University of Tripoli

Faculty of Engineering

Department of Electrical and Electronic Engineering

EE491 Computer Application & design Lab (Mini Project)

Dual-Mode Smart Solar Tracker with Matlab response Analysis

Name : Alaa Elfallah

ID : 2200208066

Instructor : dr.Ali Elmelhi

Spring 2025

1. Abstract

This project introduces a smart solar tracking system designed to maximize energy efficiency by automatically aligning solar panels with the sun. Using two Light Dependent Resistors (LDRs) for light detection and a PID controller for precision, the system adjusts panel position via a servo motor, ensuring optimal orientation through real-time feedback. It supports both automatic and manual modes, with performance comparisons between PID-controlled and non-PID operation. Real-time system response was analyzed and visualized using MATLAB.

2. Introduction

As the demand for renewable energy continues to rise, improving the efficiency of solar power systems has become a critical goal.

One major factor affecting solar panel performance is its orientation relative to the sun. Fixed panels often fail to capture the maximum available solar energy throughout the day. To address this issue, this project proposes a smart solar tracking system that dynamically adjusts the panel's direction to always face the most intense sunlight.

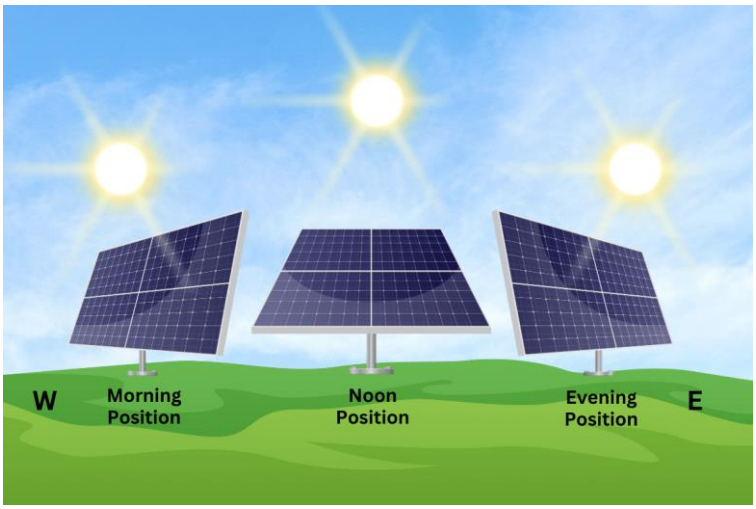


Figure 1

The system relies on two Light Dependent Resistors (LDRs) to sense the intensity of sunlight on either side of the solar panel. The difference between the two LDR readings provides input to a PID (Proportional-

Integral-Derivative) controller, which calculates how the servo motor should adjust the panel's angle. The aim is to reduce the error between the two LDRs to nearly zero — ensuring the panel stays directly aligned with the sun.

This closed-loop feedback mechanism continuously updates the panel's position in real time. When the panel moves, the LDRs shift their orientation, leading to a new set of readings and a new error signal. This creates a continuous tracking cycle that adapts dynamically throughout the day.

To evaluate and compare performance, the system response was visualized using MATLAB in both cases — with and without the PID controller. The analysis clearly demonstrated that the PID controller significantly reduced oscillations and stabilized the panel's movement, proving its crucial role in enhancing system performance.

Additional features of the system include:

- Dual-mode operation:Automatic tracking using PID control, and manual control via an IR remote.
- User feedback system:LED indicators show the active mode; a buzzer gives audible alerts on mode switching or startup.
- Real-time monitoring:An OLED display shows live system data such as tracking mode, LDR readings, and servo angle.

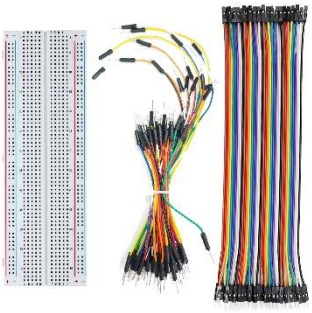
This smart tracking system demonstrates how fundamental control theory and affordable electronics can be combined to build an efficient, low-cost solution for maximizing solar energy collection.

3. Objectives

The main goal of this project is to design and implement an efficient solar tracking system that maximizes solar energy collection by automatically adjusting the orientation of a solar panel using real-time feedback.

4. Equipment

component	*figure*	*functionality*
Arduino UNO		The central microcontroller used to process sensor readings, run control logic (PID), and drive outputs like the servo and display.
Light Dependent Resistors (LDRs)		sensors used to detect light intensity from both sides of the solar panel. Two LDRs provide the basis for feedback control.
Servo Motor (SG90)		A small DC servo motor used to rotate the solar panel in the direction of maximum light based on control output.
OLED Display (128x64 SH1106):		Displays real-time system data such as LDR readings, current panel angle, and operating mode.
IR Receiver Module:		Used in manual mode to receive directional commands from a standard IR remote control**.**

IR Remote Control		Allows the user to manually adjust the panel's direction by sending left/right commands to the IR receiver.
Push Button:		Toggles between automatic and manual operation modes when pressed.
LEDs (Green and Yellow):		Green LED indicates automatic mode is active. Yellow LED indicates manual mode is active.
Buzzer:		Emits a short beep on system startup or when switching modes, providing audio feedback.
Breadboard and Jumper Wires:		Used for prototyping the circuit and connecting all components without soldering.

5. System Overview and Working Principle

This project implements an automatic solar tracking system based on the principle of closed-loop feedback control. The system continuously monitors sunlight intensity using two Light Dependent Resistors (LDRs) positioned on opposite sides of the panel.

The difference in light intensity between the two sensors indicates the direction in which the sun has moved.

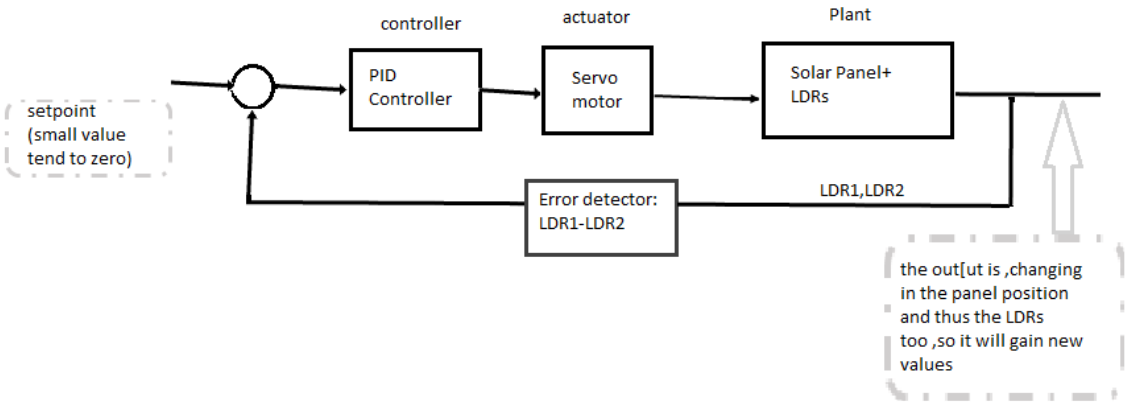


Figure 2:Block diagram of the solar tracking system with feedback

4.1. **How It Works**:

4.1.1. Light Detection:

Two LDRs sense the intensity of sunlight on each side of the panel.

These analog readings are converted into percentage values (0–100%) using the Arduino's ADC.

4.1.2. Error Calculation (Feedback):

The difference between LDR readings (LDR1 - LDR2) determines the error signal.

This error reflects how far off the panel is from facing the sun directly.

4.1.3. PID Controller (in automatic mode):

The error is fed into a PID algorithm to calculate the proper adjustment.

The controller computes a new servo angle to minimize the error by rotating the panel.

In the non-PID version, a basic fixed-step adjustment is used instead (less smooth and less stable).

4.1.4. Actuation (Servo Motor):

The servo motor receives the corrected angle and adjusts the panel's direction accordingly.

This change affects the LDR readings again — closing the loop.

4.1.5. Modes of Operation:

- **Automatic Mode:** PID continuously updates the panel's angle based on LDR feedback.
- **Manual Mode:** The user controls the panel's position using an IR remote, overriding the PID.

4.1.6. User Feedback System:

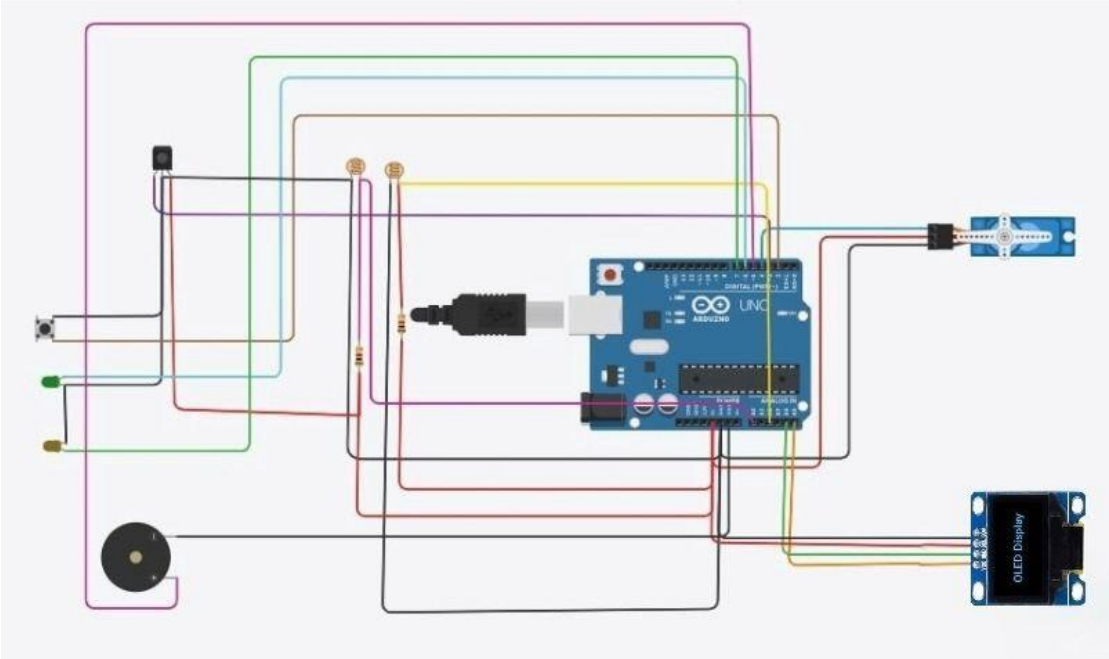
- **OLED Display:** Shows real-time data such as LDR readings, mode, and panel angle.
- **Green & Yellow LEDs:** Indicate the active mode (auto/manual).
- **Buzzer:** Beeps to confirm mode switching or system startup.

4.1.7. Serial Communication:

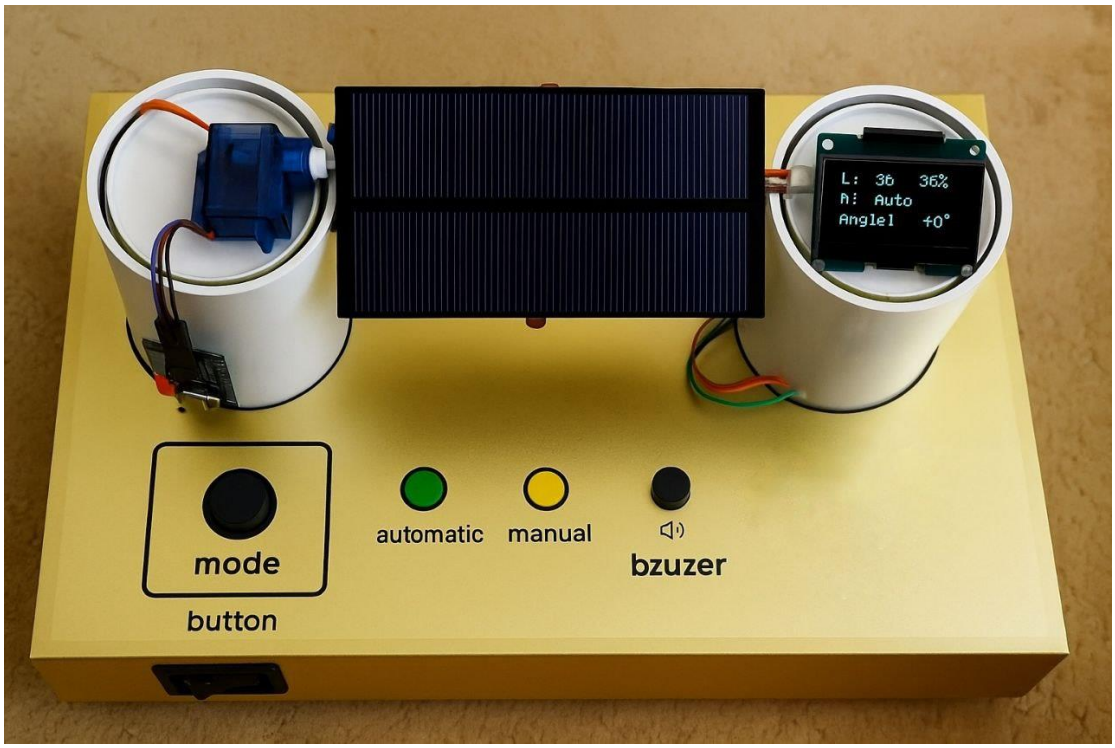
The Arduino sends data to the serial monitor which is the then will be used in Matlab for performance analysis.

(Used for comparing PID vs Non-PID behavior).

4.1.8. The circuit diagram:



4.1.9. The system model:



6. Software Implementation

The solar tracking system integrates both hardware and software components to achieve dynamic, real-time sun tracking with and without a PID controller. The implementation relies on the Arduino IDE for embedded control and MATLAB for real-time data visualization and performance analysis.

6.1.Software Structure

- **Arduino IDE (C++ Implementation):**The Arduino is programmed using the Arduino IDE to perform all embedded control operations. Its responsibilities include:
  - Reading real-time data from the two LDR sensors.
  - Executing the control algorithm (either with or without a PID controller).
  - Controlling the servo motor to adjust the panel's position.
  - Sending continuous data — including LDR readings, servo angle, and operation mode — to a connected computer via serial communication.
- **MATLAB (Visualization and Analysis Tool): A** custom MATLAB Live script is used to receive and process serial data from the Arduino in real time (live). It performs the following tasks:
  - Visualizing system variables : light intensity (LDR values), servo angle, and control mode in real-time.
  - Enabling direct comparison between PID and non-PID system responses.
  - Assisting in performance evaluation through plotted outputs that reveal stability, oscillation, and responsiveness.

6.2.System Execution Workflow

To test and visualize both modes:

- **PID-Controlled Mode :** Upload the PID version of the Arduino code .

Run the MATLAB Live script to observe system response.

The panel responds smoothly and accurately to light changes.

- **Non-PID Mode:** Re-upload the non-PID version of the Arduino code to the same board.

  MATLAB Live script remains the same (no need to modify it).

  The system response becomes more oscillatory and slower, as seen in the plots.

Note: The only change required between experiments is the code uploaded to the Arduino. The MATLAB visualization tool is compatible with both versions.

See Section **10: Appendix** for complete source code listings.

7.Results and Comparison (with MATLAB Analysis)

The solar tracking system was evaluated in two different configurations to assess and validate its performance:

- With PID Controller
- Without PID Controller

To visualize and compare the system's dynamic behavior, real-time data were recorded and plotted using MATLAB. This included the servo motor angle (which is the primary output influenced by the PID controller) along with the LDR sensor readings and the system's operating mode (automatic/manual).

Each MATLAB output consisted of multiple synchronized subplots displaying:

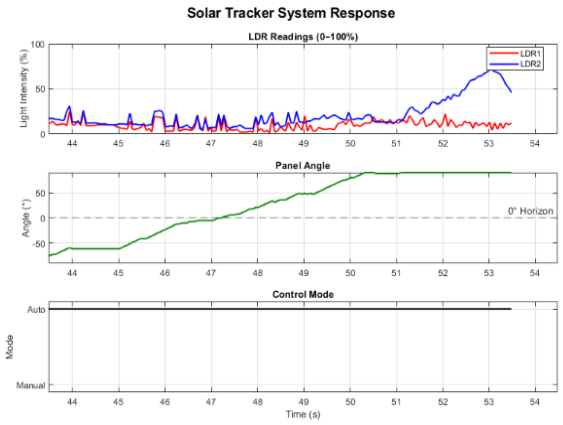
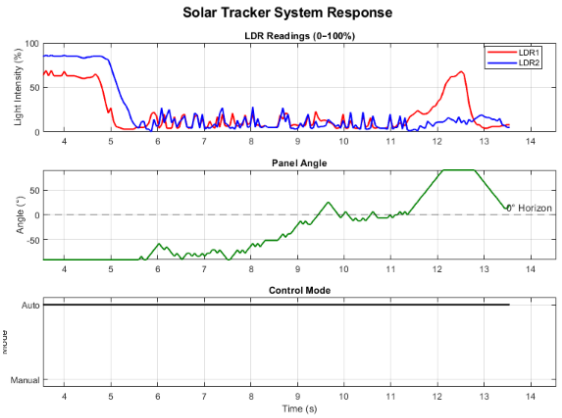
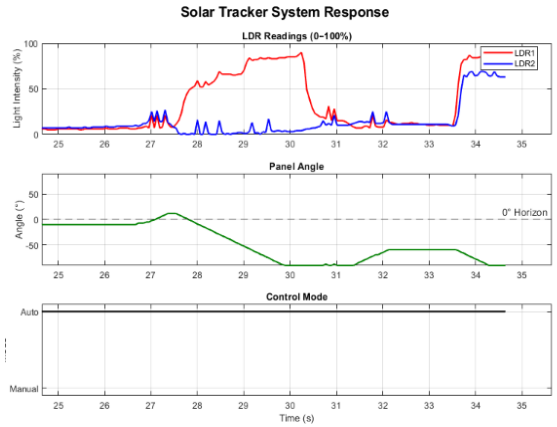
• **LDR1 and LDR2 readings :** light intensity sensed on each side of the panel (may show fluctuations due to environmental factors and sensor position).

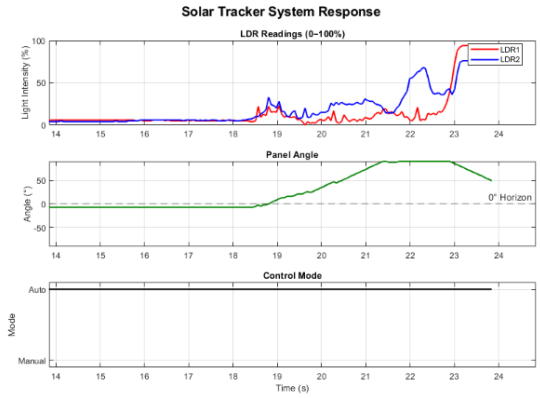
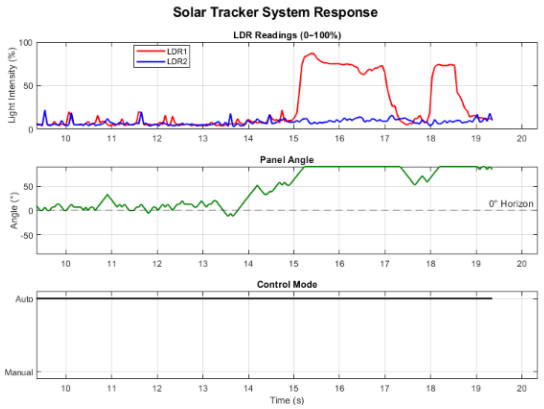
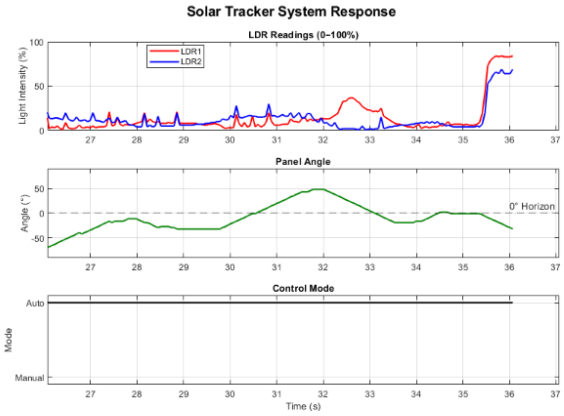
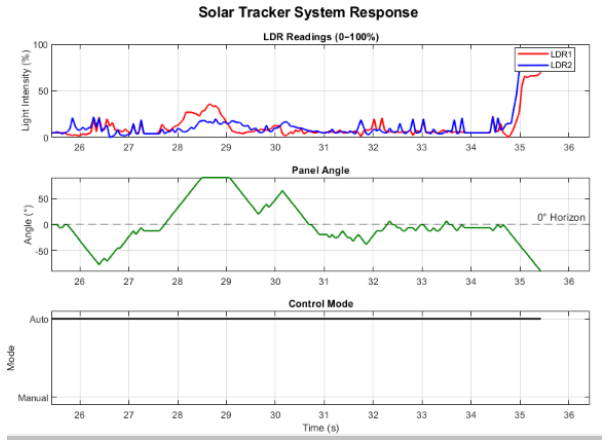
• **Servo motor angle :** the panel’s position over time, which directly reflects the control system’s performance and stability.

- **Mode of operation :** whether the system was in automatic (PID or non-PID) or manual control.

This visualization helped assess how effectively the PID controller minimized oscillations and improved response smoothness compared to the non-PID configuration.

The following snapshots are taken from several test scenarios under both configurations. These figures aim to highlight the contrast in performance between the system with and without PID control and demonstrate the effectiveness of feedback-based control.

	With PID	Without PID
moving from -90 to +90		
moving from 0 to -90		

	With PID	Without PID
moving from 0 to +90		
Random case		

From the figures in the Table above, it is clearly observed that the PID-controlled system exhibits significantly reduced oscillations, resulting in smoother and more stable panel movements. Additionally, the panel responds more quickly and aligns more accurately with the sunlight direction compared to the non-PID configuration.

These results confirm that implementing a PID controller substantially enhances the system's performance, especially in terms of responsiveness, stability, and precise solar alignment.

8. Discussion

This project successfully demonstrated a closed-loop solar tracking system using PID control to enhance panel alignment with the sun. The system reduced oscillations and improved tracking accuracy, as shown in the MATLAB plots.

Real-time data visualization in MATLAB provided valuable insights into the panel's response and confirmed the PID controller’s effectiveness. The system’s interactive features(mode switching, display, buzzer, and LEDs) made it user-friendly and well-integrated.

Overall, the project combined control theory and embedded systems to build a reliable, responsive solar tracker.

9. Conclusion

This project presented a practical and efficient solar tracking system that utilizes a PID-based feedback control mechanism to optimize the orientation of a solar panel. By continuously monitoring light intensity using two LDR sensors, the system effectively adjusted the panel’s position to maintain optimal sunlight exposure.

The comparison between the PID-controlled and non-PID-controlled versions clearly demonstrated the PID controller's significant role in reducing oscillations and improving tracking stability. Real-time performance visualization in MATLAB further validated the system’s behavior and control accuracy.

The successful integration of hardware components, control algorithms, and user interaction features highlights the effectiveness of combining embedded systems with control engineering principles to develop intelligent and energy-efficient solutions.

10. Appendix:

This section contains the complete code implementations for all software modules used in the system. Each version is labeled and separated for clarity.

Appendix A: Arduino Code (With PID Controller):

```
#include <Wire.h>
#include <U8x8lib.h>
#include <Servo.h>
#include <IRremote.h>

U8X8_SSD1306_128X64_NONAME_HW_I2C oled(/* reset=*/ U8X8_PIN_NONE);

Servo solarServo;

// === Pin Definitions ===
#define SERVO_PIN     4
#define LDR1_PIN      A0
#define LDR2_PIN      A2
#define BUTTON_PIN    2
#define GREEN_LED     6
#define YELLOW_LED    7
#define IR_PIN        3
#define BUZZER_PIN    5

#define IR_LEFT   0xF40B0707
#define IR_RIGHT  0xF8070707

const int PHYS_MIN = 25;
const int PHYS_MAX = 165;
const int PHYS_MID = 95;
const int DISP_MIN = -90;
const int DISP_MAX = 90;

//const float Kp = 1.2;
//const float Ki = 0.08;
//const float Kd = 0.5;

const float Kp = 10;
const float Ki = 0.08;
const float Kd = 0.5;


int currentPulse = PHYS_MID;
bool autoMode = true;
bool lastButtonState = HIGH;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 200;
float error = 0, lastError = 0, integral = 0, derivative = 0;
unsigned long lastTime = 0;

```

```
46  void setup() {
47    Serial.begin(115200);
48    Wire.begin();
49    oled.begin();
50    oled.setFont(u8x8_font_chroma48medium8_r);
51    oled.clearDisplay();
52    oled.drawString(0, 0, " Solar Tracker");
53
54    pinMode(BUTTON_PIN, INPUT_PULLUP);
55    pinMode(GREEN_LED, OUTPUT);
56    pinMode(YELLOW_LED, OUTPUT);
57    pinMode(BUZZER_PIN, OUTPUT);
58    digitalWrite(BUZZER_PIN, LOW);
59    digitalWrite(GREEN_LED, HIGH);
60    digitalWrite(YELLOW_LED, LOW);
61
62    IrReceiver.begin(IR_PIN, ENABLE_LED_FEEDBACK);
63    solarServo.attach(SERVO_PIN);
64    solarServo.write(PHYS_MID);
65
66    playStartupBeep();
67    delay(1000);
68  }
69
70  void loop() {
71    static unsigned long lastSendTime = 0;
72    const unsigned long sendInterval = 20;
73
74    handleModeToggle();
75    digitalWrite(GREEN_LED, autoMode);
76    digitalWrite(YELLOW_LED, !autoMode);
77
78    int ldr1 = readLDRPercent(LDR1_PIN);
79    int ldr2 = readLDRPercent(LDR2_PIN);
80    float pidOutput = 0;
81    float currentAngle = pulseToAngle(currentPulse);
82
83    if (autoMode) {
84      unsigned long now = millis();
85      float dt = (now - lastTime) / 1000.0;
86      lastTime = now;
87
88      error = ldr1 - ldr2;
89      integral += error * dt;
90      integral = constrain(integral, -100, 100);
91      derivative = (error - lastError) / dt;
92      pidOutput = Kp * error + Ki * integral + Kd * derivative;
93      lastError = error;
94
95      if (abs(error) > 3) {
96        currentPulse += constrain(pidOutput, -2, 2);
97        currentPulse = constrain(currentPulse, PHYS_MIN, PHYS_MAX);
98        solarServo.write(currentPulse);
99      }
100   } else {
101     if (IrReceiver.decode()) {
102       uint32_t code = IrReceiver.decodedIRData.decodedRawData;
```

```
      if (code == IR_LEFT) currentPulse -= 5;
      else if (code == IR_RIGHT) currentPulse += 5;
      currentPulse = constrain(currentPulse, PHYS_MIN, PHYS_MAX);
      solarServo.write(currentPulse);
      IrReceiver.resume();
    }
  }

  updateOLED(ldr1, ldr2, currentAngle);

  if (millis() - lastSendTime >= sendInterval) {
    Serial.print(millis()); Serial.print(",");
    Serial.print(ldr1); Serial.print(",");
    Serial.print(ldr2); Serial.print(",");
    Serial.print(currentAngle, 1); Serial.print(",");
    Serial.print(pidOutput, 2); Serial.print(",");
    Serial.println(autoMode ? '1' : '0');
    lastSendTime = millis();
  }
}

void handleModeToggle() {
  bool reading = digitalRead(BUTTON_PIN);
  if (reading == LOW && lastButtonState == HIGH &&
      (millis() - lastDebounceTime > debounceDelay)) {
    autoMode = !autoMode;
    lastDebounceTime = millis();
    playToggleBeep();
  }
  lastButtonState = reading;
}

void updateOLED(int ldr1, int ldr2, float angle) {
  oled.clearLine(1); oled.setCursor(0, 1);
  oled.print(" L:"); oled.print(ldr1); oled.print("  R:"); oled.print(ldr2);

  oled.clearLine(2); oled.setCursor(0, 2);
  oled.print(" Mode: "); oled.print(autoMode ? "Auto" : "Manual");

  oled.clearLine(3); oled.setCursor(0, 3);
  oled.print(" Angle: ");
  if (angle >= 0) oled.print("+");
  oled.print(angle, 0);
  oled.print((char)176); // degree symbol
}

int readLDRPercent(int pin) {
  int raw = analogRead(pin);
  return constrain(map(raw, 50, 950, 0, 100), 0, 100);
}

float pulseToAngle(int pulse) {
  return map(pulse, PHYS_MIN, PHYS_MAX, DISP_MAX, DISP_MIN);
}

void playStartupBeep() {
  digitalWrite(BUZZER_PIN, HIGH); delay(150);
  digitalWrite(BUZZER_PIN, LOW);  delay(100);
```

```
161     digitalWrite(BUZZER_PIN, HIGH); delay(200);
162     digitalWrite(BUZZER_PIN, LOW);
163   }
164
165   void playToggleBeep() {
166     digitalWrite(BUZZER_PIN, HIGH);
167     delay(100);
168     digitalWrite(BUZZER_PIN, LOW);
169   }
```

Appendix B: Arduino Code (Without PID Controller)

```
1   #include <Wire.h>
2   #include <LiquidCrystal_I2C.h>
3   #include <Servo.h>
4   #include <IRremote.h>
5   #include <U8x8lib.h>
6   U8X8_SSD1306_128X64_NONAME_HW_I2C oled(/* reset=*/ U8X8_PIN_NONE);
7
8   Servo solarServo;
9
10  #define SERVO_PIN     4
11  #define LDR1_PIN      A0
12  #define LDR2_PIN      A2
13  #define BUTTON_PIN    2
14  #define GREEN_LED     6
15  #define YELLOW_LED    7
16  #define BUZZER_PIN    5
17  #define IR_PIN        3
18
19  #define IR_LEFT   0xF40B0707
20  #define IR_RIGHT  0xF8070707
21
22  const int PHYS_MIN = 25;
23  const int PHYS_MAX = 165;
24  const int PHYS_MID = 95;
25
26  int currentPulse = PHYS_MID;
27  bool autoMode = true;
28  bool lastButtonState = HIGH;
29  unsigned long lastDebounceTime = 0;
30  const unsigned long debounceDelay = 200;
31
32  float pulseToAngle(int pulse) {
33  return map(pulse, PHYS_MIN, PHYS_MAX, 90, -90);
34  }
35
36  int readLDRPercent(int pin) {
37  return constrain(map(analogRead(pin), 50, 950, 0, 100), 0, 100);
38  }
39
40  void playStartupBeep() {
41  digitalWrite(BUZZER_PIN, HIGH); delay(150);
```

```
42  digitalWrite(BUZZER_PIN, LOW);  delay(100);
43  digitalWrite(BUZZER_PIN, HIGH); delay(200);
44  digitalWrite(BUZZER_PIN, LOW);
45  }
46
47  void playToggleBeep() {
48  digitalWrite(BUZZER_PIN, HIGH); delay(100);
49  digitalWrite(BUZZER_PIN, LOW);
50  }
51
52  void setup() {
53  Serial.begin(115200);
54  Wire.begin();
55  oled.begin();
56  oled.setFont(u8x8_font_chroma48medium8_r);
57  oled.clearDisplay();
58  oled.drawString(0, 0, "Solar Tracker");
59
60  pinMode(BUTTON_PIN, INPUT_PULLUP);
61  pinMode(GREEN_LED, OUTPUT);
62  pinMode(YELLOW_LED, OUTPUT);
63  pinMode(BUZZER_PIN, OUTPUT);
64  digitalWrite(BUZZER_PIN, LOW);
65
66  IrReceiver.begin(IR_PIN, ENABLE_LED_FEEDBACK);
67  solarServo.attach(SERVO_PIN);
68  solarServo.write(PHYS_MID);
69
70  digitalWrite(GREEN_LED, HIGH);
71  digitalWrite(YELLOW_LED, LOW);
72
73  playStartupBeep();
74  }
75
76  void loop() {
77  static unsigned long lastSendTime = 0;
78  const unsigned long sendInterval = 20;
79
80  handleModeToggle();
81  digitalWrite(GREEN_LED, autoMode);
82  digitalWrite(YELLOW_LED, !autoMode);
83
84  int ldr2 = readLDRPercent(LDR1_PIN);  // Right
85  int ldr1 = readLDRPercent(LDR2_PIN);  // Left
86  float currentAngle = pulseToAngle(currentPulse);
87
88  if (autoMode) {
89  if (ldr1 > ldr2) currentPulse -= 5;
90  else if (ldr2 > ldr1) currentPulse += 5;
91
92  currentPulse = constrain(currentPulse, PHYS_MIN, PHYS_MAX);
93  solarServo.write(currentPulse);
94
95  } else {
96  if (IrReceiver.decode()) {
```

```
uint32_t code = IrReceiver.decodedIRData.decodedRawData;
if (code == IR_LEFT)  currentPulse -= 5;
if (code == IR_RIGHT) currentPulse += 5;

currentPulse = constrain(currentPulse, PHYS_MIN, PHYS_MAX);
  solarServo.write(currentPulse);
  IrReceiver.resume();
}

}

updateOLED(ldr1, ldr2, currentAngle);


if (millis() - lastSendTime >= sendInterval) {
Serial.print(millis()); Serial.print(",");
Serial.print(ldr1); Serial.print(",");
Serial.print(ldr2); Serial.print(",");
Serial.print(currentAngle, 1); Serial.print(",");
Serial.print(0); Serial.print(",");  // No PID
Serial.println(autoMode ? '1' : '0');
lastSendTime = millis();
}
}

void handleModeToggle() {
bool reading = digitalRead(BUTTON_PIN);
if (reading == LOW && lastButtonState == HIGH && millis() - lastDebounceTime > debounceDelay) {
autoMode = !autoMode;
lastDebounceTime = millis();
playToggleBeep();
}
lastButtonState = reading;
}

 void updateOLED(int ldr1, int ldr2, float angle) {
   oled.clearLine(1); oled.setCursor(0, 1);
   oled.print("L:"); oled.print(ldr1); oled.print(" R:"); oled.print(ldr2);

   oled.clearLine(2); oled.setCursor(0, 2);
   oled.print("Mode: "); oled.print(autoMode ? "Auto" : "Manual");

   oled.clearLine(3); oled.setCursor(0, 3);
   oled.print("Angle: ");
   if (angle >= 0) oled.print("+");
   oled.print(angle, 0);
   oled.print((char)176); // degree symbol
 }
```

Appendix C: MATLAB Visualization Script

This script visualizes LDR readings, servo position, and operation mode in real time using the serial data received from the Arduino.

```
%% Solar Tracker Live Visualization (Improved & Stable Version)
clear; clc; close all;

%% Serial Setup
port = "COM6";       % <-- Replace with your actual COM port
baud = 115200;

try
    s = serialport(port, baud);
    configureTerminator(s, "LF");
    flush(s);
    pause(2);  % Let Arduino initialize
    disp(['Connected to ' port]);
catch ME
    error('Connection failed: %s', ME.message);
end

%% Discard initial unstable readings
for i = 1:10
    if s.NumBytesAvailable > 0
        readline(s);
    end
    pause(0.05);
end

%% Figure Setup
figure('Color','w','Position',[100 100 900 600]);
sgtitle('Solar Tracker System Response','FontSize',14,'FontWeight','bold');

% Subplot 1: LDR Readings
ax1 = subplot(3,1,1);
hLDR1 = plot(nan, nan, 'r-', 'LineWidth',1.5, 'DisplayName','LDR1'); hold on;
hLDR2 = plot(nan, nan, 'b-', 'LineWidth',1.5, 'DisplayName','LDR2');
title('LDR Readings (0-100%)');
ylabel('Light Intensity (%)');
ylim([0 100]);
grid on;
legend('Location','best');

% Subplot 2: Panel Angle
ax2 = subplot(3,1,2);
hAngle = plot(nan, nan, 'Color',[0 0.5 0], 'LineWidth',1.5);
title('Panel Angle');
ylabel('Angle (°)');
ylim([-90 90]);
grid on;
yline(0,'--','0° Horizon');

% Subplot 3: Control Mode (Auto/Manual)
ax3 = subplot(3,1,3);
hMode = stairs(nan, nan, 'k-', 'LineWidth',1.5);
title('Control Mode');
ylabel('Mode');
xlabel('Time (s)');
yticks([0 1]);
yticklabels({'Manual','Auto'});
ylim([-0.1 1.1]);
grid on;

%% Data Buffers
bufferSize = 1000;
timeBuffer  = nan(1, bufferSize);
ldr1Buffer  = nan(1, bufferSize);
ldr2Buffer  = nan(1, bufferSize);
angleBuffer = nan(1, bufferSize);
modeBuffer  = nan(1, bufferSize);
ptr = 1;
```

```
%% Plot Refresh Setup
updateInterval = 0.1;  % seconds
lastUpdateTime = 0;
startTime = tic;

disp('Reading data... Close figure to stop.');

%% Main Loop
while ishandle(gcf)
    while s.NumBytesAvailable > 0
        try
            line = readline(s);

            % Check data structure (must have 5 commas → 6 fields)
            if count(line, ',') ~= 5
                continue;
            end

            data = sscanf(line, '%f,%f,%f,%f,%f,%f');
            if numel(data) ~= 6
                continue;
            end

            % Optional: skip idle responses (both LDRs zero)
            if data(2) == 0 && data(3) == 0
                continue;
            end

            % Store in circular buffer
            timeBuffer(ptr)  = data(1) / 1000;  % Convert ms to seconds
            ldr1Buffer(ptr)  = data(2);
            ldr2Buffer(ptr)  = data(3);
            angleBuffer(ptr) = data(4);
            modeBuffer(ptr)  = data(6);  % skip PID value (data(5))
            ptr = mod(ptr, bufferSize) + 1;
        catch
            % Skip malformed lines silently
        end
    end

    % Refresh plots every updateInterval
    if toc(startTime) - lastUpdateTime >= updateInterval
        valid = ~isnan(timeBuffer);
        t = timeBuffer(valid);

        if isempty(t), continue; end

        set(hLDR1, 'XData', t, 'YData', ldr1Buffer(valid));
        set(hLDR2, 'XData', t, 'YData', ldr2Buffer(valid));
        set(hAngle, 'XData', t, 'YData', angleBuffer(valid));
        set(hMode,  'XData', t, 'YData', modeBuffer(valid));

        % Auto X-axis scrolling (last 10 seconds)
        xLimits = [max(0, t(end)-10), max(10, t(end)+1)];
        set(ax1, 'XLim', xLimits);
        set(ax2, 'XLim', xLimits);
        set(ax3, 'XLim', xLimits);

        drawnow limitrate;
        lastUpdateTime = toc(startTime);
    end

    pause(0.001);  % Prevent CPU overload
end

%% Cleanup
clear s;
disp('Stopped.');
```