HelioTrak

Arduino-Based Single-Axis Solar Tracking System

Project Overview

HelioTrak is a smart, Arduino-powered single-axis solar tracking system designed to maximize solar energy efficiency. It automatically adjusts the orientation of a solar panel based on sunlight intensity detected by two LDR sensors, ensuring improved power generation throughout the day.

This document includes:

- Project summary and objectives
- Components and wiring
- Arduino code (ready to paste into the Arduino IDE)
- Tuning tips, safety notes, and suggested enhancements
- A ready-to-use logo design prompt for AI/graphic tools
- License (MIT)

Objective

Design a low-cost and efficient single-axis solar tracking system that increases solar panel exposure to direct sunlight by automatically aligning the panel using light sensors and servo actuation. The project demonstrates embedded systems skills, sensor processing, and basic automation.

Key Features

- Automatic single-axis tracking using two LDRs
- Arduino control logic and servo actuation
- Deadband and smoothing to avoid servo jitter
- Low-cost, compact prototype suitable for academic submission
- Easily expandable to dual-axis or IoT-enabled versions

Components Required

Component	Qty	Notes
Arduino Uno / Nano	1	Main microcontroller
LDR (Photoresistor)	2	Left and right sensors
10 kΩ resistors	2	For LDR voltage dividers
Servo motor (5V)	1	Standard hobby servo for prototype

Mini solar panel	1	Prototype panel for testing
Breadboard & jumper wires	–	For prototyping
Power supply (5V)	1	External supply recommended for larger servos
Optional: RTC module	1	For time-based enhancements
Optional: Wi-Fi module (ESP8266/ESP32)	1	For IoT telemetry

Wiring & Circuit

1. **LDR Left:** +5V → LDR → A0 → 10kΩ → GND.
2. **LDR Right:** +5V → LDR → A1 → 10kΩ → GND.
3. **Servo:** Signal → D9; VCC → +5V; GND → Arduino GND (common ground).

Important: For larger servos draw significant current, use an external 5V power supply and connect the grounds together. Add mechanical limit switches or physical stoppers to protect the system from over-rotation.

Algorithm / Working Principle

1. Continuously read analog values from the two LDRs.
2. Compute the difference: $\Delta = LDR_{left} - LDR_{right}$.
3. If $|\Delta|$ exceeds a deadband threshold, move the servo a small step toward the brighter side.
4. Use smoothing (moving average) and a deadband to prevent jitter and reduce unnecessary servo motion.

Arduino Code (SolTrak / HelioTrak)

Listing 1: HelioTrak Arduino sketch — single-axis tracker (LDRs + Servo)

```
// SolTrak / HelioTrak - Single-axis solar tracker
// Arduino + 2 LDRs + Servo
#include <Servo.h>

// Pins
const uint8_t PIN_LDR_LEFT  = A0;
const uint8_t PIN_LDR_RIGHT = A1;
const uint8_t PIN_SERVO     = 9;

// Control parameters
const int DEAD_BAND = 25;           // ADC diff under which servo won't move
const int SERVO_STEP = 1;           // degrees per adjustment step
const unsigned long MOVE_DELAY_MS = 60; // min time between moves (ms)

// Servo limits
const int SERVO_MIN_ANGLE = 0;
const int SERVO_MAX_ANGLE = 180;
const int SERVO_START_ANGLE = 90;  // initial center

```

```
// Debug
const bool DEBUG = true;

Servo panelServo;
int currentAngle = SERVO_START_ANGLE;
unsigned long lastMoveTime = 0;

void setup() {
  Serial.begin(9600);
  panelServo.attach(PIN_SERVO);
  panelServo.write(SERVO_START_ANGLE);
  currentAngle = SERVO_START_ANGLE;
  if (DEBUG) {
    Serial.println("HelioTrak started");
    Serial.print("Start angle: "); Serial.println(currentAngle);
  }
}

void loop() {
  int ldrLeft  = analogRead(PIN_LDR_LEFT);
  int ldrRight = analogRead(PIN_LDR_RIGHT);
  int diff = ldrLeft - ldrRight;

  if (DEBUG) {
    Serial.print("L:"); Serial.print(ldrLeft);
    Serial.print(" R:"); Serial.print(ldrRight);
    Serial.print(" diff:"); Serial.println(diff);
  }

  unsigned long now = millis();
  if (now - lastMoveTime < MOVE_DELAY_MS) {
    delay(5);
    return;
  }

  if (abs(diff) > DEAD_BAND) {
    if (diff > 0) {
      // More light on left -> rotate towards left
      int newAngle = currentAngle - SERVO_STEP; // invert sign if
          mechanics reversed
      newAngle = constrain(newAngle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE);
      if (newAngle != currentAngle) {
        panelServo.write(newAngle);
        currentAngle = newAngle;
        lastMoveTime = now;
        if (DEBUG) { Serial.print("Servo -> "); Serial.println(
            currentAngle); }
      }
    } else {
      // More light on right -> rotate towards right
      int newAngle = currentAngle + SERVO_STEP;
      newAngle = constrain(newAngle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE);
      if (newAngle != currentAngle) {
        panelServo.write(newAngle);
        currentAngle = newAngle;
        lastMoveTime = now;
        if (DEBUG) { Serial.print("Servo -> "); Serial.println(
            currentAngle); }
```

```
      }
    }
  }

  delay(20);
}
```

Tuning Tips & Notes

- **DEAD_BAND:** Increase to reduce oscillation near center; decrease for higher sensitivity. Typical range: 10–60.
- **SERVO_STEP:** Smaller steps give finer adjustments but slower tracking.
- **MOVE_DELAY_MS:** Rate-limits servo movement to protect mechanical parts.
- **Smoothing:** For very noisy readings, implement a moving average (buffer) over several samples before computing the difference.
- **Power:** Use an external 5V supply for medium/large servos; always common-ground with Arduino.
- **Mechanical Limits:** Add physical stops or limit switches to prevent over-rotation.
- **Servo Direction:** If the panel moves the wrong way, invert the sign used when changing `currentAngle`.

Optional Enhancements

- **Dual-axis tracking:** Add tilt control (second servo) and two more LDRs or a sun-position algorithm.
- **RTC + astronomical algorithm:** Use sunrise/sunset and solar azimuth to track without LDRs (useful on cloudy days).
- **IoT telemetry:** Send sensor data to ThingSpeak, Blynk, Firebase, or a custom server using ESP8266/ESP32.
- **Data logging:** Record energy production and tracking performance to SD card or cloud.
- **Improve sensors:** Use photodiodes/phototransistors for faster response and better linearity.

Logo Design Prompt (Use with Canva / DALL · E / Figma / Adobe Firefly)

> Create a minimal, modern technology logo for a project named **"HelioTrak"**. The logo should represent solar tracking, automation, and renewable energy.
>
> Style: futuristic, geometric, professional, flat design.
> Colors: Primary solar yellow (approx. #FFD54F), secondary sky blue (approx. #2196F3), accent charcoal gray (#333333).
> Elements (subtle): a sun icon or solar panel symbol; a circular/orbital tracking arc;

a simple line-based sensor/signal graphic to hint at Arduino electronics.
Text: “HelioTrak” in bold, clean sans-serif font (Poppins, Roboto, or similar).
Tone: Smart, sustainable, technical.

Project Metadata / README Snippet (Short)

HelioTrak — Arduino-powered single-axis solar tracking system. Uses two LDRs and a servo to maximize solar exposure and energy capture. Ideal as an academic mini-project demonstrating embedded systems and renewable energy fundamentals.

License

This project is released under the **MIT License**.

MIT License (summary)
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, subject to the conditions in the full MIT license.

Full license text can be added to `LICENSE` file in the repository.

Acknowledgements & Inspiration

Designed for sustainable energy innovation and to teach practical embedded control techniques. Good luck with your project presentation and report!

Prepared for: HelioTrak — Student Engineering Mini Project
Document generated: November 5, 2025