Garikapati Sai Harshith – EE22BTECH11022

ELEKTRONIKA

TELESCOPE AUTOMATION PROJECT

Main Objective:

The objective is to ensure that a telescope continuously points towards a given planet, starting from an initial position. As the planet's position changes over time, the telescope's pointing direction must be adjusted accordingly. For example, the telescope's alignment with the planet should be verified and adjusted every second to account for any changes in the planet's position.

Explanation:

To ensure that a telescope rotate and point correcting we can divide this project into three parts 1) Mechanical, 2) Software , 3) Electronics.

Mechanical: A normal telescope used to observe planets is rotated using two circular discs, each driven by a motor at its center. These circular discs are connected to the telescope in such a way that one motor's rotation results in horizontal movement, while the other motor's rotation results in vertical movement.

Software: The code is designed to track the real-time position of a specified planet from a given observer's location on Earth. It calculates

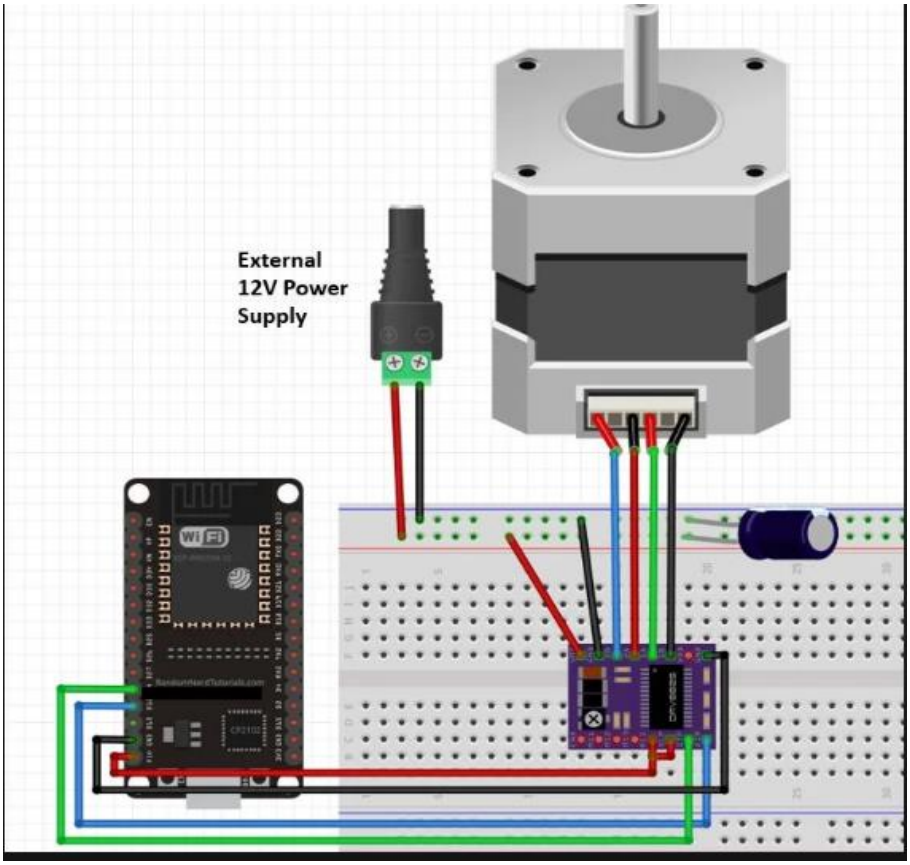
the planet's altitude and azimuth, which are essential for directing a telescope to observe the planet. The code uses the Skyfield library for astronomical computations and the ‘pytz’ library for handling time zones accurately. Inputs for the code for planet name, latitude and longitude position of where the telescope is pointing initially. The observer's location on Earth is defined using the ‘Topos’ class. Using the observer's location and the current time t, the function calculates the astrometric position of the specified planet. It then extracts the apparent altitude and azimuth from this position and returns these values. The core of the real-time tracking functionality is encapsulated within a ‘try-while’ block. This loop continuously updates the planet's position every second. Within the loop, the function ‘get_planet_position’ is called to get the current altitude and azimuth of the planet. The code is

```
lat = 17.38 (input in degrees)
long = 78.48 (input in degrees)
from skyfield.api import Topos, load
from datetime import datetime
import pytz
import time
planet=None
def get_planet_position(planet_name, observer_latitude, observer_longitude, t):
    # Load the ephemeris
    global planet
    planets = load('de421.bsp')
    earth, planet = planets['earth'], planets['planet_name']

    # Set observer's location
    observer = earth + Topos(latitude_degrees=observer_latitude,
longitude_degrees=observer_longitude)

    # Compute position
    astrometric = observer.at(t).observe(_)
    alt, az, _ = astrometric.apparent().altaz()

    # Return altitude and azimuth
    return alt.degrees, az.degrees

# Example observer's location (replace with your actual coordinates)
observer_latitude = lat # Latitude of New Delhi
observer_longitude = long # Longitude of New Delhi

# Example planet to track
planet_name = "Jupiter Barycenter"
```

```
# Get current time in IST
ist = pytz.timezone('Asia/Kolkata')
now = datetime.now(ist)

# Create a Time object using Skyfield's timescale
ts = load.timescale()

try:
    while True:
        # Get current position of the planet
        altitude, azimuth = get_planet_position(planet_name, observer_latitude,
observer_longitude, ts.utc(now))

        # Output the current position
        print(f"Altitude: {altitude} degrees, Azimuth: {azimuth} degrees")

        # Update every 1 second
        time.sleep(1)

except KeyboardInterrupt:
    print("Tracking stopped.")
```

Electrical: We made a soldered board connecting and containing esp32 and motor driver TMC2208(power modulator that integrates between motor and controller) which is connected to the motor.

ESP32 to motor connection:



We have create a sample code to our electrical components are correct.

```
import RPi.GPIO as GPIO
import time


# Define GPIO pins
motor_pin_1 = 17  # IN1 on the motor driver
motor_pin_2 = 18  # IN2 on the motor driver
enable_pin = 27   # ENA on the motor driver


# Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(motor_pin_1, GPIO.OUT)
GPIO.setup(motor_pin_2, GPIO.OUT)
GPIO.setup(enable_pin, GPIO.OUT)
```

```
# Set the PWM frequency and initialize PWM
pwm = GPIO.PWM(enable_pin, 1000)  # 1 kHz PWM frequency
pwm.start(0)  # Start PWM with 0% duty cycle

# Function to rotate the motor
def rotate_motor(degrees, speed_dps):
    duration = degrees / speed_dps  # Calculate the duration for rotation
    pwm.ChangeDutyCycle(100)  # Set duty cycle to 100% to rotate the motor

    # Rotate motor clockwise
    GPIO.output(motor_pin_1, GPIO.HIGH)
    GPIO.output(motor_pin_2, GPIO.LOW)

    time.sleep(duration)  # Rotate for the calculated duration

    # Stop the motor
    GPIO.output(motor_pin_1, GPIO.LOW)
    GPIO.output(motor_pin_2, GPIO.LOW)
    pwm.ChangeDutyCycle(0)  # Stop PWM

try:
    rotate_motor(200, 1)  # Rotate the motor for 200 degrees at 1 degree per second
finally:
    pwm.stop()  # Stop PWM
    GPIO.cleanup()  # Clean up GPIO settings
```

Using this code the motor rotates for 200degrees with the speed of 1degree/sec.

Status:

We have almost finished the hardware. The remaining task is to write the microcontroller code that performs the function described in the software part, calculating the specific rotation angles with respect to the azimuth (north) and altitude (horizon), so that the motor rotates to the required planet.

What if there is more time?

Since we are left with some software parts, we will probably complete it before the sci-tech day. If we had more time, we could optimize the speed of motor rotation and make some improvements.

What I learned from this project:

I was able to interact and work with people from different branches. I have done my first time soldering on a board in this project. I learned the basics of micro python Thonny ide. I did the electronics and a little part of software.