Solar Tracking System Project

Index:

Abstract

The Solar Tracking System is an automated mechanism designed to maximize the efficiency of solar panels by orienting them towards the direction of maximum sunlight. This system utilizes Light Dependent Resistors (LDRs) and a servo motor controlled by an Arduino UNO to dynamically adjust the position of the panels throughout the day. By continuously aligning with the sun, the system ensures optimal energy absorption, thereby enhancing the overall performance of solar power generation. This report provides a detailed overview of the hardware components, the Arduino and Python code used for real-time data visualization, and the system's implementation and results.

Literature Survey

Solar tracking systems have been extensively researched and developed to improve the efficiency of solar panels. Various studies highlight the effectiveness of solar tracking in increasing energy capture compared to stationary panels.

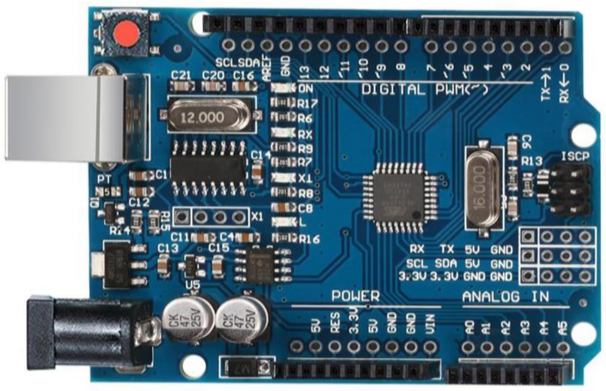
- **Single-Axis vs. Dual-Axis Tracking Systems**: Mousazadeh et al. (2009) demonstrated that single-axis tracking systems could increase solar energy capture by up to 27% compared to fixed systems. In contrast, dual-axis tracking systems, which follow the sun's path both vertically and horizontally, provide even higher efficiency gains, albeit at a higher complexity and cost. Duffie and Beckman (2013) further explored different tracking mechanisms and concluded that dual-axis trackers offer the highest efficiency improvements, making them suitable for high-performance applications.
- **Arduino-Based Solar Trackers**: The use of Arduino microcontrollers in solar tracking has also been well-documented. Arduino-based systems offer a cost-effective and flexible solution for implementing tracking algorithms. Kumar and Sudhakar (2015) successfully implemented a low-cost solar tracker using Arduino, LDRs, and servo motors, demonstrating significant improvements in energy capture. Their study highlights the advantages of using open-source platforms like Arduino for developing accessible and scalable solar tracking solutions.
- **LDRs in Solar Tracking**: Light Dependent Resistors (LDRs) are commonly used in solar tracking systems for their simplicity and effectiveness in detecting light intensity. Research by Patel and Agarwal (2018) showcased a solar tracking system that utilized LDRs to continuously monitor sunlight and adjust the panel's position accordingly, resulting in improved energy efficiency compared to static panels.

Introduction

The Solar Tracking System is an automated setup designed to orient solar panels towards the direction of maximum sunlight using Light Dependent Resistors (LDRs) and a servo motor controlled by an Arduino UNO. The primary objective of this project is to enhance the efficiency of solar panels by ensuring they receive maximum sunlight throughout the day. This report provides a comprehensive overview of the hardware setup, Arduino code, and Python code for real-time data visualization.

Hardware Components

1.Arduino UNO Board



The Arduino UNO is a versatile and user-friendly microcontroller board based on the ATmega328P chip, making it an excellent choice for both beginners and experienced developers. It features 14 digital input/output pins, with 6 of these capable of providing pulse-width modulation (PWM) signals. Additionally, the board includes 6 analog input pins for reading sensor data. With a 32 KB flash memory for storing code (of which 0.5 KB is reserved for the bootloader), 2 KB of SRAM, and 1 KB of EEPROM for persistent data storage, the Arduino UNO is well-equipped for a variety of projects. It operates at a clock speed of 16 MHz and is powered via a standard USB-B connection or an external power jack, accepting a voltage range of 7-12V. The board also includes an ICSP header for programming and additional expansion capabilities. Its ease of use and robust features make it a popular choice for prototyping and learning in the world of electronics.

2. LDRs (Light Dependent Resistors)

Light Dependent Resistors (LDRs), also known as photoresistors, are components whose resistance changes in response to light levels. The resistance of an LDR decreases as the intensity of light falling on it increases, making it useful for a variety of light-sensing applications.

Here’s how they work: LDRs are made from semiconductor materials that exhibit a change in electrical resistance when exposed to light. When light strikes the LDR, photons are absorbed by the semiconductor, which in turn lowers the resistance. This property makes LDRs ideal for use in circuits where light detection is required, such as in automatic lighting systems, light meters, and solar tracking systems.

LDRs are often used in conjunction with a voltage divider circuit, where the LDR is paired with a fixed resistor. As the light level changes, the voltage across the LDR changes, which can then be measured by a microcontroller or analog-to-digital converter to determine the light intensity.

Overall, LDRs are a straightforward and effective way to measure light levels and are widely used in various electronic and electrical projects.

3. Resistors (10k Ohm)

A 10k ohm resistor is a component used in electronic circuits to limit the current flow and divide voltages. The "10k" refers to the resistance value of the resistor, which is 10,000 ohms. Here are some key points about 10k ohm resistors:

1. **Function**: They are commonly used in voltage divider circuits, pull-up or pull-down configurations, and to protect other components by limiting the current flowing through them.
2. **Voltage Divider**: In a voltage divider setup, a 10k ohm resistor can be used in combination with another resistor to produce a desired voltage drop, which is useful for adjusting signal levels or providing a reference voltage.

3. **Pull-up/Pull-down**: In digital circuits, 10k ohm resistors are often used as pull-up or pull-down resistors to ensure a known state for a digital input pin when no active signal is present.
4. **Power Rating**: The power rating of a resistor indicates how much power it can safely dissipate without damage. Common power ratings for resistors include 1/8 watt, 1/4 watt, and 1/2 watt. A 10k ohm resistor is typically available in these ratings, with 1/4 watt being a common choice.
5. **Tolerance**: Resistors come with various tolerance levels, indicating how close the actual resistance is to the specified value. Common tolerances are ±1%, ±5%, and ±10%.

10k ohm resistors are versatile and used in many applications due to their standard resistance value and ease of integration into various circuit designs.

4. Servo Motor (SG90)

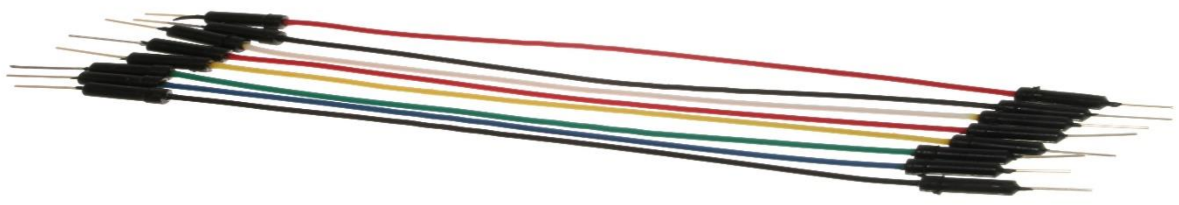
The TowerPro SG90 is a popular and compact servo motor known for its affordability and versatility, making it a common choice for hobbyists and small-scale robotics projects. Here are some key features and details about the SG90:

1. **Size and Weight**: The SG90 is a micro servo, meaning it's small and lightweight, which is ideal for applications where space is limited.
2. **Torque**: It provides a modest amount of torque, typically around 1.8 kg-cm (kilogram-centimeter) at 4.8V. This torque is sufficient for light-duty tasks such as steering mechanisms in small robots or controlling small flaps.
3. **Speed**: The SG90 can rotate to its full range of motion in approximately 0.1 to 0.2 seconds, depending on the applied voltage and load. This relatively fast response time makes it suitable for applications requiring quick movements.

4. **Range of Motion**: It generally has a range of approximately 180 degrees. However, the exact range can vary slightly based on the specific model and manufacturer's specifications.
5. **Operating Voltage**: The SG90 typically operates at a voltage range of 4.8V to 6V. It is designed to work with standard 5V systems, which is common in many hobbyist electronics projects.
6. **Control Signal**: It is controlled via pulse-width modulation (PWM) signals, where the width of the pulse determines the position of the servo. Typically, a pulse width of 1 ms to 2 ms corresponds to the full range of motion.
7. **Applications**: The SG90 is commonly used in various projects including robotic arms, small robots, model airplanes, and other applications where precise control of movement is required.

Overall, the TowerPro SG90 is valued for its small size, cost-effectiveness, and ease of use, making it a popular choice for hobbyists and beginners in robotics and electronics.

5.Connecting Wires



Connecting wires are essential components used to link various parts of an electronic circuit. They come in different forms, including jumper wires, which are pre-made with connectors for quick connections, and hook-up wires, which are flexible and used for general wiring. These wires ensure reliable electrical connections between components, facilitating the construction and testing of electronic circuits on breadboards or in more permanent setups.

Methodology

The Solar Tracking System project follows a structured methodology to design, implement, and evaluate the performance of the tracking system. The key steps involved are as follows:

1. **System Design and Component Selection**:
   - **Hardware Components**: The primary components include an Arduino UNO board, LDRs, a servo motor, and connecting wires. The Arduino UNO serves as the central control unit, reading inputs from the LDRs and controlling the servo motor.

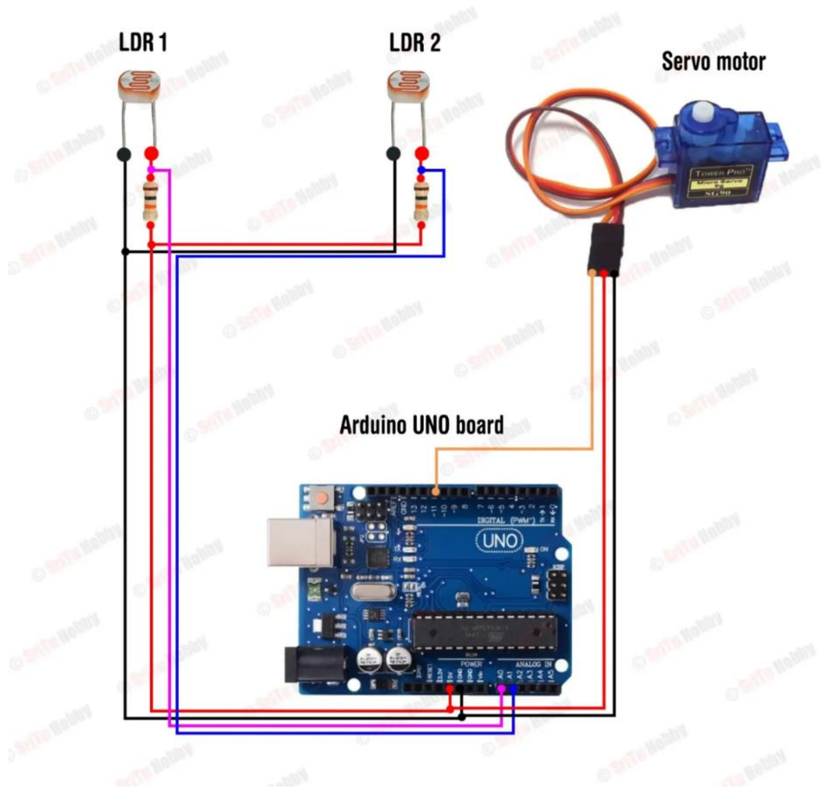
- **Circuit Diagram and Connections**: The LDRs are connected to the analog input pins of the Arduino, while the servo motor is connected to a digital PWM pin. The circuit is designed to ensure accurate readings from the LDRs and precise control of the servo motor.

2. **Arduino Code Development**:
   - **Code Overview**: The Arduino code reads the values from the LDRs and adjusts the servo motor's position to align the solar panel with the direction of maximum sunlight. The code also sends the LDR values and the servo angle to the serial port for real-time monitoring.
   - **Libraries and Constants**: The Servo.h library is used to control the servo motor. Constants are defined for the LDR pins and the servo pin.
   - **Setup Function**: Initializes serial communication at 9600 baud rate and attaches the servo motor to pin 9, setting its initial position to 90 degrees.
   - **Loop Function**: Continuously reads values from the LDRs, calculates the difference between the readings, maps the difference to a servo angle in the range of 0 to 180 degrees, constrains the servo angle within the valid range, moves the servo motor to the calculated position, and sends the LDR values and the servo angle to the serial port.
3. **Python Code Development for Real-Time Data Visualization**:
   - **Establish Serial Connection**: Sets up a serial connection on the specified port at 9600 baud rate.
   - **Initialize Lists**: Creates lists to store LDR readings, servo angles, and timestamps.
   - **Setup Plotting**: Uses matplotlib to create a figure with two subplots for LDR readings and servo angle. Sets titles, labels, and grids for the plots and sets up a text display to show the latest readings.
   - **Main Loop**: Continuously reads data from the serial port, parses the data to extract LDR readings and servo angle, appends the data to the respective lists, updates the plots with the new data, adjusts the x-axis limits to show the last 10 seconds of data, draws the updated plots, and handles keyboard interrupt to safely close the serial connection.
4. **System Testing and Calibration**:
   - **Initial Testing**: The system is initially tested to ensure all components are functioning correctly. The LDRs' readings and the servo motor's movements are monitored to confirm they respond appropriately to changes in light intensity.
   - **Calibration**: The LDRs and servo motor are calibrated to ensure accurate alignment with the sun. This involves adjusting the scaling factor in the Arduino code to map the LDR readings to the correct servo angles.

- **Data Collection**: The system is deployed in a controlled environment to collect data on its performance. LDR readings, servo angles, and timestamps are recorded and analyzed to evaluate the system's accuracy and efficiency.

5. **Performance Evaluation**:
   - **Data Analysis**: The collected data is analyzed to determine the system's ability to track the sun accurately and optimize energy absorption. The performance is compared to a stationary solar panel to quantify the efficiency gains.
   - **Visualization**: The Python code is used to visualize the real-time data, providing insights into the system's behavior and performance over time.
6. **Conclusion and Future Work**:
   - **Summary of Results**: The results of the performance evaluation are summarized, highlighting the efficiency improvements achieved by the solar tracking system.
   - **Recommendations**: Suggestions for future improvements and enhancements to the system are provided. These may include adding more sensors, improving the mapping algorithm, integrating with a more robust data visualization and logging system, and exploring alternative tracking mechanisms.

Circuit Diagram



Circuit Connections

1. LDR 1:
   - One end connected to 5V.
   - The other end connected to a 10k ohm resistor, which is then connected to GND.
   - The junction between LDR and resistor is connected to analog pin A0 of the Arduino.

2. LDR 2:
   - One end connected to 5V.
   - The other end connected to a 10k ohm resistor, which is then connected to GND.
   - The junction between LDR and resistor is connected to analog pin A1 of the Arduino.

3. Servo Motor:
   - Signal pin connected to digital pin 9 of the Arduino.
   - VCC connected to 5V.
   - GND connected to GND.

Arduino Code

The Arduino code reads the values from the two LDRs and adjusts the servo motor's position to align with the direction of maximum sunlight. Additionally, it sends the LDR values and the servo angle to the serial port for real-time monitoring.

```
#include <Servo.h>


const int LDR1_PIN = A0;   // LDR 1 connected to analog pin A0
const int LDR2_PIN = A1;   // LDR 2 connected to analog pin A1
const int SERVO_PIN = 9;   // Servo connected to digital pin 9


Servo myServo;
void setup() {
  Serial.begin(9600);       // Start serial communication for debugging
  myServo.attach(SERVO_PIN); // Attach servo motor to pin 9
  myServo.write(90);       // Start servo at 90 degrees (mid-point)
}
void loop() {
  int ldr1Value = analogRead(LDR1_PIN);  // Read LDR 1 value
  int ldr2Value = analogRead(LDR2_PIN);  // Read LDR 2 value

  // Calculate the difference between LDR readings
  float difference = float(ldr2Value - ldr1Value);

  // Map the difference to servo angle range (0 to 180 degrees)
  int servoAngle = 90 + int(difference * 0.1); // Adjust the scaling factor as needed

  // Constrain the servo angle to 0 - 180 degrees
  servoAngle = constrain(servoAngle, 0, 180);

  // Move the servo to the calculated position
  myServo.write(servoAngle);

  // Send LDR values and servo angle over serial
  Serial.print("LDR1:");
  Serial.print(ldr1Value);
  Serial.print(",LDR2:");
  Serial.print(ldr2Value);
  Serial.print(",Angle:");
  Serial.println(servoAngle);

  // Add a delay between servo movements
  delay(1000); // Adjust delay time as needed
}
```

Arduino Code Explanation

Libraries and Constants:

- `Servo.h`: Library to control the servo motor.
- `LDR1_PIN`, `LDR2_PIN`, `SERVO_PIN`: Define the pins connected to the LDRs and the servo motor.

Setup Function:

- Initializes serial communication at 9600 baud rate.
- Attaches the servo motor to pin 9 and sets its initial position to 90 degrees.

Loop Function:

- Reads values from the two LDRs.
- Calculates the difference between the LDR readings.
- Maps the difference to a servo angle in the range of 0 to 180 degrees.
- Constrains the servo angle within the valid range.
- Moves the servo motor to the calculated position.
- Sends the LDR values and the servo angle to the serial port.
- Delays the loop for 1 second.

Python Code

The Python code establishes a serial connection with the Arduino, reads the real-time data, and plots the LDR readings and servo angle using matplotlib.

```
import serial

import time

import matplotlib.pyplot as plt


# Establish serial connection (change the port and baud rate as needed)

ser = serial.Serial('COM3', 9600)  # Replace 'COM3' with your Arduino's serial port
```

```
# Initialize lists to store data
ldr1_values = []
ldr2_values = []
servo_angles = []
timestamps = []

# Setup for plotting and text display
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Plot setup for LDR readings
ax1.set_title('LDR Readings')
ax1.set_xlabel('Time')
ax1.set_ylabel('Value')
ax1.grid(True)

# Plot setup for Servo angle
ax2.set_title('Servo Angle')
ax2.set_xlabel('Time')
ax2.set_ylabel('Angle')
ax2.grid(True)

# Display text setup
text_disp = ax1.text(0.02, 0.95, '', transform=ax1.transAxes, verticalalignment='top')

# Main loop to read and plot data
try:
    while True:
        if ser.in_waiting > 0:
            # Read data from Arduino
            data = ser.readline().decode().strip()
    # Parse the data
    if data.startswith('LDR1:'):
        parts = data.split(',')
        ldr1_value = int(parts[0].split(':')[1])
        ldr2_value = int(parts[1].split(':')[1])
        angle_value = int(parts[2].split(':')[1])
        timestamp = time.time()  # Timestamp for x-axis

        # Append data to lists
        ldr1_values.append(ldr1_value)
        ldr2_values.append(ldr2_value)
        servo_angles.append(angle_value)
        timestamps.append(timestamp)

        # Update plot
        ax1.plot(timestamps, ldr1_values, label='LDR1', color='b')
        ax1.plot(timestamps, ldr2_values, label='LDR2', color='g')
        ax2.plot(timestamps, servo_angles, label='Servo Angle', color='r')

        # Update text display
        text_disp.set_text(f"LDR1: {ldr1_value}\nLDR2: {ldr2_value}\nAngle: 
{angle_value}")

        # Adjust plot limits for better visualization (optional)
        ax1.set_xlim(max(0, timestamp - 10), timestamp + 1)  # Adjust x-axis limit to show 
last 10 seconds
        ax2.set_xlim(max(0, timestamp - 10), timestamp + 1)

        # Draw the plot
        fig.tight_layout()
```

```
            plt.pause(0.01)

except KeyboardInterrupt:
    print("Program interrupted. Exiting...")

finally:
    ser.close()  # Close serial port
    print("Serial connection closed.")
```

Python Code Explanation

Import Libraries:

- `serial`: For serial communication with the Arduino.
- `time`: For timestamps and delays.
- `matplotlib.pyplot`: For plotting the data.

Establish Serial Connection:

- Sets up a serial connection on port `COM3` at 9600 baud rate (adjust as needed).

Initialize Lists:

- `ldr1_values`, `ldr2_values`, `servo_angles`, `timestamps`: Lists to store LDR readings, servo angles, and timestamps.

Setup Plotting:

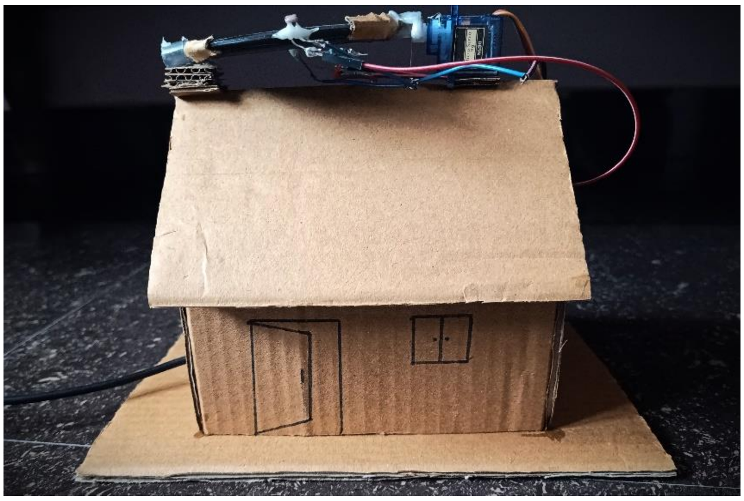
- Creates a figure with two subplots for LDR readings and servo angle.
- Sets titles, labels, and grids for the plots.
- Sets up a text display to show the latest readings.

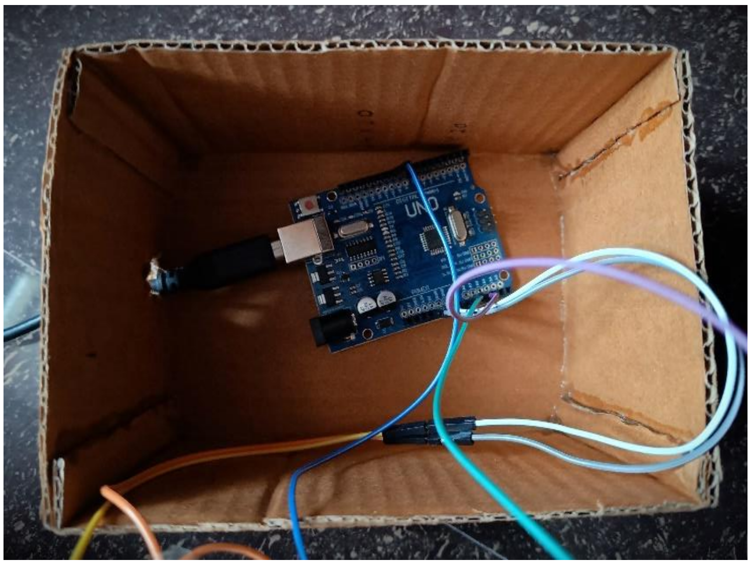
Main Loop:

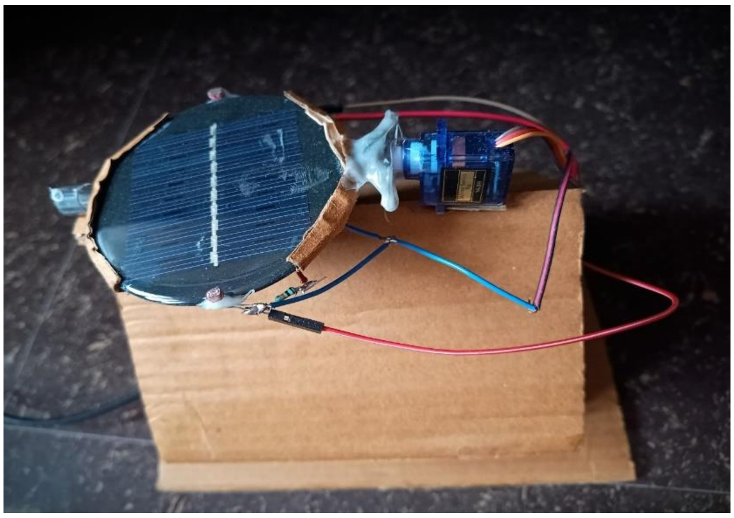
- Continuously reads data from the serial port.
- Parses the data to extract LDR readings and servo angle.

- Appends the data to the respective lists.
- Updates the plots with the new data.
- Adjusts the x-axis limits to show the last 10 seconds of data (optional).
- Draws the updated plots.
- Handles keyboard interrupt to safely close the serial connection.

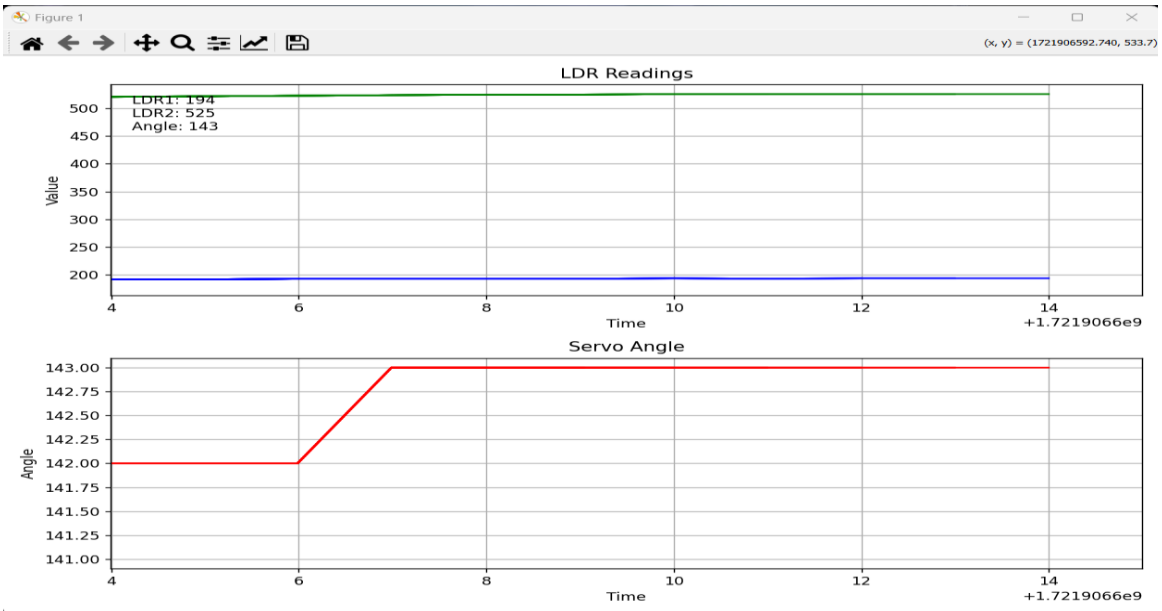
Working Model Images:

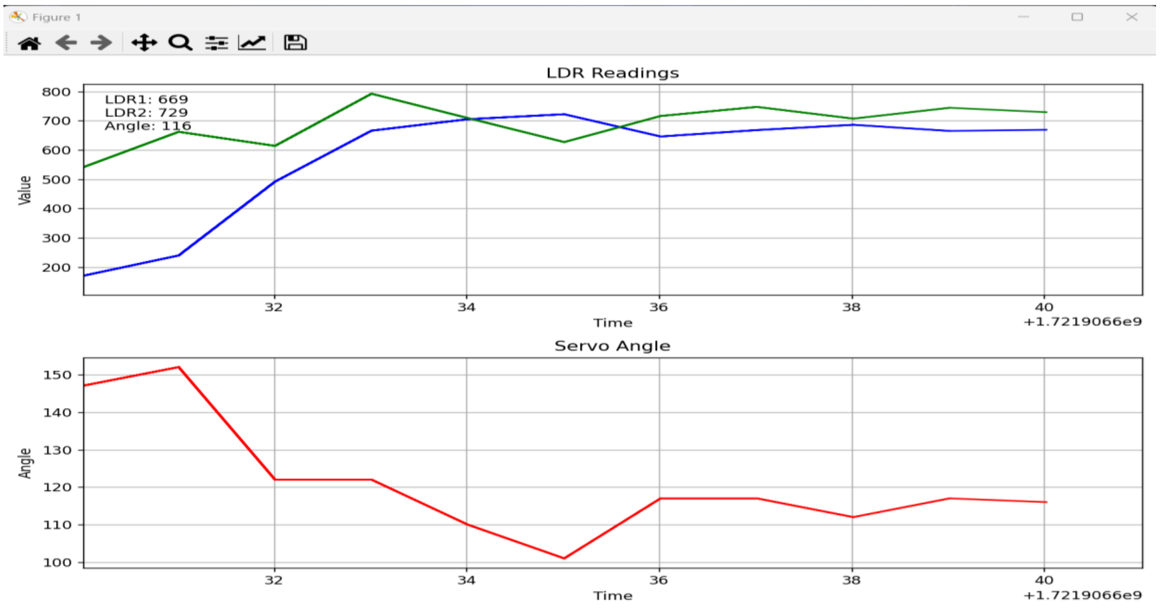


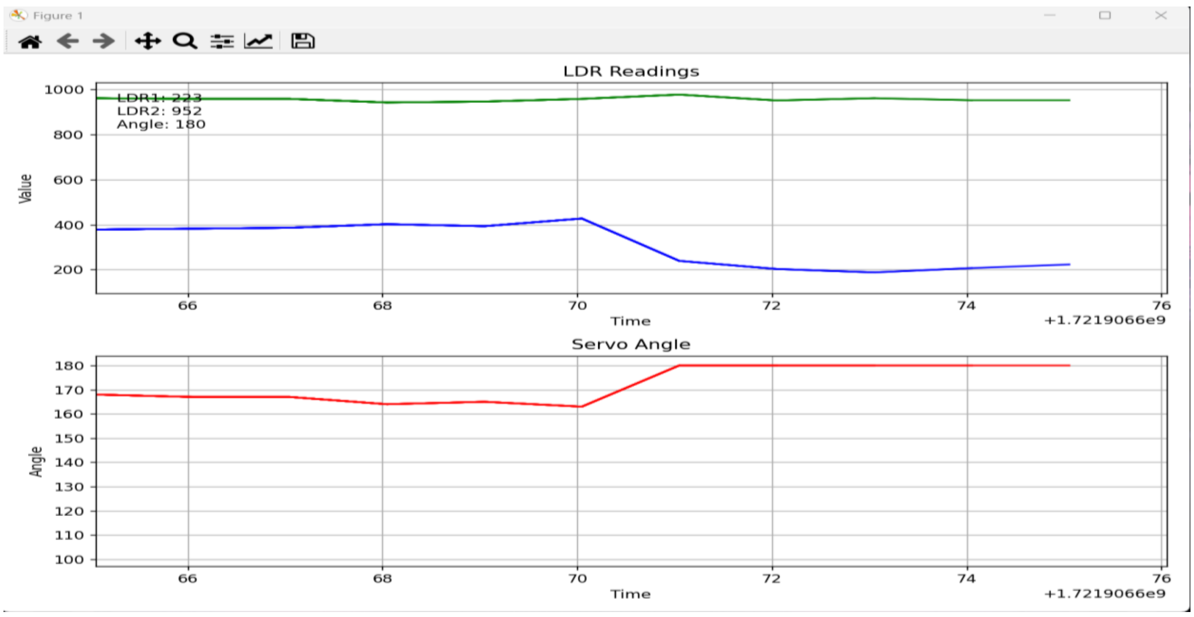




Screen-shots Of Graph:







Conclusion

This Solar Tracking System project successfully demonstrates the use of an Arduino UNO, LDRs, and a servo motor to create an automated system that tracks the sun's position. The real-time data visualization using Python provides valuable insights into the system's performance, making it easier to monitor and adjust as needed. This project can be further enhanced by adding more sensors, improving the mapping algorithm, or integrating with a more robust data visualization and logging system.

References:

- **Website:** https://srituhobby.com/how-to-make-a-solar-tracking-system-using-arduino-step-by-step/
- **OpenAI:** https://chatgpt.com/