# DOCUMENTATION PAGE

| | |
|---|---|
| Student Surname | Blomley |
| First Names | Angus |
| Student Number | 97156721 |
| Course Name and short code<br>(example: Fashion, FAS) | Broadcast Engineering, BEN18303 |
| Academic year | 23-24 |
| Unit Title | Dissertation |
| Unit Code | C18301 |
| Year, Term | Year 3, Semester 2 |
| Six keywords<br>(separated by commas) | Broadcast Engineering, TV, Technology, Esports, Production, Live |
| Your LEAD QUESTION | Track and identify celestial objects. |

By submitting this document:

1. I certify that this assignment is my/our own work and that I am familiar with Ravensbourne's Plagiarism Policy. I also understand that plagiarism is a serious academic offence.
2. I certify that this assignment has not been previously submitted for assessment on this programme.
3. Where material has been used from other sources it has been properly acknowledged.
4. I confirm that I have retained an electronic copy of this assignment and understand that written assignments may be submitted to the JISC Plagiarism Detection Service, I must therefore be able to produce electronic copies of written assignments.
5. I understand that Ravensbourne is at liberty to delete submitted work 12 months after assessment.
6. I also understand that Ravensbourne may wish to use my work (or copy) for future academic purposes in accordance with Ravensbourne's regulations.

Ravensbourne
University London

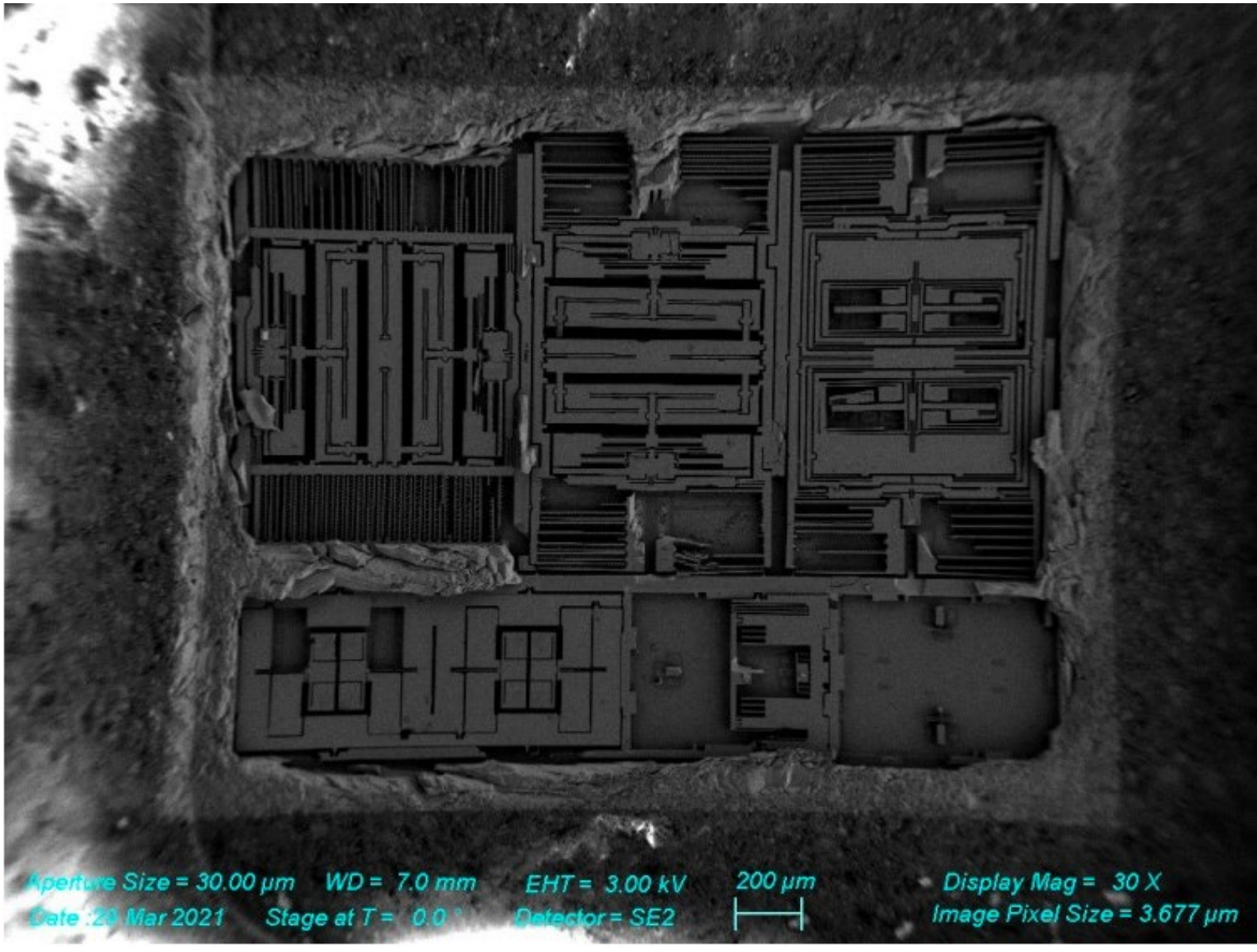
# Celestial Object Tracker

Final Major Project

Angus Blomley

BEN18303

April 2024



## Executive Summary

The celestial object tracker is an electronic device that returns live information of chosen celestial objects on a screen. Similar technology is used in many industries, like astronomy and astro physics, satellite tracking and space exploration, defence and military, automation and robotics, environmental monitoring and more.

For amateur astronomers and educational institutions, the accessibility for highly precise automated tracking systems have been reserved for the well-funded observatories, leaving it unavailable due to the cost and complexity. To solve this, a project has been developed for affordable, scalable, and replicable celestial tracking system.

By combining stepper motors for precise movement, an Arduino, a Raspberry Pi for control processing and a camera for real-time celestial observations, the project's objective is to advance the education in astronomical and photographic studies. The design of this system allows for automated tracking and alignment of a camera to specified coordinates in space, that has been extracted from the skyfield API.

The outcome is a cost effective and reliable celestial tracking system easily available for space scientists, amateur astronomers, and photography enthusiasts. There is a guide in building this project on the GitHub page and is openly available to be developed further to test and enhance precision, user friendliness for community and collaboration improvements.

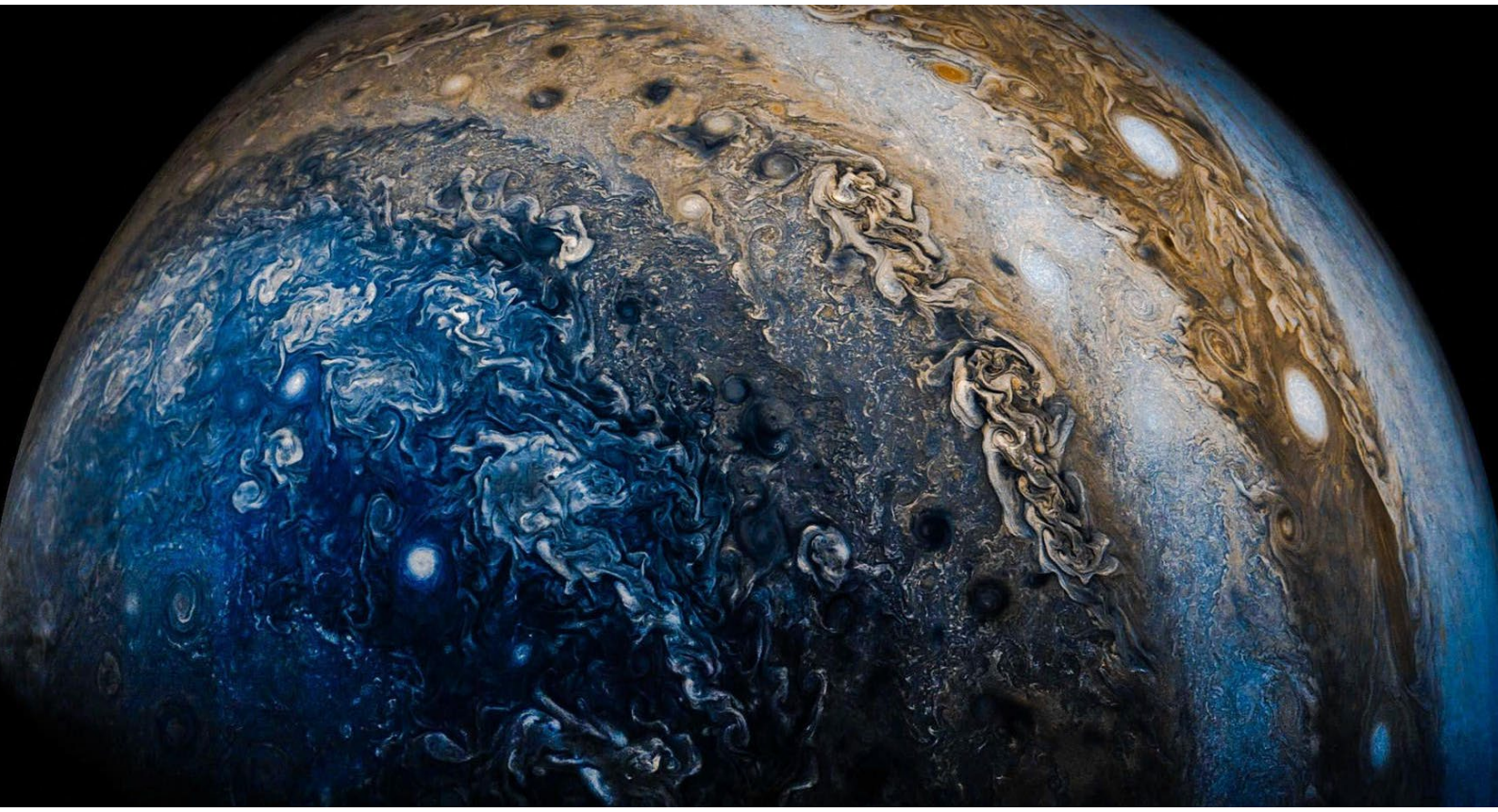


**TABLE OF CONTENTS**

## LIST OF FIGURES

## 1.0 INTRODUCTION

**Celestial tracking systems can vary in cost, from £1000 - £ 1,900,000,000. It is optimal to find a system that is accessible to most people all over the world. To resolve this, Angus Blomley has designed their own software, procedures, and guide to putting together the project.**

## 2.0 Goals / Aims / Objectives

This project has the capability to be improve endlessly, the scope of the project is limited by the skills, technology, timeframe, and monetary requirements. Within these requirements a list of goals aims and objectives have been defined:

- Design and print a 3D structure for all components using Fusion 360.
- Gather Live GPS(Global Positioning System) Data.
- Gather Live IMU(Inertial Measurement Unit Data.
- Gather Live data from Skyfield API.
- Use Arduino and Raspberry Pi to configure the received Data.
- Create GUI(Graphical User Interface) to display information and controls.
- Command motors to point in the direction of selected celestial object.
- Calibrate the device as one whole system for orientation and accuracy.
- Prove that where the device is pointing is accurate.
- Take photos and display information of chosen celestial object.

The final goal for an ideal scenario requires that the Celestial Object Tracker will point the camera precisely in the direction of the chosen celestial object via a GUI and take a photo highlighting the object and display information about it on a monitor.

## 3.0 Review of Research

To understand if this project is possible to conduct, thorough research is necessary. Projects of similar nature have been achieved in the past, which makes this a good starting point to build and improve on using a personal approach, with the provided skills already attained from the past. The skills required for this project include:

- Good Programming knowledge (Python, C++)
- Some knowledge of electronics
- Mechanical Knowledge
- Design Software (Fusion 360)
- Mathematics
- Astronomy

Findings in the following information has allowed me to gather useful information that has led to the outcome this project:

- ‘Satellite Newsgathering’
- ‘Automated #RaspberryPi Planet Tracking GOTO Telescope’ – YouTube
- ‘NRC Publications Archive Archives des publications du CNRC Inertial Measurement Unit (IMU) testing procedure Chaulk, Mitchell’
- ‘CNC V3 Shield Arduino Pinout (Which Arduino Pins Connect Where On A CNC V3 Shield?)’ – YouTube
- ‘How to use the GT U7 GPS module’ - YouTube
- ‘Raspberry Pi Camera Module 3 NoIR’ - PiHut
- ‘2PCS TMC2208 V1.2’ – Amazon
- ‘https://rhodesmill.org/skyfield/’ - Skyfield API

As for the review of research, it was very clear that adequate research is needed. During the development process of creating a working project, hundreds of problems needed to be solved. By taking extra time in doing research, many of these problems could have been mitigated, leading to numerous hours being saved.

### 3.1 Background To Science

Due to the nature of this project, great scientific measures must be taken to be successful. These measures include using extreme precision to combine the calculations of the following live data:

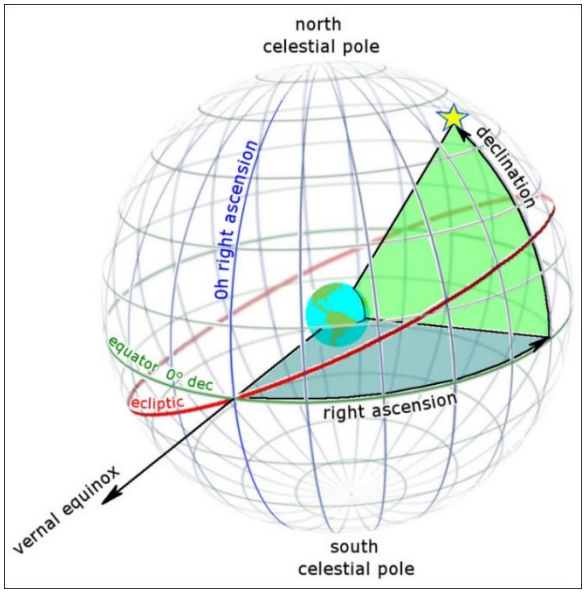
- IMU (Inertial Measurement Unit) for orientation
- GPS (Global Positioning System) for location
- Skyfield API (Application Processing Interface) for Celestial Coordinates
- Engineering the movement (Gear ratios and stepper counts)

To achieve the extreme accuracy required, the correct calculations must be applied. To do this, we use the collected data. ‘The position of a particular object in space can be represented in numerous ways, a widely used method is the equatorial coordinate system. This uses the centre of the Earth and an observation point and a much larger celestial sphere which encompasses the Earth.' (Derek, 2020).

### 3.2 Theory

We need to define two parameters to locate an object along this celestial sphere, the first is RA (Right Ascension) which measure the angular distance of an object eastward along the celestial equator from the vernal equinox. The second parameter is the declination, which measures the angle of an object perpendicular to the equator, north being a positive value and south being negative value. (Derek, 2020). But the observer location will not be from the centre from the Earth, the observations will be made from the Live GPS Coordinates. The object that will be tracked is moving on its

on trajectory, therefore the direction of which the camera points will need to be continuously updated. To do this, data can be gathered from the Skyfield API. The celestial object tracking device can pan and tilt, in astronomy this terminology translates to elevation and azimuth. The reason for obtaining the azimuth and elevation coordinates with time, is to calculate the angle necessary to point in the correct direction.



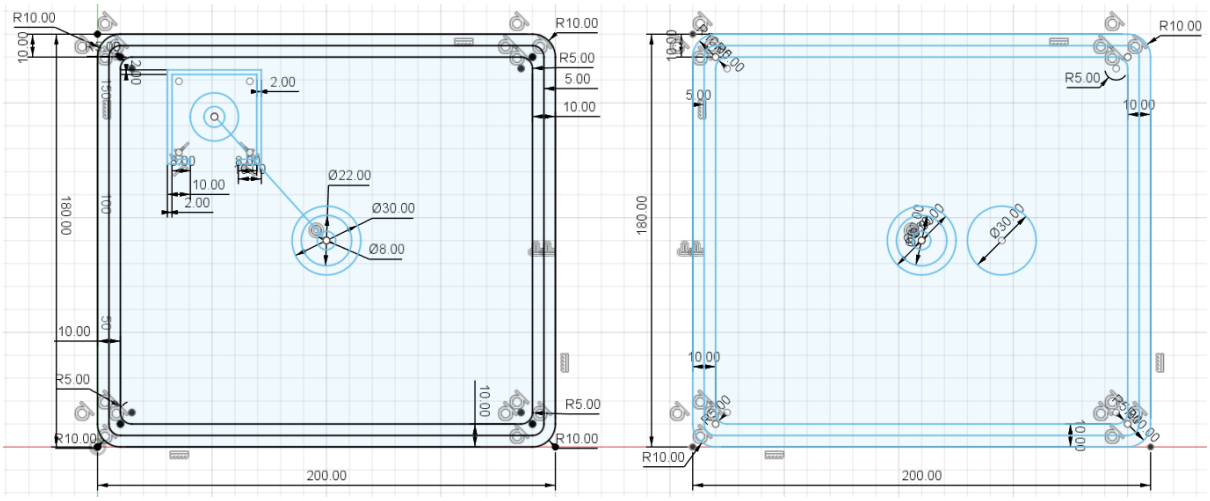

Here's Earth inside the big ball. Declination (green) is measured in degrees north and south of the celestial equator. Right ascension, akin to longitude, is measured east from the equinox. The red circle is the Sun's apparent path around the sky, which defines the ecliptic. [KING, 2019]. – ***FIG.1***

## 4.0 Methodology

This section follows a comprehensive approach of the design process, integrating hardware, software, data processing and the graphical user interface for the Celestial Object Tracker. The goal was to achieve a system able to track celestial objects by combining GPS and MPU data with stepper motor function to align the Picam to specified celestial coordinates.

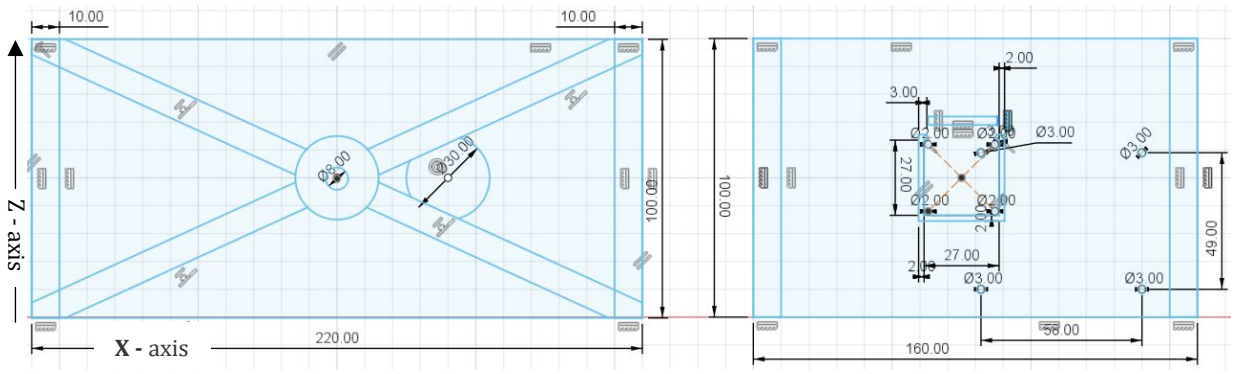
## 4.1 Structural Design

The structure of the pan and tilt was designed using fusion 360, building upon iterations of previously designed models.



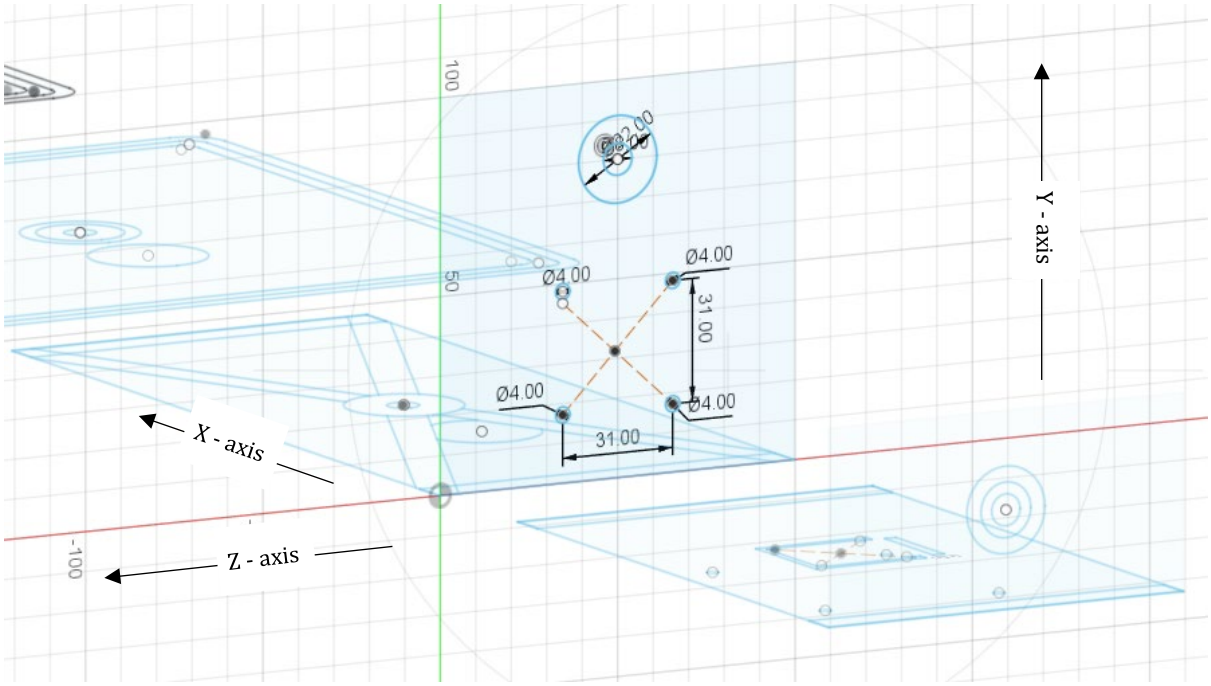

*Fusion 360 Sketch of the base and the lid, scale in mm. This is where the bulk of electrical equipment is stored.* ***FIG2***

The base of the pan and tilt structure is responsible for securing the X(pan) direction motor. Both the X and Y(tilt) motors are connected to a 20 teeth aluminium wheel, attached to a 200mm timing pulley and then turning a 40 teeth wheel. A 2:1 gearing ratio is applied, for every revolution on the motor makes it half on the 40 teeth gear. Having more teeth on the larger wheel give a much higher access to accuracy, which is crucial for this type of project.

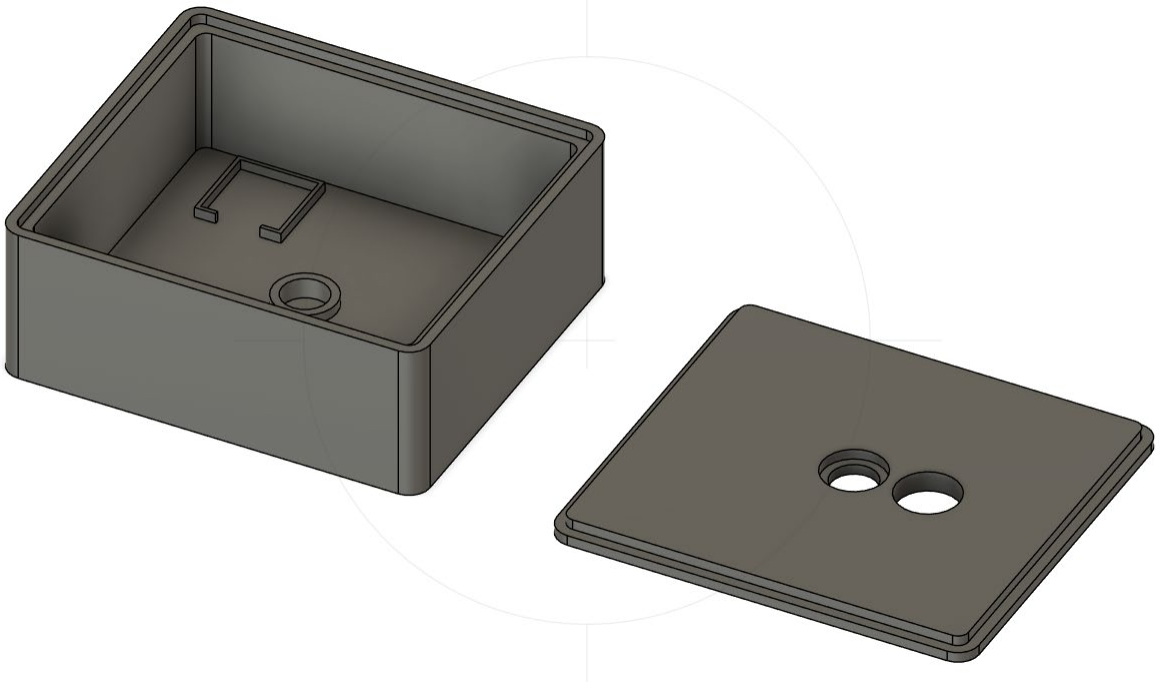



*Fusion 360 Sketch of the pan and tilt in axis X and Z scale in mm.* ***FIG3***

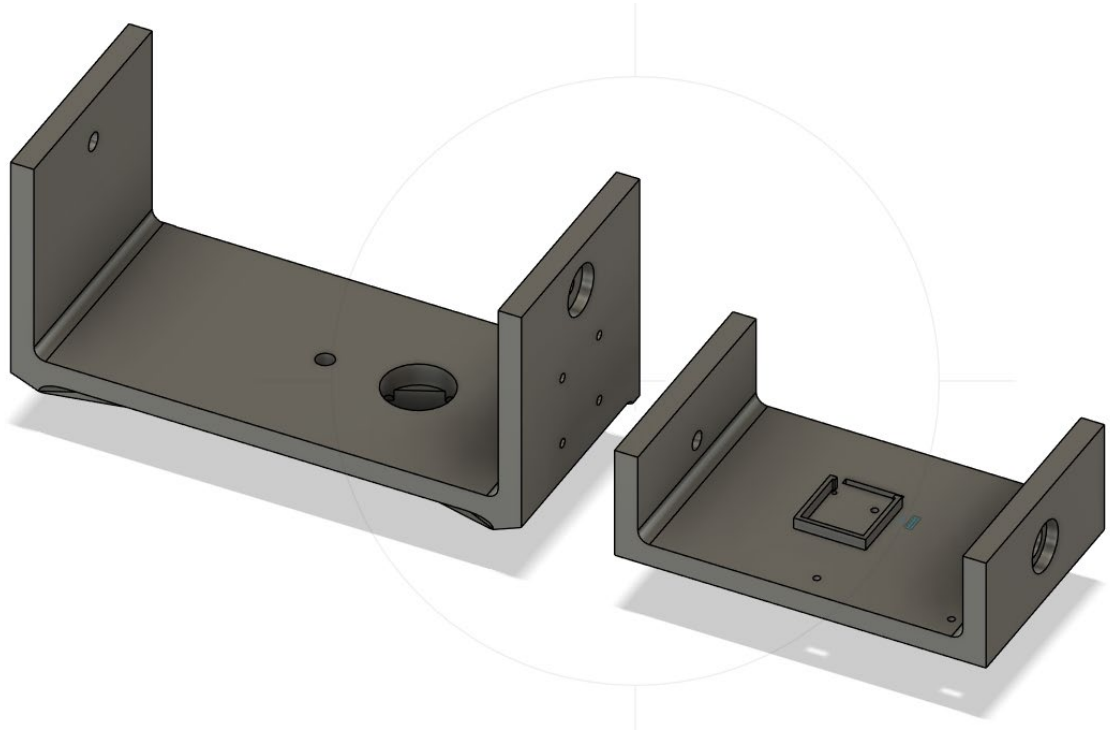
The top of the pan and tilt structure is for housing the tilt mechanism, along with the operation of the Picam Module 3, IMU and Raspberry Pi 4.




*Fusion 360 Sketch of the pan right side Y and Z axis, for the areas labelled in measurements are for the mounting of the Nema 17 tilt stepper motor labelled as motor(Y). Scale in mm.* ***FIG4***



*Here is the extrusion of the base and lid, resembling the model of the 3D print.* ***FIG5***



*Extruded sketch of the pan and tilt, resembling the model of the 3D print.* ***FIG6***

*Completed 3D print and assembly of the celestial object tracking system.* ***FIG7***

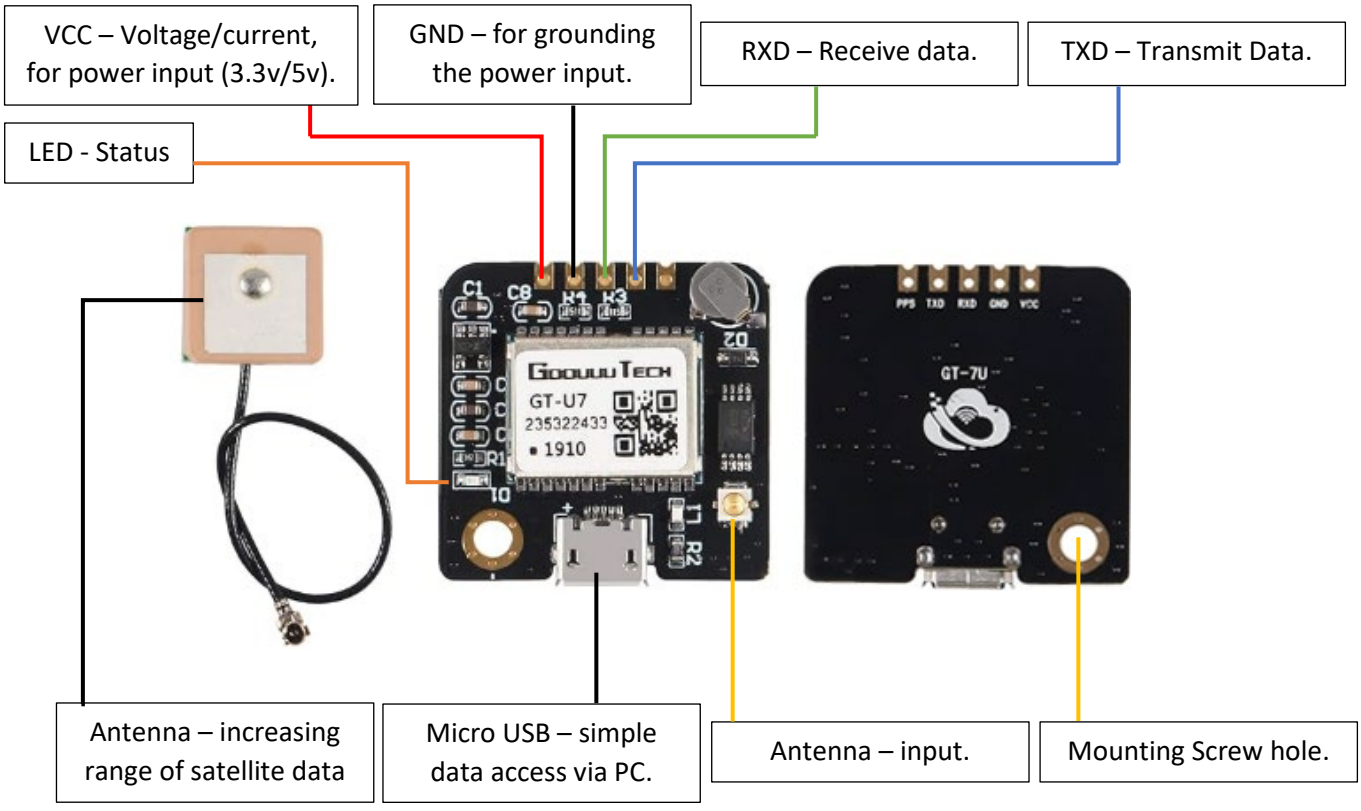
## 4.2 Module setup and installation

This project involves 3 sets of live data to be received. GPS, IMU and celestial coordinates. Identification of observer's position must be relative to celestial coordinates; this can be achieved using live data capture via GPS and IMU.

### 4.2.1 GPS – Global Positioning System

This project uses the GT-U7 GPS module, this device has two areas of power and data transmission shown in figure 10.

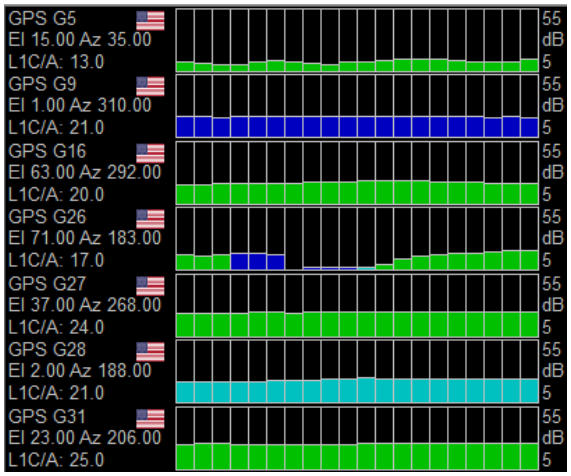
GT-U7 inputs/outputs:




*GPS Module – GT-U7.* ***FIG8***

This device has been tested indoors and outdoors for reliability. Ublox developed a program named 'U-center', this can capture all the information that the GT-U7 can obtain.

The first test was done indoors beside a window. Results after 10 minutes from startup:

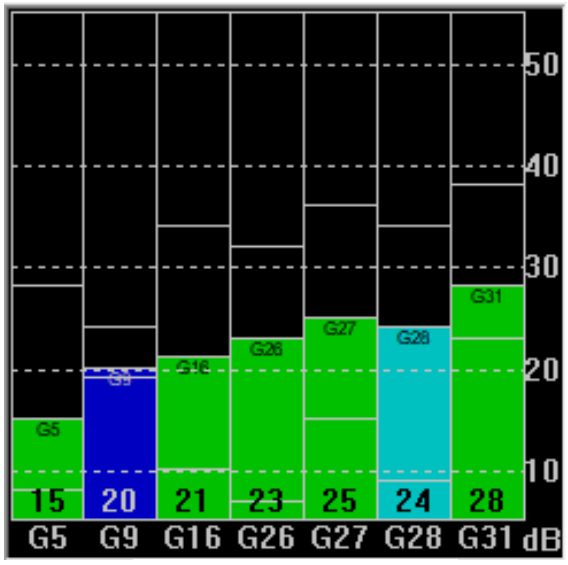



*Satellite level history.* ***FIG9***

This graph shows the history of satellite connection, this helps identify the stability of the GPS connection to satellites over time. Colour codes:

Green - Valid 3D navigation fix
Blue - Degraded navigation fix
Cyan - Valid 2D navigation fix
Red - No or invalid navigation fix

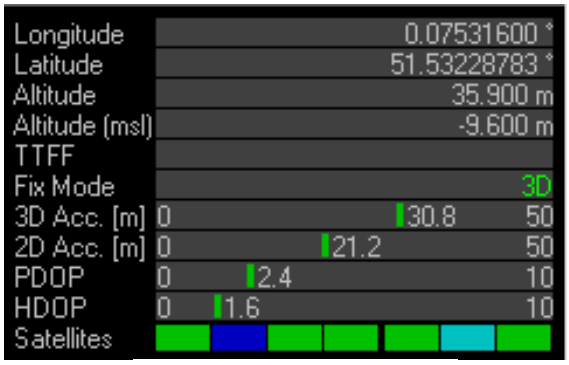
[U-Blox, 2022]




*Live satellite level.* ***FIG10***

The live satellite level graph displays the name of the satellite on the x axis(e.g. G26) and the signal strength in dBhz (decibels relative to hertz).

SNR(signal-to-noise ratio) is usually expressed in terms of decibels. It refers to the ratio of the signal power and noise power in a *given bandwidth*. [GNSS, 2010]

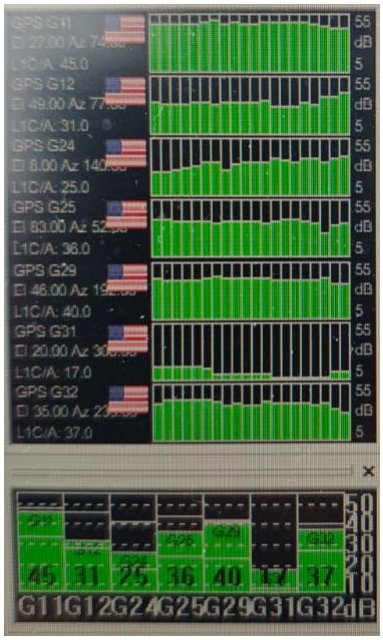



*Live GPS data.* ***FIG11***

Live GPS data frame shows the vital information needed for this project to work. The GPS must be in a permanent state of '3D' in 'Fix Mode'. When more than four satellites are connected and a signal strength above 30dBhz, a fix is available and accurate location data can be calculated.

The second test consisted of taking the GPS outdoors and connecting it to a laptop:

*GT-U7 active outdoor and sending signal to laptop.* ***FIG12***




*Decoded GPS information on U-Centre.* ***FIG13***

### 4.2.2 MPU – Motion Processing Unit

The MPU6050 module used in the project is a commonly used unit. The MPU6050 is a Micro-Electro-Mechanical Systems (MEMS) that consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion-related parameters of a system or object. [Joseph, 2022]

The information that this module provides will be able to make the device identify what orientation it is, thus the direction.

**VCC** – Powers Module via 5v/3.3v

**GND** – Ground pin

**SCL** – Serial Clock, clock pulse for I2C communication

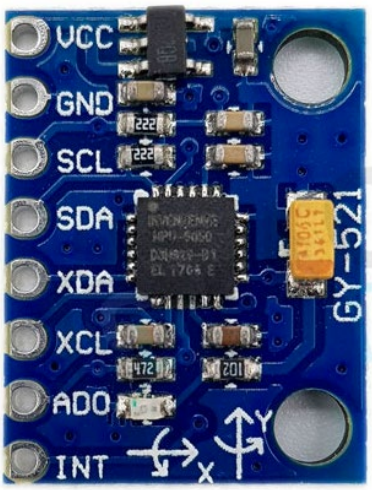
**SDA** – Serial Data through I2C communication

**XDA** – Auxiliary Serial Data

**XCL** – Auxiliary Serial Clock

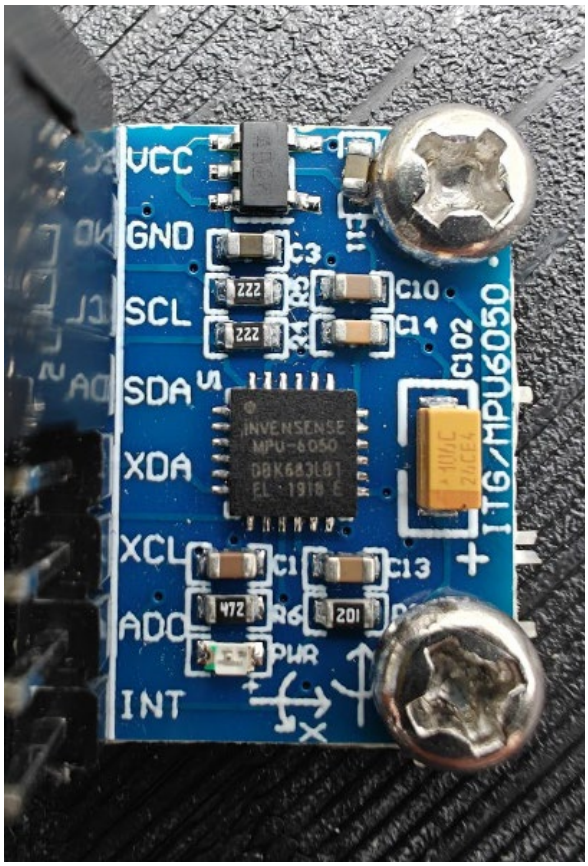
**ADD/ADO** - Address select pin if multiple MPU6050 modules are used.

**INT** - Interrupt pin to indicate that data is available for MCU to read. [Joseph, 2022]

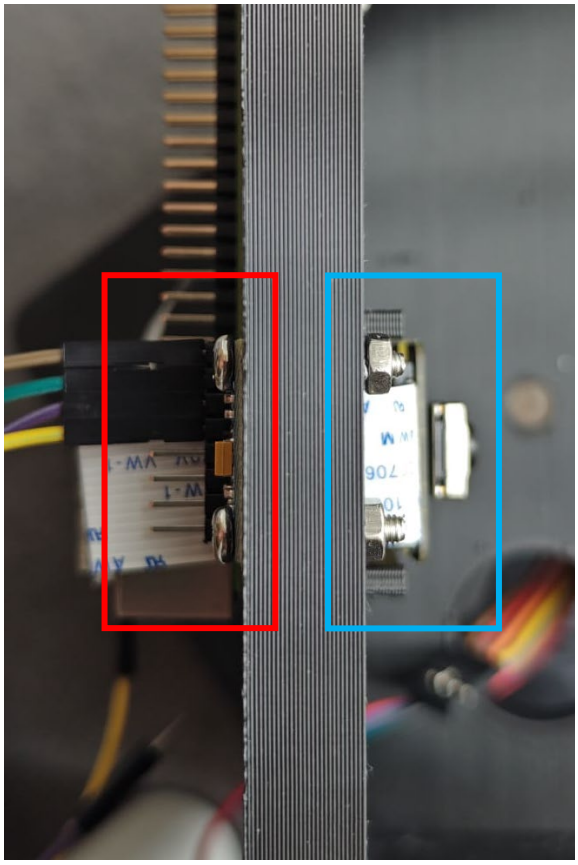



*MPU6050 Module.* ***FIG14***

The IMU sensor must be mounted to the camera, this allows the object required to be tracked is relative to the camera. The sensor detect pitch, roll, yaw and needs to be calibrated every time the device is turned on or moved to allow for accurate readings and measurement.



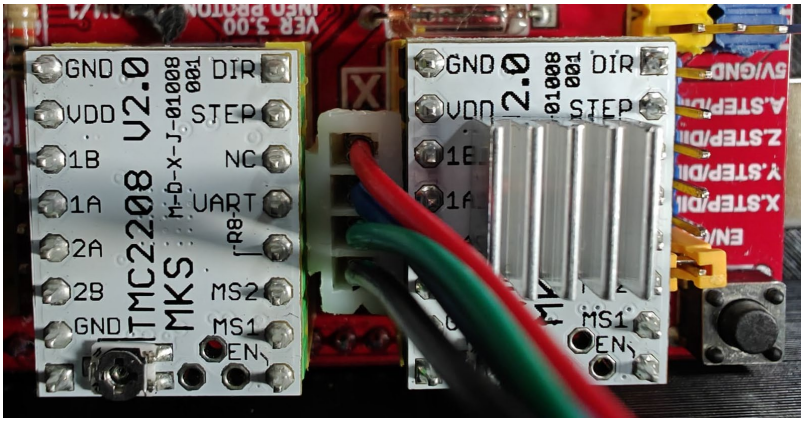

*A close-up photo of the IMU, model is the MPU-6050. -* ***FIG15***



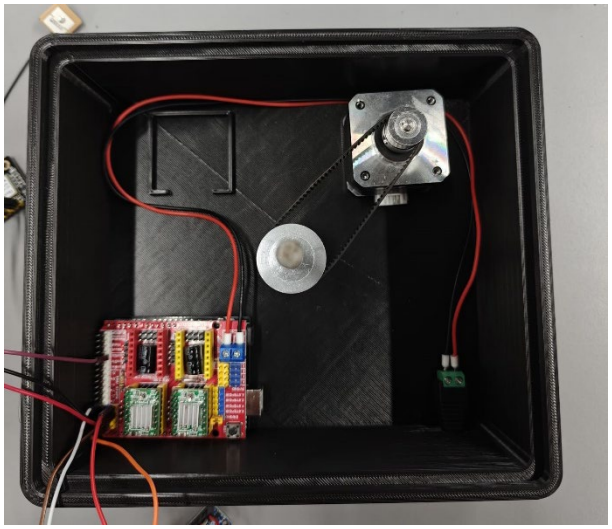
*Camera highlighted in a blue box and IMU in red. –* ***FIG16***

### 4.2.3 Motors, Gearing and Drivers

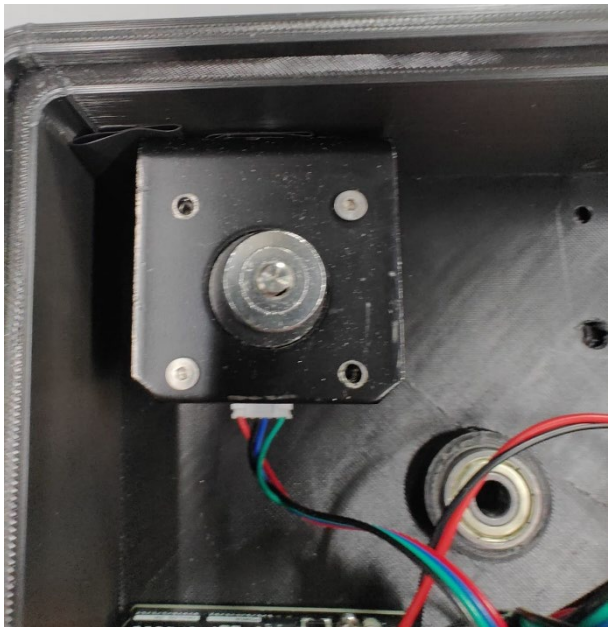
Nema17 motors are used in this project due to low cost and simplicity. These motors combined with two TMC2208 stepper motor drivers allow the motor to make micro steps of up to 16 times, significantly increasing the accuracy.



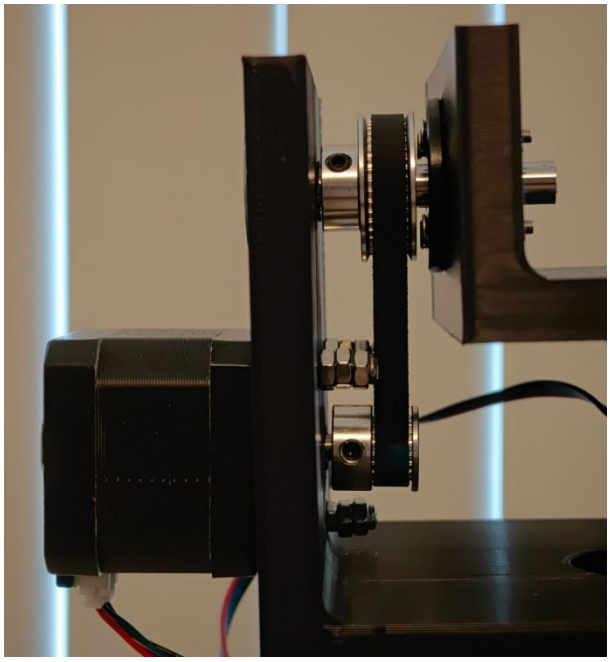

*A close-up photo of the TMC2208 stepper motor drivers installed into the Arduino CNC shield.* ***FIG17***



Base motor(X) Layout. **FIG18**



Base motor(X) Mount. **FIG19**



Tilt motor(Y) Mounted. **FIG20**

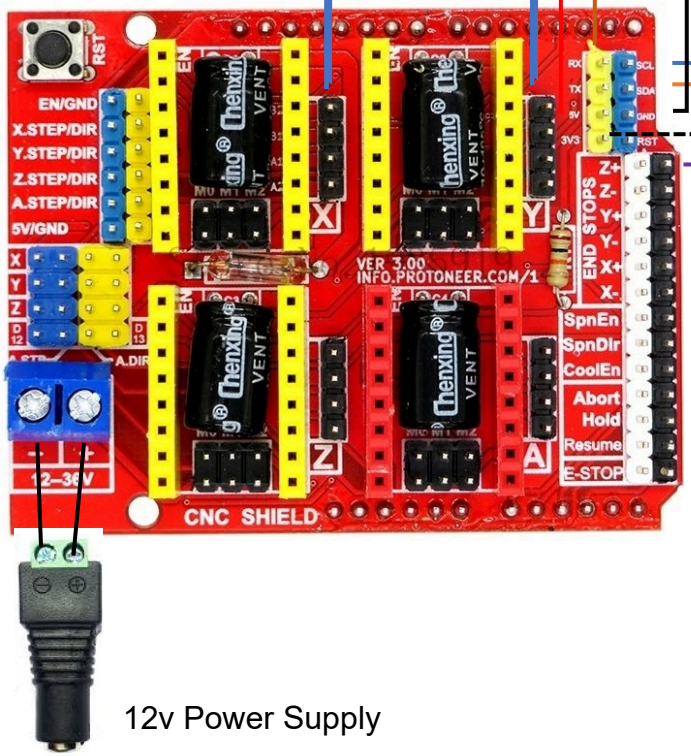
Nema17

Nema17

Combining the modules for serial and I2C communication.

GT-U7

CNC Shield




MPU6050


I2C

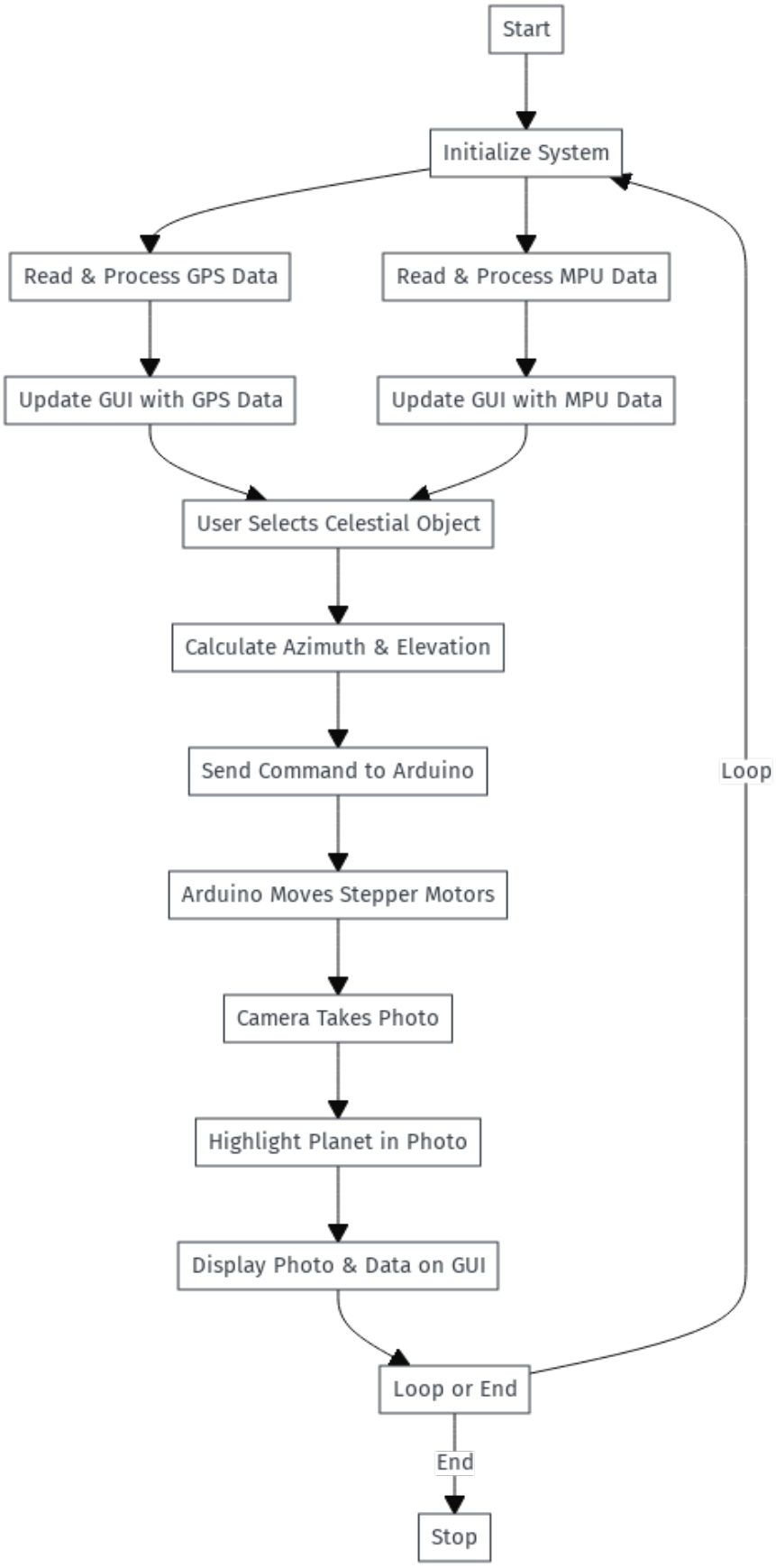
Legend

- TX – RX, Arduino receives data from GPS.
- 5V power from Arduino to GPS.
- GND(ground) for GPS.
- SCL, Serial Clock, clock pulse for I2C communication.
- Serial Data through I2C communication
- 3.3v power from Arduino for MPU6050
- GND(ground) for MPU
- Nema17 power, ground, stepDir, stepCount

12v Power Supply

### 4.3 Software Environment

Below is high-level data flow diagram of the celestial star tracker project:



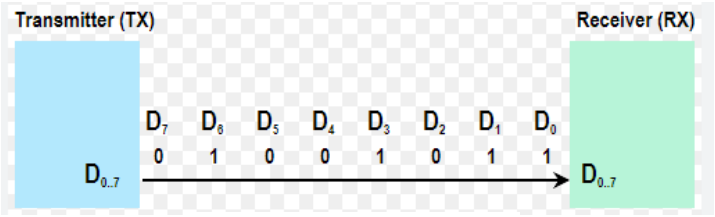
### 4.3.1 Libraries

The core of how the technology works in this project lies in the programming. Utilizing the power of already existing libraries makes creating this project far more achievable. For Raspberry Pi, Python is the chosen programming language due to its vastly existing astronomy community and widely used applications. Arduino must use its own programming language which derives from C++.

To acquire data, the libraries must be installed and imported as follows:

Python:

- Serial – Allows for serial connections, this refers to “sending data one bit at a time, sequentially, over a communication channel or computer bus” (Mackenzie, 1980).




*Serial Interface example.* ***FIG21***

- Time – The `time` module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code. (Ronquillo, n.d.)
- Tkinter – The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems. (Foundation, 2024)
- Skyfield.api – Skyfield computes positions for the stars, planets, and satellites in orbit around the Earth. Its results should agree with the positions generated by the United States Naval Observatory and their *Astronomical Almanac* to within 0.0005 arcseconds (half a “mas” or milliarcsecond). (Rhodes, 2024)

Arduino:

- Wire – allows you to communicate with I2C devices, a feature that is present on all Arduino boards. I2C is a very common protocol, primarily used for reading/sending data to/from external I2C components. (Arduino, Wire, 2023)
- I2Cdev – collection of uniform and well-documented classes to provide simple and intuitive interfaces to an ever-growing collection of I2C devices. Each device is built to make use of the generic i2cdev code, which abstracts the I2C bit-level and byte-level communication away from each specific device class, making it easy to keep the device code clean while providing a simple way to modify just one class to port the I2C communication code onto different platforms (Arduino, PIC, simple bit-banging, etc.). (Rowberg, 2024)

- MPU6050 – MPU6050 Combines a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor(DMP) which processes complex 6-axis MotionFusion algorithms. (Cats, 2024)
- AccelStepper – This library allows you to control unipolar or bipolar stepper motors.” (Stepper, 2018)
- TinyGPS++ - A compact Arduino NMEA (GPS) parsing library (Lee, 2019)
- SoftwareSerial – The SoftwareSerial library allows serial communication on other digital pins of an Arduino board, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signalling for devices which require that protocol. (Arduino, SoftwareSerial Library, 2022)

These libraries get imported and included into each of the development environments as shown in figures 11 and 12:

```
1   import tkinter as tk
2   from tkinter import font as tkFont
3   from skyfield.api import Loader, Topos, load
4   import serial
5   import time
```

*Python libraries import.* ***FIG22***

```
1   #include "Wire.h"
2   #include "I2Cdev.h"
3   #include "MPU6050.h"
4   #include <AccelStepper.h>
5   #include <TinyGPS++.h>
6   #include <SoftwareSerial.h>
```

*Arduino  libraries include.* ***FIG23***

### 4.3.2 Initialization and variables

Initializing the system is asking all the modules to bootup so they are ready to send and receive data:

```
8   //initialize
9   MPU6050 mpu;
10  AccelStepper stepperX(AccelStepper::DRIVER, 2, 3), stepperY(AccelStepper::DRIVER, 4, 5);
11  SoftwareSerial serial_connection(17, -1);
12  TinyGPSPlus gps;
```

*Start Arduino initialization.* ***FIG24***

In figure 24 line 9, we call ‘MPU6050 mpu’ – this defines MPU6050 from the library into a shorter name ‘mpu for readability.

Line 10:

‘AccelStepper stepperX(AccelStepper::DRIVER, 2, 3), stepperY(AccelStepper::DRIVER, 4, 5);’

is defining the names for the motors followed by the pin numbers attached to the CNC shield.

Line 11:

‘SoftwareSerial serial_connection(17, -1);’

SoftwareSerial is defined as the serial connection between the GPS and the Arduino, which is responsible for the arduino to receive GPS data.

Line 12:

‘TinyGPSPlus gps;’

Boots the ‘TinyGPSPlus’ library as ‘gps’ for readability.

Figure 25 displays initialization code and variables for the python script:

```
7   #Declare Variables and initialize
8   ser = serial.Serial('COM4', 9600, timeout=0.1)
9   time.sleep(2)
10  loader = Loader('~/.skyfield-data')
11  planets = loader('de421.bsp')
12  ts = loader.timescale()
13  last_print_time = time.time()
14  last_print_time = time.time()
15  root = tk.Tk()
16  root.title("Celestial Object Tracker")
17  observer_location = Topos(latitude_degrees=51.5074, longitude_degrees=-0.1278)
18  current_lat = None
19  current_lng = None
20  accel_x, accel_y, accel_z = None, None, None
21  gyro_x, gyro_y, gyro_z = None, None, None
22  current_lat, current_lng, current_altitude, current_satellites = None, None, None, None
```

*Python script initialization and variables* ***FIG25***

Line 8: Serial library is called and defines the port(‘COM4’), baud rate(‘9600’) and timeout(0.1 seconds) for the USB connection between the Arduino and the Raspberry Pi 4. This information is stored the variable named ser for readability and time saving purposes.

The ‘loader’ and ‘planets’ variables store the value of skyfield.api library being called. This allows us to fetch useful information like celestial coordinates for calculations later.

‘root = tk.Tk()’ is defined to call from the tkinter library which is responsible for the GUI(graphical user interface).

All the other information is set to either ‘0’ or ‘none’, this makes it ready to receive data in preparation for being filled with live data.

```
celestial_objects = {
    "Mercury": planets['mercury'],
    "Venus": planets['venus'],
    "Mars": planets['mars'],
    "Jupiter": planets['jupiter barycenter'],
    "Saturn": planets['saturn barycenter'],
    "Uranus": planets['uranus barycenter'],
    "Neptune": planets['neptune barycenter'],
    "Moon": planets['moon'],
    "Sun": planets['sun']
}
```

*Define celestial objects into an object array* ***FIG26***

Arduino variables shown in Figure 27.

```
unsigned long lastMPUUpdate = 0;
unsigned long updateIntervalMPU = 500; // Interval for MPU updates
unsigned long lastGPSUpdateTime = 0;
const unsigned long gpsUpdateInterval = 500; // Update GPS data every 1000 milliseconds (1 second)

const float STEPS_PER_REV = 1600; // Steps per revolution of your stepper motor
const float GEAR_RATIO = 2; // Gear ratio of any gearing attached to the motors
const float STEPS_PER_DEGREE = (STEPS_PER_REV * GEAR_RATIO) / 360.0; //

int currentAzimuthPosition = 0; // Current azimuth step position
int currentElevationPosition = 0;

String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;
```

*Define Arduino variables.* ***FIG27***

### 4.3.3 Data acquisition and functions for Arduino

Figure 28 is the start of the ‘void loop’ function and the ‘while’ function for extracting the GPS data using the tinygpsplus library. If the GPS is collecting data this function will return the Latitude, Longitude, Altitude, and number of satellites connected with appropriate measurements and decimal places given. This function will run immediately and every time the void loop function is called.

```
46  void loop() {
47    // GPS data processing
48    while (serial_connection.available() > 0) {
49      if (gps.encode(serial_connection.read())) {
50        unsigned long currentMillis = millis();
51        if (currentMillis - lastGPSUpdateTime >= gpsUpdateInterval) {
52          lastGPSUpdateTime = currentMillis; // Update the last update time
53
54          // Now, check if the GPS data is valid and if so, send it
55          if (gps.location.isValid()) {
56            Serial.print("Lat: ");
57            Serial.println(gps.location.lat(), 6); // Print latitude with 6 decimal places
58            Serial.print("Lng: ");
59            Serial.println(gps.location.lng(), 6); // Print longitude with 6 decimal places
60          }
61
62          if (gps.altitude.isValid()) {
63            Serial.print("Altitude: ");
64            Serial.print(gps.altitude.meters());
65            Serial.println(" Meters"); // Ensure to add line break for altitude
66          }
67
68          if (gps.satellites.isValid()) {
69            Serial.print("Satellites: ");
70            Serial.println(gps.satellites.value());
71          }
72        }
73      }
74    }
```

*Start of Void Loop and GPS Extraction function using tinygpsplus library.* ***FIG28***

Figure 29 extracts live data from the functioning of the MPU:

```
76    // MPU data processing
77    unsigned long currentMillis = millis();
78    if (currentMillis - lastMPUUpdate >= updateIntervalMPU) {
79      lastMPUUpdate = currentMillis;
80
81      int16_t ax, ay, az;
82      int16_t gx, gy, gz;
83      mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
84
85      // Serialize the MPU data
86      Serial.print("MPU: ");
87      Serial.print(ax);
88      Serial.print(",");
89      Serial.print(ay);
90      Serial.print(",");
91      Serial.print(az);
92      Serial.print(",");
93      Serial.print(gx);
94      Serial.print(",");
95      Serial.print(gy);
96      Serial.print(",");
97      Serial.println(gz);
98    }
```

*MPU live data extraction function using MPU6050 library.* ***FIG29***

```
void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
      inputString.trim(); // Remove any trailing whitespace

      if (inputString.startsWith("RTH")) {
        returnToHome();
      } else if (inputString.startsWith("MOT")) {
        int firstCommaIndex = inputString.indexOf(',');
        int secondCommaIndex = inputString.indexOf(',', firstCommaIndex + 1);
        if (firstCommaIndex != -1 && secondCommaIndex != -1) {
          // Extracting the azimuth and elevation from the command
          String azStr = inputString.substring(firstCommaIndex + 1, secondCommaIndex);
          String elStr = inputString.substring(secondCommaIndex + 1);
          float azimuth = azStr.toFloat();
          float elevation = elStr.toFloat();
          track_celestial_object(azimuth, elevation);
        }
      }
      // Reset for the next command
      inputString = "";
      stringComplete = false;
    }
  }
}
```

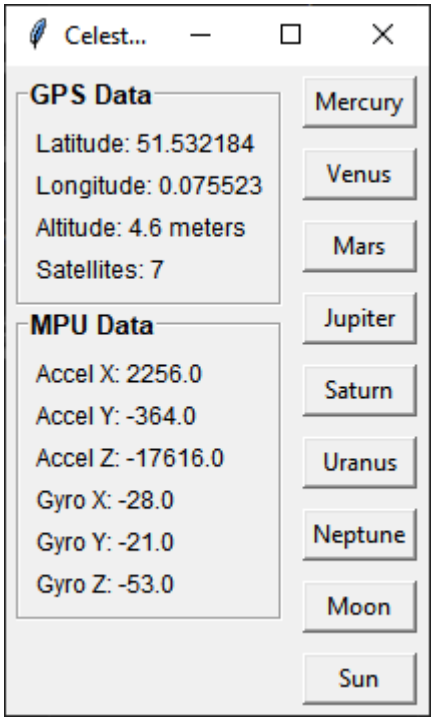
*Void serialEvent function parses information and receives commands from python script.* ***FIG30***

```
void track_celestial_object(float azimuth, float elevation) {
    // Calculate target steps for each motor
    int targetAzimuthSteps = azimuth * STEPS_PER_DEGREE;
    int targetElevationSteps = elevation * STEPS_PER_DEGREE;

    // Calculate the difference between the current and target positions
    int moveStepsAzimuth = targetAzimuthSteps - currentAzimuthPosition;
    int moveStepsElevation = targetElevationSteps - currentElevationPosition;

    // Move the motors to the target positions
    stepperX.move(moveStepsAzimuth);
    stepperY.move(moveStepsElevation);

    // to ensure motors reach their target positions
    while (stepperX.distanceToGo() != 0 || stepperY.distanceToGo() != 0) {
        stepperX.run();
        stepperY.run();
    }
```

*The track celestial object function receives the azimuth and elevation coordinates and calculates the correct positions for the motors.* ***FIG31***

### 4.3.4 The Python Script

The script is developed to create a graphical user interface(GUI) application using ‘tkinter’ for tracking celestial objects and reading live data. It can communicate to Arduino via serial communication and parse any data transferred between devices as Python is high-level and Arduino is low-level. Live data about the location and orientation of the device is sent to the script, it uses this information to send commands for aligning with chosen celestial bodies that have been put in the format of buttons on the GUI.




*Graphical User Interface.* ***FIG32***

Figure 32 shows a screen shot of the project’s graphical user interface with the live GPS and MPU data being updated every 500ms.

We use the MPU data to make sure that the information being displayed on the Accel and Gyro is within 5 numbers of the zero’ d position. The zero’ d position is the numbers that will be output for all the axes after the MPU has been calibrated.

The MPU must be calibrated every time the device has moved.

The safe read line function shown in figure 33 safely reads a line from the serial connection and handles exceptions.

```
36  def safe_read_line():
37      try:
38          return ser.readline().decode('utf-8').strip()
39      except Exception as e:
40          print(f"Error reading line: {e}")
41          return None
```

*Safe Read Line.* ***FIG33***

The most complex part of this project was developing the 'update_data()' function, the beginning of this function starts in figure 34.

```
48  def update_data():
49      global current_lat, current_lng, current_altitude, current_satellites
50      global accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z
51      global print_mpu_data
52      global last_print_time
53
54      line = safe_read_line()
55      if line:
56          try:
57              if line.startswith("MPU:"):
58                  # Remove "MPU: " prefix and then split the line
59                  parts = line.replace("MPU: ", "").split(',')
60                  if len(parts) == 6:  # Ensure there are exactly 6 parts for the MPU data
61                      accel_x, accel_y, accel_z = float(parts[0]), float(parts[1]), float(parts[2])
62                      gyro_x, gyro_y, gyro_z = float(parts[3]), float(parts[4]), float(parts[5])
63
64                      # Update the GUI elements with new data
65                      mpu_vars['accel_x_var'].set(f"Accel X: {accel_x}")
66                      mpu_vars['accel_y_var'].set(f"Accel Y: {accel_y}")
67                      mpu_vars['accel_z_var'].set(f"Accel Z: {accel_z}")
68                      mpu_vars['gyro_x_var'].set(f"Gyro X: {gyro_x}")
69                      mpu_vars['gyro_y_var'].set(f"Gyro Y: {gyro_y}")
70                      mpu_vars['gyro_z_var'].set(f"Gyro Z: {gyro_z}")
71
```

*Insufficient data else print statement for MPU.* ***FIG34***

The relevant global variables are added to this function, this function will be responsible for receiving the Arduino information by using 'line = safe_read_line()' to read the serial port and using the if statement to see if data is available continue to process the data.

An if statement is created to read the MPU data which is called by

'if line.startsWith("MPU: ", "").split(',')'

The python script is using Serial library functions 'line.startsWith ' and '.split' to read through the serial connection and select the code that was written in the Arduino sketch. The collected code contains the live data from the MPU, it is then inserted into the global variables 'global accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z ' by selecting the 6 parts of information and assigning it via an index method.

Now that the data is parsed, ready and available it can be used in the GUI. 'mpu_vars' is the GUI string(text) that is being displayed to the user.

```
            mpu_vars['gyro_y_var'].set(f"Gyro Y: {gyro_y}")
            mpu_vars['gyro_z_var'].set(f"Gyro Z: {gyro_z}")

        else:
            print("Insufficient MPU data elements.")
```

*Insufficient data else print statement for MPU.* ***FIG35***

If the data cannot be read from the line or correctly parsed a print statement will be sent to the terminal displaying the message "Insufficient MPU data elements".

The process for the GPS remains similar in figure 36:

```
            elif "Lat:" in line:
                current_lat = float(line.split("Lat: ")[1])
                gps_vars['lat_var'].set(f"Latitude: {current_lat}")
            elif "Lng:" in line:
                current_lng = float(line.split("Lng: ")[1])
                gps_vars['lng_var'].set(f"Longitude: {current_lng}")
            elif "Altitude:" in line:
                alt_data = line.split("Altitude: ")[1]
                # Remove the " Meters" text to isolate the numeric value
                alt_data = alt_data.replace(" Meters", "")
                try:
                    current_altitude = float(alt_data)
                    gps_vars['alt_var'].set(f"Altitude: {current_altitude} Meters")
                except ValueError:
                    # Handle the case where alt_data cannot be converted to float
                    print(f"Could not convert altitude to float: '{alt_data}'")
                    gps_vars['alt_var'].set("Altitude data is not valid.")
            elif "Satellites:" in line:
                sat_data = line.split("Satellites: ")[1]
                if sat_data != " data is not valid.":
                    current_satellites = int(sat_data)
                    gps_vars['sat_var'].set(f"Satellites: {current_satellites}")
                else:
                    gps_vars['sat_var'].set(sat_data)
            else:
                print(f"Unrecognized line format: {line}")
        except ValueError as e:
            print(f"Value parsing error: {e}")
        except Exception as e:
            print(f"Unexpected error: {e}")

    # Check if all required data (GPS and MPU) have been updated
    if current_lat is not None and current_lng is not None and current_altitude is not None and current_satellites is not None:
        pass
    if all(v is not None for v in [accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z]):
        pass
    else:
        print("Waiting for complete data...")

    root.after(100, update_data)
```

*GPS Data receive from Arduino and parsed for python script use.* ***FIG36***

A final check is done to ensure the data has been updated to check the global variables to see if their value is none.

‘`Root.after(100, update_data)`’ – is a mark to end the function, update the root(GUI) with the function update_data and do this every 100ms.

Figure 37: Button command function, when a button is clicked, and the data is available the function will send the command through the code and received by the motors to move the pan and tilt head in the direction of the chosen celestial body.

```
def button_command(body, name):  # Add the 'name' parameter
    if current_lat is not None and current_lng is not None:
        track_celestial_object(body, current_lat, current_lng)
        print(f"Moving towards {name}...")  # Use the 'name' parameter here
    else:
        print("GPS data not available.")

def return_to_home():
    # Sends "RTH\n" to signal the Arduino to return motors to the home position
    send_str = "RTH\n"
    ser.write(send_str.encode('utf-8'))
```

*Button command and return to home functions.* ***FIG37***

Return to home function moves the device back to its zero’ d calibrated position.

Figure 38: Track celestial object function is responsible for gathering the coordinates of the celestial object using the skyfield.api library, also setting the observers location variable ready to receive live data.

```python
def track_celestial_object(body, lat, lng):
    t = ts.now()
    earth = planets['earth']
    observer_location = Topos(latitude_degrees=lat, longitude_degrees=lng)
    observer = earth + observer_location
    astrometric = observer.at(t).observe(body)
    alt, az, dist = astrometric.apparent().altaz()
    alt_deg = int(alt.degrees)
    az_deg = int(az.degrees)
    # Ensure the command is formatted as "MOT,azimuth,elevation\n"
    send_str = f"MOT,{az_deg},{alt_deg}\n"
    ser.write(send_str.encode('utf-8'))
```

*Track celestial object function.* ***FIG38***

The main function is where the final stop for all the data in the project is processed and output. All the functions created throughout the project will be called in the main function.

```python
def main():
    global gps_vars, mpu_vars
    gps_vars = {var: tk.StringVar() for var in ['lat_var', 'lng_var', 'alt_var', 'sat_var']}
    mpu_vars = {var: tk.StringVar() for var in ['accel_x_var', 'accel_y_var', 'accel_z_var', 'gyro_x_var', 'gyro_y_var', 'gyro_z_var']}
    data_frame = tk.Frame(root, width=200)
    data_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=5)
    buttons_frame = tk.Frame(root)
    buttons_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    return_home_button = tk.Button(root, text="Return to Home", command=return_to_home)
    return_home_button.pack(padx=5, pady=5, fill=tk.X)

    title_font = tkFont.Font(size=10, weight="bold")
    data_font = tkFont.Font(size=9)

    # Create labeled frames for GPS and MPU data with borders
    gps_frame = tk.LabelFrame(data_frame, text="GPS Data", font=title_font, padx=5, pady=5)
    gps_frame.pack(fill="both", expand="yes")
    mpu_frame = tk.LabelFrame(data_frame, text="MPU Data", font=title_font, padx=5, pady=5)
    mpu_frame.pack(fill="both", expand="yes")

    # Populate data_frame with GPS and MPU data labels
    tk.Label(data_frame, text="").pack()
    for var in ['lat_var', 'lng_var', 'alt_var', 'sat_var']:
        tk.Label(gps_frame, textvariable=gps_vars[var], anchor="w", font=data_font).pack(fill="x")
    tk.Label(data_frame, text="").pack()
    for var in ['accel_x_var', 'accel_y_var', 'accel_z_var', 'gyro_x_var', 'gyro_y_var', 'gyro_z_var']:
        tk.Label(mpu_frame, textvariable=mpu_vars[var], anchor="w", font=data_font).pack(fill="x")
    for name, body in celestial_objects.items():
        button = tk.Button(root, text=name, command=lambda body=body, name=name: button_command(body, name))
        button.pack(padx=5, pady=5, fill=tk.X)

    update_data()
    root.mainloop()
```

*Main function.* ***FIG39***

Here is where most of the graphical user interface is designed and updated with the all the functions created previously.

## 5.0 Critical Reflection Report

This project is the result of weeks of research, hundreds of problems solved, constant trial and error, analysis, testing and calibration. Overcoming the difficulties delt with communication differences between hardware and software, difficulties with the gearing and calibration phase, aligning everything using all the measurements and maths involved reflect the importance of proper testing and calibration.

## 6.0 Brief Description Exhibition Element

A showcase of the celestial object tracker being used outdoors in Realtime will display an effort to fulfil an engineering problem that will offer a tangible connection with the vastness of space.

## 7.0 Findings / Conclusions

The project can track objects from which have been chosen at an affordable rate. Analysis has revealed that combining GPS and MPU data can significantly improve tracking accuracy. Though signal interference can be a limitation as the device must use a clear line of sight to the sky.

## 8.0 Recommendations / Next Steps

- Making this project open source could bring new ideas and skills to develop this project further.

- Better accuracy could be achieved using larger teeth on the gearing making even more micro steps. Using a much larger ratio.
- Creating an interactive app to operate the device for ease of use.

- Using a much more powerful camera attached to a telescope for nighttime use to see more detail of the planets.

- Adding more celestial tracking objects to be available in the application, e.g. the moons of other solar system planets and stars.

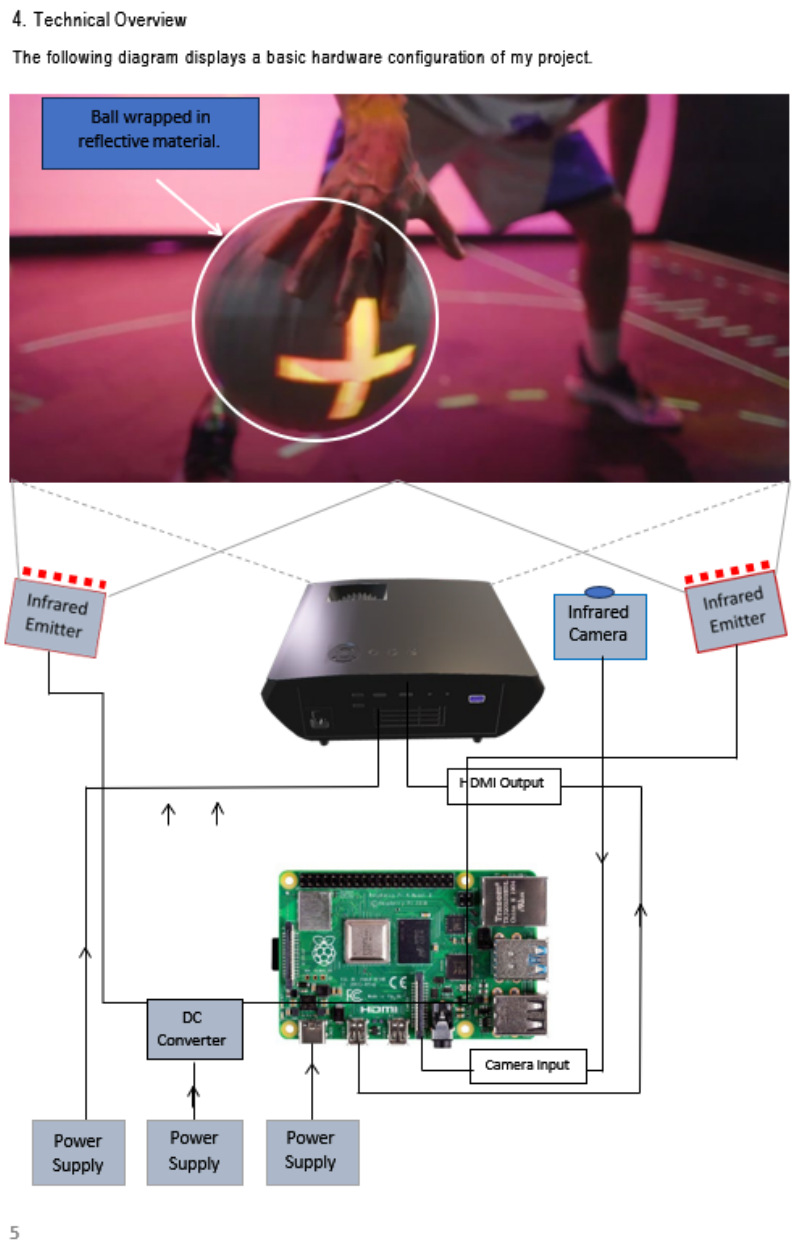
## 9.0 Evolution of the project

The original project was initially focused on an interactive projection mapping device, using Realtime tracking from infrared technology and a raspberry pi. The idea was a great one, though after testing and finding performance issues regards to data transfer speed from the camera input to the processing of information and then output on to the projector. This project would only be possible at the expense of a large processing unit, rendering the cost of the project to skyrocket. It was possible

to make it work but not at the speed it was initially intended for, running at 2 frames per second.

After considering multiple other options, the decision to change the project was inevitable. Conjuring up new ideas using the existing hardware from the original project was the only way to revive the situation. Time had become a critical factor at this point, this brought the idea of using existing skills and existing hardware to transfer the technology from the interactive projection mapping device the new celestial object tracking device. Immediately a fast-tracked development plan was created, a scope was prepared, and the project began.

Using existing skills in Fusion 360, programming, electronics, and project planning the project developed fast and the outcome is the result of 100% focus for weeks on end.




*Original Project Idea.* ***FIG54***

## 10.0 Testing and validation

Iterative testing with mock celestial bodies was essential when dealing with the complications of live data from GPS. Obtaining live data from the GPS in a testing environment proved to be a difficult task. Mock GPS data is used for testing the real MPU data to ensure the device is working as intended. The moon could be seen in the sky out the window, the first time the project worked with live data the devices X motor correctly pointed in the direction of the moon, though the Y motor did not seem to move.. To solve this problem, I needed to test each part of my project through a process of elimination.

An entire day of testing was done to ensure the code was working correctly, the hardware was all corrected and the problem was eventually solved when the decision to eliminate the motor, one of the motors had overheated as the day was a warm one beside the bedroom window. By ordering a new TMC2208 the problem was solved.

Further tests were done to ensure the alignment was correct, this came with another problem, to much slack was apparent on the Y motor when the tilt plate went beyond 80 – 90 degrees. The plate would fall, and the timing belt would not catch on to the teeth. To fix this, the 200mm timing belt was shortened to about 190mm which solved this issue.

To simulate a testing environment to know if the camera is point in the correct direction, an application called ‘Stellarium(a free open-sourced planetarium)’ can be used to visually verify the positions of celestial objects at the local time, these can be compared with the projects calculated values to ensure accuracy.

After this test is conducted, the ability to accurately track and display information on celestial objects within  a 2% margin of error with responsiveness of the project and feedback about reliability the success and failure of the project will be determined.

## 11.0 Meeting original specification

The original specification compared to the outcome of this project has deviated significantly, with the shifted focus from interactive projection mapping to celestial object tracking. Although the original idea was not as accurate as the outcome, the overall idea of aiming to improve educational tools through technology is met.

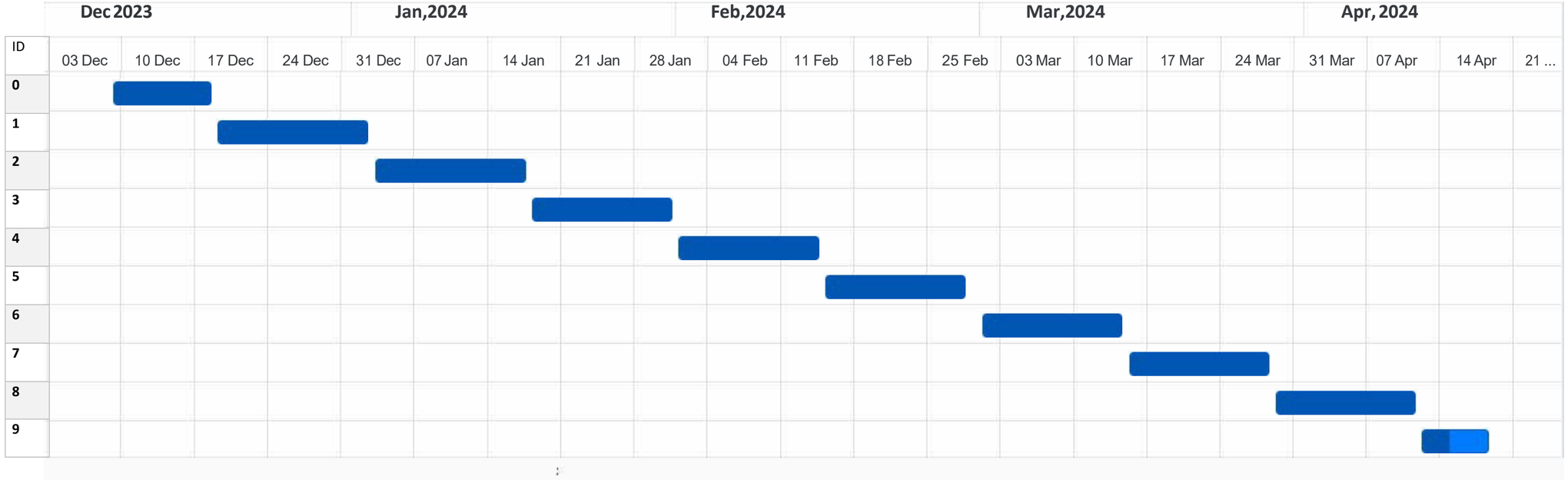
The valued outcome of the second project is perceived as more purposeful and brings more value and impact.

## 12.0 Gantt Chart Overview

The following is a Gantt chart of progress from the secondary project proposed.

Gantt chart information table:

| ID | Name | Start | Finish | Duration | % Complete |
|---|---|---|---|---|---|
| 0 | Research Phase | 09/12/2023 | 18/12/2023 | 10 days | 100 |
| 1 | Assembling and Initial Testing of Hardware | 19/12/2023 | 02/01/2024 | 15 days | 100 |
| 2 | Development of GPS, MPU, and Stepper Motor Control Software | 03/01/2024 | 17/01/2024 | 15 days | 100 |
| 3 | Integrating Hardware with Software for Data Acquisition and Motor Control | 18/01/2024 | 31/01/2024 | 14 days | 100 |
| 4 | Creation and Testing of Algorithms for Celestial Object Tracking | 01/02/2024 | 14/02/2024 | 14 days | 100 |
| 5 | Development of Graphical User Interface for System Control and Data Display | 15/02/2024 | 28/02/2024 | 14 days | 100 |
| 6 | Comprehensive Testing and Calibration of the Celestial Object Tracker | 01/03/2024 | 14/03/2024 | 14 days | 100 |
| 7 | Optimization for Performance, Usability, and Reliability | 15/03/2024 | 28/03/2024 | 14 days | 100 |
| 8 | Compilation of Project Documentation and Critical Reflection Report | 29/03/2024 | 11/04/2024 | 14 days | 100 |
| 9 | Preparing and Rehearsing for the Final Presentation | 12/04/2024 | 18/04/2024 | 7 days | 40 |


Dec 2023
Jan,2024
Feb,2024
Mar,2024
Apr, 2024
ID
03 Dec
10 Dec
17 Dec
24 Dec
31 Dec
07 Jan
14 Jan
21 Jan
28 Jan
04 Feb
11 Feb
18 Feb
25 Feb
03 Mar
10 Mar
17 Mar
24 Mar
31 Mar
07 Apr
14 Apr
21 ...
0
1
2
3
4
5
6
7
8
9

# 13.0 APPENDIX

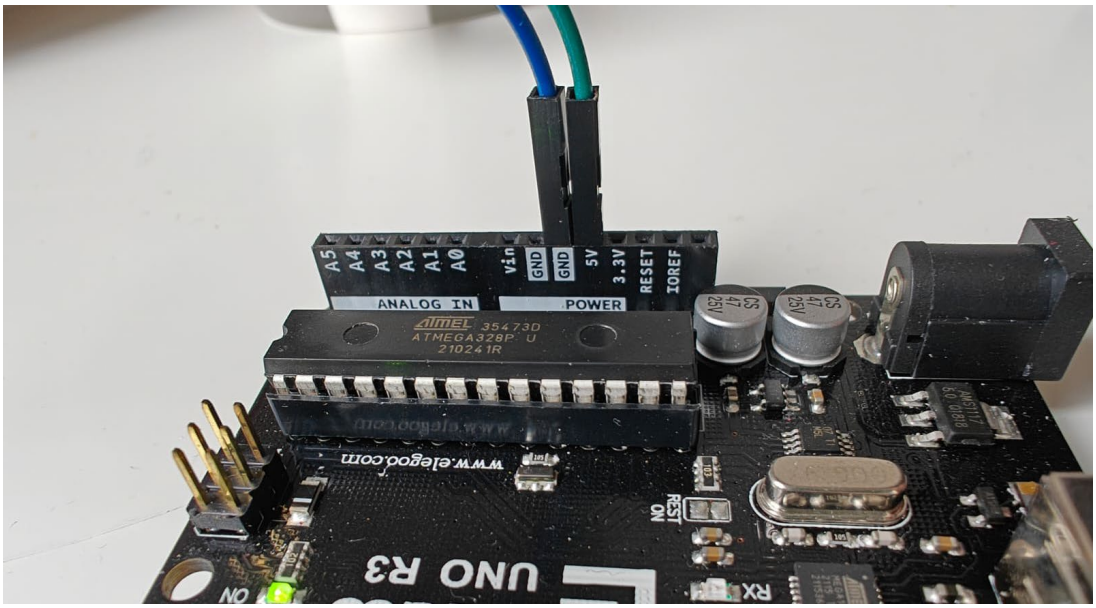
Serial output to Arduino IDE serial monitor.

After the first successful test of getting a satellite fix I was able to print all the information that is available on the GT-U7 device.

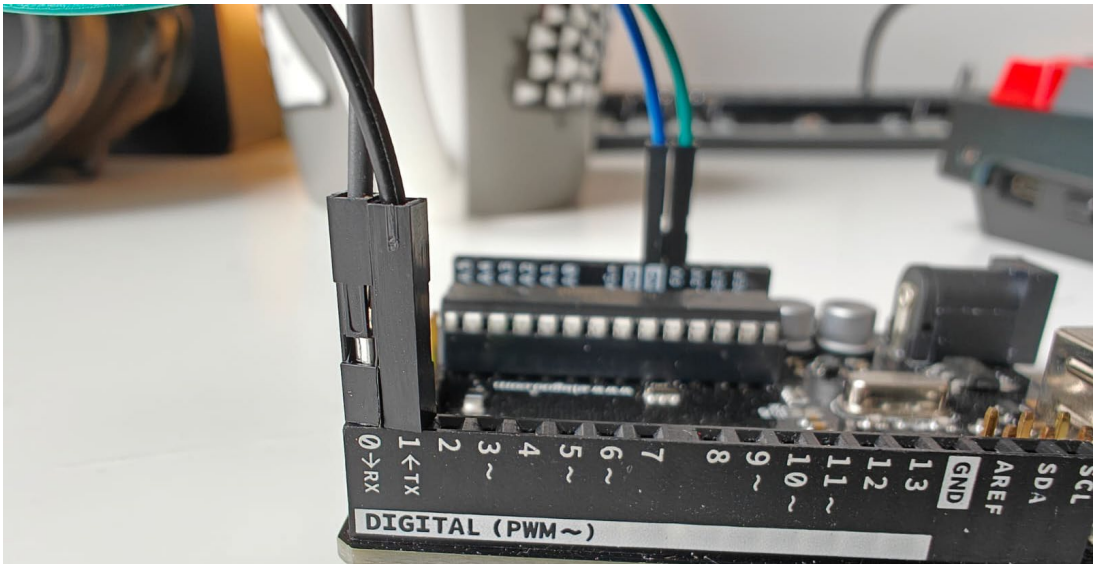
Connected via the TX serial pin from Arduino. Tested directly on to a PC

```
Satellite Count:
7
Latitude:
51.532001
Longitude:
0.075584
Speed MPH:
0.26
Altitude Feet:
62.01
```

*GPS Satellite output to Arduino serial monitor . **FIG40***

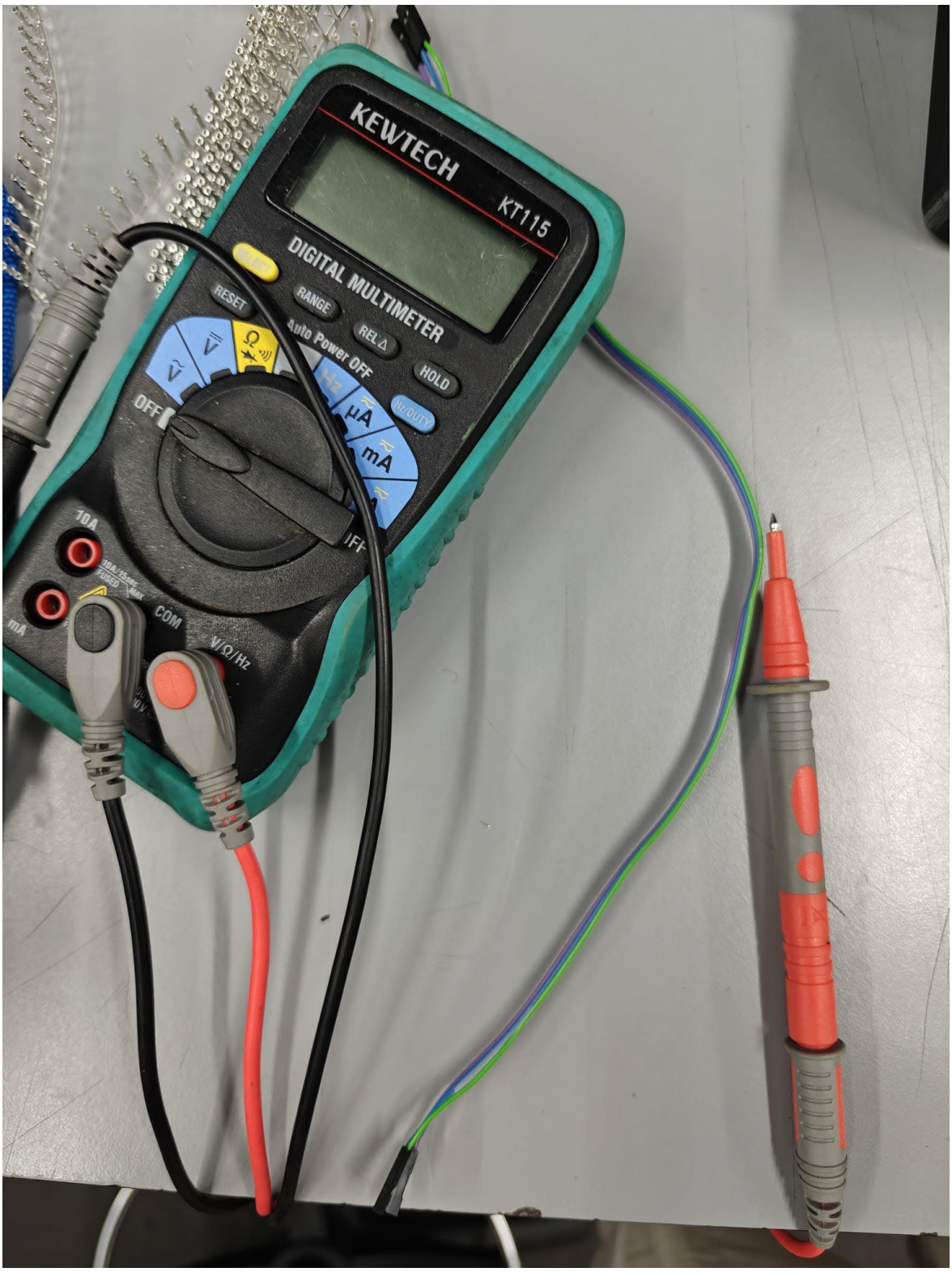



*Arduino GND and 5V connection to GPS. **FIG41***



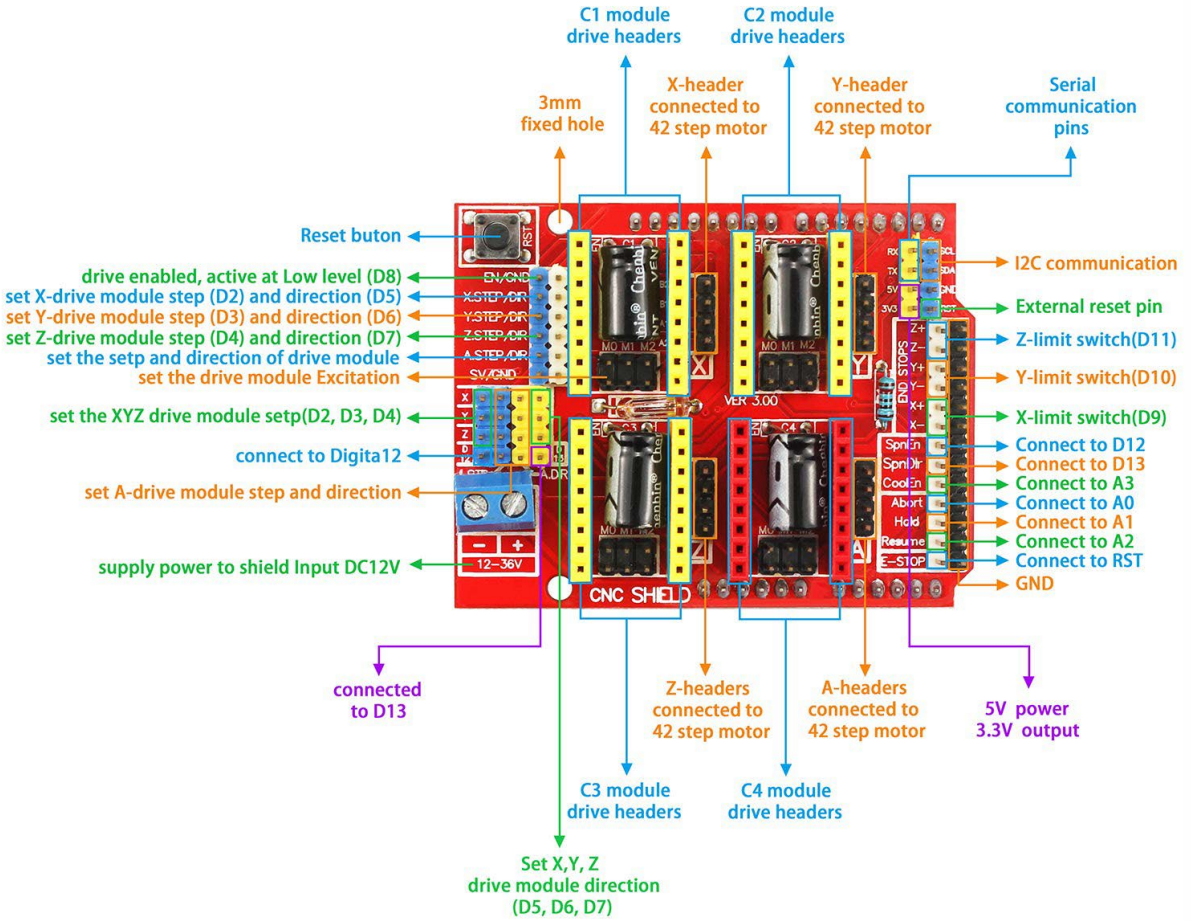

*Arduino pins 0(Receive) and 1(Transmit) . **FIG42***

Testing wires that were created for serial and power connection using a multimeter.



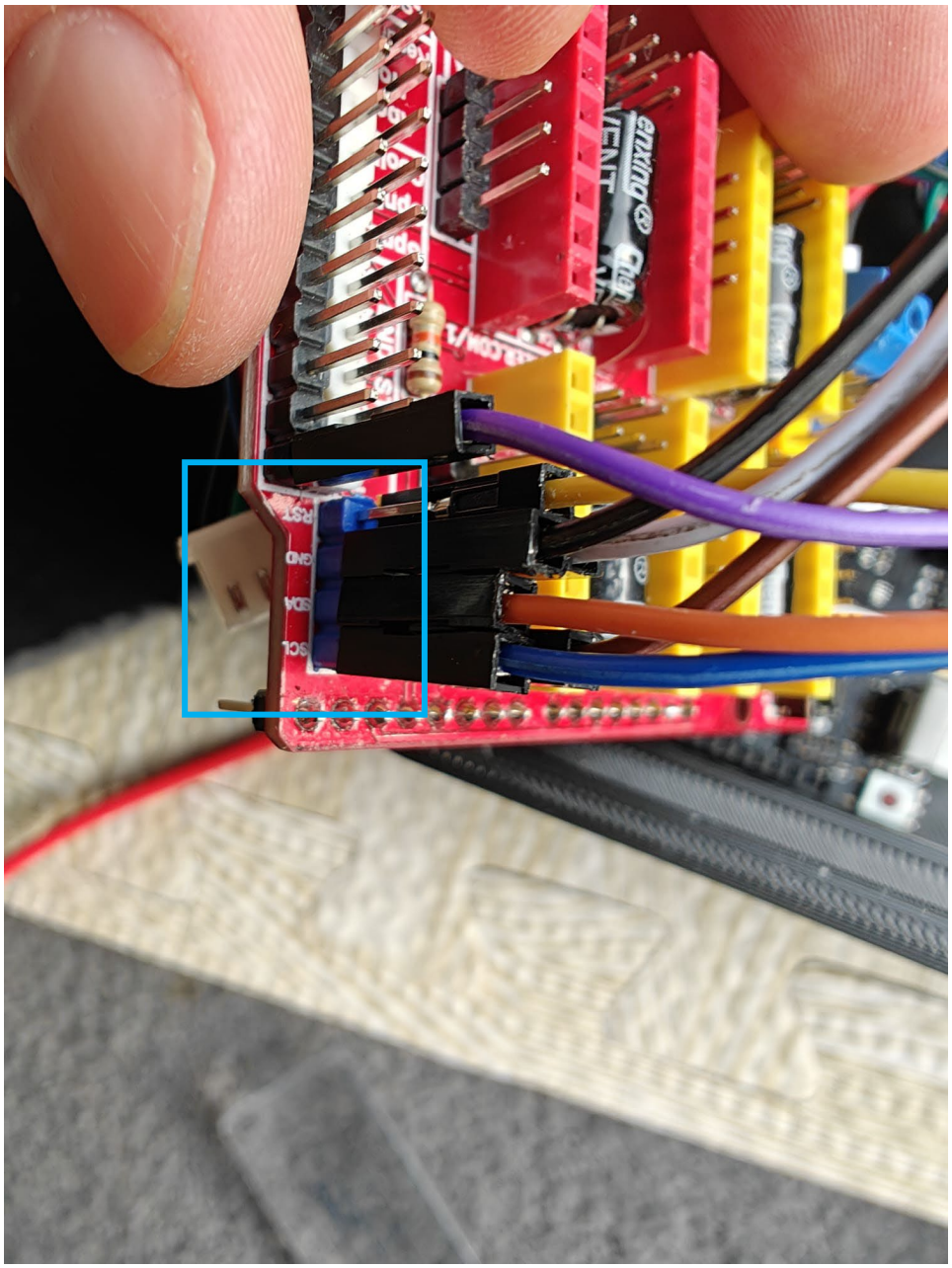

*Mulitmeter test.* ***FIG43***

Arduino CNC shield Diagram.



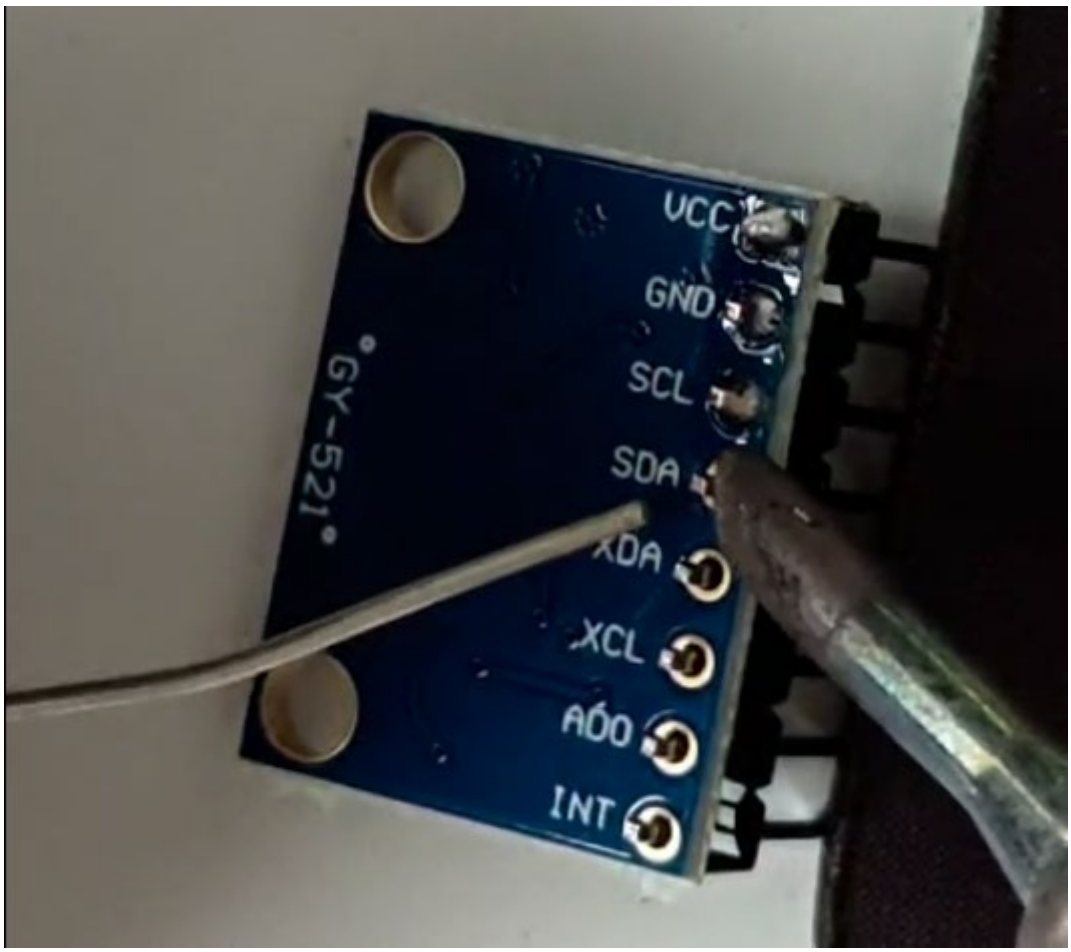

*CNC shield schematic.* ***FIG44***

Connecting I2C communication between CNC shield and MPU:



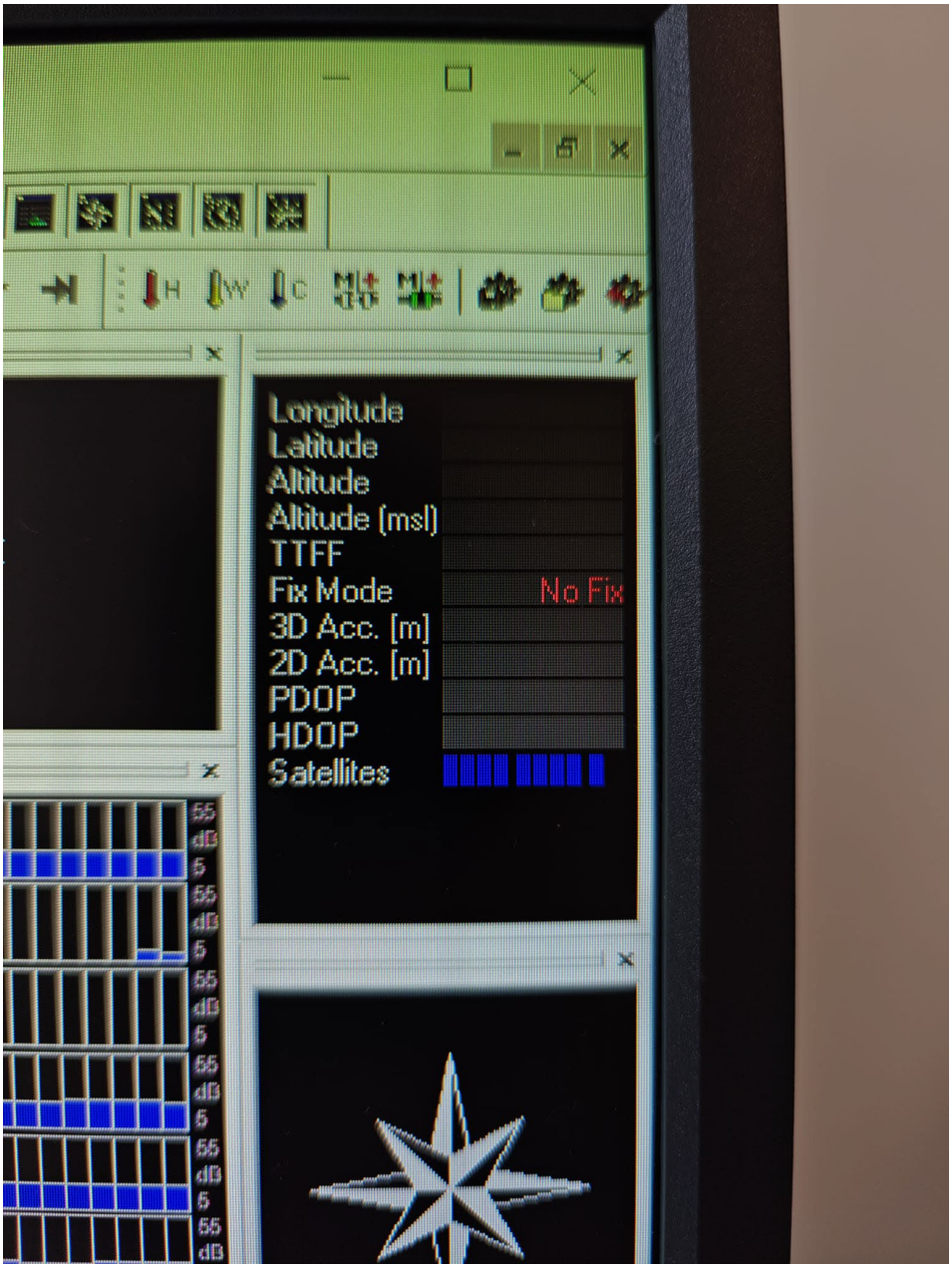
*CNC I2C pins.* ***FIG45***

Soldering the pin out connector to the MPU6050:



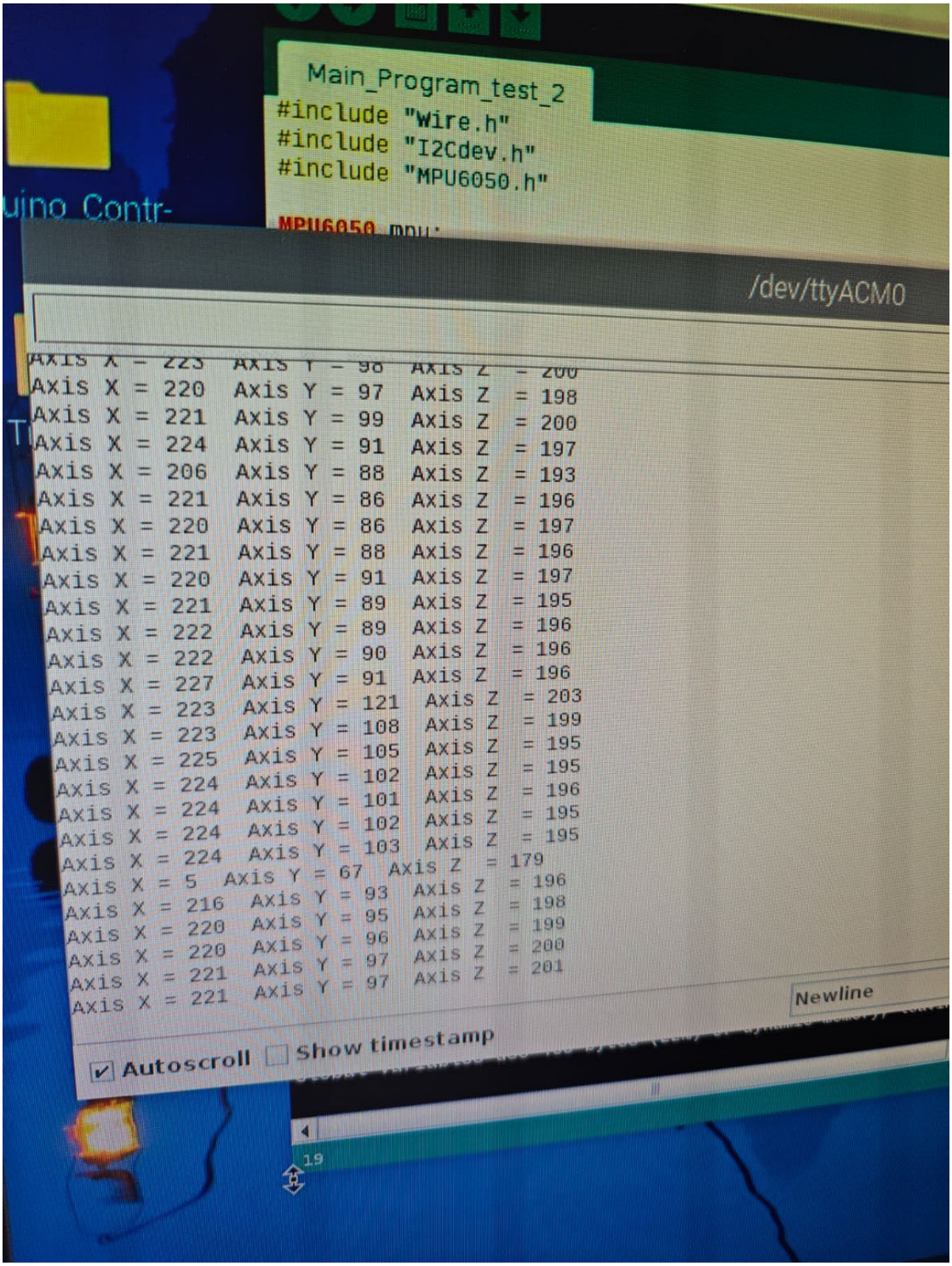

*Soldering pin out process.* ***FIG46***

No Fix to receive sattelite data:



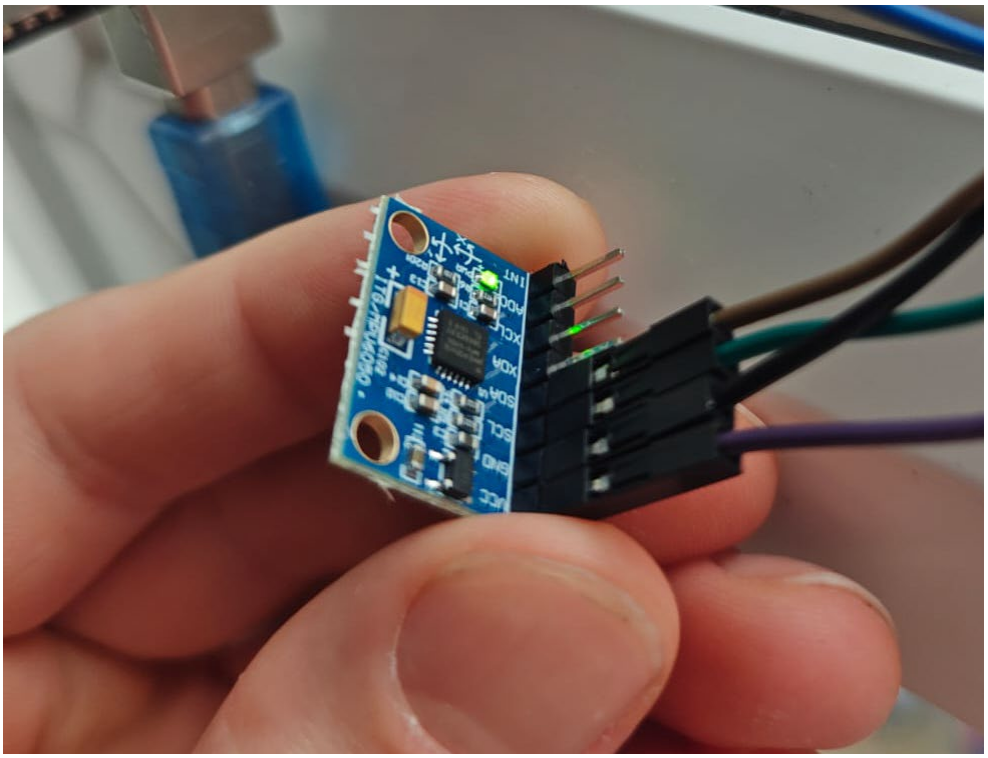

*No satellite fix.* ***FIG47***

First time reciving live MPU data using the MPU6050 Library. Using the example code that comes with the Library install.

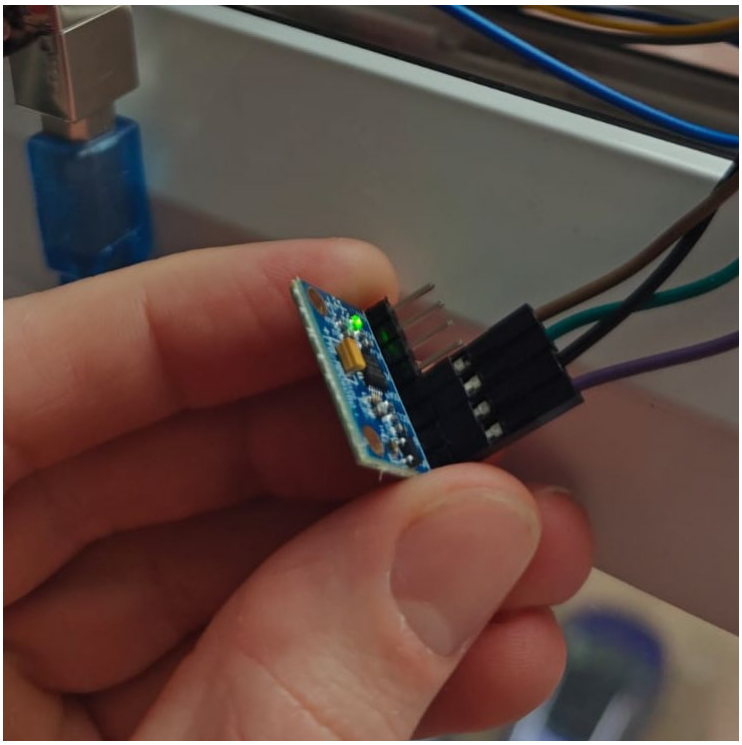



*MPU serial output.* ***FIG48***

Testing orientation of MPU:



*Orientation of MPU.* ***FIG49***



*Orientation of MPU(2).* ***FIG50***

Outdoor GPS tesing:


*Outdoor setup.* ***FIG51***

U-Blox U-Center software colour codes:

## 4.1 Color and satellite coding scheme

In the graphical views and some docking windows, colors are used to indicate data quality. Table 1 shows the color codes for graphical views depending on the quality of the navigation solution.

| | Color | Meaning |
|---|---|---|
| + | Yellow | Current value |
| + | Orange | Valid 3D navigation fix + Dead Reckoning |
| + | Green | Valid 3D navigation fix |
| + | Cyan | Valid 2D navigation fix |
| + | Magenta | Dead Reckoning fix |
| + | Blue | Degraded navigation fix |
| + | Red | No or invalid navigation fix |

**Table 1: Color-coding scheme for graphical views**

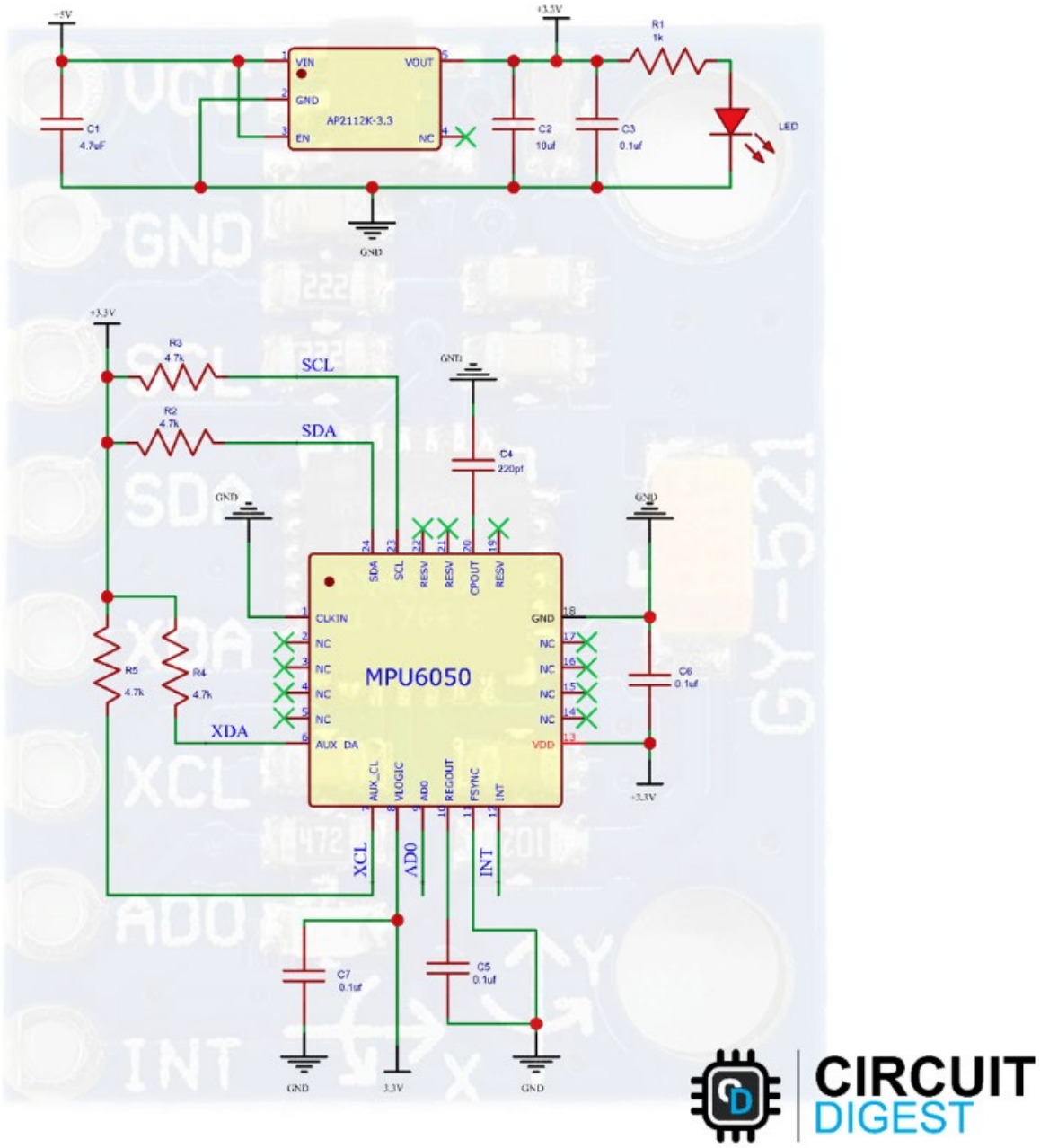
Table 2 gives the color-coding scheme for the docking windows and sky view. It indicates the state of each satellite.

| | Color | Meaning |
|---|---|---|
| | Green | Satellite used in navigation (with Ephemeris) |
| | Olive | Satellite used in navigation (with Ephemeris and PPP) |
| | Dark Green | Satellite used in navigation (with aiding data: AssistNow Autonomous, AssistNow Online/Offline) |
| | Cyan | Satellite signal available, available for use in navigation |
| | Blue | Satellite signal available, not available for use in navigation |
| | Red | Satellite signal not available |

**Table 2: Color-coding scheme for the docking windows and sky view**

*Colour code of U-Center.* ***FIG52***

MPU6050 Circuit design [Joseph, 2022]:




*MPU6050 Circuit.* ***FIG53***

# 14.0 Bibliography


Arduino. (2022, 06 15). *SoftwareSerial Library*. Retrieved from Arduino: https://docs.arduino.cc/learn/built-in-libraries/software-serial/

Arduino. (2023, 12 05). *Wire*. Retrieved from Arduino: https://www.arduino.cc/reference/en/language/functions/communication/wire/

Cats, E. (2024, March 17). *mpu6050*. Retrieved from GitHub: https://github.com/electroniccats/mpu6050

Derek. (2020, March 6). *Automated #RaspberryPi Planet Tracking GOTO Telescope*. Retrieved from YouTube: https://www.youtube.com/watch?v=v2pqya3c1_M&t=922s

Developers, S. (2024, March 31). *Skyfield: Elegant Astronomy for Python*. Retrieved from Skyfield: https://rhodesmill.org/skyfield/

Foundation, P. S. (2024, March 31). *tkinter — Python interface to Tcl/Tk*. Retrieved from Python: https://docs.python.org/3/library/tkinter.html

GNSS, I. (2010, December 2). *Measuring GNSS Signal Strength*. Retrieved from insidegnss: https://insidegnss.com/measuring-gnss-signal-strength/

Joseph, J. (2022, May 16). *How Does the MPU6050 Accelerometer & Gyroscope Sensor Work and Interfacing It With Arduino*. Retrieved from circuitdigest: https://circuitdigest.com/microcontroller-projects/interfacing-mpu6050-module-with-arduino#:~:text=The%20MPU6050%20is%20a%20Micro,of%20a%20system%20or%20object.

KING, B. (2019, February 26). *RIGHT ASCENSION & DECLINATION: CELESTIAL COORDINATES FOR BEGINNERS*. Retrieved from skyandtelescope: https://skyandtelescope.org/astronomy-resources/right-ascension-declination-celestial-coordinates/

Lee, J. (2019, November 1). *TinyGPS*. Retrieved from GitHub: https://github.com/neosarchizo/TinyGPS

Mackenzie, C. E. (1980, March 14). *Serial_communication*. Retrieved from wikipedia: https://en.wikipedia.org/wiki/Serial_communication

Ronquillo, A. (n.d., n.d. n.d.). *A Beginner's Guide to the Python time Module*. Retrieved from Real Python: https://realpython.com/python-time-module/

Rowberg, J. (2024, March 31). *Forums*. Retrieved from i2cdevlib: https://www.i2cdevlib.com/

Shilleh. (2023, Janruary 2). *How the MPU6050 Works | MEMS Overview*. Retrieved from YouTube: https://www.youtube.com/watch?app=desktop&v=MsyqsOUBQuU

*Stepper*. (2018, August 22). Retrieved from GitHub: https://github.com/arduino-libraries/Stepper

U-Blox. (2022, May 24). *u-center_user_guide*. Retrieved from U-blox: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.u-blox.com/sites/default/files/u-center_Userguide_UBX-13005250.pdf

Figure 10. https://m.media-amazon.com/images/I/51jU-C7lgLL._AC_.jpg

Close up image of MPU circuit, Front Page of report - https://circuitdigest.com/microcontroller-projects/interfacing-mpu6050-module-with-arduino#:~:text=The%20MPU6050%20is%20a%20Micro,of%20a%20system%20or%20object.

Image of Saturn, Executive Summary page - https://wallpapers.com/images/featured/jupiter-4k-u0vxq4j32tby8tox.jpg