Astronomy Made Easy: An Overview of the Design and Fabrication of an All-In-One Stargazing Solution

Rich Baird, Miguel Gomez, Tyler Liddell, Hyrum Saunders
University of Utah Computer Engineering

***Abstract*—Society has seen considerable benefits from science and engineering projects when done on a large scale. Minimizing the costs of these projects allows ordinary people to enjoy the fruits of that labor. One such realm that many would like to enter is astronomy. The first telescopes were very cost-prohibitive and took an unquantifiable amount of funds and resources to see construction. While they have come a long way, such as the Hyperia telescope designed by Vaonis, the costs are still too significant for most but low enough for more amateurs to join the astrophotography game. The designs discussed are comparatively inexpensive and more manageable for those who have some photography gear on hand–enabling avid photographers to turn their lenses to the sky. While it would be best to create a system that does not come with an impact on the environment, an analysis of the various aspects of our strategy to ensure there is as little impact as possible must be considered. The materials, size, and cost will be essential for success in creating a design focusing on a small form factor while still building a tracker that can perform.**

***Index Terms*—application programming interface (API), android package (APK), digital single lens reflex (DSLR), microcontroller unit (MCU), publication subscription (Pubsub), small board computer (SBC)**

I. Introduction

FOR too long there has been an enormous barrier to entry in astrophotography. As humans, the authors consistently look to the sky in awe, yet the authors do not have the means to capture these inspiring sights. Many telescope companies exist which allow you to gaze upwards, but the costs are too great for everyone to take part in the capture process.

Astrophotography requires lots of technical knowledge: knowing how quickly stars move across the sky so you can track them properly, setting up specialized equipment to achieve a smooth track at the correct speed, configuring your camera settings just right for the perfect exposures, and many other minute details you need to know to achieve high-quality photographs of the night sky.

Additionally, the cost associated with the various instruments needed for astrophotography is prohibitive for most people. Getting started in astrophotography will cost a minimum of $600 for equipment alone, and that will only get you used, inferior equipment. If you want to get truly amazing pictures, professional astrophotography equipment can cost $3,000 or more. That is only taking into account the hardware; much of which doesn't function properly without the software licensed with the hardware, which can cost you $100s extra each year [1]. Other automated, all-in-one astrophotography stations exist. But they are few and far between and can cost anywhere from $4,000 and beyond [2].

The proposed solution is a cost-friendly system that will take care of all the technical details while capturing these inspiring sights; an automated point-and-shoot system that takes images of stars using a Sony camera as the sensor. An app controls a telescope and camera system. The ability to have a device that could be easily printed and fabricated will allow amateur astronomers to enter the astrophotography community without the burden of needing to spend an exorbitant amount of time and money gaining specialized photography skills and instruments.

This simplest approach will involve using existing hardware (a camera and lens) with custom 3D printed hardware for the mount and movement system, driven by motors and voice coil actuators. Software will automate camera movements, capture pictures, and will include a user application on a phone to select which celestial bodies are to be photographed.

A system that can track a celestial body requires projection mapping or spherical coordinates. A method of calculating changes in position over time will be implemented on a microcontroller (MCU) or small board computer (SBC) interfacing with a star-tracking Application Programming Interface (API) to obtain astronomical data. A few options exist for implementing such designs in hardware alone or could allow for some more software to handle the heavy lifting.

Taking pictures in remote areas will spawn software problems, as the software must work without a network connection. When connected to a network, the system can download information on celestial bodies, but once disconnected from the network it must perform its own calculations to track celestial bodies based on the last known location (the downloaded information) available to them.

There are additional complications beyond hardware and software issues. As governments, corporations, and even private organizations continue filling the sky with low earth orbit (LEO) satellites, the night sky is becoming more difficult to view from earth. Photographing the night sky results in streaked pictures as satellites travel at different speeds than stars [3].

For now, it is still possible to capture the night sky, provided a plan of when and where it will be done, to reduce satellites. As the sky continues to fill up, this may become impossible. Future astrophotographers will need to have their own high-orbit satellites in space to capture high-quality photos. When

that happens it will require some truly extraordinary projects to eliminate the barrier to entry for astrophotography.

II. Background

Whereas the monetary barrier to entry may be overcome with sufficient resources, the technical barrier to entry is not so easily overcome. Successful astrophotography requires at a minimum, a rudimentary understanding of the fields of photography, astronomy, and geometry. Several tools exist to lower this barrier of entry and reduce the level of knowledge required to be successful. These systems fall primarily into one of four categories: alignment devices and systems, image processing software, star tracking tools, and all-in-one solutions. There are several difficulties associated with the hardware and software. Large gear ratios will be needed to make any camera movement as smooth as possible so that when photographing things such as time-lapses, the picture quality will not suffer from small movements. Balance will also be a primary concern, the system needs to remain small and lightweight, but counterweights may be necessary to keep everything balanced. The description and purpose of each, as well as some examples of existing solutions, are detailed herewith.

A. Alignment Devices and Systems

To capture an image of a celestial body many light years away, long exposure times are required. This poses a significant challenge for a camera exposing the image from the perspective of the earth. As the earth rotates, the objects in the sky appear to move from the camera's perspective. Thus, with sufficiently long exposures, the image will be exposed in multiple positions in the same image, creating a motion blur effect. In some cases, this is desired, such as when photographing star trails as shown in Fig. 2. In the case of photographing a static image, however, such blur may detract from the photographer's intention. To overcome this issue, careful alignment of the camera mount with the north pole, allows the photographer to fix one axis of movement, and rotate the camera precisely as shown in Fig. 1 with the movement of the earth. There are several commercial systems available to assist with this including any of a number of equatorial mounts, Polaris spotting scopes, and alignment apps for the phone or desktop. These are extremely useful; however, they all assume that one understands the underlying importance of proper equatorial alignment, as well as how to locate a north indicator such as Polaris in the sky.

B. Image Processing Software

Another challenge related to the long-exposure nature of astrophotography is noise. Longer exposures are required to obtain enough detail to distinguish the object from the night sky. While exposing for longer times allows the image to be properly exposed, it also creates the potential for noise. Consider that exposure is to a digital light sensor as gain is to a microphone. In order to obtain high-fidelity recordings, a large gain is required, however, the challenge becomes filtering the intended recording from the background noise the microphone will inevitably pick up. This is why expensive microphones typically include directional sound filtering, active noise canceling, and other features that will help to increase the signal-to-noise ratio.

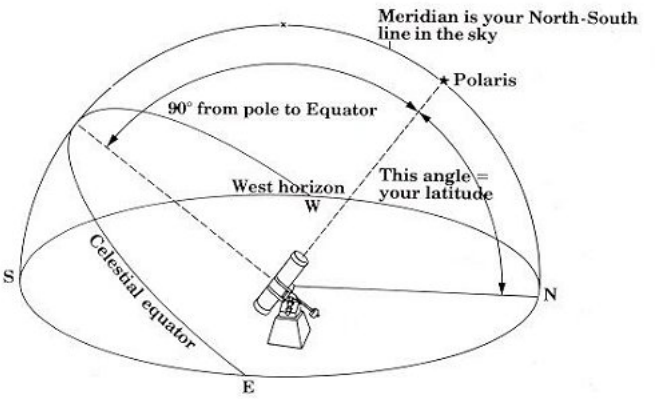

Fig. 1: Diagram of polar alignment using Polaris (A.K.A. The North Star) [4]

One way to increase the signal-to-noise ratio in photography is a process known as photo stacking. Photo stacking is the process of taking multiple shots at varying exposure lengths, rather than a single shot exposed purely for the subject, and stacking the highest fidelity portions of each image while masking their low fidelity counterparts. This creates an image that is well exposed across the entire image, and not just for the subject. Before digital photography, this technique was popular with professional film photographers, who would do the work of stacking and masking in a dark room using real film. Today there exist many tools and computer algorithms aimed at automating this process. One of the most popular of such tools is deep sky stacker, an open-source tool written for windows. Other implementations exist as libraries based on OpenCV. Proprietary tools such as Adobe Photoshop are also popular. Photo stacking and image processing software has been around for a long time, however, it has always been an additional step. Integrating this technology into a seamless pipeline adds value to the system by further reducing the technical barrier to entry, and reducing the mean time to a finished product. This is accomplished in the final product utilizing algorithms available as part of the OpenCV library.

C. Star Tracking Tools

It doesn't matter how much equipment one has if they don't know where to find the image they want to photograph. This is where star tracking tools come into play. There is no small number of tools with varying levels of user interactivity featured. Some simply provide coordinates, while others display images and constellations on a mobile phone in real-time as it is panned about the night sky. The final system calculated the right ascension and declination of requested stars, having the mount setup convert those into necessary motor movements. This process is explained in detail in later sections.

Fig. 2: Circumpolar stars in star trails [5]

D. All-In-One Solutions

Aside from the proposed solution, few all-in-one solutions exist for this niche. One notable exception is Stellina by Vaonis. Stellina is an all-in-one smart telescope designed specifically for astrophotography that incorporates all of the same features noted here. At a price tag of $3,999.00 however, it is priced at the same level as many of the professional tools, so while it boasts a truly modern form factor and polished mobile app, it remains cost prohibitive to be purchased for a hobby and too limited in its output and configurability to be useful for enthusiasts and professionals. The system proposed here is configuration agnostic, and while providing a streamlined start-to-finish solution, leaves the raw image files available to be processed by hand if so desired.

III. Related Work

Careful alignment of the mount is required not only initially, as discussed in the background section, but also while tracking the image across the sky. In his several works on the topic, Suzynski introduces and refines a novel way to maintain the alignment using an affordable webcam and a convolution on the brightness signal received from the webcam. [6] [7].

Image stacking is a common technique for increasing the signal-to-noise ratio by stacking images atop each other and masking to reveal only the best exposed portions from each image. Whereas this is a computationally intensive task that typically is reserved for post-processing, authors Zhou and Yu demonstrate a method for performing this process in real-time using the correlation of the signal phase [8].

A mention must be made of Suzynski's detailed explanation of his full ground-based astronomical observatory [9]. His setup incorporates all the features and several more that the authors intend to implement. Although it is evident that great care has been taken with each design choice in the build, Suzynski's design is inherently immobile. Many design considerations are made in the build trade portability for image quality and focal distance. As the intent of the design is to be a true personal observatory, these decisions make sense. This, however, is antithetical to the design of the authors, that is, to implement a portable astrophotography station. Still, this entry bares acknowledging and further examination, as portions of the design may provide guidance during the design and assembly process.

IV. Project Components

A. Overview

The system required many design choices. The following section will contain in-depth information on each component and how they function together to create a cohesive unit. Fig. 16 in Appendix A shows an overview of the entire system.

B. Alpaca

The project followed the ASCOM Alpaca Interface, which is an industry-standard interface provided for telescopes, cameras, and other mount devices. In this case, the telescope and camera interfaces were followed. Alpaca provides definitions for functionalities such as $startexposure$, $slewtocoordinatesasync$, $imagearray$, and more. These endpoints provide a definition of what data is passed through the interface and what data is passed back. Alpaca does nothing to implement any of this functionality, but rather defines the methods to create a standard. Alpaca uses HTTP PUT and GET requests with specific body information included for each endpoint. The advantage of using Alpaca is that one may switch out their mount system with another mount system that is also Alpaca compliant, and the software sending these requests can still control the new mount. On the flip side, one may use different control software and their mount will still be controllable. Looking again at Fig. 16 in Appendix A, the project can be viewed as a "left side" and a "right side" where Alpaca requests are created and sent from the left side, while the mount system satisfies those requests and responds on the right side. Fig. 3 shows an extremely simplified view of the setup, highlighting the place of the Alpaca interface. Choosing to follow the Alpaca Interface was done very early

Fig. 3: Very simplified design of system

in the project development, and influenced many of the design choices made. It required a split of the left side and right side, and also required any information passed from the two sides to go through the Alpaca Interface, while also limiting what information was possible to the defined endpoints from Alpaca.

C. Software-left side

The "left side" of the interface contains all software necessary to provide a UI for a user as well as sending Alpaca requests through the interface. It includes the StargazerApp, an app written with the Flutter framework, the StargazerServer-AlpacaClient, responsible for interfacing with many other programs, and the ImageProcessor.

1) StargazerApp: The StargazerApp was an app written with the Flutter framework and is compatible with Android, IOS, and even as a web app. The app implemented an "Autonomous Mode", which allows users to search for stars that are visible to them in the sky according to their latitude as well as their sidereal time. Once a user has selected a star they would like to photograph, they can select among many qualities, which dictate how many images to take of the star, and for how long each image should be. The purpose of these multiple images is explained later with image processing. Users are able to add many different stars, reorder, or remove them. The app also supports user-specific favorites which persist whenever reopening the app. Once all of the images have been taken of the requested star and then processed, it appears as a finished entry in the queue, where the user is able to view the final image. The app supports an arbitrarily-long queue, meaning a user may select many stars to be taken in a row, and then leave the setup to take all of the requested pictures. The StargazerApp, from the user's perspective, removes much of the required knowledge of astrophotography and provides an easy interface to select the stars they want while the system takes care of the rest.

2) StargazerServer-AlpacaClient: The StargazerServer-AlpacaClient is a group of Python threads that work together to fulfill much of the left side. The AlpacaServer is responsible for communicating with any users from the StargazerApp. It runs a server, that receives connection requests from the app, which it accepts, creates a client connection, and hands off to a new thread to receive data. The communication between the StargazerApp and the StargazerServer uses a custom JSON interface, which allows for actions such as getting the current star queue, requesting to change this queue, searching for visible stars, getting and setting favorite stars, and getting finished pictures. Because individual threads handle client connections, multiple concurrent users may connect to the StargazerServer and view or modify the queue, search stars, and display completed images simultaneously. This communication is implemented on top of TCP sockets. To allow this connection to occur easily, the Raspberry Pi was set up as an access point, where users can simply connect to it as a WiFi network. A network bridge was also set up on the Pi to allow connected devices to still have access to the internet through that same WiFi network. This setup was chosen due to the ease of use with TCP sockets, as well as the built-in error detection and guaranteed delivery provided by TCP.

The StargazerServer uses the Rhodesmill Skyfield library, which allows reading from a .dat file which is a list of celestial catalog entries and their absolute positions at a specific time. Stars move very predictably, so the library computes the right ascension and declination of those same objects given a sidereal time. This project used the Hipparcos Catalog, which is a catalog of 118,218 stars and their positions at a past time. These entries were fed into Skyfield in order to calculate the current right ascension and declination, which is used later. Using this library and the mentioned .dat file allows the system to work completely remote, without any internet connection. This is important for this application, as dark sky locations are generally very remote. Given a specific latitude, some stars will be visible at some times of the night, some stars will always be visible, and some stars will never be visible. Given a user's current $latitude$ on the earth, any stars with a declination in the range $90 > declination > (90 - latitude)$ will always be visible to that user. Any stars with a declination of $(90 - latitude) > declination > latitude$ will be visible at only certain points in the day, as they will pass behind the horizon. Lastly, any stars with a declination of $latitude > declination > 0$ will never be visible at that latitude. The StargazerServer maintains which stars are visible from its current position, and only provides these stars when given a search query from the StargazerApp.

After a user has made a request through the StargazerApp into the StargazerServer, a request is put into a queue to be sent to the AlpacaClient. The AlpacaClient is responsible for talking with the Alpaca Interface, sending requests for the telescope to slew, start or stop tracking, begin exposures of specified lengths, getting image data from the last exposure, and more. The requests the AlpacaClient receives from the StargazerServer contain information such as what star is to be imaged, how many images, and for how long. It first computes the current right ascension and declination of that star and calls the $slewtocoordinatesasync$ endpoint in the Alpaca Interface. The specifics of how the right side of the interface function will be discussed later. Once the telescope has finished slewing, the AlpacaClient requests for it to begin tracking in order to keep the camera pointed directly at the star for all of the exposures. The AlpacaClient takes the requested number of pictures for the specified duration using the $startexposure$ endpoint in the interface. After completing each exposure, it requests the image data from the $imagearray$ endpoint, and stores it in Redis, the database of choice, also keeping track of the key the images are stored under. After all the images are complete, the AlpacaClient will put a request into the outgoing queue going to the image processor thread and is now able to take the next star request out of the incoming queue.

The image processor thread in the StargazerServer-AlpacaClient program simply interfaces with the ImageProcessor Go program, which is explained in the next section. Putting this waiting in its own thread allows the AlpacaClient to continue to slew the telescope and take images, while the image processing works on any previous jobs. After an image has been stacked and processed, the image processor thread puts a finished entry into a queue going to the StargazerServer. From here, the StargazerServer may display the finished entry and allow users of the StargazerApp to request the final image from a job.

3) Image processing: The image processor depends on OpenCV to perform the heavy lifting. A C-compatible API wraps the relevant OpenCV C++ functions to facilitate interfacing with the GO front end. Go listens for requests from the relevant Redis channel in a separate thread and upon receipt queries the image data from the Redis image database. Go is chosen for this task due to its performant nature and pre-existing libraries for interfacing with Redis. As a so-called high level language, Go provides many convenience features

that C does not but also has a C compatibility layer for interfacing directly with C when the raw performance of the C language is required. Once the Go front-end has received the image data from the Redis database, the data is passed directly to the C program where OpenCV is used to align and stack the images.

For alignment OpenCV relies on the enhanced correlation coefficient (ECC) maximization method is used to align the images, and the values are stacked and normalized as 8-bit color values. The ECC method relies on a method described in a paper by Evangelidis et al. [10]. The essence of the method is the correlation between the pixel values of two like images. A threshold is provided to terminate the process once the aligned pixels are within the given threshold. To facilitate this method, the images are first converted to grayscale so that pixels may be compared directly without the need for exponential looping that would otherwise be required to account for different color channels. Though the process is a linear one, the number of iterations required to reach a given threshold can be enough to greatly overwhelm even capable desktop machines. Thus the parameters are chosen such that the process may be complete within a threshold of three minutes.

Once stacked, the image has a false color profile applied to convert the grayscale image to a full-color one. This is similar to the approach taken by NASA radio and IR telescopes, except that NASA images a single subject with this process multiple times using separate filters each time so that each emission spectrum may be individually recognized and analyzed.

4) Redis: Redis is a scalable, in-memory database designed for rapid transit of large amounts of information. Incorporating both a database and messaging service, Redis serves as the communication relay between the Alpaca server and the image processor and camera controller. Messages are encoded as JSON objects with an ID, command, and arguments or payload, depending on whether the message represents a request or a response. As Redis only provides a one-way messaging protocol, the ID is preserved between request and response messages so that the two may be correlated after the fact.

D. Software-right side

As opposed to the left side, the "right side" of the interface deals with receiving requests across the Alpaca Interface and controlling the camera and telescope according to those requests. It includes a Django server, the Camera Controller, and a Raspberry Pi Pico responsible for controlling the mount setup.

1) Alpaca Server: Our alpaca server uses Django, a python web framework. In retrospect, this may have been a bit much for our simple use case, especially running on a pi. It is likely we would have seen performance improvements using something more lightweight, like Flask. We ended up using Django because we were more familiar with it at the time. In our server, we included support for all of the alpaca endpoints. As mentioned previously, Alpaca simply defines what information is sent and received over different technologies. We used the Alpaca HTTP interface, so we had to implement URL endpoints for all the telescope and camera functions that Alpaca supports. Building this out was a bit tedious since most of Alpaca was not directly applicable to the project. A custom interface would have been much simpler and easier to implement. Ultimately Alpaca was used because it has the benefit of making the mount compatible with any alpaca-compliant controller or app.

The most interesting parts of the Django server are a specialized queue for UART communication with the Pico - this takes all the different threads running on the Django server for each individual request, and makes them synchronous before they are sent to the Pico since the Pico can't handle the same sort of concurrency that a pi can. When the Django server gets a response back from the Pico it is split back into the proper thread using unique IDs. A timeout decorator was also built for communications with the camera and with the Pico so the Alpaca server won't get hung up on a request if something in another part of the project breaks. A lot of effort was put into making individual parts of the project safe against other parts of the system breaking. Keeping different sections of the system separate and only interacting through robust communication channels allows easy recovery from errors. The project is capable of recovering from errors that crash separate parts of the project. As an example, if the image processor crashes because the pi runs out of memory while trying to stack images, it will immediately be restarted and all the other pieces of the project continue functioning in the meantime.

2) Camera Controller: The camera controller listens for a Capture request from the Redis command channel. Once received, the controller defers to the GPhoto2 command line interface to begin a capture for the time provided as an argument in the request object. GPhoto2 is an open-source tool and library provided for the purpose of remote-controlling digital cameras. This software is extremely popular amongst technical photographers where precision is of utmost importance. The image data is streamed back to the caller and saved in memory to be delivered to the Alpaca Server upon request.

3) Pico-Interfaces: The mount is an equatorial mount. A later section will go into more depth, but the benefit is that it allows the use of two motors that are completely independent of one another. Only one motor is needed to track a star, and while slewing the two can move one at a time. It is not necessary to move them together, like with an alt-az mount.

One of the motors on the mount controls the right ascension and the other declination, allowing for simpler logic and smooth tracking of stars. Both motors are stepper motors and are driven using two TMC2209 motor drivers. The Pico is communicating with the motor drivers over UART. TMC2209 motor drivers have a nice interface allowing for an angular velocity to be written to a register. It also allows more traditional control through pulses. This project uses both methods. The first method of writing to the register is used for homing, tracking, and declination movements. The second method is used for right ascension since it simplified the math used to find a star. This will be explained in more depth in the next sections.

The Pico is also connected to a nine-axis IMU sitting below

the camera that returns the current orientation of the mount in Euler angles, namely roll, pitch, and yaw. Converting the Euler angles into equatorial coordinates is used to slew the mount to the correct position. This will be described in the next sections. Last, the Pico interfaces with an OLED screen that displays information to the end user during polar alignment and throughout the operation of the mount.

E. Equatorial Coordinates

Before discussing the mount firmware operations, it will be helpful to have a basic understanding of equatorial coordinates. Equatorial coordinates are analogous to spherical coordinates except they are independent of the observer's location on the earth and the time of observation. Right Ascension and declination are absolute, and the observer's position and time on the earth create an offset right ascension and declination that must be handled in calculations. Another way of thinking of it is that right ascension and declination are like longitude and latitude, except instead of giving a location on the earth they are giving a location on a giant imaginary sphere in space around the earth.

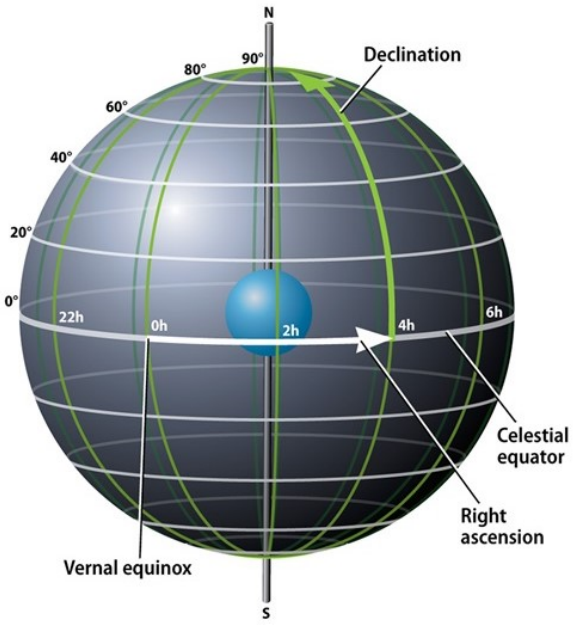

Fig. 4: Equatorial Coordinates [11]

F. Mount Firmware

The Pico has two cores. One core is devoted entirely to controlling the mount. This includes calculations to find stars and convert Euler angles to equatorial coordinates. The very first thing the mount does on startup is the process of polar alignment. This is the only process in the project that is not completely automated, as it requires the end user to make sure the entire assembly points north and tilt the mount once the OLED screen instructs them to do so.

Polar alignment first requires the end-user to align the front of the mount with the north. After this is done the declination motor angles the camera to ninety minus the mount's current latitude, in degrees above the horizon. At this point, the OLED screen is used to help the user tilt the entire mount until the camera is pointing to the horizon. Finally, the declination motor once again angles the camera, this time to the latitude of the mount, in degrees above the horizon. The mount is now polar aligned and ready for use. This entire process is very easy for a user, as they only need to adjust the mount twice, and the OLED screen will tell them exactly what to do.

The rest of the mount software was more complicated. Finding the right ascension and declination that the camera was currently pointing to proved to be one of the most difficult aspects of the project. Many of the sensors that were tested had too much drift in every axis, making calculations unreliable and resulting in oscillatory movements in the mount. When a suitable sensor was finally found, the math itself posed another challenge. Euler angles can be converted to Right Ascension and Declination using trigonometry. Declination took some work, but it worked well using this method. Right ascension is more difficult since at one point in the rotation the yaw from the sensor would drift a tiny bit and flip between 0 and 360, throwing off calculations.

To prevent this the mount rotates the right ascension motor to a 'homed' position using a hall effect sensor between each slew and the right ascension motor is controlled using pulses instead of registers because the number of pulses can be counted to know exactly how many degrees the mount has rotated. This results in the declination always being the pitch of the sensor from the homed position, and the only offset in the right ascension is something astronomers call the 'local hour angle.' This is really just the local sidereal time. Local sidereal time can be determined from local solar time, and the time of year, since a sidereal day is approximately 4 minutes shorter than a solar day.

G. Startup-Docker Compose

All of the setup and running of the system is facilitated through Docker Compose. Components of the system were separated into different Docker images, such as the Image Processor, StargazerSetup, which includes the StargazerServer-AlpacaClient and AlpacaServer, Redis, and the camera controller. Each image included an entry point, so by specifying the startup order of each within the Docker Compose, the project could be started with simply "docker compose up".

The order in which Docker Compose brings all of the containers up is very important. First, Redis is brought up and initialized. Next, the Image Processor and camera controller are brought up. Finally, StargazerSetup is run. Docker Compose was also extremely useful in creating a robust design. The components of the project proved to be very capable and rarely errored or crashed, however hardware limitations on the Raspberry Pi, especially with the intensive nature of the image processor, could cause errors or crashes. The system was designed to be able to recover from errors in one single part of the project, which was aided by Docker Compose's restart feature.

1) StargazerSetup: As mentioned earlier, the StargazerSetup was one of the Docker images created. This container is responsible for getting and passing required data to different pieces through startup. The project used a Swarm Eval kit, with the Swarm M138 modem. The Swarm allows 2-way communication through Swarm's satellite network. The project primarily used the M138 in order to receive both GPS coordinates as well as receiving the current greenwich mean time (GMT). The Swarm can also be used to send data packets across the internet using satellite communication, however as discussed earlier, the entire project is able to operate without any internet connection. The M138 modem communicates over a specific UART NMEA format. StargazerSetup completes the following steps. First, it sets the message rate on the Swarm. Next, it waits to receive GPS data, extracts the latitude, and sends it to the Pico, which requires the latitude for its homing sequence. Once it gets a response from the Pico, it then starts the AlpacaServer. After that has started up, it begins the StargazerServer-AlpacaClient, which also requires GPS data as well as GMT data to calculate star positions. After all of these steps, StargazerSetup has finished.

2) Docker Compose-Flags: Many flags were also implemented that could be turned on or off in the Docker Compose to allow skipping of setup steps. These flags included skipping camera connection attempts in the camera controller, skipping waiting for GPS data to communicate and setup the Pico, skipping getting the required GPS and GMT data for the StargazerServer-AlpacaClient and using default hard-coded data, or creating a mock telescope and camera in the AlpacaClient, where custom objects are used in place of the camera and telescope that return predictable values on method calls and do not make any requests to the Alpaca Interface. Creating such flags allowed extremely quick tuning, development, and troubleshooting of the system without going through each setup step every time.

H. Hardware

The mechanical aspects necessary for the system to operate as intended required a combination of tight tolerances and smooth operation with little to no backlash. Many revisions of the printed parts as well as the choice of driving strength were needed to dial in a working system that was still within the range of power that we targeted.

1) Motors: To ensure these were not an issue, Miguel decided stepper motors should be used instead of DC motors due to their ease of control and ability to adjust accurately. The use of these stepper motors also helps with the amount of power needed to run the system as the steppers and drivers chosen for the project are able to run efficiently with lower power usage than others on the market. The TMC2209 motor driver from Trinamic is shown in fig. 5 and is one of the best in the business when a smooth and stable operation is of the utmost importance.

These motor drivers and steppers are not too expensive either since they have become a popular choice for 3D printer manufacturers. The motors have a $1.8°$ step size which is split with the 256-step sequencer within the tmc2209 driver

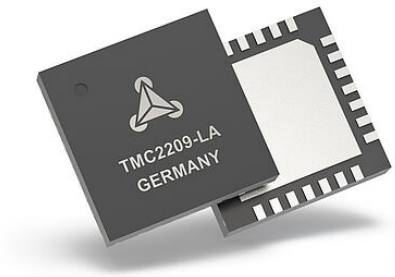

Fig. 5: Trinamic Driver [12]

Fig. 6: 42-34 Stepper Motors [13]

shown in fig. 17 Appendix A. The team initially planned on having custom boards for the drivers to go on and have those be part of the system that together broke out the signals to the motors. However, problems stemming from the cost of manufacturing a 4-layer board with buried signal traces, and parts acquisition constraints from sourcing the components to build these ourselves, became apparent once the PCBs were designed and ready for fabrication.

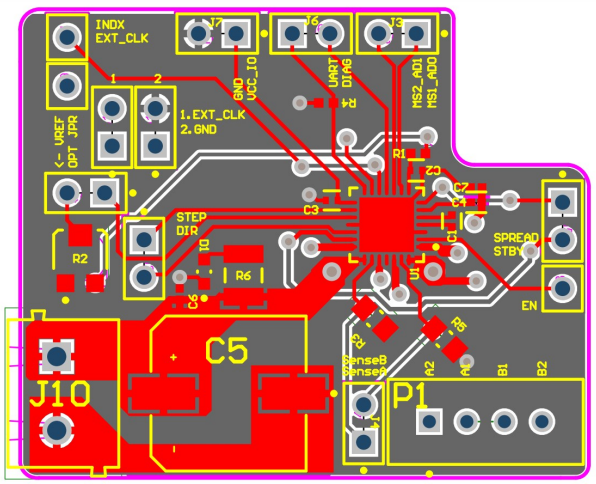

Fig. 7: Custom Trinamic Driver board

Thankfully, because these drivers are popular and used by

many in 3D printers, there exists a breakout board containing all the necessary components for use in the design. These boards were manufactured and sold by B.T.T.(Big Tree Tech) and are a popular choice in RepRap and custom core XY printers-allowing the plans of using the TMC2209 driver to continue.

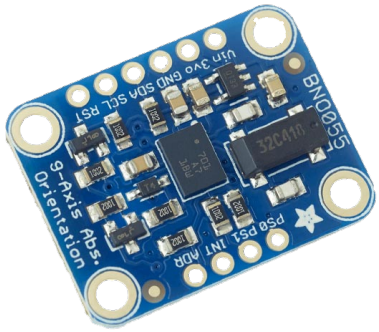

Fig. 8: IMU Adafruit [14]

2) IMU sensor: Figuring out the orientation of the mount and camera requires a sensitive and accurate sensor to supply the necessary Euler angles for the conversion to relative coordinates. After months of work attempting to get accurate data from multiple sensors, the Adafruit BNO055 9-axis I.M.U. was the obvious winner. This inertial measurement unit allowed the capture of correct orientation data after performing the calibrations and was not susceptible to the same drift issues experienced with the 3 other sensors explored in the testing process.

In addition to the hardware used for the sensing and driving of the system, the need to take into consideration of controllers that would do the heavy lifting was vital. To ensure the system met the power requirements, a Raspberry Pi Pico as the microcontroller for its two cores and low power consumption revealed itself to be a great option. As a bonus to these aspects, the team had some experience working with python, and using a microcontroller that used this framework was helpful in allowing the best programmers on board to work with the embedded portions of the project in a more comfortable environment.

I. Power

As astrophotographers would want to make their way out into the less populated parts of the world to get the best photos, it was essential to make the system work off the grid. To manage the power needs of the project and ensure the system had the ability to run without a power connection to some sort of generator, the use of a battery pack as the power source was settled. The battery would have to supply enough power to last at least a full night before powering down and would need the pack to stay cool to ensure the device does not have thermal runaway problems where it could cause a fire. The design of the battery would have been best using a newer technology, such as a Lithium-sulfur battery-which has been shown to carry more power per pound as compared to a lithium-ion cell, which has the negative side-effect of being a fire hazard [15].

Unfortunately, the cost of these cells is still too great to have them in the project and had to go with some lithium-ion cells instead. The cells chosen were Samsung INR18650-30Q batteries due to their discharge rate and power density. The batteries needed to be as close to each other in their internal resistance as possible to make sure there would be no overheating of the cells while discharging or charging. This design also called for a charging circuit as well as a balanced discharger for ensuring no individual cells were stressed.

Fig. 9: Samsung 18650 Cell Battery [16]

The final group of cells used came close to being exactly what was needed, even if the cells obtained were slightly outside the desired range when it came to their claimed internal resistance. This is an unfortunate effect that has been a problem in batteries for some time since there are many counterfeit cells out there. In this case, the cells are genuine but happened to be improperly stored at some time since they are higher than the average cell and roughly four times the internal resistance, as stated in the manufacturers' datasheets.

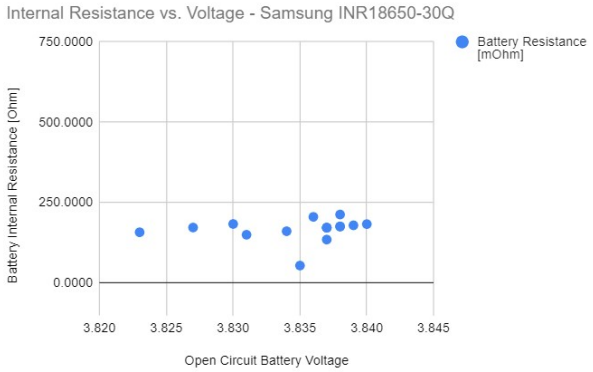

Fig. 10: Samsung 18650 Cell Resistances

Shown in 10, the resistances are higher than the expected $45m$ from the datasheets, but there is little variation given their

voltage - which is good for the pack we needed.

Creating a battery large enough to handle the draws expected from the system requires at least a 28V pack, given the lower end of the estimation from the draw seen with a Pico, Pi - 4B, two motors, and the various peripherals. As it should ideally be above the needed values, an 8 series 4 parallel pack was built to have the $4 \cdot 3Ah$ per battery of capacity and $8 \cdot 4.2V$ nominal voltage. Giving a pack of $34V$.

The total power requirements for the design were accounted for using the power estimates with this battery pack. table I gives us the average, as well as the high end of the expected peak draw, and the battery built can satisfy that with room to spare.

Table I
Power Requirements

Device	Max/Pk.	Avg.
Pico at 5V	2.5Wh	0.5Wh
Pi 4B at 5V	15Wh	8Wh
Motors at 25V	37.5Wh	6.25Wh
Noctua Fan at 12V	.6Wh	.6Wh
Buck Converters	10%	10%
Totals	61.1Wh	16.89Wh

percentage above refers to the losses in power conversion

J. 3D printing

The printing of the design was a considerable challenge. Since the group members are not structural or mechanical engineering students, the project started with a base design created by a user on Thingiverse. The design by user isaac879 was a great place to start the development of the setup needed for the implementation. This design was provided using a creative commons license and would not be the final design for a monetized assembly. Still, it was a great way to get started with modifications needed for the assembly and provide a good proof of concept build.

The design required many iterations to get the parts printed correctly. In addition, considerable work was required to get the printers in the lab into working condition to get the printing of the part without issues. This included re-leveling, disassembly, cleaning, fitting, tightening, and customizing the firmware on the machine to print with the necessary speeds to ensure accuracy. A few different materials were tested for their robustness, and high-impact polystyrene (HIPS) filament was the choice that won out for its durability and weight. Being very tough yet light allowed for the parts to hold everything together but not be so heavy that it became a problem for the motors. It combines the hardness of polystyrene with the elasticity of rubber to produce a high-impact thermoplastic that is tough and strong without being brittle.

1) Slip Ring: To ensure the design was functional and simple to use, a slip ring was used to pass the data and power lines from the lower part of the system to the upper part that needed to rotate 360°. This part was essential and tough to integrate as the data lines must be handled carefully when introducing noise. This slip ring is visible in fig. 15 and 11. This part caused some needed revisions to the base design, and changes were made to facilitate this addition. The slip ring design allowed the entire assembly to be removed in case something needed to be worked on by disconnecting the cables to the PCBs. Since signaling within the slip ring is can be a problem for some applications, special care was needed when making connections to ensure the signal integrity was not compromised. For USB, a modification to a USB 2.0 cable was made to force a connected device to see it as a hub. This allowed the signals to be read on either end without issues. This design worked well with a laptop, but the Raspberry Pi 4B would not connect without another hub connected between the male USB-A side and the Pi.

Fig. 11: Slipring type - 24 wire used, 12 wire pictured

2) Bearing: One of the toughest parts of 3D printing the entire assembly and gearing setup was the tolerances needed to ensure proper movement in the gears and the internal bearing assembly. There were many revisions to the bearing that caused some grief in the early days of the project, and purchasing a bearing from McMaster Carr was considered due to this complexity. However, the cost of purchasing such a niche part was too great. A few materials were floated as the ones to use for the ring, and iterations of the bearing were long and hard. Due to the size and shape of the bearing, printing in resin took a considerable amount of time with the printer in the Lab. The Form2 resin printer is a great machine for a plethora of applications. Unfortunately, this was not one of those applications. Its build plate is not large enough for the bearing to possibly print flat, and this causes structural weak points along the bearing. The orientation of the model is shown in fig. 12 and is cut at roughly half the print completed.

Because of this, there was a significant number of failures in the printing where the part had some cracks left behind from layers that did not adhere as cleanly as they should have. After many prints, a few came out well enough to create a working bearing for the design. However, this is where the problem of being brittle comes into play. As the strain of the bearing on the inner walls goes up from the metal ball bearings within, tiny cracks begin to form as the bearing takes that stress. This results in binding issues that cannot be avoided and would be

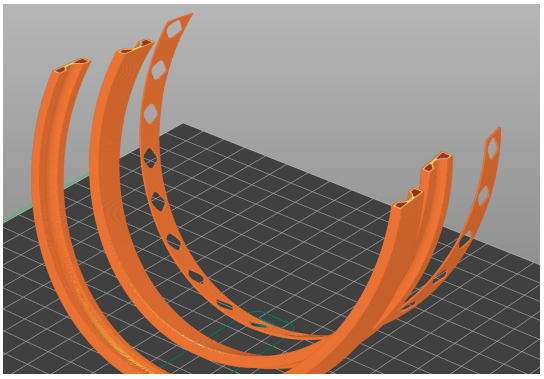

Fig. 12: Print Orientation

solved by regular maintenance in the form of a replacement bearing. The time between the start of a good bearing and the need for a replacement was too short for this to be a good solution.

After many iterations of this, Miguel decided to go back to basics and printed something out of a different material. This time something that would be able to provide some give instead so that the internal stresses could be mitigated - removing the need for replacement bearings. After searching for a material with the proper hardness, Miguel settled on thermoplastic polyurethane (TPU); A successor to thermoplastic Elastomer (TPE) materials that came before it. This turned out to be the best choice in the printing process as the first bearing Miguel printed after dialing in the new settings for the material was the exact one used for testing and, ultimately, the demo. The problem of micro-cracks in operation was no longer an issue as the TPU allowed for some flex in the movement, and the material naturally has some lubrication from the polyurethane. STP automotive grease was used to ensure smooth movement in the bearing, as it is the best lubricant Miguel has ever used for these applications.

3) Gears: The gears in the design are herringbone gears, also known as dual helical gears. These limit the possibility of slop causing any shifting in the gear position during operation and have a much lower probability of slippage during use. In a typical setting using aluminum or stainless steel for the gears, these would be considered an added cost on top of a single helical design. This is because of the tooling necessary and the extra work required to fabricate them. Since the designs of the project are 3D printed there was no worry about subtractive manufacturing techniques. Using additive processes instead, these can print without issue in varying orientations. The first designs were printed in resin to get the best possible resolution on the parts, as limiting any shakiness caused by small defects in the print was necessary. However, the problem of brittle prints continues to creep in again.

While very smooth and nice at the start, over time, there was a buildup of material from the micro-cracks that would end up causing some of the teeth on the gears to break off. Instead of using the resin-printed parts, after the successes in the bearing, Miguel printed the gears we used in the same TPU material, and they had no problems during operation. The best orientation for printing these was planar, with the normal of

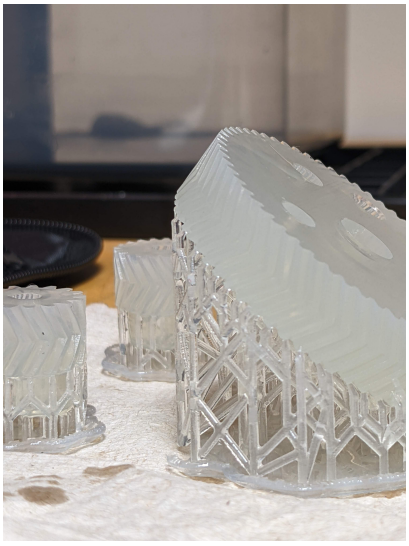

Fig. 13: Print Orientation

the print being equal to the axis of rotation shown in fig. 13 in the small gears on the left and back, and simplified the printing process by removing the need for any support materials during the process.

4) Assembly: Many of these parts are held together using a combination of screws and nuts with sizes varying from M2-M4 and some heat-set threaded inserts. These inserts allow for parts to be screwed together to have threads in place to prevent the parts from losing their bite over time. This is a common theme with 3D-printed parts that can be detrimental to a project, and adding these inserts removes that problem entirely. Aside from mounting hardware to connect parts, there was also the use of what are referred to as compliant mechanisms. This means a part that has some flex or give to it that would lock in place without the need for external hardware.

K. Testing and Development

Pieces of the project were created incrementally, ensuring testing along the way. Testing individual pieces thoroughly lessened the overall work when integrating parts.

1) StargazerServer-AlpacaClient: The creation of the app and the StargazerServer was done mostly in parallel, as the communication between the two was established. Testing of the AlpacaClient was simpler, as ASCOM provides a simulator for a telescope and camera object. This was used in initial testing to ensure the interface was behaving correctly.

2) AlpacaServer: The AlpacaServer was flexible in testing and development. Since Alpaca requests are simply HTTP with specific body data, we were able to use Postman very frequently when trying to test certain endpoints. Integration with the Raspberry Pi was slightly more tricky, however, when the UART queue was created and working properly, was simplified greatly.

3) Pico-Firmware: Much thought and design went into making the Pico work as desired. The math required to calculate current right ascension and declination took time to understand and implement correctly. When fully understood,

however, it was simplified to be very manageable. After many iterations, the calculations and motor movements were proven to be extremely accurate to where stars were actually located.

4) Camera Controller: The camera controller was initially custom software written in C utilizing the gphoto2 library. This effort was abandoned after several months due primarily to a lack of support and documentation. The command line interface, though less performant, was found to be stable and reliable for all the purposes of the project.

5) Image processor: The Image processor went through many iterations. Initially, an attempt was made to do all the work in C++; however, C++ lacked a simple way to communicate with the calling Alpaca interface. An additional attempt was made at writing a CLI for the image processor in the same way that the camera is controlled using a CLI build of gphoto2. Ultimately it was determined that there existed too many challenges with writing a robust CLI capable of being adaptable to the situations required and that a large amount of data could cause a buffer overflow with the standard in and output buffers of a typical terminal emulator. Ultimately it was determined that the most performant way was to embed the image processor into the listener program using a custom C-API callable by the Go-C compatibility layer.

L. Components Used

A final bill of materials with an itemized view is included with the documentation. List below serves as an overview.

1) Motors to control the assembly: Stepper motors salvaged from an old 3D printer with a bad main board were used. These have low power requirements and a small footprint compared to some other stepper motor designs available on the market.
2) 3D printed parts: Many parts were 3D printed, including much of the platform, the camera mount, mounting plates for PCBs, and casings for the necessary components. A full case for all parts was planned but having everything sectioned off in parts allowed for the whole system to be displayed easily at the demo.
3) Power supply: the motors and other electrical components required an external power source. The specifics of this supply were considered when designing the battery pack for the project. This was completed as part of the build using the custom $34V$ battery pack with charge/discharge protections. In addition to the pack, the necessary voltages split to the various components were completed with various switching regulators to supply their requirements.
4) Swarm dev kit: We appreciate Swarm for being very generous and donating the Swarm dev kit for the project. It was used to get necessary GPS and GMT data.

M. Efforts and Individual Responsibilities

Below is an overview of the work that was accomplished and how the work was divided.

- Rich Baird: Primarily responsible for interfacing the Camera with the rest of the system and implementing the image processing features. Rich designed the communication protocol between these and the Alpaca interface and served as the SME for the photography and camera-related elements of the project. He also designed the 9-axis IMU interface and assisted with some of the wiring and electrical assembly.
- Miguel Gomez: Responsible for the hardware design choices and implementation/assembly of the electrical and physical systems. Built the battery used to power the system as a whole and layout for the power distribution across all components in the design. In addition to the electrical design. Miguel implemented the interface for communicating with the motor drivers and was responsible for the fabrication and iterations necessary to implement the physical aspects of the assembly, including the mounting, interfacing, and integration for the mechanical portions of the project.
- Tyler Liddell: Along with Hyrum, responsible for almost all of the software, excluding the camera controller and image processing. This included the StargazerApp, StargazerServer-AlpacaClient, AlpacaServer, and MicroPython written for the Raspberry Pi Pico.
- Hyrum Saunders: Along with Tyler, responsible for almost all of the software, excluding the camera controller and image processing. This included mobile phone, windows, and web apps, software that acted as both the server to those apps and the client to the Alpaca interface, software acting as the Alpaca server in the interface, and embedded software controlling the mount.

V. Future Work

The final project was completely functioning end-to-end, so the baseline goals were met. In addition, the image processor stretch goal was accomplished. While the image processor is fully functional, it could benefit from further experimentation. This could include different filters on the camera, testing at various locations, and adjustments in settings, like what color to use when colorizing the image during the processing itself. This is planned work for warmer weather when it's easier to take the project out and test it.

Along with the planned refinements of the system, work could be done to expand the project in entirely new ways. One exciting idea would be to allow public control of the system through an internet interface or portal. One could create a "time-sharing" system where users add requests to the queue, and the image would automatically be taken, processed, and returned to the user. This would allow any person with an internet connection to experience the art of astrophotography.

A completely custom design for the assembly would be needed as well since every portion of the printing process should be optimized. One that adheres to a set size and has a specific camera footprint in mind would best suit the needs of the project and could easily have swappable mounts for other cameras. The addition of two more motors, one for a panning system to align to true north, and another to level out the camera while aligning to the equatorial reference frame would remove the need to manually adjust the mount prior to

use. In this setting, a user could simply place the system on the ground where it could level and align itself.

VI. Conclusion

This project set out to automate astrophotography. It does so, and at a much lower price than other all-in-one astrophotography systems. Additionally, being Alpaca compliant allows it to be swapped out with existing mounts and controllers, further decreasing costs for individuals who may already own some astrophotography equipment.

An app on a phone or computer allows the selection of stars, and the mount will then slew to those stars, take pictures, process them and return them to the app. The mount has proved to be very accurate in finding different stars, provided it was polar aligned correctly on startup.

The image processing is also fully functional, though further experimentation with different locations, camera filters, and processing settings is needed to produce the highest-quality images. As mentioned in the Future Work section, this is planned for the Spring and warmer weather.

Overall, the project was a success. Everything that it set out to do was accomplished. After final adjustments, the project will be ready for the original goal of releasing it as an open-source project, making astrophotography easier and more available than ever.

Appendix A
Larger Figures and Diagrams

Fig. 14: Image of the completed setup

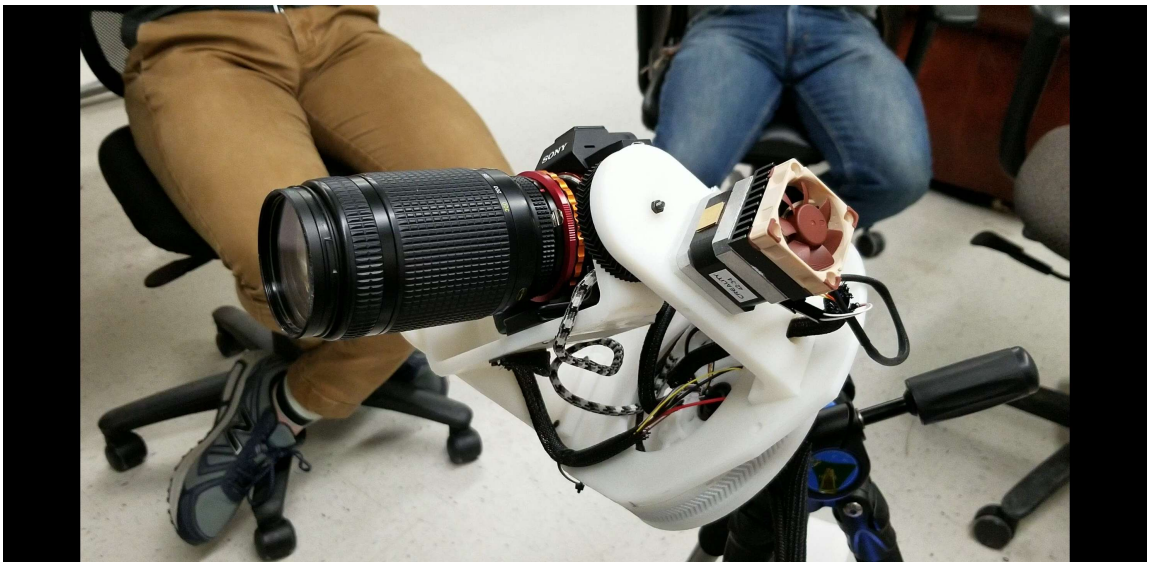

Fig. 15: Closer image of the mount setup

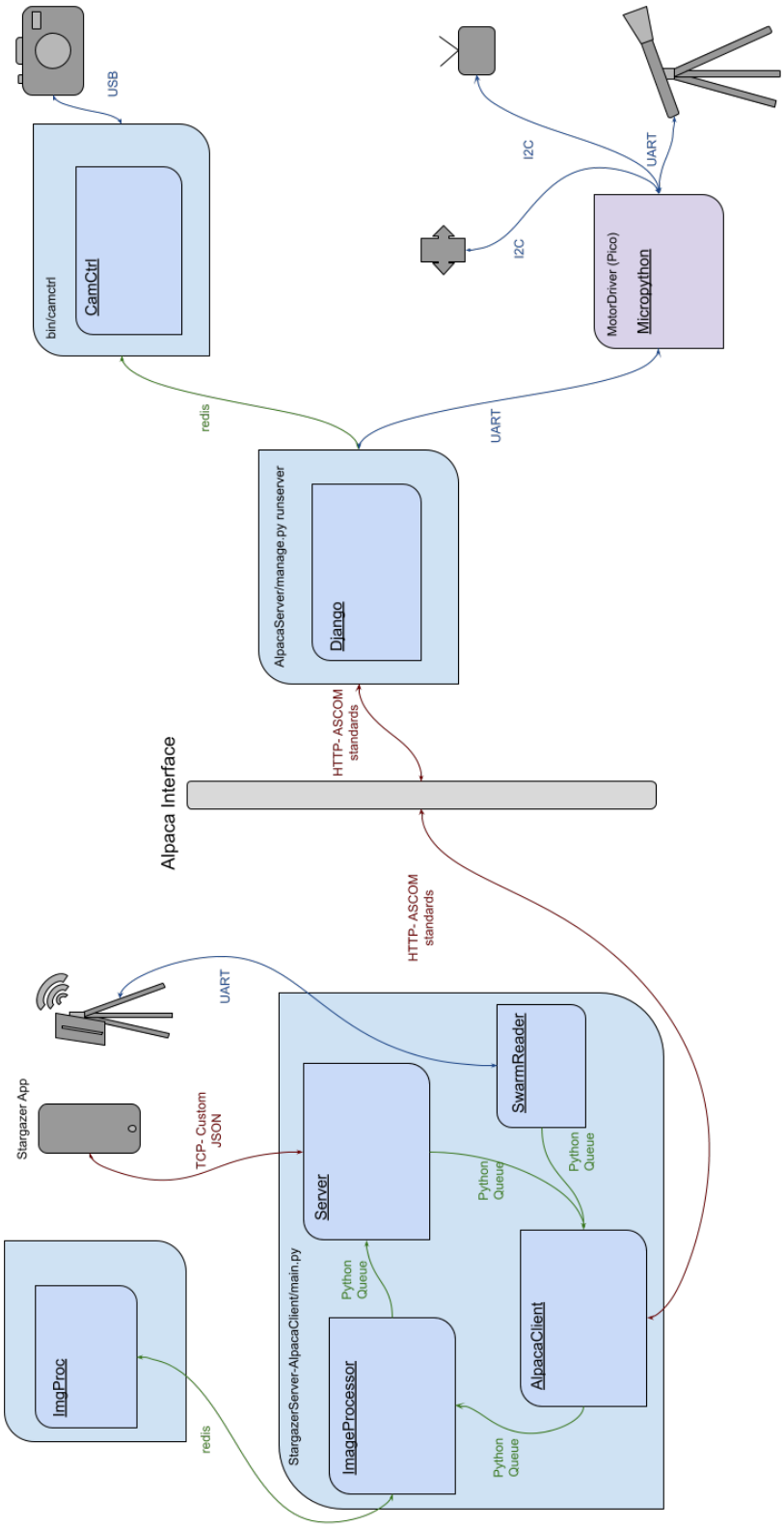

Fig. 16: Overview of Entire System

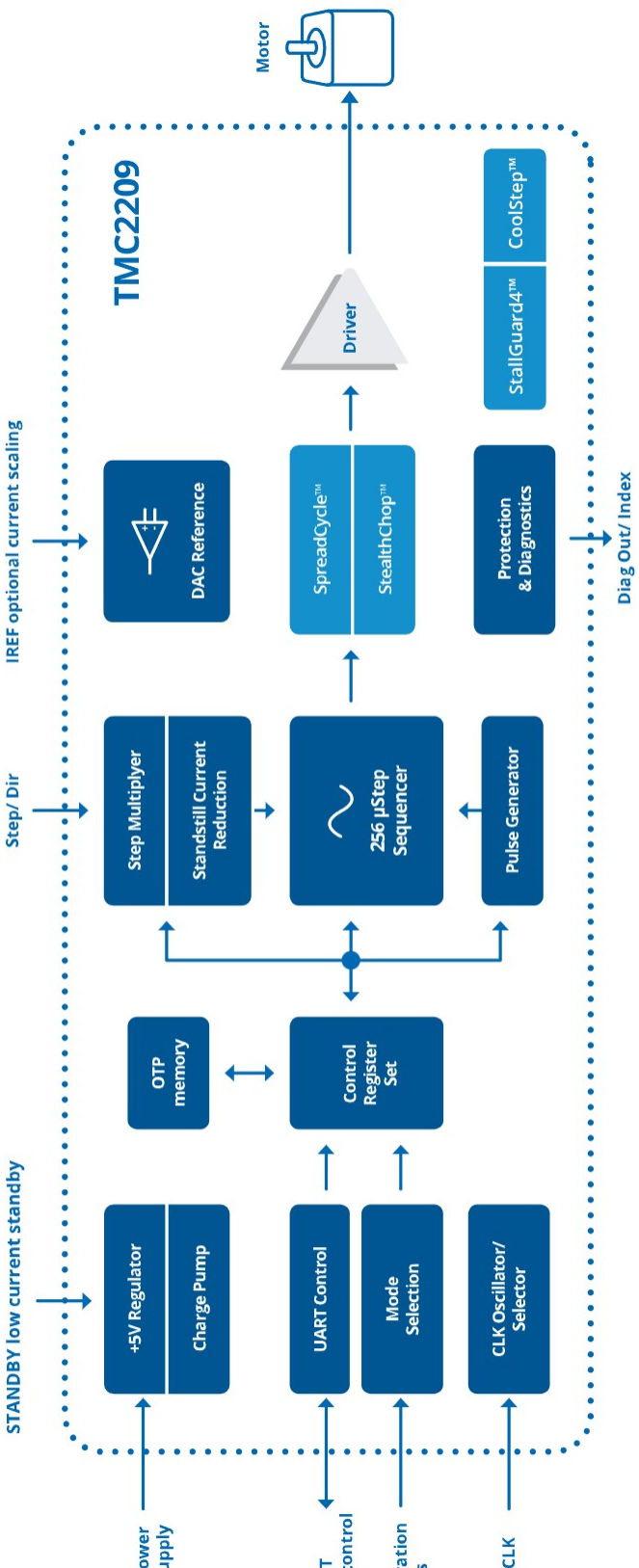

Fig. 17: Block Diagram of TMC 2209 Driver [12]

REFERENCES

[1] J. Menendez, "Is Astrophotography Expensive?" [Online]. Available: https://midnightphotographer.com/is-astrophotography-expensive/

[2] "Meet STELLINA, Our Observation Station — Vaonis." [Online]. Available: https://vaonis.com/stellina

[3] A. Woodward, "SpaceX's Starlink Satellites Are Photobombing Astronomy Images, Study Says." [Online]. Available: https://www.wsj.com/articles/spacexs-starlink-satellites-are-photobombing-astronomy-images-study-says-11644062404

[4] J. Huggins, "A Graphic representation of Polar Alignment and the Celestial Sphere," Apr 2000. [Online]. Available: http://www.astronomy.net/articles/4/polaralign.html

[5] 0x010C, "File:2018-03 Gorges de la Loire Nature Reserve star trails.jpg," Mar 2018. [Online]. Available: https://commons.wikimedia.org/wiki/File:2018-03_Gorges_de_la_Loire_Nature_Reserve_star_trails.jpg

[6] R. Suszynski, "Convolution Method For CCD Images Processing For Auto-guiding Astrophotography System," 2008 International Conference on Computer Engineering Systems, Nov 2008. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.utah.edu/document/4772970

[7] ——, "Digital Processing Of CCD images For Auto-guiding Astrophotography System," 2008 9th International Conference on Signal Processing, Oct 2008. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.utah.edu/document/4697272

[8] H. Zhou and Y. Yu, "Planetary Image Live Stacking Via Phase Correlation," 2016 9th International Symposium on Computational Intelligence and Design (ISCID), Dec 2016. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.utah.edu/document/7830782

[9] R. Suszynski, "Stand-alone Station For Deep Space Objects Astrophotography," 2009 52nd IEEE International Midwest Symposium on Circuits and Systems, Aug 2009. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.utah.edu/document/5236086/figuresfigures

[10] G. D. Evangelidis and E. Z. Psarakis, "Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 10, pp. 1858–1865, 2008.

[11] M. Stefik, "What Do The Abbreviations R.A. and Dec. Mean, and How Do You Use Them To Find Objects In The Sky," Astronomy Magazine, Jul 2013. [Online]. Available: https://astronomy.com/magazine/ask-astro/2013/07/sky-coordinates

[12] tmc, "TMC2209-LA." [Online]. Available: https://www.trinamic.com/products/integrated-circuits/details/tmc2209-la/

[13] creality, "Accessories Spare Parts." [Online]. Available: https://creality3d.shop/collections/accessories

[14] A. Industries, "Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055." [Online]. Available: https://www.adafruit.com/product/2472

[15] J. Zhou, T. Wu, X. Zhou, and J. Xi, "Advanced Cathodic Freestanding Interlayers For Lithium–sulfur Batteries: Understanding, Fabrication, and Modification," Jul 2022. [Online]. Available: https://pubs.rsc.org/en/content/articlelanding/2022/CP/D2CP02097A

[16] lygte, "Test Of Samsung INR18650-30Q." [Online]. Available: https://lygte-info.dk/review/batteries2012/Samsung%20INR18650-30Q%203000mAh%20(Pink)%20UK.html