Automatic Planet Tracker

Amar Barucija, Daniel Toone, Jake Steers

***Abstract*—Amateur astronomy is a hobby in which average people observe the stars and planets using small budget, do-it-yourself builds. This project takes inspiration from amateur astronomy through the design of an automatically driven planet tracking telescope. The telescope is capable of automatically following planets or stars as they travel through the sky. Users can choose what celestial body to track in a mobile phone based app. The user is also capable of manually controlling the telescope movement through this app. Movement of the telescope is achieved by the use of motors with gearing connected to a custom built mount upon which the telescope is mounted. The motors movement is controlled by a micro-controller device that resides on the mount. This micro-controller receives instructions from a computer device that interprets user input from the mobile app. The lens of the telescope itself will not automatically adjust to the planet it is pointing towards. Users will manually focus the lens of the telescope after choosing which planet to track.**

I. Introduction

THIS report details the design of a motorized mount for a telescope to be attached onto. The project was inspired out of amateur astronomy projects where people create home made telescopes, mounts and automated systems for telescopes. Automatic telescope controllers are known as GoTo telescopes in the world of amateur astronomy. Creating a motorized telescope mount involves mechanical, electrical and software engineering challenges that will make for an interesting telescope for people to enjoy.

The telescope mount will have 3D-printed parts attached to allow for rapid prototyping of said parts. This mount will specifically be an alt-azimuth mount, which allows for relatively simple horizontal and vertical movement compared to other mount types. The modified mount will have slots for motors that will be screwed in for stability and strength.

There will be two motors, one for vertical and one for horizontal movement. The motors themselves will be stepper motors which allow for precision movement. These motors will also be accompanied by motor driver circuit boards. Gears will be attached to the motors as the gearing will further increase the precision of movement required for tracking planets and stars. The motor driver boards will be connected to a micro-controller to provide necessary analog to digital converters to read motor RPMs. The micro-controller can provide digital outputs, allowing us to specify rotation direction and speed of actuation. Digital inputs will be used to receive coordinates from a communicating computer device.

The micro-controller itself is in charge of translating its received coordinates into rotational commands for the motors to rotate into new positions. These translations will be written into code stored on the micro-controller where writing the code for such a translation will be one of the main tasks of this project. The computer device of our system will store code of its own utilizing an external library to provide coordinates of a given planet to the micro-controller device.

To control the mount, the user will use a mobile application to select a planet to track. The mobile application will then communicate this selection to the computer device via Bluetooth. The computer device will send the coordinates of the user selected planet to the micro-controller. The micro-controller will then point the mount to the coordinates. This flow describes our final deliverable, a fully functioning automatic telescope mount.

II. Background

Building telescopes is very common in amateur astronomy, as such, there is a litany of work from which to draw from. As we are building an automatic telescope, it is worth noting that these are generally referred to as GoTo telescopes within the amateur astronomy community. As there is an increased degree of complexity with telescope motorization, we have done research into other related work focused on automatically driving a telescope.

A. Related Work

Our system will require software engineering to create motor manipulating control logic within the system's micro-controller for manipulating the stepper motors to rotate the alt-azimuth mount. This system design of allowing a user to select a planet for the telescope

The authors are in the Computer Engineering department at the University of Utah.

to automatically track is known as a GoTo telescope. The effort in coding the control logic of our micro-controller will face many bugs and code reiterations before getting the mount to rotate automatically. However, an open source control logic for driving a GoTo system exists under the name of OnStep and provides clear specifications on how to build such a system using OnStep [4]. Using this guide as a resource to design our system's automatic tracking technology will be beneficial to avoid any coding or design pitfalls related to the micro-controller. The OnStep website also has builds that other people have done using this technology. However, we are not directly using OnStep because a major aspect of our project will be writing the control logic which is what OnStep provides. Therefore, some of the work provided may not be applicable to our system. However, we can still use these for reference and examples when building our project.

Designing a GoTo telescope will require knowledge of the mathematical units utilized in calculating astronomical positions and mount angles. Basic GoTo telescopes systems use a simple approach of calculating the direction of a given target and converting said direction into mount angles for which the telescope must accurately point towards. The direction of a given target is calculated in nominal mount coordinates where mount angles or rotations are calculated using parallactic angles. To elaborate, a parallactic angle is the spherical angle between the hour circle and the vertical circle passing through the zenith of a given celestial object. However, these calculations can be known to have modest accuracy whereas an approach utilizing a rigorous matrix algorithm would yield far greater precision, as stated by Wallace [6]. Although the simple approach mentioned prior will be implemented into our automatic telescope system, this other approach using a rigorous matrix algorithm will be beneficial if accuracy of our telescope becomes a concern.

The design for our telescope will require the use of hall effect sensors placed onto the telescope's mount for calibration before regular use. However, the concern here is that these sensors will not nearly be as accurate for calibrating as it will be to manually point the telescope to specific stars. Therefore, a quality of life addition to this project will be to attach a camera or imaging device onto the telescope to make the calibration process automatic. This addition is considered to be a stretch goal for our project and will be utilized to automatically align our telescope to the sky with great precision, as detailed by this article [2]. The automated calibration works by utilizing point pattern matching and normalized correlation matching. This is done by taking a picture of the sky and synthesizing an image of the sky, both relative to the telescope's current angle. Both images are then compared to identify more precise locations of the stars found in both images. If time permits, an addition like this will be a significant quality of life improvement to the user experience of our automatic telescope.

III. Implementation

This section details the final implementation of our alt-azimuth GoTo mount, an automatic telescope to track planets. All files associated with the 3D designs, mobile application code, and Raspberry Pi Python scripts can be found in our team's GitHub page [3]. The alt-azimuth, as shown in Fig. 1, allows for relatively easy horizontal and vertical movement compared to other mount types. We adapted an existing alt-azimuth mount to reduce the time and complexity in automating the telescope. This turned out to be both a boon and a limitation as, while it was easier to design around the existing mount, it also restricted us because our designs could not interfere with the rotation of the telescope or base mount. The mount's horizontal movement is powered by a stepper motor with a planetary gearbox attached to it. A planetary gearbox provides a very large step down for accuracy. The vertical movement utilizes the same kind of stepper motor with a planetary gearbox. As planned from the proposal, the telescope does work using GoTo logic written in Python code. One major change from the initial plan of the proposal was combining the computer device and micro-controller into the Raspberry Pi. The Raspberry Pi now serves those two roles, that of taking in the requested planet, getting the coordinates, and then driving the motors based on those coordinates.

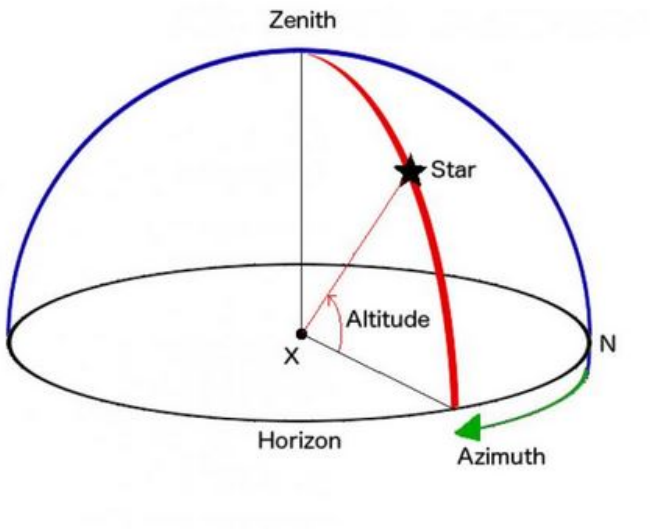

Fig. 1. Showing the function of an alt-azimuth mount [1]

A. Raspberry Pi

The user's selection of a planet in the mobile application is be sent to the Raspberry Pi via Bluetooth. The Raspberry Pi receives the coordinates from the SkyField library based on the selected planet using Python [5]. The SkyField library provides the astronomical coordinates and angles of planets alongside functionality to calculate angles of planets from a given longitude and latitude on earth's surface. The library's versatility and ease of use greatly simplified our project. One major change is that, instead of sending the coordinates to a STM32F072 Discovery Board controlling the mount's motors, the Raspberry Pi controls the motors. This was done as a simplification of our workflow and only required us to write python code for the Raspberry Pi, forgoing the embedded systems C that would've been required for the STM32F072 Discovery. The Raspberry Pi's GPIO pins were used to signal the motor drivers. One concern was that because the Raspberry Pi ran a full operating system, the speed of the GPIO signalling was subject to the operating system's scheduler. This meant we could not go any faster than 1μs without workarounds. However, this ended up being sufficient as the higher speeds caused issues with the physical movement anyway. The code as written sets up a Bluetooth server and awaits a connection from the user app. Once it receives this connection it can be manually moved into position by the user to perform a calibration by moving the telescope to point north and vertically level. After this calibration, the user is free to select any planet and the code will retrieve the location information from the SkyField library. This provides an expected position against the telescopes current position. Signals to the motor drivers will be sent until these match each other, moving the telescope towards the selected planet. Calibration of the telescope mount and stepper motors are required to ensure that the mount slews in the correct direction. Initially we were going to use hall effect sensors to do an automatic calibration but realized that this would only work for the vertical. The horizontal would require an onboard compass to point north. It was eventually decided for the sake of time that manual calibration would be sufficient for our purposes, though we are confident that it could be automatic with some additional hardware. The system diagram of communication from the mobile application to the Raspberry Pi to the motor drivers can be seen below in Fig. 2. The code went through several iterative designs because we had to work around the Bluetooth communication protocol.

1) Blocking: The first approach we used to communicate with the mobile application involved using Python's native Bluetooth sockets. With this approach, we simply connected the Pi and mobile device to then allow calibration logic to be handled. However, tracking a planet did not work because once it moved to a selected planet, the Python script expected more data from the user instead of continually tracking said planet. As a result, the script would continually wait in this loop expecting data. One solution for this issue was to put in a timeout for this waiting loop. If it timed out, it was able to continue tracking the user's last selected planet. However, this still meant this timeout would always occur, resulting in the Python script constantly relying on catching exceptions to do its normal logic. Furthermore, the timeout became in issue for calibration. Since calibration is an interruptible process where the user can quickly select one button after another, the program could not work fast enough to rotate the telescope since it was bound to the duration of the timeout. As a result, the Python script required a restructure.

2) Multi Threaded: The final and most effective restructure of the Python script was to use a multi threaded approach with only two threads. One thread is responsible for connecting to and communicating with the mobile application. The other thread is responsible for moving the telescope based on the last read instruction in the first thread. This means that the instructions sent by the mobile application become stored in global variables for both threads to have access to. As a result, thread locks were utilized to prevent any form of race conditions or thread exceptions. This solution worked wonderfully as it no longer bound the calibration speed to a timeout. Furthermore, since a single thread became responsible for receiving and storing instructions, the other thread could easily be interrupted unlike previously. A user could choose a planet and then a different planet shortly after without any miscalculations from the Python script or socket exceptions from the mobile application. To elaborate, the GoTo logic is continuously tracking the telescope's current position. Therefore, changing the telescope's trajectory while already pointing to a planet will still maintain accuracy. This accuracy is what resulted in the mobile application no longer needing a loading screen between each planet input.

B. Mount design

We used an existing alt-azimuth mount with a mounted XT8 telescope that was loaned to us. Given this, we were hesitant to cause any serious modifications at first. However, we are confident that the telescope can be de-motorized and used as normal. The alt-azimuth mount was chosen as it is relatively simple to motorize compared to other mount types. Conceptually, it also easy

to understand, as there are two independent movement axes. Stepper motors are necessary to provide accuracy for moving the alt-azimuth mount. Considering that most stepper motors have a 360 degree rotation with 200 steps, each step would move the motor 1.8 degrees. However, 1.8 degrees of movement per step is not accurate enough to appropriately move the mount and telescope. It is predicted that the accuracy of the angles for which the telescope must point to any given planet will be in the thousandths of degrees. To accommodate this accuracy, it was necessary to use a high ratio planetary gearboxes attached to the mount's stepper motors. These planetary gearboxes provide up to a reduction of 100:1, resulting in a step angle of 0.018 degrees, which grant even more accuracy to the alt-azimuth mount's movements. We initially thought that this kind of reduction would still be insufficient. However, our purchased motor drivers had the ability to do micro-steps which could further subdivide our movements for precision. Our gearing system adds further reductions which meant our telescope was very precise while still being able to move relatively quickly. For gearing our initial plan was to use bevel gears to drive the vertical and horizontal mount as they were relatively cheap, easy to find, and required no significant mounting supports other than attaching the motor directly against the wood of the mount. However, the bevel gears proved to be insufficient for both vertical and horizontal purposes as described in their respective sections.

1) Vertical movement: The vertical bevel gear would push the telescope up instead of rotating it as the telescope was not held down except by gravity when placed in the mount. We eventually replaced the vertical bevel gears with a synchronous timing belt. This had the advantage of holding the telescope down based on tension, and more efficiently transferred force from the motors. This implementation still had issues though, as it required the motor and gear to be on the same plane. This meant we had to create a 3D printed mount that would extend outwards from the mount to hold the motor in place. Additionally, it was discovered the belt did not have sufficient tension to move the telescope upwards once it was past a certain point due to it being heavier in the front. This was resolved by adding a bearing to put tension on the belt. The use of the synchronous timing belt also resolved our issues with backlash as we could not detect any when calibrating the telescope. The final design of the vertical mount is shown in Fig. 3

2) Horizontal movement: The horizontal mount was able to rotate with the bevel gears for a time until we believe that the threads that held the bolt in place were stripped. This meant that the bolt freely rotated instead of rotating the top half of the base mount against the bottom half. To fix this, we used a 3D printed plate screwed onto the bottom horizontal platform that held the bolts head in place. This achieved a similar effect as earlier. However, we ran into another issue where, with the bolt held into place, the bevel gears forces meant it would push the 3D printed mount that held the motor in place against the top horizontal base. This eventually lead to the 3D print breaking and no longer holding the motor against the bevel gear. To resolve this issue, we decided that a synchronous belt gear would avoid this issue of forces. While it was a tight fit, we were able to fit the motor housing vertically on the forward part of the mount. However, due to the weight of the telescope the belt teeth would skip over the gear instead of rotating it. To resolve this we increased the tension of the belt by adding several bearings to push against the belt. This allowed the mount to rotate but was still unreliable as

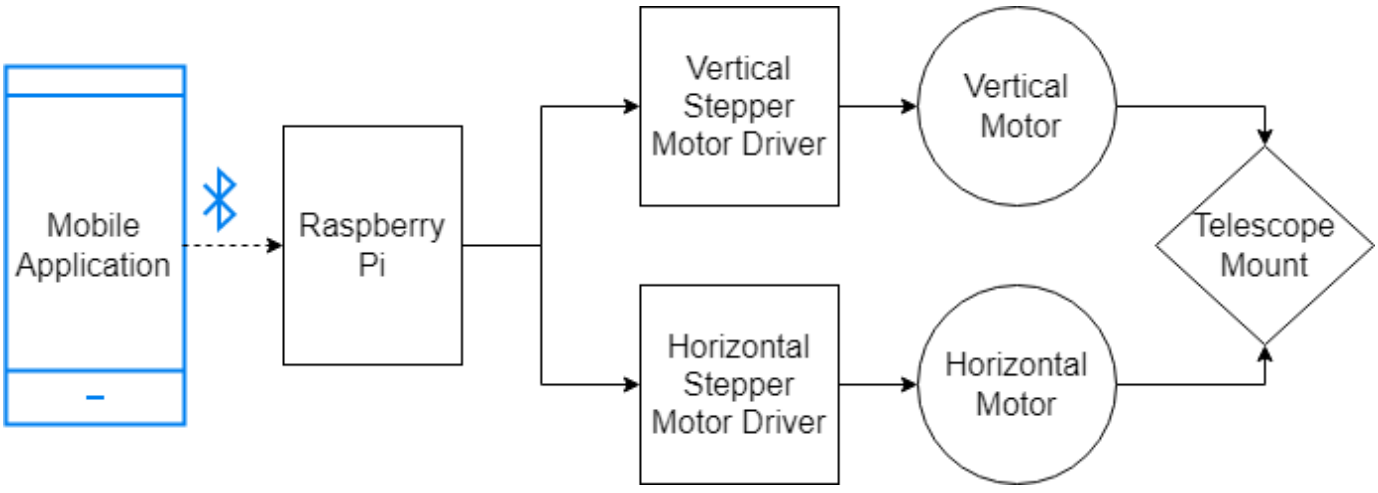

Fig. 2. The planned flow of the command logic

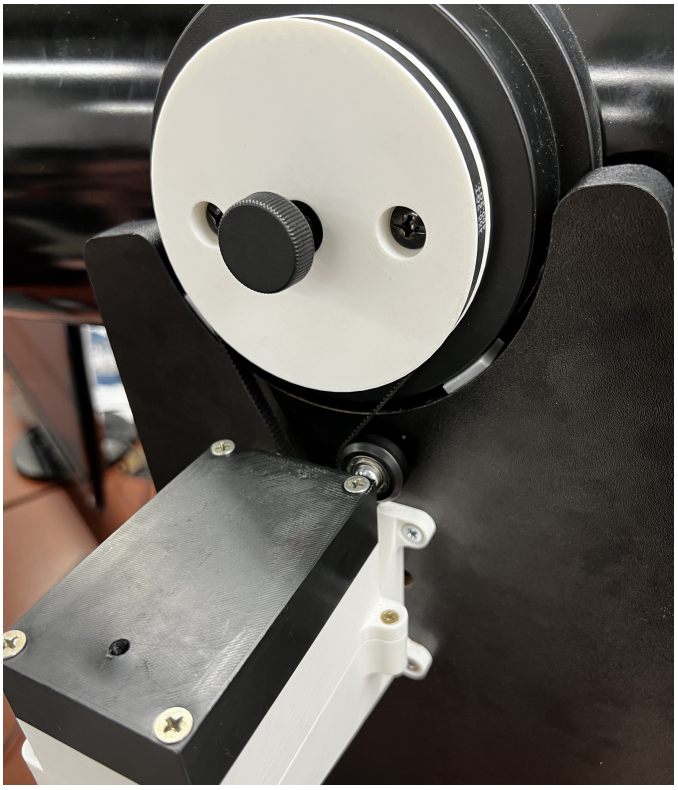

Fig. 3. The final design of the vertical mount

the belt sometimes slipped off the bearings which would result in it not moving. To mitigate this we stacked two bearings on top of each other to allow the belts to move around more without slipping off. The final design is shown in Fig. 4

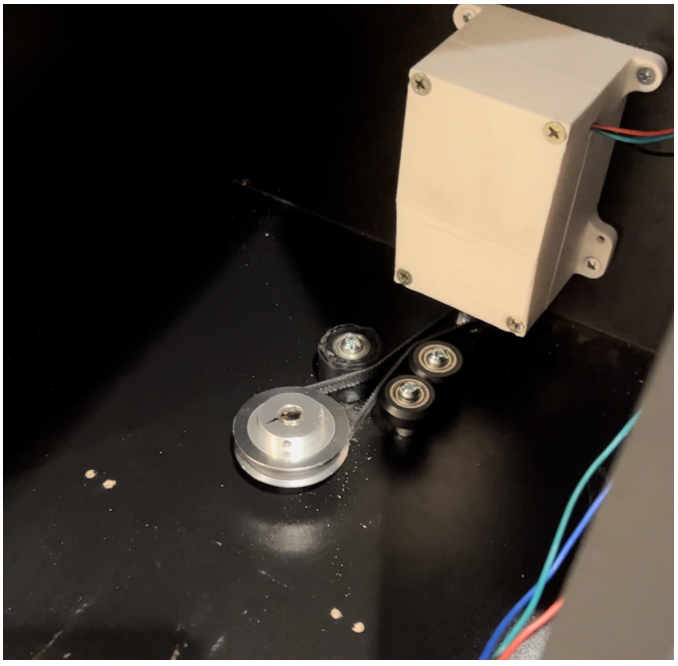

Fig. 4. The final design of the horizontal mount

C. Mobile Application

1) Flutter SDK: The mobile application was designed and implemented using the Flutter SDK. Flutter was chosen primarily for its convenience in platform portability. To elaborate, an app developed through Flutter can be ran on either an Apple or Android device without requiring specific code changes. Platform portability was an important focus at the beginning of this project as both Apple and Android devices were present on the team.

2) Bluetooth: Bluetooth was chosen as the communication protocol between the mobile application and the Raspberry Pi script since it allowed the devices to communicate together without the need of internet. Furthermore, experience was already present on the team regarding Bluetooth socket programming via Python. To our surprise, programming Bluetooth logic in Dart (Flutter SDK's language) was significantly different from Python.

Programming for Bluetooth on Flutter became an issue at the very start of the project. It was quickly discovered that such communication would only be possible on Android devices and not Apple Devices. This is due to the fact that the Raspberry Pi 4 operates on the Radio Frequency Communication (RFCOMM) protocol for its Bluetooth module where Apple devices have been using the Bluetooth Low Energy (BLE) protocol for over a decade. Therefore, Bluetooth communication from an Apple device to a Raspberry Pi 4 is impossible due to the differing communication protocols. On the other hand, Android devices are capable of such communication since many still utilize the RFCOMM protocol. Once this knowledge was discovered, the mobile application development was shifted towards testing and utilizing the Flutter application solely on an Android device. The Android device used by the team was a Google Pixel XL.

3) Application Design: The user interface of the application was designed to be a scrolling list of planets for the user to be able to easily identify their inputs. The top row of the application was also reserved for buttons whose functionality aimed towards connecting to the Raspberry Pi alongside calibrating the telescope. Furthermore, the application was designed to disable all buttons except for the connect button until a successful connection was established with the Raspberry Pi, as shown in Fig. 5. With a connection established, the connect button would green while enabling all other buttons with a blue color, as shown in Fig. 6. A calibration menu was specifically designed to look like a remote where the user can send commands to point the telescope up, down, left, or right. Additionally, a centered button on

the calibration menu can be used to stop the telescope from moving, as shown in Fig. 7.

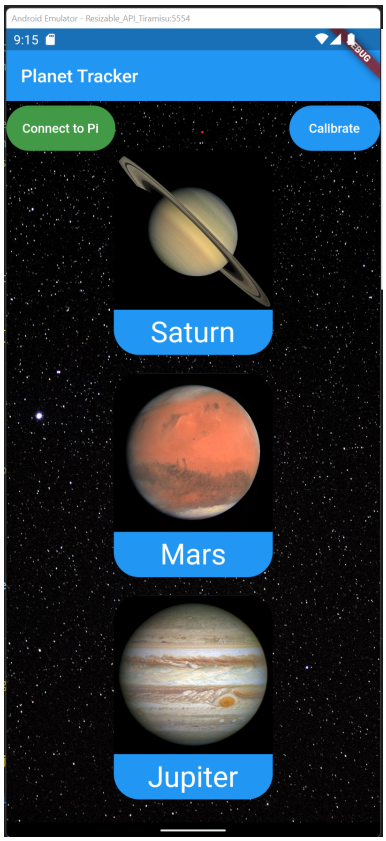

Fig. 5. The design and layout of the mobile application's main menu. All buttons but the connect button are disabled, as indicated by the grey color of the other buttons.

4) Automatic Calibration: With the project's initial design, it was anticipated that automatically calibrating the telescope would be possible without the need of manual movement from the user. Eventually, a separate menu was designed as a prototype for such functionality. This menu contained a compass and would utilize the device's gyroscope to determine how far off the device is from facing North. However, the Flutter library to read data from an Android device's gyroscope was failing to work properly. Whenever gyroscope data was attempted for reading, an exception would be thrown indicating that the gyroscope sensor was offline. After some further investigation, it was discovered that the Google Pixel XL does not contain a gyroscope module. As a result, this menu, alongside the automatic calibration functionality of the project, was scrapped due to the inhibiting factors of our Android device. The design of the automatic calibration menu, alongside the movement of the compass based on the rotation of the Android device, can be found in Fig. 8 and Fig. 9.

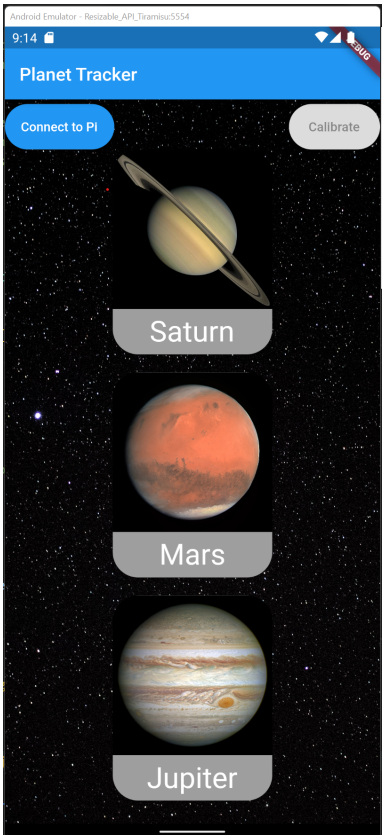

Fig. 6. The mobile application's main menu once connected to the Raspberry Pi. Here, the connect button is highlighted green to indicate a successful connection, and all other buttons appear enabled via their blue color.

5) Loading Screen: One of the greatest challenges with designing the mobile application's Bluetooth communication was engineering a solution towards preventing user input. The project's system of communication between the mobile application and Raspberry Pi was designed to only allow user input for a single planet until the Raspberry Pi replies with a completion message. This way, the user cannot spam other planets and eventually cause socket related exceptions for the mobile application or miscalculated angles for the Raspberry Pi. The challenge was mitigated through the use of a loading screen as shown in Fig. 10. More specifically, a spinning circle on the center of the screen to indicate to the user that the application is loading and will not be done loading until the GoTo planet rotation is completed.

Ultimately, this loading screen was used for almost the entirety of the project, but was scrapped by the end due to some asynchronous issues the mobile application kept encountering. Since the loading screen would be placed as the topmost layer of the application's layout, the code would have to manually pop it out of existence to reintroduce the menu that the loading screen was layered upon. However, this became an issue when the mobile

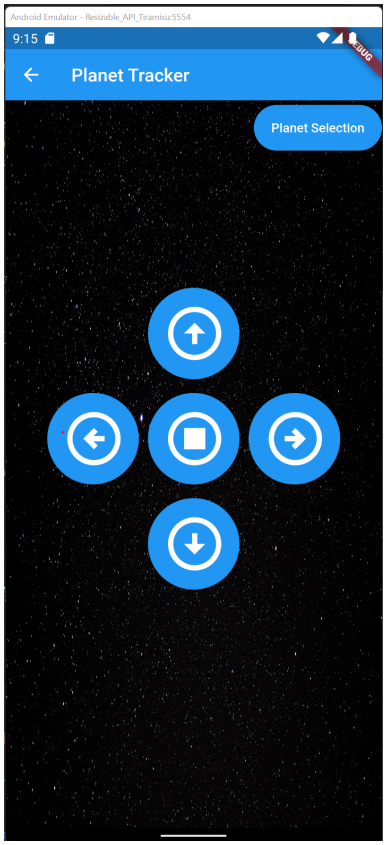

Fig. 7. The design and layout of the mobile application's calibration menu.

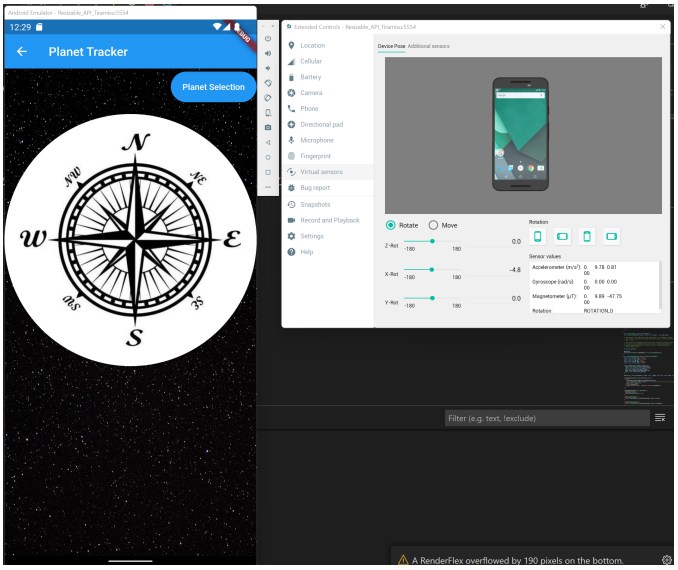

Fig. 8. The design and layout of the automatic calibration menu with the emulated Android device set upright.

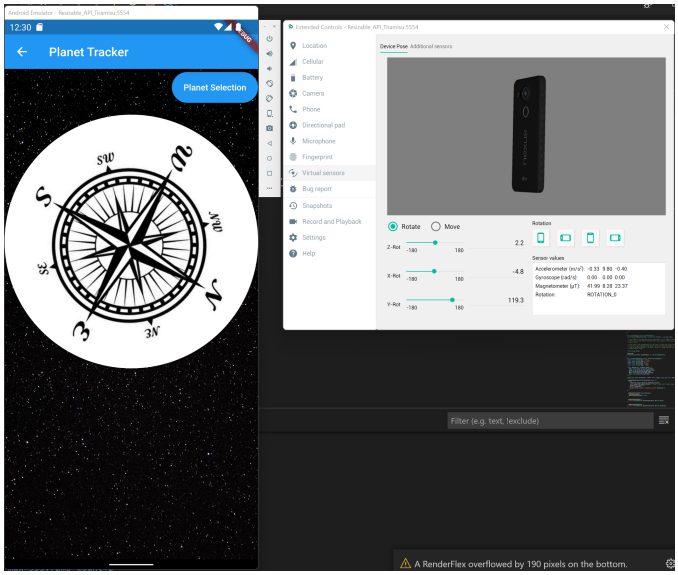

Fig. 9. The automatic calibration menu where the compass is rotated alongside the rotation of the emulated Android device.

Fig. 10. The appearance of the loading screen built into the application as a way to prevent users from spamming inputs.

application would interpret a single response from the Raspberry Pi as multiple responses. Therefore, the logic to pop off the topmost layer would be applied twice, thus popping off the main menu itself. Since the Raspberry Pi ultimately shifted its script to a multi threaded structure, the need for a loading screen became unnecessary since continuous user input no longer had any negative impacts on the Pi's tracked logic.

IV. Summary

This project was a significant learning experience for the entire group, especially because of it's mechanical components. This provided a real appreciation for the challenges and difficulties that mechanical engineers face when designing reliable and effective systems. Despite the many difficulties we think this was an effective learning tool that put the team outside of it's comfort zone and provided an interesting project not only to design but to use from an user perspective. It was engaging to develop a multi-level system that required mobile development, scripting, operating system troubleshooting, and 3D designing. When put altogether, this project started the team with many interesting but difficult tasks and left the team with better knowledge on how to approach such a system in the future. Due to our consistency in clear communication, the team was able to effectively overcome the many obstacles faced during this project. We were pleased that our project was received well at demo day, and we are proud of the work we accomplished for this fitting computer engineering project. The complete project, set up and ready for demo day, can be found in Fig. 11

References

[1] P. G. Abel. Mounting your telescope. https://britastro.org/2016/mounting-your-telescope, Jan. 2016.

[2] Y. Azzam, K. Kosuge, Z. Wang, A. Alawy, and Y. Hirata. Telescope Automatic Alignment and Pointing using Pattern Matching. In *The Fourth International Conference on the Advanced Mechatronics*, volume 2004.4, pages 96 – 102, 10 2004.

[3] A. Barucija, D. Toone, and J. Steers. Planet tracker. https://github.com/Abarucija/Planet-Tracker.

[4] H. Dutton. Main@onstep.groups.io: Wiki. https://onstep.groups.io/g/main/wiki.

[5] B. Rhodes. Skyfield: High precision research-grade positions for planets and Earth satellites generator. Astrophysics Source Code Library, July 2019. ascl:1907.024.

[6] P. T. Wallace. Rigorous algorithm for telescope pointing. In H. Lewis, editor, *Advanced Telescope and Instrumentation Control Software II*, volume 4848, pages 125 – 136. International Society for Optics and Photonics, SPIE, 2002.

Fig. 11. The complete design of the telescope utilized during demo day.