# Drone Calibration Device in the Optical Spectrum

**Declan Baker**

**Under the guidance of:**
**Dr. Bradley Johnson**

**Date:** May 7, 2024

**Department of Astronomy**
**University of Virginia**

**This thesis is submitted in partial completion of the requirements of the:**
**BS Astronomy-Physics Major**

# 1 Overview and Motivation

## 1.1 Abstract

This project is a drone calibration device to find the orientation of the polarization of light in the optical spectrum. It is a proof of concept project; therefore, if it works in the optical spectrum it will then be done for millimeter wavelength light. Ultimately, the goal of this project will be to use the polarization of the ambient background of millimeter wavelength light to determine if gravitational waves from the early universe can be detected.

The project itself is devised of two main instruments: the polarimeter and the drone calibration source. The polarimeter, in order of light hitting the instrument, consists of a rotating half wavelength plate, a linear polarizer, a lens, and a camera. Additionally, there is an encoded motor that rotates the half wavelength plate to vary the intensity of a specific frequency of light. As the half-wavelength plate is rotated, it rotates the polarization of the light; therefore, allowing more or less light in through the linear polarizer. The half-wavelength plate is set for a specific frequency so it doesn't block all light just light at a specific frequency, here the frequency lines up with the color green. Thus, the goal of the polarimeter is to show the correlation of the calibration source's intensity versus motor angle position, which in theory will create a sine wave. Then one can use the phase of this sine wave to find the orientation of the polarization of the light. The second half of the project is the calibration device, an LED of the same frequency as the half wavelength plate mounted on a drone along with another camera and a beamsplitter. The drone acts as a calibration source for the polarimeter that is at an "infinite" distance away from the polarimeter. Thus, the polarization of the source can then be used as a standard measuring tool against the ambient background light. Through using the orientation of the drone and the polarization of the background light, the effects of gravitational waves can possibly be detected. The future goal for this project is upgrading both the drone to a CubeSat that will orbit the Earth and the polarimeter to a finer tuned polarimeter, possibly one at the Simon's Observatory Telescope.

## 1.2 Why One Would Care

The Cosmic Microwave Background, CMB, is light that developed around 380,000 years after the Big Bang. [27] Around this time the early Universe was cold enough that particles such as protons and neutrons could start forming. Therefore, studying this light one can understand more about how the early Universe evolved. Specifically, this project would ultimately point in the direction of whether one can detect gravitational waves generated from inflation.

Understanding the early Universe can help humanity find deeper insight into new realms of physics, such as dark energy, dark matter, gravitational waves, particle physics, plasma physics, and many more sub-fields. [27] These new discoveries can help humanity evolve, creating new technology and philosophy. For

example, during the Cold War both the USA and USSR used prior knowledge of how stars collapsed and radiation pressure to help create the atomic bomb. This lead to a new invention of the atomic bomb and a more important discovery: the philosophy of mutually assured destruction. [31]

Additionally, this project is happening at a timely manner because currently the Simon's Observatory Telescope is being constructed. This Telescope will be the deepest probe and highest resolution of the CMB.[2] Thus, having an accurate calibration source would be very beneficial in analyzing the CMB data collected by the Simon's Observatory Telescope.

## 2 SolidWorks

In order to accomplish this large project planning and designing were fundamental cornerstones. SolidWorks[1] 2022, a computer aided design (CAD) software, allows its user to make a 3D model of the project, which is helpful for visualizing and piecing together the project. The user can create unique parts, import existing parts, assemble the parts together, and look for optimization and flaws in the design. In this project two assemblies were created: one for the polarimeter setup and one for the drone calibration setup.

Starting with the polarimeter, the chassis, stepper motor supporting parts, the belt, and both gears were imported from ServoCity[25] that were necessary for building the structure. SolidWorks allows the user to create parts as well so the base plate, supporting rods, the camera model, the lens model, the camera-lens adapter, and a unique part that is used to mount the camera to the polarimeter as parts for the assembly were crafted by hand. The Raspberry Pi 4 (RPI4), the NEMA17-AMT112S bipolar stepper motor, and the M4 screws were downloaded for free from GrabCAD[10]. Then these parts were assembled all together in a single file. Look at Figure 1 and Figure 2 to see the SolidWorks CAD design of the polarimeter setup.

On the drone side, the Cage Cube-Mounted Wire Grid Beamsplitter Cube SolidWorks file was imported from THORLABS[30]. The green LED and the M4 screws were imported from GrabCAD[10]. The same RPI 4 model, camera model, lens model, and lens adapter as was used in the polarimeter were also used in this model. The rest of the assembly was crafted by hand, which includes the walls, supporting beams, the drone legs, and the beamsplitter holder along with the rods to hold that together. Then these were all assembled together in a single file. Look at Figure 3 and Figure 4 to see the SolidWorks CAD design of the drone setup.

SolidWorks is a useful tool because of the portability of a complex idea. A full setup can be designed and sent to the others for recommendations and suggestions. Additionally, it can be sent to a machinist for easy implementation into the machines, guiding them on the precise cutting of parts.

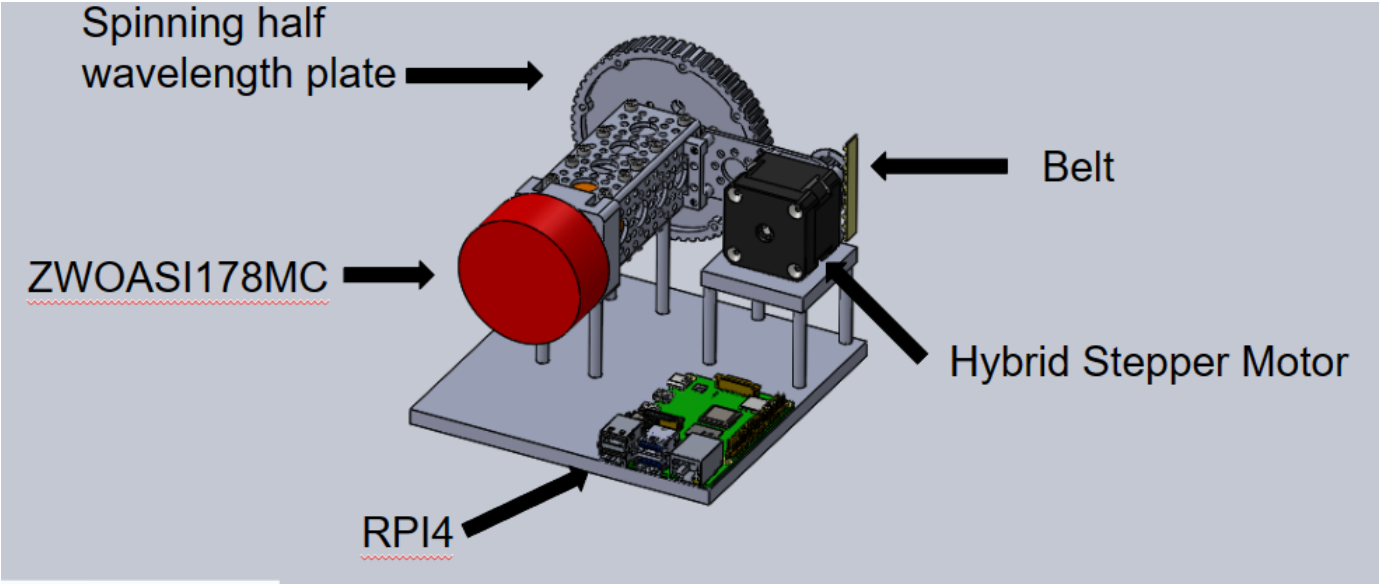


Figure 1: This is the backside image of the polarimeter. From this one can see the bipolar stepper motor also called a hybrid stepper motor, the Raspberry Pi 4, the ZWOASI178MC camera, the half-wavelength plate, the belt, and the infrastructure used to combine and support it.

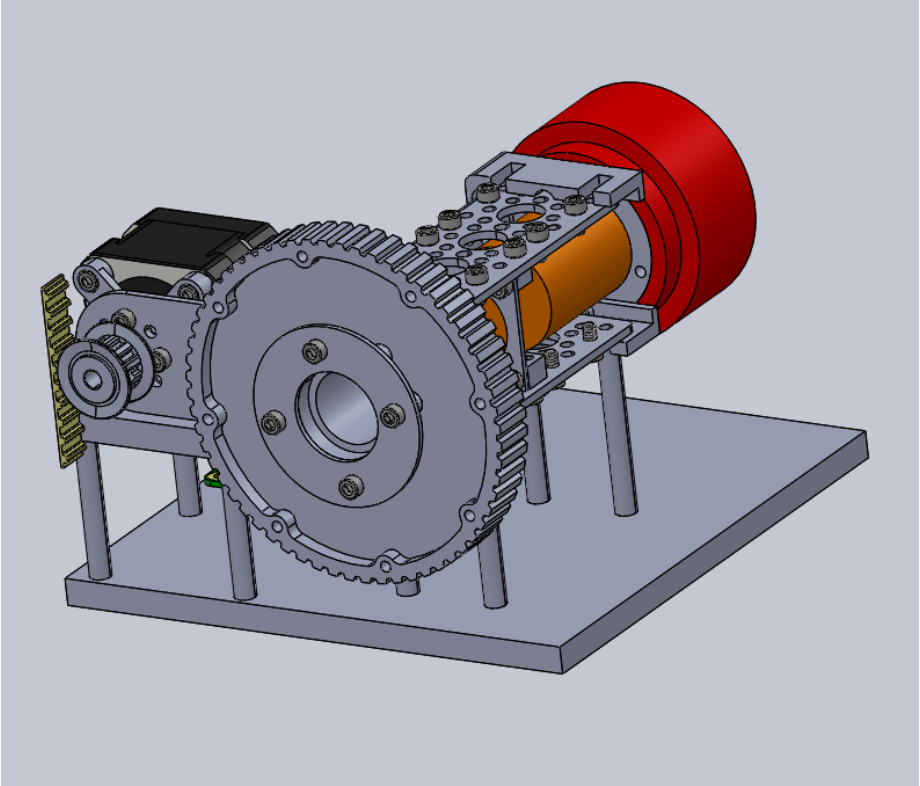

Figure 2: This is the front side image of the polarimeter. This angle highlights the intricacies of the half-wavelength plate and gear as well as the infrastructure. It also shows the relationship of the size of the two gears in the front.

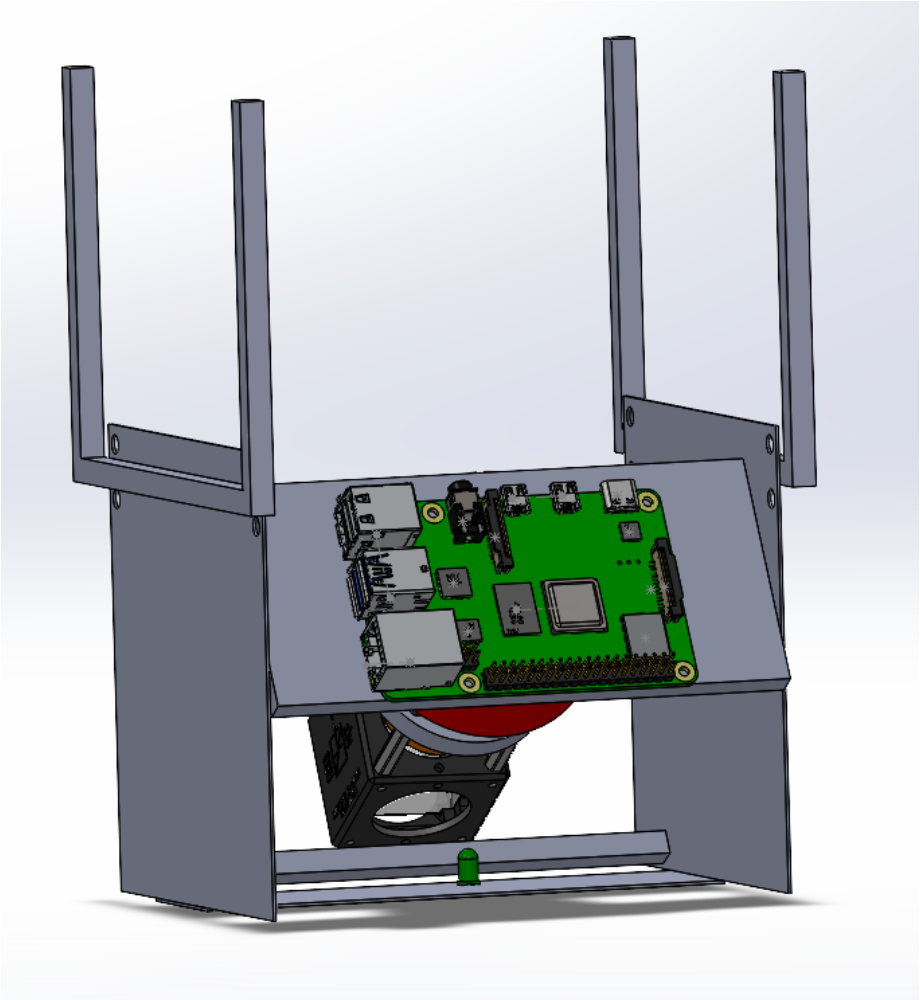

Figure 3: From this backside image of the drone calibrater one can see that the Raspberry Pi 4 is attached to the back. The camera is at a 45 degree below the horizon. There is the green LED on the underside in the middle. The two large "U-shaped" pieces represent the legs of the drone it will be attached to.

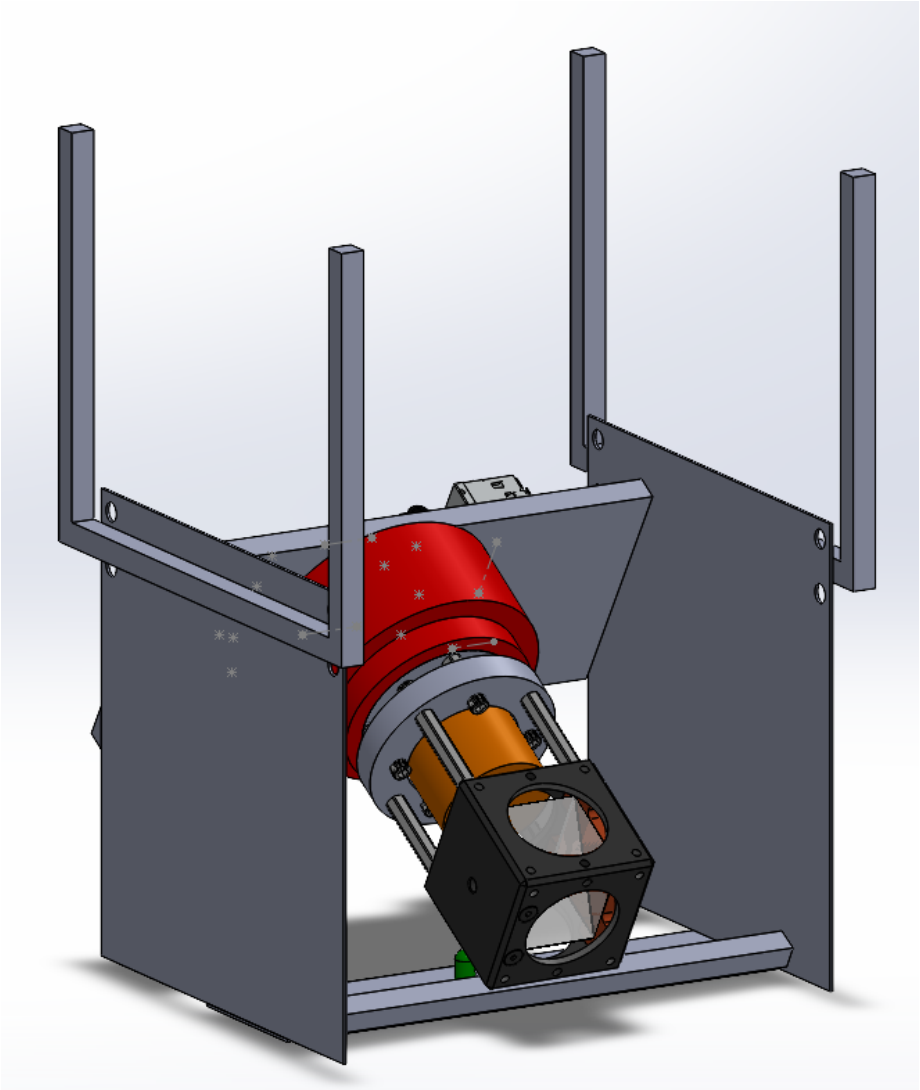

Figure 4: This front side shot of the drone calibrater highlights the intricacies of the way that the beamsplitter is attached to the camera. Also it is easier to see that the camera beamsplitter setup is at a 45 degree angle below the horizon. It can also be seen that there is a supporting beam holding the beamsplitter in place.

# 3 Optics

The path of light in order from source to camera starts with the source then goes through the beamsplitter. Next, it travels some distance before hitting the half-wavelength plate then the linear polarizer, lens, and camera, respectively. The source of light is a green LED, officially called "LED GREEN CLEAR 5MM ROUND T/H" and can be found at DigiKey[8]. The beamsplitter is the Cage Cube-Mounted Wire Grid Beamsplitter Cube from THORLABS[30], the same one downloaded in the drone CAD model. The lens is the FUJINON HF50HA-1S[12]. The camera is a ZWOASI178MC[7].

The source is below the beamsplitter; therefore, the beamsplitter is changing the angle of light by 90 degrees so it will be in the line of sight of the polarimeter. The beamsplitter also acts as a linear polarizer so the light from the source will be at a unique polarization, although this polarization is unknown. To get an intuition for what is going on, let's say the polarization that is allowed by the beamsplitter is in the North-South direction. The spinning half-wavelength plate rotates the polarized light. So for example, that North-South polarization will get rotated to a unique angle with respect to the half-wavelength plate. Then the second linear polarizer will the allow light in at a distinct polarization. Again for simplicity lets say this second polarizer is set for East-West polarization. Initially, the North-South polarized light will not be able to get through. Although, as the light is rotated by the half-wavelength plate then slowly more and more light will get through until it is all rotated to East-West then all the North-South polarized light will get through. As it continues to rotate it will allow less and less light until it is again North-South polarized. Therefore, the camera will capture the brightness of the green LED light versus half-wavelength plate angle as a sine wave. This is a simplified overview of what is going on. It reality it will have four peaks and four troughs because the half-wavelength plate is birefringent, more on this under the half-wavelength plate section. See Figure 5 to see the lens and camera. See Figure 6 and Figure 7 to see the path of light.

## 3.1 Camera

Deciding on a camera ultimately came down to a subset of features: retrieving a fits file, compatibility with the microcontroller, quality of sensor, frames per second (FPS), and flexibility of the features. These important features slowly bubbled to the surface through trial and error.

The first family of cameras were ArduCam[6] and PiCameras sold through adafruit[15], which are designed for RPIs specifically. Two examples that were tried or looked into were Arducam Ultra Low Light 2MP IMX462 STARVIS Camera, 1080P HD Wide Angle Pivariety Camera Module for Raspberry Pi[5] and Arducam USB Low Light Wide Angle Camera[4]. This was an obvious first choice since the hardware was compatible. However, it quickly became noticeable that the features of the camera were restrictive. The software that came with the camera was difficult to install and use. Although, the real problem

Figure 5: Here one can see the ZWOASI178MC camera with the FUJINON HF50HA-1S and the linear polarizer on the end. The blue cord is the USB-B 3.0 on the camera side and a USB-A 3.0 which connects to the Raspberry Pi 4.

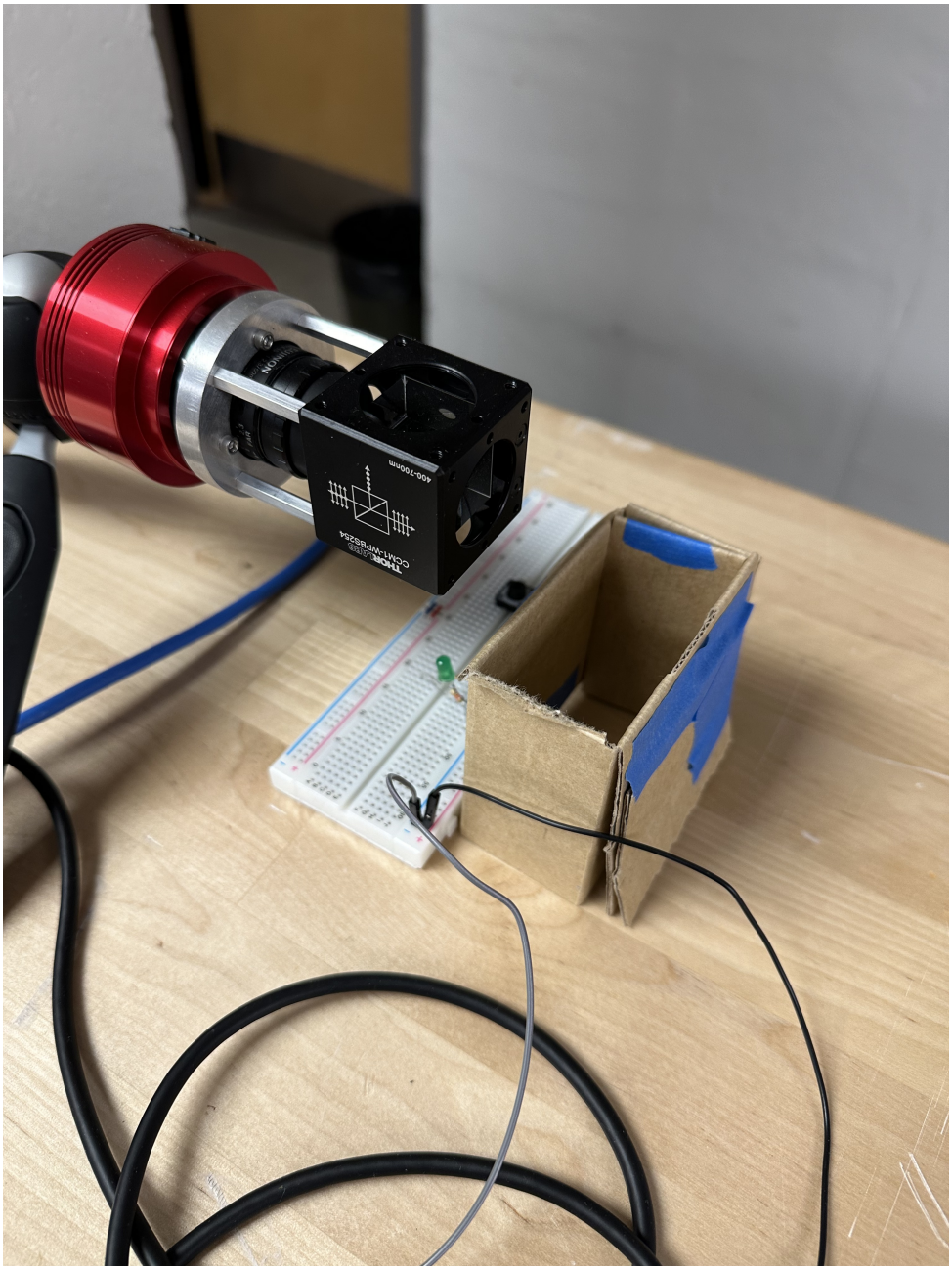


Figure 6: This is the start of the light's path. It starts from the LED and goes up through the beamsplitter and out toward the right of this picture where it eventually will hit the polarimeter. The beamsplitter here acts as a linear polarizer also. See Figure 7 for the rest of the light's path.

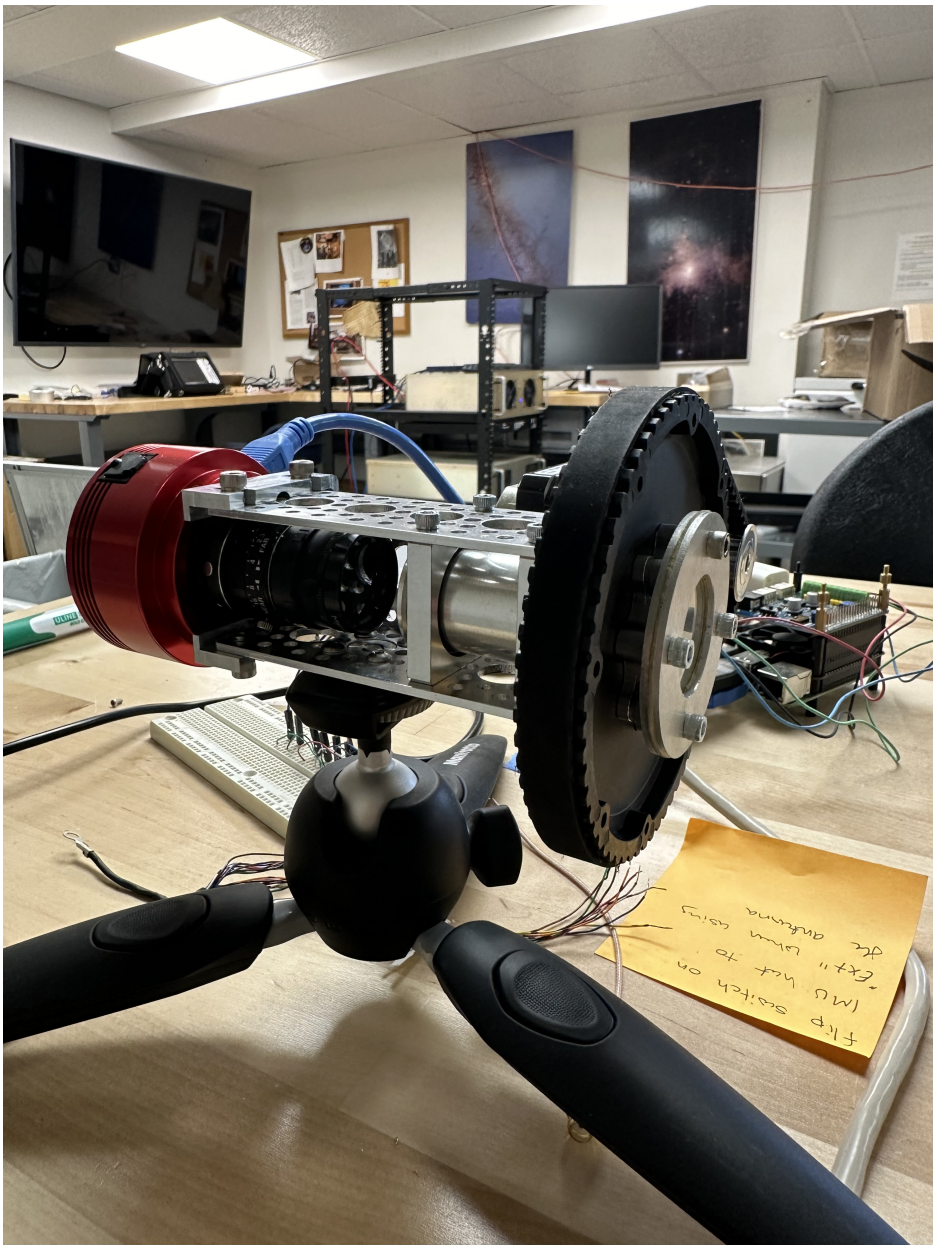

Figure 7: The light comes from the source and through the beamsplitter. See Figure 6 for the first half of the light's path. This picture looks flipped compared to Figure 6, but that was done intentionally to show the inside of the infrastructure and make the path of light inside the polarimeter clearer. So in this example the light is coming from the right. It then first hits the half-wavelength plate, which is the small circle sticking out of the first gear. It then goes down the tube and hits the polarizer, the lens, and the camera, respectively.

was that there was very little ability to adjust the features of the camera from a software side. Although, for certain cameras the focus could be adjusted manually on the outside of the camera. One of the features of the camera that became the biggest problem was not being able to retrieve a fits file. When not using a raw data format, the raw data gets modified in order to effectively transport the data to storage. Although this is more effective, this defeats the purpose of the experiment. The raw pixel count is very vital for the science we are trying to accomplish. (There will be more on compression data below.) Some other notable features in the ArduCam or PiCamera families that were not available to change or were statically set deep in the code were: exposure time, FPS, and gain. In the end, we landed on the ZWOASI178MC camera.[7]

The exposure time is highly critical for determining how much light can get through. Now depending on the camera there are different depths of electron wells. After these wells are over filled you will start to see blooming or over saturation which leads to skewed data. For the experiment at hand, the exposure time needed to be able to be adjusted according to the amount of background light that was available. This is mostly important for later on down the line when astrometry is done on the images. If there are not enough stars, there is over saturation from the source, or over saturation from a bright star in the background then it will make it hard for astrometry to be calculated correctly. Adjusting the exposure time can very easily fix this problem. The Arducam and PiCamera families made it very difficult to change these features. However, the ZWOASI178MC camera has a exposure time range of 32 microseconds to 1000 seconds. [7] See Figure 22 to see how to do this in Kstars.

The frames per second, FPS, is critical for determining the refresh rate of the camera. This is useful for quick successive pictures at low exposure times. For example, if the FPS is 4, meaning it refreshes 4 times a second or every 0.25 seconds, and the exposure time is 0.1 seconds then two pictures could easily be taken at the same frame. For this experiment, the data would then show that there is the same pixel value at two neighboring motor angle, which from the physics of the half-wavelength plate is impossible. Thus the higher the FPS then the more accurately the correct pixel value is to the corresponding motor angle. For example, with 60 FPS then at most there is a 0.01667 second delay between the taken pixel value and the theoretically correct pixel value. Also this value is not consistent for every picture due to exposure time and download time, so it cannot be uniformly subtracted out. Again for the PiCamera and Arducam families there was trouble with getting these cameras' software to work smoothly with the RPI. The standard FPS for these cameras was concerningly low, which means that the delay between the correct pixel value and taken pixel value was high and would skew the data. The ZWOASI178MC camera has 60 FPS, which is suitable for this application. [7] See Figure 22 to see how to do this in Kstars.

Lastly, the gain is a feature that can change the weight of each individual electron. For example, if the source is incredibly dim then the gain can be increased to 10 or even 100 so each electron counts as 10 or 100, respectively in the picture. In a lot of cameras the gain is already set at a distinct non-zero

value (at least that is what was found with ArduCam), but in order to find the true raw data it was vital that the gain was set to zero. The ZWOASI178MC camera allows you to easily set the gain. See Figure 22 for how this is done in Kstars.

Now on top of already not having these variable features, the sensors were not as high quality as one could get for a relatively decent price. For example, in the Arducam USB Low Light Wide Angle Camera the sensor is 2MP Sony IMX291[14]. While the in the ZWOASI178MC camera has the Sony IMX178 Sensor[26]. First just looking at resolution, which is indicative of how many pixels the sensor has, the IMX178 has 6.4 Megapixels while the IMX291 has only 2 Megapixels. Both of these sensors have the same pixel size of 2.9 micrometers by 2.9 micrometers. The pictures that are captured need to have astrometry done on them, therefore the quality and accuracy of the location of the star depends on the quality of the resolution. Another way of saying this is that the better resolution means smaller error bars.

## 3.2 Compressed Files versus Lossless Files

The most important reason for choosing the ZWOASI178MC camera is the benefit of having the output file be a fits file. Raw, uncompressed data is vital for accomplishing our scientific goal. There are only a few data types that collect and store raw electron count per pixel without using interpolation or compression. A common raw form of collecting data is using a fits file. The benefit to these is that they store this raw, uncompressed data; however, the downside is that the file is very large. This leads to slow download speeds as well, which can slowdown the rate at which the pictures can be taken. In other words it can create a bottle-neck. This trade off was worth taking in this scenario, due to the benefit of collecting raw data. Here initially we looked into getting a video of the data, but found it more practical to use multiple sequential pictures. Thus below video codecs and single frame codecs are discussed.

An good question to ask is how much does compression affect the data? Is it worth going through the hassle of finding a good camera with compressionless data storage when there are cheap cameras that use compressed data storage? In order to fully answer this question it is valuable to dive into the details of what compression does. This is a very generalized approach, but it highlights enough of the problem to give an educated answer. When a photon of light hits a pixel on the sensor of a camera it excites an electron in the silicon giving it enough energy to escape from the silicon. This is an example of the Photoelectric effect. Now this electron can be recaptured by another layer of metal and measured as a voltage. Therefore, a pixel effectively converts light to voltage in an equivalent amount. To convert this voltage to binary code that the computer can read a ADC is used, an analog to digital converter. An ADC creates no or very minimal loss over all so not a lot of time will be spent here. To convert back to an image that is displayed on the screen of a computer a DAC is used, a digital to analog converter. This is the same idea as an ADC but in reverse. In between the ADC and the DAC is where the interesting bits happen. This is due to wanting to

move a large file around quickly so it is smart for most applications to compress the file, move the file, then decompress the file. However, through this process there are key features that are changed. For a picture of a dog this is completely fine because the computer knows how to interpret, interpolate, and smooth the image to make it look like a dog, but for a data set this will introduce a lot of unnecessary rounding that will skew the data. See Figure 8 for a pipeline of compression and decompression for a typical camera.

When looking at some of the Arducam cameras or PiCamera cameras it will say what types of data formats they have. For example, the Arducam USB Low light Wide Angle Camera says it supports three data formats MJEPG (compressed), H.264 (compressed), and YUY2 (uncompressed). MJEPG means it takes every frame and compresses it individually as a JPEG so there is no time component that is maintained from one picture to another. Now JPEG uses Chroma Subsampling to compress its data, which basically takes a small subset of pixels lets say 3x3 and converts it all to one color. [16] On top of this, it uses clever mathematical formulae to store sets of pixels as waves. Here it also takes into account that the human eye sees logarithmically so humans are less sensitive to higher frequencies. Thus it also goes through a series or "rounding" or quantization to put more weight on the lower frequencies and get rid of the higher frequencies. There are more steps to this process to continually reduce the data however they get quite complicated, but can be read about here [33]. Then once the picture is decoded it will interpolate this data in the best way possible, but will clearly create some loss in the midst. All in all, it can be seen that this format is not conducive to getting raw data.

H.264 is also a lossy data format for video. Here it hinges mostly on predictions. It will only store the difference between the current frame and the next frame. Thus it will remove all of the repeated information. As above in order to reduce the data to the smallest possible size it will follow similar techniques, such as grouping pixels, representing pixels as mathematical formulae, and quantization. In the end, it will create a lossy data set that will not be useful to this experiment.[29]

YUV2, YUV, UYUV, YCrCb, and similar techniques claim to be uncompressed, which is true, but they are not lossless. When the color format is changed from RGB to YUV or a similar variant loss is introduced. In this technique "Y" stands for luminance, how bright the picture is. The "U" and "V" stand for the color. Meaning the luminance defines a plane with a dynamic range of black versus white then there can be a 2D grid of one axis as "U" and the other as "V" overlaid on top of this to create a color map. Look at Figure 9 for a visualization and more detail. When images are converted from RBG to YUV or different color map there is some loss because more weight is given to the luminance than to the colors themselves. This is usually done to match what the human eye sees, thus making the picture look prettier and more appealing. However for the project we want to see like a computer sees and find the raw electron count, the raw voltage, therefore this method of using a color map does not work either. [20]

Ultimately, being able to capture raw data in a fits file that creates no loss

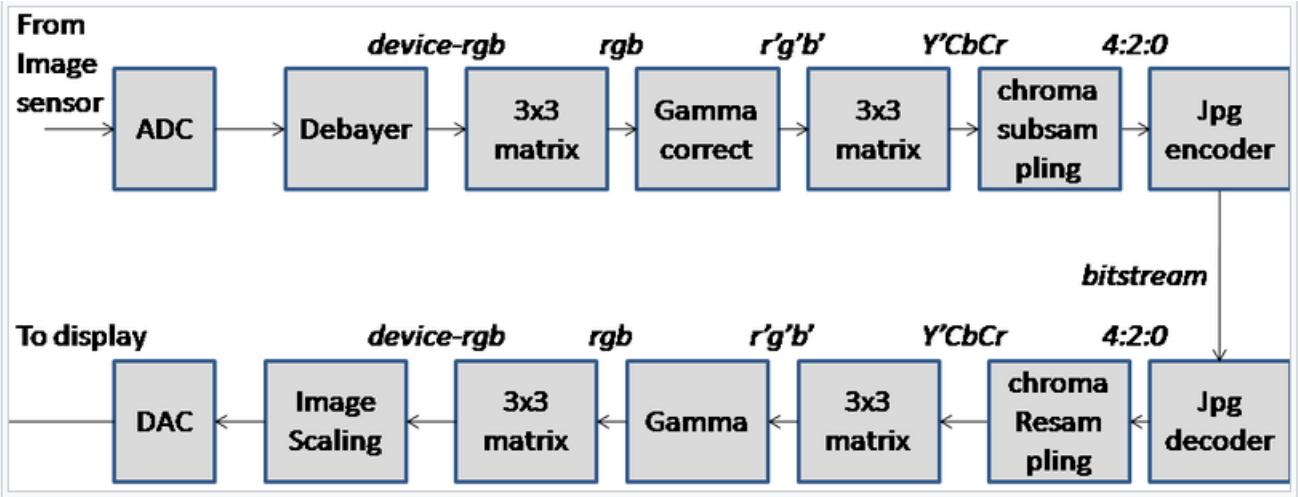


Figure 8: This image highlights a simple process that a typical camera will go through in order to compress the data then uncompress it after it is moved. This creates a lossy image because there is interpolation and smoothing. This picture was taken from https://en.wikipedia.org/wiki/Color_image_pipeline.

is why the ZWOASI178MC camera was chosen. However, in saying this there is one hiccup. How can a colored camera record color? Or in other words how can a photon indicate what color or frequency it is to the electron it excites? Well the Sony IMX178 itself is monochrome so it only knows black and white, electron or no electron. However above the sensor there is a Bayer filter, which is a grid of pixels each with its own color filter. They are usually broken up into 2x2 sets of pixels where there are 2 green color filters diagonal from each other, one blue color filter, and one red color filter. Then when light comes in it gets filtered by these color filters. Now to find the true color demosaicing needs to be done.[18] This is a process where the 2x2 pixel set is combined to give an accurate description of the color using the RGB format, where one pixel is described by 3 numbers one for red, one for green, and one for blue, ranging from 0 to 255. Now this does create loss from the raw light emitted from the source; however, it is minimal compared to the other versions of compression described above. The size of a pixel is 2.9 micrometers by 2.9 micrometers, thus the demosaicing would not have a large visual effect on the image, especially if the source is large enough to take up multiple pixels, which in the case of this project it is about 10 pixels by 10 pixels. Also since this is a quasi-uniform loss then trends will not be affected just the base line. Therefore, the goal of trying to find the correct sine wave from this data set will not be changed drastically.

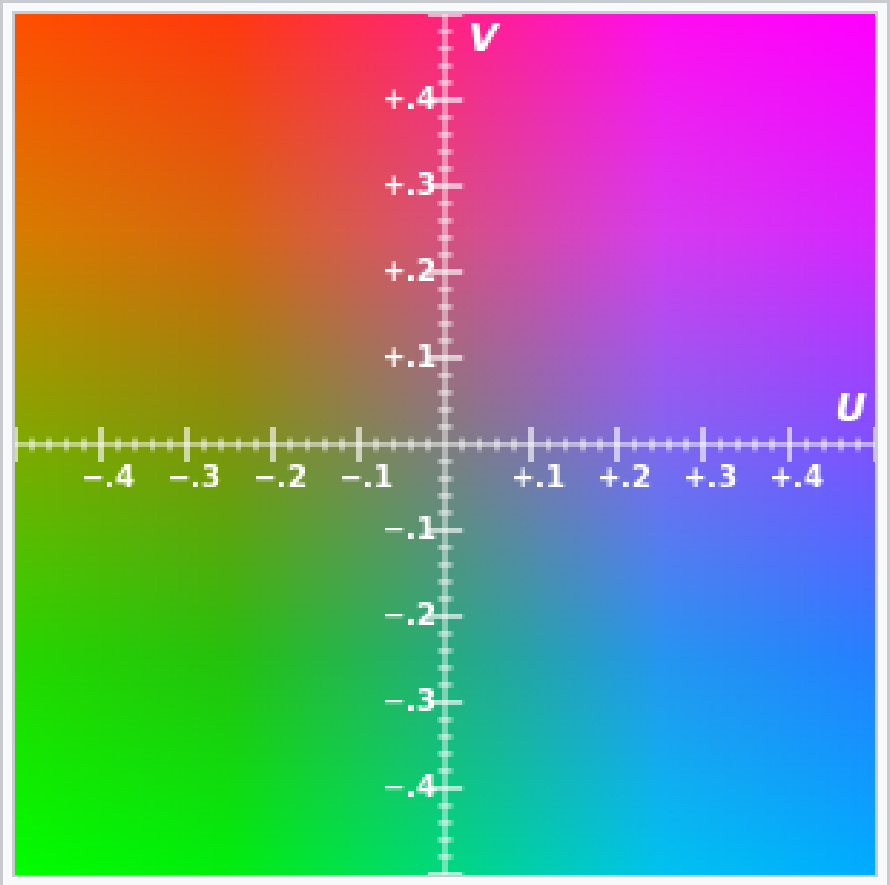


Figure 9: This is a YUV colormap. It shows that a luminance, "Y" was chosen then on top the "U" and "V" color schemes were mapped. The image was taken from https://en.wikipedia.org/wiki/Y%E2%80%B2UV

## 3.3 Lens

Identifying the correct lens to use is very project dependent. In this project, the polarimeter needs to be able to see the LED on the drone. Now this drone needs to be at an "infinite" distance away; meaning that if the source was at a further distance the physics of the problem would not change just a better lens would be needed, so to solve this the drone is put at the maximum range possible. This is saying the scalability of the problem is not complex, which long term is useful if one wants to scale this to apply to Simon's Observatory. Now to measure something at a far distance there needs to be a larger focal length in order to magnify the source. However, increasing the focal length does decrease the field of view (FOV), which needs to be taken into consideration because a wide enough FOV is needed in order to get enough stars to do astrometry on the image. On the other hand a fast f-number (low or short f number is referred to as a fast f number because it converges the light quickly), allows for more light to enter into the lens. This is especially good for low light settings, which for observing stars is good. A slower f-number would allow for a better depth into the FOV, but this could be bad for the experiment because the source could over saturate the FOV if it is too close or takes up too much of the FOV. Using this data we landed on FUJINON HF50HA-1S[12] which has a 50mm focal length and an adjustable f# from 2.3 to 22, but for this experiment it is kept at 2.3. It is attached via a C-Mount. Thus a M-42 to C-Mount adapter was implemented to connect it to the ZWOASI178MC camera.

Another characteristic quickly deemed important is the back focal length. This characteristic determines the distance the back of the lens can be from the sensor. If the back focal length is out of the range of the mount then it is impossible to focus the lens. Therefore, when starting with the DF6HA-1B lens[11] its back focal length was 11.44mm, but the possible range of back focal length of the adapter setup is 12.5-17.5mm depending on how tightly it is screwed in. Therefore there was no way that this lens could get in focus. That is why we switched to FUJINON HF50HA-1S which has a back flange of 15.24mm which is dead middle of that range.

## 3.4 Half-Wavelength Plate and Linear Polarizers

To understand the half-wavelength plate's importance it is important to review the path of light. It starts at the LED then goes through one linear polarizer which allows one unique polarization through, lets say East-West for simplicity. Then it hits the rotating half-wavelength which rotates this East-West polarization to a unique polarization, depending on the orientation of the half-wavelength plate. Next it hits another linear polarizer before it hits the camera sensor. If the second linear polarizer is North-South then without the half-wavelength plate changing the polarization there would be no way light could get through, but as the plate spins the more light is allowed through until it reaches its peak then it starts letting less in until it hits the trough. This process keeps repeating therefore it goes is a cyclical cycle.

The half-wavelength plate is birefringent meaning that there are two different materials in the plate each with their own refractive index. Thus it splits the light into two polarized components and offsets one by half a wavelength, hence the name, making sure there is constructive interference after leaving the half-wavelength plate. The offset by half a wavelength is done by controlling the thickness of the plate to match the frequency being manipulated. The constructive interference is what creates this polarization shift in the light. Therefore, it is necessary to have birefringence instead of single refringence because there is no constructive interference in single refringence. The biggest issue is that if the source is not matched to the frequency of the half-wavelength plate then there could be light that is not fully blocked which means at the troughs there is still some light getting through. See Figure 10 for a picture of the half-wavelength plate.

### 3.5 Errors

One error that needs to be noted is the variability of the LED. Initially, the LED was being powered by the Raspberry PI through GPIO pins (more on this jargon later), which lead to inconsistent voltage supply. Afterwards the power supply was switched to a battery, which helped stabilize it a little, but it is worth noting that there is still variability in the brightness produced by the LED. Ultimately this can cause concern to the graph of pixel intensity versus motor angle. However hopefully the analysis will smooth over this and minimize the error, yet it needs to be taken into consideration. Look at Figure 11 for the LED-RPI4 setup and look at Figure 12 for the LED-battery setup.

## 4 Stepper Motor

In order to rotate the half-wavelength plate a belt system was implemented. The driving gear to driven gear gear ratio is 1:6. The driving gear is attached to a stepper motor. This means that as the stepper motor makes six full rotations then the driven gear, which the half-wavelength plate it attached to, rotates once. This is very important information for the encoder and for doing analysis, which will both be touched on in more detail later. For this section, it is important to note that the main goal is to find a motor that has accurate motor positioning that can be readout easily. In the end, it was determined that the NEMA17-AMT112S is best suited.

A typical DC, direct current, motor will receive a current which will cause it to spin. There is no way of determining the motor angle, one can only turn it on or off. Therefore, these motors are not precise. A stepper motor, on the other hand, is set up to have a set number of distinct steps. For example, the NEMA17-AMT112S has 400 steps. Therefore when it receives a signal it causes the motor to move one step. Therefore, it is very easy to track and control the positioning of the motor angle.

Additionally, a bipolar stepper motor was chosen over a unipolar stepper

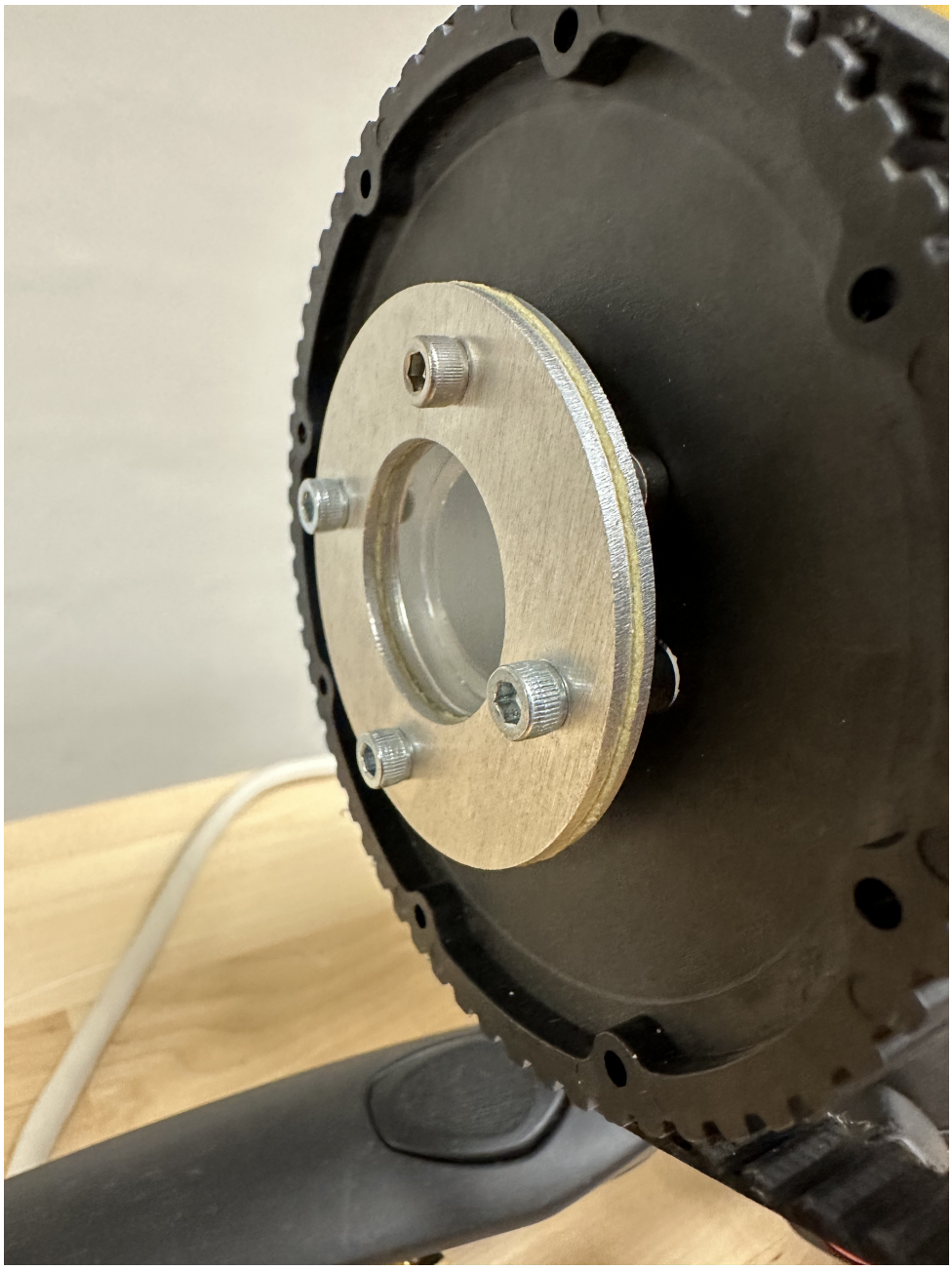

Figure 10: Here it can be seen there is the big black gear and then the small half-wavelength plate attached in the front. The half-wavelength plate is actually very thin and sandwiched between two metal annuluses.

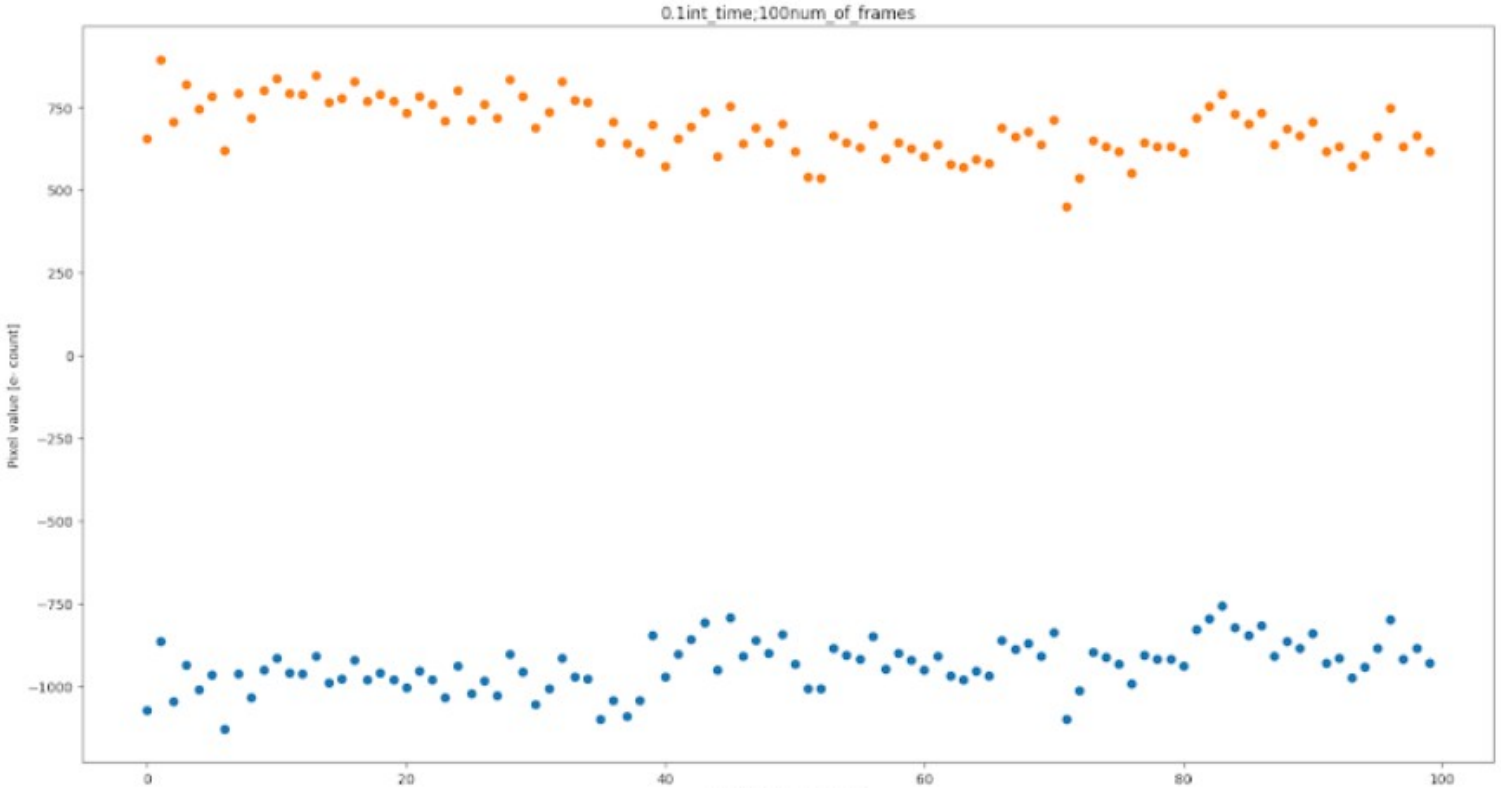


Figure 11: Here it can be seen that when the LED is attached to the RPI4 then it has a pretty unstable output. In this graph there are 100 instance with the x-axis just counting from 1 to 100 and the y-axis is the intensity of the pixel.

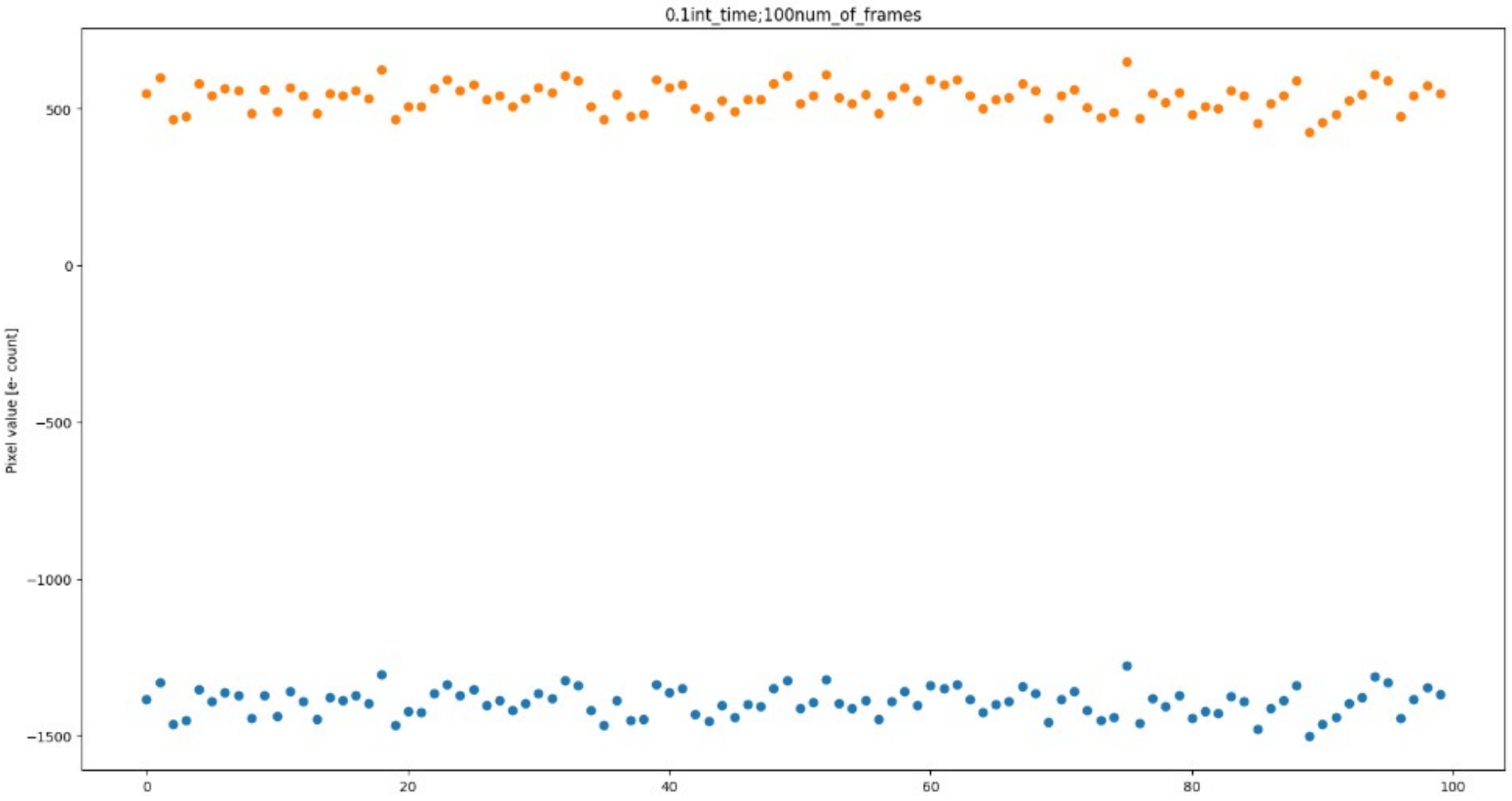


Figure 12: Here it can be seen that when the LED is attached to the battery then it still has an unstable output, but is almost twice as precise as the LED-RPI4 setup. In this graph there are 100 instance with the x-axis just counting from 1 to 100 and the y-axis is the intensity of the pixel.

motor due to the depth of precision and accuracy. A bipolar stepper motor has two coils for each phase which allows for more control over position and minimizes slipping of steps. This means that it has four wires one for each coil; this increases difficulty of implementation, but only slightly.

Script files that raised and lowered the correct GPIO pins controlled enabling, disabling, stopping, and choosing direction. Another script file was created that allowed the user from the command line to choose how faster the motor spun and then immediately started spinning the motor. See Figure 13 to see the bipolar stepper motor.

# 5 Microcontroller and Accessories

Microcontrollers have many advantages over ordinary laptops. To begin they are a lot cheaper than an average laptop that would accomplish the same goal. They have low power consumption. They are pretty low level for changing, morphing hardware and software is manageable. There is not a lot of red tape that one needs to go through in order to make changes. The biggest reason to use microcontrollers is the amount and control of inputs and outputs. The versatility of interfaces allows for one to do a multitude of projects on the same microcontroller and have all of these devices be interconnect. For instance the Raspberry Pi 4 has two HDMI ports, two USB 3.0 ports, two USB 2.0 ports, Ethernet port, audio jack, CSI camera connector, DSI display connector, MicroSD card slot, USB-C power connector, forty programmable general purpose input output (GPIO) pins, and a few other ports. Additionally, one can add Raspberry Pi HATs (Hardware Attached on Top) that have special functions, such as GPS, fans for cooling, stepper motor hats, etc.

## 5.1 Raspberry Pi

Due to the depth of online support, commonality, reliability, and quality the Raspberry Pi 4 (RPI4) was chosen as the microcontroller. Initially RPI3 model B+ was chosen, but the RPI4 had two USB 3.0 ports which allowed for faster download speeds from the camera to the RPI4. This was important for speeding up the bottle-neck of download speed of the images. Additionally, the overall quality and functionality was upgraded from the RPI3 to RPI4, specifically the processor and RAM. The only downside is that RPI4 tend to over heat so a heat sink or a fan is needed. The choice for the an RPI over an Arduino was based more on the idea that I have more experience with Raspberry Pis that I do with Arduinos. Thus I thought it would be better to grow my knowledge base on science and engineering and spend less time on setup.

Attached to the RPI4 is a terabyte microSD card in the microSD card slot, a fan HAT on top for cooling, a stepper motor HAT on top of the fan HAT, the stepper motor is attached to the stepper motor HAT, the power cable is attached via the stepper motor HAT, the ZWOASI178MC camera connected via the USB 3.0 port. The HATs are attached to the RPI4 via the GPIO pins.

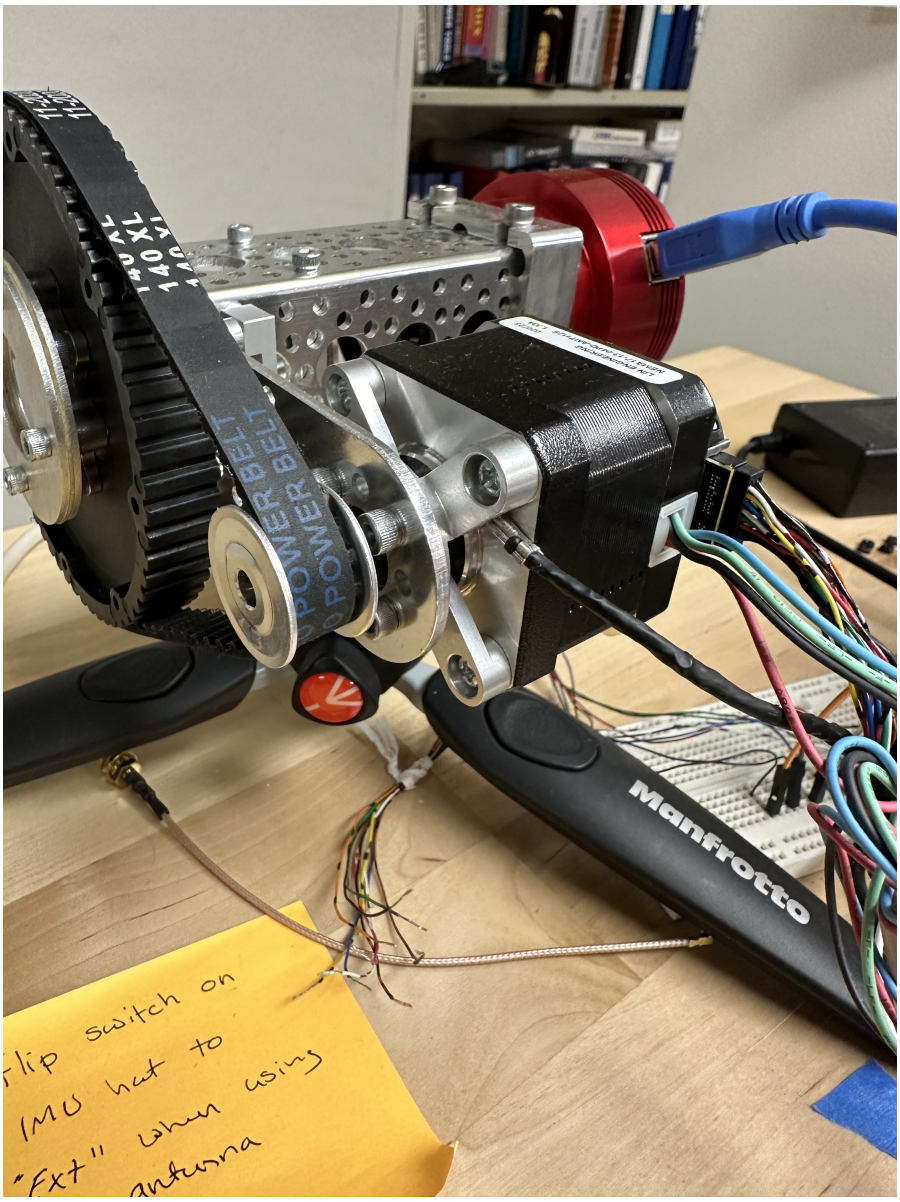


Figure 13: This is the NEMA17-AMT112S bipolar stepper motor. A gear has been attached to the motor shaft then a belt has been attached to that gear. In the back there is the AMT-17C-1-036 encoding wire. Also the red, green, blue, and black wires are the A+, A-, B+, B- wires that directly go to each coil.

The GPIO pins slide into the through holes in the HATs to make an electrical connection. See Figure 14, Figure 15, and Figure 16 for RPI4 setups.

General purpose input and output pins or GPIO Pins are designed to be easily interactable through the software of the Raspberry Pi. These pins can either be pulled high (on) or low (off). Some of the GPIO pins are preset to do specified jobs such as UART, I2C, SPI, PWM, ground, and power. UART or Universal Asynchronous Receiver/Transmitter are used specifically for serial communication between the Raspberry Pi and other devices such as a GPS, Bluetooth device, or other microcontrollers. The I2C or Inter-Integrated Circuit are commonly used for sensors and other integrated circuits. SPI or Serial Peripheral Interface is useful for high-speed communication between the Raspberry Pi and other devices. PWM or Pulse Width Modulation allows one to create an analog like signal which is useful for motors and LEDs. The ground pin acts as an electrical ground. The power pin can either be a 3.3V or 5V which supplies a more or less steady voltage to that pin when turned on. These preset pins do not consist of all the pins there are still other "free" pins that are programmable. [17]

## 5.2 Waveshare Stepper Motor HAT

In order to effectively communicate with the stepper motor we chose to use the Waveshare Stepper Motor HAT.[28] The HAT streamlines the process of timing and signal precision through the motor driver chip. The motor driver chip is complex circuitry specifically designed for operating motors in the most effective and safe way. This minimizes error through skipping steps, stepping backwards, or overstepping. On top of this, the Waveshare Stepper Motor HAT implements dual H-bridges, which are a grouping of switches that improve control over speed, direction, and efficiency of the stepper motor. The "dual" means that there are two H-bridges, but they are separate. This is just another level of precision that achieved from using the HAT. The stepper motor could have been attached and programmed through the GPIO pins; however, using a motor driver increases the precision of the signal, ease of connection, and overall effectiveness of the process.

When connecting the stepper motor to the HAT itself there are special ports that the HAT provided for a bipolar stepper motor. These correspond to the H-bridges mentioned above. There were eight ports, four for the first H-bridge labeled "M1" for motor one and four ports for the next H-bridge labeled "M2" for motor two. Recall from above that the stepper motor has two wires for one phase and two wires for another phase, these are labeled A+, A-, B+, B- by convention. The A's are connected and the B's are connected. Thus when plugging in it is crucial to match the A's together and the B's together. Which end is first, A+ then A- or A- then A+, doesn't matter it will only change the direction that the motor is spinning. See Figure 14 to see how the stepper motor is attached to the Waveshare Stepper Motor HAT. See Figure 15 for another angle of the Waveshare Stepper Motor HAT.

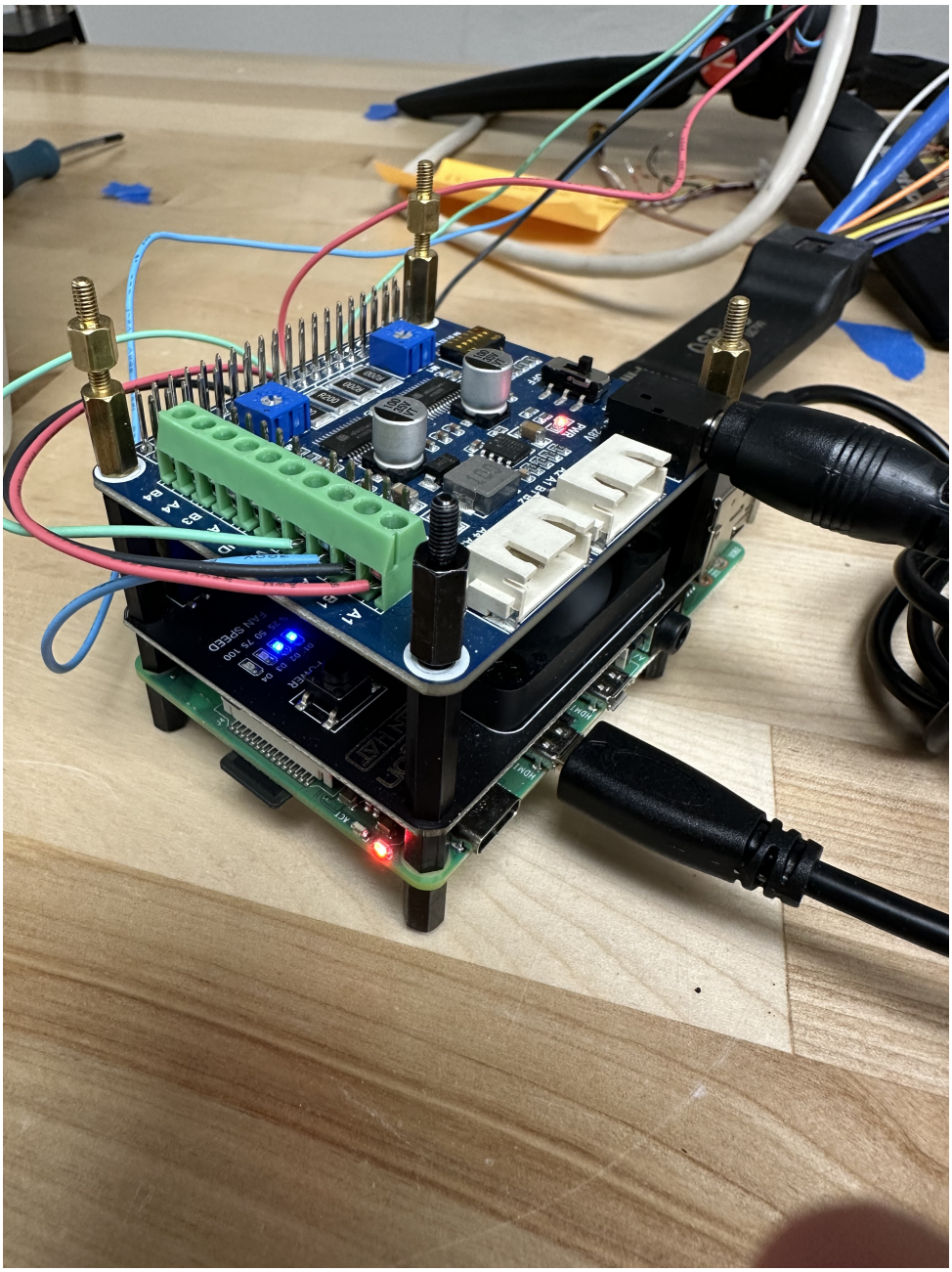

Figure 14: This is the Raspberry Pi 4 with a fan HAT and the Waveshare Stepper Motor HAT. From this angle it can be seen how the stepper motor is attached to the Waveshare Stepper Motor HAT. Also, the power supply can be seen attached to the Waveshare Stepper Motor HAT as well. Additionally, it can show the micro USB that allows the RPI4 to interface with a keyboard and monitor.

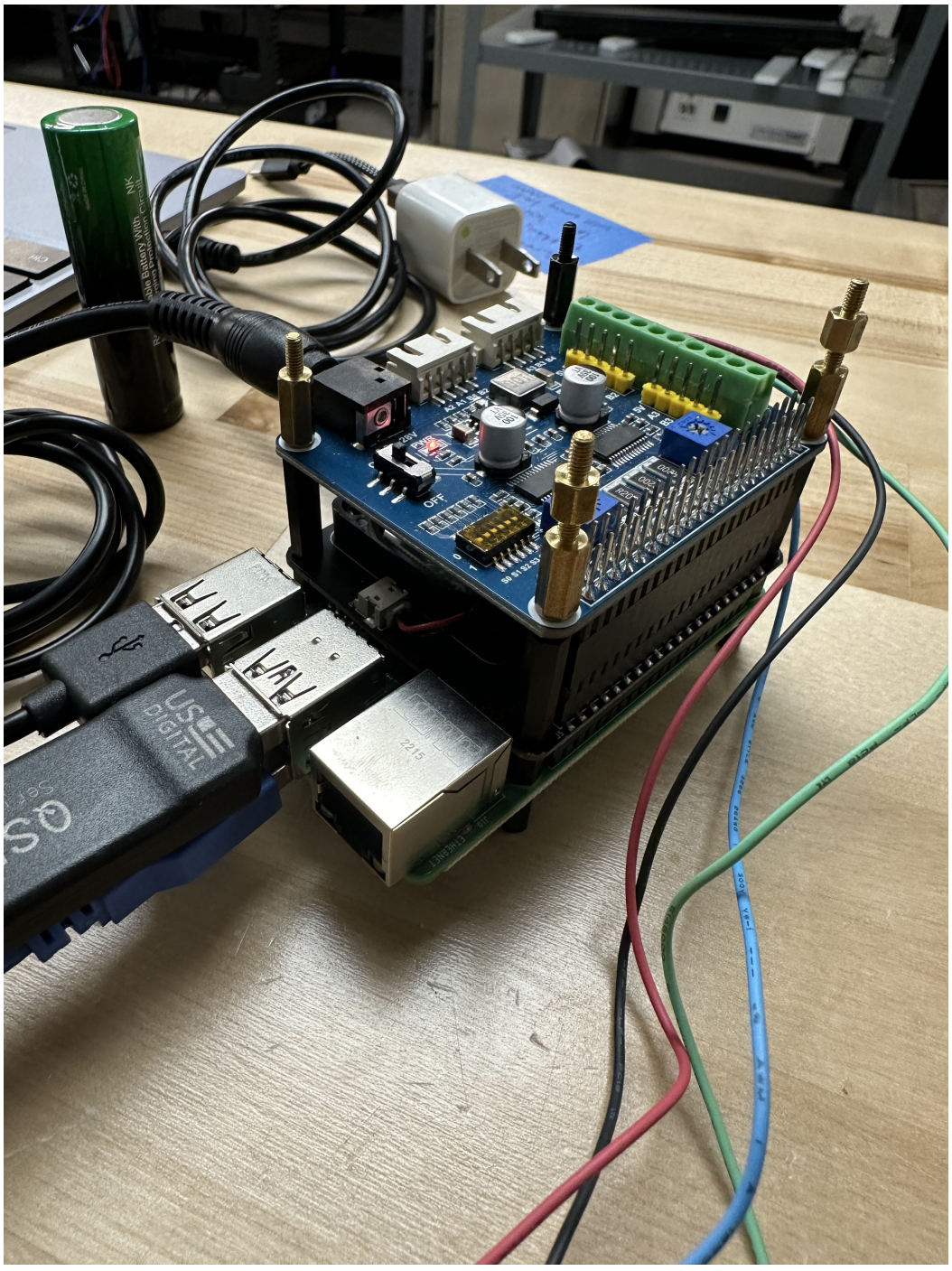


Figure 15: This angle emphasizes all the USB ports. Here it can be seen that plugged into the USB ports are the US Digital QSB stick, the ZWOASI178MC camera, and keyboard/monitor cable allowing the RPI4 to interface.

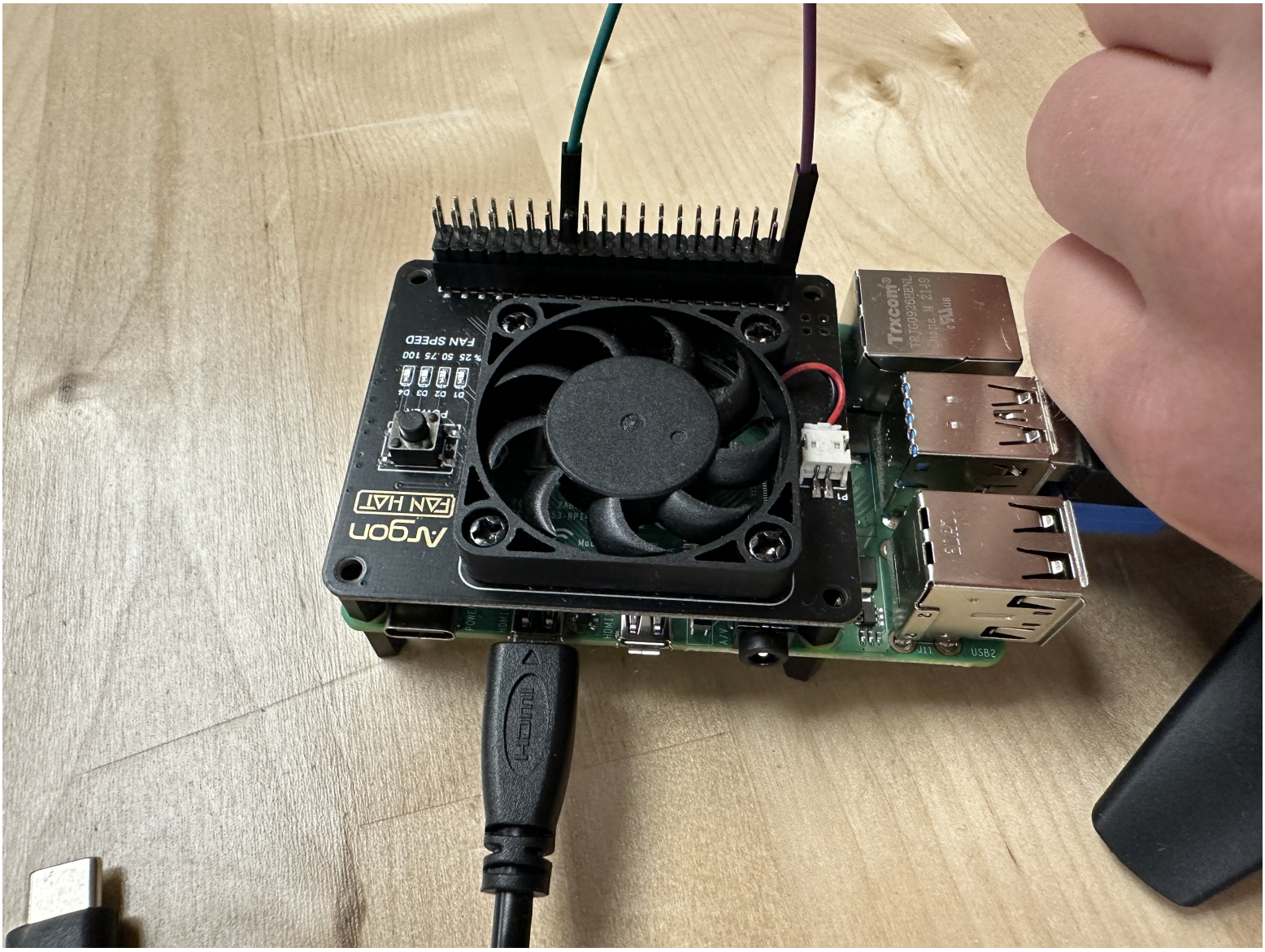


Figure 16: This is a more indepth look at the RPI4 without the Waveshare Stepper Motor HAT. It does still have the fan HAT on it as well as some GPIO extensions. There are also two wires connected to two GPIO pins, which was used to control the LED before it was found that it is more stable to use a battery. Also it can be seen that there is an micro HDMI that allows it to interface with a monitor and keyboard.

## 5.3 US Digital QSB

Attached to the stepper motor is an encoder; however, it needs a readout system and a decoder. Therefore, we attached the AMT-17C-1-036[3] encoder wire. This had 17 connections on it, which neatly plugged into the back of the stepper motor. The compatibility of this wire with the NEMA17-AMT112S stepper motor is the reason this was chosen. In a naive initial attempt, the signal from these wires was read using the A+, A-, B+, and B- wires. These correspond to the wires that the stepper motor uses to spin. There were thirteen other wires that were left unconnected. The output from these was either a "1" or a "0" when each value was graphed, yet each one seemed to fluctuate at different times. After talking to some senior colleagues it was found out that the signals that were being received were square waves. The motor receives signals that are off half a phase in order to know when to rotate and in what direction. The length of the square wave determines the speed of the steps, where shorter waves create faster steps. Thus, to understand the encoder one needed to decode these waves. Although there are multiple methods for doing this, the best approach to decoding these waves is to use a quadrature decoder. [9]

We landed on using a US Digital QSB-D quadrature decoder[21]. This device was chosen because a colleague had received good results from using a similar product. Quickly it was discovered that the scope of this device was well out of the range of what was attainable. The code had to be compatible with Linux and it had to be computationally quick so it was written in C++. I do not know C++ beyond the basics. It was tremendously important to reach out to the engineer who worked on this project and ask for help. Aside from the science this was a great lesson to learn and into the future. Reach out to those who understand, rather than suffering in silence.

In order to connect the encoding wire to the quadrature decoder an intermediate electrical breadboard was used because both wires were male. In the end, the wires from the encoder that were important were the A+, A-, B+, B- all for getting the pulse information, the Index+ (Z+), Index- (Z-), ground, and 5V power. See Figure 17 for the visualization of the setup, Figure 18 for the connections chart, and Figure 19 for the QSB-D pin layout.

The US Digital QSB-D device was critical for turning these square wave pulses into step numbers. In the end it output numbers from 0 to 399 for one revolution before it would reset and start counting from 0 again. Since there is a difference in gear sizes, 1:6, between the encoder and the half-wavelength plate the count was adjusted to say that instead of 0-399 steps it will not reset until it reaches 2399. Thus for six rotations of the small encoder's gear the big gear will rotate once and it will count each of these steps of the big gear from 0 to 2399.

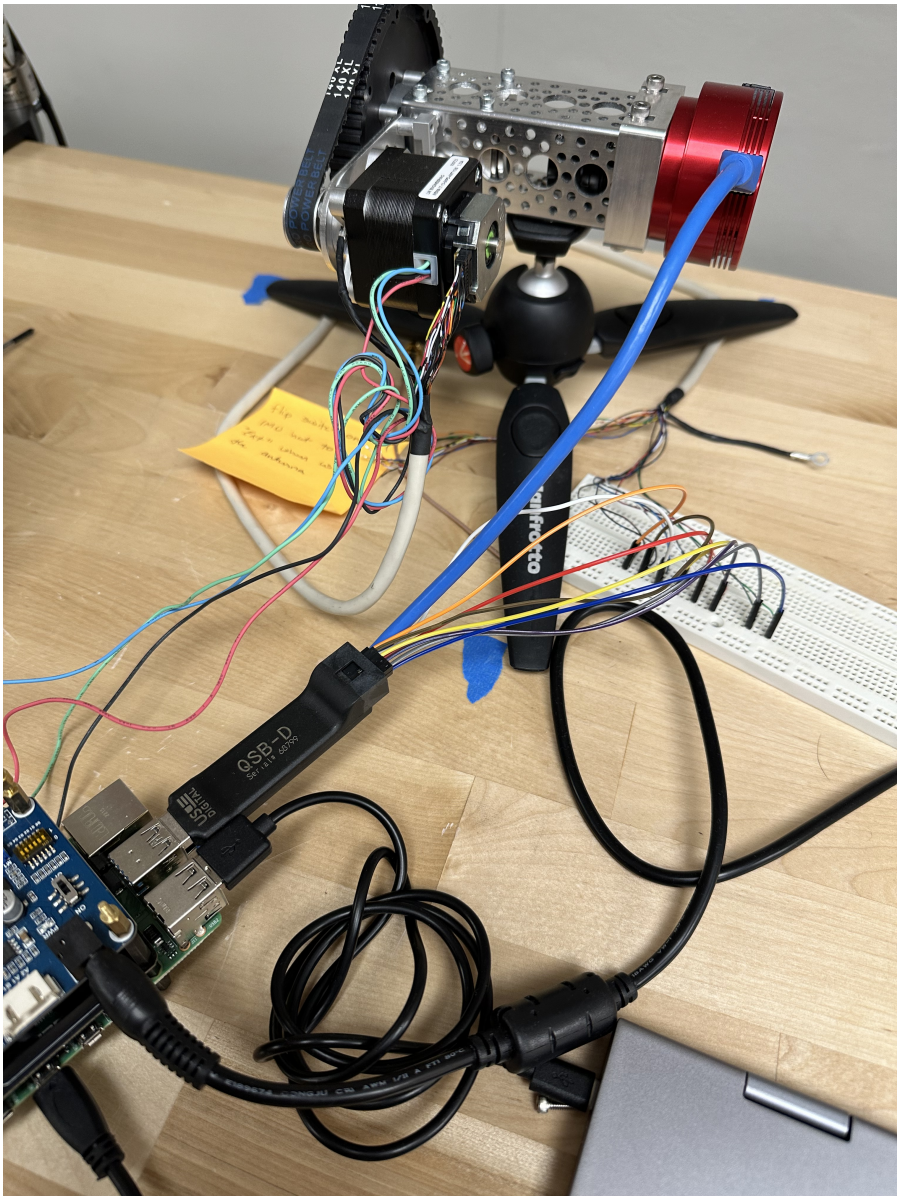


Figure 17: This shows the full setup of the AMT-17C-1-036 encoder wire hooked up to the breadboard. Then the breadboard hooked up to the US DIGITAL QSB-D quadrature decoder.

| QSB Port: | Wire color & function: | Declan's wire: |
|---|---|---|
| 2 | Black with red (check)<br>GND | orange |
| 4 | Blue with black<br>Z+ | white |
| 6 | White with black<br>A+ | red |
| 7 | Red with black<br>5+ | purple |
| 10 | Green with black<br>B+ | blue |
| 3 | Black and blue<br>Z- | brown |
| 9 | Black with green<br>B- | gray |
| 5 | Black with white<br>A- | yellow |
| | | |

Figure 18: Figure 17 presents a messy interpretation of how the encoding wire is matched with the bread board and the then the QSB-D stick. Here it shows the necessary connections. This is coupled with Figure 19.

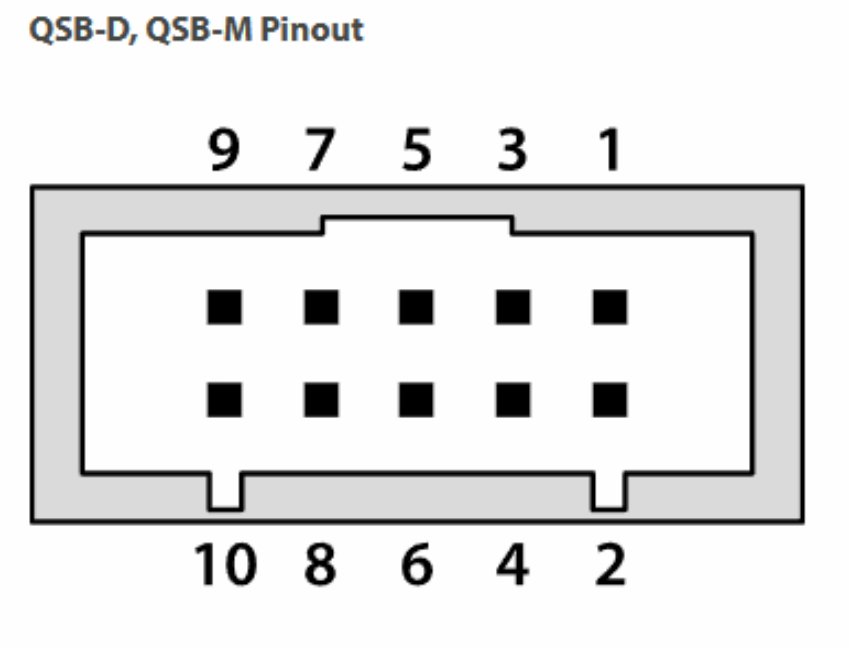


Figure 19: This is the pin details of the QSB-D stick. It is coupled with Figure 18 to show a clear representation of Figure 17. Pin 1 is the Digital I/O Channel 0, pin 2 is the ground, pin 3 is the Index negative, pin 4 is the Index positive, pin 5 is A negative, pin 6 is A positive, pin 7 is 5V power, pin 8 is no connection, pin 9 is B negative, and pin 10 is B positive.

# 6 Software

At first it was decided to use the default Raspbian Bullseye OS that is specifically suited for the RPI. However, further down the line it was found that there was no suitable software for the camera. Therefore, we switched over to Linux Ubuntu 22.04 Jammy Jellyfish[32]. Similar to the Rasbian OS, this could also be obtained by using Raspberry Pi Imager[22], which is a software tool that allows one to burn OS onto a microSD or SD card. Linux is a great operating software to use because it is low level. One can easily make changes to the software, install packages and libraries, and delete unnecessary parts with little red tape.[34]

## 6.1 Kstars and Ekos Live

In order to take, display, and store picture there needed to be a software that is compatible with the RPI4 and with the camera itself. On top of this, it had to directly talk to the camera and change some of the settings so the pictures would execute in the manner desired. Kstars is a desktop planetarium, but also a basis for Ekos live, which allows remote observatory management and image capturing. Kstars and Ekos live are connected to an INDI server, which allows the devices to communicate with each other. Another reason that this setup was picked is that INDI has a decently large forum of dedicated amateur astronomers who post and answer questions. Look at Figure 20 to see Kstars.

Once connected to Ekos live, the user needs to setup a script file of sorts in order for the software to understand what it is looking for and how it can achieve this. For example, the ZWO CCD camera was selected and it was selected that it was attached locally. After that start the INDI server and the camera will automatically connect if done correctly. After this go to the tab for the camera and select the characteristic wanted. For example, the exposure time, gain, picture count, and output storage location are usually modified. Then from this tab one can do a quick check by using the video button near the bottom right. This has very low quality, but for positioning it is a good check. Next, add it to the queue in the upper right then click the start button and the camera will take pictures and display it to the screen one at a time, while also storing it. Look at Figure 21 to see Ekos live and Figure 22 to see how to operate the camera.

# 7 Analysis

## 7.1 Data Collection

The analysis portion of this code nicely marries the software and hardware of the project in order to achieve the scientific goal of determining the polarization angle of the light. So far there is code to spin the stepper motor, code supplied by US Digital to readout the encoder values to the screen, and Kstars and Ekos live to capture the image. Now the goal is to create a graph with the motor

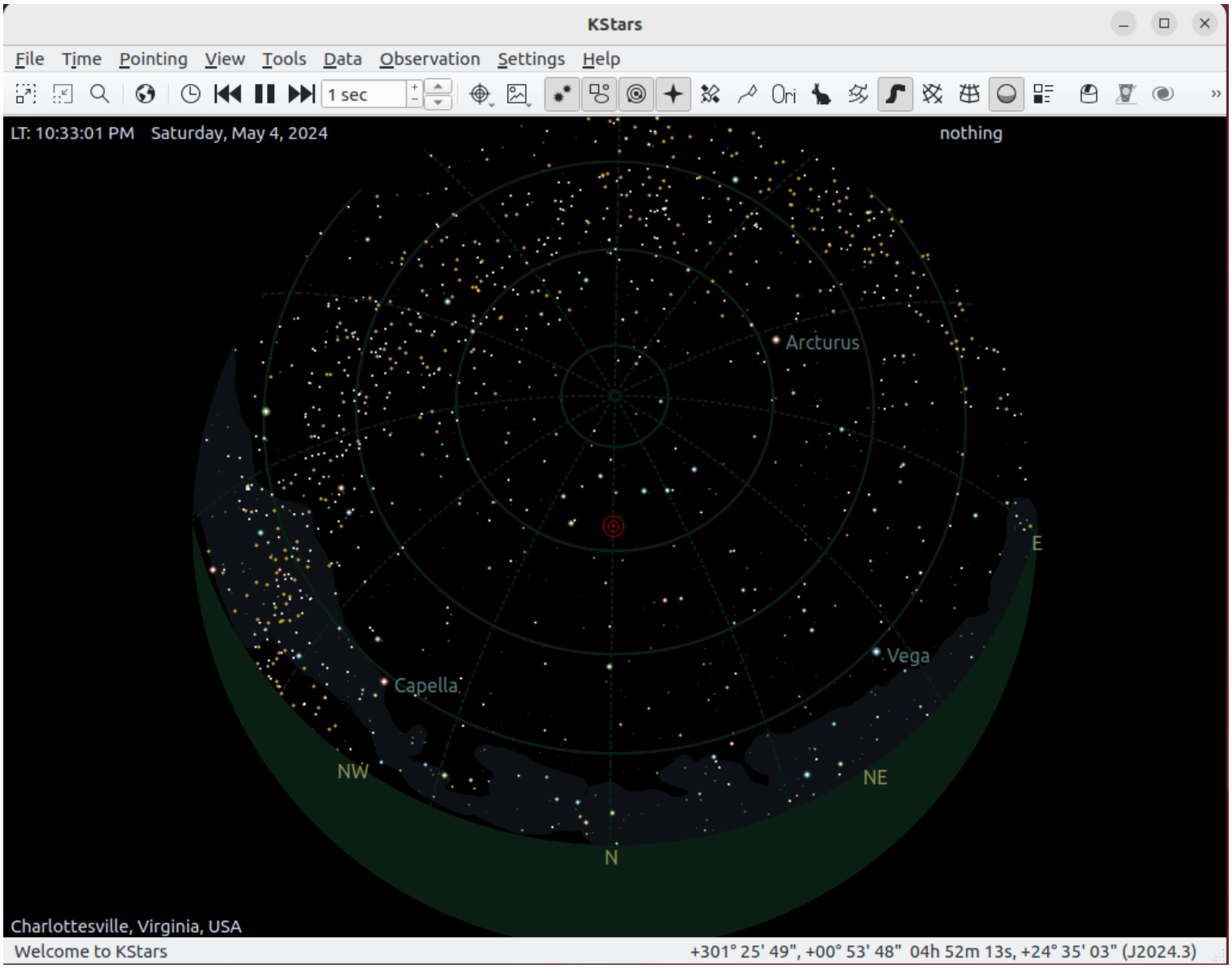


Figure 20: This is the opening screen of Kstars. It shows an interactive planetarium. The user can adjust the time and location to match up with their night sky. Currently the location is set to Charlottesville, Virginia, USA and that can be seen in the bottom right corner. There is an abundance of possible tools to play around with here.

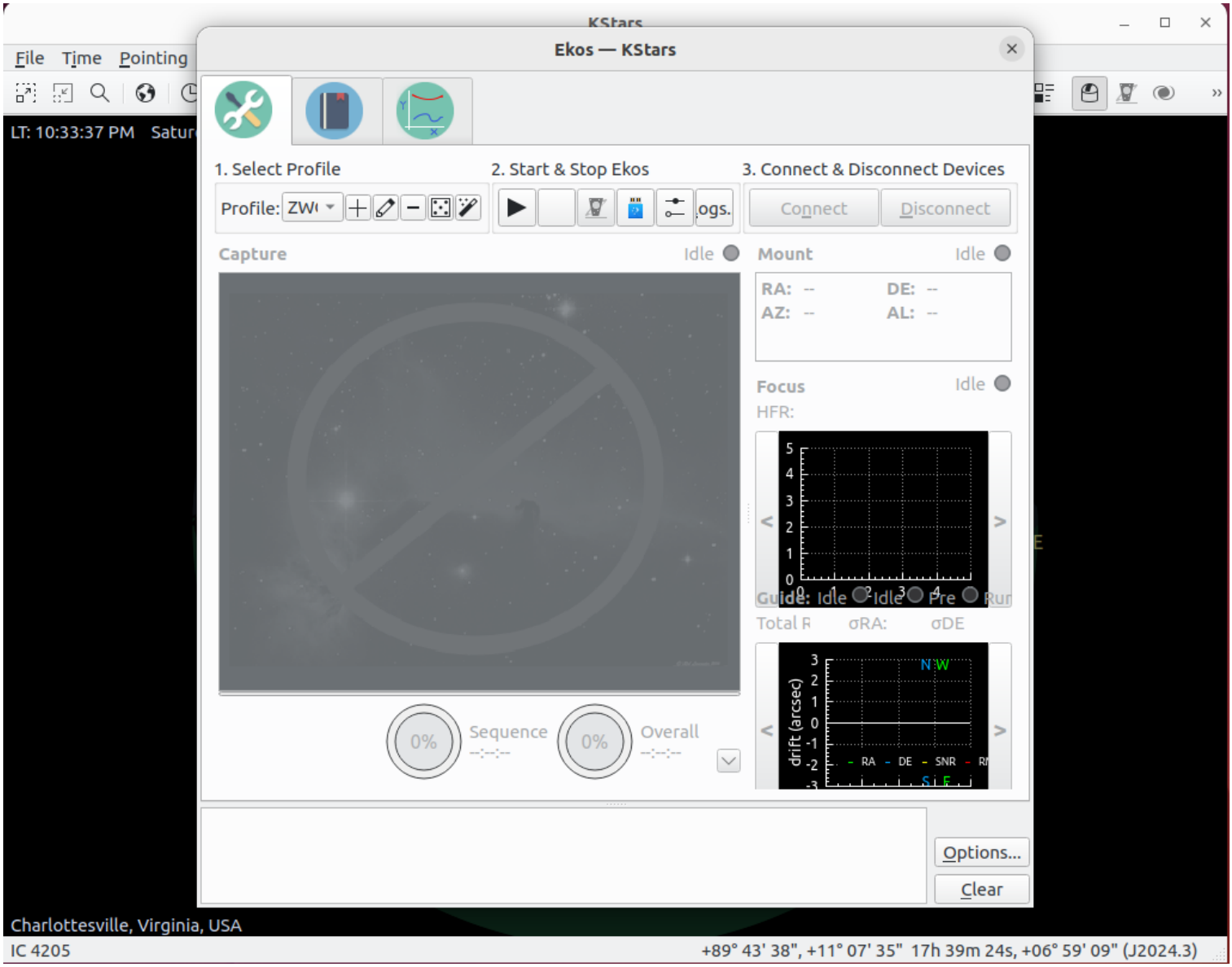


Figure 21: This is Ekos live. It can be opened by pressing "ctrl+k" on the Kstars screen (Figure 20). From here under "Select Profile" select the "+" to add a new profile. It will have the user input all the equipment information. After saving, under "Start & Stop Ekos" click the play button to start the INDI server. This will then automatically bring up the next screen (Figure 22). There are other tabs at the top that are useful for tracking stars and analysis of images, but we did not use any of those in this project.

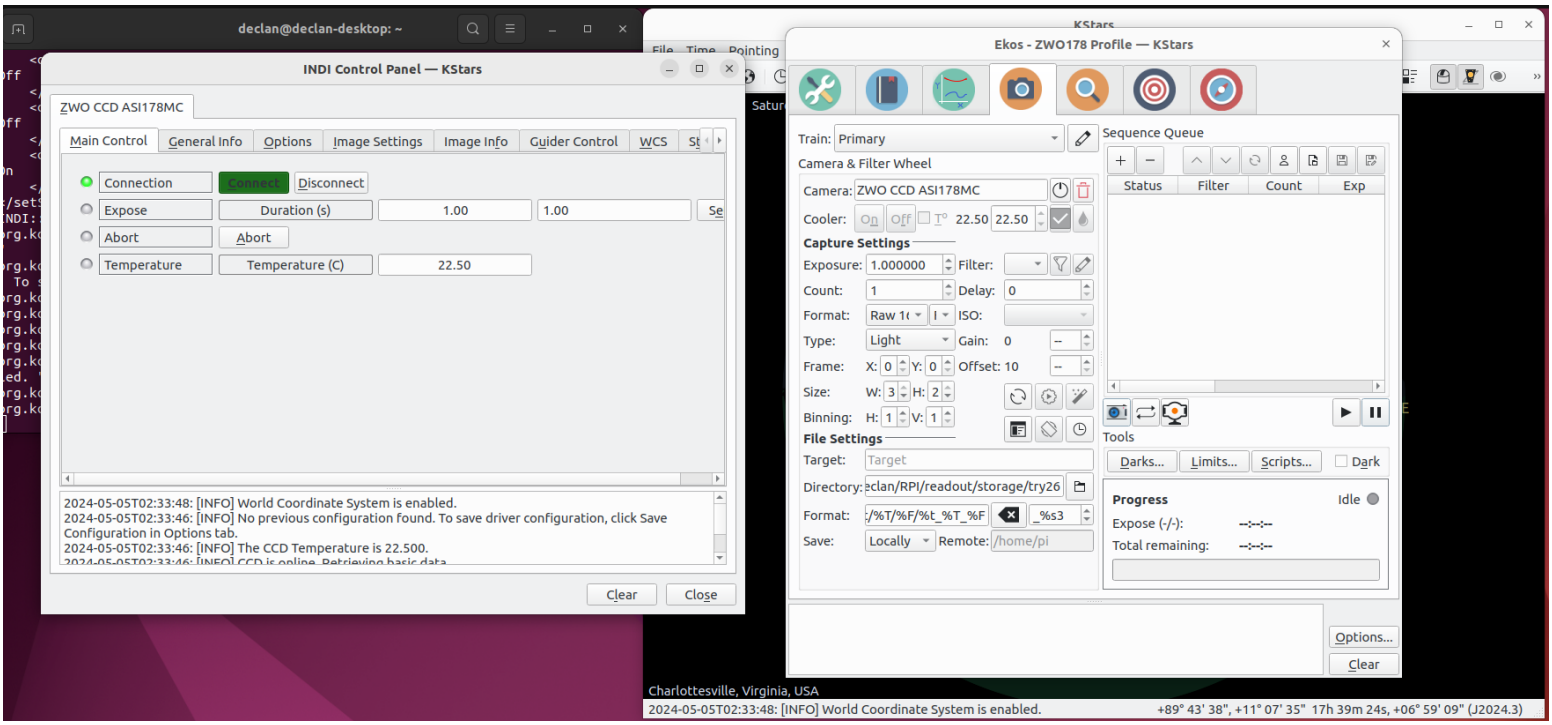


Figure 22: Once the INDI server is started a camera tab will appear and the INDI Control Panel will pop up. Under the camera tab, already selected above, the user can change the exposure time, the number of pictures (count), the gain, the output location, and many more features. Once the options are selected under "Sequence Queue" click the "+" to add the selection to the queue. Then to start click the play button on the right side under the sequence queue. The progress bar on the bottom right will show where the progress is at, approximate time left, and how many pictures out of the total have been taken.

angle on the x-axis and the pixel intensity on the y-axis. In order to do this first let's look at the encoder first then Kstars.

The US Digital code creates an output to the screen updating the motor position. The code was modified to output the encoder values to a pickle file instead. Inside the pickle file contains a dictionary with the eastern time zone time as the key and the motor position (0 to 2399) as the value. This pickle file can then be saved and used later.

From the images that Kstars saved to our storage, the code goes through each picture and takes the pixel intensity of a pixel. It will also collect the time the picture was taken at. The time that the picture is taken is embedded in the metadata so it is trivial to obtain.

An additional function, the box function, was created that will take the first picture in and create a 5x5 array of white pixels on it at a location, that the box function takes as an input. Thus the user can use this to navigate, by trial and error, to find the x and y location of the central LED pixel. For example if the picture is 100 x 100 pixels. This box function will allow the user to input 41, 32 and then it will start there and make 41-46, 32-37 white to see if this is a good representation for the next function to take as the LED intensity. The only downside here is we are assuming that the LED is not moving location from one picture to the next. In the lab this is a safe assumption, but is something to be noted when in the field. If this is a good representation that one can use the values of 41 and 32 in the next function. If not then the user can continually try again until it is found.

The idea behind finding the time and value of the encoder and the time and value of the pixel is that the time from when the picture was taken and the time of the motor position will lineup for a unique value so these two values can be matched. For example, the encoder values come in approximately every 0.1 seconds, but the camera takes roughly 1.5 seconds to take the picture and download it before it can take the next picture. So let's say that a picture was taken at 3:21.0000 PM but the encoder will have values ranging from 3.20.991 PM, 3.20.992 PM, ... 3.21.000 PM, 3:21.001 PM, 3:21.002 PM. Therefore, if the values subtracted from each other are less than a certain tolerance then the function will match the pixel value to the motor angle. One must use a tolerance because the time of the encoder will not always lineup exactly to the millisecond to the milliseconds of the camera. Now once there is a pair found each one is appended to a list and put in an output file. So a single output file will have two large lists in it. Additionally, there is the option to graph these two lists to see if the sine function is obtained.

Now that the theoretical side is established to actually run the experiment there needs to be code implemented to make sure the timing is precise. First set up Kstars and get it ready with all the characteristics of the camera and the output. It should be one button away from taking pictures. Next in order to take care of the motor side there is another code program that takes advantage of the subprocess command. This allows a user to execute command line functions in a python file. This is used to turn on, enable, set direction, and start the stepper motor. Next there is a small wait then the encoder is started. After this

is ran and the encoder output is being displayed to the screen then click the go button on Kstars to have the camera start taking picture. Next wait until the camera is finished taking pictures, then stop the encoder and stepper motor. After this use the box function to find where the LED is on the image. Finally use the analysis function to match the stepper motor angle to the pixel intensity and graph it. Look at Figure 21 for "output_25" raw data and Figure 26 for "output_26" raw data. These are two data sets that were collected. "output_25" is collected using a paper diffuser between the LED and the the beamsplitter. This scatters the light a little introducing more noise. This is helpful if the light source is non-uniform. "output_26" is the data without a diffuser so should be a lot less noisy, but it could implement non-uniform features.

## 7.2 Fitting

Now in order to find the phase of the sine wave a proper sine wave needs to be fit to the data. In order to do this these techniques were used Scipy's curvefitting[24] technique and Markov Chain Monte Carlo[19], MCMC. However, before this could happen the data needed to be clean up. First off the data wrapped around itself because when there was more than one rotation, which there were multiple rotations, then the x-axis would not plot linearly. To solve this problem the lists of x values and y values were grouped as tuples for each pair, sorted, then changed back to two lists. Next the x-axis was set to go from 0 to 2399 so then it was rescaled to go from 0 to $2\pi$. Lastly, the ideal function of the sine wave is $sin(2x + \phi) + sin(4x + \phi)$, but this can cause complications for fitting programs because of the nature of phases being of the nature of $\phi = \phi + 2\pi$; therefore, it is easier to switch it to $sin(2x) + cos(2x) + sin(4x) + cos(4x)$ to help out the fitting models.

Both Scipy's curvefit and MCMC take a model and general parameters then find parameters that best suit the data. Using both of these techniques allows us to see if there is consistency in the answers and mitigates systematic error. Curvefitting is a more trivial version with the advantage of speed and simplicity, but struggles on complex problems. MCMC is usually used for a problem with multiple parameters and a complicated shape. The model of $sin(2x)+cos(2x)+sin(4x)+cos(4x)$ is not overly complicated, but MCMC still gives unique insight. MCMC shows us all possible answers and the likelihood of the parameters via corner plots. Corner plots plot one variable's likelihood versus another variable's likelihood. Then on the edge it shows the cross section distribution of the graphs. An ideal perfect variable would create a circle, darkest at the middle and slowly fading to white as the radius increases. Then the cross section distribution would be a Gaussian distribution. This plot can also help show if not enough iterations have been taken. While the variables are wandering and finding the perfect value they could easily get stuck in a local minimum not the global minimum. Thus the corner plot can help highlight this error or if there are two points that the variable is stuck between. Therefore, letting it run over more iterations could allow it to figure out the true minimum. [23] [13] For "output_25" look at Figure 24 for Curvefit, Figure 25 for MCMC all possible values, Figure 26 for MCMC

best fit, and Figure 27 for corner plots. For "output_26" look at Figure 29 for Curvefit, Figure 30 for MCMC all possible values, Figure 31 for MCMC best fit, and Figure 32 for corner plots.

Once the both models' outputs were acceptable answers. Then it was time to turn $sin(2x)+cos(2x)+sin(4x)+cos(4x)$ back to $sin(2x+\phi)+sin(4x+\phi)$. Therefore one can extract the phase. In order to find the phase we starting by having the equation

$$sin(2x+\phi)+sin(4x+\phi) = sin(2x)cos(\phi)+cos(2x)sin(\phi)+sin(4x)cos(\phi)+cos(4x)sin(\phi) \tag{1}$$

This comes from the identity:

$$sin(\theta+\phi) = sin(\theta)cos(\phi)+cos(\theta)sin(\phi) \tag{2}$$

Now here I let the model be

$$a*np.sin(2*x)+b*np.cos(2*x)+c*np.sin(4*x)+d*np.cos(4*x)+g \tag{3}$$

which means "a" does not necessarily equal "c" and "b" does not necessarily equal "d". This was done on purpose to see if the fitting models could find similar parameters. However, this leads to us having to be careful when turning the data back into the correct forms. Therefore trivially one could think the above trigonometry identity could be used. However instead one must use:

$$b*cos(2x) = b*sin(2x+\pi/2), d*cos(4x) = d*sin(4x+\pi/2) \tag{4}$$

Now we have:

$$a*sin(2x)+b*sin(2x+\pi/2)+c*sin(4x)+d*sin(4x+\pi/2) \tag{5}$$

$$asin(2x)+bsin(2x+\pi/2) = Rsin(2x+\phi) \tag{6}$$

$$R = sqrt(a^2+b^2), \phi = arctan(b/a) \tag{7}$$

Similarly,

$$csin(4x)+dsin(4x+\pi/2) = Qsin(4x+\psi) \tag{8}$$

$$Q = sqrt(c^2+d^2), \psi = arctan(d/c) \tag{9}$$

If $\psi$ or $\phi$ was negative $2\pi$ was added to them so that the statistics would be smoother and also changing it won't affect the wave. Then from here I compared $\phi$ and $\psi$. The average of them was taken and then to find the error the absolute value of the mean minus phi was found. This creates symmetric error bars and the distance from the value to the mean.

# 8 Conclusion and Future

A polarimeter was successfully built and implemented. Although the drone infrastructure is still being built, the beamsplitter, camera, and LED source are

Output 25

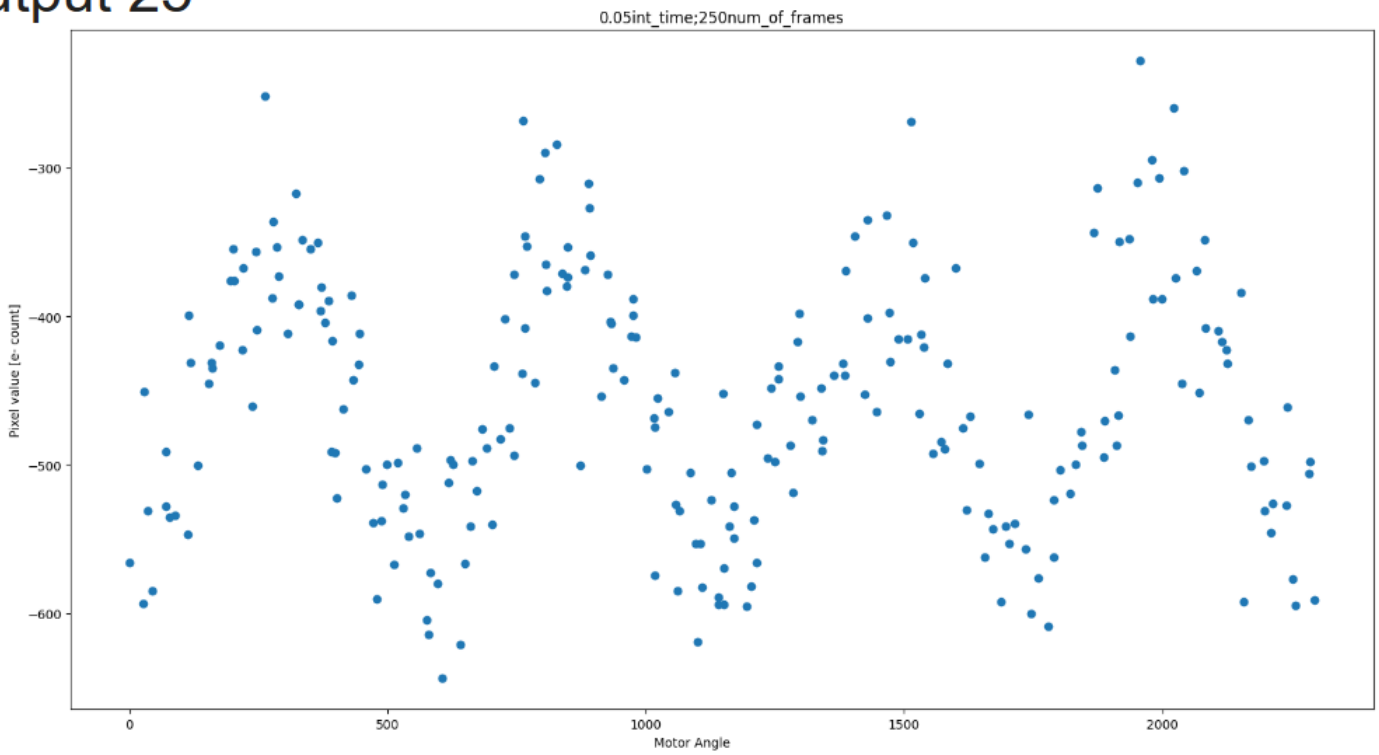


Figure 23: Here is the raw data collected from the camera and encoder plotted together. The data has not been adjusted so the x-axis domain is from 0 to 2399. The title of this graph refers to the exposure time and number of frames.

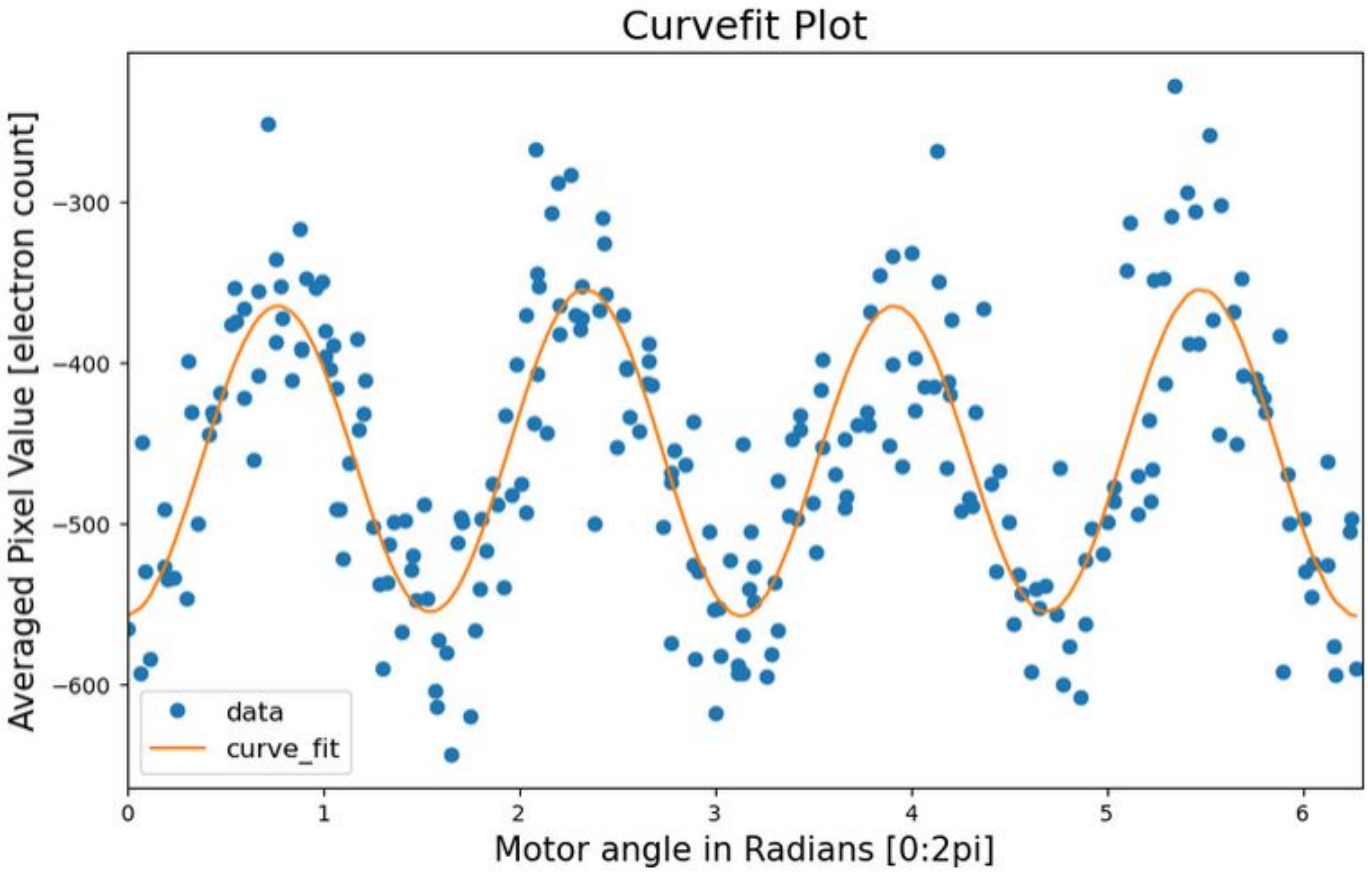


Figure 24: This is the Scipy Curvefit plot. Here the x-axis has been adjusted to 0 to $2\pi$ and the wrapping has been removed. The model of the fit is: $a * np.sin(2 * x) + b * np.cos(2 * x) + c * np.sin(4 * x) + d * np.cos(4 * x) + g$ and the values are: $a = -5.134, b = -1.416, c = 5.827, d = -97.950$, and $g = -458.024$.

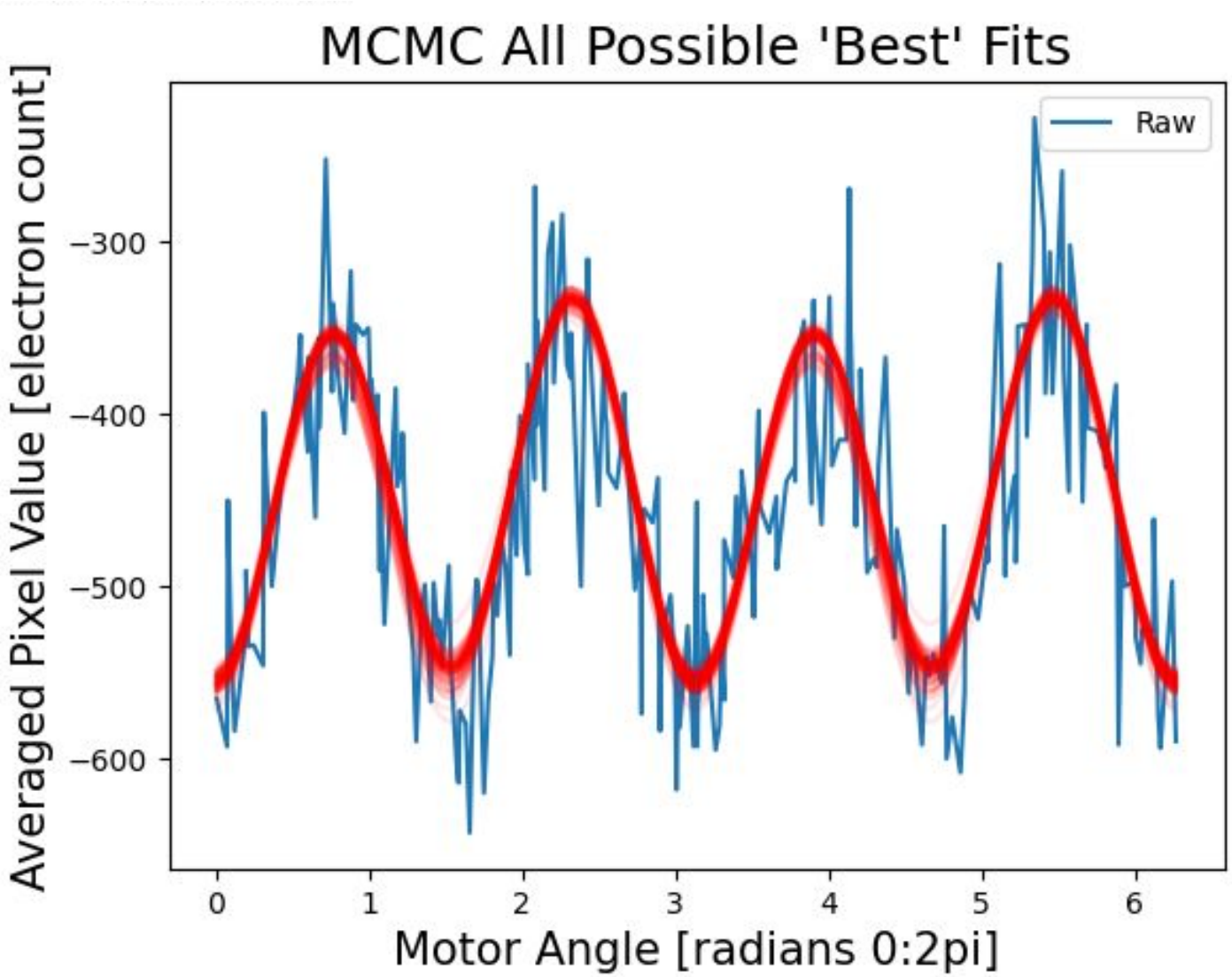


Figure 25: This is all possible values for each variable using Markov Chain Monte Carlo. Again the model inputted is $a * np.sin(2 * x) + b * np.cos(2 * x) + c * np.sin(4 * x) + d * np.cos(4 * x) + g$.

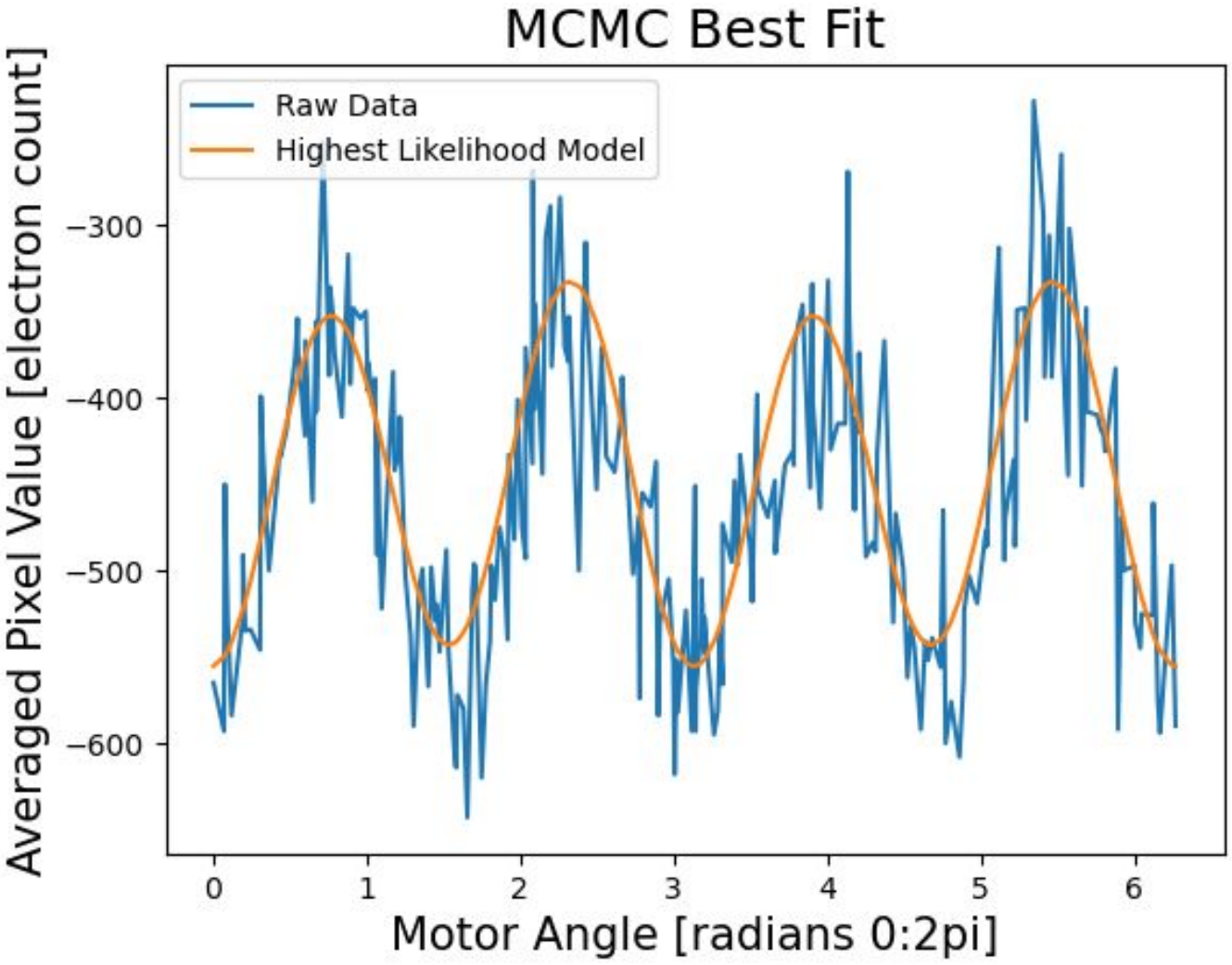


Figure 26: Here is the best fit values for Markov Chain Monte Carlo. Again using the model $a*np.sin(2*x)+b*np.cos(2*x)+c*np.sin(4*x)+d*np.cos(4*x)+g$. The values are $a = -9.6167724, b = -6.56610473, c = 9.22481963, d = -102.80981107$, and $g = -445.88880427$.

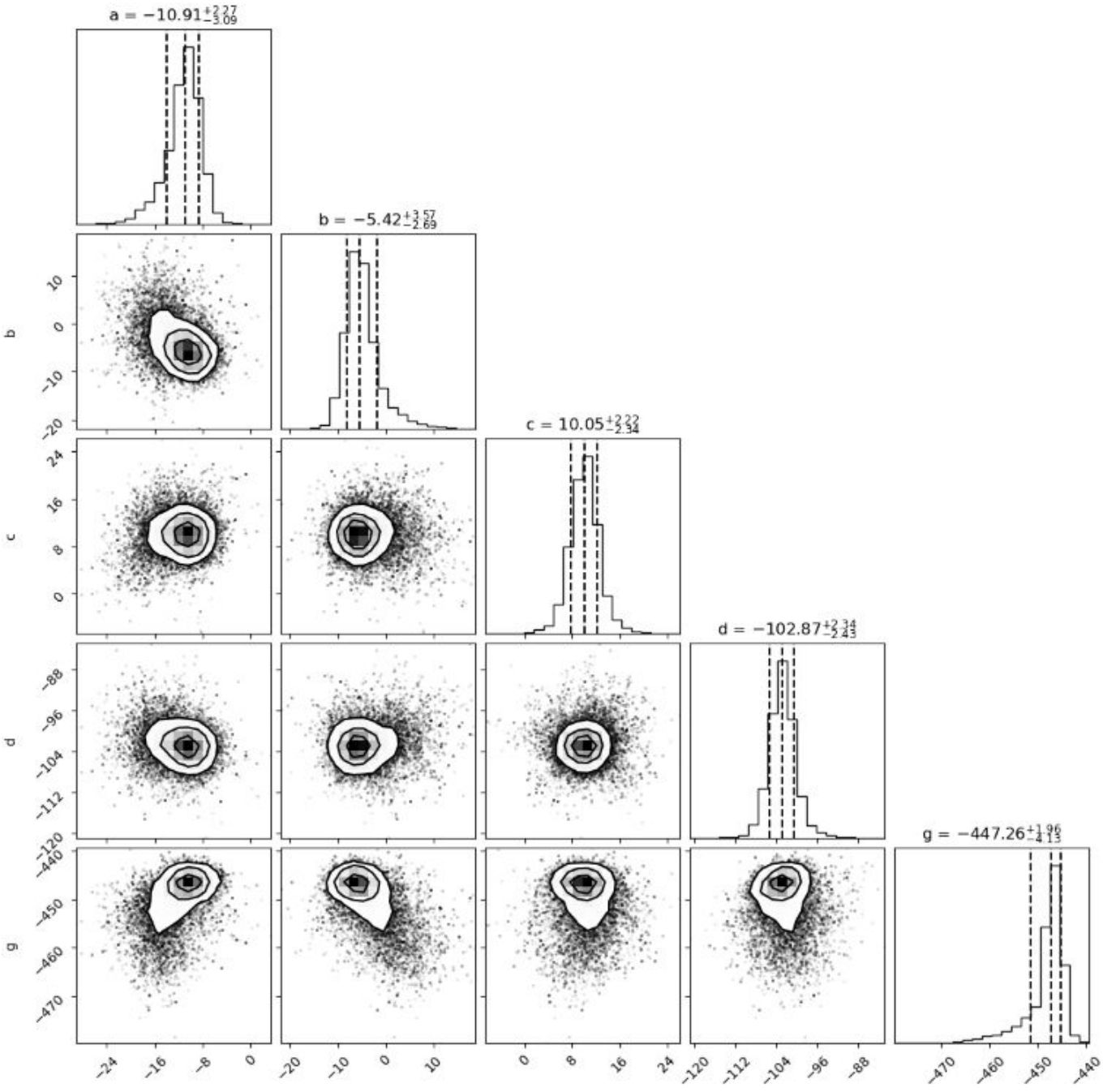


Figure 27: Here is the corner plot of output_25. This shows how each variable is correlated to another variable. Then the graphs on the far right are a cross section distribution of how each variable did against the rest. The more the cross section cut looks like a Gaussian distribution the better the correlation.

Output 26

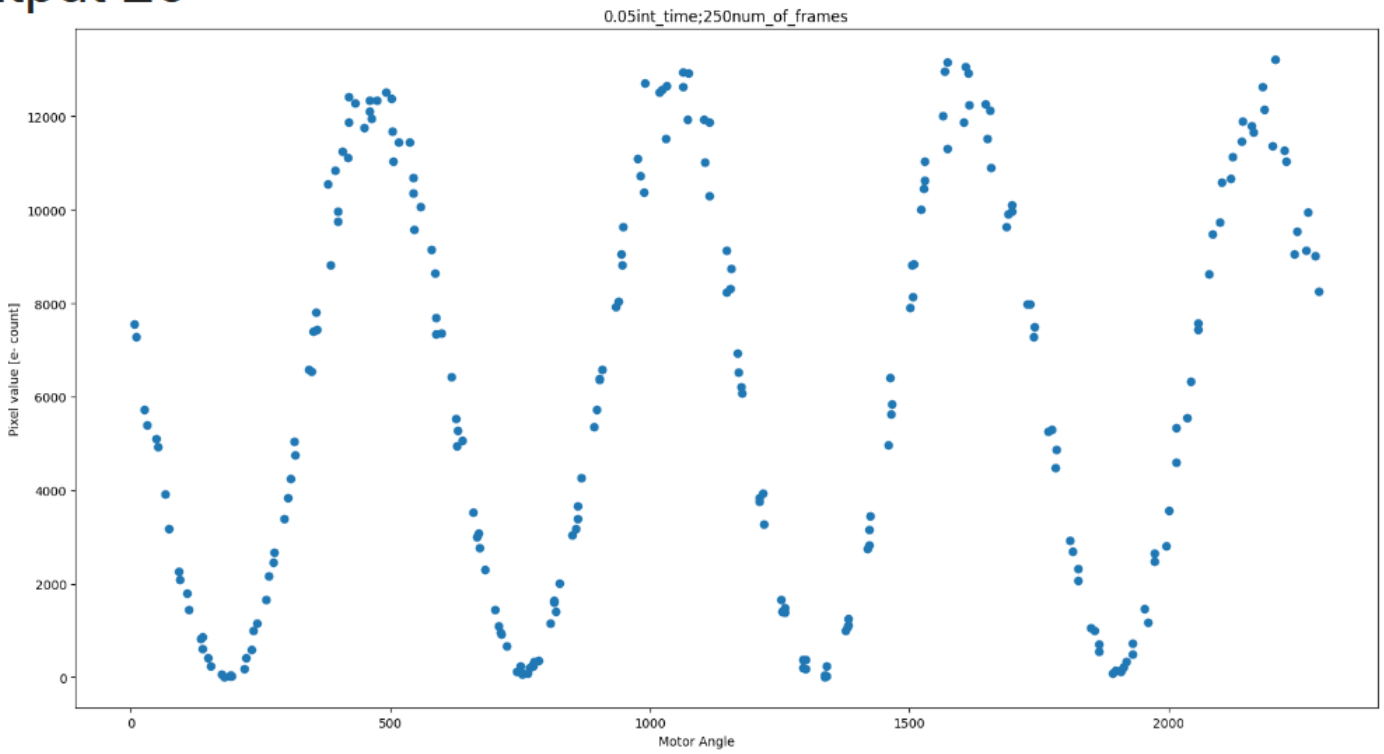


Figure 28: Here is the raw data collected from the camera and encoder plotted together. The data has not been adjusted so the x-axis domain is from 0 to 2399. The title of this graph refers to the exposure time and number of frames.

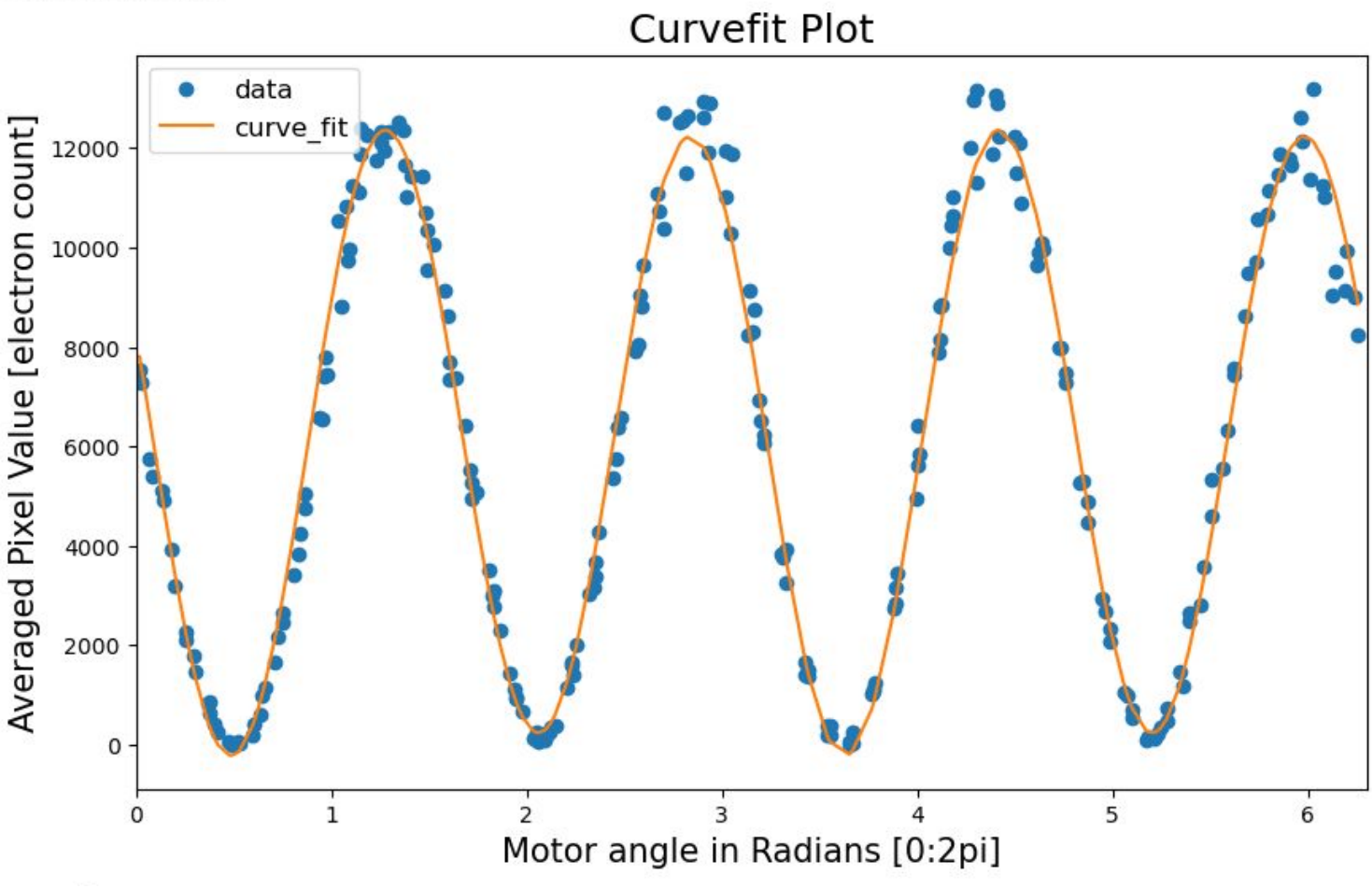


Figure 29: This is the Scipy Curvefit plot. Here the x-axis has been adjusted to 0 to $2\pi$ and the wrapping has been removed. The model of the fit is: $a * np.sin(2*x)+b*np.cos(2*x)+c*np.sin(4*x)+d*np.cos(4*x)+g$ and the values are: $a = -153.231, b = -182.004, c = -5706.282, d = 2278.774, g = 6153.187$.

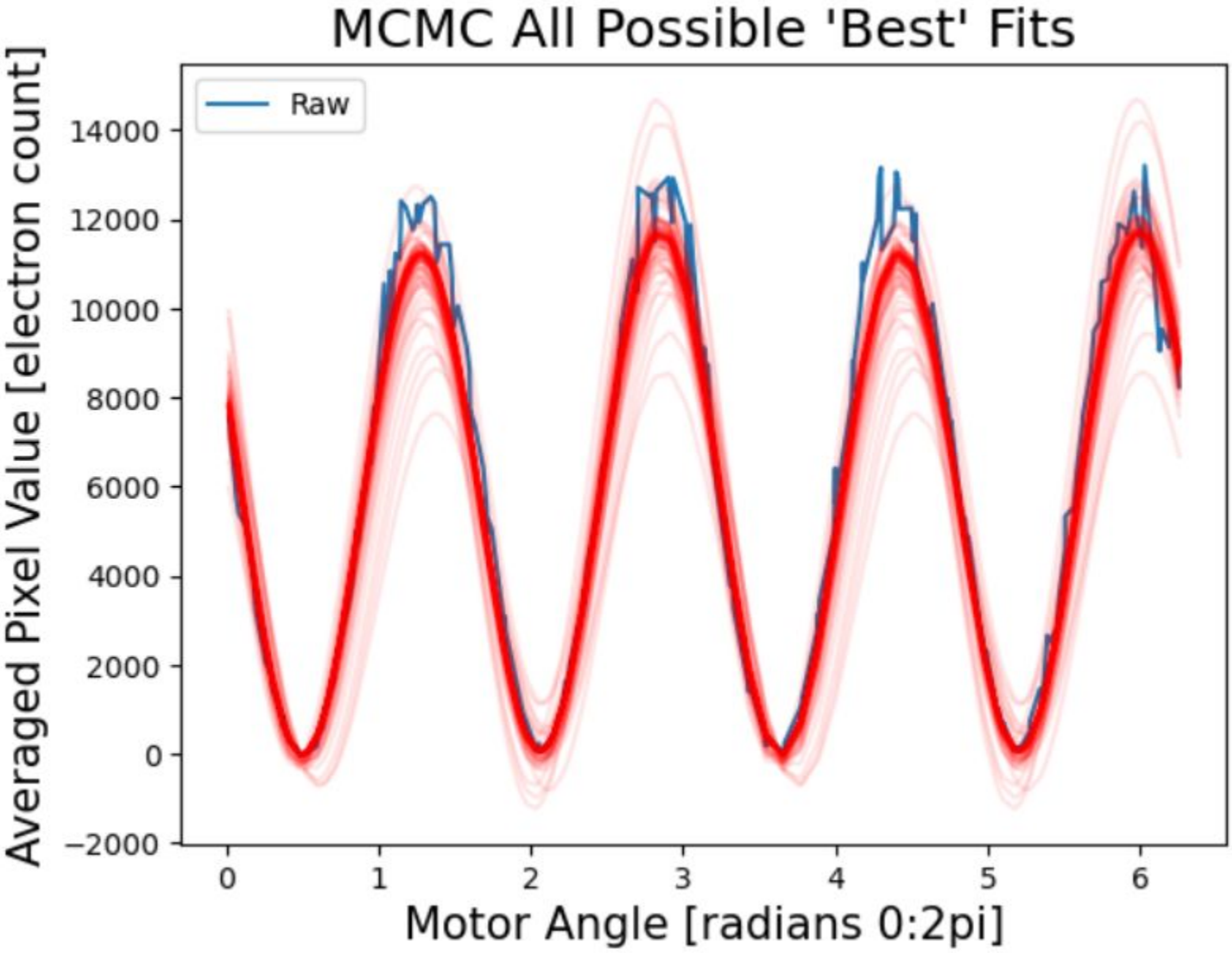


Figure 30: This is all possible values for each variable using Markov Chain Monte Carlo. Again the model inputted is $a * np.sin(2 * x) + b * np.cos(2 * x) + c * np.sin(4 * x) + d * np.cos(4 * x) + g$.

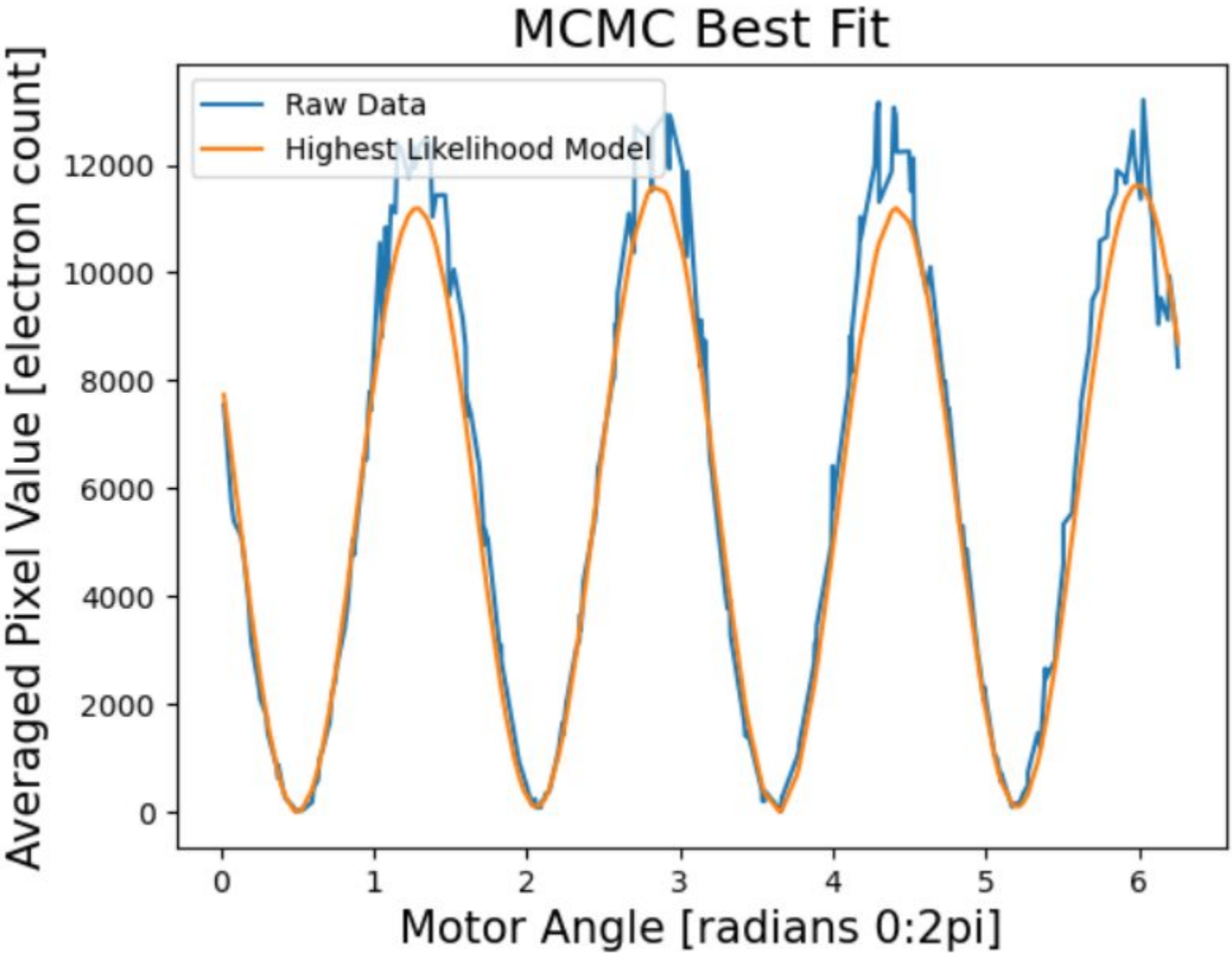


Figure 31: Here is the best fit values for Markov Chain Monte Carlo. Again using the model $a * np.sin(2 * x) + b * np.cos(2 * x) + c * np.sin(4 * x) + d * np.cos(4 * x) + g$. The values are $a = -157.91093875, b = 155.23523396, c = -5214.39403245, d = 2258.54482411$, and $g = 5727.35656701]$. I think leads to further inquiry because the best fit does not match the graph visually as well as expected.

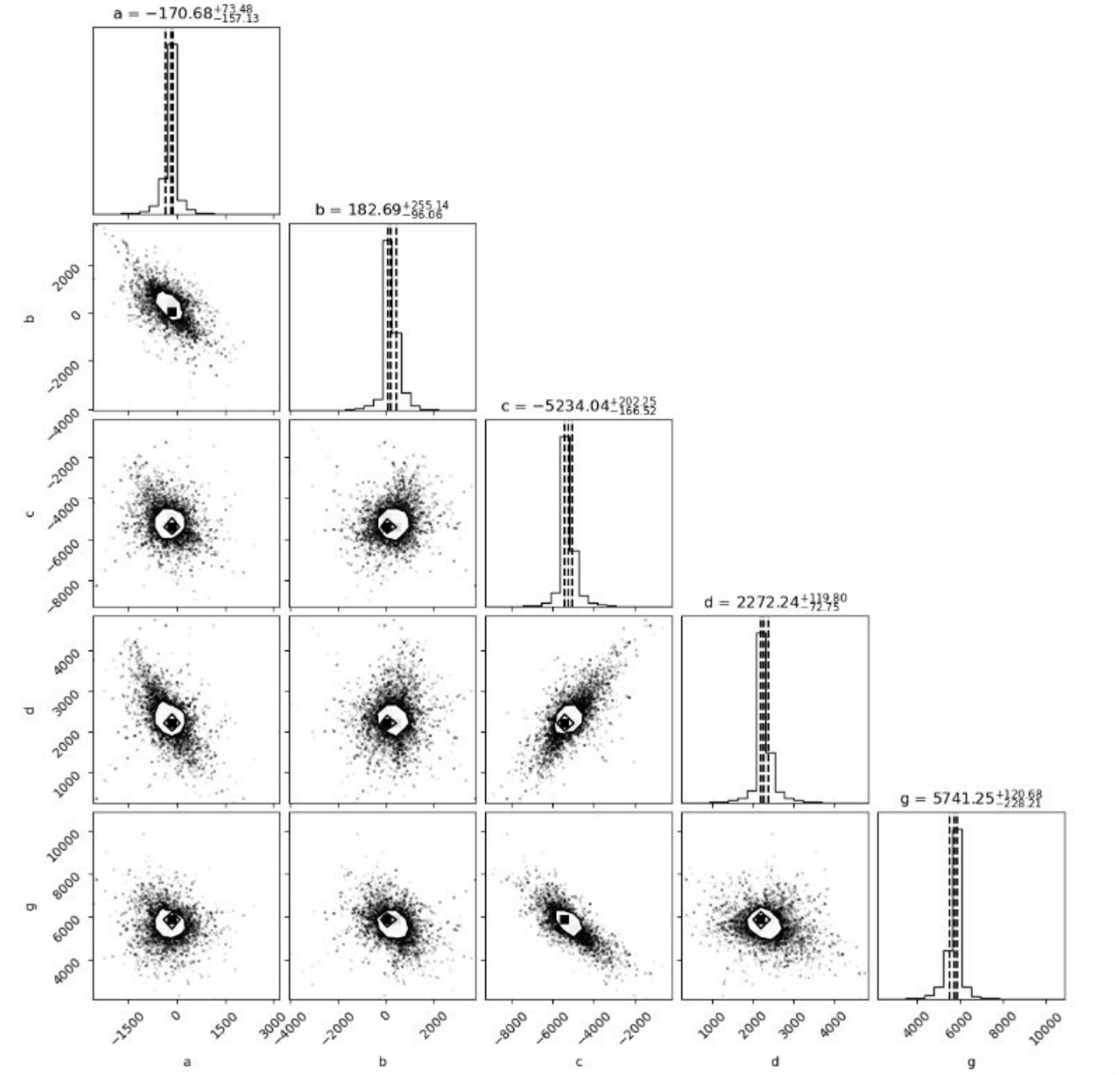


Figure 32: Here is the corner plot of output_26. This shows how each variable is correlated to another variable. Then the graphs on the far right are a cross section distribution of how each variable did against the rest. The more the cross section cut looks like a Gaussian distribution the better the correlation. Here it can be seen that there are two peaks in a lot of these cross sectional plots. This means that it would be advantageous to either redo the model or let the simulation run for a longer period of time for the program to identify which is the global minimum and which is the local minimum.

assembled and work well in the lab. For the optical spectrum it was found that a sine wave of pixel value versus motor angle could be obtained. On top of this, a phase that is useful to finding the orientation of the polarization of light could be obtained. The values obtained from "output_25" for Curvefit is 4.091211398769081 +/- 0.6805962025574508 radians and for MCMC is 4.28040400379436 +/- 0.5243057574298748 radians. These both give answers that are within each others error bars, which gives it some validity. The values obtained from "output_26" for Curvefit is 3.38712987592558 +/- 0.6254786914332024 radians and for MCMC is 2.549369996077477 +/- 0.1831174360754355 radians. Here the Curvefit and MCMC do not match as nicely. Neither of the phases say that the other phase fits in the error bars. This does raise some concern about how the models evaluated the answers and should be investigated more. Look at Figure 33 and Figure 34 for a complete picture of "output_25" and "output_26", respectively.

In the future the goal is to take what was done in the lab and take data in the field. This means taking the CAD model of the drone calibrator, Figure 3 and Figure 4, assemble it, attach it to the drone, and take a test flight. Then the polarimeter could take data of the source while it is flying on the drone. This would be monumental to see if the experiment would be feasible or if there are any additional unseen hurdles.

Continuing on one would need to figure out how to find the orientation of the drone relative to the zenith. Here two GPS HATs would be implemented, one for the drone and one for polarimeter. Additionally the camera on the drone would help find the orientation relative to the stars in the sky via astrometry and relative to the ground via landmarks measured out by hand. All three of these different measurements would give the orientation of the drone. Then using this orientation, the polarization angle of the light source, and some fancy math one could find the orientation of the polarization of the background ambient light. This is a huge generalization of the process, but it gives a small window into what else can be done with this project.

If this proof of concept works well, then next this exact process can be done for a millimeter light source, which would then give insight into the Cosmic Microwave Background. Finding the angle of polarization of different parts of the CMB would provide insight into whether the early universe was affected by gravitational waves, possibly from inflation. Again if this works well it could be scaled up and done with a CubeSAT instead of a drone as the calibration source and Simon's Observatory as a finer tuned polarimeter. This is a generalization and very optimistic, but it is important to show that this project is scalable and provides practical long term science.

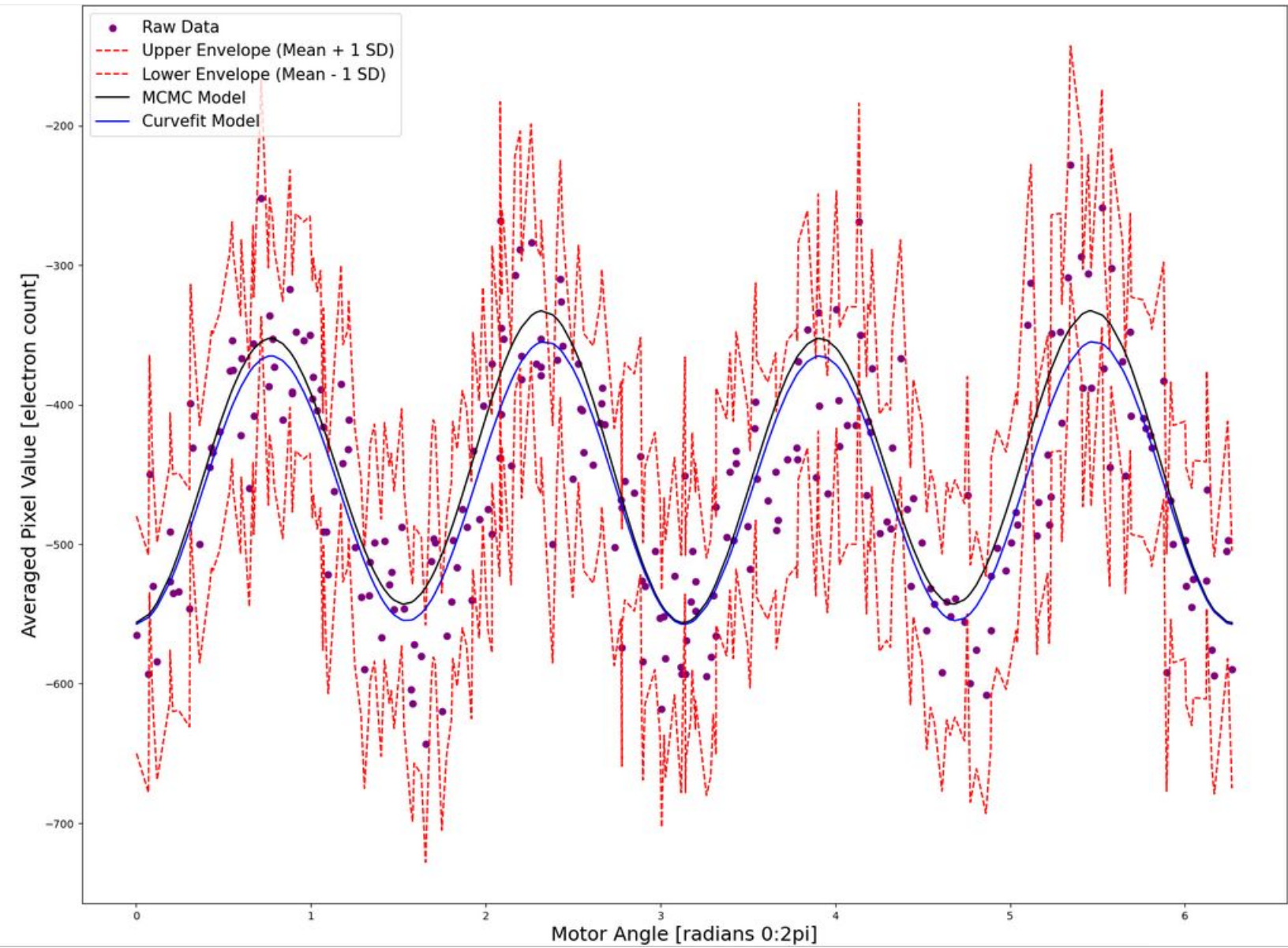


Figure 33: This is the Curvefit model, MCMC model, the raw data, and two extra lines finding the bounds one standard deviation away from the mean. This creates a complete overview of the data.

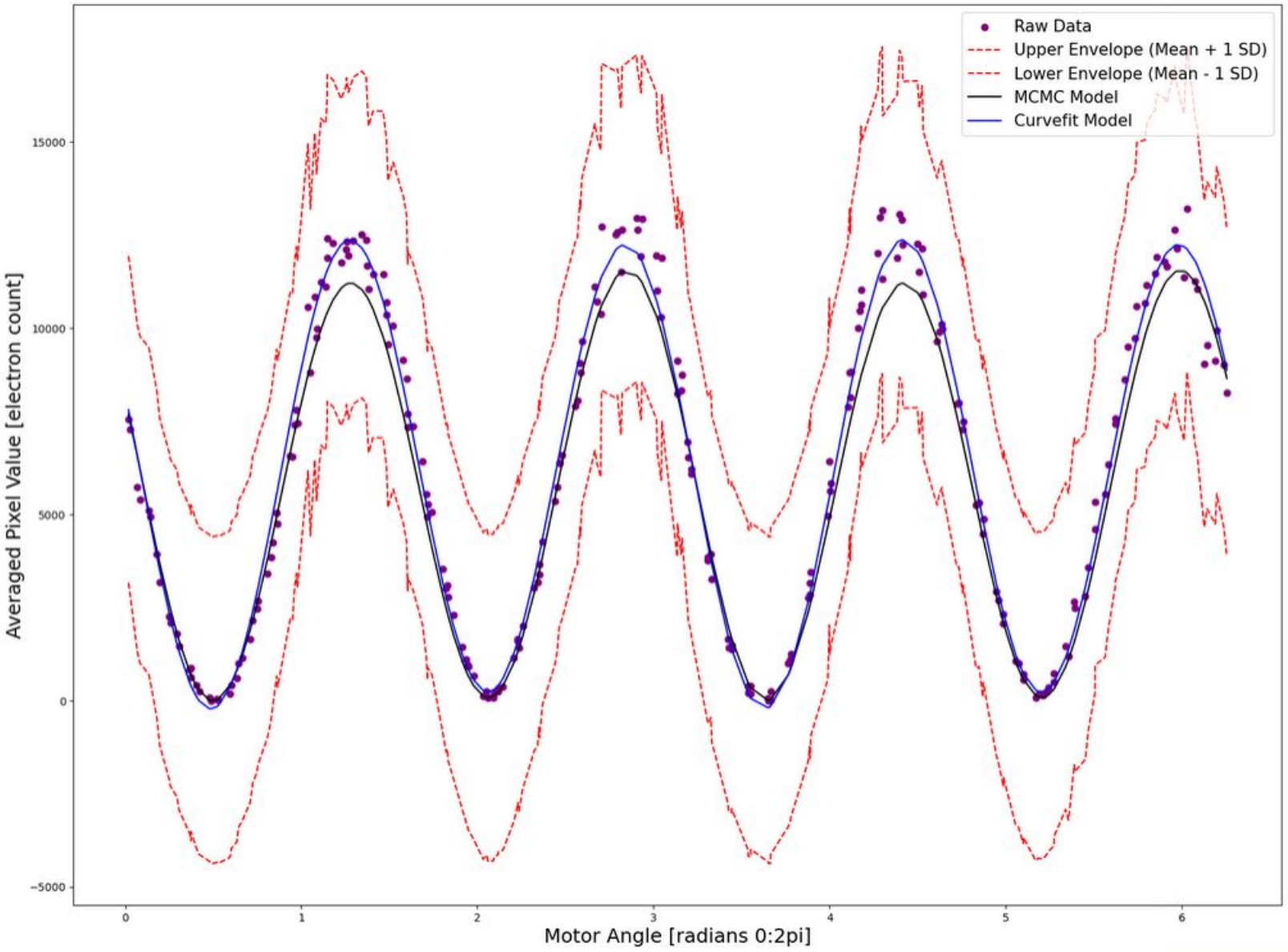


Figure 34: This is the Curvefit model, MCMC model, the raw data, and two extra lines finding the bounds one standard deviation away from the mean. This creates a complete overview of the data.

# References


[1] *3D CAD Design Software*. https://www.solidworks.com. 2018.

[2] Peter Ade et al. "The Simons Observatory: science goals and forecasts". In: *Journal of Cosmology and Astroparticle Physics* 2019.02 (2019), p. 056.

[3] *AMT-17C-1-036 — AMT Cables — Encoder Accessories — Rotary Encoders*. https://www.cuidevices.com/product/motion-and-control/rotary-encoders/encoder-accessories/amt-cables/amt-17c-1-036.

[4] *Arducam Fisheye Low Light USB Camera for Computer*. https://www.arducam.com/product/arducam-fisheye-low-light-usb-camera-for-computer-2mp-1080p-imx291-wide-angle-mini-h-264-uvc-video-camera-board-with-microphone.

[5] *Arducam for Raspberry Pi Ultra Low Light Camera*. https://www.arducam.com/product/arducam-for-raspberry-pi-ultra-low-light-camera-1080p-hd-wide-angle-pivariety-camera-module-based-on-1-2-7inch-2mp-starvis-sensor-imx462-compatible-with-raspberry-pi-isp-and-gstreamer-plugin.

[6] *Arducam Raspberry Pi Camera Solution*. https://www.arducam.com/raspberry-pi-camera-solution.

[7] *ASI178MC (Color) – ZWO ASI*. https://astronomy-imaging-camera.com/product/asi178mc-color. 2016.

[8] *DigiKey Electronics - Electronic Components Distributor*. https://www.digikey.com.

[9] Michael Don, Mark Ilg, and CCDC Army Research Laboratory Aberdeen Proving Ground United States. *Quadrature Decoder Design Using Microcontroller On-Chip Configurable Logic*. Tech. rep. 0019. 2020.

[10] *Free CAD Designs, Files & 3D Models — the GrabCAD Community Library*. https://grabcad.com/library. 2019.

[11] *Fujinon DF6HA-1S 6mm 1.5MP 1/2" F/1.2 - F/16 C-Mount Lens*. https://machinevisiondirect.com/products/fujinon-df6ha-1s.

[12] *Fujinon HF50HA-1S 50mm 1.5MP 2/3" F/2.3 - F/22 C-Mount Lens*. https://machinevisiondirect.com/products/fujinon-hf50ha-1s.

[13] Charles J. Geyer. "Practical Markov Chain Monte Carlo". In: *Statistical Science* 7.4 (1992), pp. 473–83.

[14] *IMX291LQR PDF*. http://pdf1.alldatasheet.com/datasheet-pdf/view/1132546/SONY/IMX291LQR.html.

[15] Adafruit Industries. *Raspberry Pi Camera Module 3 Standard [12MP Autofocus]*. Accessed: May 6, 2024. 2024. URL: https://www.adafruit.com/product/5657?src=raspberrypi.

[16] Douglas Kerr. *Chrominance Subsampling in Digital Images*. 2012.

[17] Sándor Laki et al. “P4Pi: P4 on Raspberry Pi for networking education”. In: *ACM SIGCOMM Computer Communication Review* 51.3 (2021), pp. 17–21.

[18] Xin Li, Bahadir Gunturk, and Lei Zhang. “Image demosaicing: A systematic survey”. In: *Visual Communications and Image Processing* 6822 (2008).

[19] *MCMC*. `https://prappleizer.github.io/Tutorials/MCMC/MCMC_Tutorial.html`.

[20] Michał Podpora. “YUV vs. RGB–A comparison of lossy compressions for human-oriented man-machine interfaces”. In: *III SWD conference proceedings*. Glucholazy, 2009.

[21] *QSB Quadrature to USB Adapter*. `https://www.usdigital.com/products/accessories/interfaces/usb/qsb`.

[22] *Raspberry Pi OS*. `https://www.raspberrypi.com/software`.

[23] Ariel Rokem. “A short course about fitting models with the scipy.optimize module”. In: *Journal of Open Source Education* 2.16 (2018).

[24] *Scipy.optimize.curve$_f$it|SciPyV*1.13.0*Manual*. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`. 2024.

[25] *ServoCity - Servos, Actobotics, Gears, Motors and More!* `https://www.servocity.com`.

[26] *Sony Starvis IMX178LQJ-C*. `https://www.framos.com/en/products/imx178lqj-c-13508`.

[27] Suzanne Staggs et al. “Rep. Prog. Phys.” In: 81 (2018), p. 044901.

[28] *Stepper Motor HAT - Waveshare Wiki*. `https://www.waveshare.com/wiki/Stepper_Motor_HAT`.

[29] G. J. Sullivan and T. Wiegand. “Video Compression - From Concepts to the H.264/AVC Standard”. In: *Proceedings of the IEEE* 93.1 (Jan. 2005), pp. 18–31. DOI: 10.1109/JPROC.2004.839617.

[30] *Thorlabs*. 2024. URL: `https://www.thorlabs.de/thorProduct.cfm?partnumber=CCM1-WPBS254/M`.

[31] Kip S. Thorne. *Black Holes and Time Warps: Einstein’s Outrageous Legacy*. New York: W.W. Norton, 1994.

[32] *Ubuntu 22.04.2 LTS (Jammy Jellyfish)*. `https://www.releases.ubuntu.com/jammy`.

[33] G. K. Wallace. “The JPEG still picture compression standard”. In: *IEEE Transactions on Consumer Electronics* 38.1 (Feb. 1992), pp. xviii–xxxiv. DOI: 10.1109/30.125072.

[34] Brian Ward. *How Linux works: What every superuser should know*. no starch press, 2021.