ECE 3332 – 002

Final Report of “Hans Off!” Project

Jason Bissias

Eric Cuomo

Edward Onasanya

Zach Gray

Texas Tech University

Electrical and Computer Engineering Department

April 2024

Abstract

This paper describes the “Hans Off!” project state to date. In this specific iteration, special emphasis is placed on the celestial body query system, with a detailed overview of the hardware mounting system, camera system, and power system.

Acknowledgements

My thanks as always to our Professor, Mr. Haustein, for his advice and patience. A special thank you also to my friend and roommate, Colin Tipton, who without, any of the web-development before, during, and after, this project would not be possible.

Table of Figures

1. Introduction

The “Hans Off!” project describes an automatic telescope, where the only user input needed to operate is the desired celestial body to be photographed and the time at which the photographing should occur. The automation necessary to make this possible requires that the device can autonomously determine its location and orientation, automatically query the necessary data, apply the appropriate mathematics for the time and location, and automatically focus and photograph the requested celestial body.

This paper, like the project, is split into two main sections: Software and Hardware. The Software mainly comprises the systems to calculate the location of the telescope, to transfer data between telescope and server, to calculate celestial body coordinates, to control the camera and movement, and the user interface. The scope of the Hardware is in the mount, the motors, the sensors, the camera, and the telescope itself.

2. Software

2.1 Celestial Body Querying System

The celestial body querying system is composed of two data sources, one for the distant stars and one for local planets and moons. The reason for this split is in the perceived movement of each body type. Whereas distant stars are relatively fixed in our sky (astral databases often update every 50 years) [**Error! Reference source not found.**], the local bodies are far more fluid in their perceived location. The necessary mathematics for locating and adjusting stars in a given geolocation and time are relatively simple, but local bodies need far more work.

The computer of the telescope is a Raspberry Pi 3 Model B, so all the software is written in Python, and the UI is scripted via the Jinja2 HTML extension for ease of interoperability. The entirety of this system's software is hosted on the web using a Hetzner VPS and delivered on the Web via Flask, uWSGI, and Nginx. This allows for an appealing, easy-to-create, and easy-to-use user interface while still maintaining usage of Python programming, which is important for working with the systems on the Raspberry, for familiarity in development, and for ease of modularity.

Some additional features of the system is that it has API integration with resources that can return magnetic declination (the difference between true and polar North) and time zone. This data is retrieved purely by location coordinates. This allows for a smoother user experience, less data needed from hardware, and more accurate calculations. The timezonefinder API is a simple-to-use software tool that, with just longitude and latitude, returns the querier's time zone based on a point-in-polygon check, as each time zone is stored as a polygon. [2] This tool (and therefore all polygon data) can be installed locally as a Python library, or one can simply call the API directly via a URL, akin to the Horizons System API. For the time being, the latter method is being used for simplicity,

however a proper install of the library may become necessary with time out of courtesy of the creator.

2.1.1 Hipparcos-2

The Hipparcos-2 database is a comprehensive catalogue of stars published in 2007, with a plethora of information on each one. The necessary information we need is Right Ascension, Declination. We can query that information using the TOPCAT querying tool. Of course, having every star in the database is unnecessary, so for optimization we can query for a specific number of stars under certain conditions. As an example, the initial plan of the project required a heavy optimization of the amount of data. Therefore, as can be seen in Figure 1, we queried for a certain number of stars at random within a specific place above the latitude at which we reside.

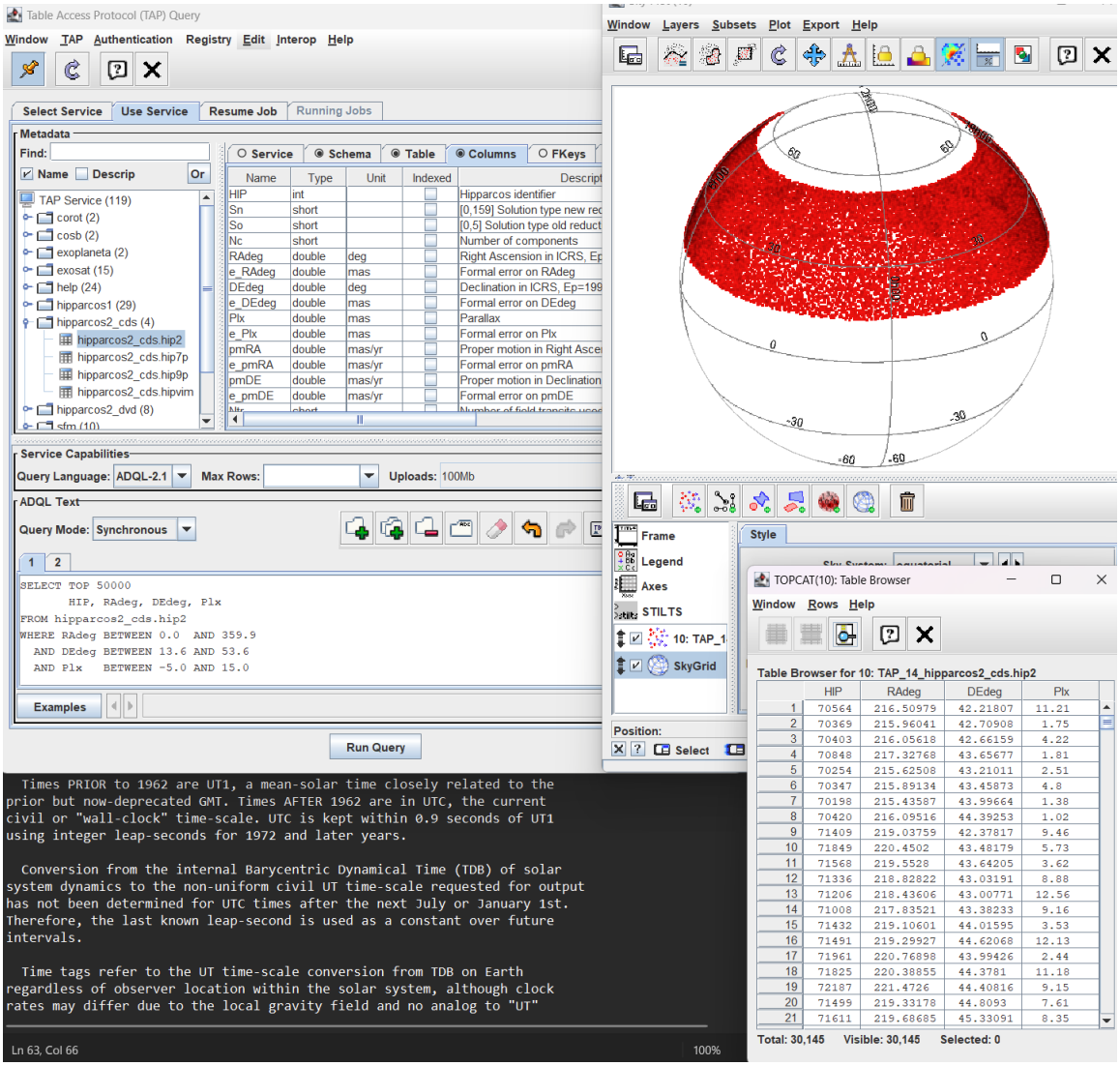

Figure 1: Visualization of Heavily Optimized Queried Data. The TOPCAT Tool with the Query can be Seen in the Top Left, the Visualization of the Queried Data in the Top Right, and the Table of the Queried Data in the Bottom Right

However, with the lack of such need for optimization given the current computing capabilities, we expand our query to far wider reaches so that the tool can be used by users in other locales. Another small addition to the query involves the 96 commonly named stars per the

ESA's Hipparcos Common Star Names Table [3]. These are added manually so that users may query for those stars by name as well as by HIP code.

2.1.2 Horizons System

The integration with the Horizons System API is accomplished via URL querying. By generating a URL with the viewing location, the desired celestial body, and the time and period of the viewing, we receive back a text file with all the necessary coordinate data. We can also format the text file to place the data in a CSV format.

In some respects, this data source is far more convenient that Hipparcos-2, as all of the desired coordinates are in the appropriate system. However, there are limitations in the querying capabilities, namely with time period. The Horizons System does not accept seconds or smaller as a valid unit for the time interval between each data point. This can be circumnavigated by requesting that a certain number of steps are taken within a certain timeframe. Thankfully, we already use this number to determine how to truncate the text file from the Horizons System, the only effort is in adjusting the URL for each situation. After we receive our data, the entire system is plug-and-play.

2.1.3 Calculating Coordinates

Since the Horizons System provides us with all we need, the only celestial bodies we need to apply any math to are the Hipparcos-2 stars. This is because our telescope is on an Altitude/Azimuth mount, whereas Hipparcos-2 provides star information in Right Ascension and Declination. The difference between the two systems is that Right Ascension and Declination are a 2-dimensional projection of the stars that is locked with the Earth's rotation. Meanwhile, Altitude and Azimuth are the directions an observer has to look towards in their specific time and place.

The conversion is relatively simple. To calculate the necessary coordinates, we first need to know our Local Sideral Time (LST). [**Error! Reference source not found.**] To calculate LST, we need to know the time in days since the J2000 timestamp (12:00 UT on the 1st of January, 2000 C.E.), our longitude, and what the current time is in UT. By using the following equation, the calculation becomes trivial:

$$LST = 100.46 + 0.985647 * daysSinceJ2000 + longitude + 15 * UT$$

From there, Hour Angle is simply LST minus Right Ascension, which is given to us by both data sources. If Hour Angle is negative, we add 360° to fit it within range. Once we have Hour Angle, we can now convert to Altitude and Azimuth. The process now just involves some simple trigonometry that can be seen in Figure 2 below. The main thing to note is that Azimuth is dependent on Altitude, so that is necessary first.

```python
def RADECtoALTAZ(dateStr, UT, coordLong, LAT, RA, DEC):
    #LST calculation:
    #need to convert from string to datetime, calculate since J2000, then from datetime to float (in seconds) to float (in days)
    dateNow = pd.to_datetime(dateStr, yearfirst=True, format="%Y-%b-%d %H:%M")
    J2000 = pd.to_datetime("2000-Jan-01 12:00", yearfirst=True, format="%Y-%b-%d %H:%M")
    sinceJ2000 = dateNow - J2000

    utHour = int(UT[12:14])
    utMins = int(UT[15:17])
    UTS = pd.to_datetime(UT, yearfirst=True, format="%Y-%b-%d %H:%M")
    UTM = pd.to_datetime(UTS - dt.timedelta(hours=utHour, minutes=utMins), yearfirst=True, format="%Y-%b-%d %H:%M")
    UTS = UTS - UTM
    sinceJ2000N = sinceJ2000.total_seconds()
    UTN = UTS.total_seconds()
    UTHd = UTN / (60 * 60)
    sinceJ2000d = sinceJ2000N / (24 * 60 * 60)

    LST = 100.46 + 0.985647 * sinceJ2000d + coordLong + 15*UTHd

    if (LST > 0):
        while (LST > 360):
            LST = LST - 360.0
    elif (LST < 0):
        while (LST < 0):
            LST = LST + 360.0

    #HA calculation

    HA = LST - RA


    if (HA < 0.0):
        HA = HA + 360.0

    #ALT and AZ

    sinDEC = np.sin(degToRad(DEC))
    sinLAT = np.sin(degToRad(LAT))
    sinHA  = np.sin(degToRad(HA))
    cosDEC = np.cos(degToRad(DEC))
    cosLAT = np.cos(degToRad(LAT))
    cosHA  = np.cos(degToRad(HA))

    sinALT = sinDEC * sinLAT + cosDEC * cosLAT * cosHA

    ALT = radToDeg(np.arcsin(sinALT))

    cosALT = np.cos(degToRad(ALT))

    cosA = ( sinDEC - sinALT * sinLAT ) / ( cosLAT * cosALT )

    if(cosA < -1.0):  #just in case -- if this happens, means rounding errors....
      cosA = -1
    elif(cosA > 1.0):
      cosA = 1

    A = radToDeg(np.arccos(cosA))

    if (sinHA < 0):
        AZ = A
    else:
        AZ = 360 - A

    # print("Point your telescope at ALT: ", ALT, " and AZ: ", AZ)

    altAzCoords = [ALT, AZ]

    return altAzCoords
```

Figure 2: Script for Calculating Altitude and Azimuth from Right Ascension and Declination

2.2 User Interface

The system is interfaced via the web, for a few reasons. Firstly, it provides the easiest way to make a GUI for a Python program, using Flask in conjunction with HTML. Secondly, it allows for faster computing and querying of large datasets. The idea is that it is faster for the server to query, retrieve, calculate, and truncate data than it is for the Raspberry Pi. This also has

the bonus of allowing usage by anyone on demand, whether they have a compatible set of hardware as us or should they want to manually gather data to use with a standard telescope.

Pictured below is the input and the output of this graphical interface. There are many manual possibilities, but the main fields that are necessary are: Whether they are using the full automatic system, which celestial body they want to see, and when they want to see it. If they do not have the full automatic system, they may input their coordinates on their own. There is also the choice for manually inputting time zones, though that is not necessary since it can be computed by the server based on the location coordinates.

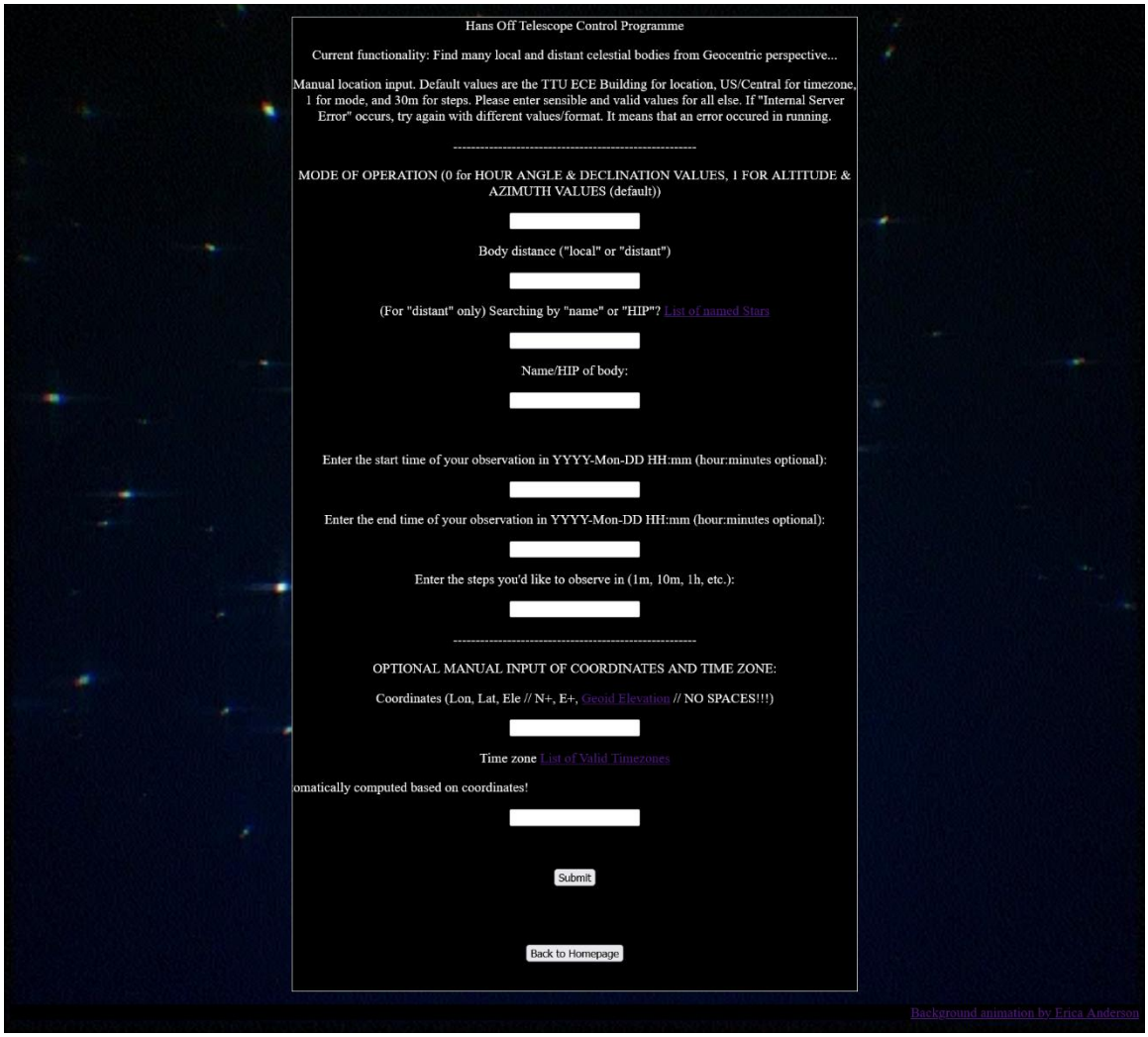

Figure 3: Celestial Body Query System Input Interface

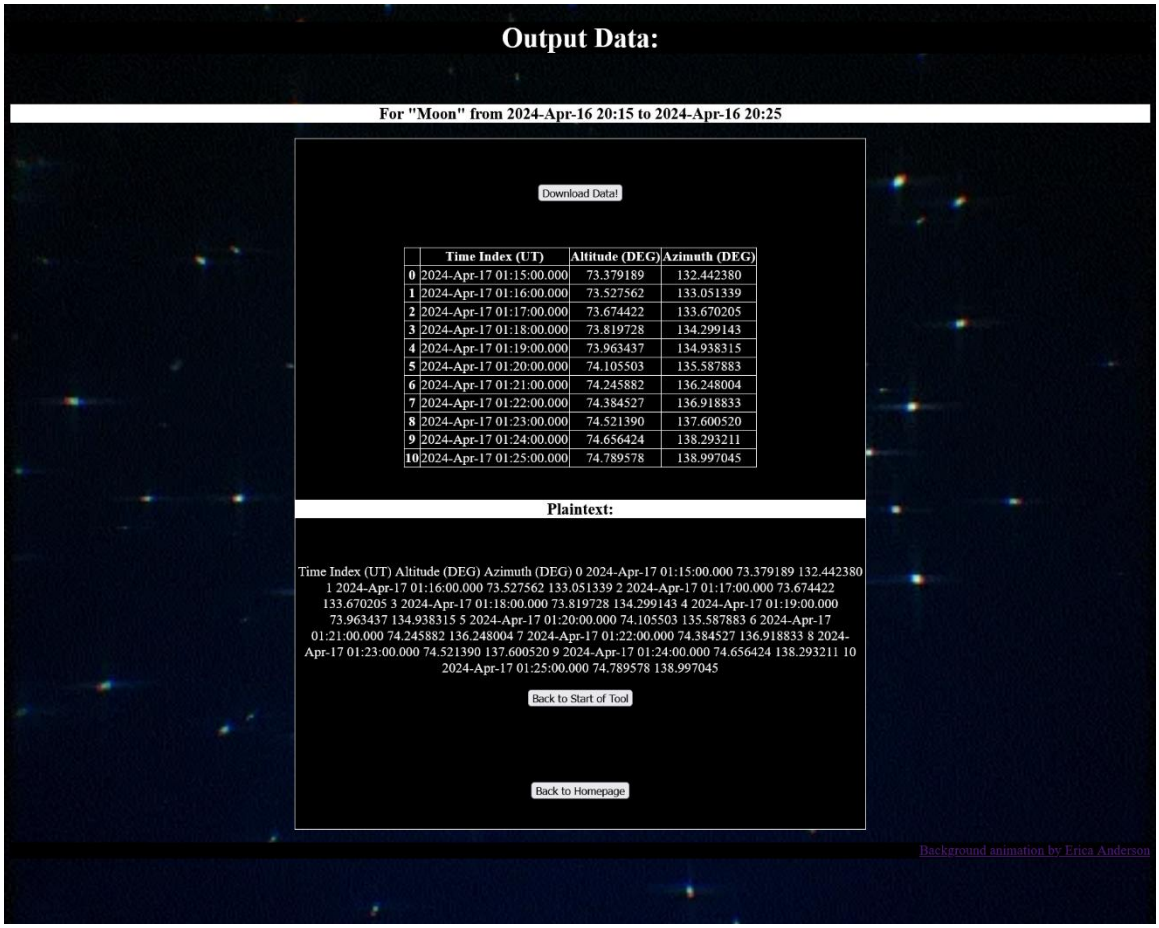

Figure 4: Celestial Body Query System Output Interface

2.3 SFTP Communications

In order to transfer data between the Raspberry Pi on the telescope and the server, we use SFTP from the Raspberry Pi's perspective. Simply, the Pi will gather onto a file the necessary information for the server: coordinates. The file will have a unique title so that it can find a match later. The Pi sends the file up to the server via SFTP and begins to wait for the matching

response file to appear in the same directory. After the user inputs their query on the server's UI, the data is generated and the necessary file is created. The Pi notices this file and downloads it.

On some more details on process specifics, the way that the Pi "notices" the new file is by it repeatedly trying to access the metadata via the "stat" command. Once the command returns valid metadata, it will download the file.

Setting up the SFTP environment involved created a "chroot jailed" user, which means their only permissions are to use SFTP commands within a specific directory that they see as the "root". This was done for security and organizational purposes, as it allows for a soft barrier from any random attacks and queries as well as it allows us to easily manage the data from this project. On a strong note, though, chroot jailing is not a security feature. If someone truly wanted to, they could easily bypass the system, especially as it is a decades-old method of containment.

2.4 Camera

The camera system on this project is fairly simple. Photographing itself is done via a standard command- "raspistill". The trickier portion is the focusing of the camera. For this, OpenCV sharpness metrics are used to determine how clear the image is. From there, the program tells the hardware controlling the focal length of the camera which way to turn. The program will try to find the clearest possible picture by aiming towards a direction, passing over a maximum sharpness value, and then returning to that maximum. Some repetition is done of this process to deal with false positives of the maximum being reached.

2.5 Geo-location

The geo-location aspect of this project is decently simple: a GPS module relays the location

of the telescope and a magnetometer the orientation. Using simple Python libraries, the GPS data easily interfaces with the SFPT communications portion to provide the calculation script with precise information on where to calculate for. The magnetometer is a bit more involved, and requires some mathematics to set up.

Depending on if the magnetometer has just a gyroscope or a gyroscope and an accelerometer, the complexity (but also accuracy) improves greatly, especially on inclines (which is necessary for calculating the telescope's Altitude). Essentially, however, both systems involve computing the vectors between two axes (each instrument has three axis- x, y, and z). In the case that both instruments are used, the accelerometer aids as a support for the gyroscope in the Azimuth direction, but is the only instrument used for the Altitude direction. Without the accelerometer, it is hard to gauge the actual pitch of the device, causing the resultant mathematics mere estimation. Unfortunately, whether by programming error or by hardware error, this system was not fully realized.

3. Hardware

3.1 Hardware Mounting

The chosen method of mounting hardware was the Altitude/Azimuth mount, often called an ”Altazimuth” mount. The reasons for this is that it is simple to use and build. Additionally, our telescope arrived with a non-mechanized Altazimuth mount already, so fitting the necessary hardware is decently simple. Originally, we meant to use an equatorial mount (a 4-axis mount that follows along the rotation of the stars in the sky), however the increase to four motors, need for a gearbox, and difficulty with connecting the various electrical components together proved too challenging. Additionally, it is not necessary for most photography, especially what we were aiming for.

Figure 5: Telescope on the Altazimuth Mount

3.2 Motors

The motors used are NEMA-17 stepper motors and are drivern by A4988 stepper drivers. The motors are controlled decently simply. They are connected via stepper drivers into GPIO pins that are simply turned on and off, with the step signals having a brief on state, the direction signals being switched sparingly, and the enable pins only being switched on and off at the start and end of the runtime. There are three motors, one for the Altitude axis, one for the Azimuth axis, and one for controlling the telescope's focus. The motors are controlled by two variables: a current value and the desired value. In the case of the Alt/Az axis motors, for examples, these values are degrees. The motors are steppers and after setting them up/turning them on, are each controlled with two pins: one for taking a step and one for changing direction. Using the magnetometer in conjunction with the desired direction for the telescope to point at, we turn the motors until the two values match.

3.3 Sensors

The location identification system contains two hardware components: a GY-NEO6MV2 GPS Module and a QMC5883L Magnetometer Module or an MPU6050 Accelerometer & Gyroscope Module. With these combined, we can easily determine where the telescope is and where it is oriented. An additional component is the time zone determiner, which is necessary for Hour Angle and Altitude/Azimuth calculations.

The GY-NEO6MV2 receives GPS data from satellites, which we can easily write to a text file using a Python script. It connects via UART, so the signals that come in are very simple to receive legible data.

The QMC5883L, a simple magnetometer, is essentially an electronic compass based on its measurements of the magnetic field. The MPU6050, which is a 3-axis accelerometer and gyroscope, is a more sophisticated device, as explained in Section 2.5. These sensors are crucial during telescope runtime, as they will dictate where exactly the telescope must point to in the sky. As the aforementioned Python script controls the motors based on the coordinates from the queries, these pieces of hardware will feed their data into that script to keep it on track.

3.4 Camera

The camera system is comprised of a Raspberry Pi High Quality Camera, a C-mount to 1.25” adapter, another worm gear mechanism, another stepper motor, and the telescope itself: a Heritage 150 Tabletop Dobsonian. The camera attaches to the C-mount adapter atop the eyepiece which is connected electronically to the Raspberry Pi. The Pi stores the image locally. The stepper motor and worm gear control the focus by rotating the eyepiece attachment. The system camera and focus system is shown in Figure 6, with the C-mount attached onto the gearing system so that it may move the camera up and down to adjust focus.

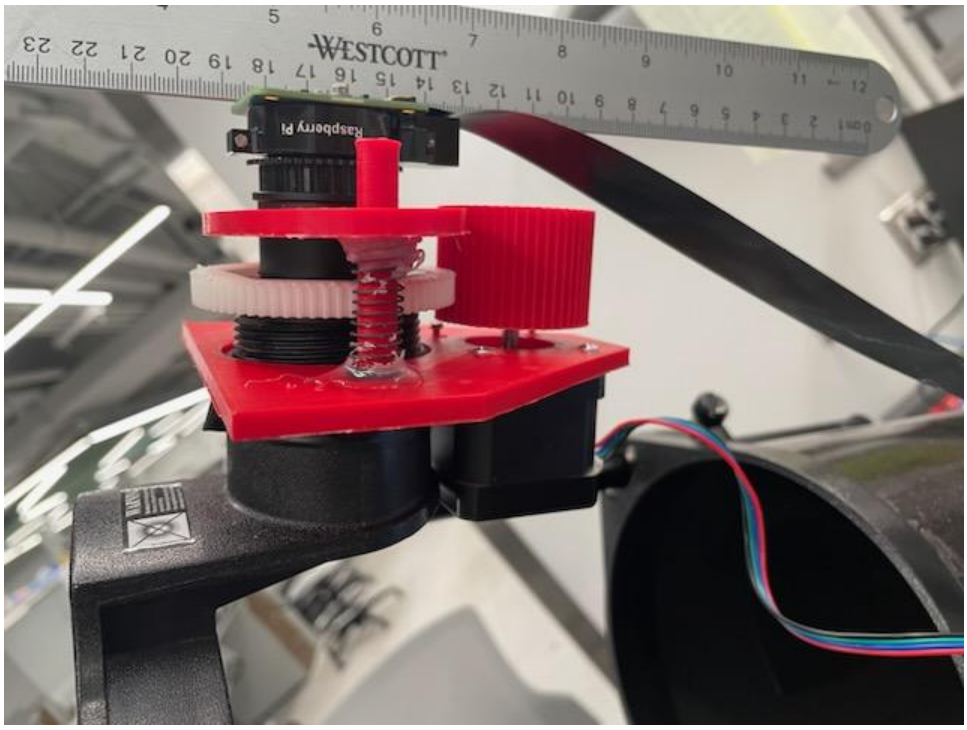

Figure 6: Camera Mount with Motor and C-mount Attached

4. Considerations

Though our project is a one-off production of a low-power and low-footprint device, there are still some considerations we as a group must keep in mind for the safety and wellbeing of ourselves and others.

4.1 Safety, Public Health, and Welfare Considerations

When building this project, special care was taken for safety. When using power tools, all appropriate PPE was worn and safety precautions taken. During and between work, organization and cleanliness was maintained to avoid any dangerous trips, cuts, or other bodily damage. Any electrically conductive and capable materials and devices were stored in safe places to avoid any damage to components or persons.

4.2 Global, Cultural, Social, Environmental, and Economic Factor Considerations

Our project faces a miniscule impact in global, cultural, social, environmental, and economic issues. The inefficiencies of our project are minimal, even in their developmental stage, as they do not run for long periods. When complete, the device will be optimized to use as little power as possible, will use rechargeable batteries, and be in operation only when necessary.

5. Conclusion

The final state of the “Hans Off!” project is one of a disappointing shortcoming. Most of the systems work as needed, but two crucial systems are preventing any meaningful result. First, the magnetometer, as previously explained, was not functioning as needed. For both devices, the raw values were inconsistent, even with significant calibration attempts. This could be due to magnetic interference or perhaps faulty hardware (they were used devices, so it is not out of the question). This issue is remediable by using a more primitive approach of simply counting steps from a set origin point, however there are two issues with this approach: It requires user intervention to have it begin from a specific point and it is inaccurate due to how the motors are controlled. The second issue could be remedied with more sophisticated stepper drivers and more complicated code, but those were unreachable by the time the issue was realized. The second issue was in the hardware not suitably built for the task. The telescope was unfortunately far too heavy and in effect either the motors would slip or the mechanical gears could not keep friction. That was the main reason we switched from an equatorial to an Altazimuth mount- so as to reduce complexity. However, by the time we switched, it was far too late to fix the underlying issues.

In total, it was an interesting project with a massive potential, but it was cut short by unfortunate last-minute troubles. If it were to be repeated, I would advise anyone to keep it simple and to avoid trying to recreate the wheel.

References

1. Study Country. “How often are star maps updated?,” https://www.studycountry.com/wiki/how-often-are-star-maps-updated, March 19, 2024
2. Michelfeit, Jannik. “About — timezonefinder /bin/sh: 1: poetry: not found documentation,” https://timezonefinder.readthedocs.io/en/latest/3_about.html, March 18, 2024
3. European Space Agency. “Common Star Names,” https://www.cosmos.esa.int/web/hipparcos/common-star-names, March 18, 2024
4. Burnett, Keith. “RA and DEC to ALT and AZ,” http://www.stargazing.net/kepler/altaz.html, January 2024

Appendix 1

Gantt Chart:

Task Name	% Done	Eric	Eddy	Jason	Zach
Subtasks					
Information Reading					
Research Integration	95%			x	
Star/Planet Data reading	100%			x	
Star/Planet Data parsing	100%			x	
Telescope Location	80%			x	
Necessary action output	80%			x	
Debugging/polishing				x	
Telescope movement					
mount design	100%	x			
mount building	100%	x			
stepper integration	100%	x			x
software integration	50%	x		x	
Telescope UI					
Research Integration	95%		x	x	
Develop	95%			x	
Debugging/polishing	10%			x	
Data(base) management				x	
Compile Data	100%			x	
Optimize	100%			x	
Picture Taking					
Camera Research	95%		x		
Camera Integration	90%		x		
Picture Taking Process	95%		x		
Debugging/polishing	0%		x		
auto focus	50%	x	x		

In our Gantt Chart, we generally had decent progress. The main issues we failed to observe were in the integration of movement software and the telescope location. Additionally, there was an unaddressed unbalance in workload, which surely contributed to our issues.

5.1 Budget Chart:

	Running Total			Total Estimate		
Direct Labor:						
Category or individual:	*Rate/Hr*	*Hrs*		*Rate/Hr*	*Hrs*	
Eric	15	190	$2,850.00	15	165	$2,475.00
Eddy	15	162	$2,430.00	15	165	$2,475.00
Jason	15	146	$2,190.00	15	165	$2,475.00
Zach	15	120	$1,800.00	15	165	$2,475.00
DL Subtotal (DL)		Subtotal:	$6,420.00		Subtotal:	$9,900.00
Labor Overhead	*rate:*	100%	$6,420.00	*rate:*	100%	$9,900.00
Total Direct Labor (TDL)			$12,840.00			$19,800.00
Contract Labor:						
Lab Assistant	40	0	$0.00	40	3	$120.00
Classmate	15	2	$30.00	15	5	$75.00
Instructor	200	0	$0.00	200	1	$200.00
Total Contract Labor (TCL)			$30.00			$395.00
Direct Material Costs:			$339.57			$250.00
(from Material Cost worksheet)						
Total Direct Material Costs: (TDM)			$339.57			$250.00
Equipment Rental Costs:	Value	Rental Rate	Total	Value	Rental Rate	Total ~20 Day
		0.20%	$0.00		0.20%	$0.00
		0.20%	$0.00		0.20%	$0.00
		0.20%	$0.00		0.20%	$0.00
		0.20%	$0.00		0.20%	$0.00
Total Rental Costs: (TRM)			$0.00			$0.00
Total TDL+TCL+TDM+TRM			$13,209.57			$20,445.00
Business overhead		55%	$7,265.26		55%	$11,244.75
Total Cost:		**Current**	$20,474.83		**Estimate**	$31,689.75

SUMMARY:	
Labor + OH	$12,840.00
Contract Labor	$30.00
Materials & Equip Rental	$339.57
Overhead	$7,265.26
TOTAL	$20,474.83

As per the budget, most of it was kept within, with the hours being a bit of an overshoot by a couple of us. I would like to note that the project's cost is not indicative of real cost on account of relaxed reporting, since a lot of components were either already at hand or we bought them on our own.