Emergent Space Technologies Internship 2021

Rex E. McAllister

South River High School

Abstract

This report outlines the experience of a summer intern at Emergent Space Technologies (EST) located in Laurel, MD. EST is a small business that develops systems for civil, military, and commercial space missions. I researched spacecraft trajectory modeling and worked with an aerospace engineer at EST to create a device that calculates satellite orbits and physically points to satellites as they pass overhead. To complete this project, I learned and developed skills from many disciplines, including mathematics to predict satellite positions, programming to implement math and interface with motors and sensors, computer-aided design (CAD) for 3D printing, and electrical engineering to design circuits and connect sensors and motors to the main computer.

Introduction

Emergent Space Technologies is a leader in designing flight software for multi-spacecraft missions, with expertise in systems engineering, orbital mechanics, modeling and simulation, integration and testing, and guidance, navigation, and control. Headquartered in Laurel, MD, they work with customers such as Lockheed Martin Space Systems and multiple NASA departments. They also have employees nationwide at customer sites in Maryland, Virginia, New Mexico, Colorado, and California. Their expert knowledge of software and 19 years of mission experience has allowed them to excel in projects such as NASA GSFC's Formation Flying Test Bed (2004-present) and Magnetospheric Multi-Scale mission (MMS) (2012-2017); NASA ARC's Starling mission (2017-present); and DARPA/TTO's System F6 (2011-2014) and Blackjack (2019-present) programs.

Overview of my Internship Experience

To bridge the summer between junior and senior year, all South River High School STEM students are required to complete a 135-hour internship experience to give them experience in a potential career field. I applied to the Emergent Space Technologies Spacecraft Trajectory Modeling internship because I have an interest in pursuing aerospace engineering in the future and was excited for the opportunity to explore it during high school. This internship was a self-paced, at-home project which required regular meetings with my mentor at EST, Mr. Bill Bamford.

Project Selection:

This internship was designed as a learning experience and did not have a set task, meaning I was given the opportunity to select my own project. I decided to build a device that calculates satellite positions and points to them as they cross the sky. This project incorporates many different areas that I want to work with - math, programming, electrical engineering, design, and fabrication. Rather than just simulating satellite orbits on a computer, this project allowed me to create a tangible satellite pointer to bring the math into the "real world" to track satellites, as well as expanding the scope of the project to use many more engineering disciplines.

Learning the Math:

After deciding on a project, my mentor spent two weeks teaching me the math behind satellite orbit calculation. Data for the majority of earth satellites is openly available from sites like CelesTrak and n2yo and is encoded as two-line elements (TLEs). TLEs contain information such as the satellite catalog number and classification, and more importantly, four of the six Keplerian elements (see Fig. 1) that define an orbit in an earth-centric, non-rotating reference frame (inclination, eccentricity, right ascension of the ascending node, argument of perigee). The other two elements (true anomaly and semimajor axis) can be calculated using the initial Keplerian elements alongside the TLE mean anomaly and mean motion. An interesting algorithm is used during this process to find the eccentric anomaly (see Fig. 2), which is a number needed for calculating the true anomaly. Instead of using a direct algebraic equation, an application of the Newton-Rapson method is used. This method feeds the output of an equation back into itself until the previous and current results converge, generally requiring less than 10 iterations for this application.

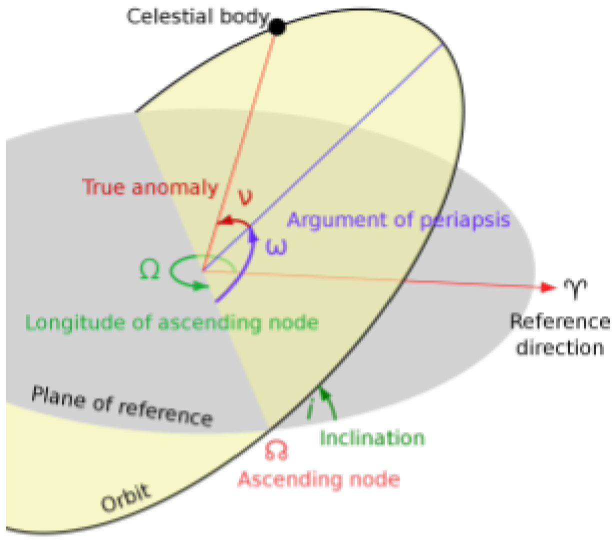


Fig. 1: Orbital Elements - Lasunncty at the English Wikipedia, CC BY-SA 3.0

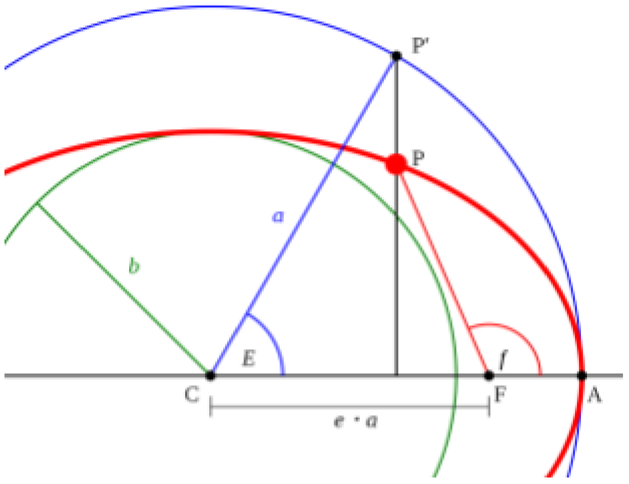


Fig. 2: The Eccentric Anomaly of point P is angle E - Wikipedia

The next step after defining the orbit is to translate those numbers into 2D cartesian coordinates within the orbital plane. This is done by first calculating the straight-line distance from the satellite to the center of the earth, then doing a simple transformation from polar to rectangular coordinates using the true anomaly angle. The distance to the central body [$r_c(t)$] is multiplied by the cosine of true anomaly [$v(t)$] to get the orbital plane "x" coordinate [$o_x(t)$]: $o_x(t) = r_c(t) * \cos(v(t))$. The "y" coordinate is then calculated using: $o_y(t) = r_c(t) * \sin(v(t))$. After working through a practice problem manually to make sure I understood everything correctly, I wrote test scripts in Python to try to get the same results and to graph the data over time. Figures 3 and 4 show the outputs of my test programs for true anomaly and orbital plane cartesian coordinates over time.

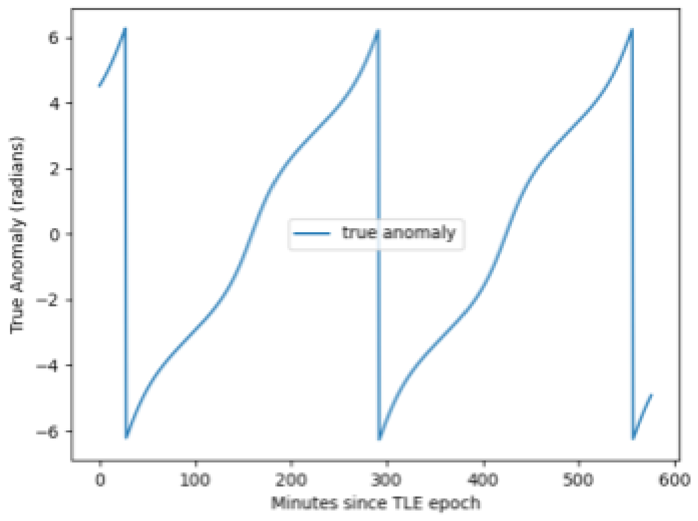


Fig. 3: Plot of a satellite's calculated true anomaly over time

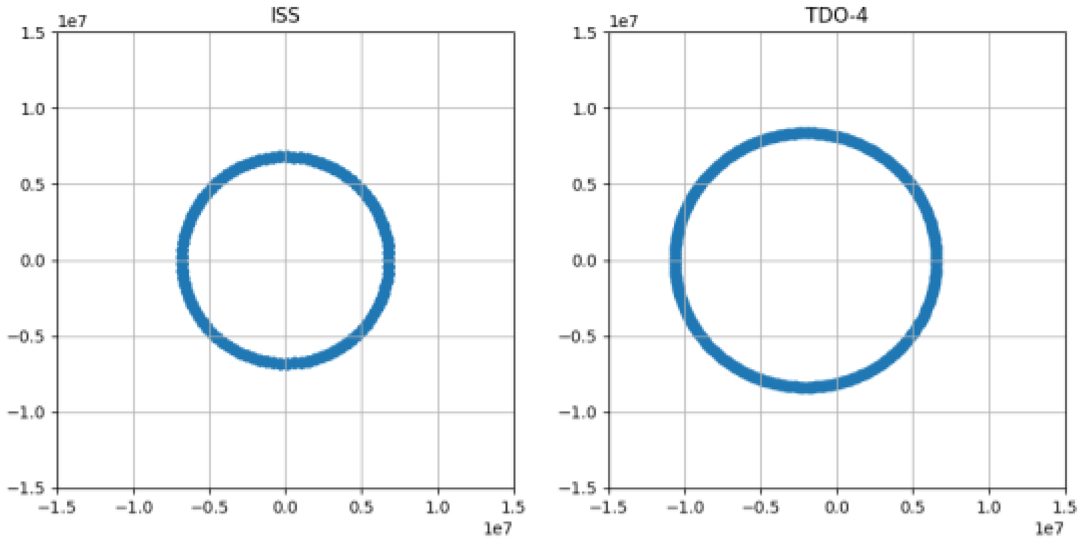


Fig. 4: Graphed orbits of two satellites, in meters from the center of Earth.

Next up came my first introduction to vectors and matrix rotation. A combination of the R_x and R_z rotation matrices is used to move from the 2D orbital plane to a 3D vector in the inertial frame, which is a non-rotating frame centered around the earth. The rotation required the multiplication of two R_z matrices, an R_x matrix, and the orbital plane coordinates. As I do not have a background in linear algebra, my mentor pointed me to a reference sheet that gave me the final equation. I simply had to plug the orbital x and y coordinates, the argument of periapsis, the longitude of the ascending node, and the inclination to get the inertial x, y, and z coordinates. Plotting these coordinates over time gave an interesting 3D graph of the satellite position over time, as seen in Figure 5.

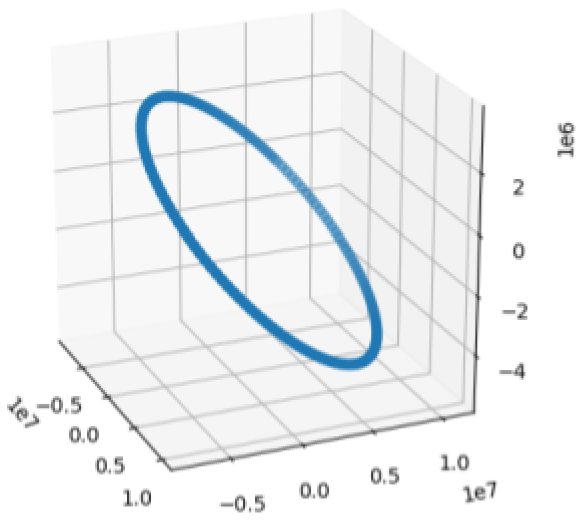


Fig. 5: Graph of satellite position in the non-rotating inertial frame.

To get the final altitude and azimuth of the satellite, I needed to first get the observer and satellite position into the same coordinate system. For this application, the Earth-Centered, Earth-Fixed (ECEF) frame is used because it is the simplest transformation for both the satellite and the observer position. This frame is a rotating coordinate system that is fixed to the direction of the prime meridian. The observer ECEF coordinates came from plugging in latitude and longitude into a fairly straightforward set of equations that my mentor gave me. The satellite

ECEF coordinates were slightly more complicated to calculate as they required computing the exact Julian date, using that to calculate the Greenwich Mean Sidereal Time (the current angle of the earth from the inertial frame reference heading), and finally plugging numbers into and taking the dot product of two matrices. Next, the line of sight vector between the satellite and observer is calculated by subtracting the observer ECEF position from the satellite position. The final step is to take the dot product of the line of sight vector and a 3x3 matrix of latitude, longitude, sines, and cosines. After performing one final equation you finally receive the predicted altitude and azimuth of the satellite.

First Major Setback:

After finishing the code for predicting the altitude and azimuth of a satellite, I was dismayed to find that the numbers didn't match the ones I was expecting from an astronomical calculations program. I spent the following week looking through each piece of code I had written and verified numbers against what my mentor expected at every point in the orbital calculations. Although this process did catch a few errors in my code, we were not able to get an accurate altitude and azimuth from the current set of algorithms. Our current assumption is that there simply is not enough precision in some of the equations we are using, or that there is still a small, yet-to-be-discovered error in the code that prevents it from working correctly.

In the interest of keeping the project moving forward, I decided to get help from external resources. I found a Python library called PyEphem which uses the "libastro" C library to compute the positions of objects in the sky. This code uses the same types of mathematical transformations I was performing, but it is provided in a robust, community-maintained package that is relatively simple to use. Even though it is disappointing to not do all of the calculations using code that I wrote, I still learned all the principles behind orbit calculations and using an astronomical calculations library let me move into working on the hardware for the project.

Bringing the Project to Life:

Now that I was able to get my program to output the correct positions of satellites, I switched my focus to building the electronics that would point to the satellites as they went overhead. I had some pan/tilt camera mechanisms salvaged from old security camera units which would give me a foundation for the motion portion of the tracking mechanism. I also prepared the other components of the project, some of which I ordered and others I already had from various other projects. I already had components like a Raspberry Pi and stepper motor controllers, but I needed to purchase items like a 12v battery for the main power source, a step-down power regulator for the Pi, and a project case to house all the electronics.

I started out by setting up a test circuit to run the stepper motors on the pan/tilt mechanism. I was able to find the datasheets for both the motors and motor controllers online which helped me to set up my test circuit correctly. I was able to use an Arduino to give the correct number of step pulses to move the motors a precise number of degrees. The motors for this project are eight-wire stepper motors, meaning I could wire them up in bipolar series, bipolar parallel, or unipolar. Each option changes the current-carrying ability of the motor, and I opted for the bipolar series which reduces the amount of current needed to drive the motors. The control of the motors is an open-loop system where each pulse to the motor controller results in a 1.8-degree full step on the motor, or fractions of 1.8 when the stepper motor controller is in micro-stepping mode. The final rotational output is geared down in a 2:1 ratio for the elevation and a 5:1 ratio for the azimuth.

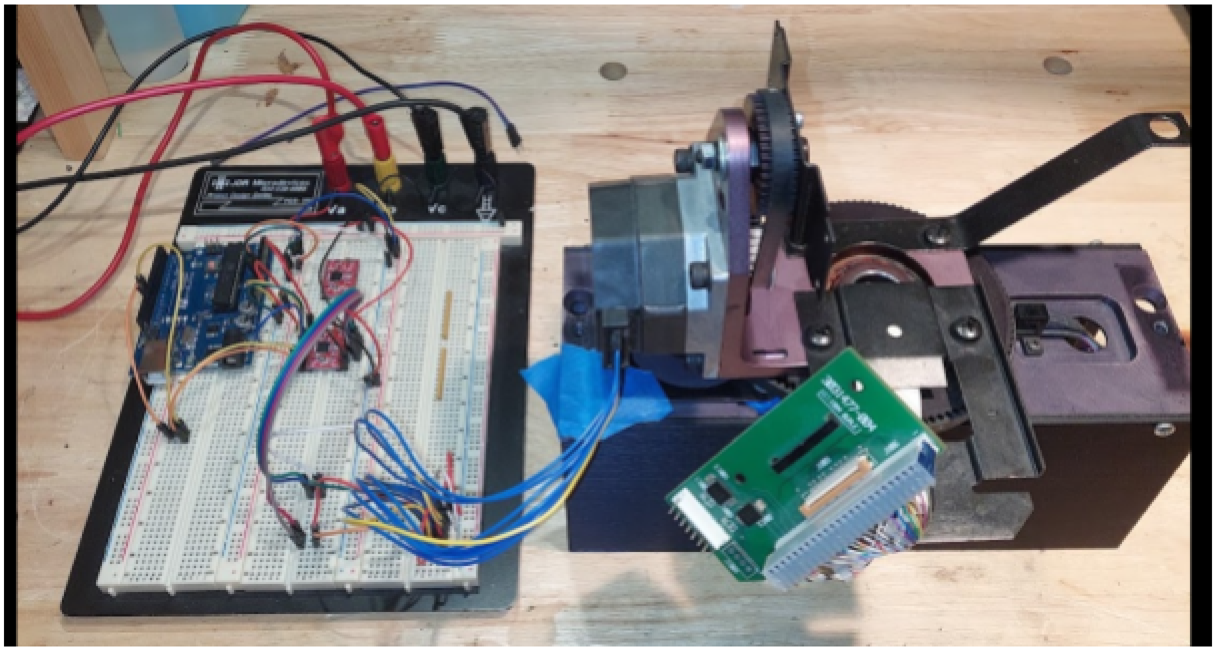


Fig. 6: A simple circuit to test the elevation stepper motor

After testing the motors, I moved from writing C++ for the Arduino to writing code in Python for the Raspberry Pi. The Raspberry Pi single-board computer (SBC) has a lot of benefits over a microcontroller, including higher clock speeds, an operating system to run a graphical user interface (GUI), onboard WiFi capabilities, and enough GPIO to interface with motor controllers, limit switches, the GPS module, and the compass. I set up the Pi for a quick test run of the motors before I needed to make the first of two protoboards for this project.

To keep wires from tangling while the azimuth motor is spinning, a 24-pin slip ring runs up the center of the pan/tilt mechanism. To run the upper motor off of these pins, I made a protoboard that plugs into the slip ring breakout pins and houses the elevation stepper motor controller, the stepper motor's connector, and a hookup for the elevation optical limit switch. To connect all of the pins I used soldering to tack components in place and wire-wrapping to run signals throughout the board. I was introduced to wire-wrapping by an experienced computer/electrical engineer who uses it to mock up test circuits. It involves using a special tool that accepts a piece of 30 AWG wire, and as you rotate the tool like a screwdriver it securely wraps the ends of the wire around the circuit board pins on the underside of the protoboard.

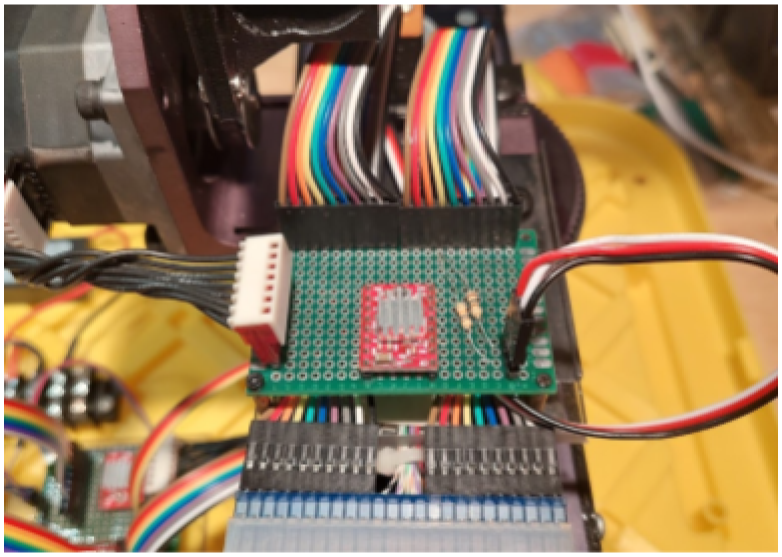


Fig. 7: Elevation stepper motor controller and slip ring breakout board

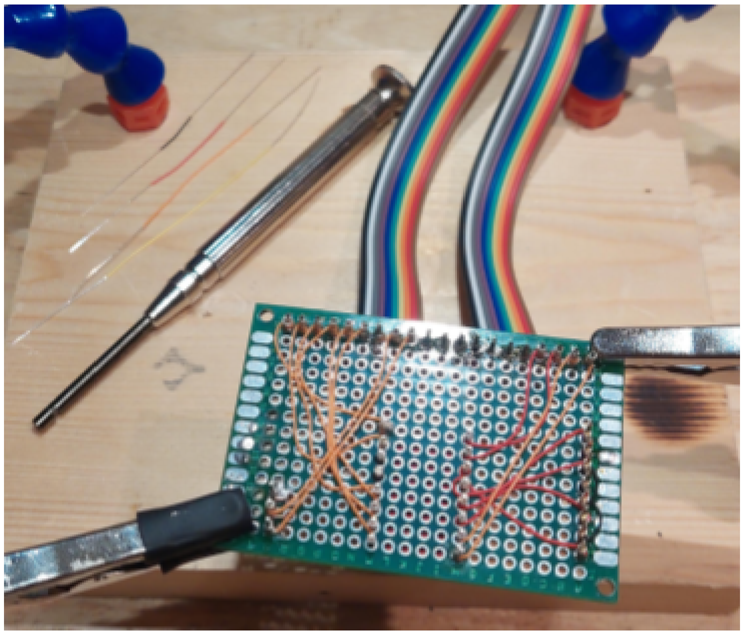


Fig. 8: Underside of the upper slip ring breakout board with a wire-wrapping tool and an assortment of wires in the background

After setting up the slip ring breakout board, I was able to combine the satellite calculations code with the motor control code. I wrote a simple program that used hard-coded satellite TLE data for the calculations and then drove the motors to track the satellite at the start of program execution. The program needed to assume that the motors started out level and pointing north, as I hadn't added the limit sensors or compass yet. Even though it was only a simple test, it was incredibly exciting to watch my project jump to life and start tracking satellites across the sky for the first time.

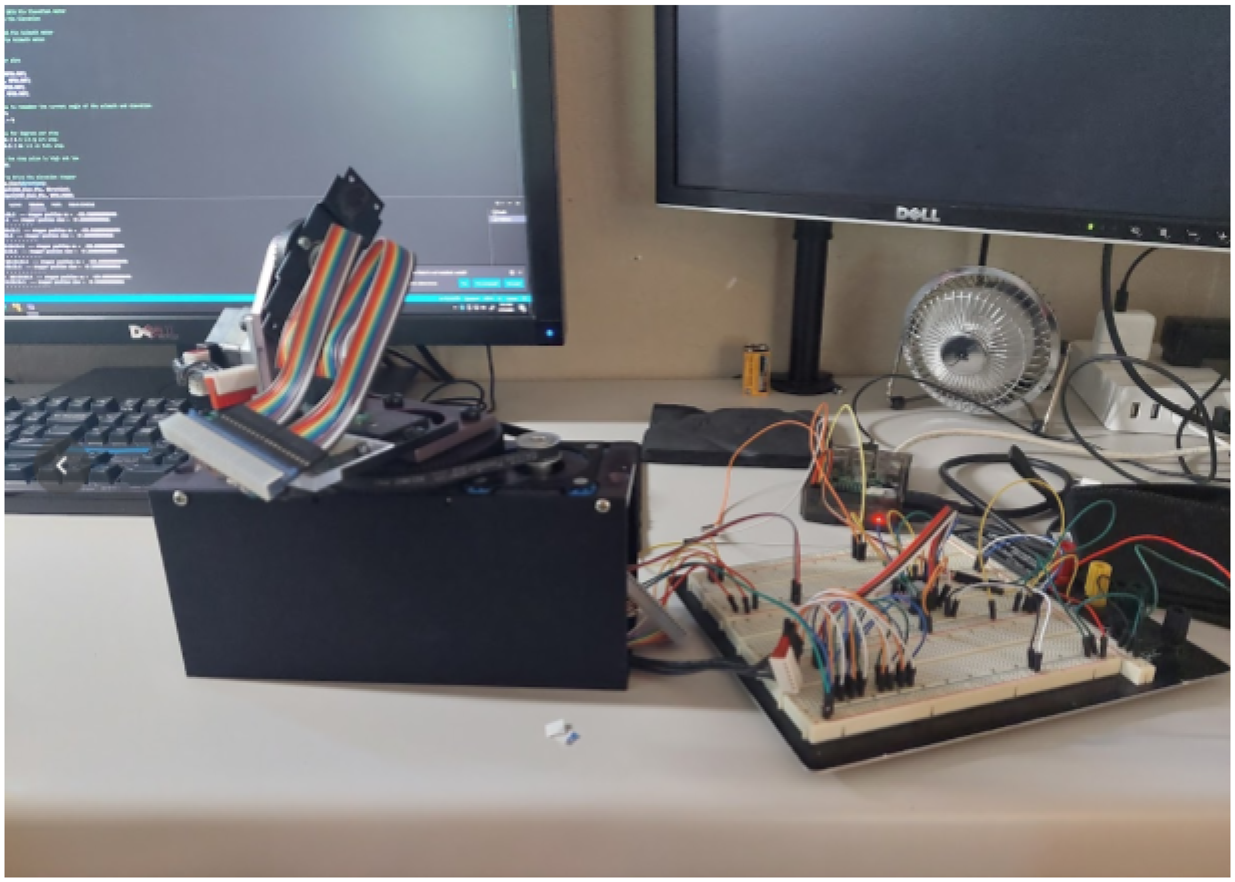


Fig. 9: My first desktop-based test with dual-axis tracking of satellites across the sky

Following the successful tracking test, I was able to move on to the final assembly of the project. At this point, the azimuth motor still relied on a breadboard, meaning I needed to make another protoboard (Fig. 9) to house the azimuth motor controller, lower slip ring breakout, and azimuth limit sensor hookup. This board mainly served as a breakout board to plug in peripherals and send one consolidated set of ribbon cables into the Raspberry Pi. After completing the new board, I was able to bolt most of the final hardware to the lid of my project case (Fig. 10). All of the electronics had convenient holes that accepted standoffs, but the pan/tilt mechanism required me to design and 3D print a mounting box (Fig. 11). I designed the part in Autodesk Inventor before running a 15-hour 3D print on my home printer. The box has a captive nut design, meaning the bolt can be directly threaded into the mounting points as if they were tapped. Finally, as I was mounting the final hardware, I added the GPS unit. This was a relatively simple addition as it mounts with two standoffs and then communicates with the Raspberry Pi over the Universal Asynchronous Reciever Transmitter (UART) protocol, which was easy to integrate into code on the Pi. I then strapped the 12v battery into the bottom of the project case, and the hardware was ready to go.

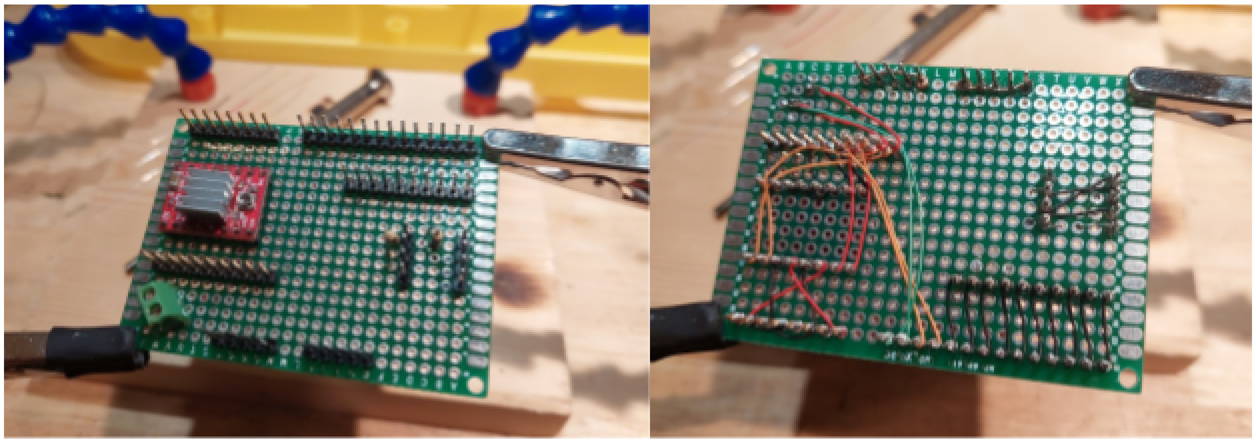


Fig. 9: Top and bottom of the main signal breakout board

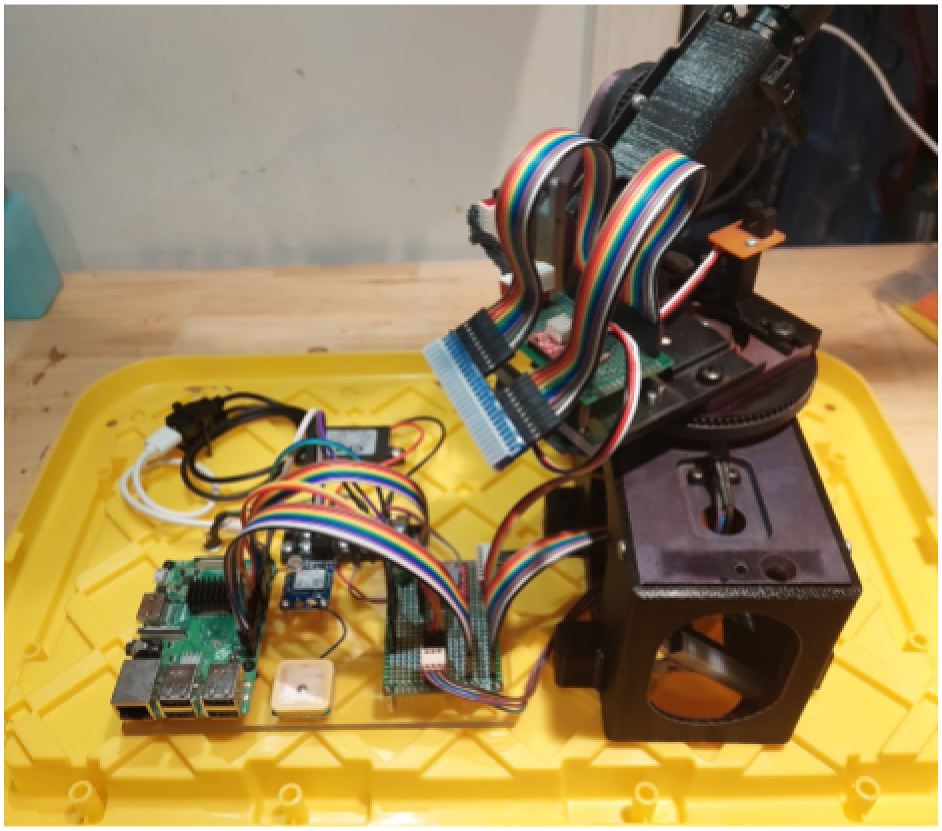


Fig. 10: All electronics attached to the lid of the project case

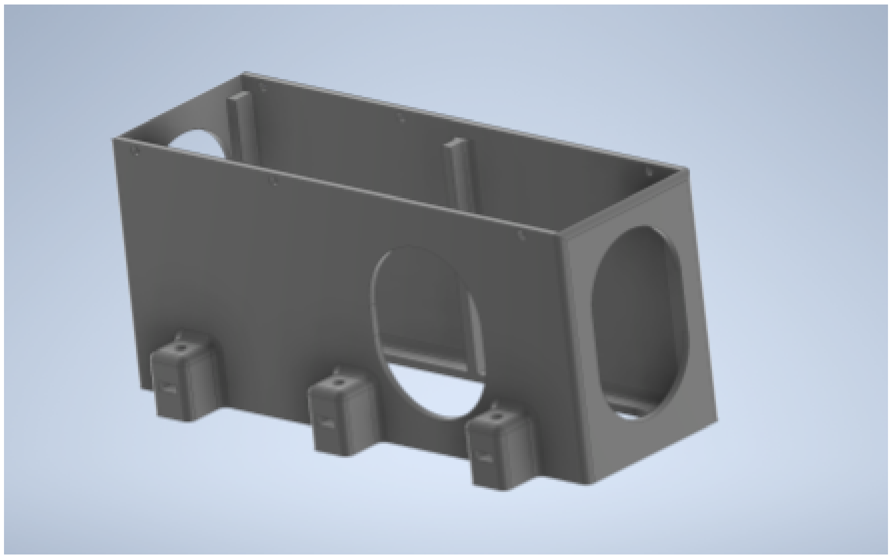
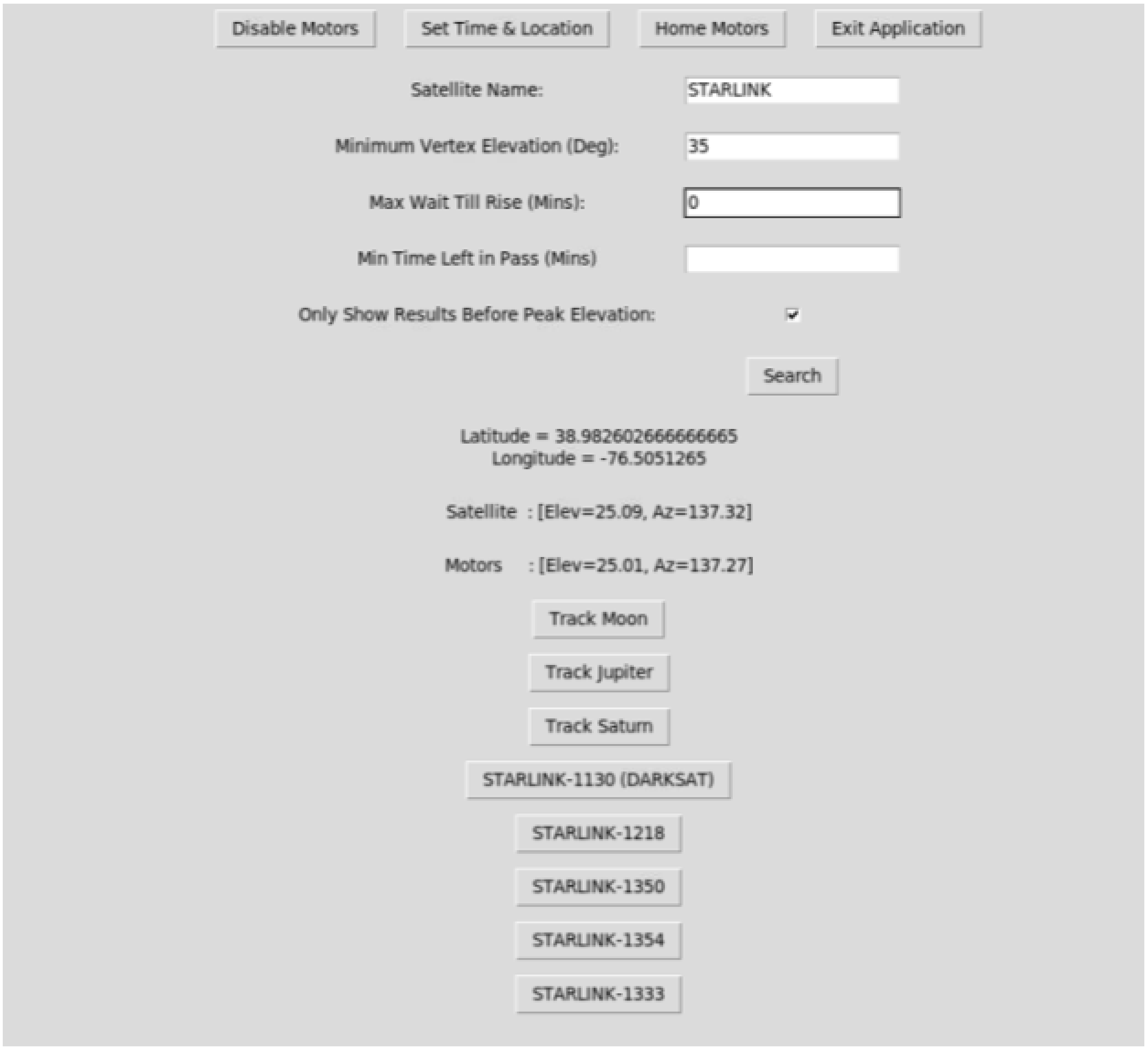


Fig. 11: CAD Model of the pan/tilt mounting box

The final piece of the project was to create a user interface. I set up the Pi to run its own WiFi access point. From here I could connect either a touchscreen tablet or a laptop to let the user interact with the program. I initially tried to create a locally hosted website for the user to access over the WiFi connection, but this proved to be quite difficult to create and I didn't like how far the web design took me away from the main focus of the project. I changed to using a Python module called Tkinter to create a GUI on the Pi which was then accessed using a remote desktop connection. This made it a lot easier to create the button layout and integrate button presses and search fields into the rest of my code for satellite tracking. The final app (Fig. 12) has three main sections: top control bar, center search section, and bottom results and tracking section. When the user first enters the app, they have to click a button to initialize the GPS position and set the local system time based on the GPS data. Next, the motors have to be enabled and set to their home position. This was an interesting part of the code to get working, as the Tkinter app runs in its own event loop until the program is stopped. This prevents me from running the motors with the precise timing needed for smooth rotation. My solution was to use multithreading to start a separate motors thread, which runs in parallel to the main app thread. To home the motors, the main app "tells" the motors thread to run the motors to the home position, and then the motors thread spins the motors until they reach their optical limit switches.



Satellite Name: STARLINK

Minimum Vertex Elevation (Deg): 35

Max Wait Till Rise (Mins): 0

Min Time Left in Pass (Mins)

Only Show Results Before Peak Elevation: ✓

```
Latitude = 38.982602666666665
Longitude = -76.5051265

Satellite  : [Elev=25.09, Az=137.32]

Motors     : [Elev=25.01, Az=137.27]
```

Fig. 12: Final version of the satellite tracker GUI

Once GPS and home positions are set, the user can search for satellites. There are multiple optional search criteria, including the name of the satellite and the maximum allowable wait time until the satellite is visible. The app then computes the orbits for all satellites in its TLE database until it reaches five results that match the search criteria. The user can then click on the name of any of those five results and the system slews to that satellite's position and starts tracking. While tracking, the program also updates the app with the current position of the motors and the current satellite position, GPS position, and system time on GPS initialization. Finally, I added three extra buttons to the app to let the user point at the moon, Jupiter, or Saturn. The PyEphem library had built-in options for calculating planet positions, and the three options I added are bright, easy to locate visually, and they stay in the sky for multiple hours which allows for easier demonstrations of the project. With the app done, I was able to power the Pi with the 12v battery, connect my laptop or tablet to the WiFi and remote desktop connections, and then watch the system track satellites outdoors for the first time.

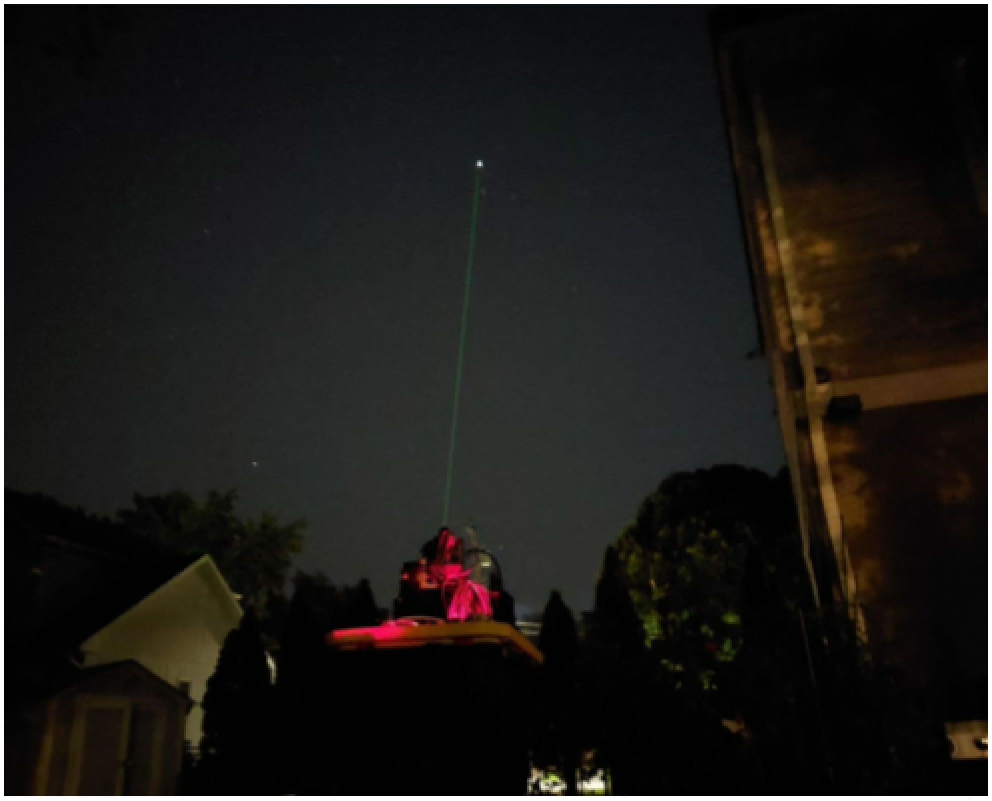


Fig. 13: Final test of the system, which is currently pointed at Jupiter

Key Takeaways

This internship proved to be a valuable experience, and the hours invested into it definitely paid off. One of my biggest takeaways from this opportunity is good time management. Being a virtual internship with no building to report to or any enforced time schedules, it was up to me to make sure I stayed on track throughout the whole project. Without the diligence to get up every day and give the project the attention it needed, I would have not come close to being able to finish with the success I had. Additionally, the patience to stick with the project even through tough parts was very important. There were a lot of times when I expected to complete a simple piece in a small amount of time, but it ended up taking far longer than I would have ever anticipated. For example, I expected to be able to assemble the protoboards in under an hour because they seemed simple, but they each took an entire afternoon to solder, wire, and test all the connections.

During the construction of the project, I also learned and reinforced skills that I can apply again in the future. This project was the first time I had ever used multithreading or created a GUI, both of which I will be applying to future projects to increase speed and allow for better user interaction. The math that I learned to define satellite orbits and rotate between different reference frames gave me a good starting point for understanding more advanced math classes that I will be taking in my senior year of high school and throughout college. Finally, when putting the hardware together I practiced my CAD skills again while designing some of the different pieces and I used Ohm's law to calculate the correct resistors needed to make voltage dividers and pull-ups for the optical limit switches.

In the end, I learned a lot of things from this project that are very difficult to learn from a school. Selecting a project and seeing it through to the end provided me with numerous opportunities and challenges that simply can't be taught in any other way. A school can prepare you with all the knowledge necessary to solve a problem, but finding your own unique ways to apply that knowledge reinforces concepts and ideas and gives you a new insight that can be reused later. I am very grateful for the opportunity to participate in this internship and I can't wait to see where this knowledge will take me in the future.