Advanced DarkSky Observatory Synchronization

NILAPALLE RUSHIKESH SHESHERAO
Roll No: ES20BTECH11022
IIT Hyderabad, Sangareddy, Telangana

Under the Supervision of Dr. Mayukh Pahari

B-Tech Project Report

May 2, 2024

Declaration

I, **Rushikesh**, hereby declare that the project proposal entitled "Advance DarkSky Observatory Synchronization," presented in this document, is my original work. Wherever the ideas or words of others are used, appropriate citations and references have been provided to acknowledge the original sources. I attest that I have upheld the principles of academic honesty and integrity throughout the development of this proposal.

I understand the consequences of academic dishonesty and am committed to maintaining the highest standards of ethical conduct in my academic pursuits. Any violation of these principles may result in disciplinary action by the institute and may also incur penalties from the sources that have not been properly cited or from which proper permission has not been obtained.

Nilapalle Rushikesh Shesherao
ES20BTECH11022

Approval Certificate

This thesis entitled "**Advance DarkSky Observatory Synchronization**" by **Nilapalle Rushikesh Shesherao** (Roll No: ES20BTECH11022) is approved for the degree of B-Tech from Indian Institute of Technology Hyderabad and also is to certify that the work contained in this project submitted by **Nilapalle Rushikesh Shesherao** to Indian Institute of Technology Hyderabad towards the partial requirement of Engineering Science has been carried out by him under the supervision of **Dr. Mayukh Pahari** and that it has not been submitted elsewhere for the award of any degree.

Supervisor, Dr. Mayukh Pahari

Acknowledgment

I am profoundly grateful to my supervisor, Dr. Mayukh Pahari, for their invaluable guidance, insightful feedback, and unwavering support throughout the duration of my project. Their expertise and encouragement played a pivotal role in shaping the direction and success of this endeavor. Furthermore, I extend my deepest appreciation to my family and friends whose unwavering support and encouragement were instrumental in helping me navigate through challenges and stay motivated during the course of this project.

Abstract

The automation of astronomical observatories represents a significant advancement in the field of astronomy, enhancing the efficiency, accessibility, and capabilities of observational facilities. This project focuses on the design, development, and implementation of an automated observatory system for the campus observatory, equipped with a 20-inch robotic telescope and a CCD camera. The system integrates hardware components such as the telescope, mount, dome control, and gyro-sensors, along with software for controlling and coordinating their operation.Future enhancements and applications may include the integration of additional instruments or sensors, expansion of remote access capabilities, and collaboration with other observatories for joint observations and data sharing.

Github Link :
https://github.com/Rushikesh2311/Observatory-Dome-Synchronization

Contents

1 Advance DarkSky Observatory Setup

Observatories have two main components:a)Observatory Dome b)Telescope

1.1 Telescope

Advance DarkSky observatory hosts a giant 20inch Reflecting Telescope with Automated Dobsonian mount.The motorized operation of the telescope in both Altitude (Alt) and Azimuth (Az) axes further enhances its usability. This allows the telescope to slew automatically to designated celestial coordinates, providing users with a seamless observing experience. Additionally, the inclusion of Freedom-Find Dual-Encoder technology empowers users to make manual adjustments to the telescope's position without compromising its alignment or losing track of their observational targets. This combination of motorized precision and manual control gives users unparalleled convenience and flexibility during observing sessions.

1.2 Observatory Dome

Our observatory dome operates manually, requiring control of its movement through a mechanical switch located inside the dome itself. Equipped with two 24-volt DC motors, the dome facilitates rotation, while an additional motor manages the opening and closing of the dome window. Power for the dome is sourced from nearby solar panels, ensuring sustainable and environmentally-friendly operation.

1.3 Need of Automated Synchronization System

The presence of a manual switch poses limitations on the capabilities of our 20-inch robotic telescope with go-to function. This manual control mechanism imposes unnecessary burdens on manpower and hinders the full potential of the telescope system. Managing the telescope's current celestial target and coordinating dome movement accordingly becomes challenging, leading to potential errors and inefficiencies.

Moreover, the manual operation of the switch requires continuous monitoring and adjustment, particularly when observing celestial targets for extended periods. This constant intervention not only adds to the workload but also introduces the risk of errors and inconsistencies in data collection.

In addition, automated synchronization systems are standard in modern observatories due to their essential role in precisely aligning the dome

with the telescope's target. Achieving this level of precision manually is inherently challenging and prone to errors, highlighting the need for an automated solution.

By implementing an automated synchronization system, we aim to streamline operations, reduce manual intervention, and enhance the accuracy and efficiency of our observatory's observational capabilities. This will not only alleviate the burden on manpower but also ensure precise alignment between the dome and telescope targets, ultimately optimizing the quality and reliability of our research data.

2 Hardware requirements of the Synchronization system

2.1 Relay

Electromagnetic relays are devices that utilize an electromagnet to control the flow of electrical current in a circuit. These relays consist of a coil, an armature, and one or more sets of contacts. When current flows through the coil, it generates a magnetic field, which attracts the armature and causes the contacts to either close or open, depending on the relay's configuration.

In the context of our observatory project, electromagnetic relays are being employed to replace the manual switch for controlling the movement of the observatory dome. By using electromagnetic relays, we can achieve digital control over the dome's rotation and window operation. This allows for precise and reliable positioning of the dome in alignment with the telescope.

Figure 1: Relay

We used relay in our project as shown in the above figure.Specifications of the same are as follows :

Type: 8 Channel Isolated Relay Module
Input Voltage: 5V
Maximum Current Capacity: 10A
Opto-Isolation: The module uses opto-couplers to provide electrical isolation between the input (control signal) and output (relay switching).
Compatibility: The module is designed to be compatible with a wide range of microcontrollers.

2.2 ESP32

The ESP32 is a versatile microcontroller developed by Espressif Systems, tailored for Internet of Things (IoT) applications. It boasts a dual-core Tensilica Xtensa LX6 microprocessor, enabling concurrent processing and multitasking. This architecture enhances performance, enabling efficient handling of complex tasks.

One of the ESP32's standout features is its built-in Wi-Fi and Bluetooth connectivity. This capability facilitates seamless communication with Wi-Fi networks for internet access and interaction with other devices. Additionally, Bluetooth connectivity enables wireless communication with peripherals like smartphones and sensors.

The microcontroller offers a rich set of peripheral interfaces, including SPI, I2C, UART, GPIO, ADC, and DAC. These interfaces facilitate integration with various sensors, displays, and actuators, making the ESP32 suitable for diverse IoT and embedded applications.

Despite its powerful capabilities, the ESP32 prioritizes energy efficiency. Its advanced power management features, such as sleep modes and voltage scaling, minimize power consumption. This makes it suitable for battery-powered and energy-efficient applications.

Programming the ESP32 is flexible, supporting various languages and development environments like Arduino IDE, ESP-IDF, and MicroPython.

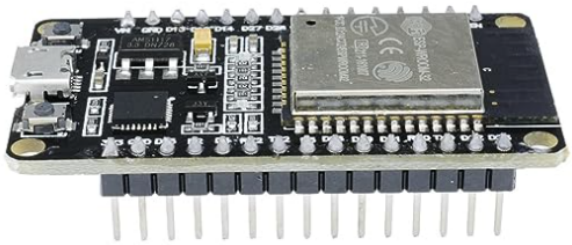

Figure 2: ESP32

2.3 MPU6050 (gyrosensor)

The MPU6050 is a versatile motion-tracking sensor widely used in various applications, including robotics, gaming, and motion-based input devices. Developed by InvenSense (now TDK), the MPU6050 combines a gyroscope and an accelerometer into a single integrated circuit, providing accurate motion sensing capabilities.

The gyroscopic component of the MPU6050 enables precise measurement of rotational motion in three axes: pitch, roll, and yaw. This allows the sensor to detect changes in orientation and angular velocity with high accuracy and responsiveness. The accelerometer, on the other hand, measures linear acceleration along the same axes, complementing the gyroscopic measurements to provide a comprehensive picture of the sensor's motion.

One of the key features of the MPU6050 is its digital interface, which allows it to communicate with microcontrollers and other devices using standard communication protocols such as I2C (Inter-Integrated Circuit). This makes it easy to interface the sensor with a wide range of microcontrollers, including popular platforms like ESP32, Arduino and Raspberry pie.

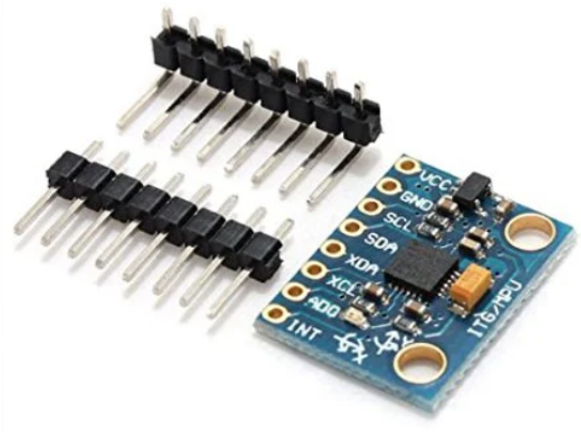

Figure 3: MPU6050

2.4 HMC5883L Magnetometer

The HMC5883L is a three-axis magnetometer (compass) sensor manufactured by Honeywell. It is commonly used to measure the strength and direction of a magnetic field in three dimensions.

2.4.1 Features

- **Three-Axis Magnetometer:** The HMC5883L can measure magnetic fields along three orthogonal axes: X, Y, and Z.
- **Digital Output:** The sensor provides digital output, making it easy to interface with microcontrollers or other digital systems via I2C communication protocol.
- **High Sensitivity:** It has high sensitivity with a wide measurement range, typically ±1.3 to ±8 Gauss.
- **Integrated Temperature Sensor:** The HMC5883L includes an integrated temperature sensor for temperature compensation.
- **Applications:** Common applications include compasses, navigation systems, robotics, and orientation detection in electronic devices.

To use the HMC5883L magnetometer in a project, you typically need to perform the following steps:

1. Initialize the sensor by configuring its settings, such as measurement range and data output rate.
2. Read data from the sensor's registers via I2C communication.
3. Convert the raw sensor data to meaningful magnetic field values using calibration and scaling techniques.
4. Interpret the magnetic field data to determine orientation or heading information.

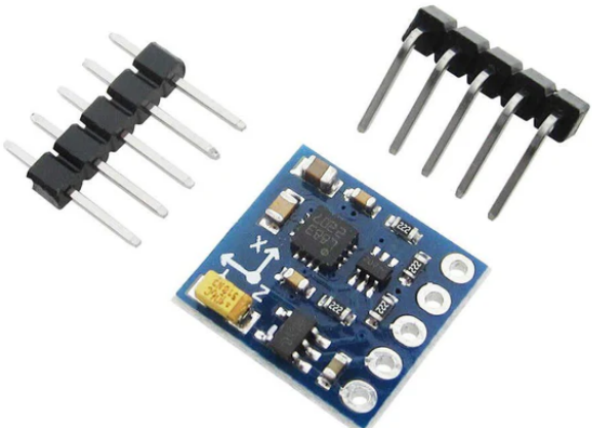

Figure 4: Magnetometer

3 System Integration

3.1 Observatory Control System

OCS is the overall system for the automatic synchronization of dome with telescope. It consists of 3 components:

1. Motor Control Module
2. Telescope Module
3. Dome Module

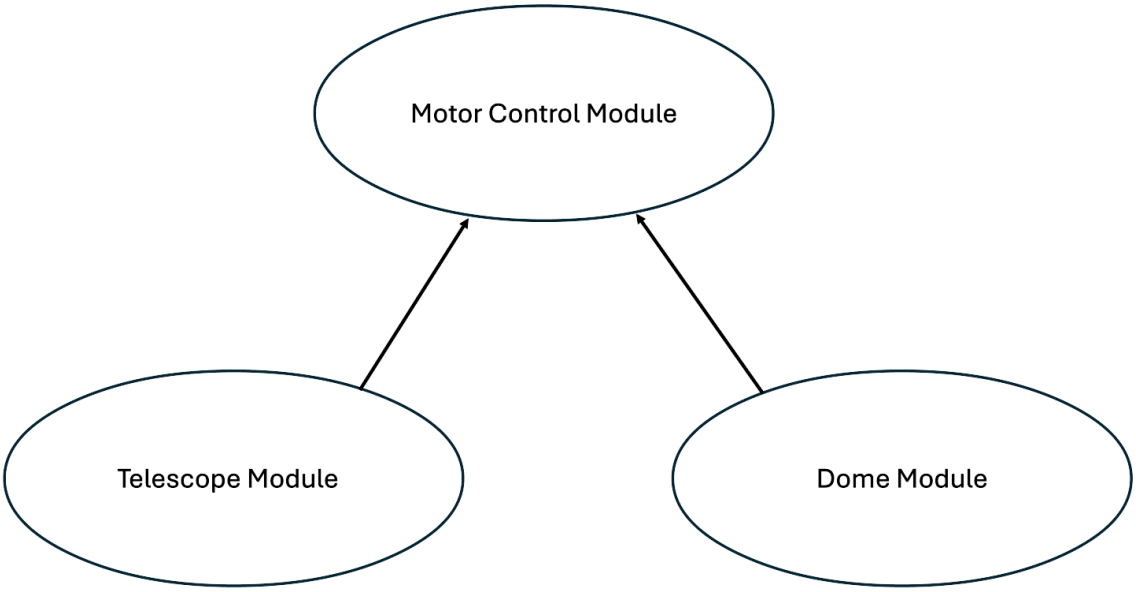

Figure 5: Observatory Control System

3.2 Motor Control Module

3.2.1 Functionalities

- Act as a Wifi connection point.
- Input rotation details from Telescope Module.
- Input rotation details from Dome Module.
- Calculate the angle difference.
- Rotate the motor according to the difference.

3.2.2 Hardware

- ESP32
- Relay
- breadboard
- Motor wires connection
- Jumping wires

3.2.3 Circuit Diagram

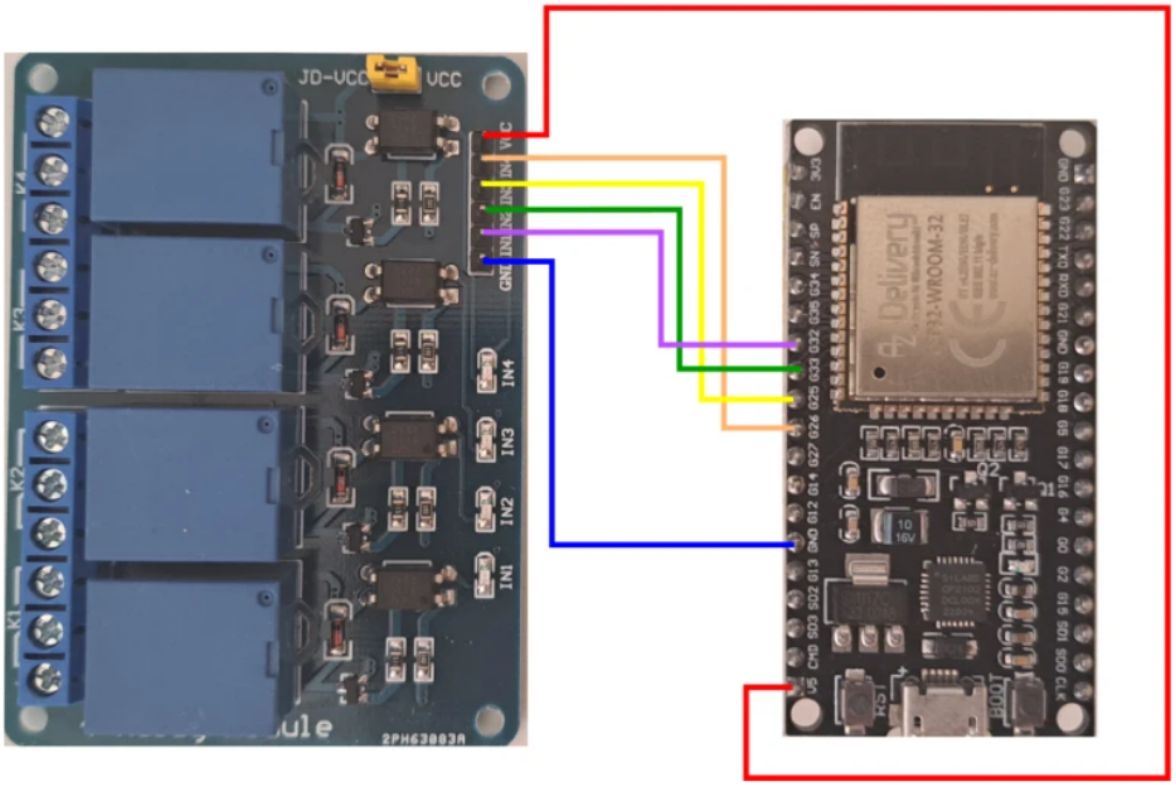

Figure 6: Motor Control Module Circuit Diagram

3.2.4 Circuit Image

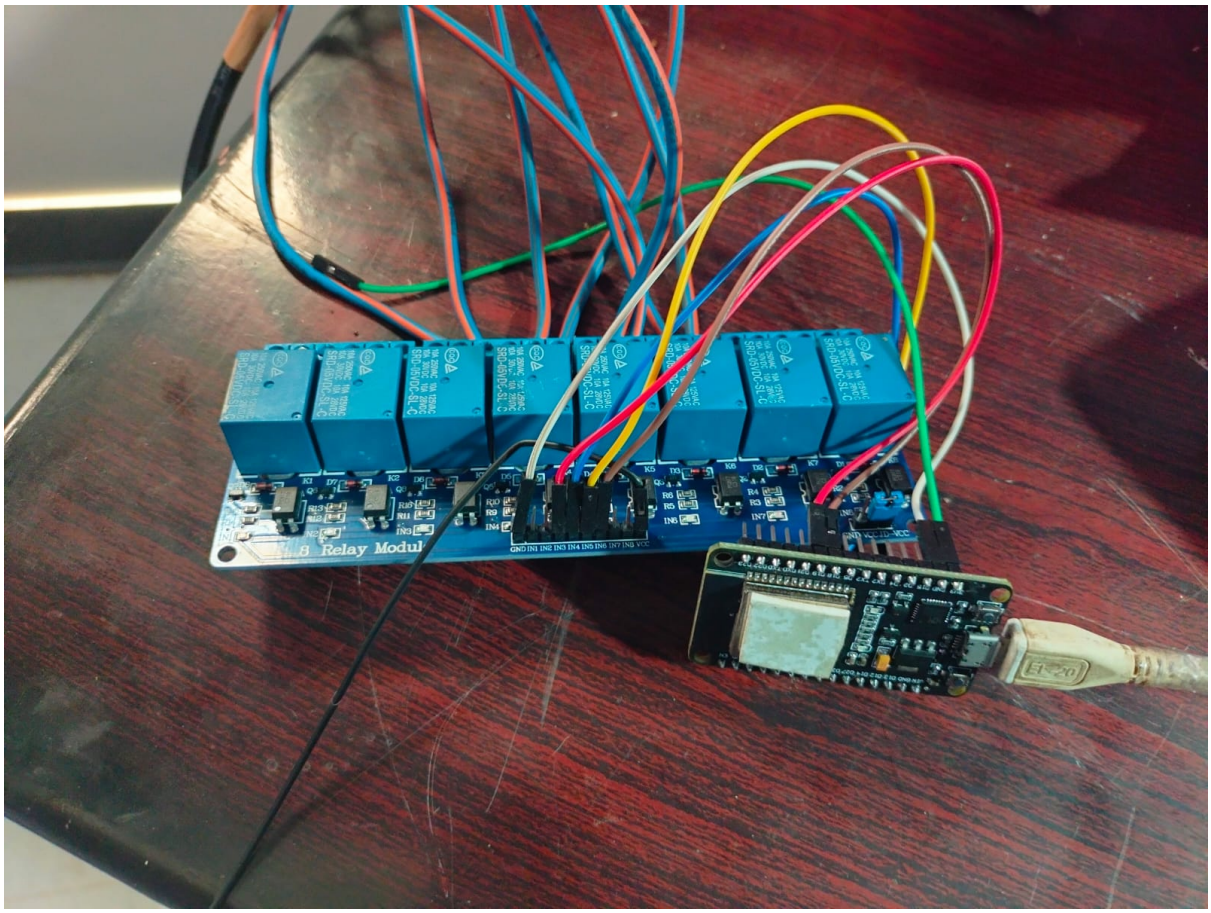

Figure 7: Motor Control Module Circuit Image

3.3 Telescope Module

3.3.1 Functionalities

- Establish connection with Motor Control Module.
- Measure the angle of rotation of Telescope.
- Send the rotation data to Motor Control Module via http requests.

3.3.2 Hardware

- ESP32
- HMC5883L
- breadboard
- Jumping wires

3.3.3 Circuit Diagram

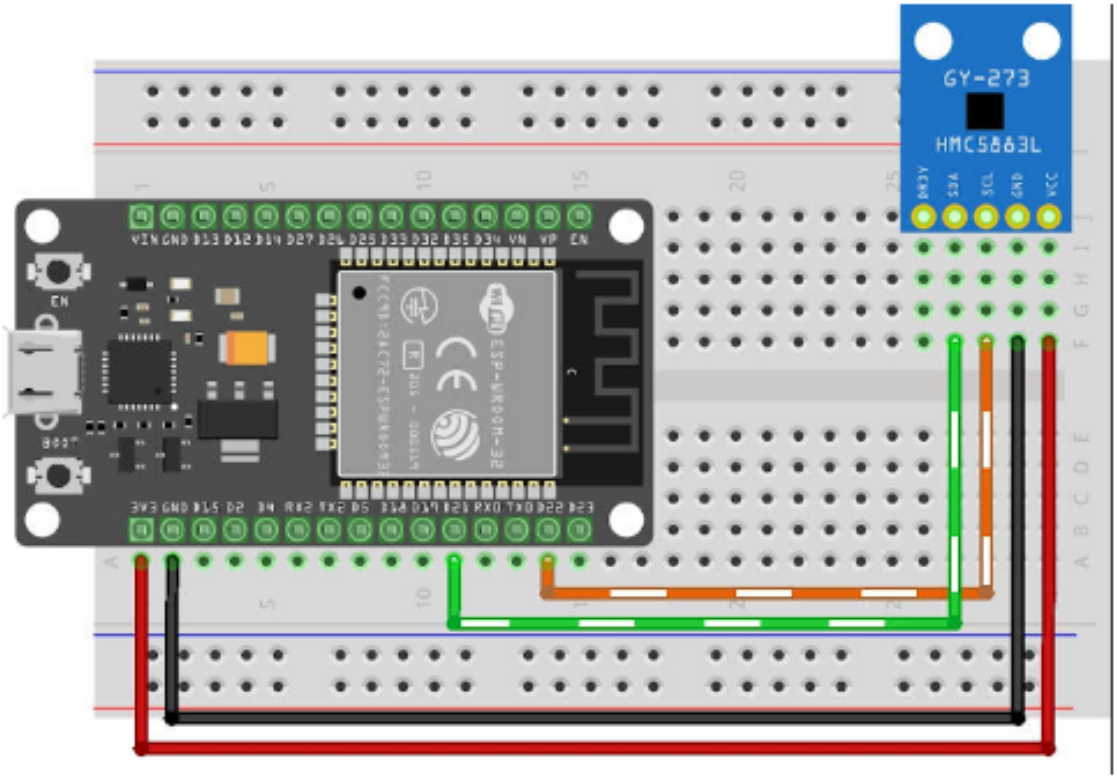

Figure 8: Telescope Module Circuit Diagram

3.3.4 Circuit Image

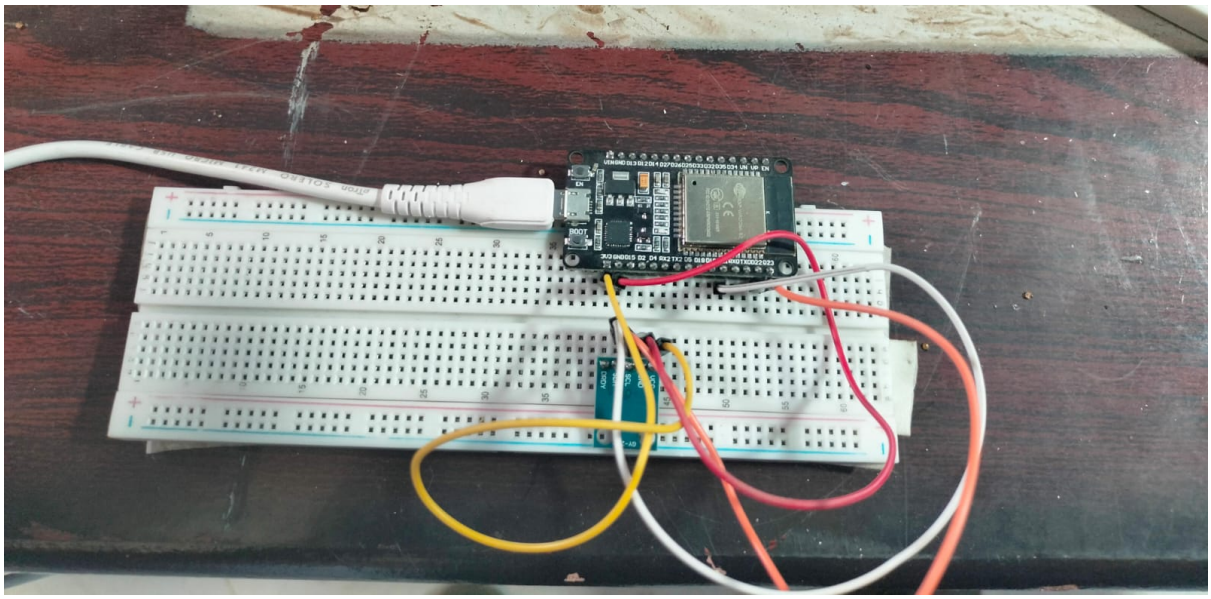

Figure 9: Telescope Module Circuit Image

3.4 Dome Module

3.4.1 Functionalities

- Establish connection with Motor Control Module.
- Measure the angle of rotation of Dome.
- Send the rotation data to Motor Control Module via http requests.

3.4.2 Hardware

- ESP32
- HMC5883L
- breadboard
- Jumping wires

3.4.3 Circuit Diagram

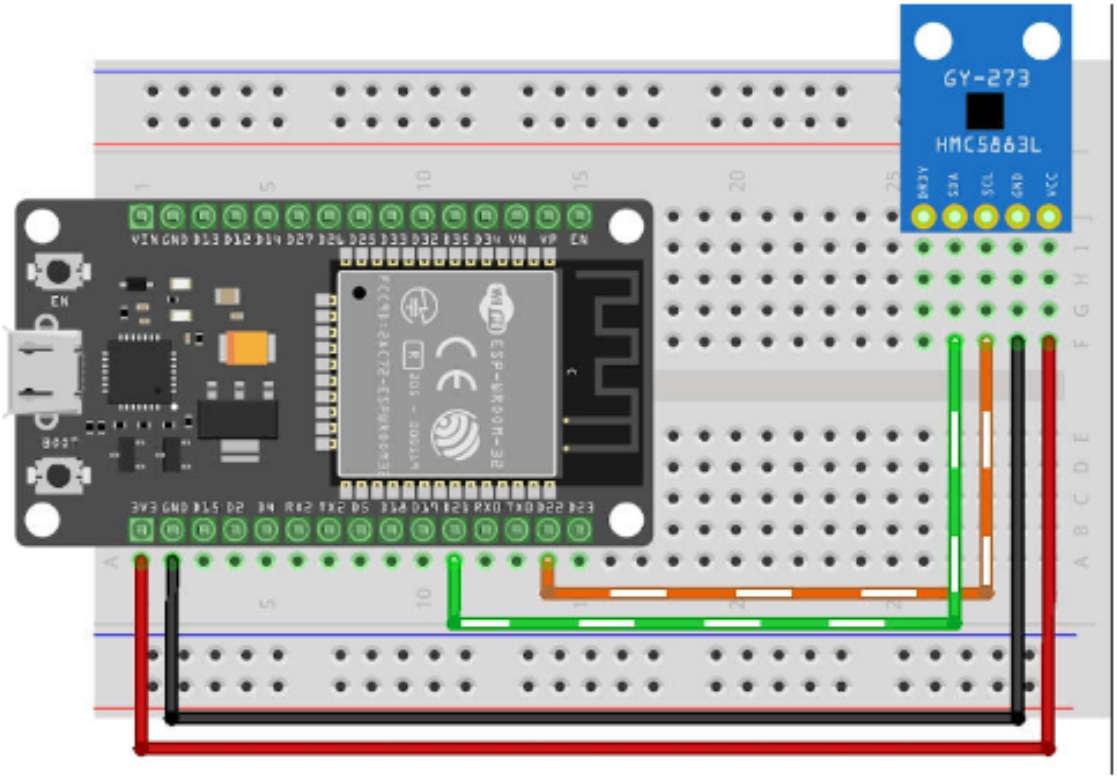

Figure 10: Dome Module Circuit Diagram

3.4.4 Circuit Image

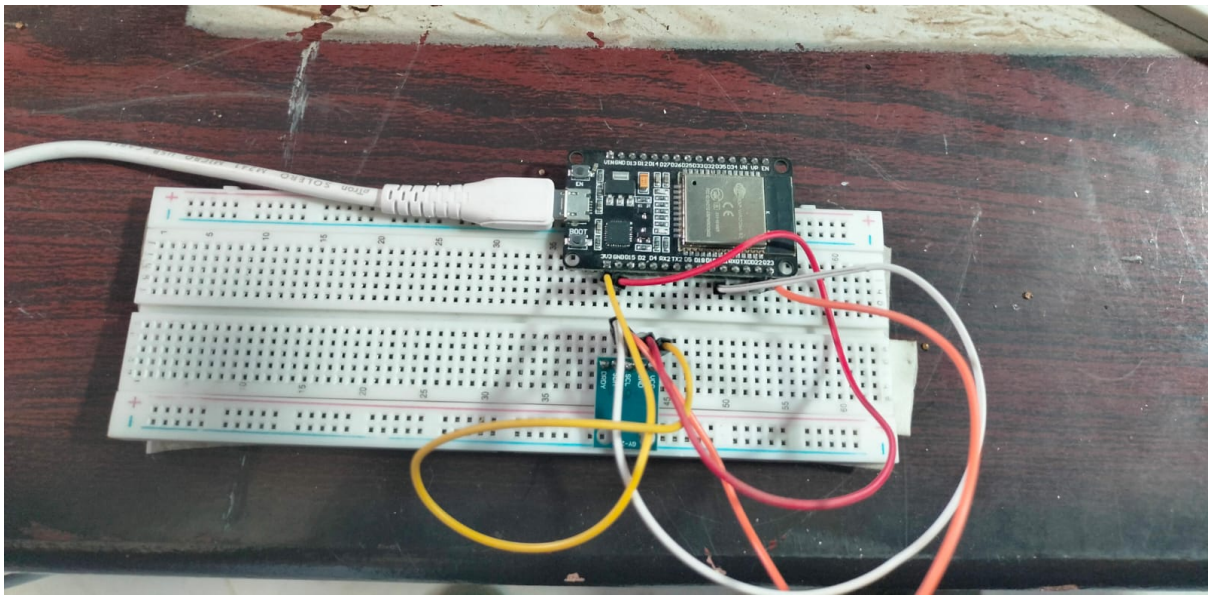

Figure 11: Dome Module Circuit Image

4 Control System Design

4.1 Motor Control Module

4.1.1 Wifi Access Point

ESP32 can act as wifi access point and sustain upto 8 active connections.WiFi.h library has this functionality.

Code for the same is as follows:

```
#include <WiFi.h>
const char* ssid = "yourAP";
const char* password = "yourPassword";
WiFiServer server(80);
```

Explanation:

- **Include Library:** The code begins by including the WiFi library (`WiFi.h`), which provides functions and classes for configuring and managing Wi-Fi connections on ESP32.
- **SSID and Password:** The `ssid` variable stores the name of the Wi-Fi access point (AP) that will be created by the ESP32. Replace "yourAP" with the desired name for your access point. The `password` variable stores the password required to connect to the access point. Replace "yourPassword" with the desired password.
- **WiFiServer Object:** The `WiFiServer` object named `server` is created to handle incoming client connections on port 80. This means that the ESP32 will listen for connections from clients (e.g., web browsers) on port 80, which is the default port for HTTP communication.
- **Access Point Creation:** With the provided credentials (`ssid` and `password`), the ESP32 will create a Wi-Fi access point with the specified SSID and password. Devices can connect to this access point using the provided credentials.

4.1.2 Handling HTTP requests from telescope/dome

```
while (client[i].available()) {
    Serial.println("h2");
    char c = client[i].read();
    if (c == '\n' || c == '\r' || c == '/') {
        if (currentLine.length() > 0) {
            received = handleMessage(currentLine);
            currentLine = "";
        }

        if (!sender.isEmpty() && received) {
            messageSummary();
            if (sender == "telescope") {
                angle_telescope = angle;
            } else if (sender == "roof") {
                angle_roof = angle;
            }
            break;
        }
    } else {
        currentLine += c;
    }
}
```

Explanation:

- **Loop Condition:** The code snippet starts with a while loop that checks if there are bytes available to read from the client connection.
- **Reading Bytes:** Inside the loop, a byte is read from the client connection using the `read()` function.
- **End of Line Check:** The code checks if the read byte is a newline character, carriage return character, or a forward slash.
- **Processing Message:** If the byte indicates the end of a line or message field, the current line is processed using the `handleMessage()` function.
- **Message Summary:** After processing the message, if both the sender and the message were received successfully, a summary of the message is generated using the `messageSummary()` function.

- **Angle Assignment:** Depending on the sender of the message (telescope or roof), the angle value is assigned to the corresponding variable (angle_telescope or angle_roof).

4.1.3 Digital Switching

Transformer is converting the AC supply of observatory to 24Volt DC. There is the input wire (I) and output wire (O).Input and output wire have two more wires red and black in them.In total we have 4 main motor connection wires namely IR(input red wire), IB(input Black Wire), OR(Output Red Wire), OB(Output Black Wire).

Rotation Condition:

- Left Rotation : short (IB,OB) and short(IR,OR)
- Right Rotation : short(IB,OR) and short(IR,OB)

We are doing this switching digitally with the help of Relay.

4.1.4 Motor Rotation

```
double delta = (angle_telescope - angle_roof);


if (millis() - prev_millis > 1500) {
    Serial.println("Delta: " + String(delta));
    if (delta > 5) {
        Serial.println(" Moving left");
    } else if (delta < -5) {
        Serial.println("Moving right ");
    } else {
        Serial.println("Not moving");
    }
    prev_millis = millis();
}
```

Explanation:

- **Delta Calculation:** The code calculates the difference between the angles of the telescope and the roof and stores it in the variable `delta`.

- **Timing Condition:** If the time difference since the last execution of the block is greater than 1500 milliseconds, the block of code is executed.
- **Serial Output:** The value of `delta` is printed to the serial monitor.
- **Relay Control:** Depending on the value of `delta`, the relay is controlled to move motor left, right, or remain stationary.

4.2 Telescope Module

4.2.1 Establishing connection with WiFi Server

```
#include <WiFi.h>
#include <WiFiMulti.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
WiFiMulti WiFiMulti;

void setup()
{
    Serial.begin(115200);
    delay(10);


    WiFiMulti.addAP("yourAP", "yourPassword");

    Serial.println();
    Serial.println();
    Serial.print("Waiting for WiFi... ");

    while(WiFiMulti.run() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    Serial.println("");
    Serial.println("WiFi connected");
```

```
    Serial.println("IP address:");
    Serial.println(WiFi.localIP());

    delay(500);
}
```

Explanation:

- **Library Inclusion:** The code includes the necessary libraries for interfacing with the MPU6050 sensor and connecting to a WiFi network.
- **Global Objects:** An instance of the 'AdafruitMPU6050' class named 'mpu' and a 'WiFiMulti' object are declared globally.
- **Setup Function:** The 'setup()' function initializes the serial communication, adds the WiFi network credentials to the 'WiFiMulti' object, and establishes a WiFi connection.
- **WiFi Connection:** The code waits until a WiFi connection is established, printing dots while waiting.

4.2.2 Rotation Calculation and HTTP GET request

```
while(1){
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    double readings = g.gyro.z;
    readings=(readings>0.1)||(readings<-0.1) ? readings:0;
    readings = readings * 180 / PI;

    delay(dt);
    new_millis = millis();
    angle += (new_millis - curr_millis) * readings / 1e3;

    curr_millis = new_millis;
    if (curr_millis - prev_millis > 500){
        Serial.print(String(angle) + "\n");
        client.print("GET /sender=Telescope/angle="
        +String(angle)+ "/ HTTP/1.1\n\n");
        prev_millis = curr_millis;
    }
```

```
}
delay(3000);
```

Explanation:

- **Loop:** The code enters an infinite loop.
- **Sensor Reading:** Inside the loop, sensor readings for acceleration, gyro, and temperature are obtained using the ‘getEvent()‘ function.
- **Processing Readings:** The gyro reading along the z-axis is extracted and converted to degrees. If the reading is within a small range, it is set to zero.
- **Angle Calculation:** The code calculates the change in angle based on the gyro reading and time elapsed since the last iteration.
- **Data Transmission:** If a certain time interval has passed, the angle value is printed to the serial monitor and sent to a client using an HTTP GET request.
- **Delay:** After each iteration of the loop, there is a delay of 3000 milliseconds.

4.3 Dome Module

4.3.1 Establishing connection with WiFi Server

```
#include <WiFi.h>
#include <WiFiMulti.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
WiFiMulti WiFiMulti;

void setup()
{
    Serial.begin(115200);
    delay(10);
```

```
    WiFiMulti.addAP("yourAP", "yourPassword");

    Serial.println();
    Serial.println();
    Serial.print("Waiting for WiFi... ");

    while(WiFiMulti.run() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    delay(500);
}
```

Explanation:

- **Library Inclusion:** The code includes the necessary libraries for interfacing with the MPU6050 sensor and connecting to a WiFi network.
- **Global Objects:** An instance of the 'AdafruitMPU6050' class named 'mpu' and a 'WiFiMulti' object are declared globally.
- **Setup Function:** The 'setup()' function initializes the serial communication, adds the WiFi network credentials to the 'WiFiMulti' object, and establishes a WiFi connection.
- **WiFi Connection:** The code waits until a WiFi connection is established, printing dots while waiting.

4.3.2 Rotation Calculation and HTTP GET request

```
while(1){
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    double readings = g.gyro.z;
    readings=(readings>0.1)||(readings<-0.1) ? readings:0;
```

```
    readings = readings * 180 / PI;

    delay(dt);
    new_millis = millis();
    angle += (new_millis - curr_millis) * readings / 1e3;

    curr_millis = new_millis;
    if (curr_millis - prev_millis > 500){
        Serial.print(String(angle) + "\n");
        client.print("GET /sender=Telescope/angle="
        +String(angle)+ "/ HTTP/1.1\n\n");
        prev_millis = curr_millis;
    }
}
delay(3000);
```

Explanation:

- **Loop:** The code enters an infinite loop.
- **Sensor Reading:** Inside the loop, sensor readings for acceleration, gyro, and temperature are obtained using the 'getEvent()' function.
- **Angle Calculation:** The code calculates the change in angle based on the gyro reading and time elapsed since the last iteration.
- **Data Transmission:** If a certain time interval has passed, the angle value is printed to the serial monitor and sent to a client using an HTTP GET request.
- **Delay:** After each iteration of the loop, there is a delay of 3000 milliseconds.

4.4 Dome Control via Mobile

4.4.1 GET HTTP Request Page

Listing 1: code for HTTP Request Buttons

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
```

```
  content="width=device-width, initial-scale=1.0">
  <title>HTTP Request Buttons</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    h1 {
      text-align: center;
    }
    button {
      display: block;
      margin: 10px auto;
      padding: 10px 20px;
      font-size: 16px;
      background-color: #007bff;
      color: #fff;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
  </style>
</head>
<body>

<h1>Control Panel</h1>

<button onclick="sendRequest('http://192.168.4.1/
sender=device/angle=1/')">Rotate right</button>
<button onclick="sendRequest('http://192.168.4.1/
sender=device/angle=-1/')">Rotate left</button>
<button onclick="sendRequest('http://192.168.4.1/
sender=device/angle=0/')">Stop rotation</button>

<script>
```

```
function sendRequest(url) {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", url, true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      if (xhr.status === 200) {
        console.log("Request successful!");
        location.reload();
      } else {
        console.log("Request failed!");
        location.reload();

      }
    }
  };
  xhr.send();
}
</script>

</body>
</html>
```

4.4.2 User Interface

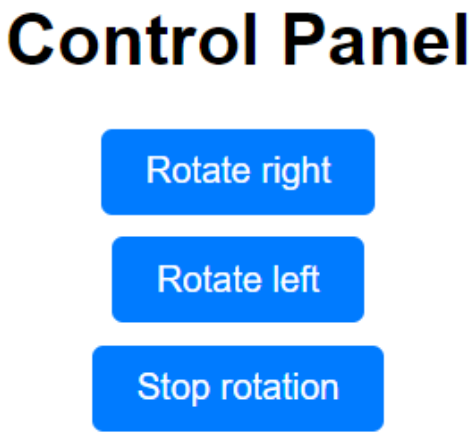

Figure 12: UI GET