Powered Newtonian Telescope

Senior Project Final Report – Winter 2023

Submitted to: Troy Scevers
Submitted by: Nathan Wiley

Version: 4.1

Submitted in partial fulfillment of the requirements of CST472.

ESET Senior Project, Oregon Institute of Technology.

© 2023, Nathan D. Wiley. All rights reserved.

REVISION HISTORY

Version:	**Revision Date:**	**Revision Description:**	**Author:**
0.0	10/14/2022	Rough draft	Nathan Wiley
1.0	10/26/2022	Requirements complete	Nathan Wiley
2.0	10/27/2022	Added comparison matrices and part selection	Nathan Wiley
3.0	12/12/2022	Reformatted as report, added memos, diagrams, and table of figures	Nathan Wiley
4.0	3/7/2023	Added wiring diagrams, notes, purchased parts, and code sections	Nathan Wiley
4.1	3/22/2023	Final Report for Winter: Reformatted to include appendix. Added software documentation.	Nathan Wiley

TABLE OF CONTENTS

TABLE OF FIGURES

CONCEPTUAL OVERVIEW

The basic idea of this project is to create a telescope that will adjust what it points at with an embedded device. The movement would be calibrated to coordinate system, and commands could be sent to the device over a network that would allow the telescope to point to, and track, a given coordinate. The main requirements are necessary for the project to be considered complete. The stretch goals are what the project would look like if all main requirements are met, and the time and resources to achieve them are available.

1. REQUIREMENTS SPECIFICATION

1. **Shall pivot a telescope.**
    a. Telescope shall pivot at a rate of at least 0.25 degrees per minute.
    b. Shall be accurate enough to point the telescope within 3600 arcseconds (1 degree) of the desired angle.
2. **Shall be able to keep accurate time.**
    a. Shall keep onboard time within 2 seconds of time kept at www.time.gov.
3. **Shall use a control system.**
    a. The control system shall compute the local sidereal time within 2 seconds of real time (co-dependent on req. 2a).
    b. Control system shall reference a database of stellar objects.
    c. Shall receive instructions over Ethernet.
    d. Shall execute movement instructions (co-dependent on req. 4b).
4. **Shall have a network interface.**
    a. Shall use Ethernet and ethernet protocols.
    b. Shall receive movement instructions over Ethernet.
5. **Shall use a display.**
    a. Display shall signal at least two of the states the system is in (I.E. Moving, or Idle).

1.1 STRETCH GOALS

1. **May have a camera.**
    a. Should have a resolution of at least 2MP (1080p).
    b. Should have functionality to save to an SD card.
    c. May include networked file transfer of images.
    d. May include counter rotation functionality (hardware or software).
2. **May use RTC module.**
3. **May include GPS module.**
4. **May include Wi-Fi in addition to Ethernet.**
5. **May include cooling system for primary mirror.**
    a. May be a fan, if so, should move air at least 10 CFM.

2. PART SELECTION AND DESCISION MATRICES

In each section below, a comparison matrix for each component will be shown followed by discussion and selection of the component. The discussion will highlight the reasons and thought process behind the choice of the component. Each selected part will be compiled at the end of the report with web links to their respective online stores. Ordered parts will have the estimated arrival date shown.

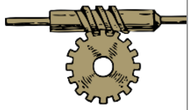
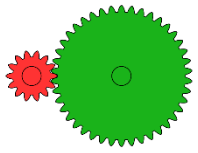
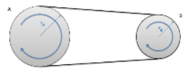
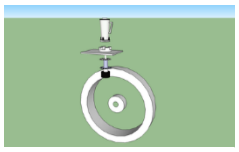
2.1 DEVELOPMENT BOARD

Picture:			
Board Name:	BeagleBone Black - Rev C	Raspberry Pi 3 Model B V1.2	Raspberry Pi 4 Model B
Manufacturer:	Texas Instruments	Raspberry Pi Foundation	Raspberry Pi Foundation
Processor Clock Speed:	AM3358 @ 1GHz	BCM2837 @ 1.2 GHz	BCM2711 @ 1.5 GHz
Memory:	512MB DDR3L DRAM 4GB 8-bit eMMC Flash	1GB RAM	4GB LPDDR4-2400 SDRAM
Network Interface:	Ethernet	WiFi, Ethernet	Dual Band WiFi, Gigabit Ethernet
General Purpose I/O pins:	2x 40 pin headers,	1x 40 pin header	1x 40 pin header
Extra Features:	NEON floating-point accelerator, microHDMI, 1x USB 2.0	HDMI, 4x USB 2.0	HDMI, 2x USB 2.0, 2x USB 3.0
Cost:	$65.99	$0 (Already owned)	$168.88

Discussion: The BeagleBone's main attractive features were the high pin count and the floating-point accelerator, it could prove useful when driving stepper motors at high pulse rates. However, since this project should only include a maximum of two motors and a few peripherals, the pins would be excessive. The Raspberry Pi 4's upgrades over the 3 are nice, but they don't justify themselves with the current price. The Raspberry Pi 3 seems to be the sweet spot with an included RTC module, WiFi, Ethernet, SD card slot, and potential camera connector.

Selection: Raspberry Pi 3 Model B V1.2

2.2 GEARING

Picture:					
Drive Criteria:	Worm Gear	Spur Gear	Belt Drive	Friction Drive	Direct Drive
Positioning:	good	good	good	very good	good
Tracking:	good	fair	good	excellent	good
Stiffness:	poor	poor	very low	very high	fair/good
Smoothness:	good	fair	fair/very good	excellent	good
Gear Ratio:	very high	low	low/moderate	high	n/a
Efficiency:	very low	high	high	very high	very low
Zero Backlash:	no	no	yes	yes	n/a
Periodic Error:	small/mod	large	small	very small	very low
First Period:	2-4 min	1/tooth	2-4 hours	1 hour	n/a
Cost:	high	moderate	low	moderate	very high

Discussion: The table above is from DFM Engineering Inc., it shows the pros and con of the different ways to drive a telescope. Initially, the worm gear was going to be the implemented drive type because it offers very high torque with precise control, but the prices were too high. Another reason against the worm drive, as well as the spur gear and friction drive, is that it requires very precise centering, and because this is an embedded project and not a manufacturing or mechanical project, there is no guarantee of precision. The belt drive can be easily adjusted with tensioners, which allow for some error in the centering of the pulleys. The belt drive also has zero backlash, which will help the telescope to remain stable and not rattle or vibrate when operating. Because of these reasons, and the fact that it is cheaper than the other options, the belt drive is going to be what connects directly to the vertical and horizontal movement axis.

Selection: Belt Drive

Note: After working with a GT2 timing belt and pulleys, the telescope would sway and slip and was easy to misalign by hand. A chain would sway and slip less and would prove to be a sturdier control method and should provide similar benefits to the belt drive.

2.3 MOTORIZATION TYPE

Picture:			
Drive Name:	Stepper Motor	DC Motor	Servo Motor
Speed Control Method:	Individual steps, micro-stepping	Voltage	PWM
Torque at low RPM:	High	Low	High
# of wires	4 – 8	2	3
Continuous rotation:	Yes	Yes	No
General Cost:	Intermediate	Low	High

Discussion: Above is a table of generalization about the different motor types. Since the telescope require to be precisely controlled with high torque, the stepper motor is the clear winner once cost is factored in. Now that the type of motor is selected, an individual stepper motor still needs to be selected. Below is a table of different stepper motors and their pros and cons.

Selection: Stepper Motor

2.4 STEPPER MOTORS

Picture:			
Model Number:	17HS19-1684D-PG100-E1000	23HS22-2804S-HG50	17HS15-1584S-PLMG100
Stepper Class:	NEMA 17	NEMA 23	NEMA 17
Gear Type and Ratio:	Economy Planetary, 100:1	High Precision Planetary, 50:1	High Precision Planetary, 100:1
Backlash:	<= 1 deg (3600 arcseconds)	<=1.6 deg (5760 arcseconds)	<=0.83 deg (3000 arcseconds)
Max Torque (With Gearbox):	4Nm (566.56oz.in)	20Nm (2832.8oz.in)	5Nm(708oz.in)
Encoder:	Yes	No	No
Current:	1.68A	2.8A	1.58A
Cost:	$63.08	$90.81	$37.16

Discussion: For stepper motor selection, a high gear ratio is a must because it must have enough torque to move the scope and enough steps per rotation that it provides smooth movement after being geared down. The second option doesn't have as much precision as the other two. The first option includes an encoder on the back of the stepper motor itself which seems attractive; however, the higher backlash and price is higher than the third option while providing less torque. If the need for an encoder arises, then it can be added on after. The third option's gearbox also has a waterproof rating of IP54 which protects the gears from dust and moisture.

Selection: 17HS15-1584S-PLMG100, NEMA 17

2.5 STEPPER MOTOR DRIVER

Picture:			
Driver Name:	DM542T	DRV8825	A4988
Max Rated Amperage:	4.20A	2.2A	2A
Micro stepping:	Up to 1/256	Up to 1/32	Up to 1/16
Price:	$20.99	$18.95	$14.45

Discussion: The selected motor requires 1.58A to run, and the A4988 could only provide that if it was cooled with heatsinks and fans. The DRV8825 could handle that load better, but not with much room to spare, so it would likely get hot too. The DM542T can easily handle the load with high micro stepping resolution as well.

Selection: DM542T

2.6 SELECTED & PURCHASED PARTS

Part:	Link:	Order Date:	Price:
Raspberry Pi	https://thepihut.com/products/raspberry-pi-3-model-b?src=raspberrypi	N/A (Already Owned)	0
Nema 17 Stepper Motor	https://www.omc-stepperonline.com/nema-17-stepper-motor-l-40mm-gear-ratio-100-1-plm-series-planetary-gearbox-17hs15-1584s-plmg100	10/28/2022	$37.16
DM542T Digital Stepper Driver	https://www.omc-stepperonline.com/digital-stepper-driver-1-0-4-2a-20-50vdc-for-nema-17-23-24-stepper-motor-dm542t	10/28/2022	$20.99
Power Supply	https://www.amazon.com/dp/B08GFQZFC1?ref=ppx_yo2ov_dt_b_product_details&th=1	11/28/2022	$39.99
Timing Belt	https://www.amazon.com/dp/B07JKT5BZQ?psc=1&ref=ppx_yo2ov_dt_b_product_details	02/05/2023	$14.99
8mm Shaft	https://www.amazon.com/dp/B01B27MJC6?psc=1&ref=ppx_yo2ov_dt_b_product_details	02/05/2023	$7.99
8mm Bearings	https://www.amazon.com/dp/B07TNNX9YX?psc=1&ref=ppx_yo2ov_dt_b_product_details	02/05/2023	$6.99
5mm to 8mm Coupling	https://www.amazon.com/dp/B07RMZCLZ3?psc=1&ref=ppx_yo2ov_dt_b_product_details	02/05/2023	$12.99
3.3V to 5V Level Shifter	https://www.amazon.com/dp/B06XWVZHZJ?psc=1&ref=ppx_yo2ov_dt_b_product_details	02/05/2023	$8.99

Note: 3.3V to 5V level shifter did not work as intended. Instead, 2222NPN transistors were used to convert the 3.3v from the Raspberry Pi to drive the stepper driver to 5V or ground.

3. BLOCK DIAGRAMS

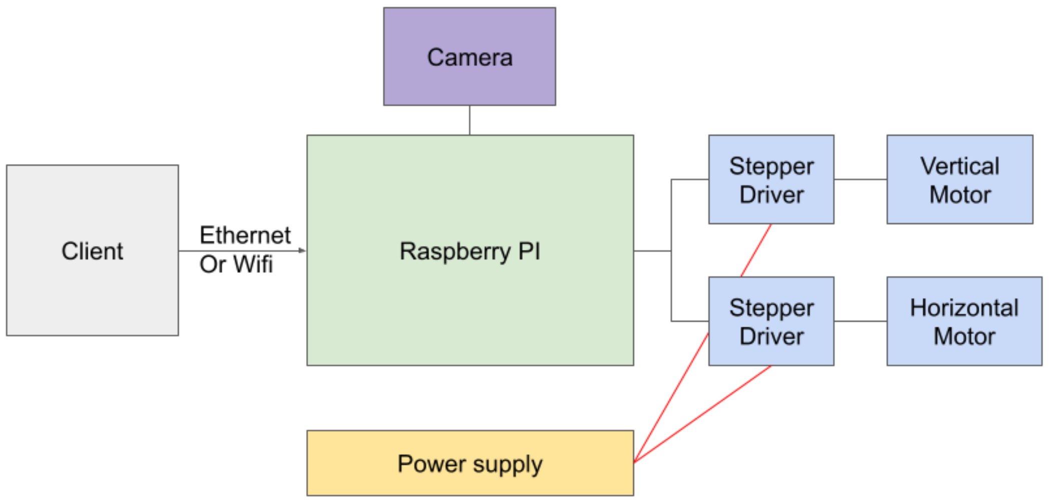


Figure 1: Block Diagram

Above is a block diagram of the project. This is a very general overview to show how each part will connect with one another. Below is the detailed wiring diagram showing each connection made and the respective parts they connect to. This diagram is representative of the build as of the handheld controller demo. This is subject to change for the final product.

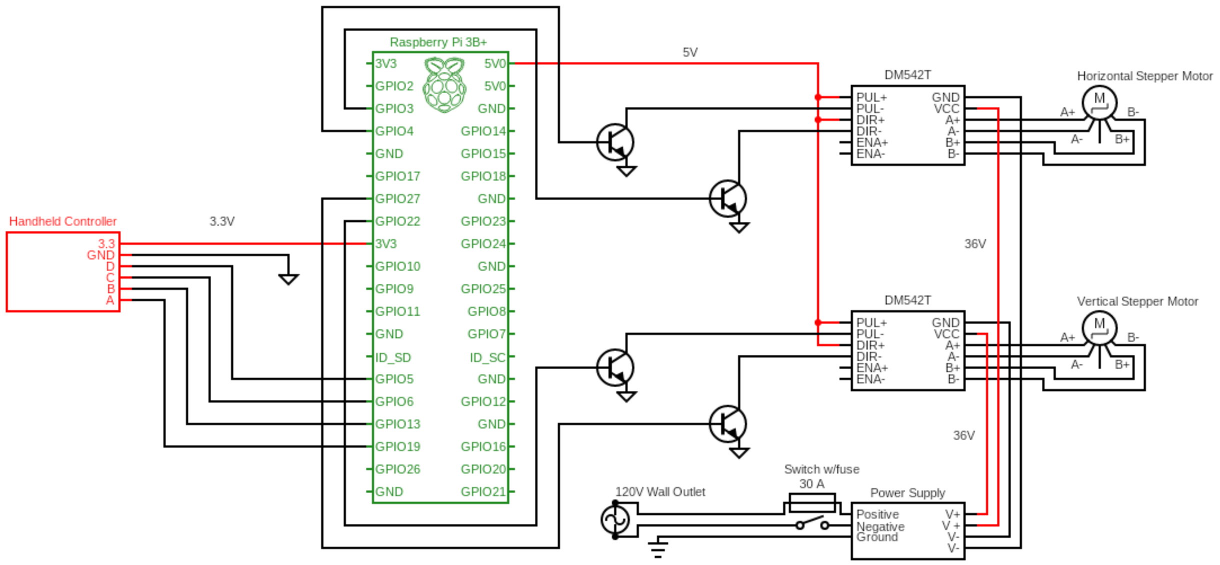


Figure 2: Wiring Diagram

4. SOFTWARE DEVELOPMENT ENVIRONMENT

Since the project is running in a Linux environment, for some it might seem natural to adopt IDEs like Geany or Thonny, and editors such as VIM or Emacs, while using python as the primary language. These options would take valuable time to learn and get accustomed to. Instead, since it is familiar, Visual Studio, C++, and GitHub will be used. The Home PC will be configured to remotely build, debug, and execute code through Visual Studio to the Raspberry Pi over SSH. The source control is set up through GitHub which is accessed through Visual Studio as well. This will result in much faster software development as it allows for the write, build, debug, test cycle to be accomplished all in one place, without the need for the Home PC to be directly connected to the Raspberry Pi. This will not only speed up the development time, but also allow the device's code to be updated on the fly once it is complete.

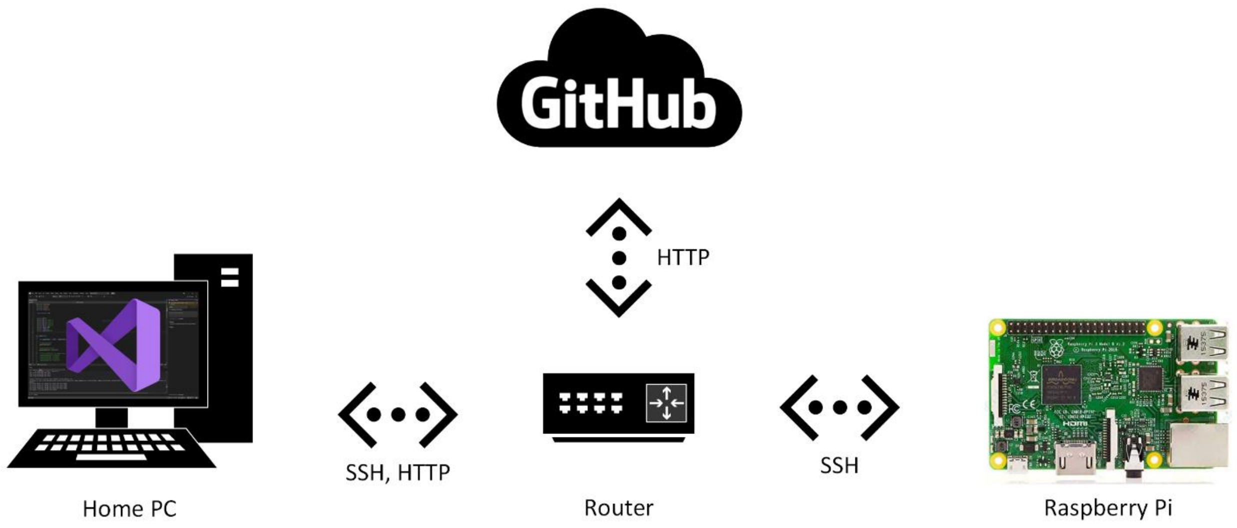


Figure 3: Software Development Environment

This system works quite well. The process for switching between the home PC and the laptop by pushing to GitHub through Visual Studio's source control integration and pulling from the other device takes about a minute's time. This allowed rapid development of code without the need to be tied down to one place. Being able to program and debug remotely has been a huge advantage as code was written and tested for the time calculations, motor controls, and coordinate functions without removing the Raspberry Pi from the telescope or connecting to it directly every time it was built.

5. SOFTWARE DEVELOPMENT

Late into the winter term is when the software began to take shape. In order to use the Raspberry Pi’s GPIO pins the options were to use the deprecated but more documented WiringPi library, or the updated “PIGPIO” library. The use of WiringPi was attempted, but several issues were encountered along the way. One main advantage that PIGPIO has is a much higher clock speed on the GPIO pins – up to 20KHz. This will be useful when micro stepping the motors because they have to be stepped very high speeds.

5.1 SOFTWARE OVERVIEW

The goal of the project’s software is to know where the telescope is in relation to the earth and calculate where it needs to point in order to view the target.

To do this, it must know:

- The telescope’s Latitude and Longitude.
- Current time in UTC/GMT. See Appendix 5.3
- Number of fractional Julian Days (days since noon Universal Time on January 1, 4713 BC). See Appendix 5.4
- The Earth’s rotation angle (ERA). See Appendix 5.5
- The Greenwich Mean Sidereal Time (GMST). See Appendix 5.6
- The Local Mean Sidereal Time (LMST) using Lat/Long Coordinates to offset GMST. See Appendix 5.7
- The target object’s Right Ascension (RA) and Declination (Dec).
- The current Altitude / Azimuth (Alt/Az) position where the telescope is pointing.

After all of the above information is gathered, then it needs to calculate where to point the telescope in its local Alt/Az coordinate sphere. This function can be found in Appendix 5.8.

The plan is to split the time calculations and the coordinate calculations into separate classes with static functions to be used in the main control function.

As of the end of the Winter 2023 term, all functions have been written except for the last listed one that knows where it is pointing. The plan is to calibrate it by manually controlling the telescope to align with a star and then align it’s Alt/Az coordinate sphere to match. In theory, this could be done by leveling the telescope and aligning with one star or using multiple stars without leveling.

The code for this project can be publicly accessed on GitHub here: *https://github.com/ninjaiceflame/Stepper*

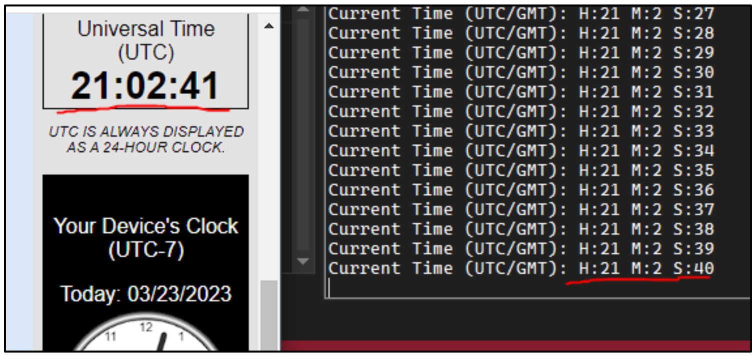
5.2 SIDEREAL CLASS

This class is responsible for all time calculations and includes utility functions for converting relevant data types to each other. The first six in the table below are utility functions to help convert or display units of time or angle. The last five in the list are the main calculation functions described in 5.1 Software overview.

Function	**Purpose**
`displayTmMMDDYYYY`	Prints Tm struct in MM:DD:YYYY format to console
`displayHHMMSS`	Prints hourMinuteSeconds struct in HH:MM:SS format to console
`hmsToDeg`	Converts hourMinuteSeconds struct to degrees (double)
`degToHms`	Converts degrees (double) to hourMinuteSeconds struct
`degToDms`	Converts degrees (double) to degreeMinuteSeconds struct
`dmsToDeg`	Converts degreeMinuteSeconds struct to degrees (double)
`getGMT`	Returns TM struct with GMT Month, Day, Year, and Hour, Minute, Seconds
`getLMST`	Returns a double containing decimal degrees of Local Mean Sidereal Time
`getJulianDate`	Calculates current Julian date based on getGMT() and account for leap years
`getERA`	Returns a double containing the ERA in radians
`getERAcomplex`	Another way to calculate, but takes longer

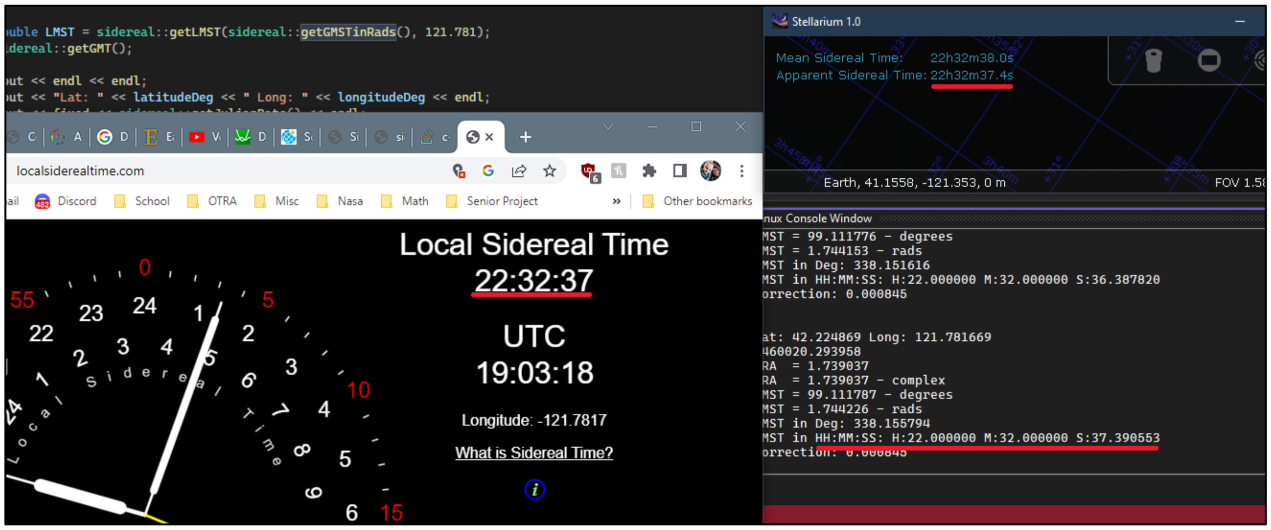
The combination of the above functions results in an accurate measurement of UTC/GMT (See figure 4.). This is in fulfillment of requirement 2A.

Calculation for Local Sidereal Time that has sub-second accuracy and proven below in Figure 5. This is in fulfillment of requirement 3B.


Universal Time (UTC)
21:02:41
UTC IS ALWAYS DISPLAYED AS A 24-HOUR CLOCK.
Your Device's Clock (UTC-7)
Today: 03/23/2023

```
Current Time (UTC/GMT): H:21 M:2 S:27
Current Time (UTC/GMT): H:21 M:2 S:28
Current Time (UTC/GMT): H:21 M:2 S:29
Current Time (UTC/GMT): H:21 M:2 S:30
Current Time (UTC/GMT): H:21 M:2 S:31
Current Time (UTC/GMT): H:21 M:2 S:32
Current Time (UTC/GMT): H:21 M:2 S:33
Current Time (UTC/GMT): H:21 M:2 S:34
Current Time (UTC/GMT): H:21 M:2 S:35
Current Time (UTC/GMT): H:21 M:2 S:36
Current Time (UTC/GMT): H:21 M:2 S:37
Current Time (UTC/GMT): H:21 M:2 S:38
Current Time (UTC/GMT): H:21 M:2 S:39
Current Time (UTC/GMT): H:21 M:2 S:40
```

Figure 4: UTC/GMT measured within 1 second.



```
uble LMST = sidereal::getLMST(sidereal::getGMSTinRads(), 121.781);
idereal::getGMT();

ut << endl << endl;
ut << "Lat: " << latitudeDeg << " Long: " << longitudeDeg << endl;
```

```
MST = 99.111776 - degrees
MST = 1.744153 - rads
MST in Deg: 338.151616
MST in HH:MM:SS: H:22.000000 M:32.000000 S:36.387820
orrection: 0.000845


at: 42.224869 Long: 121.781669
460020.293958
RA  = 1.739037
RA  = 1.739037 - complex
MST = 99.111787 - degrees
MST = 1.744226 - rads
MST in Deg: 338.155794
MST in HH:MM:SS: H:22.000000 M:32.000000 S:37.390553
orrection: 0.000845
```

Figure 5: LMST measured with sub-second accuracy.

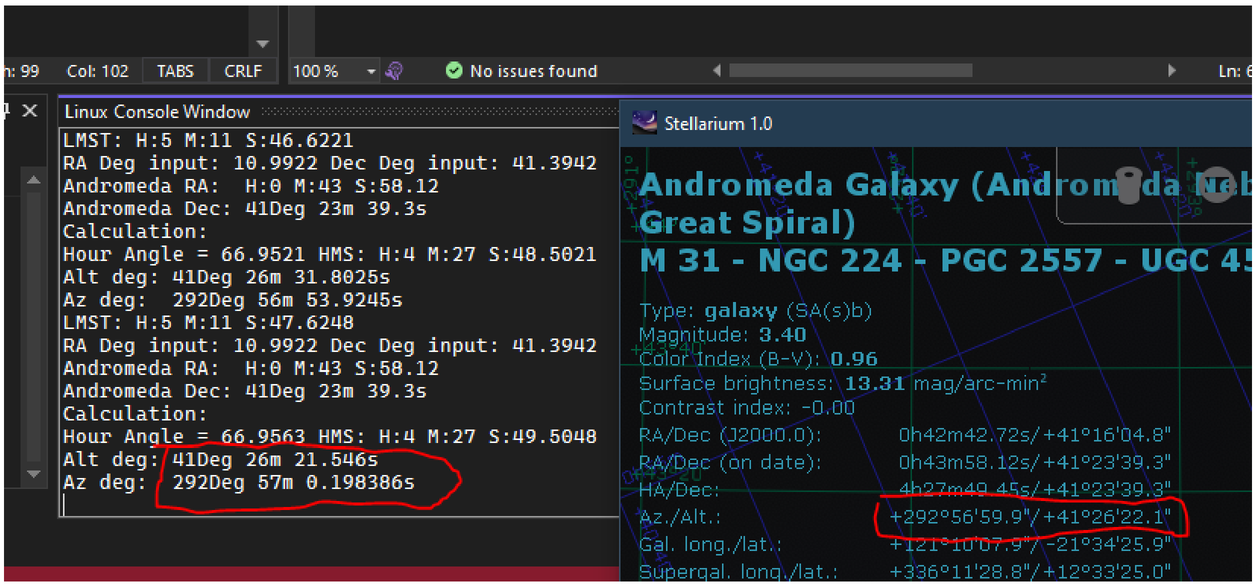
5.3 COORDINATE CLASS

This class is used to do the calculations for the equatorial coordinate sphere which is where stars are relative to the Earth in general (RA/Dec), and the local horizontal coordinate sphere of the telescope, which is where the stars are relative to the telescope at the current time (Alt/Az).

As of now, this class only has one function:

```
equatorialToLocal(double RA, double Dec, twoAxisDeg myPositionDeg);
```

which provides conversion from equatorial Right Ascension / Declination to local Altitude / Azimuth coordinates. More functions maybe be needed if a GPS or Accelerometer is ever added.



```
LMST: H:5 M:11 S:46.6221
RA Deg input: 10.9922 Dec Deg input: 41.3942
Andromeda RA:  H:0 M:43 S:58.12
Andromeda Dec: 41Deg 23m 39.3s
Calculation:
Hour Angle = 66.9521 HMS: H:4 M:27 S:48.5021
Alt deg: 41Deg 26m 31.8025s
Az deg:  292Deg 56m 53.9245s
LMST: H:5 M:11 S:47.6248
RA Deg input: 10.9922 Dec Deg input: 41.3942
Andromeda RA:  H:0 M:43 S:58.12
Andromeda Dec: 41Deg 23m 39.3s
Calculation:
Hour Angle = 66.9563 HMS: H:4 M:27 S:49.5048
Alt deg: 41Deg 26m 21.546s
Az deg:  292Deg 57m 0.198386s
```

Andromeda Galaxy (Andromeda Neb
Great Spiral)
M 31 - NGC 224 - PGC 2557 - UGC 4

Type: **galaxy** (SA(s)b)
Magnitude: **3.40**
Color Index (B-V): **0.96**
Surface brightness: **13.31** mag/arc-min^2
Contrast index: -0.00

RA/Dec (J2000.0):	0h42m42.72s/+41°16'04.8"
RA/Dec (on date):	0h43m58.12s/+41°23'39.3"
HA/Dec:	4h27m49.45s/+41°23'39.3"
Az./Alt.:	+292°56'59.9"/+41°26'22.1"
Gal. long./lat.:	+121°10'07.9"/-21°34'25.9"
Supergal. long./lat.:	+336°11'28.8"/+12°33'25.0"

Figure 6: Alt/Az calculated from RA/Dec with sub-degree accuracy.

The above screenshot is a demonstration of the Alt/Az calculation given the RA/Dec of the target. This is in fulfillment of requirement 3B.

The header file of this class also includes three definitions for three types of structs:

`twoAxisDeg, twoAxisRads, and twoAxisDms`

Each one holds two data members for the horizontal and vertical coordinate in a coordinate sphere. They are largely the same but are separately defined so one doesn't get accidentally assigned to another through an "=" operator.

5.4 MOTOR CONTROL

The code for driving the motors to the calculated Alt/Az coordinates has not been written, but basic code has been written to drive the motors with a step period of 200 microseconds which is a frequency of 5KHz. 5000 steps per second. To find steps per revolution: 1:8 microstepping yields 1600 microsteps per revolution, with a gear ratio of 100:1 from the planetary gearbox and a 30/12 (2.5:1) ratio connecting the gearbox to the telescope gives 400,000 steps per revolution.

400,000 steps per revolution / 5000 steps per second = 80 seconds per revolution.

360 degrees / 80 seconds per revolution = 4.5 degrees per second This is in fulfillment of requirement 1A.

This calculation also provides the theoretical accuracy of the telescope which is:

360 degrees/400,000 steps = 0.0009 degrees. This is in fulfillment of requirement 1B.

5.5 SOFTWARE VERSIONS AND SOURCE CONTROL

All code for this project is written in C++ version C++11.

Microsoft Visual Studio Community 2022 (64-bit) - Version 17.4.2.

The Raspberry Pi is running on Raspbian GNU/Linux 11 (bullseye).

The library used to interface with the GPIO pins is "PIGPIO" version 79 and can be found here: https://github.com/joan2937/pigpio/

The version of the GCC/G++ compiler is "g++ (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110"

On PC: git version 2.39.0.windows.1

On Raspberry Pi: git version 2.30.2.

All code for this project is currently located on GitHub: https://github.com/ninjaiceflame/Stepper

6. PHYSICAL CONSTRUCTION

Most of the documentation on the physical construction of the telescope is in the memos, so this section will be a brief overview of the progress so far. The telescope parts that were already acquired were the primary and secondary mirrors, tube, eyepiece, and star finder. The parts that needed to be built were the base, vertical and horizontal axis, and tube rings to hold the tube or body of the telescope. Autodesk Inventor was used to design plywood and 3D printed parts to accomplish this.

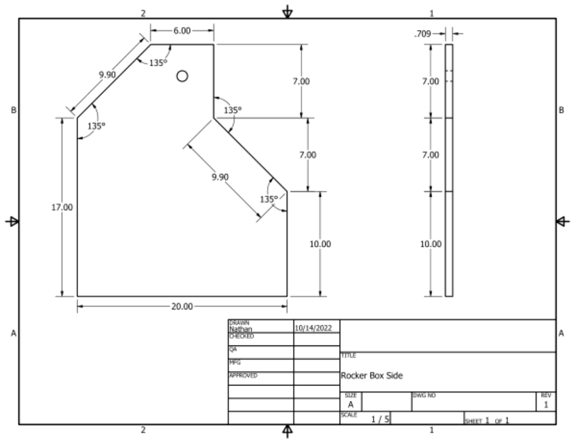


Figure 7: Side Panels

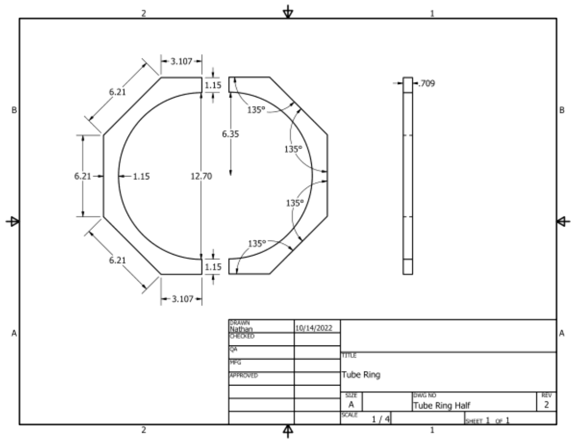


Figure 8: Tube Rings

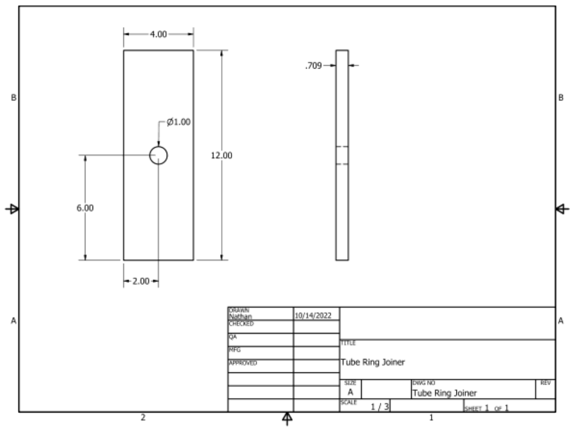


Figure 9: Tube Ring Joiner

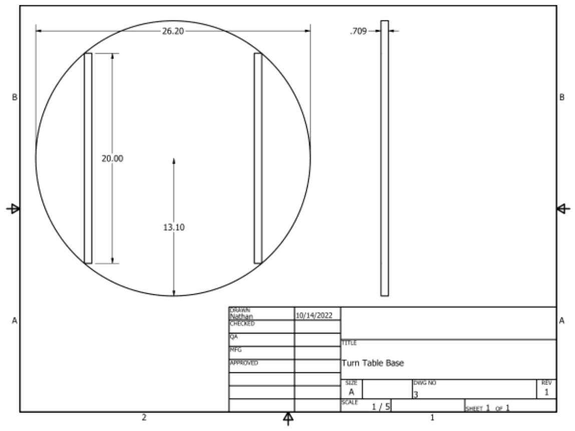


Figure 10: Turntable Base

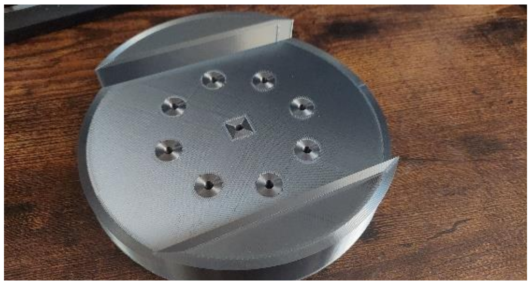


Figure 12: 3D Printed Bearing

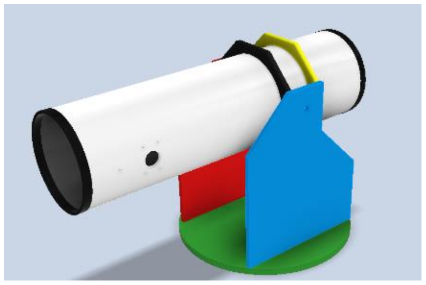


Figure 11: Autodesk Inventor Assembly

6.1 BELT DRIVE TO CHAIN DRIVE CONVERSION

After assembling the belt and pully system, testing showed that the thin GT2 belt wasn't suited for this application. The small radius of the pulley proved to not provide enough mechanical advantage to control the telescope. It would sway and stretch and the belt would slip on the small teeth of the pulley. This aspect needed a resign that could provide much more leverage and wouldn't slip.

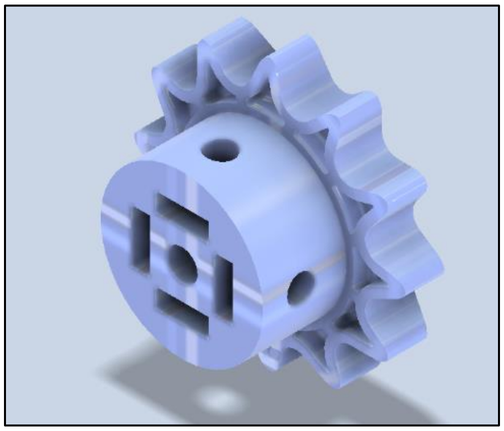


Figure 13: Rendering of 12 tooth gear.

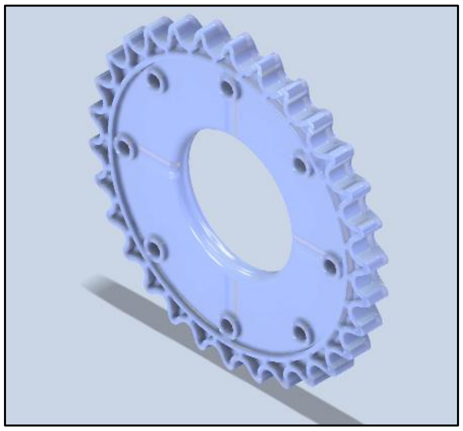


Figure 14:Rendering of 30 tooth gear.

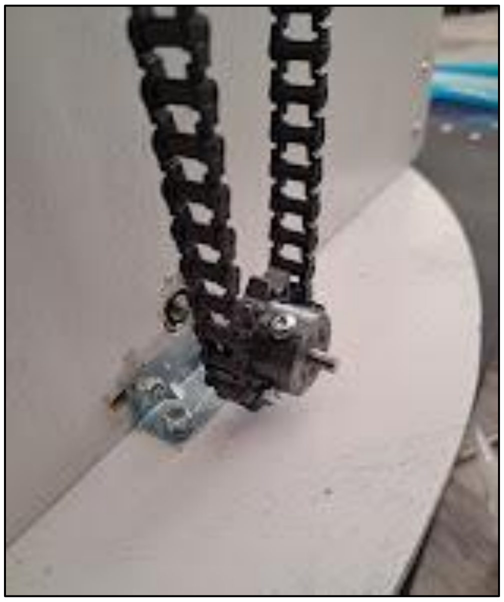


Figure 15: 3D printed 12 tooth gear in action.

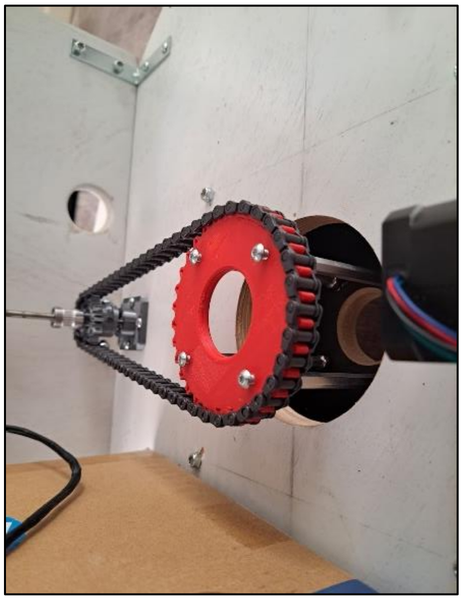


Figure 16:: 3D printed 30 tooth gear in action.

7. TIMELINE

This term’s progress was substantial with only moderate delays. Much time and effort went into staying on track. The main goals for this term were achieved and that leaves refinement, testing, and work on stretch goals for next term.

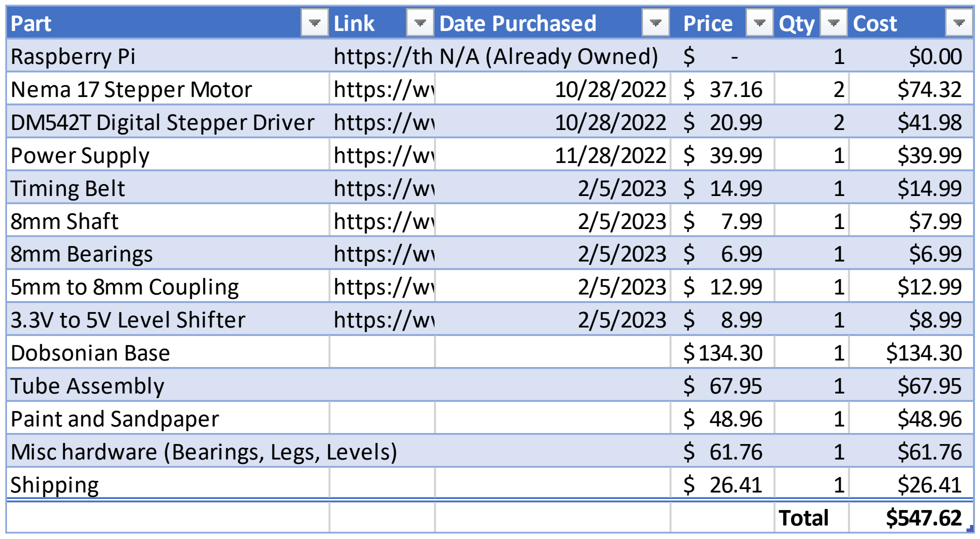
Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
1. Finish non-motorized telescope assembly.	🟩	🟩	🟩								
a) Drill holes into turntable and assemble uprights.	🟩	🟩	🟩								
b) Drill holes and screw in horizontal struct piece.	🟩	🟩	🟩								
c) Paint unfinished pieces				🟩	🟩						
2. Create gearing system.				🟩	🟩						
a) Design and print large pulley gear.				🟩	🟩						
b) Order smaller gear and timing belts.				🟩	🟩						
3. Connect power adapter to power supply.				🟩	🟩	🟩					
4. Connect power supply to stepper drivers.				🟩	🟩	🟩					
5. Connect level shifter for the 3.3V I/O of Raspberry Pi to 5V of the stepper motor driver.				🟩	🟩	🟩					
6. Develop software to drive motors.				🟩	🟩	🟩	🟩				
a) Develop software to drive 2 motors at once.					🟩	🟩	🟩				
7. Develop software to aim telescope.						🟩	🟩	🟩	🟩	🟩	🟩
a) Find the math behind aiming 2 axis control in 3D geometric space.						🟩	🟩	🟩	🟩	🟩	🟩
b) Apply the math to the code for driving the motors.								🟩	🟩	🟩	🟩
8. Develop software to track a coordinate as the earth rotates.								🟩	🟩	🟩	🟩
a) Develop software for calculating sidereal time.								🟩	🟩	🟩	🟩
9. Create or purchase board and electronic housing.				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 17: Winter 2023 Gantt chart

Task	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
Design Frame	🟩								
Order Parts for Frame	🟩								
Build Frame	🟩								
Design Hardware	🟩	🟩	🟩						
Order Hardware	🟩	🟩	🟩						
Assemble Hardware		🟩	🟩						
Develop Motor Software			🟩	🟩	🟩				
Develop Tracking Software				🟩	🟩	🟩			
Develop Networking Capabilities					🟩	🟩			
Motor Testing							🟥	🟥	🟥
Motor Control Testing							🟥	🟥	🟥
Tracking Testing							🟥	🟥	🟥
Stretch Goals (If ahead of schedule)	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥
Documentation	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥

Figure 18: 2022-2023 Overview Gantt chart

8. COST



Part	Link	Date Purchased	Price	Qty	Cost
Raspberry Pi	https://th	N/A (Already Owned)	$ -	1	$0.00
Nema 17 Stepper Motor	https://w	10/28/2022	$ 37.16	2	$74.32
DM542T Digital Stepper Driver	https://w	10/28/2022	$ 20.99	2	$41.98
Power Supply	https://w	11/28/2022	$ 39.99	1	$39.99
Timing Belt	https://w	2/5/2023	$ 14.99	1	$14.99
8mm Shaft	https://w	2/5/2023	$ 7.99	1	$7.99
8mm Bearings	https://w	2/5/2023	$ 6.99	1	$6.99
5mm to 8mm Coupling	https://w	2/5/2023	$ 12.99	1	$12.99
3.3V to 5V Level Shifter	https://w	2/5/2023	$ 8.99	1	$8.99
Dobsonian Base			$ 134.30	1	$134.30
Tube Assembly			$ 67.95	1	$67.95
Paint and Sandpaper			$ 48.96	1	$48.96
Misc hardware (Bearings, Legs, Levels)			$ 61.76	1	$61.76
Shipping			$ 26.41	1	$26.41
				Total	**$547.62**

The NRE cost of this project is $547.62

The engineering cost for this term is as follows:

15 hrs./week * 10 weeks = 150 hrs. @ $50/hr. = $7,500

This added to the part cost = $8,047.62 for this term.

CONCLUSION

Substantial progress was made this term including finishing the physical construction, assembling the hardware, developing accurate timekeeping and coordinate calculations, and controlling the motors. All of these completed tasks will leave room for work stretch goals and testing for text term.

REFERENCES

artificial satellite - Time, UTC, Julian Date, TLE epoch - how are they related quantitatively? (n.d.). Space Exploration Stack Exchange. Retrieved March 23, 2023, from https://space.stackexchange.com/questions/13809/time-utc-julian-date-tle-epoch-how-are-they-related-quantitatively

Capitaine, N., & Wallace, P. (2009). *Implementation Implementation of of the the IAU IAU 2000 2000 definition definition of of UT1 UT1 in in astronomy astronomy*. https://www.atnf.csiro.au/iau-comm31/pdf/2009_IAUGA_JD6/JD06_capitaine_wallace.pdf

DFM Engineering, Inc. (2005, March 9). *Engineering Article: Comparing Telescope Drive Technologies*. Www.dfmengineering.com. https://www.dfmengineering.com/news_telescope_gearing.html

Duffett-Smith, P. (1989). *Practical Astronomy with your Calculator*. Cambridge University Press.

Miller, G. (n.d.). *Convert RA/DEC to Alt/Az*. Astrogreg.com. Retrieved March 23, 2023, from https://astrogreg.com/convert_ra_dec_to_alt_az.html

Texereau, J. (1957). *How to Make a Telescope*.

APPENDIX

5.1 LIBRARIES USED AND GPIO DEFINES FOR DEMO CODE:

```
#include <iostream>
#include <thread>
#include <chrono>
#include <pigpio.h>
#include <stdio.h>

using namespace std;

//Stepper motor 1 - Horizontal
#define ENA1 2
#define DIR1 3
#define PUL1 4

//Stepper motor 2 - Vertical
#define ENA2 17
#define DIR2 27
#define PUL2 22

//#define _DELAY 64
#define _DELAY 100
#define _STEPS 1600
#define _GEAR_RATIO 100 * 2

//Controller pins
#define D_BTN 5
#define C_BTN 6
#define B_BTN 13
#define A_BTN 19
…
```

5.2 INITIALIZATION CODE AT THE START OF MAIN:

This initializes the GPIO pins on the Raspberry Pi and sets up the stepper drivers. It also creates a “stepsPerRev” variable that is useful for changing the amount of micro-stepping or gear ratio easily.

```
int main(void)
{
    int stepsPerRev = _STEPS * _GEAR_RATIO;

    //Initialise GPIO
    gpioInitialise();

    //Set up GPIO pins for stepper motor
    gpioSetMode(ENA1, PI_OUTPUT);
    gpioSetMode(DIR1, PI_OUTPUT);
    gpioSetMode(PUL1, PI_OUTPUT);

    //Set up GPIO pins for stepper motor
    gpioSetMode(ENA2, PI_OUTPUT);
    gpioSetMode(DIR2, PI_OUTPUT);
    gpioSetMode(PUL2, PI_OUTPUT);

    //Set Controller pins to input
    gpioSetMode(D_BTN, PI_INPUT);
    gpioSetMode(C_BTN, PI_INPUT);
    gpioSetMode(B_BTN, PI_INPUT);
    gpioSetMode(A_BTN, PI_INPUT);

    //Start at 0
    gpioWrite(ENA1, PI_LOW);
    gpioWrite(DIR1, PI_LOW);
    gpioWrite(PUL1, PI_LOW);

    //Start at 0
    gpioWrite(ENA2, PI_LOW);
    gpioWrite(DIR2, PI_LOW);
    gpioWrite(PUL2, PI_LOW);
…
```

5.3 UTC/GMT FUNCTION

This function is quite simple, but it is the foundation of all the other time calculations. It returns a TM struct with GMT Month, Day, Year, and Hour, Minute, Seconds. From this, all the current time info can be extracted and constantly updated.

```
/***********************************************************************
* Function:               getGMT
* Purpose:                Prints out Greenwich Mean Time (GMT)
* Precondition:           none
* Postcondition:   Returns TM struct with GMT Month, Day, Year, and Hour, Minute,
Seconds
************************************************************************/
tm sidereal::getGMT()
{
      time_t m_rawTime;
      //Get time
      time(&m_rawTime);

      //Set timeInfo struct equal to return value of gmtime()
      tm *timeInfo = gmtime(&m_rawTime);

      return *timeInfo;
}
```

5.4 JULIAN DATE FUNCTION

This function is necessary in order to calculate sidereal time accurately. It doesn't take in any parameters, and it returns a double containing the Julian date including fractional days.

```
/***********************************************************************
* Function:               getJulianDate
* Purpose:                Calculates current Julian date based on getGMT() and
account for leap years
* Precondition:           None
* Postcondition:   Returns a double containing the Julian date including
fractional days\
* Sources:                https://space.stackexchange.com/questions/13809/time-
utc-julian-date-tle-epoch-how-are-they-related-quantitatively
************************************************************************/
double sidereal::getJulianDate()
{
      double julianDate;
      struct tm timeInfo = getGMT();

//Create a variable for number of days this year including the fractional current
day (current time, noon = .5 of a day), The + 1 on tm_yday is to account for it
starting at 0 instead of 1 for Jan 1st.
      double days = (timeInfo.tm_yday + 1)+ (((timeInfo.tm_hour)) / 24.0) +
(timeInfo.tm_min / 1440.0) + (timeInfo.tm_sec / 86400.0);

      //Correct for Leap year if applicable
      if (__isleap(timeInfo.tm_year) != 0)
      {
            days = -1;
      }

//Julian date formula which accounts for leap years, 2451544.5 = Jan 01, 2000, at
00:00:00, The parts where the year is - 100 is to account for tm_Year starting at
years since 1900, formula needs years since 2000
      julianDate = 2451544.5 + ((365 * (timeInfo.tm_year - 100)) +
((timeInfo.tm_year - 100) / 4) - ((timeInfo.tm_year - 100) / 100) +
((timeInfo.tm_year - 100) / 400) + days);

      return julianDate;
}
```

5.5 EARTH ROTATION ANGLE FUNCTION

This function gets the earth's rotation angle (ERA) and is used to determine the prime meridian for Greenwich Mean Sidereal Time. It returns a double containing the ERA in radians.

```
/*********************************************************************
* Function:               getERA
* Purpose:                Calculate the Earth's Rotation Angle (ERA) as opposed to
sidereal time
* Precondition:           None
* Postcondition:   Returns a double containing the ERA in radians - outputs the
same as getERAcomplex() but is twice as fast (0.000003s)
* Sources:                https://www.atnf.csiro.au/iau-
comm31/pdf/2009_IAUGA_JD6/JD06_capitaine_wallace.pdf
*********************************************************************/
double sidereal::getERA()
{
	double theta = (2 * M_PI * (OFFSET + EARTHS_ROTATIONAL_SPEED *
(getJulianDate() - 2451545.0)));

	theta = fmod(theta, 2 * M_PI);
	if (theta < 0)
	{
		theta += 2 * M_PI;
	}

	return theta;
}
```

5.6 GREENWICH MEAN SIDEREAL TIME FUNCTION

This function uses the two previous functions to determine the sidereal time at the prime meridian. It returns a double containing the GMST in radians of a 24H clock. The formula for this calculation is from the IAU 2000 calculations for GMST. This function proved to be the most accurate to stellarium's calculations, which is what will be used to check the telescope's accuracy.

```
/*********************************************************************
* Function:              getGMSTinRads
* Purpose:               Calculate GMST and return it as a double in Radians
* Precondition:          None
* Postcondition:   Returns a double containing GMST in radians - Most accurate
with databases for stellarium - speed comparable to getERAcomplex(), (0.000006s)
* Sources:               https://astrogreg.com/convert_ra_dec_to_alt_az.html
                               https://www.atnf.csiro.au/iau-
comm31/pdf/2009_IAUGA_JD6/JD06_capitaine_wallace.pdf
*********************************************************************/
double sidereal::getGMSTinRads()
{
      //Get julian date and convert to j2000, then divide by centuries
      double t = ((getJulianDate() - 2451545.0) / 36525.0);

      //Formula for GMST from IAU 2000 expressed in seconds, and converted to
Rads
      double GMST = getERA() + (0.014506 + (4612.156534 * t) + (1.3915817 * t *
t) - (0.00000044 * t * t * t) - (0.000029956 * t * t * t * t) - (0.0000000368 * t
* t * t * t * t)) / 60.0 / 60.0 * (M_PI / 180.0);

      GMST = fmod(GMST, 2 * M_PI);

      //Correct if negative
      if (GMST < 0)
      {
            GMST += 2 * M_PI;
      }

      return GMST;
}
```

5.7 LOCAL MEAN SIDEREAL TIME FUNCTION

This function accepts a time in radians of a 24H clock and adjusts it to match your Local Sidereal Time based off the given longitude given in degrees. Positive longitude means east of the prime meridian and negative longitude means west of the prime meridian. Current accurate usage is to pass in the return value of getGMSTinRads() to the “rads” variable.

```
/***********************************************************************
* Function:              getLMST
* Purpose:               Calculate local sidereal time
* Precondition:          Pass in a double containing GMST or ERA in radians, and
longitude coordinate in degrees (positive number for East, negative for West)
* Postcondition:   Returns a double containing decimal degrees of Local Mean
Sidereal Time
When using getERA(), Approx 1m 13s behind https://www.localsiderealtime.com/
using getERA(), Approx 1m 30s behind stellarium
When using getGMSTinRads(), precisely in time with
https://www.localsiderealtime.com/ using -121.7817 longitude
When using getGMSTinRads(), precisely in time with stellarium "Apparent Sidereal
Time" using -121.7817 longitude
***********************************************************************/
double sidereal::getLMST(double rads, double longitudeEast)
{
	double LMST;
	//Convert rads to degrees and offsets based on longitude
	LMST = ((rads * 180) / M_PI + longitudeEast);
	//Correct if negative
	if (LMST < 0)
	{
		LMST += 360.0;
	}

	return LMST;
}
```

5.8 FUNCTION TO CONVERT RIGHT ASCENSION / DECLINATION TO LOCAL ALTITUDE / AZIMUTH

This function is the backbone of the telescope's ability to aim at a given coordinate. It is based on Greenwich Mean Sidereal Time and receives input for the desired location in Latitude / Longitude. This is using the "atan2" method for calculating Alt/Az. This function is in a separate coordinate class which will be further developed for other functions.

```
twoAxisDeg coordinate::equatorialToLocal(double Ra, double Dec, twoAxisDeg
myPositionDeg)
{
	//Create struct for Alt/Az and Lat/Long
	twoAxisDeg AltAz;
	twoAxisRads latLong;
	twoAxisRads RaDec;

	//Convert and store myPositionDeg from degrees to Radians
	latLong.x = myPositionDeg.x * (M_PI / 180.0);
	latLong.y = myPositionDeg.y * (M_PI / 180.0);

	//Convert and store Ra and Dec from degrees to Radians
	RaDec.x = Ra * (M_PI / 180.0);
	RaDec.y = Dec * (M_PI / 180.0);

	//Get Local Mean Sidereal Time using getGMSTinRads and inputing longitude
through myPosition (note west needs negative number, east needs positive)
	double LMST = sidereal::getLMST(sidereal::getGMSTinRads(), myPositionDeg.y);

	//Convert and store LMST decimal degrees to Radians
	LMST = LMST*(M_PI / 180.0);

	//Get Hour Angle from subtracting Right Ascension (RA) from Meridian (LMST)
	double hourAngle = LMST - RaDec.x;

	//Quadrant correction math
	if (hourAngle < 0)
	{	hourAngle += 2 * M_PI;	}
	if (hourAngle > M_PI)
	{	hourAngle -= 2 * M_PI;	}

	//Set Azimuth
	AltAz.y = (atan2(sin(hourAngle), cos(hourAngle) * sin(latLong.x) -
tan(RaDec.y) * cos(latLong.x)));
	//Set Altitude
	AltAz.x = (asin(sin(latLong.x) * sin(RaDec.y) + cos(latLong.x) * cos(RaDec.y)
* cos(hourAngle)));

	AltAz.y -= M_PI;

	if (AltAz.y < 0)
	{		AltAz.y += 2 * M_PI;		}

	//Convert to degrees decimal for output
	AltAz.x = AltAz.x * (180 / M_PI);
	AltAz.y = AltAz.y * (180 / M_PI);

	return AltAz;
}
```

5.9 DEMO CODE

Below are the basic controls of the demo. It is nothing fancy and can be made more efficient, but it provides a good process that is easy to follow. Note: this whole routine is simply contained in a while (1) loop.

```
if (!gpioRead(A_BTN))
        {
            //Set Direction UP
            gpioWrite(DIR2, PI_HIGH);
            gpioDelay(_DELAY);

            //Pulse
            gpioWrite(PUL2, PI_HIGH);
            gpioDelay(_DELAY);
            gpioWrite(PUL2, PI_LOW);
            gpioDelay(_DELAY);
        }
        else if (!gpioRead(C_BTN))
        {
            //Set Direction DOWN
            gpioWrite(DIR2, PI_LOW);
            gpioDelay(_DELAY);

            //Pulse
            gpioWrite(PUL2, PI_HIGH);
            gpioDelay(_DELAY);
            gpioWrite(PUL2, PI_LOW);
            gpioDelay(_DELAY);

        }

        //If Left or Right
        if (!gpioRead(D_BTN))
        {
            //Set Direction LEFT
            gpioWrite(DIR1, PI_HIGH);
            gpioDelay(_DELAY);

            //Pulse
            gpioWrite(PUL1, PI_HIGH);
            gpioDelay(_DELAY);
            gpioWrite(PUL1, PI_LOW);
            gpioDelay(_DELAY);
        }
        else if (!gpioRead(B_BTN))
        {
            //Set Direction RIGHT
            gpioWrite(DIR1, PI_LOW);
            gpioDelay(_DELAY);

            //Pulse
            gpioWrite(PUL1, PI_HIGH);
            gpioDelay(_DELAY);
            gpioWrite(PUL1, PI_LOW);
            gpioDelay(_DELAY);
        }
```

5.10 STRUCT DATA TYPES IN SIDEREAL.H

These structs are defined in sidereal.h and are used to easily use different ways to express an angle as a measurement of time. Degree minute seconds divide a circle into 360 degrees, and each degree into 60 minutes and each minute into 60 seconds resulting in 1296000 "seconds" per revolution. The second variable is a double which can contain fractional units as well. It is used for measuring Declination.

The hourMinuteSeconds struct is similar, but uses 24 hours instead of 360 degrees. It is based on a 24-hour analog clock and is used for measuring Right Ascension.

```
/*************************************************************************
* Struct:          degreeMinuteSeconds
* Purpose:         Holds data for degree minute seconds format. Ex. 57 Deg,29',32"
* Data members:    degrees      - Holds degrees
*                        minutes      - Holds minutes
*                        seconds      - Holds seconds
*************************************************************************/
typedef struct degreeMinuteSeconds
{
      double degrees;
      double minutes;
      double seconds;
} degreeMinuteSeconds;


/*************************************************************************
* Struct:          hourMinuteSeconds
* Purpose:         Holds data for hour minute seconds format. Ex. 23H 32M 14.23S
* Data members:    hours  - Holds hours
*                        minutes      - Holds minutes
*                        seconds      - Holds seconds
*************************************************************************/
typedef struct hourMinuteSeconds
{
      double hours;
      double minutes;
      double seconds;

} hourMinuteSeconds;
```

5.11 STRUCT DATA TYPES IN COORDINATE.H

The following are structs used to hold two data members for the horizontal and vertical coordinate in a coordinate sphere. They are largely the same but are separately defined so one doesn't get accidentally assigned to another through an "=" operator.

```
/*************************************************************************
* Struct:           twoAxisDeg
* Purpose:          Holds data for 2, decimal degree values like latitude and
longitude, RA / Dec, or Alt / Az
* Data members:     x - Holds latitude, RA, or Alt
*                   y - Holds longitude, Dec, or Az
*************************************************************************/
typedef struct twoAxisDeg
{
      double x;
      double y;
};

/*************************************************************************
* Struct:           twoAxisRads
* Purpose:          Holds data for 2, radian values like latitude and longitude, RA
/ Dec, or Alt / Az
* Data members:     x - Holds latitude, RA, or Alt
*                   y - Holds longitude, Dec, or Az
*************************************************************************/
typedef struct twoAxisRads
{
      double x;
      double y;
};


/*************************************************************************
* Struct:           twoAxisDms
* Purpose:          Holds data for 2, degree minute seconds values like latitude
and longitude, RA / Dec, or Alt / Az
* Data members:     x - Holds latitude, RA, or Alt
*                   y - Holds longitude, Dec, or Az
*************************************************************************/
typedef struct twoAxisDms
{
      degreeMinuteSeconds x;
      degreeMinuteSeconds y;
};
```