



**EasyScope:
A GoTo Telescope
Controlled with a Mobile Application**

Final Year Project Report

**DT228
BSc in Computer Science**

James Clarke

C20375736

Stephen O'Sullivan

School of Computer Science
Technological University, Dublin

12/04/2024

Abstract

This report documents the research, planning, development, and validation of system to enable newcomers to astronomy to transform their basic, entry-level telescope into a feature rich, accessible "GoTo" telescope with a low investment of time and money.

The system is designed and developed with principles that, when integrated, result in an exemplary GoTo telescope. These principles were derived from wide research into the current market landscape.

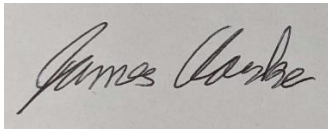
Two main components are designed and developed, a motorised telescope mount powered by Arduino designed to be constructed by the end user themselves at a low cost, and an Android application which controls the mount through a Bluetooth connection.

System evaluation is then carried out, proving that the final system produced holds up well to the standards previously defined.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A rectangular image showing a handwritten signature in black ink on a light-colored background. The signature is written in a cursive style and appears to read 'James Clarke'.

James Clarke

12/04/2024

Acknowledgements

I would like to thank and acknowledge my project mentor, Stephen O'Sullivan. Throughout the entire process, Stephen has provided consistent, invaluable guidance and encouragement. His technical knowledge and project management experience kept me on track and pushed this project to be the best it could be.

I would like to acknowledge my thesis second reader Svetlana Hensman, who's feedback on my interim demonstration helped shape the direction of my final thesis.

I would also like to give a special commandment to my father, whose support and advice was very beneficial to me throughout my work on this project.

And finally, I would like to thank my partner, friends, and family for their continued support throughout this project and their assistance in the evaluation phase.

Table of Contents

Abstract.....	i
Table of Figures.....	vii
Table of Tables.....	viii
Glossary.....	ix
1. Introduction	1
1.1. Background.....	1
1.2. Project Aims & Objectives	2
1.3. Target User	2
1.4. Document Structure	3
1.4.1. Research.....	3
1.4.2. System Design.....	3
1.4.3. Project Management	3
1.4.4. System Development.....	3
1.4.5. System Validation	3
1.4.6. Conclusion.....	3
2. Research.....	4
2.1. Research of Existing Solutions	4
2.1.1. “Classic Style” GoTo Telescopes	4
2.1.2. Modern “Smart” Telescopes.....	6
2.1.3. Amateur GoTo Telescope	7
2.1.4. Research of Existing Solutions Conclusion – Design Principles.....	8
2.2. Technology Research.....	9
2.2.1. Mobile Development Platform	9
2.2.2. Telescope Mount Computer	9
2.2.3. Wi-Fi vs Bluetooth.....	9
2.2.4. Astronomy Engine Library.....	10
2.2.5. Motors Types	10
2.2.6. Serialisation Format.....	10
2.2.7. ASCOM & INDI Libraries.....	11
2.3. Domain Research.....	12
2.3.1. Telescope Mount	12
2.3.2. Calculating the Telescope Direction	12
2.4. Conclusions.....	13
3. System Design	14
3.1. System Overview	14

3.1.1.	System Components	14
3.1.2.	System Features.....	15
3.2.	Mobile Application	16
3.2.1.	User Interface	16
3.2.2.	Automatic Object Slewing and Tracking Feature.....	18
3.2.3.	Calibration Feature	19
3.2.4.	Calibration Adjustment Feature	20
3.2.5.	Manual Slew Control Feature	20
3.3.	Telescope Mount.....	21
3.3.1.	Overview	21
3.3.2.	Arduino Software.....	23
3.4.	Serial Protocol.....	24
3.4.1.	Slew	24
3.4.2.	Set	25
3.4.3.	Manual	25
4.	Project Management	26
4.1.	Scrum Agile (Software Methodology)	26
4.1.1.	Kanban Board Version 1	26
4.1.2.	Kanban Board Version 2	28
4.2.	Issues and Risks	29
4.3.	Source Control	29
5.	System Development	30
5.1.	Prototype Mount.....	30
5.2.	Bluetooth Communication	31
5.3.	Manual Control.....	32
5.4.	Coordinate System Implementation on Mount	33
5.5.	Creating Final Mount Design	34
5.5.1.	Frame	34
5.5.2.	Dual Servo Motors	38
5.5.3.	Increasing Stepper Motor Power.....	40
5.5.4.	Power Source	42
5.5.5.	Providing Documentation for Mount Design.....	43
5.6.	Separating Application Functionality into Multiple Screens	44
5.7.	Connect Activity.....	45
5.8.	Main Activity.....	45
5.9.	Object Select Activity.....	46

5.9.1.	Star List	46
5.9.2.	Planet List.....	47
5.9.3.	Slewing & Tracking.....	47
5.9.4.	Cancelling Slew	48
5.9.5.	Calculating What Objects Are Currently Visible	48
5.10.	Calibrate Activity	49
5.11.	Viewing Apparatus.....	50
6.	System Validation	51
6.1.	Unit Testing.....	51
6.1.1.	Mount Testing.....	52
6.1.2.	Mobile Application Tests	56
6.1.3.	Full System Tests.....	57
6.2.	User Evaluation.....	60
6.3.	Conclusion.....	61
6.4.	Review of Project Objectives	61
6.4.1.	Objective 1	61
6.4.2.	Objective 2	62
6.4.3.	Objective 3	62
6.4.4.	Objective 4	63
6.4.5.	Objective 5	63
6.5.	Future Work.....	64
6.5.1.	Mount Upgrade.....	64
6.5.2.	Satellite Tracking.....	66
6.5.3.	Google Voice & Red Light Filter	66
6.5.4.	Slew Tweaking	67
6.6.	Personal Reflection.....	67
6.7.	Resources.....	67
	Bibliography	68
	Appendix	71
	System evaluation data acquired through a Google Form	71

Table of Figures

Figure 1: Meade ETX-70 (Flagship model in 2000)(7)	4
Figure 2: Meade ETX-90 (Flagship Model in 2023)(5).....	4
Figure 3: Meade Autostar Primary Menus and Options (8).....	5
Figure 4: Stellina Smart Telescope (9).....	6
Figure 5: "GoTo_Telescope_Mount" Hardware (14).....	7
Figure 6: "GoTo_Telescope_Mount" Schematic (14)	7
Figure 7: Design Principles	8
Figure 8: The INDI Protocol (24).....	11
Figure 9: Alt-Azimuth & Equatorial Mount (27).....	12
Figure 10: The Equatorial Coordinate System (28)	13
Figure 11: The Horizontal Coordinate System (28).....	13
Figure 12: System Components	14
Figure 13: Example Interaction Between System Components	14
Figure 14: System Features.....	15
Figure 15: User Journey	17
Figure 16: Object Slewing and Tracking.....	18
Figure 17: Telescope Mount Hardware	21
Figure 18: The Kanban Board Version 1.....	26
Figure 19: A Sprint Ticket	27
Figure 20: Kanban Board Version 2.....	28
Figure 21: Prototype Mount	30
Figure 22: Android Kotlin function which creates a JSON object and sends it over Bluetooth.....	31
Figure 23: Arduino C++ function which parses a JSON object after it has been received over Bluetooth.	31
Figure 24: Arduino code for processing single bytes received over Bluetooth.	32
Figure 25: Lazy Susan Piece (Highlighted).....	34
Figure 26: Unmodified stepper motor axle (left) compared to modified stepper motor axle (right) ..	35
Figure 27: Stepper motor connected to Meccano axle.	35
Figure 28: Schematics for custom piece to attach stepper motor.	36
Figure 29: Stepper motor attached with custom acrylic piece.....	36
Figure 30: Schematics for custom piece to attach servo motors.	37
Figure 31: Servo motor attached with custom acrylic piece.	37
Figure 32: Mount side view with upper rotating section highlighted.	38
Figure 33: Mount top view with servo positions highlighted.	39
Figure 34: Left Servo Range	39
Figure 35: Right Servo Range	39
Figure 36: Arduino code for mapping altitude value to left and right servo motors.	39
Figure 37: Arduino code to move servo motors in increments.	40
Figure 38: Stepper Motor Modification (incision highlighted with red circle)	40
Figure 39: Adafruit Shield Connections	41
Figure 40: AC Adapter (Disconnected).....	42
Figure 41: AC Adapter (Connected)	42
Figure 42: Component documentation in GIT "readme"	43
Figure 43: Mount Images in Git Readme.	43
Figure 44: Application Connect Screen.....	44
Figure 45: Application Calibrate Screen.....	44

Figure 46: Application Main Screen	44
Figure 47: Application Object Select Screen - Star Tab.....	45
Figure 48: Application Object Select Screen - Planet Tab	45
Figure 49: Android Function to send calibration data in Bluetooth Service.....	45
Figure 50: Format of JSON file used to store star data.....	46
Figure 51: Python Script used to translate convert exported star data to JSON.....	46
Figure 52: Function to calculate the horizontal coordinates needed to orient to a body.....	47
Figure 53: Android Code to Send Reset Command.	48
Figure 54: Arduino Code for Resetting Mount.....	48
Figure 55: Android function for checking if a body is under to horizon.	48
Figure 56: Holding phone to calibrate mount.....	49
Figure 57: Mount with spotting telescope attached.	50
Figure 58: Mount with ESP32-CAM attached.	50
Figure 59: Mount with Logitech Webcam Attached.....	50
Figure 60: Smartphone with laser attached.	52
Figure 61: Android Test Activity.....	52
Figure 62: Protractor.....	52
Figure 63: Unit Test A-01	56
Figure 64: Results of Unit Test F-02 (laser representing astronomical object highlighted)	58
Figure 65: Results of Unit Test F-02 - Time Waited (Y) = 0 (Laser Representing astronomical object highlighted)	59
Figure 66: Results of Unit Test F-02 - Time Waited (Y) = 30 minutes (Laser Representing astronomical object highlighted)	59
Figure 67: Comparison of plastic and metal geared servo motors. (42)	64
Figure 68: Nema 17 stepper motor. (41)	64
Figure 69: Object Select Screen Satellite Tab	66
Figure 70: Object Select Screen, Satellite Tab (No Internet)	66

Table of Tables

Table 1: User Interface Wireframes.....	16
Table 2: Prototype Mount Planned Hardware Components	21
Table 3: Final Mount Planned Hardware Components.....	22
Table 4: Risks.....	29
Table 5: Manual Bytes.....	32
Table 6: Star Database Fields	46
Table 7: Unit Test M-04 Images	54
Table 8: Unit Test M-05 Images	55
Table 9: Summary of Evaluation Feedback.....	60
Table 10: Final Mount Costs	62
Table 11: Upgraded Mount Costs	65

Glossary

Term	Definition
Astronomical Object	A naturally occurring physical entity that exists beyond the Earth's atmosphere.
Slew	Turn or rotate to point in a new direction (Used in relation to telescopes).
GoTo Telescope	A computerised telescope that can automatically slew to and track astronomical objects.
Horizontal Coordinate System	A coordinate system used to locate objects in the sky relative to an observer on the Earth's surface. Described using two coordinates, Altitude (the vertical direction) and Azimuth (the horizontal direction).

1. Introduction

1.1. Background

It is a great human achievement that anyone can pick up a telescope and have the ability to view the wonders of the cosmos at their fingertips. But there have always been a few inconveniences to this that can act as a barrier for newcomers to astronomy, where do you point the telescope and how do you know what you're looking at? And then when you do find an object, there is the skill required in keeping that object in view as it appears to move through the sky, due to the rotation of the Earth, the movement of the actual object in space, or both.

Since computers were small enough to fit in a telescope, these are the accessibility problems that the engineers of 'GoTo' telescopes have strived to solve. These are telescopes that appeared on the consumer market just prior to the turn of the millennium, and consist of a motorised mount controlled by a computer system. These systems take an object in space selected by the user, the location of the viewer on Earth, and the current time, and with some clever mathematics are able to compute what direction the telescope should point to give the viewer a perfect view of their chosen object. They then continuously run this calculation and update their position, in order to always keep the object in view as it moves across the sky.

The market landscape of these telescopes is extremely varied, all with widely varying price points, levels of functionality and degrees of accessibility.

1.2. Project Aims & Objectives

The goal of this project is to create a system enabling users to transform a basic, entry-level telescope into a feature-rich, accessible GoTo telescope with a low investment of time and money.

The system will consist of two components, a mobile application which the user downloads to their phone, and a motorised telescope mount which the user constructs themselves using their own telescope. The mobile application is then used to control the telescope mount wirelessly.

The mobile application will provide basic features such as the ability to manually control what direction the telescope mount is pointing, and more complex features such as providing a list of astronomical bodies that the user can select for the telescope to slew to and track.

The mount will be able to support a viewing device, receive commands wirelessly and be able to point at any location in the sky.

Therefore, the objectives of this system are as follows:

1. Create an overall exemplary GoTo telescope system providing all of the best features currently on the market at a low price point.
2. Create an accessible mobile application with features of varying complexity for controlling a GoTo telescope.
3. Design and create a motorised telescope mount from low-cost materials that can support a viewing device, receive commands wirelessly and point at any location in the sky.
4. Define a standard communication protocol to allow loosely coupled communication between the mobile application and the telescope mount.
5. Host the project in an open-source manner, so that the public can develop their own and expand upon it.

1.3. Target User

Since computerised telescopes have the ability to bring the wonders of astronomy to someone with little to no knowledge of the night's sky, this is the user that the project will be targeting. While the GoTo telescope will be fully functional and provide value to amateur astronomers, it will be developed with the assumption that the user of the system has no prior knowledge of astronomy.

1.4. Document Structure

1.4.1. Research

This section documents all of the research that was carried out for this project.

It then describes the research of existing GoTo telescope solutions, which influence the definition of the design principles that guide the development of the final system.

Research of the various technologies required to implement the system is documented, analysing the strengths and weaknesses of each.

And finally, the research of domain specific concepts is documented, which were crucial to have an understanding of before commencing the project.

1.4.2. System Design

This section provides the specifics of how the system will be implemented.

1.4.3. Project Management

This section describes how the project as a whole was managed, including what software development methodology was used, and how potential risks were anticipated.

1.4.4. System Development

This section documents the stages the system went through during its development, and any changes that had to be made to the original plan. Code snippets and diagrams are used to illustrate the functionality and development of the system.

1.4.5. System Validation

This section contains the processes followed to test and evaluate the system, and their results.

1.4.6. Conclusion

This section evaluates how well the final product met the project objectives outlined in section 1.

Future work planned for the project is outlined, and a personal reflection on the project process as a whole is included.

2. Research

2.1. Research of Existing Solutions

In this section, various existing GoTo telescope solutions are researched, with their successes and drawbacks analysed. This research is then used to inform the principles that govern the design of the system developed for this project.

2.1.1. “Classic Style” GoTo Telescopes

In the 1990s the first computerised telescopes were introduced to the world, pioneered by the two largest companies in amateur astronomy at the time Meade Instruments(1) and Celestron(2), with their flagship telescopes ETX-EC and Nexstar 5(3) respectively.

Early versions of these telescopes had a computer and database responsible for the tracking contained within, and were controlled with an external LCD controller. A popular example of this type would be the ETX-70 (Figure 1).

As technology improved, the tracking computer and database started to be contained within in the handset itself, such as in the Meade Autostar released in 1999. This handset was compatible with a number of Meade telescopes, and also allowed the user to connect it to a personal computer to upgrade its software and database. (4)

This type of telescope is still being produced today. While the technology used has been improved upon over the years, the overall design of these telescopes has stayed very much the same since their initial inception. Meade’s latest model, the ETX-90 (5) released on 2023, still bears a striking resemblance to the Meade ETX-70 released in 2000, and is still controlled by an LCD Autostar handset computer. (4)

These typically had a serial port, referred to as an AUX port, which allows them to be connected to a PC over an RS-232 serial cable to be controlled with a more user-friendly and feature rich desktop software, such as the proprietary Meade Autostar Suite and the open source Stellarium(6). Official Bluetooth and Wi-Fi adapters can also be purchased, which allow the telescopes to be controlled by more sophisticated mobile applications, such as Sky Safari and Stellarium Mobile. Each telescope communicates to these applications with their own unique serial protocol which each application must implement to be compatible, with the most popular of these protocols being the Meade Serial Protocol.



Figure 1: Meade ETX-70 (Flagship model in 2000)(7)



Figure 2: Meade ETX-90 (Flagship Model in 2023)(5)

2.1.1.1. Benefits of this Design

This design is very functional, allowing the user access to thousands of objects to view.

The fact that it is able to be used entirely without an internet connection or external power source is also a large benefit, as it means that it can be used in remote locations with no problems.

2.1.1.2. Limitations of this Design

The main limitation of this design would be that the user interface is dated and hard to navigate through, especially when compared to modern user interfaces. While this can be improved by purchasing a Bluetooth or Wi-Fi adapter, these extras add extra cost onto the already pricey telescope. These telescopes all having their own proprietary serial communication standard is also another downside, as it means application developers must implement each telescope's own unique protocol.

In order to set the telescope up so that it knows where it is, these telescopes typically employ a calibration process, involving pointing the telescope at three stars in the sky. This can prove a challenge in built up areas where there isn't a direct line of sight to the star.

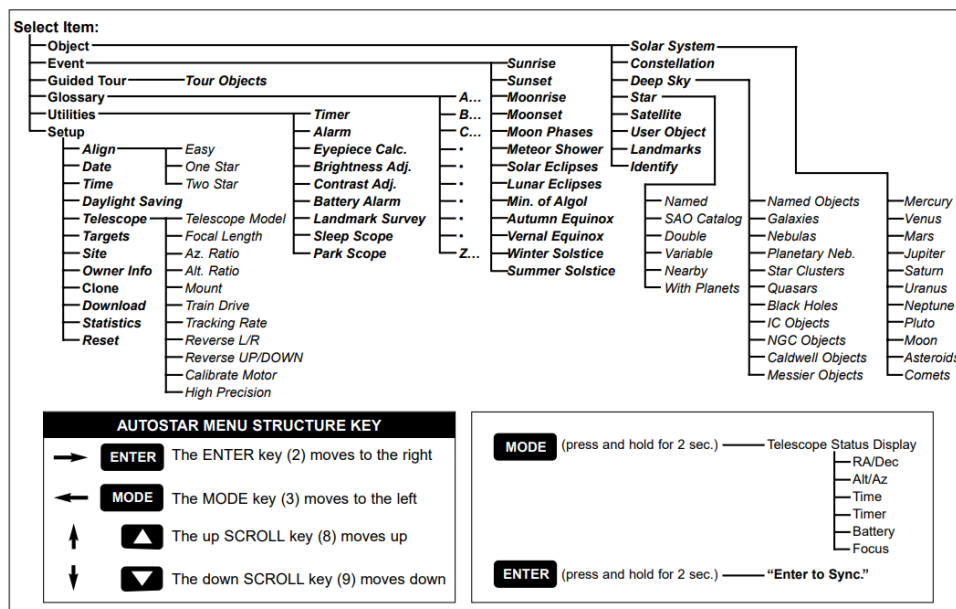


Figure 3: Meade Autostar Primary Menu and Options (8)

2.1.2. Modern “Smart” Telescopes

The latest innovation in GoTo telescope technology is the “Smart” telescope, which are telescopes that integrate with smart devices such as smartphones, to provide a smoother user experience.

An example of a smart telescope is Stellina. This telescope acts as a Wi-Fi point that allows a mobile device to connect to it. When connected, the user can control the device through a smartphone application, allowing the user to point and track the telescope at an object. There is no viewport on the device, and instead all viewing is done through a live feed on the phone. The telescope also has astrophotography features which allow the user to take photographs of the night sky. (9)



Figure 4: Stellina Smart Telescope (9)

2.1.2.1. Benefits of this Design

These telescopes can be very easy to use, utilising the dynamic large touchscreens of smart devices to provide clear and dynamic user interfaces.

They typically provide a lot of functionality, such as being able to view satellites as they move across the sky.

2.1.2.2. Limitations of this Design

A lot of these smart telescopes are expensive, with Stellina being sold at €3,999.00 (10).

These telescopes work by acting as a Wi-Fi point allowing a phone to connect to it. This drains more power than if it was connected with Bluetooth. (11) Connecting to a Wi-Fi point with a phone also means that the phone will be disconnected from whatever network it is currently connected to, inconveniencing the user.

The applications that these telescopes use are almost always closed source and can only be used with telescopes manufactured by the owner of said app, such is the case with Celestron's SkyPortal application (12), and Singularity by Vaonis (Manufacturer of Stellina) (13).

2.1.3. Amateur GoTo Telescope

'GoTo_Telescope_Mount' created by Matthew Leung (14), is an impressive amateur GoTo telescope where the designer has created their own telescope mount, which powered by Arduinos and a Raspberry Pi. The telescope has an onboard database of astronomical objects to slew to. The calculations to find where the telescope should point are carried out on the Raspberry Pi, using the Python library Astropy(15). Two motors are controlled with the Raspberry Pi and three Arduinos to read data from Optical Rotary sensors and an accelerometer which are used to keep track of the telescopes position.



Figure 5: "GoTo_Telescope_Mount" Hardware (14)

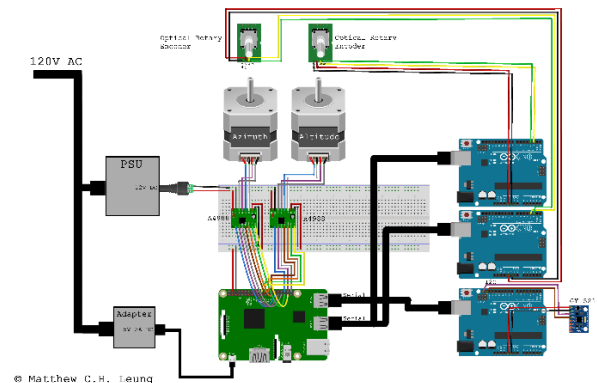


Figure 6: "GoTo_Telescope_Mount" Schematic (14)

2.1.3.1. Benefits of this Design

It is functional, being able to slew and track astronomical objects.

The design works entirely offline, having a database of astronomical objects stored on the device.

Accelerometers are used to calibrate the telescope.

The code and schematics are open sourced and uploaded on GitHub, allowing others the ability to recreate and expand the project.

2.1.3.2. Limitations of this Design

This design isn't portable, as it needs to be plugged into a power socket to be used.

It is unclear why the design needs the three Arduinos, as everything should be possible to be done on the Raspberry Pi. This adds unnecessary complexity, hardware cost and power usage.

2.1.4. Research of Existing Solutions Conclusion – Design Principles

From the research carried out, six principles were derived which when incorporated into a system will produce an exemplary GoTo telescope system.

The system will consist of a custom GoTo telescope mount controlled with a custom application on a smart device.

2.1.4.1. Easy to Use

This was a primary motivation for the project, making the act of astronomy easy and accessible for newcomers.

As seen mainly with the classic style, many GoTo telescopes have long and unintuitive menus, complicated setup calibration processes, and dated user interfaces.

To solve this problem, this system will be designed with ease-of-use and user-friendliness as core principles influencing all design choices.

2.1.4.2. Functional

The system will be feature-rich and functional, providing use and value to newcomers and seasoned astronomers alike.

2.1.4.3. Cost Efficient

Since the system will be designed with newcomers to astronomy as a prominent user, an effort will be made to keep the components as cost efficient as possible.

2.1.4.4. Offline Capable

As a common use case of telescopes is viewing in remote areas with limited or no internet connectivity, a feature-rich experience will be provided even if the user has no access to the internet.

2.1.4.5. Power Efficient

Due to telescopes commonly being used in remote areas with restricted access to electricity, the system will be designed to be able to function for long periods of time without having to be recharged.

2.1.4.6. Modular & Open-Source Architecture

The software will be designed in a modular and open-source fashion, so that other users can build upon and extend the project if they wish.

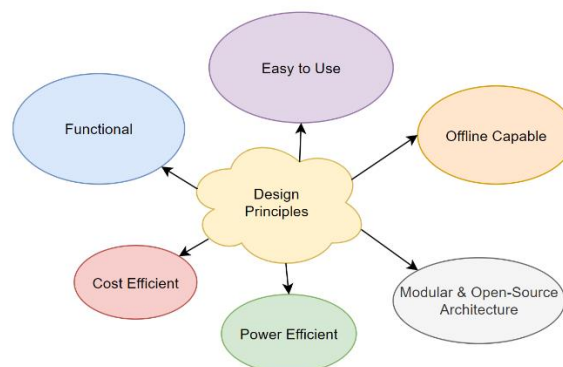


Figure 7: Design Principles

2.2. Technology Research

In this section, the research carried out of the specific technologies needed to realise a design that incorporated the design principles defined in section 2.1.4 is outlined here.

2.2.1. Mobile Development Platform

Since the mobile application will be making heavy use of the device sensors, it was decided against going with a cross development environment such as Swift or Flutter, and instead the Android development environment was chosen, as developing in a native environment allows better access to the device's hardware. The application will be developed in the Kotlin programming language, as this is now the primary language for Android application development.(16)

2.2.2. Telescope Mount Computer

To create the computerised motor telescope mount, a small on-board computer will be needed to receive commands from the smart device and move the motors. There are a number of potential solutions available for this.

A Raspberry Pi is a small single-board computer, capable of running lightweight operating systems such as Linux, and there are a few existing pieces of software that can allow the Raspberry Pi to be used as a smart telescope, such as StellarMate OS(17) and Astroberry Server(18). These work by turning the Raspberry Pi into a server machine, which the phone then connects to as a client and issues commands.

This would not suit the design, as it would violate the energy efficiency principle defined in section 2.1.4. Using a computer powerful enough to run a fully-fledged Linux operating system would be unnecessary for the system's use case, as the smart device running the application will already be a powerful computer in the system that can be used for running any intensive computing.

Instead, an Arduino Uno was chosen to control the mount. This is a microcontroller board, and although it has less computational power than the Raspberry Pi, it is completely sufficient for the system's needs. Arduino Uno boards have been proven to use three times less energy than Raspberry Pis. (19)

2.2.3. Wi-Fi vs Bluetooth

Bluetooth and Wi-Fi are the most popular methods of sending data wirelessly from a mobile device. Bluetooth is oriented towards connecting close devices, while Wi-Fi is oriented towards computer-to-computer connections. (20).

In regard to usability and convenience, with Wi-Fi, the telescope mount would have to act as a Wi-Fi connection point. For a phone to connect to and control the mount, they would have to connect to this Wi-Fi point. This would disconnect the user from the network they are currently connected to, inconveniencing them. With Bluetooth connections, users are already familiar with the process of connecting to Bluetooth devices, with the current extreme popularity of Bluetooth earbuds and speakers.

If a Wi-Fi telescope wants to receive live data from the internet, such as positions of satellites, etc. it must be connected to another Wi-Fi point itself. If this telescope was used in an area with no Wi-Fi connection, it wouldn't be able to access this data. If the telescope connected over Bluetooth, however, the user would still be able to be connected to the internet and would be able to receive these positions over mobile data.

From a power efficiency point of view Bluetooth has been proven to be more energy efficient than Wi-Fi. (11)

For Bluetooth data to be received by the telescope mount, the popular HC-05 Bluetooth module will be used. (21)

2.2.4. Astronomy Engine Library

In order to calculate the orientation that the telescope should point, described in the later section 2.3.2, and to calculate the current position of planets, the astronomy library Astronomy Engine (22) will be used, which has support for the Kotlin programming language.

2.2.5. Motors Types

To move the telescope in the mount, motors are needed.

Stepper motors are a common type of motor can rotate accurately around a full 360° of rotation, by specifying the number of “steps” for it to be rotated.

Servo motors are a common type of motor motors that can rotate 180°, and the exact angle of rotation for them to rotate to can be specified.

Since the telescope mount will need to rotate 360° horizontally, a stepper motor will be used for the base, and servo motors will be used for the vertical control, as the most the telescope will have to move in that direction is 90°.

2.2.6. Serialisation Format

To send data from the mobile device to the Arduino, the use of a serialisation format would mean that the data would be sent in a language independent, human readable fashion, adhering to the modularity principle defined in section 2.1.4.

The two most popular serialisation formats are XML and JSON.

JSON will be used, as XML has a much larger serialised size and is considerable slower to serialize and deserialize compared to JSON. (23)

2.2.7. ASCOM & INDI Libraries

ASCOM and INDI are source software libraries designed to aid in the development of software which communicates with astronomical equipment over a computer, with ASCOM focusing Windows devices and INDI focusing on support for Linux and macOS devices.

They were created to solve the issue of software being written to accommodate specific devices, and act as middleware, decoupling the software from the hardware. (24,25)

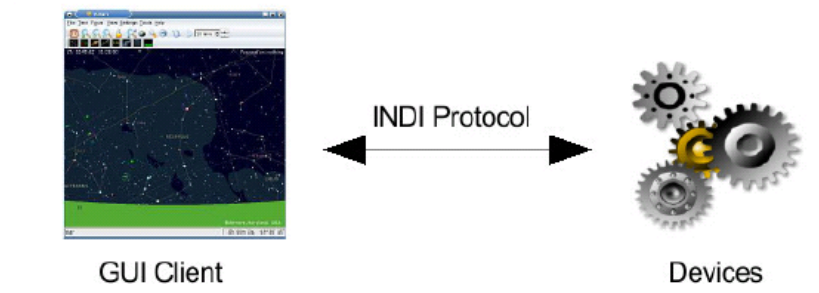


Figure 8: The INDI Protocol (24)

Since the majority of the system's software will be running on an Android device, software from the library won't be used or implemented, but inspiration will be taken from its concepts of open source and loose coupling between software and hardware when designing the communication protocol for Bluetooth transmission.

2.3. Domain Research

2.3.1. Telescope Mount

There are two main types of telescope mounts, Alt-Azimuth and Equatorial.

Alt-azimuth mounts are simple two-axis mounts that rotate the telescope on two perpendicular axes, vertical and horizontal. They are very popular for their lightweight, compact, designs and ease of use. (26)

An equatorial mount is a more complex mount that compensates for the Earth's rotation by having a polar (rotation) axis parallel to the Earth's rotational axis. It is designed to make the tracking by hand of celestial objects easier as a user does not have to account for the Earth's rotation.

Since the mount in this project will be motorised, how easy it is for a user to move by hand is not a concern, so it was decided that Alt-Azimuth mount will be created.



Figure 9: Alt-Azimuth & Equatorial Mount (27)

2.3.2. Calculating the Telescope Direction

The calculations for working out where the telescope should point requires some knowledge of various coordinate systems.

2.3.2.1. The Equatorial Coordinate System

The Equatorial Coordinate System is the standard system used to describe the position of a celestial object in relation to the Earth. The system is a geocentric coordinate system, meaning that the centre of the Earth is used as the origin.(28)

The system uses two coordinates to define a point, Right Ascension (abbreviated as RA and referred to mathematically as α), and Declination (abbreviated as Dec and referred to mathematically as δ).

RA determines the points horizontal position. It is measured in hours, minutes and seconds of time, and it is determined by how much time it would take for the Earth to rotate towards the point from a starting position known as the Vernal Equinox.

Declination determines the points vertical position. It is measured in degrees, minutes and seconds, with 60 minutes in each degree and 60 seconds in each minute. The celestial equator is 0 degrees declination, and the north and south celestial poles are +90 and -90 degrees.

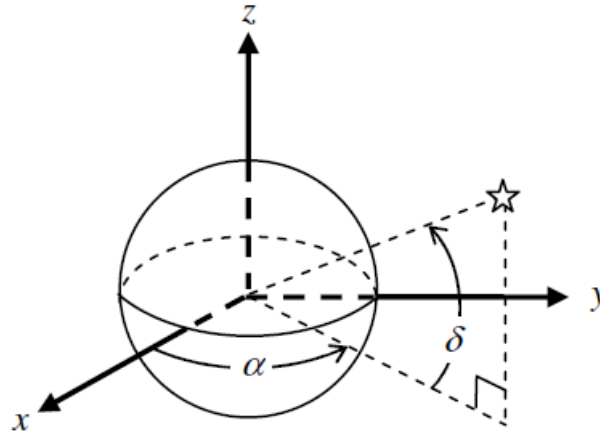


Figure 10: The Equatorial Coordinate System (28)

2.3.2.2. The Horizontal Coordinate System

The Horizontal Coordinate System is the standard system used to specify the position of a celestial object in relation to an observer located on the Earth. It is a topocentric coordinate system, meaning that the origin point is at a point on the Earth's surface. The system uses two coordinates to specify the direction of an object: altitude and azimuth. (29)

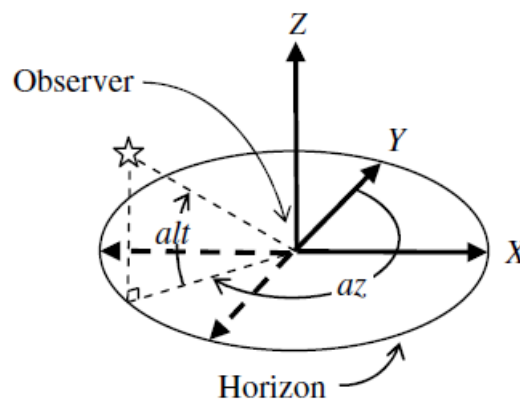


Figure 11: The Horizontal Coordinate System (28)

2.3.2.3. Calculating the Telescope Direction

The equatorial coordinates are taken of the astronomical object that will be slewed to (right ascension and declination).

These coordinates are then converted into a horizontal coordinate system (altitude and azimuth), with the viewer's location at the centre of the system.(28)

2.4. Conclusions

Informed by the research in this section, the technical implementation of the design will consist of an Android application written in Kotlin, that will control a motorised telescope mount by sending JSON data over Bluetooth. The Android application will compute the telescope's direction using the Astronomy Engine library. The telescope mount will be of Alt-Azimuth design, and will consist of an Arduino Uno board, a stepper motor, a servo motor, and a HC-05 Bluetooth module.

3. System Design

3.1. System Overview

3.1.1. System Components

The system will consist of two components: a telescope mount and a mobile application.

The telescope mount will be a structure capable of holding a small sized telescope. It will consist of two motors, one for controlling the mount's azimuth position, and one for controlling the scope's azimuth position. The motors will be controlled by an Arduino Uno board. The Arduino will accept instructions received with a Bluetooth module, and react accordingly to commands. The mount will be powered by an on-board battery pack.

The mobile application will be a user-friendly Android application that allows the user to control the telescope in a number of ways. The application will provide the user with a number of objects that the user can select to have the telescope point at. The application will contain a local database of the positions and locations of planets and stars.

In order to have all of the intensive work carried out on the smart device, the Android application will perform the calculations necessary to produce the altitude and azimuth coordinates of the direction that the telescope should point. The job of the Android program will be to receive these coordinates over Bluetooth and point the telescope in the given direction.

The system will be designed so that the mobile application and the telescope mount software are as loosely coupled as possible. A standard communication protocol will be defined that they will both implement. This will make the system modular, meaning that an entirely new telescope mount could theoretically be developed in the future, and once it implemented the same communication protocol the mobile application could remain unchanged. This would also work the other way, as another application to control the telescope mount could be developed in the future, and once it sent the same commands over Bluetooth, the telescope mount could remain unchanged. This will also bring the short-term benefits, as if there are any unforeseen circumstances that require a drastic hardware change throughout the course of the project, only the Arduino software would have to be modified and not the mobile application.

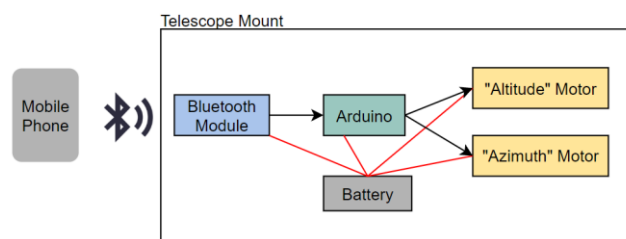


Figure 12: System Components

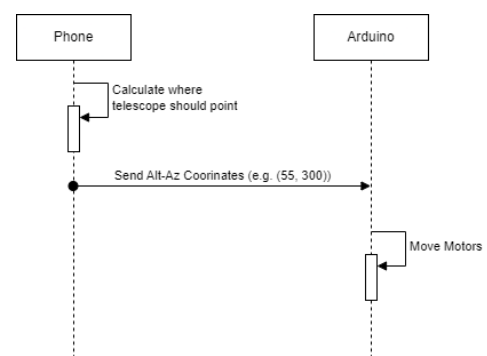


Figure 13: Example Interaction Between System Components

3.1.2. System Features

3.1.2.1. Automatic Object Slewing and Tracking

The main feature of this system is that it will provide a number of astronomical objects that can be viewed in the sky, that when selected, will move the telescope so that the object is in the telescopes view. As this object moves through the sky, the telescope will continually update its position, so that the object is always in view.

Two types of astronomical objects will be offered to the user to view: stars and planets.

The positional data of the stars and planets available to view (the Earth's moon will be classified as a planet in this case) will be stored in a locally on the application, and will be available to view even if the device doesn't have an internet connection.

3.1.2.2. Telescope Calibration

In order for the telescope to learn what direction it is pointing when it is first powered on, the application will have a "calibration" functionality. This will work with the user placing the mobile device on a set spot on the telescope mount. The user will press the "calibrate" button and the application will use the phone's internal compass to find what direction the telescope is facing in the azimuth direction (horizontal axis), and use the phone's internal accelerometer sensor to determine what direction the telescope is pointing in the altitude direction (vertical axis). This data will be transmitted over Bluetooth to the telescope mount, which will then update in its internal memory what direction the telescope is facing.

3.1.2.3. Manual Slew Control

The application will also offer the ability to "manually" control the telescope. The user will be presented with a keypad offering left, right, up and down controls. As long as one of these buttons is held down, the telescope will slew in that direction at a set pace.

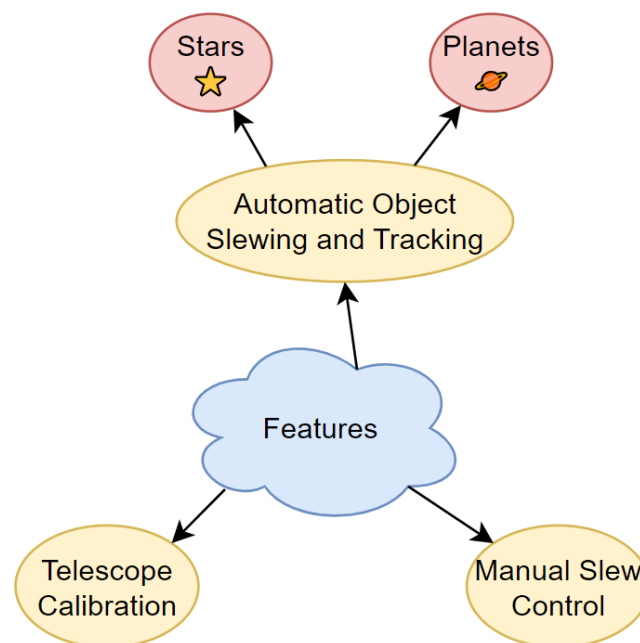


Figure 14: System Features

3.2. Mobile Application

3.2.1. User Interface



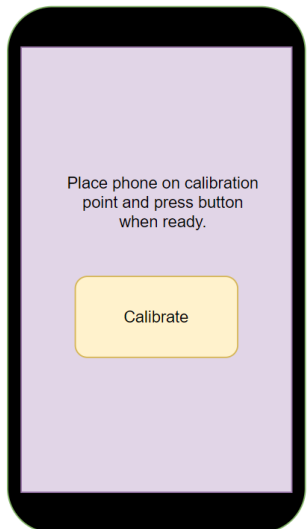
The user interface was designed adhering to Nielsen's usability heuristics.(30)

The entire interface has been designed principle of Aesthetic and Minimalist Design, with each screen having one purpose, and with as little clutter on the screen as possible.

The buttons on the main screen are designed to look like the buttons on a physical GoTo handset, following Nielsen's principle of a match between the system and the real world.

Nielsen's principle of Visibility of System status is followed, as the application displays when it is currently tracking an object. This also follows the principle of Recognition Rather than Recall, as the user doesn't have to remember the object that is currently being tracked.

Table 1: User Interface Wireframes

Screen 1: Select Telescope		
Screen 2: Calibration		

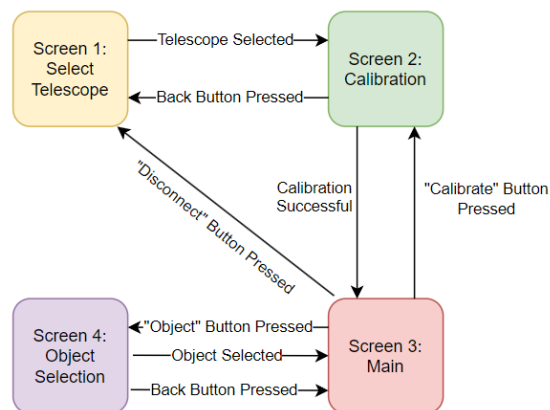
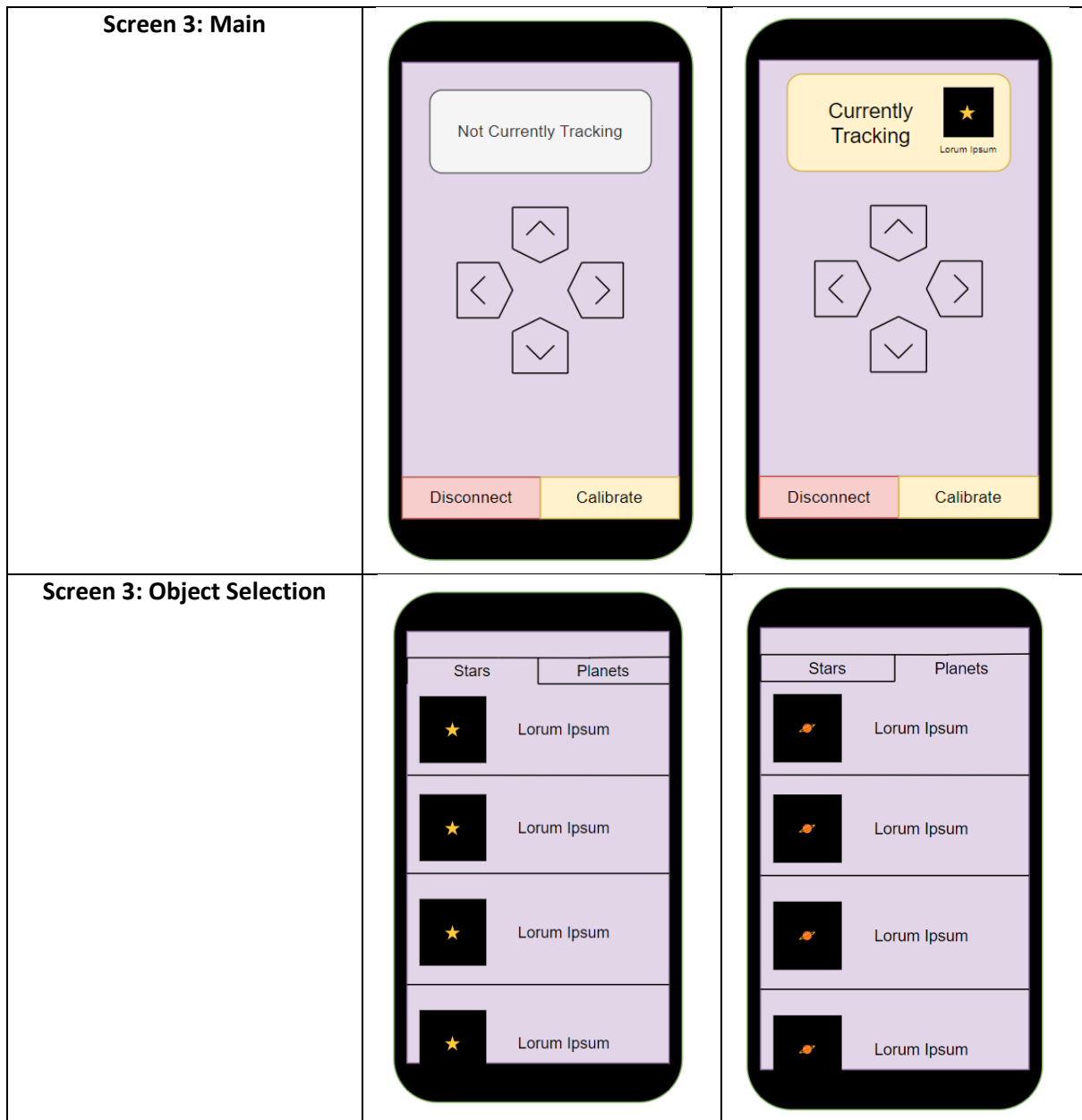


Figure 15: User Journey

3.2.2. Automatic Object Slewing and Tracking Feature

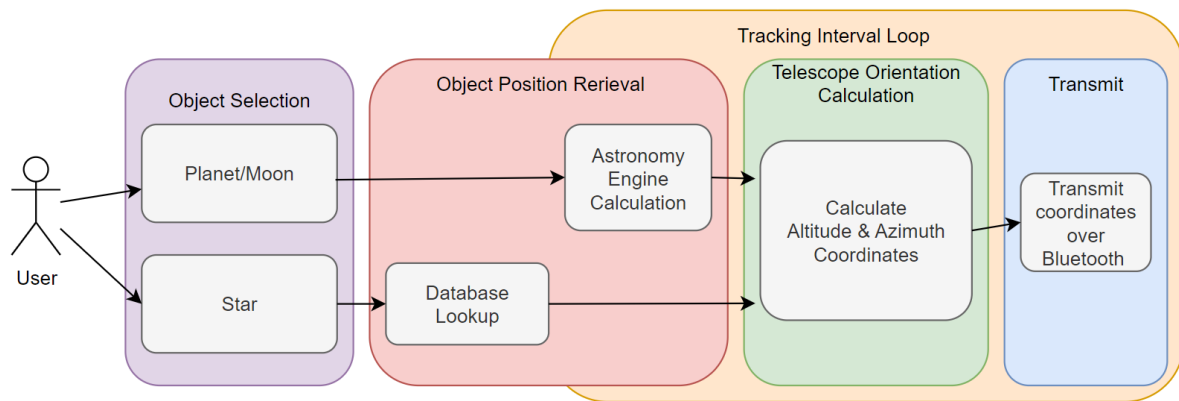


Figure 16: Object Slewing and Tracking

The object slewing feature can be broken down into four phases: Object Selection, Object Position Calculation, Telescope Orientation Calculation and Transmit. Some of the actions within these phases will then have to be repeated at a set interval in order for the telescope to track the object as it moves across the sky.

3.2.2.1. Object Selection Phase

In the Object Selection phase takes an object that the user wishes to view as an input. The objects are divided into two categories: Star and Planet.

3.2.2.2. Object Position Calculation

In this phase, the position in space of the object is computed, in the form of Equatorial Coordinates.

Due to the distance of stars from the Earth, their individual movement would be far too minor to factor into the calculation, so these coordinates can be stored and retrieved from a local database within the application.

Due to the fact that a planets position is constantly changing as it rotates around the Sun, the Astronomy Engine (22) library will be used to find the position of these, as it provides the functionality to calculate the position of all of the planets in the solar system and the Earth's moon for any given time.

3.2.2.3. Telescope Orientation Calculation

In this phase, the horizontal coordinates (Altitude and Azimuth) that the telescope should point at are calculated. This calculation takes the following inputs:

- **Current time:** This will be taken from the mobile devices clock.
- **Location of Telescope:** The location of the telescope in terms of Geographic coordinates (longitude and latitude). These are obtained from the mobile devices GPS in the calibration stage.
- **Location of Object:** The objects position in the sky, in terms of Equatorial coordinates (right ascension and declination), obtained in the Object Selection stage of the software.

The Astronomy Engine Library (22) will be used to take these inputs and produce accurate horizontal coordinates.

3.2.2.4. Transmit

In the transmission phase, the horizontal coordinates obtained will be converted to JSON format, and transmitted over Bluetooth for the Arduino to receive.

3.2.2.5. Track

In order to then track the object continuously as it appears to move through the sky, a number of these functions will have to be run continuously at a set interval.

For the object position calculation phase, the positions of the planets will be changing as they move through space, so continuous planetary position calculations will have to be carried out. The position of the stars won't be moving at a pace noticeable from Earth, so the position of these only have to be looked up once.

For the telescope orientation phase, due to the constant rotation and orbit of the Earth, and the possibility that the astronomical object's coordinates have changes, the direction that the telescope should be pointing has to be calculated continuously at a set interval.

Since a new telescope orientation is being computed at every interval, this new position will have to be retransmitted over Bluetooth so that the telescope mount can update its position.

3.2.3. Calibration Feature

For the telescope to know what direction it is initially pointing, the application will have a "calibrate scope" option.

The phone will be placed on the mount, parallel to the scope. When the calibrate option is selected, the application will use the mobile device's compass to find the telescope's offset from north, and will convert this to an azimuth coordinate. It will use the accelerometer of the mobile device to find out how much the phone is rotated in the Y axis, and convert this to an altitude coordinate. These coordinates will be transmitted to the Arduino let it know that these are the coordinates the scope is currently pointing at. The Arduino will update the values of the scope's current orientation in its memory to these new values.

When this calibration option is selected, the current longitude and latitude coordinates will be obtained from the mobile device's GPS, and saved to the mobile device for use in the object slewing and tracking feature. Since the telescopes longitude and latitude position on the Earth will not be changing during a viewing session, it is best that these values are only obtained once, during this calibration process, to reduce energy consumption.

The phone only needs to be placed on the mount for the calibration process, and can be taken off when the process is over.

3.2.4. Calibration Adjustment Feature

The option to enter this mode only becomes available if the mount is actively tracking an object. If the object does not appear exactly in the telescope's viewport, the user can use directional up, down, left, and right buttons to make slight adjustments to where the telescope is pointing. The user can adjust the telescope until the object currently being slewed to is directly in the telescope's viewport. When the telescope is being moved, the internal values of altitude and azimuth won't be altered, as the physical position of the telescope is being manually adjusted by the user to hopefully match the values that the Arduino thinks the telescope is pointing in.

3.2.5. Manual Slew Control Feature

This feature will provide four buttons: Left, Right, Up and Down. While the user holds one of these buttons down, a corresponding signal will be continuously broadcasted over Bluetooth for the Arduino to receive and react accordingly. Left and Right will adjust the Azimuth position of the telescope, and Up and Down will adjust the altitude position of the telescope.

3.3. Telescope Mount

3.3.1. Overview

The telescope mount will be a structure capable of holding a viewing device. Its frame will be constructed out of the metal construction system Meccano(31). It will be entirely self-contained, not needing to be plugged into an external power source. It will have a bracket on it for placing the smart device on for the calibration feature (Section 3.2.3).

The electronics will consist of two motors, one for aiming the telescope horizontally (it's azimuth), and one for aiming the telescope vertically (it's altitude). The motors will be controlled by an Arduino Uno. The Arduino will accept instructions received on a Bluetooth module, and react accordingly to commands. All of the components on the telescope mount will be powered by an on-board battery pack.

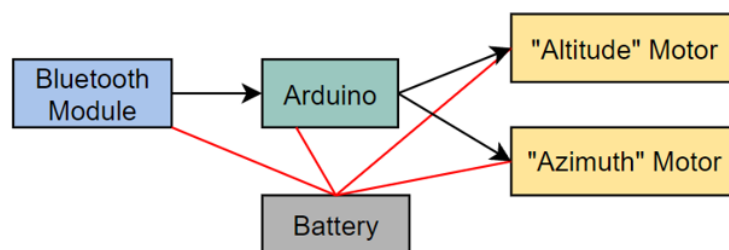


Figure 17: Telescope Mount Hardware

Table 2: Prototype Mount Planned Hardware Components

Name	Qty.	Description	Usage	Approx. Cost
28BYJ-48 5V Stepper Motor	1	Stepper Motor with 360° range.	Moving the telescope horizontally.	€4
SG90 Digital Servo	1	Small servo motor with 180° range.	Moving prototype telescope vertically.	€1
HC-05 Bluetooth Module	1	Allows the transmission and receipt of wireless serial data.	Allow the mount to receive Bluetooth commands.	€5
Arduino Uno Rev 3 AVR Development Board	1	Open-Source Microcontroller Board	Programmed to process serial commands and move motors accordingly.	€25
				€35

Table 3: Final Mount Planned Hardware Components

Name	Qty.	Description	Usage	Approx. Cost
28BYJ-48 5V Stepper Motor	1	Motor with 360° range.	Moving the telescope horizontally.	€4
Graupner C577 Servo	2	Motor with 180° range.	Moving the telescope vertically.	€10
HC-05 Bluetooth Module	1	Allows the transmission and receival of wireless serial data.	Allow the mount to receive Bluetooth commands.	€5
Arduino Uno Rev 3 AVR Development Board	1	Open-Source Microcontroller Board	Programmed to process serial commands and move motors accordingly.	€25
Miscellaneous Meccano Parts	N/A	Metal construction systems used to build small mechanical devices.	Constructing the mount mechanism.	€15
Total				€64

3.3.2. Arduino Software

The Arduino software will receive signals from the Bluetooth module, process them, and move the altitude and/or azimuth motors accordingly. It will keep track of Azimuth motors position with an Azimuth variable ranging from 0 – 359, with 0 meaning the telescope is pointing north. The Altitude motors position will be kept track of with an Altitude variable ranging from 0 – 90, with 0 meaning the telescope is horizontal with the ground, and 90 meaning the telescope is perpendicular to the ground.

3.3.2.1. Calibration Data

When the Arduino receives calibration data, it will update the Azimuth and Altitude values it has in memory to the new values sent without moving the motors.

3.3.2.2. Manual Control Data

When the Arduino receives a manual control signal, it will update the position of the corresponding motor by a small increment and update the corresponding variable. The Arduino will likely receive a large number of these signals at once from the user holding their finger on the button.

3.3.2.3. Slew to Coordinates Data

The Arduino will move the motors to point the telescope at these new coordinates and update the internal values of the variables.

3.4. Serial Protocol

A unique communication protocol was designed to facilitate communication between the application and the telescope mount. All commands will be sent from the mobile application.

A command sent from the mobile application will take the following form:

```
{COMMAND: COMMAND_DATA}
```

There are three types of commands that can be sent from the application: Slew, Set and Manual.

3.4.1. Slew

This command will be used to tell the mount to point in a specific direction, in terms of azimuth and altitude.

It will take the following form, where AZIMUTH_VALUE is a float value between 0 and 359, and ALTITUDE_VALUE is a float value between 0 and 90:

```
{
  "Instruction": "Slew",
  "Data": {
    "Azimuth": AZIMUTH_VALUE,
    "Altitude": ALTITUDE_VALUE
  }
}
```

An example of this command:

```
{
  "Instruction": "Slew",
  "Data": {
    "Azimuth": 150.5,
    "Altitude": 50.1
  }
}
```

3.4.2. Set

This command will be used to tell the mount what orientation the telescope is currently in, and will be used for the initial calibration and the calibration tweak features.

It will take the following form, where AZIMUTH_VALUE is a float value between 0 and 359 inclusive, and ALTITUDE_VALUE is a float value between 0 and 90 inclusive:

```
{
  "Instruction": "Set",
  "Data": {
    "Azimuth": AZIMUTH_VALUE,
    "Altitude": ALTITUDE_VALUE
  }
}
```

An example of this command:

```
{
  "Instruction": "Set",
  "Data": {
    "Azimuth": 50.2,
    "Altitude": 85.3
  }
}
```

3.4.3. Manual

This command will be used to tell the mount to move slightly in one of four directions, and will be used for the manual control feature.

It will take the following form, where DIRECTION_VALUE will either be the string "Left", "Right", "Up" or "Down".

```
{
  "Instruction": "Manual",
  "Manual": {
    "Direction": DIRECTION_VALUE
  }
}
```

An example of this command:

```
{
  "Instruction": "Manual",
  "Manual": {
    "Direction": "Right"
  }
}
```

4. Project Management

4.1. Scrum Agile (Software Methodology)

To manage this project, a variation of the Scrum Agile Methodology was be used, modified for use in a solo project.

The Scrum methodology is an iterative methodology that repeatedly follows three main stages: planning and estimation, implementation, and review and retrospective. (32)

This suited the project well, as firstly, due its experimental nature, a methodology with a focus on incremental development with frequent testing was a necessity.

If the project started to go down an avenue that wasn't working out, it was detected in the review and retrospective phase, and a new approach was devised in the sprint planning phase.

James, the project developer, also had a lot of real-world experience in developing under the Scum Agile Methodology, having worked under it during an internship.

Project tracking software is typically used in industry environments following this methodology. To manage this own project, it was decided to use the web-based, kanban-style list making application Trello. While this software doesn't have all of the features of industry standard project management software, such as Jira, it is well suited for use in managing a final year project.

4.1.1. Kanban Board Version 1

The project was first broken down into small chunks, with a "ticket" being created for each of these on the Kanban board.

Three 'lists' were created to sort these tickets:

- **Backlog:** The tickets that are yet to be worked on.
- **In progress:** The tickets that are currently being worked on.
- **Completed:** The tickets that have been completed.

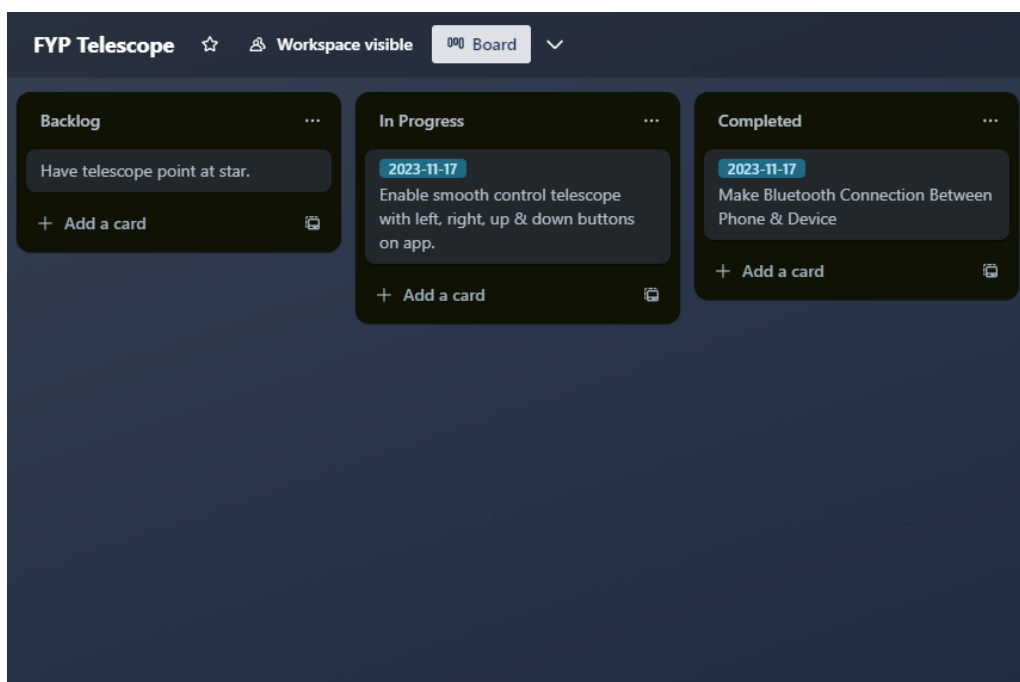


Figure 18: The Kanban Board Version 1

At the beginning, all tickets were assigned to the Backlog list. The Scrum methodology breaks project development down into "sprints", consisting of a set period of time in which a number of agreed-upon tasks must be completed. Typically, in industry, this period of time would be 2-3 weeks. Due to the comparably smaller scale of the project, it was decided to have each sprint be one week in duration. This allowed more frequent reviews of progress to ensure the project was staying on track. This chosen time frame also coincided well with the weekly logs that had to be submitted during the course of the project, and the weekly mentor meetings, as the sprints were used as a measure of progress made.

At the start of each sprint, the sprint planning phase began, where the tickets were chosen that would be taken into the sprint. Utilising the label feature in Trello, a label containing the end date of the current sprint was applied to all tickets that would be worked on.

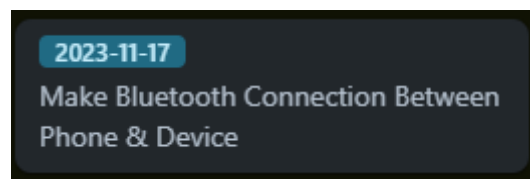


Figure 19: A Sprint Ticket

The implementation phase then began by starting the sprint. During each sprint, as many tickets as possible that were committed to were brought "over the line". New tickets were created for any features or bugs that were found that needed to be worked on and added to the backlog. If any of these newly created tickets were deemed critical to proceeding, these were pulled into the current sprint.

When a sprint finished, the Review and Planning Phase began. Tests were performed on the current iteration of the project and new tickets were created for any bugs or lacking features discovered. How well the previous sprint worked was also reviewed, and this reflection was used to improve the following sprint.

4.1.2. Kanban Board Version 2

As the project progressed, it was realised that it would be helpful to incorporate tracking thesis writeup progress into the sprints. Two new lists were created, Report Backlog and Report Completed, and two new tags, 'dev' to denote that a ticket was of the development type, and 'rep' to denote that a ticket was of the report type.

The rules feature in Trello was used, which automated adding a tag marked as 'dev' to any ticket added to the Dev Backlog list, and adding a tag 'rep' to any ticket added to the Report Backlog list. Then each sprint the rule on the Current Sprint list would be modified, to add the label corresponding to the current sprint whenever a ticket was moved into there.

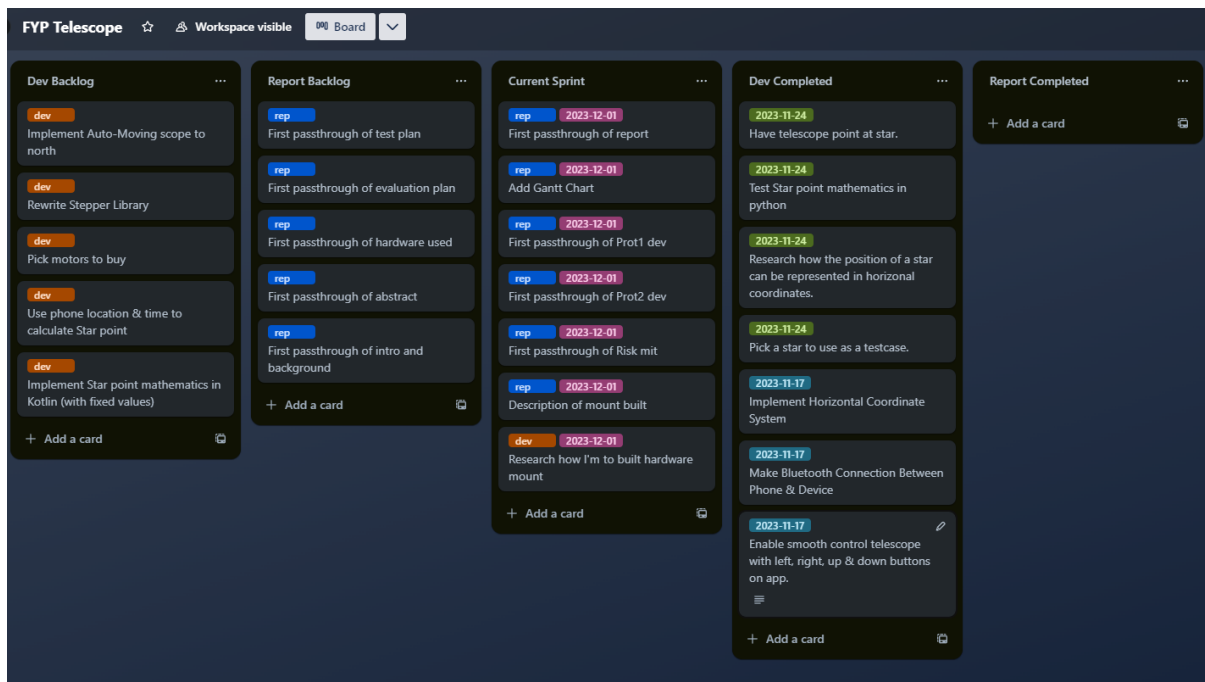


Figure 20: Kanban Board Version 2

4.2. Issues and Risks

Below are a number of risks that were anticipated and the plans for if they happened.

Table 4: Risks

Risk No.	Risk	Solution/Compromise
1	The scope does not slew with sufficient accuracy.	The calibration tweak feature can be used to manually calibrate the scope.
2	The hardware mount does not work/is extremely inaccurate.	The functionality of the application will be demonstrated with the prototype mount.
3	Data from the phone sometimes doesn't make it to the Arduino intact.	An "acknowledge" message will be implemented, which the Arduino will send out upon receiving a message from the phone successfully. If the phone does not receive this acknowledge message, the phone will resend its last command to the Arduino.
4	The time taken for the Arduino to decipher the JSON packets leads to hindered functionality.	Will not use JSON as a transmission format and instead will use a different format that is faster to parse.

4.3. Source Control

Git will be used extensively throughout the project. New features will be developed on separate branches and only merged when complete, to keep the main branch stable. For simplicity and to unify version control, one repository will be used to contain the code for the entire project, including both the Arduino codebase and the Android codebase.

5. System Development

5.1. Prototype Mount

The first step taken was creating a prototype mount to test and demonstrate the software with.

A 28BYJ-5V stepper motor was used on the base to allow the mount to move 360° on the horizontal axis, and an SG90 Digital Servo Motor was used to allow it to move in a 90° range on the vertical axis. A laser pointer was attached to the servo motor to be used to measure the mount's orientation.

These motors were controlled using an Arduino Uno Rev 3 AVR development board.

To send and receive signals, a HC-05 Bluetooth Module was used.

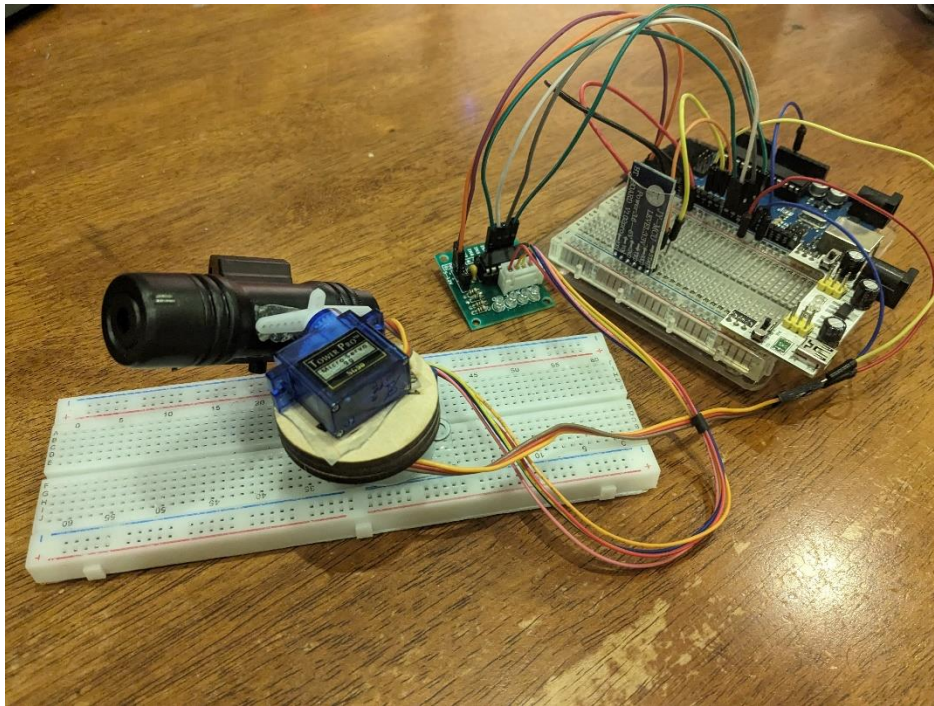


Figure 21: Prototype Mount

5.2. Bluetooth Communication

Then work on the Android Application began, starting with creating a Bluetooth socket to send signals to the telescope mount. Data is sent through this socket between the two devices in the form of a raw byte stream.

In order to decrease coupling between the Android app and the Arduino software, the JSON-based communication protocol devised in section 3.4 was implemented, which allowed various defined signals to be sent from the Android device to the mount.

To create the JSON objects in the Android application the built-in JSON support in the Kotlin language was used. The library ArduinoJSON (33) was used to process the received JSON in the Arduino C++ code.

```
override fun sendSlewCoords(altitude: Double, azimuth: Double) {  
    val dataJson = JSONObject()  
    dataJson.put("Altitude", altitude)  
    dataJson.put("Azimuth", azimuth)  
  
    val instructionJson = JSONObject()  
    instructionJson.put("Instruction", "Slew")  
    instructionJson.put("Data", dataJson)  
  
    write(instructionJson.toString())  
}
```

Figure 22: Android Kotlin function which creates a JSON object and sends it over Bluetooth.

```
void processJSON(String pJson) {  
  
    Serial.println("Processing JSON...");  
    StaticJsonDocument<100> doc;  
    char* jsonArray = new char[pJson.length()+1]; //allocate a new json array on the heap  
  
    strcpy(jsonArray, pJson.c_str()); //Copy json string into json array  
  
    DeserializationError error = deserializeJson(doc, jsonArray); //deserialize json  
  
    // Test if parsing succeeds.  
    if (error) {  
        Serial.print(F("deserializeJson() failed: "));  
        Serial.println(error.f_str());  
        return;  
    }  
  
    const char* instruction = doc["Instruction"];  
  
    if(strcmp(instruction, "Slew") == 0) {  
        slew(doc["Data"]["Altitude"], doc["Data"]["Azimuth"]);  
    } else if(strcmp(instruction, "Calibrate") == 0) {  
        calibrate(doc["Data"]["Azimuth"]);  
    } else if(strcmp(instruction, "Reset") == 0) {  
        reset();  
    }  
  
    delete[] jsonArray; //Deallocate json array (important!)  
}
```

Figure 23: Arduino C++ function which parses a JSON object after it has been received over Bluetooth.

5.3. Manual Control

"Manual" control was then implemented, which is using directional buttons on the mobile application to control the mount, with the left and right arrows controlling the horizontal stepper motor, and the up and down arrows controlling the vertical servo motor.

When pressed, each button sends a signal specific to its direction, and when the mount receives this signal, it will move in that direction by an increment.

When a button is held down, it will repeatedly send its direction signal, causing the mount to repeatedly move in that direction until it stops receiving signals.

These direction signals were first implemented in the JSON format defined in Section 3.4.3.

After testing however, it was found that the Arduino wasn't able process these "manual" commands fast enough to keep up with the frequency of "manual" commands being sent to it.

If this JSON method was continued with, the number of "manual" commands sent to the mount over time would have to be reduced, and to accommodate this, either the "manual" move speed of the mount would have to be reduced, or the responsiveness of the of "manual" movement would have to be reduced by increasing the amount a motor was incremented when it received a single "manual" command.

Instead, it was decided to use a command protocol consisting of single bytes for manual movement. This way, movement speed and accuracy would be retained, as the Arduino can process these single bytes significantly faster than a JSON string.

Table 5: Manual Bytes

Byte	Meaning
r	Move Right
l	Move Left
u	Move Up
d	Move Down

```
int handleSingleByte(char rChar) {
  switch (rChar) {
    case 'r': //Move Right
      direction.manualAzIncrease();
      break;
    case 'l': //Move Left
      direction.manualAzDecrease();
      break;
    case 'u': //Move Up
      direction.manualAltIncrease();
      break;
    case 'd': //Move Down
      direction.manualAltDecrease();
      break;
    case '{': //Enter json Mode
      Serial.println("Entering JSON mode...");
      curlyCount = 1;
      jsonString = "{";
      receiveMode = JSON;
      break;
  }
}
```

Figure 24: Arduino code for processing single bytes received over Bluetooth.

5.4. Coordinate System Implementation on Mount

To be able to specify an exact direction for the telescope to point, the horizontal coordinate system described in section 2.3.2.2 was implemented into the Arduino code.

Direction is specified by two values, altitude (the vertical orientation from 0° to 90°) and azimuth (the horizontal orientation from 0° to 359°).

This is used when commanding the telescope to slew to a specific position, as the coordinates are sent to the mount with the slew command, and the Arduino controls the motors to move the mount to this direction.

Implementing moving a servo motor to a given altitude coordinate between 0° and 90° was fairly straightforward, as these motors are controlled by specifying a position that they should point at in a given range. When a new altitude value is received, the program maps that value to the motor's range, and moves the motor to that position. For example, if a new altitude value of 45° is received (the halfway point in the altitude range of 0° to 90°), the motor will be commanded to move to a position of 90° (the halfway point between the motor range of 0° to 180°)

For slewing the stepper motor to a given azimuth value, this motor is controlled by moving it in "steps", which are divisions of the motor's 360° range. The motor's current position is kept track of with an "azimuth" variable, which is the degrees from an arbitrarily defined 0 position of the motor. When a new azimuth value is received, the difference is found between the new coordinate and the current position, mapped to the number of steps it should be moved, moved that number of steps, and then the current stored azimuth value is updated to the new azimuth.

For example, if the current position of the motor is 90°, and the motor has a 360° range and is divided into 2048 steps, the motor is currently 512 steps from its 0 position ($2048/4$). When a new azimuth value of 270° is received, it is worked out that the motor should be moved 180° ($270-90$), so it is moved 1024 steps ($2048/2$). The current stored azimuth value is updated to 270°.

5.5. Creating Final Mount Design

As the prototype mount was only capable of lifting a small laser pointer, it was upgraded with more powerful motors and given a larger surface area for equipment to be mounted to it.

5.5.1. Frame

The non-electronic components of the main frame consists of:

- Meccano, a construction system which allows structures to be built from strips of metal, nuts, and bolts.
- A metal bracket designed to house two servo motors.
- Custom pieces made from laser cut acrylic.

5.5.1.1. "Lazy Susan" Piece

To allow the mount to rotate a full 360 degrees, while evenly balancing the load on top, a "Lazy-Susan" Meccano piece was utilised, which is a circular component that sits on top of multiple small roller pieces.

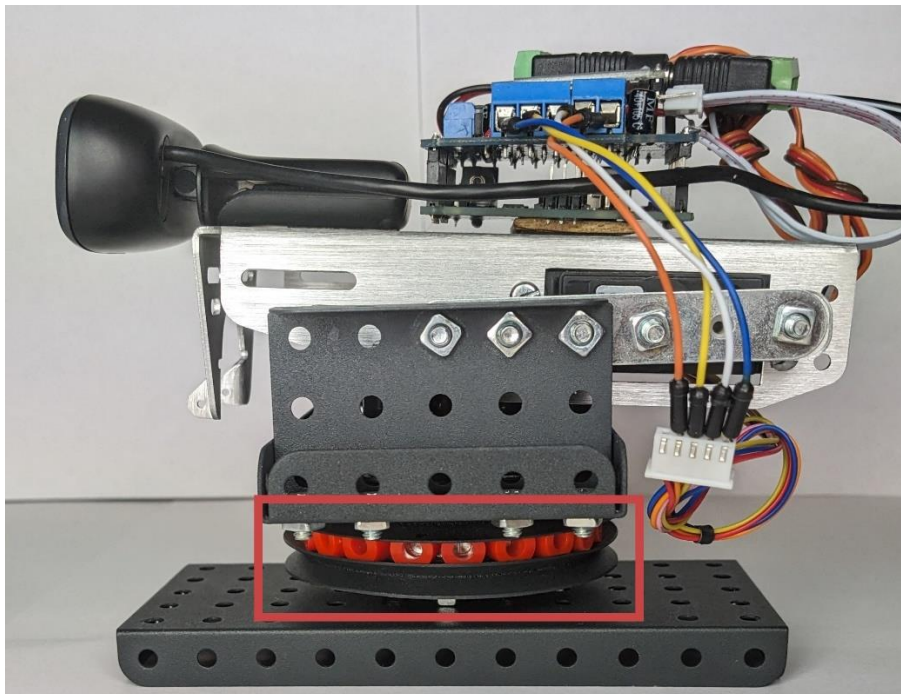


Figure 25: Lazy Susan Piece (Highlighted)

5.5.1.2. Attaching the Azimuth Stepper Motor to Axel

A 28BYJ-48 Stepper Motor was used as the azimuth motor responsible for rotating the mount on the horizontal axis.

The axel of the motor was fastened to a Meccano axle with a Meccano axle connector piece.

As this motor was not designed to be compatible with the Meccano framework, the motor's axel connector had to be shaved down with an angle grinder in order for it to fit into the axel connector.

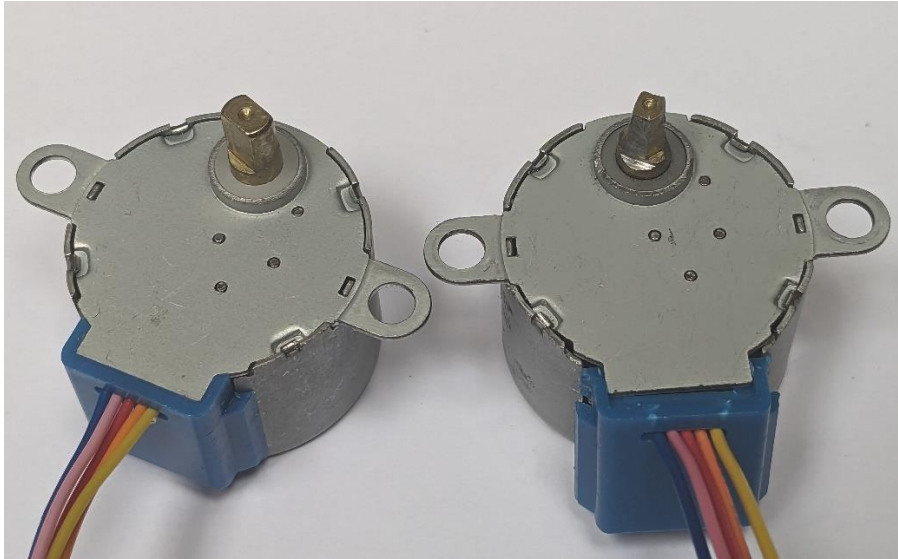


Figure 26: Unmodified stepper motor axle (left) compared to modified stepper motor axle (right)

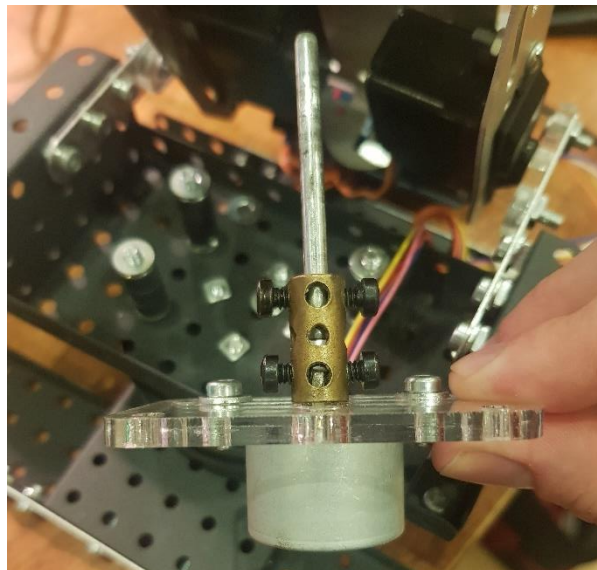


Figure 27: Stepper motor connected to Meccano axle.

5.5.1.3. Attaching Azimuth Stepper Motor to Frame

To attach the base of this motor to the mount a custom Meccano piece was cut out of acrylic using a K40 CO2 Laser Cutting Machine. This was done to create a piece that had bolt holes in the correct location to be compatible with Meccano and the holes in the stepper motor. Schematics of the piece were created in the Lightburn software (34) using measurements taken of the Meccano pieces and the motor. These schematics were then uploaded to the laser cutter to cut the piece.

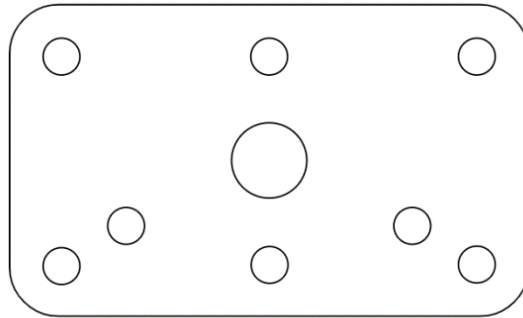


Figure 28: Schematics for custom piece to attach stepper motor.

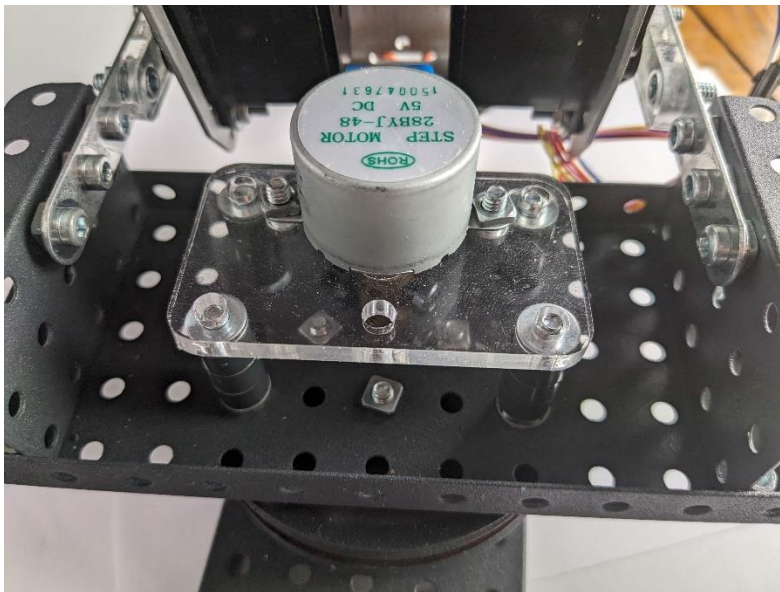


Figure 29: Stepper motor attached with custom acrylic piece.

5.5.1.4. Attaching the Altitude Servo Motors

To control the mount's vertical orientation, a metal bracket was attached designed to house two servo motors. This was used as two servo motors working together would provide a good deal of torque, and the metal bracket provides a useful large surface for attaching equipment.

The servo motors were attached to both sides of the mount using two custom cut acrylic pieces, cut in the same fashion at the piece in section 5.5.1.3.

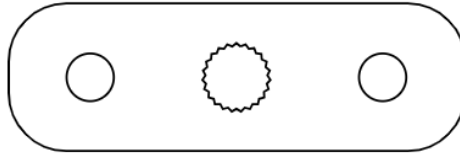


Figure 30: Schematics for custom piece to attach servo motors.

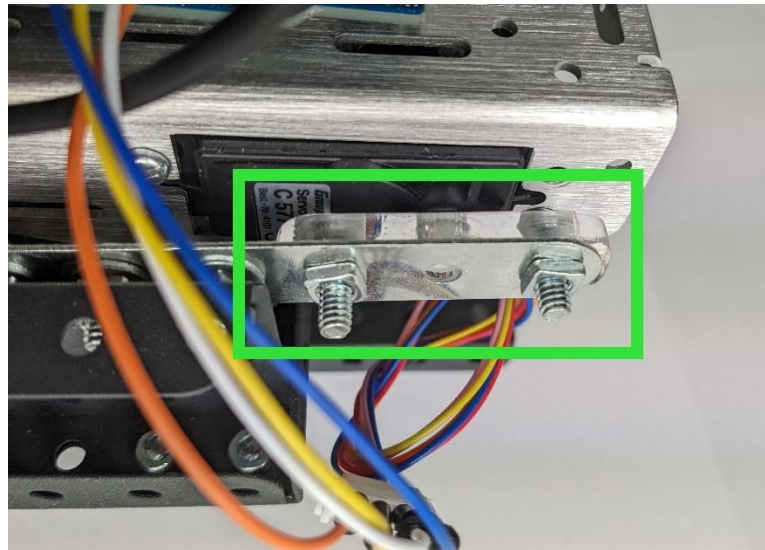


Figure 31: Servo motor attached with custom acrylic piece.

5.5.1.5. All Electronics in Top Section

A key philosophy adhered to whilst designing the mount was to have all electronics be contained in the rotating upper section.

This is because it was realised early on that no matter what, some electronics would have to be in the upper section, such as the altitude motors. If any electronics were put in the bottom section then, it would mean that wires would have to be run between the two sections, which would become tangled as the mount rotated.

So while this would add extra load for the azimuth motor to rotate, it was decided that all electronics should be contained in the rotating upper section.

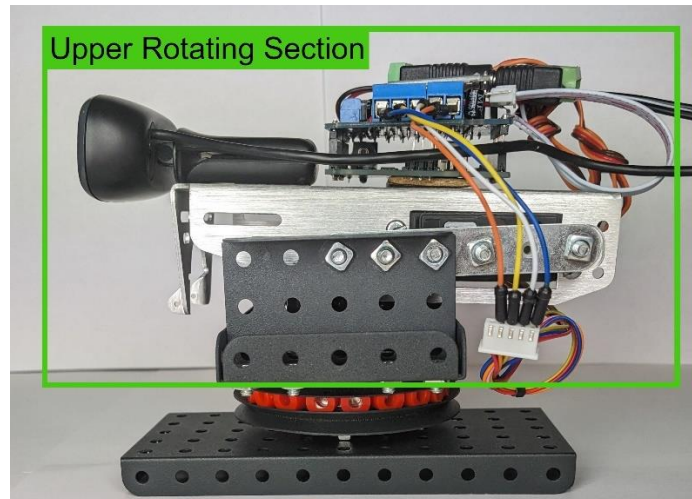


Figure 32: Mount side view with upper rotating section highlighted.

5.5.2. Dual Servo Motors

As mentioned above, it was decided to use two servo motors to increase the torque of the mount. Implementing two servo motors in parallel posed some interesting challenges that had to be overcome.

5.5.2.1. Range Mapping

The first challenge was figuring out what angle to tell each motor to move to when an altitude value was received, as both motors were flipped in opposite directions.

During the first test of this setup, this wasn't taken into account and both motors were driven in the same direction, leading to one motor getting damaged and having to be replaced.

In the diagrams below it can be seen that when applying an altitude value in a range from 0° to 90° , the value has to be mapped to a range of 90° to 0° for the left servo, and 90° to 180° for the right servo.

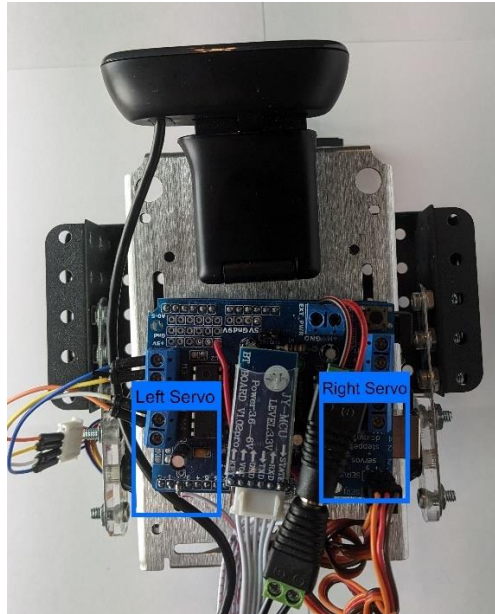


Figure 33: Mount top view with servo positions highlighted.

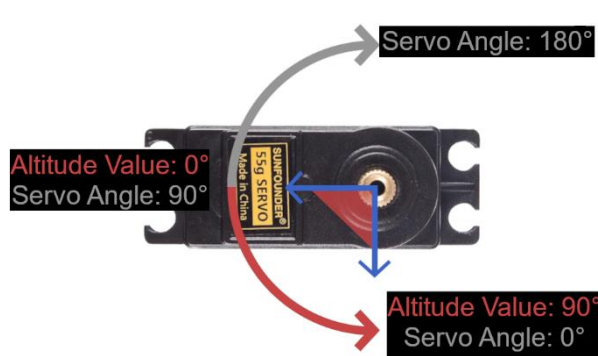


Figure 34: Left Servo Range

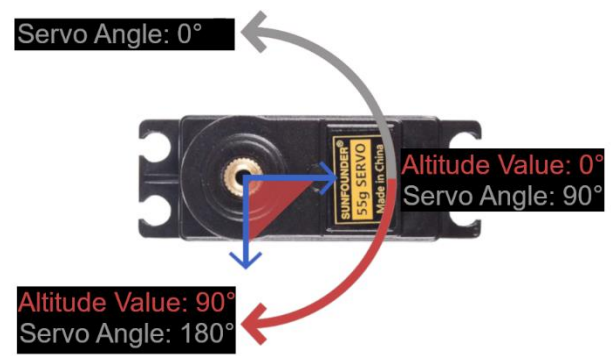


Figure 35: Right Servo Range

```
void moveLeftServo(int pAlt) {
  //for C577 servo
  //left range: 90 -> 0
  leftServo.write(map(pAlt, 0, 90, 90, 0));
}

void moveRightServo(int pAlt) {
  //for C577 servo
  //right range: 90 -> 180
  rightServo.write(map(pAlt, 0, 90, 90, 180));
}
```

Figure 36: Arduino code for mapping altitude value to left and right servo motors.

5.5.2.2. Keeping Each Servo in Sync

With both servos moving in parallel, it was found that due to the way they were manufactured, both servos moved at slightly different speeds, meaning that when moving large distances, they would gradually get out of sync. To remedy this, the Arduino program was altered to divide the amount that has to be moved into increments and command each servo to move that increment, and then wait until the other has caught up, before moving again to the next increment. This significantly improved the synchronicity of the two servo motors.

```
for (int i = increment; (alt + i) <= pAlt; i += increment) {  
    moveLeftServo(alt+i);  
    moveRightServo(alt+i);  
    delay(100);  
}
```

Figure 37: Arduino code to move servo motors in increments.

5.5.3. Increasing Stepper Motor Power

Due to the now heavier weight of the mount, a more powerful azimuth stepper motor was now needed to rotate the mount horizontally.

5.5.3.1. AccelStepper Library

The first method used to improve this was switching the stepper motor library from the default Arduino Stepper library(35) to the AccelStepper library(36). This library was more sophisticated, giving more control of the motor speed and increasing torque. While this did provide more power, it still was not enough to rotate the mount.

5.5.3.2. Bipolar Motor Modification

The next method used was modifying the unipolar 28BYJ-48 stepper motor into a Bipolar one. Through researching, it was found out that this is a well-documented modification that can be carried out on this particular kind of motor. This modification makes the motor 3 times more powerful, at the cost of requiring a more sophisticated motor control board capable of controlling a Bipolar motor (37). The modification involved disassembling the motor and severing one of the connections on the PCB with a knife.

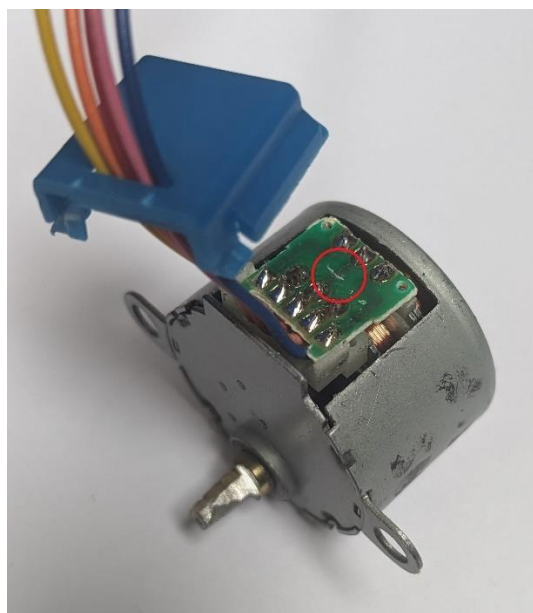


Figure 38: Stepper Motor Modification (incision highlighted with red circle)

5.5.3.3. Attaching Adafruit Motor Shield

To control the newly created Bipolar motor, an Adafruit Motor Shield V1(38) was used. This is placed directly on top of the Arduino, meaning that all components connected to the Arduino had to be connected to the Adafruit shield now instead. The shield has two dedicated ports for servo motors, so the two servo motors could be connected to it with no issues. After checking the documentation of the board, it was found that pin 2 was the only digital I/O pin on the board that was available, as the other pins were used for the control of stepper motors. This pin was soldered to the pin on the HC-05 Bluetooth module responsible for receiving data. The Arduino code was then modified to use the Adafruit stepper motor library.

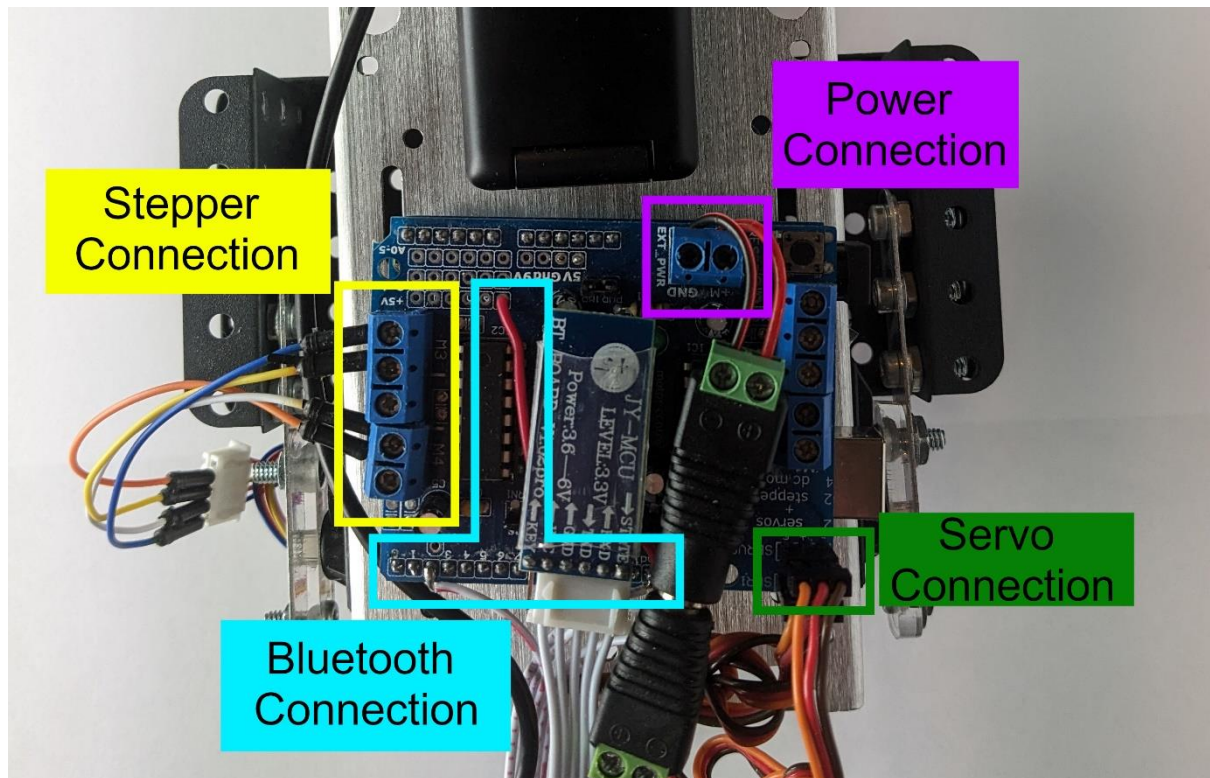


Figure 39: Adafruit Shield Connections

5.5.4. Power Source

For a power source, a battery pack was experimented with, but this proved to be too much weight for the motors to handle well.

It was decided that the aspect of the mount being powered by standalone means would be moved to future development plans, when the mount is upgraded again with more powerful motors.

Instead, a variable DC power supply was used to find the voltage that worked best with the mount, which was found to be 9 volts. A 9-volt AC adapter was then acquired, and a power supply connector was soldered to the Adafruit shield.

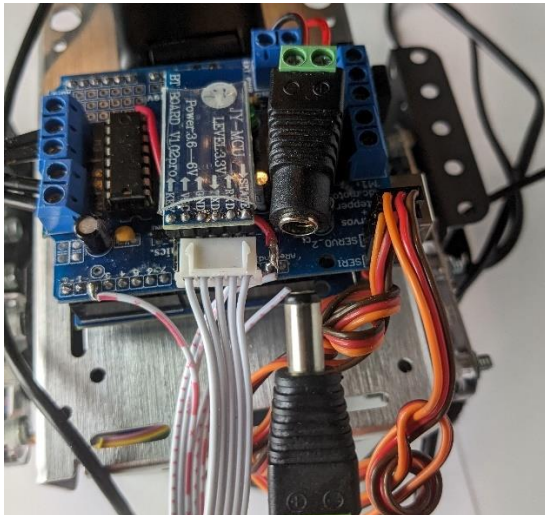


Figure 40: AC Adapter (Disconnected)

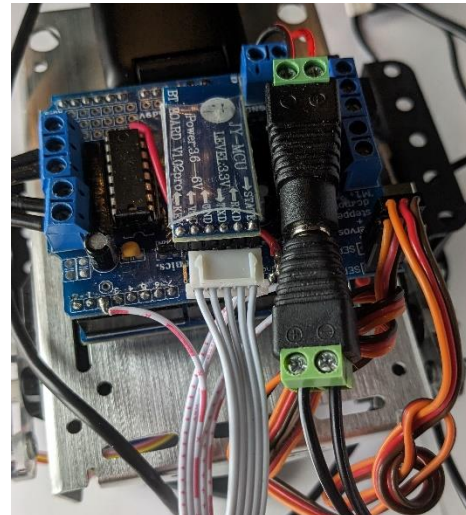


Figure 41: AC Adapter (Connected)

5.5.5. Providing Documentation for Mount Design

Contributing to the 5th objective defined in Section 1.2 of making the project open source so that users can build their own, documentation was provided for the mount design in the “readme” of the GIT repository in the form of images of the mount and a components list.

Components		
Name	Qty.	Usage
28BYJ-48 5V Stepper Motor	1	Moving the telescope horizontally.
Graupner C577 Servo	2	Moving the telescope vertically.
HC-05 Bluetooth Module	1	Allow the mount to receive Bluetooth commands.
Arduino Uno Rev 3 AVR Development Board	1	Programmed to process serial commands and move motors accordingly.
Miscellaneous Meccano Parts	N/A	Constructing the mount mechanism.
Adafruit Motor Shield v1	1	Controlling the stepper and servo motors.

Figure 42: Component documentation in GIT "readme".

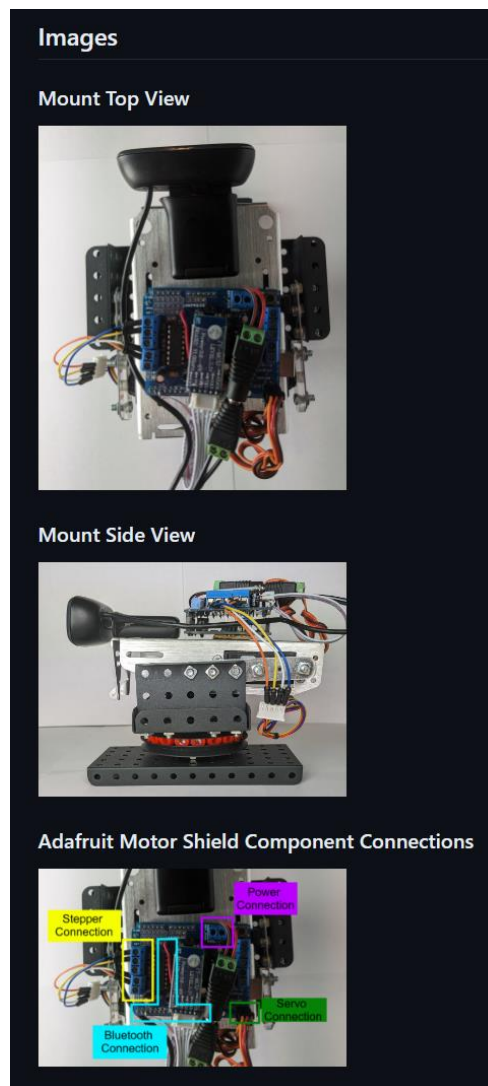


Figure 43: Mount Images in Git README.

5.6. Separating Application Functionality into Multiple Screens

The functions of the application, which was previously all contained in one screen, were separated out into the different screens defined in section 3.2.1. Each individual screen is referred to as an activity in Android Studio.

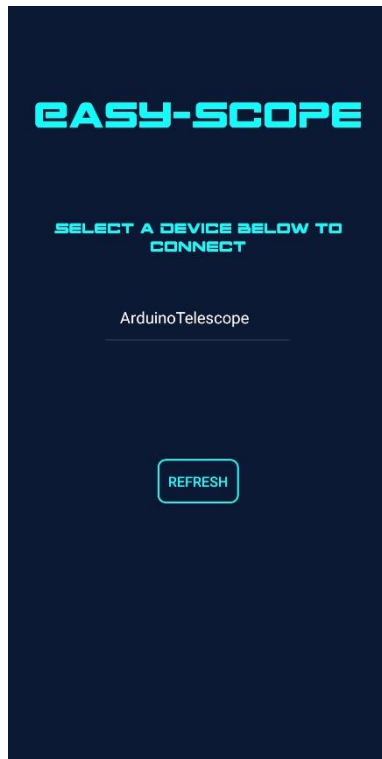


Figure 44: Application Connect Screen

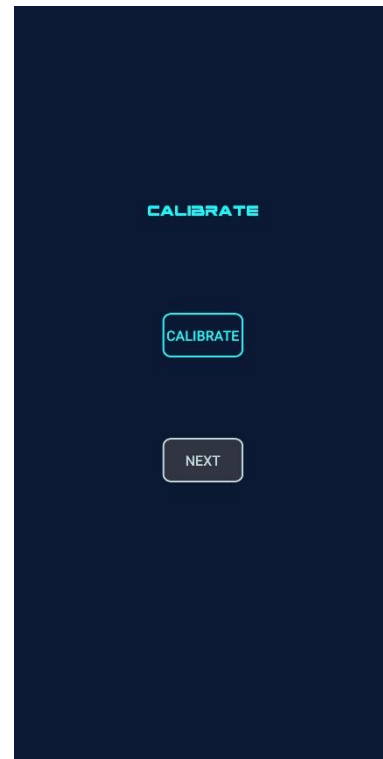


Figure 45: Application Calibrate Screen

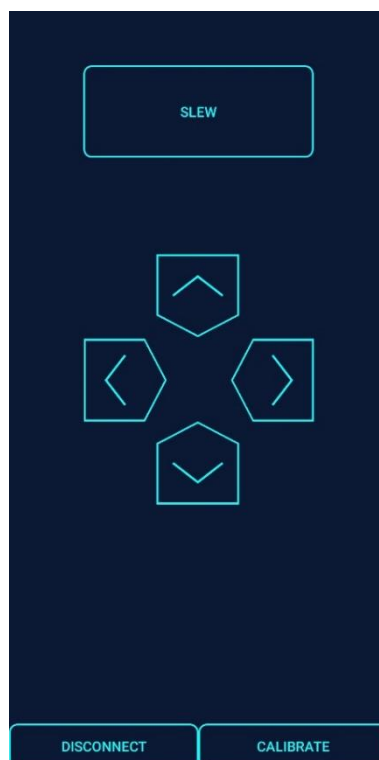


Figure 46: Application Main Screen



Figure 47: Application Object Select Screen - Star Tab



Figure 48: Application Object Select Screen - Planet Tab

5.7. Connect Activity

The logic for establishing a Bluetooth connection was moved to the 'ConnectActivity.kt' activity, seen in Figure 44. The functionality was improved with a menu that displays paired Bluetooth devices. In order to allow all activities in the application to be able to send data over Bluetooth, the code for establishing a Bluetooth connection and sending data over Bluetooth was moved to a background task called 'BluetoothService.kt'. To reduce the likelihood of bugs, this service contains functions that abstract the construction of the commands that are sent over Bluetooth. One example of this is the "sendCalibrateData" function which takes an azimuth value as a parameter, creates the JSON object and sends it over Bluetooth.

```
override fun sendCalibrationData(azimuth: Double) {
    val dataJson = JSONObject()
    dataJson.put("Azimuth", azimuth)

    val instructionJson = JSONObject()
    instructionJson.put("Instruction", "Calibrate")
    instructionJson.put("Data", dataJson)

    write(instructionJson.toString())
}
```

Figure 49: Android Function to send calibration data in Bluetooth Service.

5.8. Main Activity

The logic for manual mount control was moved to 'MainActivity.kt', which can be seen in Figure 46. This activity contains buttons that navigate to other activities of the application.

5.9. Object Select Activity

When the button labelled "Slew" in the main activity is tapped, the user is brought to the Object Select Activity, which contains two lists, one with stars and one with planets. These lists can be seen in Figure 47 and Figure 48.

5.9.1. Star List

The star list is populated from a local SQLite database added to the application. The database contains a table which stores the star's name and equatorial coordinates.

Table Field	Description
ID	The primary ID of the star
NAME	The name of the star
RA	The Right Ascension Equatorial Coordinate
DEC	The Declination Coordinate

Table 6: Star Database Fields

The database is designed to be easily updated by the developer. The star data of the database is stored in a StarData.JSON file in the assets folder of the application.

When the "DATABASE_VERSION" constant is incremented in StarDBHelper.kt, the next time the application is launched the database will be regenerated with the data in the StarData.JSON.

```
{
  "Stars": [
    {
      "Name": "Sirius",
      "RA": 6.752472222222222,
      "DEC": -16.716111111111111
    },
    {
      "Name": "Canopus",
      "RA": 6.399194444444444,
      "DEC": -52.69583333333333
    }
  ]
}
```

Figure 50: Format of JSON file used to store star data.

The database was populated with data retrieved from a NASA hosted online database containing data derived from the Bright Star Catalogue 5th Edition (39), which is a widely used source of data for stars brighter than a magnitude of 6.5 (40). The data was exported from this database to a delimiter-separated value file, and a range of fifty stars that would be visible to the naked eye was chosen. A python script was then written to translate the data into the JSON format used by the application.

```
lines = starsRaw.readlines()
for line in lines:
    values = line.strip().split('|')

    newStar = {"Name": values[0],
               "RA": timeToDec(values[3]),
               "DEC": timeToDec(values[4])
              }

    starArray.append(newStar)

jsonData = {"Stars": starArray}
json.dump(jsonData, starJson, indent=4)
```

Figure 51: Python Script used to translate convert exported star data to JSON.

5.9.2. Planet List

As mentioned in section 3.2.2.2, the equatorial coordinates for planets cannot be stored in a database similar to the stars, as the positions of planets relative to the Earth are constantly changing. Instead, the Astronomy Engine Library(22) was used, which provides functions that return the current positions of every planet in the solar system and moon, given the current time. The planets are provided in a list format identical to how the stars are provided, abstracting the difference in implementation from the user.

5.9.3. Slewing & Tracking

When either a star or a planet is selected, the user is brought back to the main activity. The Slew button now displays the object that is being tracked. In the background, the main activity calls the function `getHorCoords` regularly at a set interval. Utilising the functions provided by the Astronomy Engine Library(22), the function takes the given equatorial coordinates, the current time, the current location of the mobile device, and produces the horizontal coordinates (altitude and azimuth), that the telescope should point at to view the body. These coordinates are then formatted into a JSON command and sent over Bluetooth by the `BluetoothService.kt` activity. Since the function is called constantly at a set interval, the telescopes position is constantly being updated to keep the object in view.

```
fun getHorCoords(pBody: Body): Topocentric? {  
  
    //Get Time  
    val currTime = Calendar.getInstance()  
  
    val time = Time(  
        currTime.get(Calendar.YEAR), currTime.get(Calendar.MONTH),  
        currTime.get(Calendar.DATE), currTime.get(Calendar.HOUR_OF_DAY),  
        currTime.get(Calendar.MINUTE), currTime.get(Calendar.SECOND).toDouble()  
    )  
  
    //define observer (scope position on Earth)  
    val observer = Observer(latitude!!, longitude!!, 0.0)  
  
    //define equatorial coordinates of star for current time  
    val equ_ofdate: Equatorial = equator(  
        pBody!!,  
        time,  
        observer,  
        EquatorEpoch.OfDate,  
        Aberration.Corrected  
    )  
  
    //translate equatorial coordinates to horizontal coordinates  
    val hor: Topocentric = horizon(  
        time,  
        observer,  
        equ_ofdate.ra,  
        equ_ofdate.dec,  
        Refraction.Normal  
    )  
  
    return hor  
}
```

Figure 52: Function to calculate the horizontal coordinates needed to orient to a body.

5.9.4. Cancelling Slew

At the top of the object select screen is a 'Cancel' button. This button will return the user to the main screen without selecting a body. It will also send a reset command to the Arduino. The Arduino program was altered, so that it sets the altitude value to 0 when this 'reset' command is received.

```
override fun sendReset() {  
    val instructionJson = JSONObject()  
    instructionJson.put("Instruction", "Reset")  
    write(instructionJson.toString())  
}
```

Figure 53: Android Code to Send Reset Command.

```
void reset() {  
    direction.moveToAlt(ALT_LOW_BOUND);  
}
```

Figure 54: Arduino Code for Resetting Mount.

5.9.5. Calculating What Objects Are Currently Visible

The user shouldn't be able to select objects that are below the horizon, as the mount will not be able to point at these. A feature was added to the application where a check is run on each body in both the planet and star lists, which calculates the horizontal coordinates of the body, and prevents the user from selecting it if the altitude coordinate is below the threshold that the mount can see.

```
fun bodyUnderHorizon(body: Body): Boolean {  
    val hor = getHorCoords(body)  
    return hor!!.altitude < HORIZON_ALT  
}
```

Figure 55: Android function for checking if a body is under to horizon.

5.10. Calibrate Activity

In order for the mount to find out the azimuth direction it is pointing after it is powered on, the calibrate activity was developed. As shown in section 2.3.2.2, the azimuth value is the offset from north.

The user holds the mobile device to a specific position on the mount, and when the calibrate button is pressed, the azimuth value is read from the phone's compass sensor. This value is then sent to the Arduino mount in a 'Calibrate' JSON command. When the Arduino receives this command, it will update its current stored azimuth value to this new value.

Since the mount uses servo motors for its altitude orientation, the altitude value does not need to be calculated by the phone, as servo motors always know what position they are at in their set range, as opposed to stepper motors where it cannot be determined what their current rotation is.

In order to make calibration easier, whenever the calibration activity is entered, a reset command is sent to the mount, which sets the mount's altitude orientation to 0.

The calibration button also serves another purpose. When the calibration button is pressed, it also retrieves and saves the phone's current location for future use in the slewing to and tracking of objects. It was designed this way to minimise the number of times the application retrieves the location of the mobile device, and because the mobile device will be closest to the mount when it is being calibrated, meaning that the location gained will more accurately represent the position of the mount.

Logic was also added that will initially prevent the user from moving past the calibration screen until the mount is calibrated at least once.



Figure 56: Holding phone to calibrate mount.

5.11. Viewing Apparatus

The original plan was to attach a spotting telescope to the mount, such as the Maginon Vision Spotting Scope. After testing the mount with this attached however, it proved to be too heavy for the motors used by the mount to lift.

Instead, a wireless ESP32 camera was chosen to be used as a viewing apparatus instead, to demonstrate what the mount can see. This is a microcontroller board with a camera and WI-FI chip attached. It was set up to act has a HTTP server, which served a live feed of the camera over the network, allowing it to be viewed on a desktop computer. This was connected to the Arduino board, allowing it share the same power.

Unfortunately, during a debugging session, excessive electrical was accidently applied to the ESP32 board, causing irreversible circuit damage.

It was then decided to attach a Logitech C270 Desktop Webcam. While not wireless, this viewing apparatus provides a high-quality feed of what the mount can currently see.

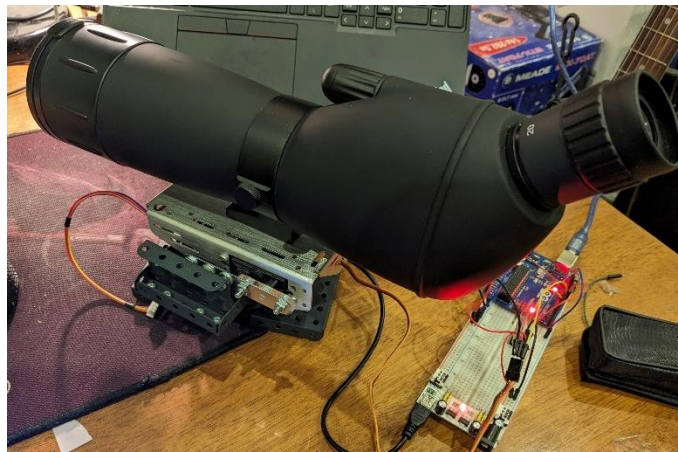


Figure 57: Mount with spotting telescope attached.

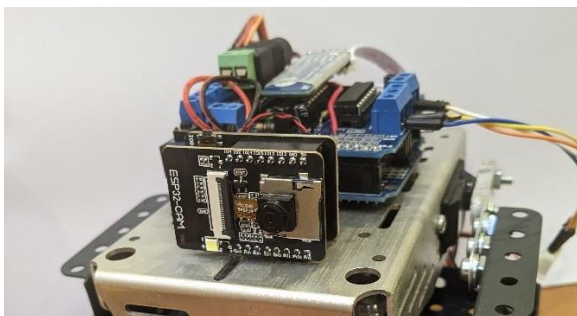


Figure 58: Mount with ESP32-CAM attached.

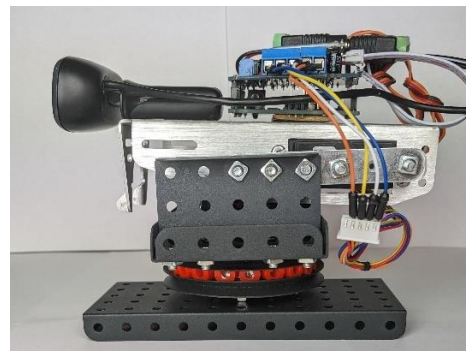


Figure 59: Mount with Logitech Webcam Attached.

6. System Validation

Since an iterative, agile approach to development was followed, the system was tested continuously throughout the project lifecycle through the use of multiple unit tests of varying complexity. Near the end of the project timeframe when the system was in a state usable by the general public, evaluation of the system was carried out with the project's target demographic.

6.1. Unit Testing

A number of unit tests were derived to test the functionality of the system. These tests were carried out multiple times throughout development any time a significant change was made, to ensure that all features of the product were still functional.

6.1.1. Mount Testing

6.1.1.1. Testing Materials

A separate branch was created in the project Git repository called 'Debug Activity' which contained an activity that allowed each command to be sent to the mount. This was used during the unit testing of the system.

In some tests a protractor was used to measure the direction the mount was facing. This is a large circular measuring device printed onto paper.

For some tests, the open source planetarium smart device application Stellarium was used, in combination with a laser attached to the smart device. Stellarium instructed the tester which direction to point the mobile device towards an astronomical body, meaning that wherever the laser hits in the room is an indoor representation of that body.



Figure 60: Smartphone with laser attached.



Figure 61: Android Test Activity

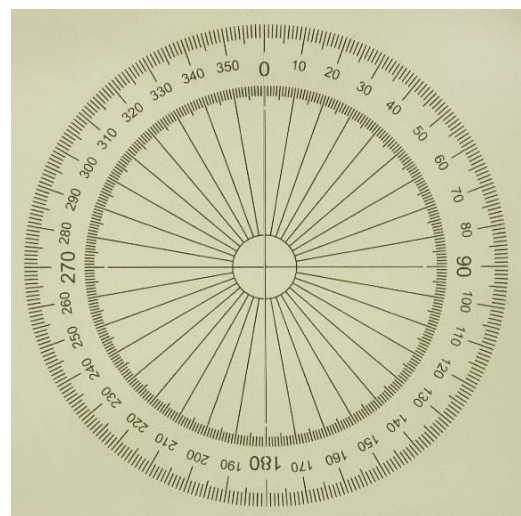


Figure 62: Protractor

Unit Test M-01

Functionality Being Tested

Manual Slewing by increments.

Test Steps:

1. Send a single 'UP' manual slew command from mobile application.
2. Send a single 'DOWN' manual slew command from mobile application.
3. Send a single 'LEFT' manual slew command from mobile application.
4. Send a single 'RIGHT' manual slew command from mobile application.

Expected Results

- Mount will slew up, down, left, and right by the same amount.
- Mount will be facing the same direction it was before the test was carried out.

Result of Last Time Run

Pass

Unit Test M-02

Functionality Being Tested

Testing speed consistency of manual slew.

Test Steps:

1. Place mount on protractor pointing towards the 0° mark and ensure the mount's vertical orientation is at its minimum.
2. Hold down right button, and time how long it takes for the mount to point towards the 90° mark.
3. Hold down left button, and time how long it takes for the mount to point back towards the 0° mark.
4. Hold down the up button, and time how long it takes for the mount to reach the maximum vertical orientation.
5. Hold down the down button, and time how long it takes for the mount to reach the minimum vertical orientation.

Expected Results

Both horizontal time values should be equal, and both vertical time values should be equal.

Results of Last Time Run

Pass

Unit Test M-04

Functionality Being Tested

Coordinate Slewing Accuracy.

Test Steps:

1. Place mount on protractor pointing towards the 0° mark.
2. Using the android debug activity, command the mount to slew 90°.
3. Command the mount to slew 90°.
4. Command the mount to slew 90°.
5. Command the mount to slew 90°.

Expected Results

- At step 2, mount should be pointing towards 90° on the protractor.
- At step 3, mount should be pointing towards 180° on the protractor.
- At step 4, mount should be pointing towards 270° on the protractor.
- At step 5, mount should be pointing towards 0° on the protractor.

Results of Last Time Run

Pass

All of the targets were hit with a 5° variance.

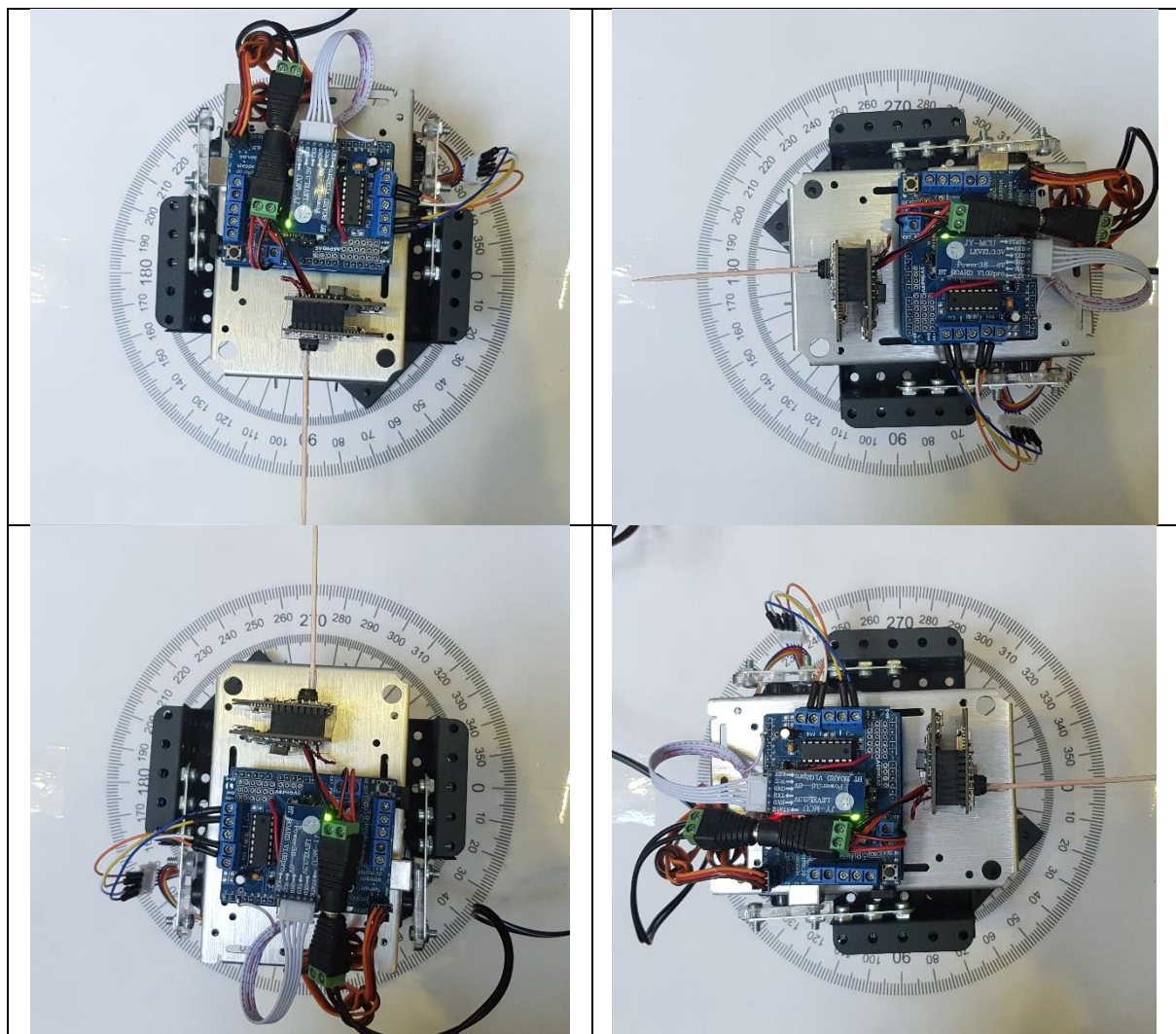


Table 7: Unit Test M-04 Images

Unit Test M-05

Functionality Being Tested

Updating Mount's Azimuth Value (Calibration).

Test Steps:

1. Ensure that the mount is slewed to its azimuth of 0° and place the mount on the protractor pointing towards the 0° mark.
2. Command the mount to slew to 180°.
3. Command the mount to update its internal azimuth value to 0°.
4. Command the mount to slew to 180°.

Expected Results

On step 4 the mount will slew to 0° on the protractor. This is because the mount's internal azimuth value's 0° is now at 180° on the protractor.

Results of Last Time Run

Pass

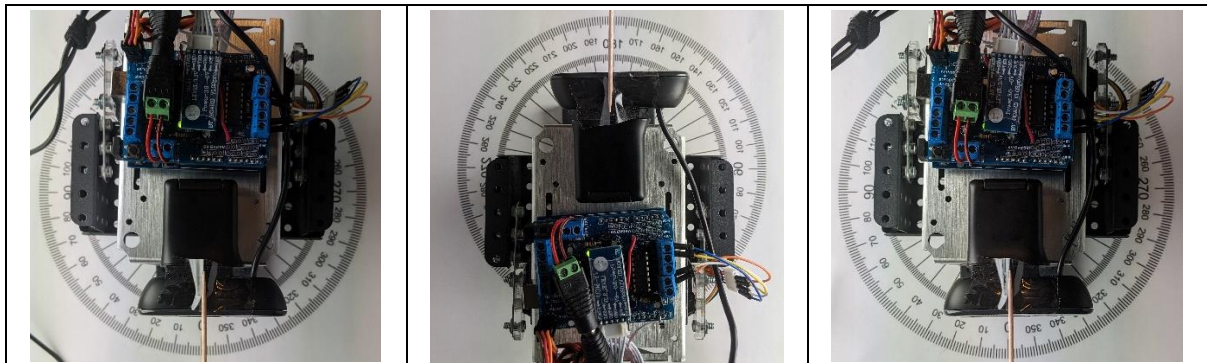


Table 8: Unit Test M-05 Images

Unit Test M-06

Functionality Being Tested

Mount Reset Functionality

Test Steps:

1. Command the mount to slew to the maximum vertical orientation.
2. Send the reset signal from the debug activity.

Expected Results

The mount's vertical orientation will be lowered to the lowest.

Results of Last Time Run

Pass

6.1.2. Mobile Application Tests

Unit Test A-01

Functionality Being Tested

Accuracy of compass sensor for calibration.

Test Steps:

1. Use the applications calibrate feature on the mount.
2. Command the mount to slew to an azimuth of 0°.
3. Use a physical compass to verify if the mount is pointing north.

Expected Results

The mount is pointing north.

Results of Last Time Run

Pass

5° of variance was observed.

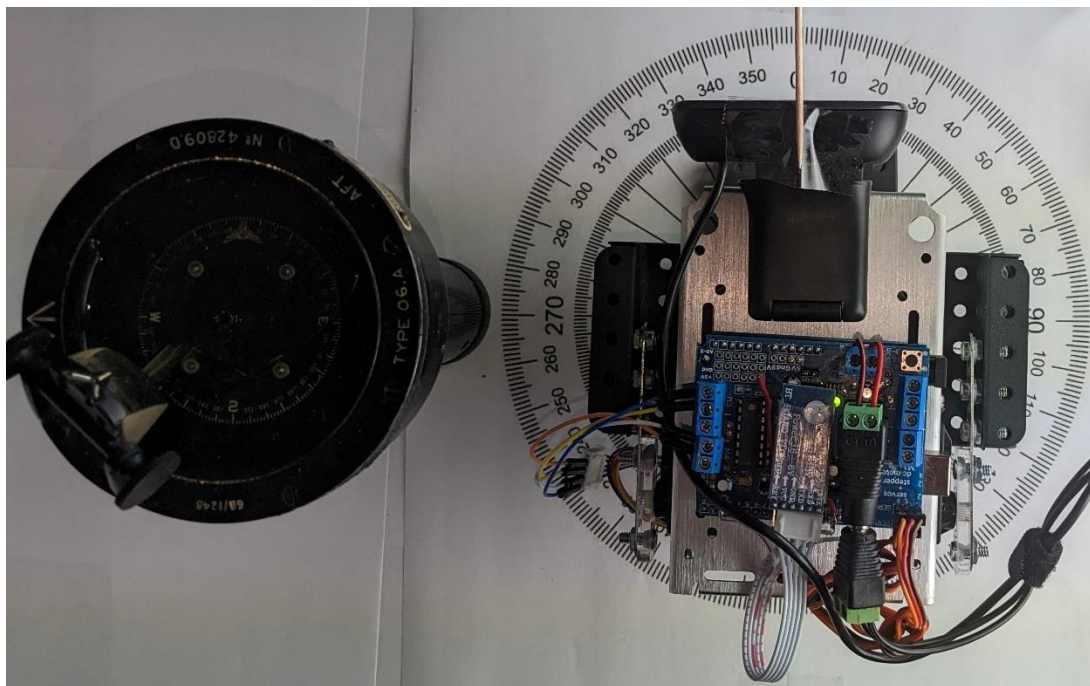


Figure 63: Unit Test A-01

6.1.3. Full System Tests

Unit Test F-01

Functionality Being Tested

Application commands are being transmitted correctly by the application and are being received correctly by the Arduino.

Test Steps:

1. Load the 'BasicBluetoothReader.ino' sketch onto the Arduino.
2. Ensure that the HC-05 Bluetooth module is connected to the Arduino.
3. Ensure that the Arduino is connected to a PC via USB and that the serial monitor is enabled and set to a baud rate of 115200.
4. Send
5. Send a slew command from the application.
6. Send a calibrate command from the application.
7. Send a reset command from the application.

Expected Results

The serial monitor should display correctly:

- The slew command.
- The calibrate command.
- The reset command.

Results of Last Time Run

Pass

Unit Test F-02

Functionality Being Tested

Mount slews accurately to an astronomical object.

Test Steps:

1. Connect to and calibrate the mount.
2. Select visible astronomical body X for the mount to slew to.
3. Select visible astronomical body X in Stellarium with the laser pointer attached to the smart device and point the smart device in the direction of the body.

Expected Results

The laser should be visible in the live camera feed.

Results of Last Time Run

Astronomical Body Selected (X): Jupiter

Observations: Body is visible through camera, although offset from centre (see Figure 64).

Result: PASS

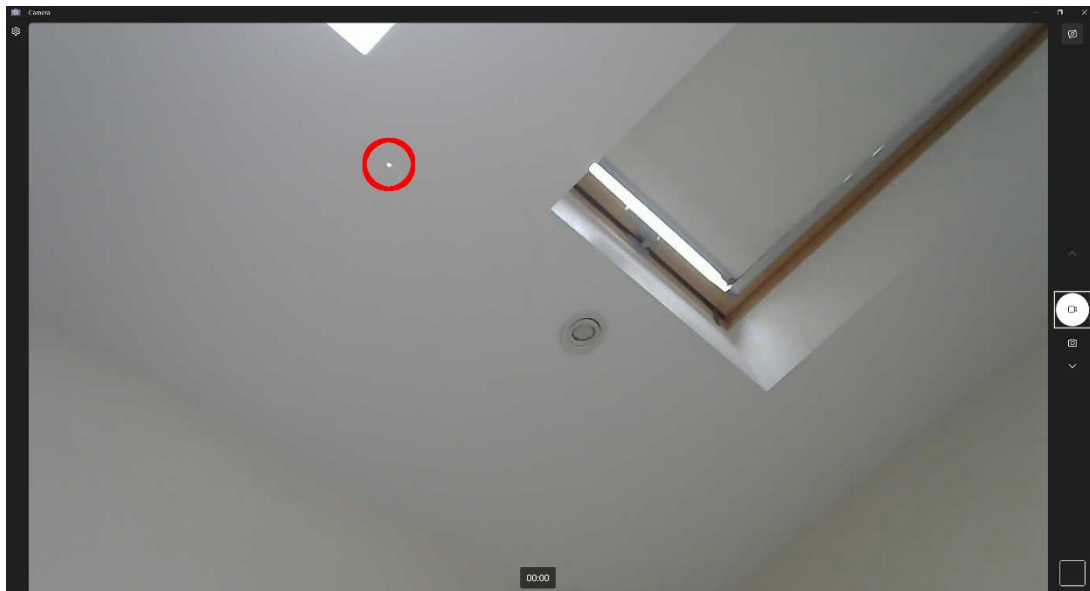


Figure 64: Results of Unit Test F-02 (laser representing astronomical object highlighted)

Unit Test F-03

Functionality Being Tested

Mount accurately tracks an object over a period of time.

Test Steps:

1. Connect to and calibrate the mount.
2. Select visible astronomical body X for the mount to slew to.
3. Select visible astronomical body X in Stellarium with the laser pointer attached to the smart device and point the smart device in the direction of the body.
4. Wait Y minutes, making sure to move the smart device so as the body moves through the sky.

Expected Results

The laser stays in the view of the camera throughout the duration of the test.

Results of Last Time Run

Astronomical Body Selected (X): Jupiter

Time Waited (Y): 30 minutes.

Observation: Object was still in view of camera after time had elapsed.

Result: PASS

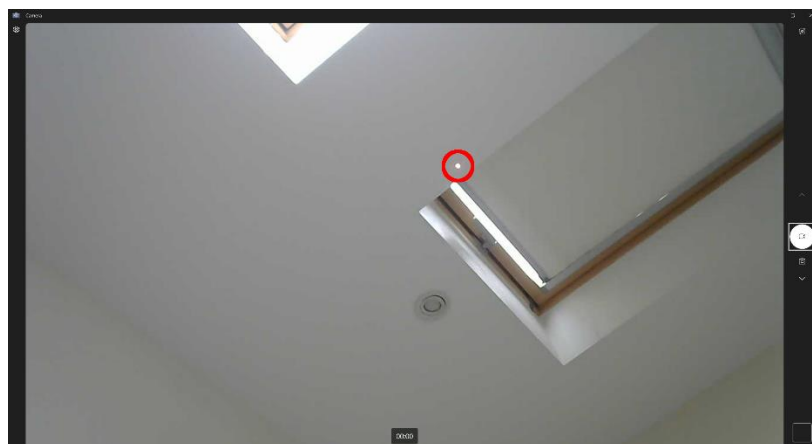


Figure 65: Results of Unit Test F-02 - Time Waited (Y) = 0 (Laser Representing astronomical object highlighted)

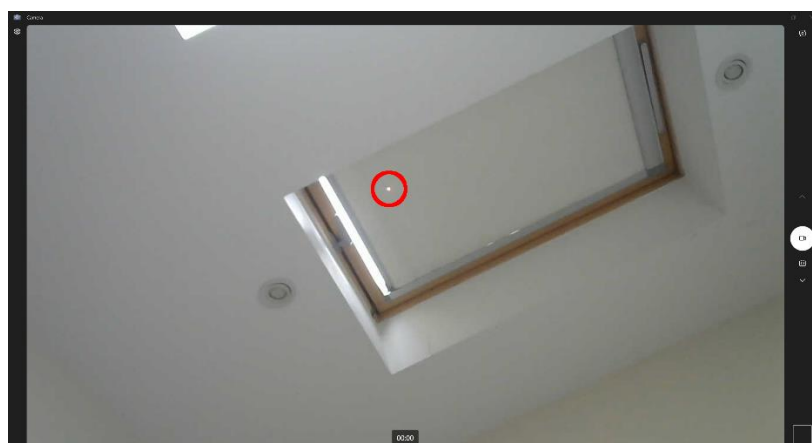


Figure 66: Results of Unit Test F-02 - Time Waited (Y) = 30 minutes
(Laser Representing astronomical object highlighted)

6.2. User Evaluation

To evaluate the system, a group was assembled to test every feature of the system. The members of the group were all of the project's target demographic, people with limited or no prior astronomy experience. All evaluators were able to use all features of the application without instruction. After using the system, each evaluator was given a survey to fill out.

Table 9: Summary of Evaluation Feedback

Question No.	Question	Type	Avg. Response
1	How easy to use did you find the setup/calibration process?	Rate 1-5	4.8
2	How easy to use did you find the object selection process?	Rate 1-5	4.8
3	How accurate/helpful did you find the object selection process?	Rate 1-5	4.6
4	How useful was the manual control feature of the telescope?	Rate 1-5	5
5	How useable was the user interface?	Rate 1-5	5
6	Did you learn something about astronomy today?	Yes/No	Yes

The user feedback received was very positive, with most participants having very little trouble using the system, which can be attributed to the accessible and simple user interface provided by the application.

The area that scored the least was question 3, which is related to the accuracy of the object selection process. This is further explored in Section 6.4.

6.3. Conclusion

6.4. Review of Project Objectives

In this section the project as a whole is reflected upon through revisiting the project objectives defined in Section 1.2 to review how well they were met.

6.4.1. Objective 1

“Create an overall exemplary GoTo telescope system providing all of the best features currently on the market at a low price point”

The research of other GoTo telescope systems in Section 2.1.4 resulted in a list of important design principles that once incorporated would result in an exemplary GoTo telescope system, which the system developed in this project has incorporated all of.

6.4.1.1. Easy to Use

The system is easy to use, as the evaluation section of the project in Section 6.2 proved that most users had no trouble using the system, due to the accessible user interface guided by Nielson's usability heuristics.

6.4.1.2. Functional

The system provides features that provide a lot of value to a newcomer to astronomy, providing the ability to wirelessly control a telescope's orientation and automatically slew to a wide array of astronomical objects. These functions were proven to work well during the testing of this project outlines in Section 6.1.

6.4.1.3. Cost Efficient

The system is cost efficient, as most intensive computation is carried out on the user's mobile device, cutting out the need of purchasing a power computer for the system. Instead, the minimal computation of the mount is carried out on an inexpensive Arduino Uno. The rest of the mount was also designed with this philosophy, and a further breakdown can be seen in Section 6.4.3.

6.4.1.4. Offline Capable

Once downloaded the mobile device does not need any internet connection to achieve full functionality.

6.4.1.5. Power Efficient

The mount uses low powered components, and only required 9 volts to achieve full functionality. The mobile application was designed with power efficiency in mind, and utilises techniques to not expend energy unnecessarily, such as only acquiring location once during the calibration stage.

6.4.1.6. Modular & Open-Source Architecture

The system is modular in that it uses a decoupled JSON protocol which is reflected upon in greater detail in Section 0, and the system has followed an open-source architecture in that everything related to the project has been uploaded to GitHub, which is reflected upon in greater detail in Section 6.4.5.

6.4.2. Objective 2

“Create an accessible mobile application with features of varying complexity for controlling a GoTo telescope”

An accessible mobile application was successfully created, with the created user interface closely mirroring the documented plan. All of the evaluators were able to successfully navigate and use the application and rated its usability highly.

6.4.3. Objective 3

“Design and create a motorised telescope mount from low-cost materials that can support a viewing device, receive commands wirelessly and point at any location in the sky”

The final mount developed meets all of the criteria defined in the project objective. The mount is fully functional, able to receive commands over Bluetooth and point at any location in the sky.

The mount is also able to support a viewing device in the form of a camera, and if upgraded as defined in Section 6.5.1, it will be able to support viewing devices of heavier weight.

While the total cost had to be increased from the original plan due to the additional of the Adafruit Motor Shield, the end cost is very low compared to most other GoTo telescope systems on the market.

Table 10: Final Mount Costs

Name	Qty.	Description	Usage	Approx. Cost
28BYJ-48 5V Stepper Motor	1	Motor with 360° range.	Moving the telescope horizontally.	€4
Graupner C577 Servo	2	Motor with 180° range.	Moving the telescope vertically.	€10
HC-05 Bluetooth Module	1	Allows the transmission and receipt of wireless serial data.	Allow the mount to receive Bluetooth commands.	€5
Arduino Uno Rev 3 AVR Development Board	1	Open-Source Microcontroller Board	Programmed to process serial commands and move motors accordingly.	€25
Miscellaneous Meccano Parts	N/A	Metal construction systems used to build small mechanical devices.	Constructing the mount mechanism.	€15
Adafruit Motor Shield v1	1	Board that sits on top of Arduino to allow stepper and servo motors to be run.	Controlling the stepper and servo motors.	€20
Total				€84

6.4.4. Objective 4

“Define a standard communication protocol to allow loosely coupled communication between the mobile application and the telescope mount”

This objective was achieved. The protocol defined in Section 3.4 was implemented in both the application and the mount. Although it had to be altered slightly, with the manual commands being changed to bytes instead of JSON commands, this is still a loosely coupled protocol.

6.4.5. Objective 5

“Host the project in an open-source manner, so that the public can develop their own and expand upon it”

The project was hosted entirely on GitHub in one “monorepo”, with the complete code for the Arduino and Android application. Other resources are also included, such as the script used to clean the star data, test scripts and the schematics for the laser cut custom pieces. The 'readme' also contains photographs and a list of components required to build the mount.

In the near future, more documentation will be created and added to the repository, such as detailed instructions of how to build the mount, and documentation of the communication protocol.

6.5. Future Work

There are many more features of the project planned that could not be implemented within the final year project timeframe. These are detailed in this section.

6.5.1. Mount Upgrade

The mount can be further upgraded with more powerful motors, which would allow it to support a heavier viewing apparatus.

The main component that would have to be upgraded would be the stepper motor, as currently the 28BYJ-48 5V stepper motor is the main component limiting further load.

It could be changed to the NEMA 17 stepper motor (41), which would provide more torque.

The servo motors could also be upgraded by switching them from plastic geared to metal geared servo motors, such as the Diymore MG996R (42).



Figure 67: Comparison of plastic and metal geared servo motors. (42)

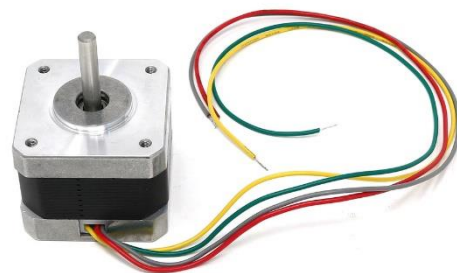


Figure 68: Nema 17 stepper motor. (41)

Table 11: Upgraded Mount Costs

Name	Qty.	Description	Usage	Approx. Cost
Nema 17 Stepper Motor	1	Motor with 360° range.	Moving the telescope horizontally.	€12
DiyMore MG996R Metal Gear Servo	2	Motor with 180° range.	Moving the telescope vertically.	€17
HC-05 Bluetooth Module	1	Allows the transmission and receipt of wireless serial data.	Allow the mount to receive Bluetooth commands.	€5
Arduino Uno Rev 3 AVR Development Board	1	Open-Source Microcontroller Board	Programmed to process serial commands and move motors accordingly.	€25
Miscellaneous Meccano Parts	N/A	Metal construction systems used to build small mechanical devices.	Constructing the mount mechanism.	€15
Adafruit Motor Shield v1	1	Board that sits on top of Arduino to allow stepper and servo motors to be run.	Controlling the stepper and servo motors.	€20
Total				€94

6.5.2. Satellite Tracking

One feature planned is the option for the telescope app is to slew to and track man-made satellites.

Due to the challenging nature of tracking the current positions of manmade satellites, the live positions of these in the form of equatorial coordinates will be retrieved from making a request to the public REST API N2YO(44). These will be displayed to the user in a list format similar to how stars and planets are currently displayed. When the satellite is selected, it will be tracked similar to how a planet is tracked, with the satellites current position being queried with an API call, and then the telescope orientation calculation being carried out.

Due to the use of a public API, this will be the only feature of the application that will require the user to have an internet connection to access.

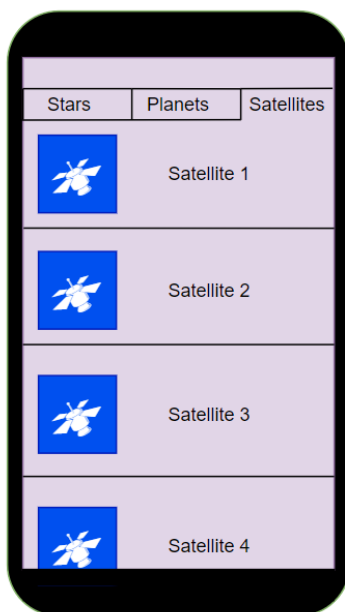


Figure 69: Object Select Screen Satellite Tab



Figure 70: Object Select Screen, Satellite Tab
(No Internet)

6.5.3. Google Voice & Red Light Filter

It is a common trend in astronomy applications to have an option to apply a red filter over the application. This preserves the astronomer's natural night vision, which is needed for looking at faint astronomical objects. This is a feature that could be added to the application, using the Android Studio theme support.

This idea of preserving night vision could also be taken one step further by implementing support for the Google Voice API. This would mean that the user could control the telescope using a command such as "Ok Google, slew telescope to Saturn", eliminating the need to look at the phone screen at all.

6.5.4. Slew Tweaking

One issue that was highlighted by testing and user evaluation was that the mount doesn't slew to objects with 100% accuracy, which could be attributed to numerous factors, such as motor hardware defects, mobile location and compass sensor accuracy and viewing device placement on mount.

One way this could be resolved, is by implementing a "tweak" feature, where after the mount slews to an object, the user can use directional buttons to manually get the object perfectly in view. This offset would be recorded for the current session and factored into future tracking calculations.

6.6. Personal Reflection

Personally, this was an extremely enjoyable, challenging and rewarding project for me.

I gained tremendous experience working on and managing this large-scale project, from formulating the idea all the way up to finalising the final product.

Being able to combine so many different technologies that I had gained experience in over the course of my degree into one working package was particularly rewarding to me.

Having limited experience in electronics and none in mechanical design did pose difficulty, as I underestimated how challenging it would be to construct the physical mount, leading to timelines and plans having to be adjusted. Thankfully, everything came together by the end of the project time frame, and I was able to successfully hit all of my project objectives.

As well as really honing and polishing my software development skills, I gained a lot of new skills due to stepping outside my comfort zone, such as learning how to debug electronics with a multimeter. Many soft skills were also developed, such as time management and report writing.

I am overall very proud of how this project turned out and I am excited to continue developing it in the future.

6.7. Resources

Links to the repository hosted on GitHub and a video demonstration can be found at the following address:

<https://linktr.ee/EasyScope>

Bibliography

1. Meade Instruments Telescopes, Solar Telescopes, Binoculars, Spotting Scopes [Internet]. [cited 2023 Dec 7]. Available from: <https://www.meade.com/>
2. Celestron - Telescopes, Telescope Accessories, Outdoor and Scientific Products [Internet]. [cited 2023 Dec 7]. Available from: <https://www.celestron.com/>
3. ASTROSETZ. The Nexstar 5 [Internet]. Astrosetz Dot Com. 2011 [cited 2023 Dec 7]. Available from: <https://astrosetz.com/2011/04/22/the-nexstar-5/>
4. Weasner M. Using the Meade ETX: 100 objects you can really see with the mighty ETX. Springer Science & Business Media; 2002.
5. Meade ETX90 AT 90mm GoTo Maksutov-Cassegrain Telescope [Internet]. [cited 2023 Dec 7]. Available from: <https://www.meade.com/meade-etx90-at-90mm-maksutov-cassegrain-telescope.pdp>
6. Chéreau F, Wolf AV, Zotti G, Hoffmann SM, Kabatsayev R, Boonplod W, et al. Stellarium [Internet]. Zenodo; 2023 [cited 2023 Dec 4]. Available from: <https://zenodo.org/record/8105940>
7. GmbH N. Meade Telescope AC 70/350 ETX-70 GoTo [Internet]. [cited 2023 Dec 7]. Available from: <https://www.astroshop.eu/telescopes/meade-telescope-ac-70-350-etx-70-goto/p,17690>
8. Meade Autostar™ User Manual. Meade Instruments Coporation; 1999.
9. Meet STELLINA, our observation station [Internet]. Vaonis. [cited 2023 Nov 30]. Available from: <https://vaonis.com/stellina>
10. GmbH N. Vaonis Smart Telescope AP 80/400 STELLINA [Internet]. [cited 2023 Dec 7]. Available from: <https://www.astroshop.eu/telescopes/vaonis-smart-telescope-ap-80-400-stellina/p,63162>
11. Putra GD, Pratama AR, Lazovik A, Aiello M. Comparison of energy consumption in Wi-Fi and bluetooth communication in a Smart Building. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC) [Internet]. 2017 [cited 2023 Dec 7]. p. 1–6. Available from: https://ieeexplore.ieee.org/abstract/document/7868425?casa_token=JZCx0KDZCwgAAAAA:yp65k0UmDEhEGIN5nWYfB8MoRzD1IOWN7e_TcmezHqJAyIO2g0IVajSnMzKf2tPlzusDzw
12. SkyPortal - Apps on Google Play [Internet]. [cited 2023 Dec 7]. Available from: <https://play.google.com/store/apps/details?id=com.celestron.skyportal&hl=en>
13. Singularity by Vaonis - Apps on Google Play [Internet]. [cited 2023 Dec 7]. Available from: <https://play.google.com/store/apps/details?id=com.vaonis.barnard&hl=en>
14. Leung M. GoTo_Telescope_Mount [Internet]. 2023 [cited 2023 Dec 8]. Available from: https://github.com/mattleung10/GoTo_Telescope_Mount
15. Astropy [Internet]. [cited 2023 Dec 8]. Available from: <https://www.astropy.org/>

16. Google Declared Kotlin As The Preferred Language For Android [Internet]. [cited 2023 Dec 8]. Available from: <https://www.mobileappdaily.com/knowledge-hub/kotlin-for-android-app-development>
17. StellarMate OS: StellarMate OS [Internet]. [cited 2023 Nov 27]. Available from: <https://www.stellarmate.com/products/stellarmate-os.html>
18. Astroberry Server [Internet]. [cited 2023 Nov 27]. Available from: <https://www.astroberry.io/>
19. Okigbo CA, Seeam A, Guness SP, Bellekens X, Bekaroo G, Ramsurrun V. Low cost air quality monitoring: comparing the energy consumption of an arduino against a raspberry Pi based system. In: Proceedings of the 2nd International Conference on Intelligent and Innovative Computing Applications [Internet]. New York, NY, USA: Association for Computing Machinery; 2020 [cited 2023 Dec 7]. p. 1–8. (ICONIC '20). Available from: <https://dl.acm.org/doi/10.1145/3415088.3415124>
20. Ferro E, Potorti F. Bluetooth and Wi-Fi wireless protocols: a survey and a comparison. IEEE Wireless Communications. 2005 Feb;12(1):12–26.
21. Bluetooth Module HC-05 Pinout, AT Commands & Arduino Programming .. [Internet]. [cited 2023 Dec 8]. Available from: <https://www.electronicwings.com/sensors-modules/bluetooth-module-hc-05->
22. Cross D. Astronomy Engine [Internet]. 2023 [cited 2023 Nov 27]. Available from: <https://github.com/cosinekitty/astronomy>
23. Sumaray A, Makki SK. A comparison of data serialization formats for optimal efficiency on a mobile platform. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication [Internet]. Kuala Lumpur Malaysia: ACM; 2012 [cited 2023 Dec 8]. p. 1–6. Available from: <https://dl.acm.org/doi/10.1145/2184751.2184810>
24. Discover INDI [Internet]. [cited 2023 Nov 26]. Available from: <https://www.indilib.org/what-is-indi/discover-indi.html>
25. 1. About ASCOM and Alpaca [Internet]. [cited 2023 Nov 26]. Available from: <https://ascom-standards.org/About/Overview.htm>
26. Ashley J. Astrophotography on the go. New York: Springer; 2014.
27. tobiassuper. What's the difference between an equatorial mount and an altazimuth mount? [Internet]. [cited 2023 Nov 15]. Available from: <https://www.skyatnightmagazine.com/advice/difference-equatorial-altazimuth-mount>
28. Teets D. The Mathematics of “Go To” Telescopes. The College Mathematics Journal. 2007 May;38(3):170–8.
29. The Horizontal Coordinate System [Internet]. [cited 2023 Nov 15]. Available from: <https://www.timeanddate.com/astronomy/horizontal-coordinate-system.html>
30. Molich R, Nielsen J. Improving a human-computer dialogue. Communications of the ACM. 1990;33(3):338–48.

31. Meccano [Internet]. [cited 2023 Dec 8]. Available from: <https://www.meccano.com/#>
32. Srivastava A, Bhardwaj S, Saraswat S. SCRUM model for agile methodology. In: 2017 International Conference on Computing, Communication and Automation (ICCCA) [Internet]. Greater Noida: IEEE; 2017 [cited 2023 Nov 13]. p. 864–9. Available from: <http://ieeexplore.ieee.org/document/8229928/>
33. BenoitBlanchon. ArduinoJson. [cited 2024 Apr 7]. ArduinoJson: Efficient JSON serialization for embedded C++. Available from: <https://arduinojson.org/>
34. LightBurn Software [Internet]. [cited 2024 Apr 10]. LightBurn Software. Available from: <https://lightburnsoftware.com/>
35. Stepper - Arduino Reference [Internet]. [cited 2024 Apr 1]. Available from: <https://www.arduino.cc/reference/en/libraries/stepper/>
36. AccelStepper: AccelStepper library for Arduino [Internet]. [cited 2024 Apr 1]. Available from: <https://www.airspayce.com/mikem/arduino/AccelStepper/>
37. Ardufocus [Internet]. [cited 2024 Apr 1]. Ardufocus | 28BYJ-48 Bipolar Mod. Available from: <https://ardufocus.com/howto/28BYJ-48-bipolar-hw-mod/>
38. Adafruit Learning System [Internet]. [cited 2024 Apr 1]. Adafruit Motor Shield. Available from: <https://learn.adafruit.com/adafruit-motor-shield/overview>
39. Warren Jr WH, Hoffleit D. The bright star catalogue. In: Bulletin of the American Astronomical Society, Vol 19, p 733. 1987. p. 733.
40. BSC5P - Bright Star Catalog [Internet]. [cited 2024 Apr 2]. Available from: <https://heasarc.gsfc.nasa.gov/W3Browse/star-catalog/bsc5p.html>
41. The Pi Hut [Internet]. [cited 2024 Apr 12]. NEMA 17 Stepper Motor - 42mm x 34mm (with bare wires). Available from: <https://thepihut.com/products/stepper-motor-nema-17-size-200-steps-rev-12v-350ma>
42. diymore 6PCS MG996R Metal Gear High Speed Torque Digital Servo Motor for RC Helicopter Airplane Car Boat Robot controls... : Amazon.co.uk: Toys & Games [Internet]. [cited 2024 Apr 12]. Available from: https://www.amazon.co.uk/diymore-Digital-Helicopter-Airplane-controls/dp/B09KZ8VTNB/ref=sr_1_1_sspa?dib=eyJ2IjoiMSJ9.soHF9LLa-wYvoRblzjl9uMHe4U63_JPrvMvHyBeP41UquHQ_ibRzTtxtNJAYjq1XcGNSgfkA9HSdyH-8MTPllbWNUltL_tclhAkj_hblQZivRJzpC-JF-F00-ACDW62gVSYsbTJ87KArDWTqBw7HOud9z3gGeK-5CD7cxrX6RL4d99gV7U2P8eES3_zVvv1iR9t_rd7iL4cSYQUYyF1jC5zJg5OZZUAc-n6vXdRO_8ByWp7KSOhfJfg5kkpfzsw31hK91i8TBqhZy8aqBljLSfsfu7qSyQPft5rNp9sHyiQ.WQmaC G13MWcTGyhflgUti6EtXVD46NzbPtXdw_DyByw&dib_tag=se&keywords=metal%2Bgear%2BServo&qid=1712950458&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1
43. Android Developers [Internet]. [cited 2024 Apr 12]. Styles and themes | Views. Available from: <https://developer.android.com/develop/ui/views/theming/themes>
44. N2YO.com API [Internet]. [cited 2023 Dec 8]. Available from: <https://www.n2yo.com/api/>

Appendix

System evaluation data acquired through a Google Form

Name Provided	Q1	Q2	Q3	Q4	Q5	Q6
Brandon Bruce Benedict Byrne	5	5	5	5	5	Yes
Bien Managbanag	5	5	4	5	5	Yes
Bill Lee	5	5	5	5	5	Yes
David Clarke	5	4	4	5	5	Yes
Aislin Woods O'Kelly	4	5	5	5	5	Yes