The black outline is an example of what a camera's sensitivity may look like [SEM17].

But how does one get from intensity to luminosity? The relationship between the two is

$$L = I * SA, \tag{2.4}$$

where L is luminosity, I is intensity, and SA is the surface area over which the energy wave has propagated. In order to find the luminosity from intensity, we need to assume that the object radiates energy equally in all directions, forming a sphere. This assumption is not too far off from the actual shape of most meteors. Making this assumption, Equation 2.4 becomes

$$L_{obs} = 4\pi r^3 I. \tag{2.5}$$

Now that the intensity is converted to luminosity, that luminosity can then be used to find it's total kinetic energy. The relationship between the two is,

$$L = \tau T, \tag{2.6}$$

where L is again luminosity, T is kinetic energy, and τ is the luminous efficiency. Again, the luminous efficiency is simply the proportion of a fireball's energy that is dissipated as light. Once the kinetic energy is known, we can relate that to mass, as

$$T = \frac{v^2}{2}\frac{dm}{dt}, \tag{2.7}$$

where v is velocity, m is mass, and t is time. Again, with only one all-sky camera assumptions about the velocity have to be made, but as most velocities of meteors are within one order of magnitude, this assumption does not create problems along the log-log graph of Figure 2.2. Again, this is easily resolved by having multiple all-sky cameras detect the event. Notice that while the mass of an object is usually treated as constant when calculating kinetic energy, since the entirety of the object's mass is burning up, we can integrate over the entire event to find that original mass. This integral can be written as

$$m = \int \frac{2L}{\tau v^2}\, dt. \tag{2.8}$$

This integration is why the entire light curve of a meteor event needs to be collected. The methods for doing so is described in the next section.

3 Methods

The sizes of meteors are determined through a long chain of physical relations that begin with its apparent magnitude. So, the first thing to do is be able to find the magnitude of the fireball when D6 detects it. We have written a script where we can find the magnitude of the fireball as long as we know the apparent magnitude of another celestial object in the sky. This is done by automatically finding the center of the meteor the user selected on the image, and then automatically detecting the edges of the object through the creation of a fitting centroid. A Gaussian curve is then fitted to its light curve to provide an estimate of the fireball's radius. All the pixels within that radius is then summed over. This summation of pixels gives us the raw magnitude for our specific camera. The general outline can be seen in Figure 3.1. Each of the steps are explained in more detail in the following section.

3.1 Photometry

Once an event is detected by D6, it can then be run through a script that will analyze the frames and output the desired photometric information. The script is written in Python 3, and the user can run it through the use of the accompanying graphic user interface. The interface can be seen in Figure 3.2. A user simply runs the script while directing it to the location of the video they would like to analyze. The program runs through the video a single frame at a time, and loops through its functions until completing analysis on all the frames in the video.

The video processing is done through using the video read functionality of `OpenCV`. The video frames are always converted to grayscale at the beginning of the loop, so there is only one channel of pixel values instead of three that would occur with color video.

Upon running the script, the user will first see the first frame of the video. This is where the user will be required to make a selection on the screen identifying where the object and reference stars are located. The user simply left clicks on the

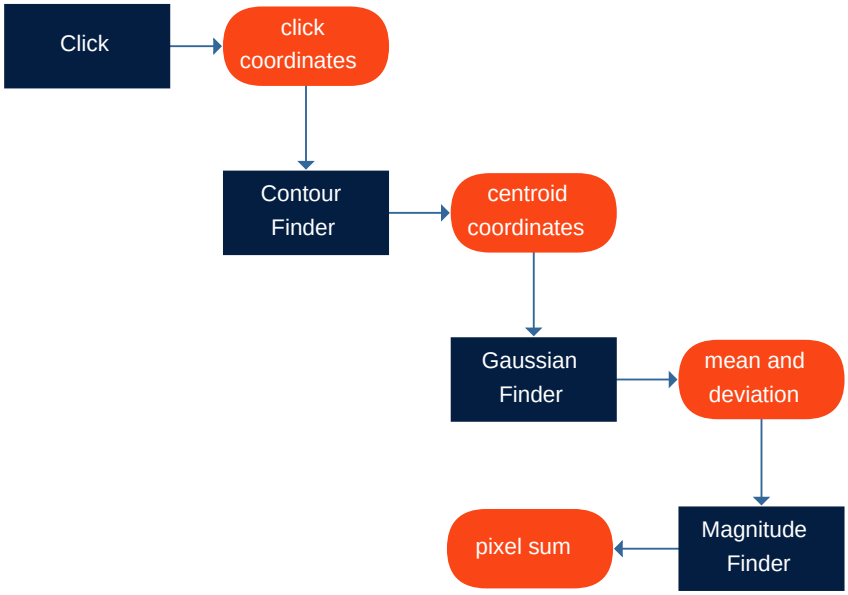

Figure 3.1: The program runs a series of functions (blue) that return specific information (orange) that is used by the following function.

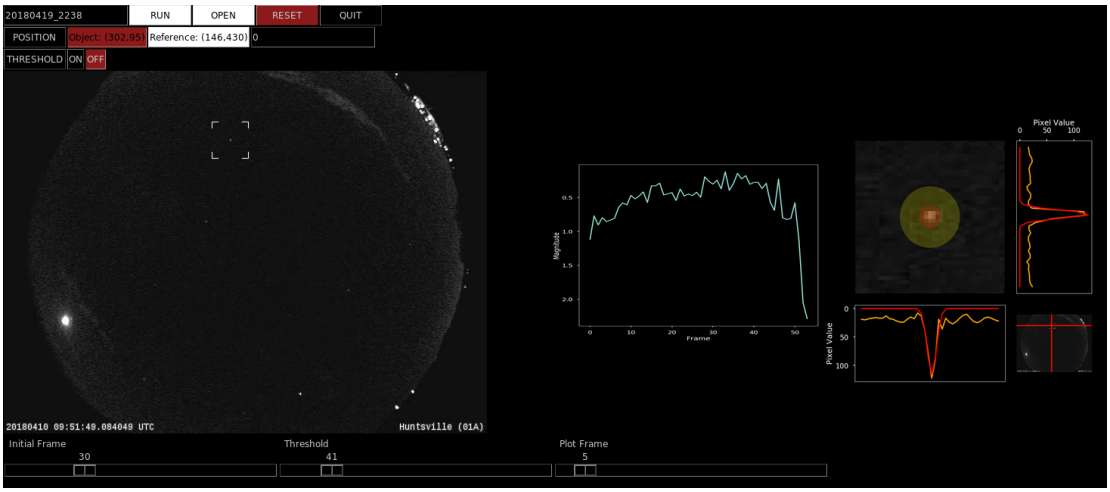

Figure 3.2: The GUI allows one to see the initial frame, and view the results all in one window.

desired object, and right clicks on a reference star of known magnitude.The user uses a slider to adjust the threshold level until both objects are clearly separated from the background. An example of a frame that a user could see is seen in Figure 3.3. The user's clicks initialize the script.

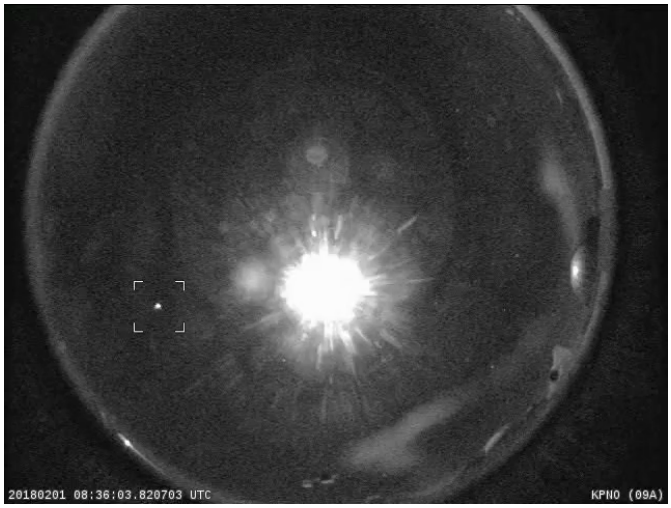

Figure 3.3: The first frame of an event. The user clicks on this image in the GUI.

3.1.1 Recognizing a User's Click

Why does the user need to click on the objects? The program needs a general idea of where the object is in question on the frame. It can follow the moving object after that, as long as it has a starting point. This is outlined in Figure 3.4

Using the `OpenCV` computer vision python package, the coordinates of the user's clicks can be extracted and used in the script. The user's left click records the corresponding X and Y coordinates as a tuple identify as the location of the object. The user's right click records the corresponding tuple as the location of the reference star. The user can click multiple times, with each click overwriting the previous one until the user is satisfied with their click accuracy. After both objects have been selected, the program's `Run` button is selectable. The user clicks the button to run the remainder of the script.

3.1.2 Fitting a contour to the object

The location of the object is now set as the exact pixel the user clicked on. The user may have provided a solid estimate on where the true center is with their click, but it most likely not actually the true center. The estimate allows the program to run its algorithm to find a more accurate location of the object in the

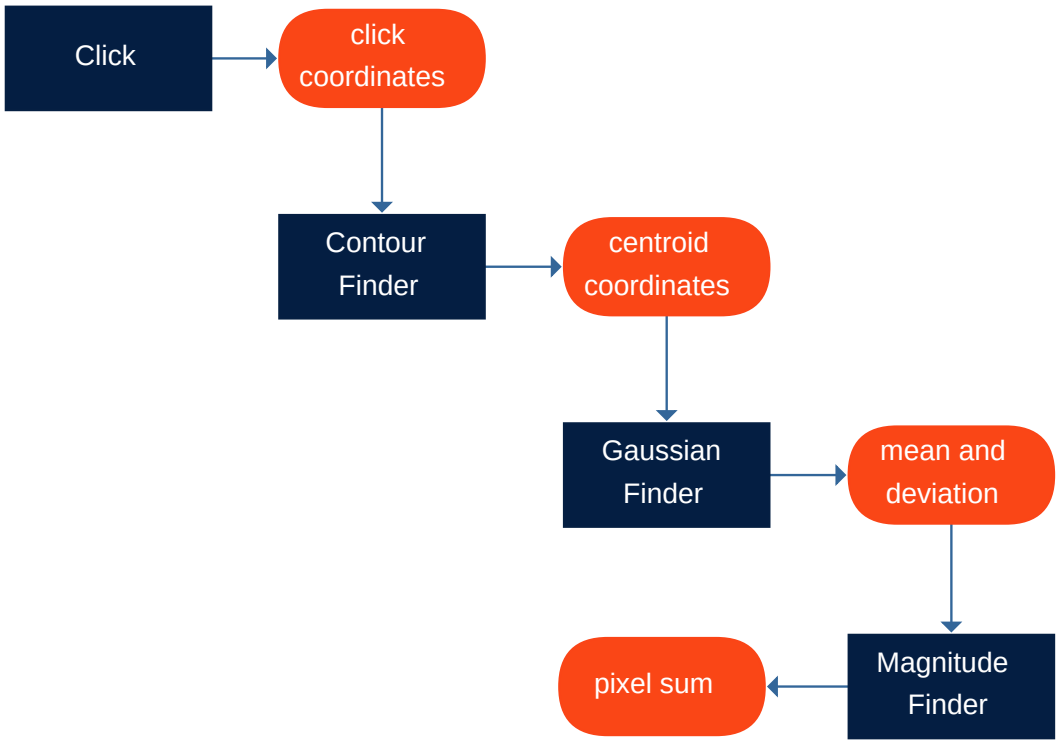

Figure 3.4: Each click gives crucial information for latter functions: The initial coordinates of the meteor and reference star.

nearby vicinity of the click. The aforementioned algorithm's process is outlined in Figure 3.5. This more accurate object location will be necessary when the program tries to fit Gaussian fits to the data, which will be needed to identify the radius of the object.

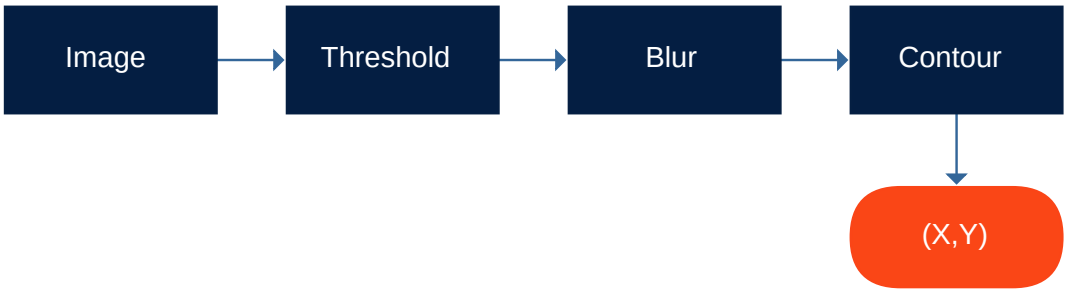

Figure 3.5: The steps the image is processed with to get a more accurate object position

The script takes the X and Y coordinates of where the user thinks the object

(a) The image before being thresholded. (b) The image after being thresholded.

Figure 3.6: A thresholded image, when adjusted properly, allows the program to locate an object easier.

is and isolates that part of the frame. The image is then blurred using an `OpenCV` command that blurs the edges of the pixels using a Gaussian equation to fade their light. This is done as an array of pixel values is discrete, and are not ideal to model a continuous equation, such as a Gaussian, off of. The image is then thresholded at a certain percentage. Thresholding essentially divides all pixels into a group lighter than the percentage and a group darker than the percentage. The darker pixels are all turned black, and the lighter pixels are all turned white. This, combined with the blurring, turns the object into a white round object. An example of this process can be seen in Figure 3.6.

This process also has the benefit of making it easier for the user to locate the object. It is especially useful for finding the reference star, which is often much dimmer than the meteor. This is evident by no stars being visible in Figure **??**, but some being visible in Figure **??**.

Another OpenCV command called `findContours` is implemented to find the center of that ellipse. Its algorithm is exactly like how one would find the center of mass of an object, but with the intensity of the pixel being the "mass" in the calculations. At this point, the program now has a much more precise idea of where the center is. With the object's coordinates updated, the thresholded frame is discarded, and the following analysis is continued with the undoctored data.

3.1.3 Fitting a Gaussian to the contour

Now that there is an accurate object location, we can fit a Gaussian curve to the data in both the X and Y dimensions. We want to do this because having a

Gaussian curve gives us a known standard deviation, and we can then calculate the radius of the event using that standard deviation. This process is outlined in Figure 3.7.

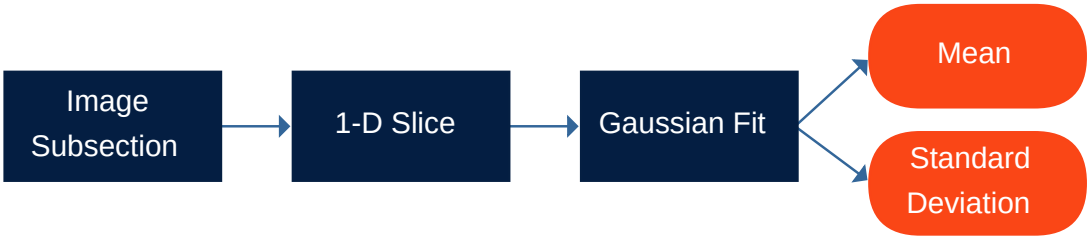

Figure 3.7: The steps taken to find the Gaussian parameters of the data.

The formula for a Gaussian curve is

$$ae^{-(x-x_0)^2/(2\sigma^2)} \tag{3.1}$$

where a is the amplitude, x is the point along the curve, x_0 is the center of the curve, and σ is the standard deviation. A function is used from the Python library called `SciPy`[1]. A Gaussian fitting will output a mean and standard deviation, but first the `SciPy` function needs a strong estimate of the mean to attempt to fit the data. This is why the center of the object was calculated using the contour method previous discussed. In our program, we calculate two one-dimensional Gaussian curves across that center. The function fits the Gaussian curves to 20 pixel slices centered at the point in the X and Y directions. The Gaussian curve also returns its mean, and the mean of the X slice and Y slice is updated to represent the objects location once again, which is used to fine-tune the location of the event. The clicking, contouring, and Gaussian fitting functions can be thought of as a sequence of steps to acquire the most accurate center, along with the acquisition of the standard deviation data mentioned. The curves can be plotted and saved as PNGs. The Gaussian curves of one such frame can be seen in Figure 3.8.

Occasionally, an anomaly or poor data will make the Gaussian function unable to actually fit the data to a Gaussian curve. In this case, an exception is called in the code to just use the estimates provided as the true outputs of the Gaussian function instead of the error message. By using the estimates, the sequence of functions can continue as-is without any additional manipulation. The frame is skipped in the creation of the final light curve. This is done to ensure the program can finish to completion, and not end in the middle. If this was not implemented, the program would be break before processing the command to create the light

[1]The author prefers the package to be pronounced "skip-ee" when discussing any matter involving this paper

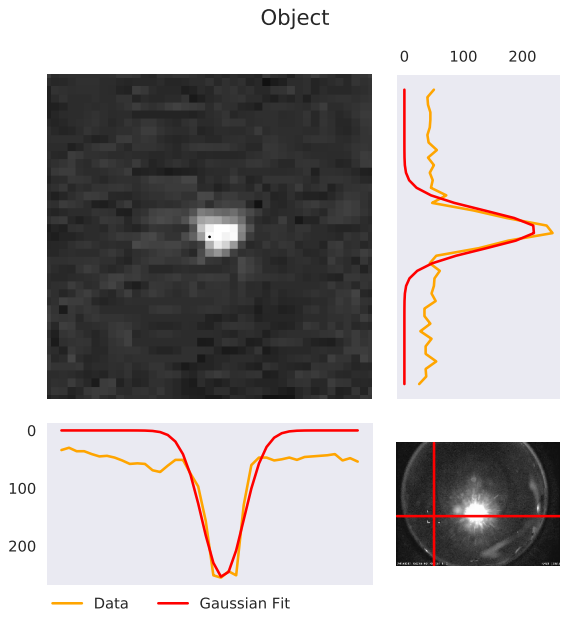

Figure 3.8: The Gaussian curves in this plot align with the image, so one can visually confirm that they are fitting the data.

curve. Later on, it is possible to view the data to ensure the Gaussian curves aligned well with the pixel values.

A Gaussian curve is to be expected from something such as a point light source as it dims out radially. An object will not have a definite edge: it will fade away into the background. This can be shown in Figure 3.9 of a Gaussian fit. The Gaussian function has the benefit that its standard deviations are directly proportional to certain percentages of the data fitted to the functions. This is commonly known as the 68-95-99.7 rule: one standard deviation covers 68% of the data, three standard deviations covers 95% of the data, and three standard deviations covers 99.7% of the data. After that, the diminishing returns are not enough to overcome the extra noise in our data.

There are few details in the calculation of the radius from the standard deviations, however. The X and Y radii are defined by multiplying their respective Gaussian curve standard deviations by two, and then rounding up to the nearest integer. While the standard deviations themselves may be decimals, the image consists of only integer values, so any other number would not be compatible. In other words, one can not sum over a fraction of a pixel; either the entire pixel is used or none of it is. The radius of the circle is then defined to be whichever of these two radii happen to be the largest. They should be the same in theory, but may vary by a pixel due to the rounding process mentioned. The pixels within

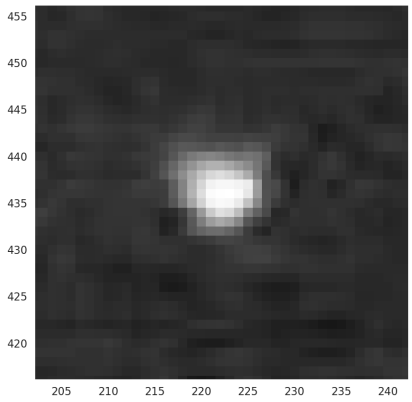

Figure 3.9: The object is located in the center of this slice, and its light fades radially out from that center.

this radius will be summed over in the next section.

Why do we sum over all the pixel values inside the radius instead of just integrating over the Gaussian fit? Figure 3.9 also provides a strong piece of visual evidence why someone wanting to just integrate over a two-dimensional Gaussian using the X and Y Gaussian slices would be oversimplifying the nuances in the data.

First off, A Gaussian function is by definition a continuous function; that is one of the reasons it is so nice for finding the radius, all of its values are usable. However, the data itself consists of pixels, a discrete form of data. This is why the program runs the radius up to the nearest integer; one cannot sum over a fraction of a pixel. This discreteness in the data is inherently different than a continuous model, and if the sum was found off that, it would be making too many assumptions about what the data *could* be instead of what it actually *is*[2].

All of this reasoning is still making one critical assumption of the data: that it is perfectly radially symmetrical. With the actual photometric data, we can clearly see there are some fluctuations in how wide the radius is at different points. In Figure 3.11, the object's appearance is stretched in the upper-left direction and in the upper half in general. This is why it is critical to remember that the Gaussian is merely an approximation used to automate the calculation of the radius. It is *not* perfect, and does not represent the data well. What it does do well, however, is gives a solid representation of where the majority of the pixels are.

[2]And while some assumptions have to be made in this photometry analysis, for that reason we cannot really afford to make any superfluous assumptions

3.1.4 Calculating the magnitude of the object

Now all of the needed information is gathered to find the magnitude of the object at that frame. With the parameters from the Gaussian function, we know where the object is centered and, by defining its radius to be three standard deviations, we know the radius of the object. After subtracting background light, we then can sum over all the pixels within that radius to find the amount of light, or the magnitude, that our camera sees from the object. Doing this every frame gives us the light curve of the event over time. A diagram describing this process can be seen in Figure 3.10.

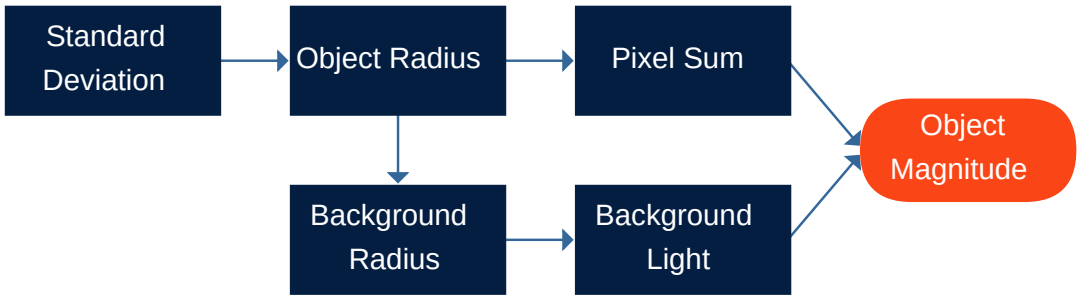

Figure 3.10: The steps the function goes through to find the magnitude of the object.

The radius can give us the lengths along the axis, but not the pixels of the entire circle we are searching over. The program takes a 10 x 10 search area around the center point, and checks all the points to see if they lie inside the given radius using the distance formula. If a point is within the radius, it is appended to a list to be used later. This can be seen in Figure 3.11.

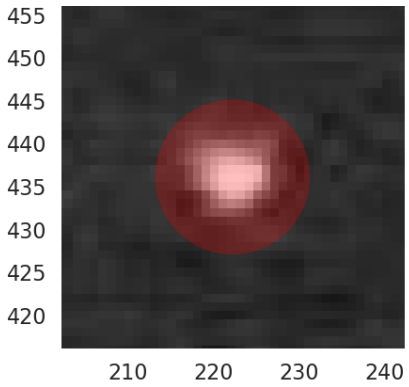

Figure 3.11: The pixels in red are within the calculated radius for this fireball.

We need to remove the background light from the calculation so we can sum over the pixel values attributed to the object's light only, not from any other source. If the background light is not subtracted the values of the pixels would be artificially elevated due to the existence of that inherent background light. To do this, we need to find a second radius of points around the circle that can be used to determine the average value of the background light. The background light can be somewhat noisy, so that is why we take an average of a group of background pixels. The pixels in question are chosen by finding pixels within a radius of the object's radius + 10 pixels, not including the pixels inside the object's radius. This creates a ring around the object's radius, as shown in Figure 3.12. In theory, if the Gaussian-based radius is too large, the background light subtraction will mitigate that overestimate.

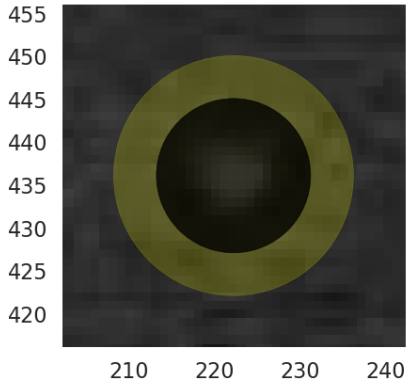

Figure 3.12: The pixels in yellow are the values being used to find the average background value. The object can be seen inside the yellow ring.

One problem that occasionally happens is the background search radius will go beyond the actual size of the image. This problem occurs when an object gets close to the sides of the image. In order to prevent this, and crashing the program, a loop goes through the pixel indices in the background list and checks to make sure they are not greater than the dimensions of the image. If any are, they are ignored.

There is also a two pixel buffer between the two areas for extra wiggle room. This is important due to the conversion to integers when finding the two radii. The radii are not perfect circles, as while the distance may be in the middle of a pixel, you can not just split a pixel. Both of the radii calculations round up, and take that whole pixel, which could create a pixel on the edge of the background radius and the object radius being used in both summations if there was not a buffer zone.This is easily seen when both the object's radius and the background ring is displayed on the same picture, such as in Figure 3.13

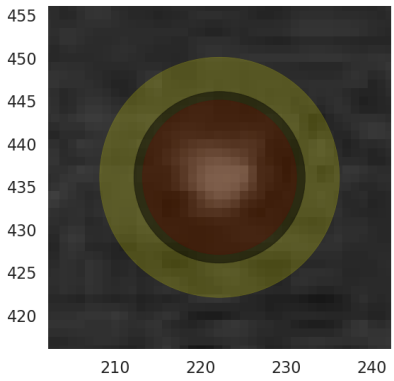

Figure 3.13: The pixels in orange are within the object's radius. The yellow pixels are the background ring. The gray area in-between is the buffer.

The program then sums all the pixel values in the list of values that were within the inner radius. This is done through a for loop that adds the value onto itself with every iteration through the list, while subtracting the average background magnitude from each individual point. This is the raw measurement of light from our specific camera.

Cameras have different sensitivities, so there is an offset that needs to be accounted for to have our specific camera be able to calibrate to the standard units of magnitude. This is then entered into Equation 3.2 to find the instrumental magnitude.

$$M_I = -2.5 \log_{10} \left(\sum \text{pixels} \right) + \text{offset} \tag{3.2}$$

This is the same equation from the previous section. The offset can be calculated by Equation 3.3

$$\text{offset} = M_I - M_C \tag{3.3}$$

where M_I and M_C are the instrumental magnitudes and catalog magnitudes of the same object.

This is why a reference star is needed. The object in question, the fireball, does not have a catalog magnitude, but the reference star does. This calculates the offset for the camera, which then can be applied to equation 3.3 for the fireball. This offset is calculated as an average over the entire event. The average is calculated due to the small amount of fluctuation each frame may have in their offsets due to variables such as atmospheric distortion or haze.

The program runs the previously mentioned functions for every frame in the video, and loops through this process until the video is completed. This overview can be seen in Figure **??**.

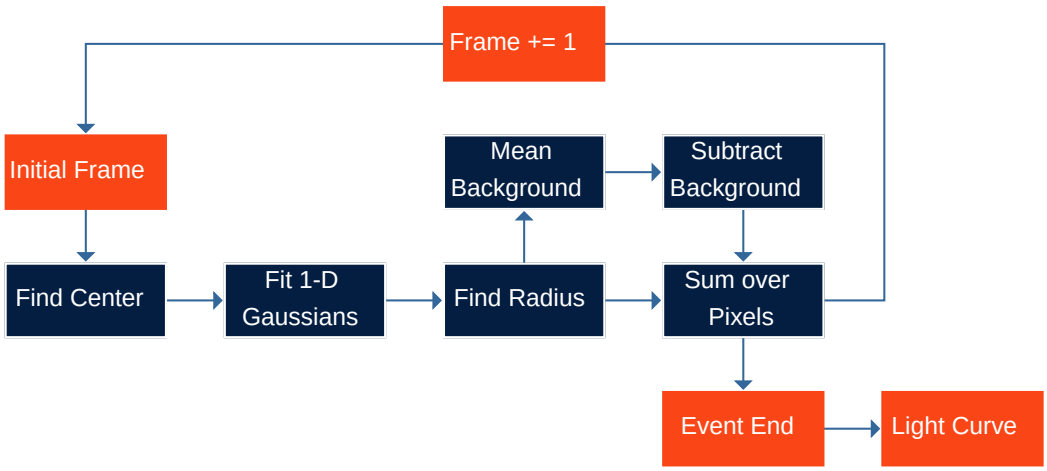

Figure 3.14: The program loops over most of the previous functions for each event.

In order to successfully loop through the frames, there needs to be a way for the program to jump from the center of one frame to the center of the next. The process of calculating a more precise center through each function allows the program to adjust the previous frame's center to the next frame's center without any difficulty, as the amount of pixels the center of the object appears to travel in each frame is usually no farther a distance than the average user's click is from the actual event for the first frame. This process is shown in Figure **??**

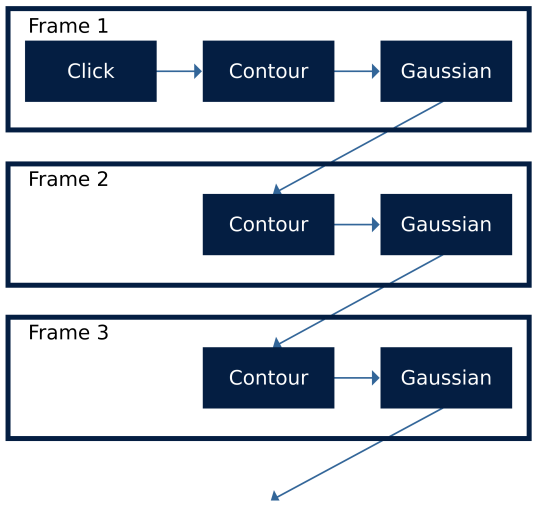

Figure 3.15: The updating process of the values used as the object's coordinates

PNGs of each frame's Gaussian fits along with the visualization of the radii are optional through the loop. If enabled, the result is exactly like Figure 3.8. If

this is enabled, the program may take up to 30 seconds to run, as it is memory intensive to save hundreds of PNGs. If it is disabled, the program finishes within a second.

4 Data

There are 3 types of data we can use to test the capability of the script. Iridium flares can be predicted before hand, allowing one to prepare to collect their data. NASA data provide a steady source of fireball events to access along with their data to compare to. Finally, fireball events detected by D6 will be tested,

4.1 Iridium Flares

The next step in testing out the script is to run it in conjunction with video collected directly from D6. A simple way to do this is to run it against a video clip of an iridium flare that D6 captured. There are two benefits of testing against an iridium flare. First, iridium flares are monitored online and their max magnitudes are recorded. We can find the maximum value from our light curve and compare it to that maximum magnitude. Second, the light curves from iridium flares are very smooth. This makes it easy to notice any discrepancies in individual frames. We have collected two iridium flare events using D6.

On *some date in fall*, D6 was able to capture an iridium flare going over Collins Hall at Willamette University in Salem, Oregon. The first frame of this event is shown as Figure 4.1.

The event was analyzed by the script, producing an accurate-looking light curve. This is shown in Figure 4.3

The first thing to check is if it qualitatively agrees with visual inspect of the event's video. The iridium flare's brightness seems to dramatically increase before dimming away. Both Figure 4.3 and Figure 4.2 agree with that assessment. So in a general sense, the photometry program is capable of following the event.

Some issues did emerge from this event, however. Ideally, we would like to be able to compare the maximum magnitude from the light curve our program created to the recorded maximum magnitude of the event. Unfortunately, the iridium flare was so bright that it oversaturated our camera. This can be seen in Figure 4.4.

Figure 4.1: The first iridium flare captured by D6 at Willamette University.

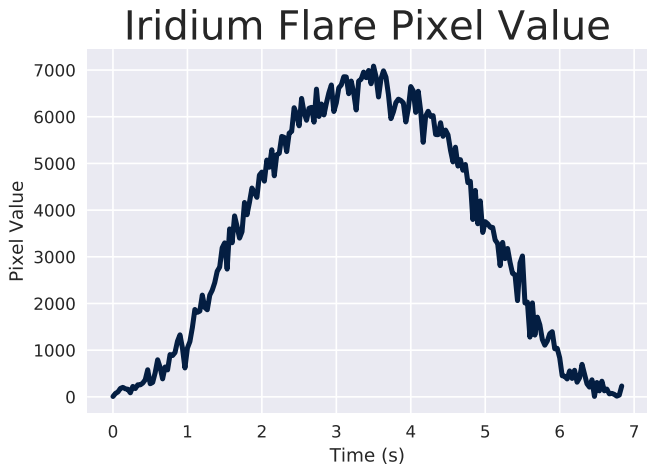

Figure 4.2: While light curves are usually displayed in terms of magnitude, it's important to remember the source of those magnitude values: pixel values.

Figure 4.4 allows us to see that the center of event is too bright for our camera's sensors. As a result, we are unable to find the current maxium magnitude of that iridium flare event. This saturation also explains why the peaks in Figure 4.3 and Figure 4.2 last for a substantial portion of the event; the true peak brightness isn't visible on those graphs.

Another issue that is apparent in the iridium flare data is an excess of noise insections of the data. Specifically, the noise at the end of the event appears most jarring in Figure 4.2. When the event nears its end, the signal to noise greatly decreases. This noise is a result of the background light that has already been mentioned. Since the event is at its end, it is becoming incredibly dim, and in its last moments its signal is basically equal with the background light

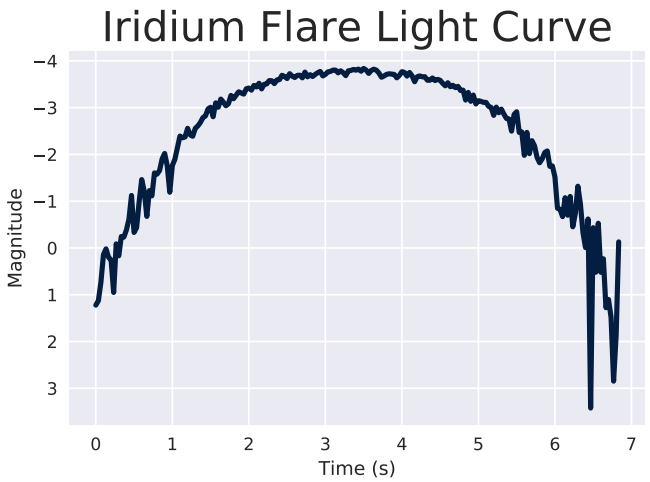

Figure 4.3: The light curve of the iridium flare event.

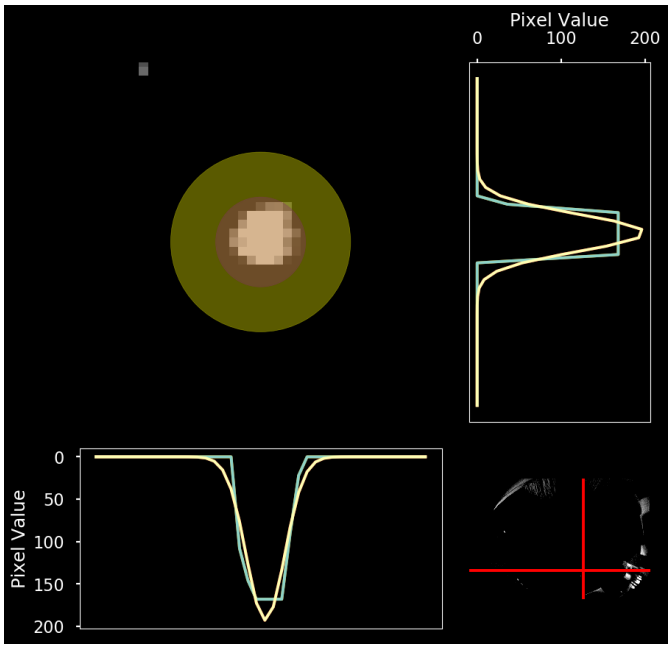

Figure 4.4: The peak of the event flattens out, strong evidence that the event was too bright for our camera.

around it. Since the background light value we are using is the average of the area around the event, there arises occasions where the pixels of the event are smaller than the background light at this stage. This creates a magnitude that is the resultant of a log of an incredibly small, or even negative number. Since taking the logarithm of a negative error would break the program if left untreated, those values are set to 0. So, at the end of an event before the program no longer

detects any trace, it has a few data points that are incredibly smaller, whose tiny size is magnified (this seems oxymoronic???) in the logarithmic-based light curve. While not aesthetically pleasing, this small values do not affect our results, as our results will be based off integrating the intensity along the light curve.

Iridium flares are somewhat useful pseudo-meteors, but ultimately the program needs to show that it can successfully track actual meteors. When running the program on an iridium flare event, the tracking algorithm is not challenged greatly; the flare's light was at a relatively constant position on the camera's frames. Meteor events that do move far across the screen pose a new challenge for the program.

4.2 Comparison to NASA Data

The methods section provided an example of an acquired light curve from the completion of analysis from a NASA video clip. By running the script against clips from NASA, we can confirm that the script is working by comparing it to NASA's light curves provided with the videos. The collected data and results of this test are discussed below.

The first event that we acquired from NASA's database was one that occurred on March 21, 2018, by one of their cameras stationed at LOCATION. The video was downloaded alongside their infographic of data involving the event. This infographic, displayed as Figure 4.5 contains the light curve of the event in the top right corner.

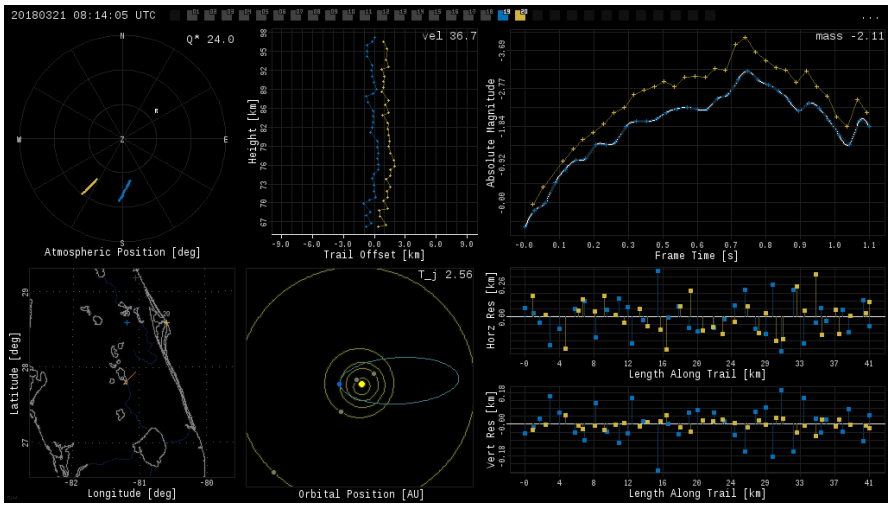

Figure 4.5: The light curve NASA attained off the same video is shown in blue.

While the light curve is crucial to compare our results to, the infographic also contains crucial information for star calibration such as the video's location,

the event's direction, and the time of the event. This allows us to make use of planetarium software to view how the stars were positioned at that time and orientation, so we can identify a correct reference star. For our purposes, we used stellarium. Stellarium is a free, open-source[1] planetarium available on all three major operating systems[2].

With the planetarium software, we were able to obtain a view of the position of celestial objects at that time, as seen in Figure 4.6.

Figure 4.6: Only celestial objects with a magnitude brighter than 2.00 are filtered through for ease of identifying them.

Figure 4.6 was then compared to the thresholded video frames of the event in order to try to identify one of the brighter objects. Assuming the stellarium was positioned correctly, celestial objects should appear to be in the same place in both images. Figure 4.7 was the frame that was compared.

Looking at Figure 4.7 alongside Figure 4.6 one can pick out which objects are which. A few pieces of data can be extracted from the two images. The only piece of data that we need is the magnitude of a single object, which can be obtained by clicking on that object in stellarium. For this calibration, Jupiter was used with its magnitude of -2.35 at the time. While only one object is needed, Jupiter and other notable stars are labeled in the identified version of the thresholded frame in Figure 4.8.

[1]Open-source projects are one the best catalysts of intelligent thought in a society. If possible,

Figure 4.7: The threshold frame allows one to focus on the easily identifiable, bright, objects in the sky.

Figure 4.8: Put Labels on this one, Luke!

Another piece of data that can be extracted is that the event in question was most likely from the θ-Virginids or π-Virginids, as the event is in the same area as those two meteor showers were on that day. This is another piece of strong evidence that the sky is oriented correctly in the images.

Now that we had a celestial object with a magnitude to calibrate to, the program was ran on the event. When it finished, the end result is the magnitude of each frame plotted over the time of the event, showing the light curve of the event. The light curve of this event from the program is shown in Figure 4.9.

The general trend shows that program's methodology in calculating the magnitude is sound. We would expect the brightness to gradually grow before dimin-

they should be used at all costs

[2]Available on most, but perhaps not all, Linux distributions

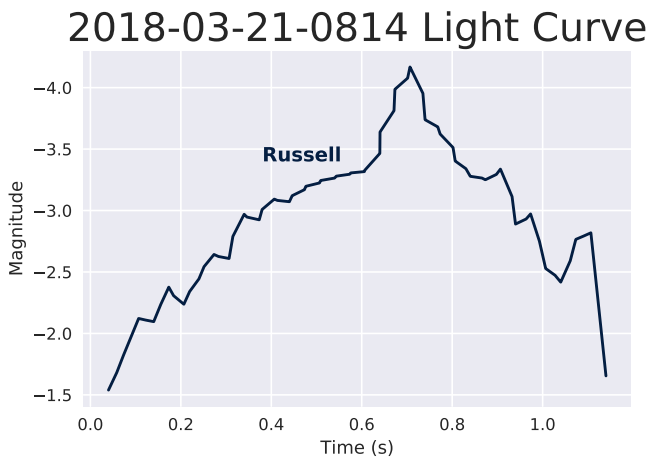

Figure 4.9: A lightcurve the program calculated off one of NASA's detected meteor events.

ishing again, and the graph agrees with that. This event in question was detected off from one of NASA's cameras. As a result, we can compare our light curve to NASA's to double check. NASA's light curve can be seen in Figure 4.10. They are pretty similar.

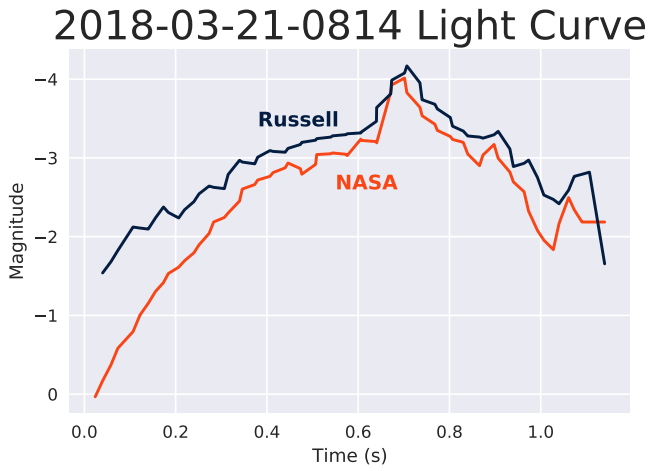

Figure 4.10: The event in question is the light curve on the top.

One aspect of the data that does not seem to fit well is the beginning of the event. This can be seen quantitatively by plotting the residuals of the two data sets and looking for outliers, which is done in Figure 4.11

The error can also be displayed with the use of a violin plot, shown in Figure 4.12. This allows us to see whether it is overestimating or underestimating

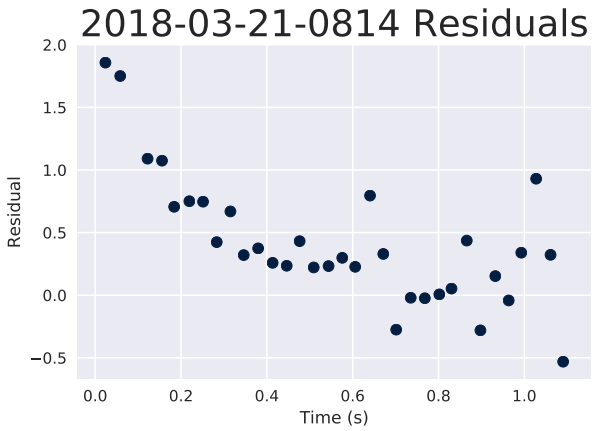

Figure 4.11

the magnitude by looking at the tails of the plot. By focusing the plot's tails, that information is not masked underneath a consistent offset some data may have.

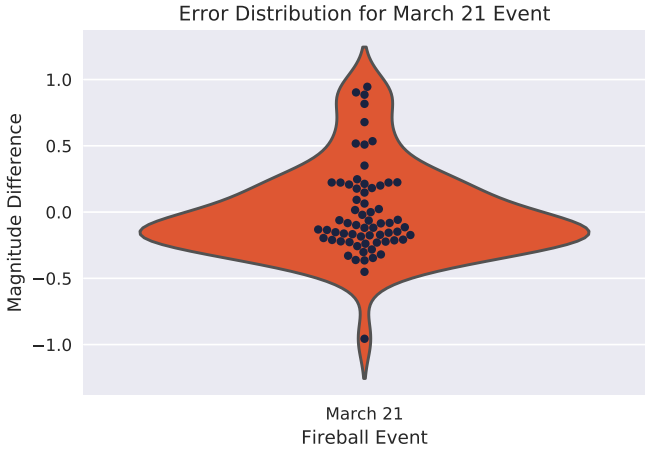

Figure 4.12: Testviolinplot

Light curve comparisons and the responding violin plots were then made for other events pulled from NASA's all-sky camera network. The light curves are shown in Figure 4.13.

The results were generally in agreement with each other. This is seen in Figure 4.14.

This discrepancy may be because the program does not take into account parameters such as atmospheric extinction. These parameters are more influential if the object moves radially across the night sky in the camera's frame. If NASA did take into account this affect, then that would explain the difference.

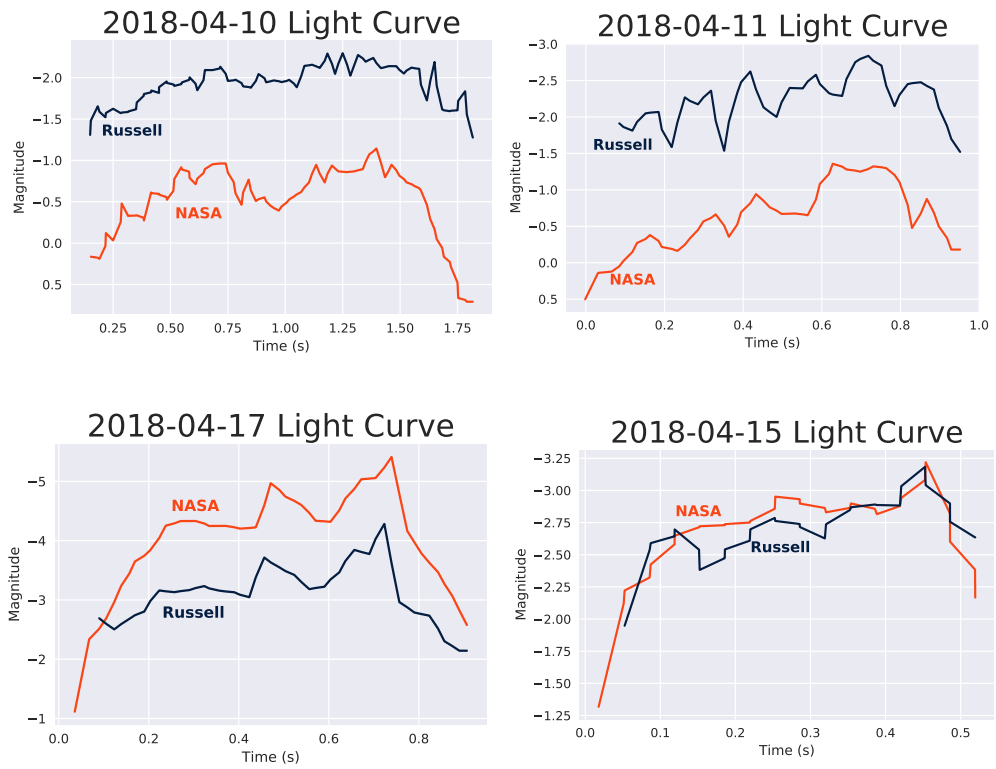

Figure 4.13: Light curves collected over multiple events.

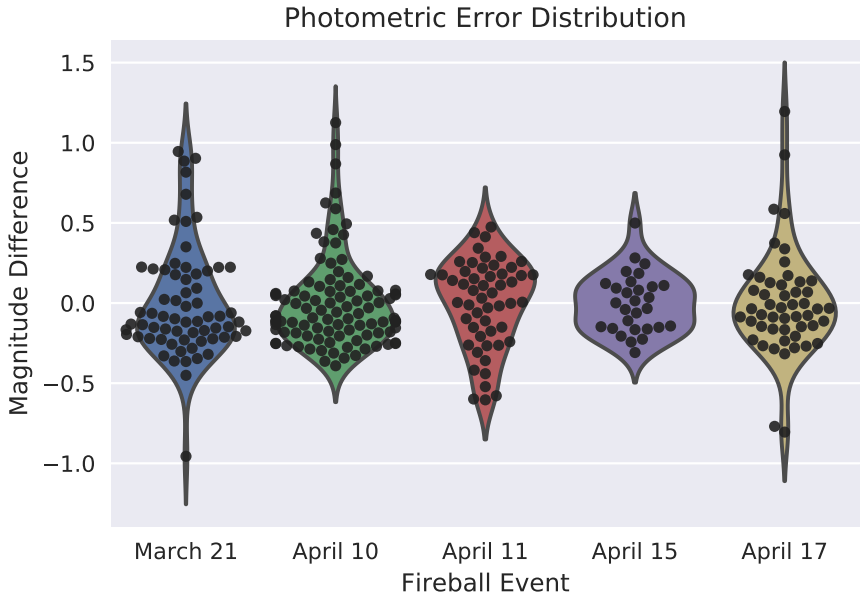

Figure 4.14: The violin plot for all these events have a few outliers.

4.3 Fireballs detected with D6.

Ultimately, however, the goal of our research is to show our all-sky camera's capabilities. The final test of the photometry script is thus the ability for it to detect new data collected from our own all-sky camera. With self-collected data, however, there are often problems that need to be troubleshooted. One problem that emerged early on in our desire to build an ultra compact and affordable unit was that our hardware was not fast enough to write the frames of the events to the hard drive cleanly. The result of this is incomplete video frames, creating gaps in the data. This can be seen in Figure 4.15.

This results in frames with much lower pixel values inside the object's radius than what one would expect, and what it would be if accurately representing the meteoritic phenomenon correctly. This is further seen in the resulting light curve, show in Figure :D6LightCurve.

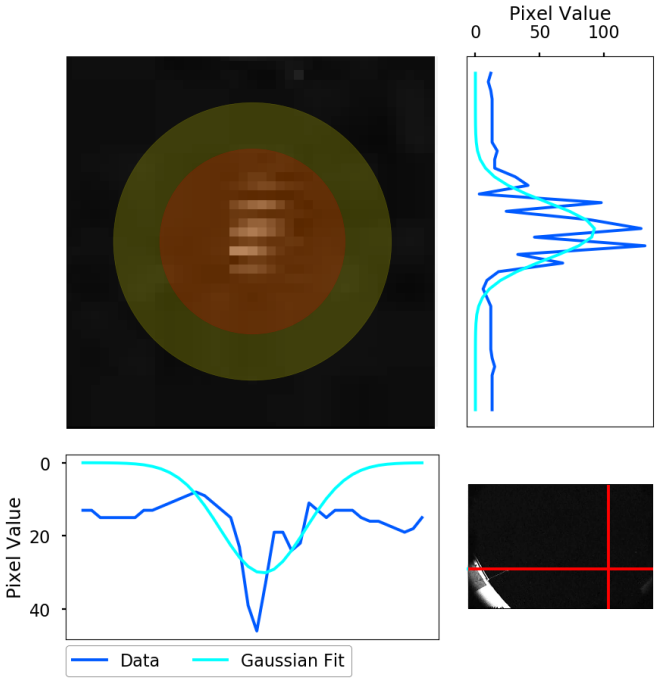

Figure 4.15: The incomplete frame data can be seen in both slices of the pixel data.

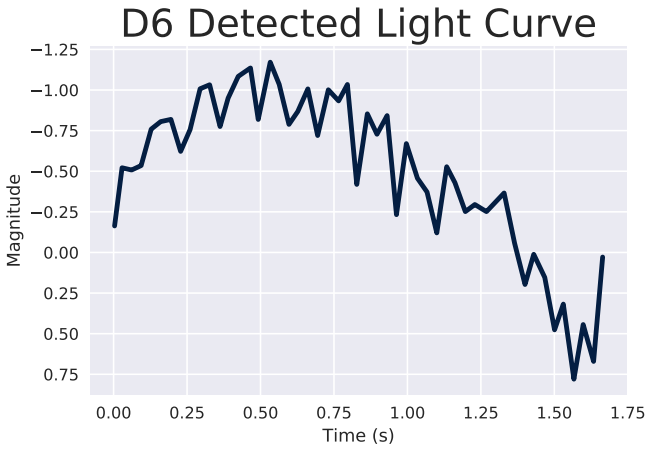

Figure 4.16: The light curve of this event is much less smooth than one would expect from a fireball

5 Conclusion

So what does this data show us regarding the current capabilities of our all-sky camera setup? There are improvements to be made, problems to troubleshoot, but on a wholistic sense, the system is most assuredly capable of photometrically analysing the events.

5.1 Critique

As mentioned, there are improvements to be made. Further examination is needed to figure out whether or not the discrepancies at the beginning of the light curve is based off a correctable parameter such as atmospheric extinction, or whether it arises from a systematic failure in the program to successfully find the magnitudes of dimmer objects.

Another potentially problematic issue is there is no current way for us to compare our calibrated light curves to other calibrated magnitudes. Each of the potential ways to do so are somewhat flawed in one aspect or the other. Comparing our data to NASA's allows us to effectively see if the shape of our light curve is accurate, but as they are uncalibrated, we cannot confirm our calibration is right then either.

The other source of comparison is the iridium flare data. The iridium data does not have light curves to compare to but they do have posted maximum magnitudes. Unfortunately, as mentioned in the data section, our camera cannot see that maximum magnitude in most iridium flare events due to oversaturation.

In order to confirm our calibration is accurate, ideally we would find data sets with calibrated magnitudes to compare to. However, it is worth noting that if the program can detect the shape of the light curve accurately, which it has repeatedly shown it can, it should have no problem with the calibration. After all, the calibration is dependent on the same algorithm, only it arises from the calculation of the reference star's magnitude instead of the object's magnitude. In fact, it should be even easier to do so, as the reference star does not move

throughout the duration of the event; the program does not need to update the position it is covering.

5.2 Outlook

¡¡This is vague¿¿ This project is far from being completed. Creating a program that appears to successfully analyse meteor events is the most important building block for collecting data, but there is much more to do beyond that. Now that the script is set up, a substantial amount of events is needed to be collected with our own all-sky camera to be able make any reasonable conclusions about the near-Earth population. The data that needed to be collected is the photometric data of the events, but there is substantial statistical analysis that needs to be done to extrapolate that photometric data into meteor size. The future of this project is thus based on optimizing the data collected and performing data analysis on that data to justify drawing any reasonable conclusion from the collection of individual events.

A Code

A.1 Photometry Script

```
#####################################################
#
#
# Identifying Objects & Their Magnitudes
#
# Luke Galbraith Russell
#
# Willamette University
#
#
#####################################################

import seaborn as sns
import numpy as np
import cv2
import argparse
import math
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import matplotlib.gridspec as gridspec
import matplotlib
from PIL import Image

END = False
REFERENCESTARLOC = (0,0)
OBJECTLOC = (0,0)
OBJECTAVGRADIUS = 0
OBJECTBACKGROUNDRADIUS = 0
```

```
REFERENCESTARAVGRADIUS = 0
REFERENCEBACKGROUNDRADIUS = 0
XFitParametersRef = []
YFitParametersRef = []
XFitParametersObj = []
YFitParametersObj = []

matplotlib.rcParams['figure.figsize']=(10,10)

#####################################################
# Creating function for mouse click
# Left Click is for Reference Star
# Right Click is for the Object
#####################################################
def click(event, x, y, flags, param):
    global REFERENCESTARLOC
    global OBJECTLOC
    if event == cv2.EVENT_LBUTTONDOWN:
        REFERENCESTARLOC = (x,y)
    elif event == cv2.EVENT_RBUTTONDOWN:
        OBJECTLOC = (x,y)

def Threshold(theimage, ThresholdRatio):
    threshold = cv2.threshold(theimage, ThresholdRatio, 255,
            cv2.THRESH_BINARY)[1]
    return(threshold)

def MaxFinder(X, Y, img):
    global END

    r = 10
    # Accessing array so Y values first
    subimg = img[Y-r:Y+r, X-r:X+r]
    blur = cv2.GaussianBlur(subimg, (5,5), 0)
    thresh = cv2.threshold(blur, 50, 255, cv2.THRESH_BINARY)[1]
    thresh = Threshold(blur, THRESHOLDNUMBER)
    im2, contours, heir = cv2.findContours(thresh, cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)

    areas = [cv2.contourArea(c) for c in contours]
    if not areas:
        END = True
```

```
            return
        max_idx = np.argmax(areas)
        cnt = contours[max_idx]

        M = cv2.moments(cnt)
        if M["m00"] == 0:
            M["m00"] = 1

        if M["m00"] != 0:
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            MaxLocShifted = (X-r+cX, Y-r+cY)
        else:
            MaxLocShifted = (X-r,Y-r)
        return MaxLocShifted

    def Gaussian(x, a, x0, Sigma):
        return a*np.exp(-(x-x0)**2/(2*Sigma**2))

    def GaussianFinder(MaxLoc, img):
        CoordinatesX = []
        CoordinatesY = []
        Range = np.arange(MaxLoc-10,MaxLoc+10)

        if MaxLoc == REFERENCESTARLOC[0]:
            Coordinates = img[REFERENCESTARLOC[1], MaxLoc-10:MaxLoc+10]
        if MaxLoc == REFERENCESTARLOC[1]:
            Coordinates = img[MaxLoc-10:MaxLoc+10, REFERENCESTARLOC[0]]
        if MaxLoc == OBJECTLOC[0]:
            Coordinates = img[OBJECTLOC[1], MaxLoc-10:MaxLoc+10]
        if MaxLoc == OBJECTLOC[1]:
            Coordinates = img[MaxLoc-10:MaxLoc+10, OBJECTLOC[0]]

        Mean = MaxLoc
        Sigma = 3
        try:
            FitParameters, pcov = curve_fit(Gaussian, Range, Coordinates,
                    p0 = [np.max(Coordinates), Mean, Sigma])
        except RuntimeError:
            FitParameters = [np.max(Coordinates),Mean, Sigma]

        return(FitParameters)
```

```
def MagnitudeFinder(Loc, XFitParameters, YFitParameters, img):
    global Radius, OBJECTBACKGROUNDRADIUS, REFERENCEBACKGROUNDRADIUS
    global OBJECTAVGRADIUS, REFERENCESTARAVGRADIUS
    YRadius = int(np.ceil(2*XFitParameters[2]))
    XRadius = int(np.ceil(2*YFitParameters[2]))
    Radius = max(XRadius,YRadius)

    Range=[]
    if Radius > 100:
        Radius = 100
    for i in range(-Radius,Radius):
        for j in range(-Radius,Radius):
            if i**2 + j**2  <  Radius**2:
                Range.append((i + Loc[0], j + Loc[1])[::-1])

    BackgroundRadius = Radius+5
    BackgroundRange=[]
    for i in range(-BackgroundRadius,BackgroundRadius):
        for j in range(-BackgroundRadius,BackgroundRadius):
            if i**2 + j**2  <  BackgroundRadius**2 and \
                i**2 + j**2 > Radius**2:
                BackgroundRange.append((i + Loc[0], j + Loc[1])[::-1])

    # BackgroundRange = np.array(BackgroundRange)
    # for i in BackgroundRange[:,0]:
        # if i > 479:
            # BackgroundRange[:,0] = 479
    # print(BackgroundRange)
    # BackgroundRange.tolist()
    # #print(BackgroundRange)

    BackgroundValues = []
    for i in BackgroundRange:
        BackgroundValues.append(img[i])
    AvgBackgroundMag = sum(BackgroundValues)/len(BackgroundValues)
    MagValue = 0
    RawMagValue= 0
    for i in Range:
        MagValue = MagValue + (img[i] - AvgBackgroundMag)
        RawMagValue = RawMagValue + img[i]
    if Loc == REFERENCESTARLOC:
```

```
            REFERENCESTARAVGRADIUS = Radius
            REFERENCEBACKGROUNDRADIUS = Radius+5
        if Loc == OBJECTLOC:
            OBJECTAVGRADIUS = Radius
            OBJECTBACKGROUNDRADIUS = Radius+5
        return(MagValue)

    def PlottingCurve(XFitParameters, YFitParameters, Radius, img,Folder):

        def getPlotSlice(XCent, YCent, Dir, PlotRange):
            """Function to return a single column or row sliced at
            the desired location
            and in the desired direction.
            XCent and YCent are integers and Dir is a stringof either "x" or "y"
            """
            # Gaussian positions are not integers so converting to integers
            # Extracting desired slice
            if Dir == 'x':
                return img[YCent,XCent-PlotRange:XCent+PlotRange]
            else:
                return img[YCent-PlotRange:YCent+PlotRange, XCent]

        XMaxLoc = int(XFitParameters[1])
        YMaxLoc = int(YFitParameters[1])

        # sns.set()
        # sns.set_style("dark")
        sns.set_context("poster")
        matplotlib.style.use("dark_background")
        gs = gridspec.GridSpec(2, 2, width_ratios=[2, 1], height_ratios=[2,1])
        ax = plt.subplot(gs[0, 0])

        ax1 = plt.subplot(gs[0])
        ax2 = plt.subplot(gs[1])
        ax3 = plt.subplot(gs[2])
        ax4 = plt.subplot(gs[3])

        PlotRange = 20

        # #Top Left
        # ax1.imshow(img,cmap='gray')
        # ax1.set_xlim(-PlotRange+OBJECTLOC[0],PlotRange+OBJECTLOC[0])
```

```
# ax1.set_ylim(-PlotRange+OBJECTLOC[1],PlotRange+OBJECTLOC[1])
# if XMaxLoc == int(XFitParametersObj[1]):
    # ax1.set_title('Magnitude of %s'%(round(ObjectCatalogValue,3)))
    # ax1.add_artist(plt.Circle((OBJECTLOC),OBJECTBACKGROUNDRADIUS,
        # color = 'yellow', alpha=.2))
    # ax1.add_artist(plt.Circle((OBJECTLOC),Radius,color='red',
        # alpha=0.2))
    # ax1.add_artist(plt.Circle((OBJECTLOC),.1, color = 'black',))
# if XMaxLoc == int(XFitParametersRef[1]):
    # ax1.set_title('Magnitude of %s' %(CatalogMagnitude))
    # ax1.add_artist(plt.Circle((REFERENCESTARLOC),Radius,
        # color='blue', alpha=0.2))
    # # ax1.add_artist(plt.Circle((REFERENCESTARLOC),
        # REFERENCEBACKGROUNDRADIUS,
        # color = 'yellow', alpha=.2))
# ax1.axis('off')

ax1.imshow(img,cmap='gray')
if XMaxLoc == int(XFitParametersObj[1]):
    ax1.set_xlim(-PlotRange+OBJECTLOC[0],PlotRange+OBJECTLOC[0])
    ax1.set_ylim(-PlotRange+OBJECTLOC[1],PlotRange+OBJECTLOC[1])
    # ax1.set_title('Magnitude of %s'%(round(ObjectCatalogValue,3)))
    ax1.add_artist(plt.Circle((OBJECTLOC),OBJECTBACKGROUNDRADIUS,
        color = 'yellow', alpha=.2))
    ax1.add_artist(plt.Circle((OBJECTLOC),OBJECTAVGRADIUS,color='red',
        alpha=.2))
    # ax1.add_artist(plt.Circle((OBJECTLOC),.1, color = 'black',))
if XMaxLoc == int(XFitParametersRef[1]):
    ax1.set_xlim(-PlotRange+REFERENCESTARLOC[0],
            PlotRange+REFERENCESTARLOC[0])
    ax1.set_ylim(-PlotRange+REFERENCESTARLOC[1],
            PlotRange+REFERENCESTARLOC[1])
    # ax1.set_title('Magnitude of %s' %(CatalogMagnitude))
    ax1.add_artist(plt.Circle((REFERENCESTARLOC),5,
        color='blue', alpha=0.2))
    ax1.add_artist(plt.Circle((REFERENCESTARLOC),REFERENCEBACKGROUNDRADIUS,
        color = 'yellow', alpha=.2))
ax1.axis('off')

#Top Right
if YMaxLoc == int(YFitParametersObj[1]):
    ax2.plot(getPlotSlice(OBJECTLOC[0],OBJECTLOC[1],'y',PlotRange),
```

```
            np.arange(-PlotRange+OBJECTLOC[1],PlotRange+OBJECTLOC[1],1),
            label='Data',color='orange')
    ax2.plot((Gaussian(np.arange(-PlotRange+OBJECTLOC[1],
        PlotRange+OBJECTLOC[1],1),*YFitParameters)),
            np.arange(-PlotRange+OBJECTLOC[1],PlotRange+OBJECTLOC[1],1),
            label='Gaussian Fit',color='red')
    ax2.set_xlabel('Pixel Value')
if YMaxLoc == int(YFitParametersRef[1]):
    ax2.plot(getPlotSlice(REFERENCESTARLOC[0],REFERENCESTARLOC[1],
        'y', PlotRange),
            np.arange(-PlotRange+REFERENCESTARLOC[1],
                PlotRange+REFERENCESTARLOC[1],1),
            label='Data')
    ax2.plot((Gaussian(np.arange(-PlotRange+REFERENCESTARLOC[1],
        PlotRange+REFERENCESTARLOC[1],1),*YFitParameters)),
            np.arange(-PlotRange+REFERENCESTARLOC[1],
                PlotRange+REFERENCESTARLOC[1],1),
            label='Gaussian Fit')
ax2.yaxis.set_visible(False)
ax2.xaxis.tick_top()
ax2.xaxis.set_label_position('top')

#Bottom Left
if XMaxLoc == int(XFitParametersObj[1]):
    ax3.plot(np.arange(-PlotRange+OBJECTLOC[0],PlotRange+OBJECTLOC[0]),
            getPlotSlice(OBJECTLOC[0],OBJECTLOC[1],'x',PlotRange),
            label='Data',color='orange')
    ax3.plot(np.arange(-PlotRange+OBJECTLOC[0],PlotRange+OBJECTLOC[0]),
            Gaussian(np.arange(-PlotRange+OBJECTLOC[0],
                PlotRange+OBJECTLOC[0]),*XFitParameters),
            label='Gaussian Fit',color='red')
    ax3.set_ylabel('Pixel Value')
if XMaxLoc == int(XFitParametersRef[1]):
    YMaxLoc = REFERENCESTARLOC[1]
    XMaxLoc = REFERENCESTARLOC[0]
    ax3.plot(np.arange(-PlotRange+XMaxLoc,PlotRange+XMaxLoc),
            getPlotSlice(XMaxLoc, YMaxLoc, 'x', PlotRange),
            label='Data')
    ax3.plot(np.arange(-PlotRange+XMaxLoc,PlotRange+XMaxLoc),
            Gaussian(np.arange(-PlotRange+XMaxLoc,PlotRange+XMaxLoc),
                *XFitParameters),
            label='Gaussian Fit')
```

```
    # ax3.legend(bbox_to_anchor=(0.,-.2, 1., .102), loc=3,ncol=2,
        # borderaxespad=0.)
    ax3.invert_yaxis()
    ax3.xaxis.set_visible(False)

    #Bottom Right
    ax4.imshow(img,cmap='gray')
    if XMaxLoc == int(XFitParametersObj[1]):
        ax4.axvline(OBJECTLOC[0], color='red')
        ax4.axhline(OBJECTLOC[1], color='red')
    # if XMaxLoc == int(XFitParametersRef[1]):
        # ax4.axvline(REFERENCESTARLOC[0], color='blue')
        # ax4.axhline(REFERENCESTARLOC[1], color='blue')
    ax4.axis('off')


    plt.subplots_adjust(wspace=.1)
    plt.subplots_adjust(hspace=.1)
    plt.draw()

    #Title
    # if XMaxLoc == int(XFitParametersObj[1]):
        # plt.suptitle('Object')
    # if XMaxLoc == int(XFitParametersRef[1]):
        # plt.suptitle('Reference Star')
    if XFitParameters[0] == XFitParametersObj[0]:
        plt.savefig(f'{Folder}/ObjectPlot{Iteration:03d}.png',
                bbox_inches='tight')
    if XFitParameters[0] == XFitParametersRef[0]:
        plt.savefig(f'{Folder}/ReferencePlot{Iteration:03d}.png',
                bbox_inches='tight')

def InitialRead(VideoName):
    Array = np.array([],ndmin=3)
    Video = cv2.VideoCapture(VideoName)
    print(VideoName)
    Grabbed, TheImage = Video.read()
    TheImage = cv2.cvtColor(TheImage, cv2.COLOR_BGR2GRAY)
    ImageStack = TheImage
    CondensedImage = np.array(TheImage,dtype=np.uint8)
    (Grabbed,TheImage) = Video.read()
    while Grabbed == True:
```

```
            TheImage = cv2.cvtColor(TheImage, cv2.COLOR_BGR2GRAY)
            ImageStack = np.dstack([ImageStack,TheImage])
            NumpyImage = np.array(TheImage,dtype=np.uint8)
            CondensedImage += NumpyImage
            (Grabbed,TheImage) = Video.read()
        CondensedImage = Image.fromarray(CondensedImage)
        CondensedImage.save('Stacked.png')
        return(ImageStack)



def main(vid_name,Folder,StartFrame, objectlocation, referencestarlocation,
            ThresholdNumber,TheCatalogMagnitude):
        global REFERENCESTARLOC
        global OBJECTLOC
        global REFERENCESTARAVGRADIUS
        global OBJECTAVGRADIUS
        global OBJECTBACKGROUNDRADIUS
        global REFERENCEBACKGROUNDRADIUS
        global THRESHOLDNUMBER
        global XFitParametersRef
        global YFitParametersRef
        global XFitParametersObj
        global YFitParametersObj
        global CatalogMagnitude
        global Iteration
        global ObjectMagValue

        record = True
        frame_no = 100
        CatalogMagnitude = float(TheCatalogMagnitude) #parameter cant be global

        REFERENCESTARLOC = referencestarlocation
        OBJECTLOC = objectlocation
        THRESHOLDNUMBER = ThresholdNumber

        vid = cv2.VideoCapture(vid_name)
        vid.set(1,StartFrame); # Where frame_no is the frame you want
        Grabbed, img = vid.read() # Read the frame
        #(Grabbed,img) = vid.read()
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
blurred = cv2.GaussianBlur(img, (5, 5), 0)
Thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]


# img2 = img.copy()
# cv2.namedWindow("window")
# cv2.setMouseCallback("window", click)
# cv2.imshow("window", img2)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# REFERENCESTARLOC = MaxFinder(REFERENCESTARLOC[0],
    # REFERENCESTARLOC[1],img)
# XFitParametersRef = GaussianFinder(REFERENCESTARLOC[0], img)
# YFitParametersRef = GaussianFinder(REFERENCESTARLOC[1], img)
# ReferenceMagValue = MagnitudeFinder(REFERENCESTARLOC,
        # GaussianFinder(REFERENCESTARLOC[0], img),
        # GaussianFinder(REFERENCESTARLOC[1], img), img)
# XFitParameters = (0,0,0)


ObjectMagValueList = []
ReferenceMagValueList = []
CatalogList = []
Instrument = []
Iteration = 1
while Grabbed == True:

    (Grabbed, img) = vid.read()
    if Grabbed == False:
        break
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    REFERENCESTARLOC = MaxFinder(REFERENCESTARLOC[0],
            REFERENCESTARLOC[1], img)
    if END == True:
        break
    XFitParametersRef = GaussianFinder(REFERENCESTARLOC[0], img)
    YFitParametersRef = GaussianFinder(REFERENCESTARLOC[1], img)
    ReferenceMagValue = MagnitudeFinder(REFERENCESTARLOC,
        GaussianFinder(REFERENCESTARLOC[0], img),
        GaussianFinder(REFERENCESTARLOC[1], img), img)
```

```
            ReferenceMagValueList.append(ReferenceMagValue)

            OBJECTLOC = MaxFinder(OBJECTLOC[0], OBJECTLOC[1], img)
            if END == True:
                break
            XFitParametersObj = GaussianFinder(OBJECTLOC[0], img)
            YFitParametersObj = GaussianFinder(OBJECTLOC[1], img)
            ObjectMagValue=MagnitudeFinder(OBJECTLOC,XFitParametersObj,
                    YFitParametersObj, img)
            if ObjectMagValue < 0:
                ObjectMagValue = 0
            ObjectMagValueList.append(ObjectMagValue)

            PlottingCurve(XFitParametersObj, YFitParametersObj, OBJECTAVGRADIUS,
                    img,Folder)
            PlottingCurve(XFitParametersRef, YFitParametersRef,
                    REFERENCESTARAVGRADIUS, img,Folder)
            Iteration += 1

        ReferenceMagValue = np.mean(ReferenceMagValueList)

        InstrumentalMagnitude = -2.5*np.log10(ReferenceMagValue)
        Offset = InstrumentalMagnitude - CatalogMagnitude

        for i in ObjectMagValueList:
            ObjectCatalogValue = -2.5*np.log10(i) - Offset
            CatalogList.append(ObjectCatalogValue)

        with open(f'{Folder}/MagValues.txt', 'w') as output:
            output.write(str(ObjectMagValueList))
        with open(f'{Folder}CatalogValues.txt', 'w') as output:
            output.write(str(CatalogList))

        plt.figure()
        # plt.plot(np.arange(0,len(ObjectMagValueList)),ObjectMagValueList)
        plt.plot(np.arange(0,len(CatalogList)),CatalogList)
        plt.xlabel('Frame')
        plt.ylabel('Magnitude')
        # plt.yticks([])
        plt.gca().invert_yaxis()
        plt.savefig(f"{Folder}/LightCurve.png", bbox_inches='tight')
    ##################################################
```

```
# open with "python /path/to/script.py --image /path/to/picture.jpg"
#####################################################
if __name__ == "__main__":
    ap=argparse.ArgumentParser()
    ap.add_argument("-i", "--video", required=True)
    args= vars(ap.parse_args())
    main(args['video'])
```

A.2 GUI Script

```
import tkinter as tk
from tkinter.filedialog import askopenfilename
import tkinter.ttk as ttk
from tkinter.constants import *
from PIL import Image, ImageTk
import glob
import Photometry
import os
import sys
import shutil
import time
import datetime
import numpy as np
import cv2


class GUI(tk.Frame):

    @classmethod
    def main(cls):
        root = tk.Tk()
        app = cls(root)
        app.grid(sticky=NSEW)
        root.geometry('800x500+50+50')
        root.grid_columnconfigure(0, weight=1)
        root.grid_rowconfigure(0, weight=1)
        root.mainloop()

    def __init__(self, root):
        super().__init__(root)
        self.master = root
```

```
        self.pack(fill=BOTH, expand=True)

        self.tk_setPalette( background = 'black',
                            foreground = 'gray80',
                            activeBackground = 'gray30',
                            selectColor = 'firebrick4',
                            selectBackground = 'firebrick4' )

        self.createwidgets()

    def createwidgets(self):
        self.Plotted = False
        self.ObjectClick = False
        self.ReferenceClick = False
        self.ThresholdToggle = False
        self.FrameNo = 1
        self.ThresholdCompleted = False
        self.ChoiceToggle = False
        # Frame01 = tk.Frame(self)
        # Frame01.pack(fill=X)



        # mess = r"""
 # ______ _           _           _ _                          _           _
# |  ____(_)         | |         | | |     /\                 | |         (_)
# | |__    _ _ __ ___| |__   __ _| | |    /  \   _ __   __ _| |_   _ ___ _ ___
# |  __|  | | '__/ _ \ '_ \ / _` | | |   / /\ \ | '_ \ / _` | | | | / __| / __|
# | |     | | | |  __/ |_) | (_| | | |  / ____ \| | | | (_| | | |_| \__ \ \__ \
# |_|     |_|_|  \___|_.__/ \__,_|_|_|_/_/    \_\_| |_|\__,_|_|\__, |___/_|___/
                                                                  # __/ |
                                                                 # |___/
                                                                 # """
        # self.Line=tk.Label(Frame01, justify=tk.LEFT, text=mess)
        # self.Line.pack()

        ######################################################
        Frame1 = tk.Frame(self)
        Frame1.pack(fill=X)
        ######################################################
        self.FolderName = tk.Entry(Frame1)
        self.FolderName.insert(0,
```

```
        f'{datetime.datetime.now().strftime("%Y%m%d_%H%M")}')
self.FolderName.pack(side=LEFT, fill=Y)

self.runButton = tk.Button(Frame1, text='RUN', width=7,
        height=1, command=self.run, bg='black',state=DISABLED)
self.runButton.pack(side=LEFT)

self.openButton = tk.Button(Frame1, text='OPEN', width=7,
        height=1, command=self.open, bg='black')
self.openButton.pack(side=LEFT)

# self.ProgressLabel = tk.Button(Frame1, text="Not Started")
# self.ProgressLabel.pack(side=LEFT)

# self.ObjectLabel = tk.Button(Frame1, text="Object:( X , Y )")
# self.ObjectLabel.pack(side=LEFT)

# self.ReferenceLabel = tk.Button(Frame1, text="Reference:( X , Y )")
# self.ReferenceLabel.pack(side=LEFT)

self.restartButton = tk.Button(Frame1, text='RESET', width=7,
        height=1, command=self.restart, bg='black')
self.restartButton.pack(side=LEFT)

self.QuitButton = tk.Button(Frame1, text="QUIT",
        command=Frame1.quit, width=7)
self.QuitButton.pack(side=LEFT)

#####################################################
Frame2 = tk.Frame(self)
Frame2.pack(fill=X)
#####################################################
self.NameButton = tk.Button(Frame2, text="POSITION", width=7)
self.NameButton.pack(side=LEFT)

self.Choice=tk.IntVar()
self.ObjectLabel = tk.Radiobutton(Frame2, text="Object: ( X , Y )",
        variable=self.Choice, value=0, indicatoron=0,
        command=self.ObjectChoice)
self.ReferenceLabel = tk.Radiobutton(Frame2,
        text="Reference Star: ( X, Y )", variable=self.Choice, value=1,
        indicatoron=0,command=self.ObjectChoice)
```

```
self.ObjectLabel.pack(side=LEFT,fill=Y)
self.ReferenceLabel.pack(side=LEFT,fill=Y)

self.CatalogValue = tk.Entry(Frame2)
self.CatalogValue.insert(0,'0')
self.CatalogValue.pack(side=LEFT, fill=Y)


#####################################################
Frame2_1 = tk.Frame(self)
Frame2_1.pack(fill=X)
#####################################################
self.ThresholdButton = tk.Button(Frame2_1, text="THRESHOLD", width=7)
self.ThresholdButton.pack(side=LEFT)

self.ThresholdVariable=tk.IntVar()
self.ThresholdOn = tk.Radiobutton(Frame2_1, text="ON",
        variable=self.ThresholdVariable, value=1, indicatoron=0,
        command=self.ThresholdRadio)
self.ThresholdOff = tk.Radiobutton(Frame2_1,
        text="OFF", variable=self.ThresholdVariable, value=0,
        indicatoron=0, command=self.ThresholdRadio)
self.ThresholdOn.pack(side=LEFT,fill=Y)
self.ThresholdOff.pack(side=LEFT,fill=Y)


#####################################################
Frame3 = tk.Frame(self)
Frame3.pack(fill=X)
#####################################################

self.CamView = []
self.canvas = tk.Canvas(Frame3, width = 720, height=480, bg='black')
self.canvas.pack(side=LEFT)
self.ImageCanvas= self.canvas.create_image(0,0, anchor=NW,
        image=self.CamView)
self.canvas.bind("<Button-1>",self.ReferenceCoordinates)
self.canvas.bind("<Button-3>",self.ObjectCoordinates)

self.LightCurve = tk.Label(Frame3, image="")
self.LightCurve.pack(side=LEFT)
```

```
        self.display = tk.Label(Frame3, image="")
        self.display.pack(side=LEFT)
        ##################################################
        Frame4 = tk.Frame(self)
        Frame4.pack(fill=X)
        ##################################################


        self.slidervar = tk.IntVar()
        self.InitialSlider = tk.Scale(Frame4, from_=0, to=100,
                orient=HORIZONTAL, label='Initial Frame',
                command=self.VideoLength, variable=self.slidervar,length=360)
        self.InitialSlider.pack(side=LEFT)

        self.thresholdvar = tk.IntVar()
        self.ThresholdSlider = tk.Scale(Frame4, from_=0, to=255,
                orient=HORIZONTAL, label='Threshold',
                command=self.Threshold,variable=self.thresholdvar,length=360)
        self.ThresholdSlider.pack(side=LEFT)

        self.var = tk.IntVar()
        self.FrameSlider = tk.Scale(Frame4, from_=1, to=2,
                orient=HORIZONTAL, length = 360, command=self.PlotFrame,
                variable=self.var, label='Plot Frame' )
        self.FrameSlider.pack(side=LEFT)


##################################################
# Functions
##################################################

    def PlotFrame(self,event):
        FrameNumber = self.var.get()
        if self.ChoiceToggle == False:
            print('False')
            PlotPath= f'{self.FolderPath}/ObjectPlot{FrameNumber:03}.png'
            self.PlotImage = Image.open(PlotPath)
            self.PlotImage = self.PlotImage.resize((360,360),Image.ANTIALIAS)
            self.PlotImage = ImageTk.PhotoImage(self.PlotImage)
            self.display.config(image=self.PlotImage)
        if self.ChoiceToggle == True:
            print('True')
```

```
            PlotPath= f'{self.FolderPath}/ReferencePlot{FrameNumber:03}.png'
            self.PlotImage = Image.open(PlotPath)
            self.PlotImage = self.PlotImage.resize((360,360),Image.ANTIALIAS)
            self.PlotImage = ImageTk.PhotoImage(self.PlotImage)
            self.display.config(image=self.PlotImage)

    def ReferenceCoordinates(self,event):
        self.ReferenceLabel.configure(text=f"Reference: ({event.x},{event.y})")
        self.REFERENCESTARLOC = (event.x,event.y)
        self.ReferenceLabel.configure(bg ='white',fg = 'black')
        self.ReferenceClick = True
        if self.ObjectClick == True and self.ThresholdCompleted == True:
            self.runButton.config(state='normal')

    def ObjectCoordinates(self,event):
        self.ObjectLabel.configure(text=f"Object: ({event.x},{event.y})")
        self.OBJECTLOC = (event.x,event.y)
        self.ObjectLabel.configure(bg ='white',fg ='black')
        self.ObjectClick = True
        if self.ReferenceClick == True and self.ThresholdCompleted == True:
            self.runButton.config(state='normal')

    def ObjectChoice(self):
        if self.Choice.get() == 0:
            print('0')
            self.ChoiceToggle = False
        if self.Choice.get() == 1:
            print('1')
            self.ChoiceToggle = True

    def ThresholdRadio(self):
        self.ThresholdSelection = self.ThresholdVariable.get()
        if self.ThresholdSelection == 0:
            self.ThresholdToggle = False
        if self.ThresholdSelection == 1:
            self.ThresholdToggle = True

    def VideoLength(self,event):
        self.FrameNo = self.slidervar.get()
        if self.ThresholdToggle == False:
            self.CamView = Image.fromarray(self.ImageStack[:,:,self.FrameNo])
            self.CamView = ImageTk.PhotoImage(self.CamView)
```

```
            self.canvas.itemconfig(self.ImageCanvas,image=self.CamView)
        if self.ThresholdToggle == True:
            self.ThresholdCompleted = True
            self.ThresholdNumber = self.thresholdvar.get()
            ThresholdView = Photometry.Threshold(self.ImageStack[:,:,
                self.FrameNo], self.ThresholdNumber)
            self.CamView = Image.fromarray(ThresholdView)
            self.CamView = ImageTk.PhotoImage(self.CamView)
            self.canvas.itemconfig(self.ImageCanvas,image=self.CamView)


    def Threshold(self,event):
        self.ThresholdToggle = True
        self.ThresholdNumber = self.thresholdvar.get()
        ThresholdView = Photometry.Threshold(self.ImageStack[:,:,self.FrameNo],
                self.ThresholdNumber)
        self.CamView = Image.fromarray(ThresholdView)
        self.CamView = ImageTk.PhotoImage(self.CamView)
        self.canvas.itemconfig(self.ImageCanvas,image=self.CamView)
        self.ThresholdCompleted = True

    def open(self):
        self.VideoName = tk.filedialog.askopenfilename(initialdir = 'Data')
        self.VideoPath = '/'.join(self.VideoName.split('/')[:-1])
        self.FolderDirectory = (self.FolderName.get())
        self.FolderPath = f'{self.VideoPath}/{self.FolderDirectory}'
        if os.path.exists(self.FolderPath):
            shutil.rmtree(self.FolderPath)
        os.makedirs(self.FolderPath)

        self.ImageStack = Photometry.InitialRead(self.VideoName)
        VideoSize = self.ImageStack.shape[2]
        self.InitialSlider.config(to=VideoSize-1)
        self.CamView = Image.fromarray(self.ImageStack[:,:,0])
        # CamView = self.CamView.resize((640,480),Image.ANTIALIAS)
        self.CamView = ImageTk.PhotoImage(self.CamView)
        self.canvas.itemconfig(self.ImageCanvas,image=self.CamView)
        self.openButton.configure(bg='white', fg='black')
        # self.runButton.configure(bg='black',fg='white')

    def run(self):
        self.Catalog = self.CatalogValue.get()
```

```
            print(self.Catalog)
            # self.runButton.configure(text="Running")
            Photometry.main(self.VideoName,self.FolderPath,self.FrameNo,
                    self.OBJECTLOC,self.REFERENCESTARLOC,
                    self.ThresholdNumber,self.Catalog)
            lightcurvepath= f'{self.FolderPath}/LightCurve.png'
            self.OG = Image.open(lightcurvepath)
            ResizedOG = self.OG.resize((360,240),Image.ANTIALIAS)
            self.IMG = ImageTk.PhotoImage(ResizedOG)
            self.LightCurve.config(image=self.IMG)
            self.runButton.configure(bg = 'white', fg='black')
            MaxFrameNumber= len(glob.glob(f"{self.FolderPath}/ObjectPlot*.png"))
            self.FrameSlider.config(to=MaxFrameNumber)
            self.restartButton.configure(bg ='firebrick4')

            FrameNumber = self.var.get()
            PlotPath= f'{self.FolderPath}/ObjectPlot{FrameNumber:03}.png'
            self.PlotImage = Image.open(PlotPath)
            self.PlotImage = self.PlotImage.resize((360,360),Image.ANTIALIAS)
            self.PlotImage = ImageTk.PhotoImage(self.PlotImage)
            self.display.config(image=self.PlotImage)

        def restart(self):
            python = sys.executable
            os.execl(python, python, * sys.argv)

    if __name__ == '__main__':
        GUI.main()
```

Bibliography

[Alc] Alcor, *OMEA All Sky Camera*.

[Ban12] Steven Bannister, *New Calibration Technique for All Sky Cameras*, Ph.D. thesis, New Mexico State University, 2012, p. 152.

[Bot07] WF Bottke, *Understanding the near-Earth object population: the 2004 perspective*, Comet/asteroid Impacts and Human Society: An Interdisciplinary Approach (Hans Rickman Peter T. Bobrowsky, ed.), no. 2003, Springer-Verlag Berlin Heidelberg, 2007.

[BVRN06] William F. Bottke, David Vokrouhlický, David P. Rubincam, and David Nesvorný, *THE YARKOVSKY AND YORP EFFECTS: Implications for Asteroid Dynamics*, Annual Review of Earth and Planetary Sciences **34** (2006), no. 1, 157–191.

[ESA17] ESA, *Hypervelocity Impacts and Protecting Spacecraft*, 2017.

[Har08] Jennifer Harbaugh, *Intro to Meteors and the NASA All-Sky Camera Net- work, The Fireball Project*, 2008.

[HGB96] Ian Halliday, Arthur A. Griffin, and Alan T. Blackwell, *Detailed data for 259 fireballs from the Canadian camera network and inferences concerning the influx of large meteoroids*, Meteoritics & Planetary Science **31** (1996), no. 2, 185–217.

[Jen06] Peter Jenniskens, *Meteor Showers and their Parent Comets*, 2006.

[JGD+11] P. Jenniskens, P.S. Gural, L. Dynneson, B.J. Grigsby, K.E. Newman, M. Borden, M. Koop, and D. Holman, *CAMS: Cameras for Allsky Meteor Surveillance to establish minor meteor showers*, Icarus **216** (2011), no. 1, 40–61.

[MG05] S. Molau and P.S. Gural, *A review of video meteor detection and analysis software*, WGN **33** (2005), 15–20.

[MM95] Michèle Moons and Alessandro Morbidelli, *Secular Resonances in Mean Motion Commensurabilities: The 4/1, 3/1, 5/2, and 7/3 Cases*, Icarus **114** (1995), no. 1, 33–50.

[R R17] T.J. Jopek R Rudawska, Z Kanuchova, *Meteor Data Center list of Meteor Showers*, 2017.

[RR15] J. Jedediah Rembold and Eileen V. Ryan, *Characterization and Analysis of Near-Earth Objects via Lunar Impact Observations*, Planetary and Space Science **117** (2015), 119–126.

[SEM17] R. M. Suggs, S. R. Ehlert, and D. E. Moser, *A comparison of radiometric calibration techniques for lunar impact flashes*, Planetary and Space Science **143** (2017), 225–229.

[Sky] LLC SkySentinel, *SkySentinel Network*.

[SMCS14] R. M. Suggs, D. E. Moser, W. J. Cooke, and R. J. Suggs, *The flux of kilogram-sized meteoroids from lunar impact monitoring*, Icarus **238** (2014), 23–36.

[Ste96] Duncan Steel, *Meteoroid orbits*, Space Science Reviews **78** (1996), no. 3-4, 507–553.

[TRML+07] Josep M. Trigo-Rodríguez, José M. Madiedo, Jordi Llorca, Peter S. Gural, Pep Pujols, and Tunc Tezel, *The 2006 Orionid outburst imaged by all-sky CCD cameras from Spain: meteoroid spatial fluxes and orbital elements*, Monthly Notices of the Royal Astronomical Society **380** (2007), no. 1, 126–132.

[TRMW+09] Josep Ma Trigo-Rodríguez, José M. Madiedo, Iwan P. Williams, Alberto J. Castro-Tirado, Jordi Llorca, Stanislav Vítek, and Martin Jelínek, *Observations of a very bright fireball and its likely link with comet C/1919 Q2 Metcalf*, Monthly Notices of the Royal Astronomical Society **394** (2009), no. 1, 569–576.

[Whi51] Fred Whipple, *A Comet Model. II. Physical Relations for Comets and Meteors*, Astrophysical Journal **113** (1951), 464.