

Design And Implementation Of A Dual-Axis Solar Tracking And Real-Time Monitoring System Using Arduino, MATLAB, And NI LabVIEW-

Anshuman Singh¹, Shubham Manish Chhaparia², Anurag³, Saravanakumar R⁴

1,2,3department Of Electronics And Instrumentation Engineering, School Of Electrical Engineering, Vellore Institute Of Technology, Vellore, India

4department Of Control And Automation, School Of Electrical Engineering, Vellore Institute Of Technology, Vellore, India

Abstract- Low-cost photovoltaic (PV) tracking prototypes are frequently reported as hardware demonstrations, with little attention to how their internal signals can be observed, compared, and reproduced. This paper presents the design and implementation of an Arduino Uno-based dual-axis solar tracking prototype that senses the direction of stronger illumination with four light-dependent resistors (LDRs) behind a cross-shaped divider, positions the panel with two hobby servos on a 3D-printed pan-tilt frame, and additionally monitors panel voltage and soil moisture, with a relay-switched pump load for threshold-based irrigation. The controller streams one nine-field comma-separated frame per cycle over USB serial. Two monitoring clients are described. A MATLAB dashboard reads the serial stream directly and plots the four LDR channels, panel voltage, soil moisture against dry and wet thresholds, and pump state in real time. An NI LabVIEW virtual instrument implements a virtual-sun model of the quadrant error calculation, deadband stepping, and a simulated solar-power estimate, and polls a local Flask HTTP endpoint through its HTTP client functions. We derive the quadrant error model, show that the LabVIEW model's errors reduce to -12 and $+12$ times the angular offsets, and report an eight-point, 140 s observation window of the live dashboard in which the top-right LDR operated close to ADC saturation. No tracking gain, energy yield, or communication latency is claimed. Closed-loop convergence, fixed-versus-tracking comparison, and endpoint latency are instead specified as a reproducible test protocol and identified as future work. The paper therefore documents what was built, what was measured, and what remains to be validated, so that the prototype can be reproduced and extended.

Keywords: Arduino, dual-axis solar tracker, light-dependent resistor, LabVIEW, MATLAB, photovoltaic monitoring, serial-to-HTTP bridge, virtual instrumentation.

I. INTRODUCTION

The power delivered by a photovoltaic module depends on the angle between the incident radiation and the module surface, so a fixed module loses energy as the sun moves through the day and the year. Sun-tracking mechanisms reduce this loss by re-orienting the module about one or two axes, and the design of such trackers has long been treated as an engineering problem in its own right [1]. Measured comparisons of tracking and fixed installations have been reported, including work that explicitly accounts for the energy consumed by the tracker itself [2–4]. That accounting matters most for small systems, where servo and controller

consumption is not negligible against the harvested energy.

A single-axis tracker follows one angle, typically azimuth or elevation, whereas a dual-axis tracker follows both [1,3]. The direction to follow can be computed from a sun-position model or sensed directly. Direct sensing with a four-quadrant arrangement of LDRs is attractive for low-cost systems because it needs no clock, location, or calendar [5], and LDR-driven trackers have been modelled and tested in single-axis form [6]. A substantial body of Arduino-based dual-axis work follows this pattern [7–12]. In that literature the emphasis is on the tracker itself and its efficiency. How the internal quantities of such a system, namely raw channel values, error signals, actuator state, and auxiliary sensor readings, are made observable to

the experimenter receives comparatively little attention. Observability is, however, a measurement problem. Photovoltaic monitoring practice is codified in IEC 61724-1 [13], and a growing set of low-cost Arduino-based loggers and I-V tracers address parts of it [14–17]. Virtual instrumentation, in which a general-purpose computer supplies the display, analysis, and storage functions around a small acquisition front end, is a natural fit for such prototypes [18, 19]. This paper documents a prototype that combines these threads on a single Arduino Uno: a four-LDR dual-axis tracker, a panel-voltage channel, and a soil-moisture channel with a relay-switched pump load. Its distinguishing feature is not a new tracking algorithm but the way the same acquisition stream is exposed to two different monitoring environments, a MATLAB dashboard and an NI LabVIEW virtual instrument, together with an explicit statement of which parts were measured, which were simulated, and which remain to be validated.

1. Objectives And Scope

The objectives are

- to describe the hardware, tracking method, firmware contract, and communication paths of the prototype in enough detail to be reproduced;
- to analyse the quadrant error model and the LabVIEW virtual-sun model;
- to describe and compare the two monitoring implementations;
- to define an evaluation protocol and report only the evidence that is actually available. The prototype is a laboratory demonstration. No long-duration outdoor measurement was performed.

2. Contributions

1. A documented low-cost prototype that integrates four-LDR dual-axis sensing, panel-voltage measurement, and threshold-based pump switching on one Arduino Uno, with a nine-field serial frame that serves as the single data contract between firmware and software.

2. An analysis of the quadrant error model, including identities showing that, for the LabVIEW virtual-sun model, the horizontal and vertical errors are exactly proportional to the angular offsets (Section V).

3. Descriptions of two monitoring clients that consume the same Arduino data, with a candid account of how each reaches the data and what each currently displays (Sections VIII–IX).

4. An evidence-bounded results section built on an eight-point observation window, and a reproducible test protocol for the quantities that have not yet been measured (Sections X and XI). We do not claim a new control algorithm, an efficiency improvement, or superior performance. What is demonstrated, simulated, and proposed is summarised in Table 7.

II. RELATED WORK

1. Fixed Versus Tracking Systems And Energy Accounting

Mousazadeh *et al.* review the principles and methods of sun tracking for maximizing the output of solar systems [1]; the review is a useful taxonomy but predates current low-cost embedded hardware. Lazaroiu *et al.* compare fixed and sun-tracking low-power PV systems while explicitly considering the tracker's own energy consumption [2], which is the correct accounting basis for any small tracker and the reason this paper lists servo energy as a required measurement in Section X. Eke and Şentürk report a measured performance comparison of a double-axis tracker against a fixed PV system [3], and Sungur describes a PLC-controlled multi-axis tracker for PV panels [4]. These studies are single-site and single-system, so their numerical gains do not transfer to other prototypes. Overviews such as [20] and design-and-simulation studies such as [21] complement them but do not provide measured hardware comparisons.

2. Ldr-Based And Arduino-Based Dual-Axis Trackers

Chin *et al.* designed, modelled, and tested a standalone single-axis active tracker using MATLAB/Simulink [6]; it demonstrates the value of model-based verification but is limited to one axis. Wang and Lu implemented an optical-sensor-based dual-axis tracker driven by a single motor and four-quadrant LDR sensing [5], using an AC motor and analog control rather than hobby servos. Arduino-class dual-axis designs include Kaur *et al.* [7], Zolkapli *et al.* [8], Othman *et al.* [9], Nguyen and Ho [10],

Chenchireddy *et al.* [11], and Bajpai *et al.* [12], and a field-oriented performance study at altitude is given in [22]. A single-axis LDR-ATmega328P-servo design is described in [23]. Most of these are short conference papers, and the available descriptions emphasise tracker construction and efficiency rather than the observability of the control signals. Table 1 compares them; a dash marks an item not assessed here.

3. Sensing And Actuation Limitations

LDRs are inexpensive but not ideal sensors. Their resistance responds nonlinearly to illumination and depends on temperature in a thermistor-like manner [24], so channel mismatch and thermal drift enter the differential error directly. Hobby RC servos hide their internal control structure and provide no torque or load feedback [25], which limits what can be inferred about positioning accuracy from commands alone. These limitations motivate the explicit calibration and servo-response tests in Section X.

4. Pv Measurement And Low-Cost Instrumentation

IEC 61724-1 defines monitoring classes and sensor requirements for PV system performance [13]. Low-cost alternatives exist at smaller scales. Ahmed *et al.* log voltage, current, and power with an Arduino under an indoor solar emulator [14]; Muñoz *et al.* built an educational I-V tracer on an Arduino Micro whose maximum power differed by about 1.3 % from a calibrated tracer [15]; Rashid and Nour describe an Arduino Uno I-V and P-V curve tracer [16]; and Oladimeji *et al.* report an Arduino-based solar parameter-measuring system with a data logger checked against standard instruments [17]. These systems are mostly indoor, low-power, or lacking a calibrated irradiance reference, which is also a limitation of the present work.

5. Labview And Matlab Monitoring

LabVIEW has been applied to PV monitoring and data acquisition, for example battery modelling in a PV system [26], module fault monitoring [27], and an integrated data-acquisition system [28]. Arduino-to-LabVIEW and Arduino-to-MATLAB acquisition has been demonstrated in teaching and laboratory settings [19, 29], and architectures with Arduino boards as building blocks for data acquisition and control are surveyed in [18]. On the MATLAB side, a GUI-based PV simulation and monitoring model is given in [30] and a real-time

serial acquisition tool for Arduino-based instruments in [31]. None of these places a LabVIEW client and a MATLAB client beside each other on one low-cost solar prototype, which is the arrangement documented here.

III. HARDWARE DESIGN AND INSTRUMENTATION

Figure 2 shows the assembled prototype. The Uno, a breadboard, and a four-channel relay module sit on a hardboard base beside the 3D-printed pan-tilt frame that carries the solar panel and the LDR cross. A resistive soil probe stands in a small cup, and a small DC motor acts as the pump load.

1. Directional Sensing

Each LDR forms a voltage divider with a fixed resistor. The project documentation states that each channel's reading rises with brighter light, which corresponds to the LDR occupying the upper leg of the divider, so that
$$N = \frac{1023}{R_f + R_{LDR,i}} \frac{V_{CC}}{V_{ref}}, (1)$$

2. Network-Based Monitoring And Irrigation

Web-based PV monitoring has been demonstrated with an embedded web server [32] and an IoT architecture for a greenhouse PV installation [33], and JSON-over-REST is an established pattern for cross-platform data exchange [34–36]. For the irrigation channel, laboratory work shows that low-cost capacitive and resistive soil-moisture sensors need calibration and recalibration [37], and a solar-powered irrigation prototype switches a pump on soil-moisture and temperature thresholds [38].

3. System Architecture

The prototype has three layers, shown in Figure 1. The field layer contains the sensors and actuators. The embedded layer is a single Arduino Uno [39] that digitises the analog channels, makes the tracking and irrigation decisions, drives the actuators, and emits one serial frame per cycle. The host layer contains the monitoring software, which receives the frame by one of two paths.

4. Division Of Functions

On the Arduino the following run every loop pass: analog acquisition of four LDRs, the panel-voltage module, and the soil probe; the tracking calculation and

servo stepping; the hysteresis decision for the pump relay; and formatting of one serial frame. On the host computer, the MATLAB dashboard parses the frame and plots it; the LabVIEW VI computes a virtual-sun model, displays its indicators, and issues an HTTP GET to a local endpoint. The Flask bridge, as referenced by the VI, is the process that would turn the serial stream into an HTTP response.

5. Component Summary

Table 2 lists the components of the prototype. Part numbers and electrical ratings are intentionally omitted where the project documentation does not give them. where $N_i \in [0, 1023]$ is the ADC count of channel i , R_f is the fixed resistor, and $V_{CC}/V_{ref} = 1$ when the default 5 V reference is used. The Uno's converter has 10-bit resolution [39], so one count corresponds to about $5/1023 \approx 4.9$ mV at the pin. Because R_{LDR} falls with illumination, N_i approaches 1023 once $R_{LDR} \ll R_f$. The ADC counts therefore saturate in strong light, a behaviour that is visible in the measurements of Section XI. The four LDRs sit in the quadrants of a cross-shaped

3D-printed divider (Figure 3). When the panel faces the light, the divider shades all quadrants equally; when the light is off-axis, it shades the opposite quadrants and the readings differ.

6. Actuation And Power

Two hobby servos move the panel in the horizontal (azimuth) and vertical (elevation) directions. The project documentation assigns the horizontal servo to D9 and the vertical servo to D10 and describes one-degree steps per loop pass. Servos draw current spikes at start-up and stall, so they should not share a weak supply with the Arduino logic; the documentation recommends an external 5 V servo supply with a common ground. The Uno is powered from the laptop USB port. The relay module's logic input is driven from one digital pin, while the pump current flows through the relay contacts from the separate 9 V battery. Because a DC motor is an inductive load, a flyback diode across the motor terminals is recommended.

Ref.	Tracking approach	Controller / sensing	Evaluation reported	Identified limitation	Relevance here
[1]	Review of tracking principles and methods	—	Literature review	Dated hardware detail	Taxonomy of tracking methods
[2]	Fixed vs. sun-tracking, low-power PV	—	Comparison including tracker energy consumption	Single site and system	Energy-accounting basis (Sec. X)
[3]	Dual-axis vs. fixed	—	Measured performance comparison	Single location, short period	Motivates fixed-vs-tracking protocol
[4]	Multi-axis, PLC control	PLC	—	Different cost class	Controller-based baseline
[6]	Single-axis active, two LDRs	Microcontroller + LDR	Simulink model checked against experiments	One axis only	Model-based verification
[5]	Dual-axis, single motor	Four-quadrant LDR, AC motor	Net energy gain on a partly cloudy day	No hobby servos	Four-quadrant sensing precedent
[7, 8]	Dual-axis, Arduino	Arduino + LDR	Short prototype tests	Short conference papers	Closest hardware class
[11]	Dual-axis, four LDRs, servos	Arduino Uno + LDR	Prototype description	No deadband or backlash discussion	Same sensing architecture
[9, 12]	Dual-axis LDR performance analysis	LDR-based	Performance analysis	Short conference papers	Evaluation approach
[14, 17]	—	Arduino V/I/light logging	Indoor emulation; comparison with standard instruments	No calibrated irradiance reference	Low-cost PV instrumentation
[26–28]	—	LabVIEW front ends	PV monitoring / DAQ / fault monitoring	Not tracker-specific	LabVIEW precedent
[32, 33]	—	IoT / ESP32 web server	Field or prototype PV monitoring	Cloud/IoT oriented	Network monitoring precedent
This work	Dual-axis, four-LDR quadrant sensing	Arduino Uno + MATLAB + LabVIEW + Flask path	Observation window only; protocol defined	No tracking gain or latency measured	—

7. Panel Voltage, Soil Moisture, And Relay

The panel voltage is read through a voltage-sensor module that scales it into the ADC range. No current sensor is installed: an INA219 power monitor is listed as planned but was not used, so the physical prototype measures voltage only. The soil probe is resistive with a

comparator board and its analog output rises as the soil dries. Resistive probes are known to drift and need calibration [37]. The controller classifies a reading S as dry above 650 and wet below 500, as described in Section VI.

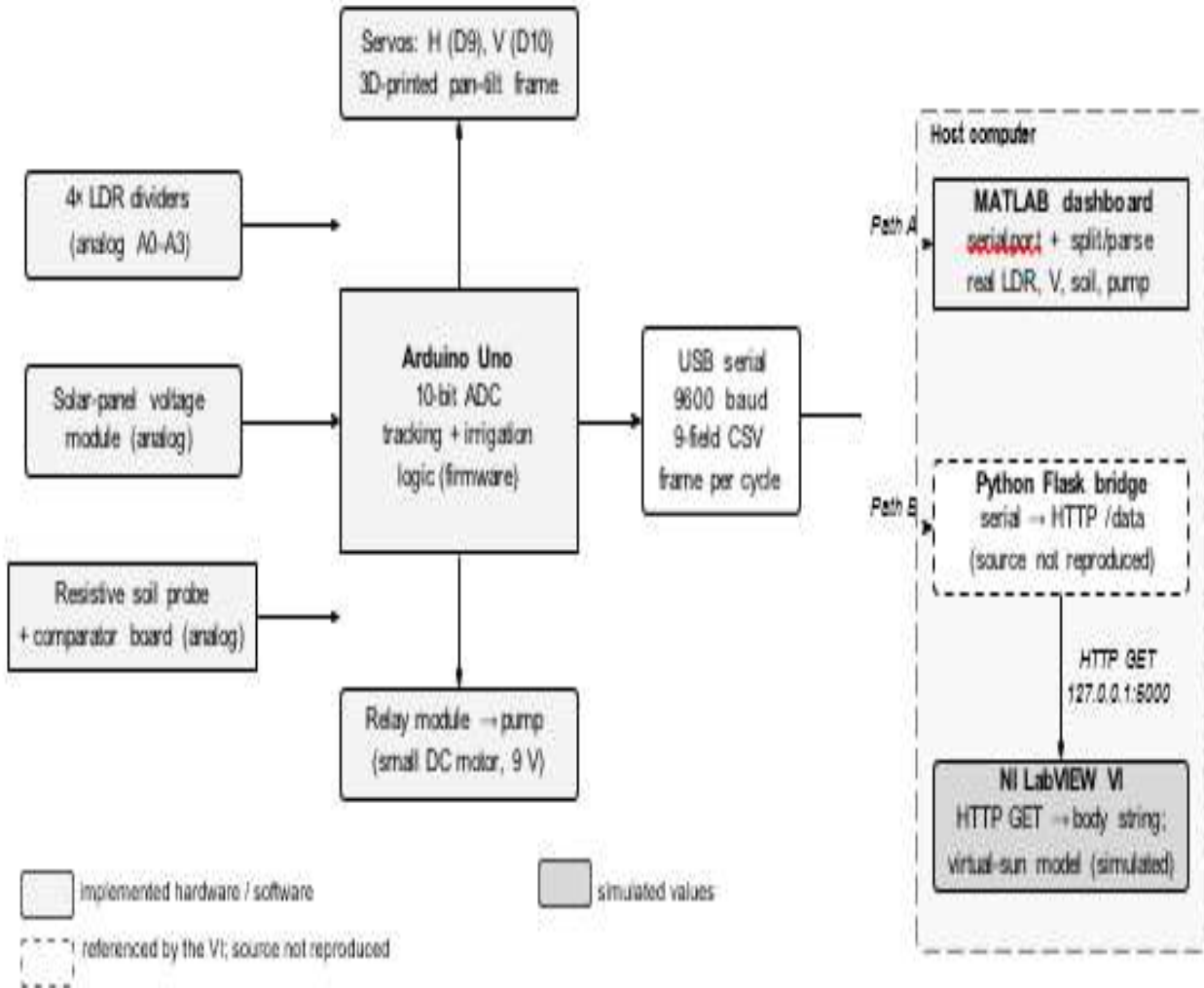


Figure 1: Overall architecture of the prototype (conceptual diagram drawn for this paper). Path A, MATLAB reading the serial port directly, is the path whose code is documented. Path B passes through a Python Flask bridge to the LabVIEW HTTP client; the bridge is referenced by the LabVIEW VI but its source is not reproduced here (dashed). The LabVIEW indicators for LDR values, errors, voltage, current, and power are generated by a virtual-sun model (shaded)

D. Pin Mapping

Table 3 gives the pin mapping of the prototype.

IV. DUAL-AXIS TRACKING METHOD

A. Quadrant Error Model

Let TL, TR, BL, and BR denote the ADC counts of the four LDRs. The left, right, top, and bottom sums are

$$\begin{aligned} SL &= TL + BL, & SR &= TR + BR, \\ ST &= TL + TR, & SB &= BL + BR, \end{aligned}$$

Component	Qty	Role
Arduino Uno (ATmega328P)	1	Acquisition, tracking and irrigation logic, serial output
LDR with fixed resistor (about 10 kΩ)	4	Quadrant light sensing (TL, TR, BL, BR)
Panel-voltage sensor module	1	Scales panel voltage into ADC range
Mini solar panel	1	PV source on the pan-tilt frame
Micro servo motors	2	Horizontal and vertical panel motion
3D-printed pan-tilt frame and cross divider	1 set	Mechanical support; quadrant shading
Resistive soil-moisture probe with comparator board	1	Analog soil reading (higher = drier)
4-channel relay module (one channel used)	1	Switches the pump load
Small DC motor	1	Pump load in the demonstration
9 V battery	1	Supply for the relay/pump circuit
Laptop, USB cable	1	Host computer; serial link and USB power

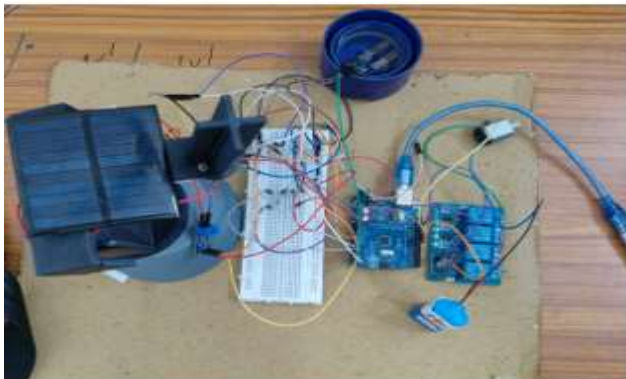


Figure 2: Top view of the prototype (photograph): pan-tilt frame with solar panel and LDR cross (left), breadboard, Arduino Uno, four-channel relay module, 9 V battery, soil probe in a cup (top), and the DC motor used as the pump load.

and the horizontal and vertical errors are

$$\begin{aligned} E_H &= S_L - S_R, & E_V &= S_T - S_B, \\ (3) \end{aligned}$$

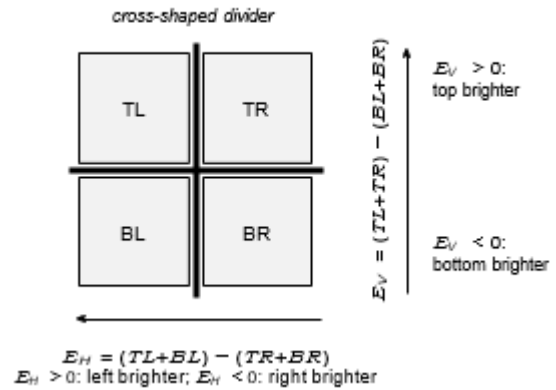


Figure 3: Four-LDR quadrant arrangement and the sign convention of the horizontal and vertical errors (conceptual diagram).

side is brighter, a positive E_V means the top is brighter, and a balanced panel yields $E_H \approx E_V \approx 0$ (Figure 3). Note that the sign of E_H is opposite to the alternative convention $E' = (TR + BR) - (TL + BL)$; only the H pairing between the sign of the error and the direction of servo motion matters, and it is fixed here by the mounting.

4. Deadband Stepping Rule

With tolerance τ (the “Tracking Threshold”), define the three-level function

$$\begin{aligned} s(E) &= \begin{cases} +1, & E > \tau, \\ -1, & E < -\tau, \\ 0, & |E| \leq \tau. \end{cases} \\ (4) \end{aligned}$$

Figure 2: Top view of the prototype (photograph): pan-tilt frame with solar panel and LDR cross (left), breadboard, Arduino Uno, four-channel relay module, 9 V battery, soil probe in a cup (top), and the DC motor used as the pump load.

5. Labview Virtual-Sun Model

The LabVIEW VI does not read LDR hardware. It generates the four channels from the difference between a virtual sun and the panel orientation. With sun angles (φ_s, θ_s), panel angles (φ_p, θ_p), offsets $\Delta\varphi = \varphi_s - \varphi_p$ and $\Delta\theta = \theta_s - \theta_p$, base level $N_0 = 600$, and gain $g = 3$ (the two constants in the block diagram),

$$\begin{aligned} TL &= N_0 + g(-\Delta\varphi + \Delta\theta) \\ TR &= N_0 + g(+\Delta\varphi + \Delta\theta), \\ BL &= N_0 + g(-\Delta\varphi - \Delta\theta), \\ BR &= N_0 + g(+\Delta\varphi - \Delta\theta). \end{aligned}$$

(6)

Substituting (6) into (2) and (3) gives the identities

$$\begin{aligned} E_H &= -4g \Delta\varphi = -12 \Delta\varphi, \\ E_V &= +4g \Delta\theta = +12 \Delta\theta. \end{aligned}$$

(7)

These identities can be checked against the front-panel screenshot of Figure 7, whose indicators read $TL = 494.957$, $TR = 825.127$, $BL = 374.873$, and $BR = 705.043$. From (6), $\Delta\varphi = (TR - TL)/(2g) = 55.03^\circ$ and $\Delta\theta = (TL - BL)/(2g) = 20.01^\circ$; then (7) predicts $E_H = -660.3$ and $E_V = +240.2$, matching the displayed -660.3 and 240.167 . The pseudocode of the model is taken from the project documentation, and the arithmetic above confirms that it reproduces the screenshot. The VI also reports a simulated solar voltage, current, and power. The displayed 13.1305 V and 1.45895 A are the products of 18 V and 2 A (constants visible in the block diagram) with the same factor 0.7295 , consistent with a common efficiency scaling of the model; the efficiency indicator itself is not visible in the front-panel screenshot of Figure 7, and $P = V I = 19.157$ W is a model output rather than a measurement.

6. Practical Considerations And Limitations

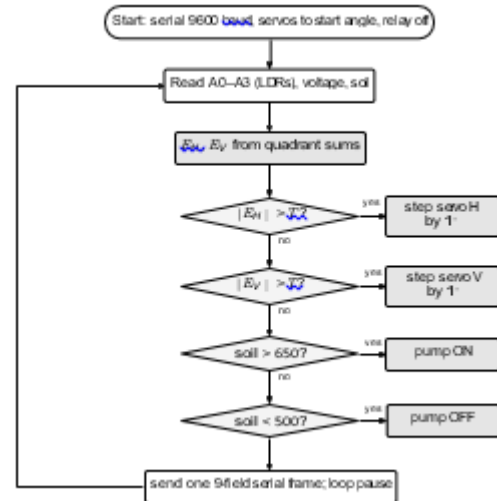
Noise and deadband. A threshold of $\tau = 1$, the value shown on the LabVIEW front panel, is far below the fluctuations of the real channels (Section XI); a physical deadband should be chosen several standard deviations above the error noise measured under static, uniform il-
interface for this is the Servo library [40]. The flowchart of the combined loop is given in Figure 4.

V. EMBEDDED FIRMWARE AND DATA ACQUISITION

The firmware listing is not reproduced here. The description below follows the project algorithm, flowchart, and serial format, and the MATLAB parser, which fixes the serial contract exactly.

1. Acquisition And Tracking

Each loop pass reads the four LDR channels, the panel-voltage module, and the soil probe with the 10-bit ADC [39], forms the sums and errors of (2) and (3), and applies(5) to the two servos. Servo positions are commanded with PWM pulses on D9 and D10; the standard Arduino which prevents the relay from chattering around a single set point. The corresponding soil classes are dry ($S > 650$), moderate, and wet ($S < 500$). The thresholds are those stated in the project report and drawn on the MATLAB dashboard; they are raw ADC values, not calibrated moisture percentages.



Figure

4: Control and data flow of the Arduino loop, reconstructed from the project report's algorithm and flowchart (the firmware listing is not reproduced).interface for this is the Servo library [40]. The flowchart of the combined loop is given in Figure 4

2. Pump Hysteresis

With raw soil reading S (higher means drier), the pump state $P[k]$ follows

$$P[k] = \begin{cases} \text{ON}, & S > 650, \\ \text{OFF}, & S < 500, \\ P[k-1], & 500 \leq S \leq 650, \end{cases} \quad (8)$$

3. Serial Frame

The controller sends one line per cycle at 9600 baud, terminated by a line feed, with nine comma-separated fields (Table 4). An example is 1022,381,320,223,26,0.20,891,ON,WAITING, which corresponds to the values shown on the MATLAB dashboard in Figure 6. The first four fields are ordered TR, TL, BL, BR; the MATLAB code reorders them to TL, TR, BL, BR. Only one servo angle is transmitted, which the dashboard labels "Servo"; no horizontal-servo field exists in the frame.

4. Timing And Error Handling

The project documentation states a 200 ms loop, and the MATLAB dashboard keeps 300 samples and displays about 150 s of history. At the firmware level no error handling is documented; at the receiving end, any line

Table 4: Serial Frame Fields (9600 Baud, Line-Feed Terminated)

#	Field	Type	Meaning (example)
1	TR	integer	Top-right LDR, ADC counts 0–1023 (1022)
2	TL	integer	Top-left LDR (381)
3	BL	integer	Bottom-left LDR (320)
4	BR	integer	Bottom-right LDR (223)
5	angle	numeric	One servo angle in degrees (26)
6	volt	float	Panel voltage in V (0.20)
7	soil	integer	Raw soil reading, higher = drier (891)
8	pump	text	Relay state ON/OFF (ON)
9	state	text	Tracker state (WAITING)

that does not contain exactly nine fields or that contains non-numeric values in the numeric fields is discarded by the MATLAB client.

VI. PYTHON FLASK COMMUNICATION ARCHITECTURE

1. Role And Evidence

The LabVIEW VI requests <http://127.0.0.1:5000/data>, a loop-back address and port that are the Flask development-server defaults [41]. The project re-port identifies Flask as the web server behind the local data endpoint. The bridge source is not reproduced here, so this section states what the bridge must do given the serial contract of Table 4 and the VI's behaviour, and does not claim measured properties of the implementation.

2. Required Behaviour

A serial-to-HTTP bridge of this kind has three duties. First, it holds the serial port open and reads lines from the Ar-duino, for example with pySerial [42]. Second, it validates and parses each line into the nine fields of Table 4 and keeps only the most recent valid reading. Third, a route

/data returns that reading in a JSON object [35] in response to an HTTP GET [36], which Flask can do with its built-in JSON response support [41]. A GET is a safe, read-only request, which suits a "latest value" endpoint, and decoupling acquisition from request handling lets several clients poll without touching the serial port. This request/response pattern follows the JSON-over-REST approach used for cross-terminal data exchange [34] and web-based PV monitoring [32, 33].

Figure 5 gives an illustrative response that mirrors the serial frame. The field names, and the two timestamp fields marked as proposals, are illustrative and are not extracted from the implementation.

3. Reliability Considerations

Stale readings. If the Arduino stops sending, a "latest value" endpoint keeps serving the last reading with no indication of age; the proposed timestamp fields allow a

```
{ "tr": 1022, "tl": 381, "bl": 320, "br": 223,
  "angle": 26, "voltage": 0.20, "soil": 891,
  "pump": "ON", "state": "WAITING", "t_ms":
  1760590000000, <- proposed "age_ms": 120 } <-
  proposed
```

Figure 5: Illustrative JSON body for the /data endpoint (field names illustrative; the two marked fields are proposals for stale-reading detection).

client to detect this. *Malformed lines.* Start-up text and truncated frames must be rejected by the same nine-field rule that the MATLAB client applies. *Connection loss.* The serial port must be re-opened when the USB cable is disturbed. *Port exclusivity.* A serial port is normally held by one process at a time, so the MATLAB client of Path A and the bridge of Path B cannot both be assumed to read the same port simultaneously. The project pro-vides no demonstration of simultaneous operation, and no latency, update-rate, or loss figures for the endpoint have been measured; a protocol for obtaining them is given in Section X.

4. Matlab Monitoring Implementation

The MATLAB dashboard is a single script that acts as a live software instrument on the serial stream. Its captured output is shown in Figure 6.

5. Data Retrieval And Parsing

The script opens the serial port at 9600 baud with a line-feed terminator and flushes stale bytes using serialport [43]. The port name in the script is a macOS device path, so it must be edited on other systems. The main loop runs while the figure exists. If no bytes are available it waits 10 ms; otherwise it reads one line, splits it on commas, and discards the line unless it has exactly nine fields. The first seven fields are converted with str2double, and any line yielding a NaN is discarded. The pump and state fields are kept as strings. The four LDR values are reordered from the serial order (TR, TL, BL, BR) to TL, TR, BL, BR.

6. Layout And Real-Time Updates

A 3×3 tiled layout [44] holds six axes. Two tiles show the four LDR traces; one tile shows a bar graph of the current LDR levels; two tiles show the panel-voltage trend; one tile

is a monospaced text panel listing the port, the four LDR values, the servo angle, voltage, soil reading, pump state, and tracker state; two tiles show the soil-moisture trend with dashed lines at 650 (dry) and 500 (wet); and one tile shows the pump state as a stair trace that is green when ON and red when OFF. Rolling buffers of 300 samples are preallocated with NaN so that nothing is drawn until data arrive. Each pass updates the line and bar objects, slides the time axis to the most recent 150 s, rescales the voltage axis to at least 5 V, rewrites the status text, and redraws with rate limiting.



Figure 6: MATLAB “Solar Tracker & Irrigation Monitor” dashboard (screenshot of captured live data, about 545–693 s). Top row: LDR trends and current LDR bar graph; middle row: panel voltage and status text; bottom row: soil moisture with dry (650) and wet (500) thresholds, and pump state. Status text: TL 381, TR 1022, BL 320, BR 223, servo 26°, voltage 0.20 V, soil 891, pump ON, tracker WAITING.

7. Error Handling

Error handling is limited to discarding malformed lines, terminating when the figure is closed, and closing the port through a cleanup object. There is no reconnection logic and no indication on the dashboard when the serial stream stops, other than a frozen display.

8. What The Captured Dashboard Shows

In the captured window the top-right LDR trace sits at the top of the scale (about 1020 counts) while the other three remain at roughly 370, 315, and 200–220 counts, so TR is the brightest and BR the dimmest quadrant. The panel voltage hovers near 0.2 V with a small ripple and shows spikes of about 1.7 V and 1.9 V near 568 s and 598 s and a smaller disturbance near 629 s (values read

from the plot). The soil trace stays near 890, far above the dry threshold, and the pump trace remains at ON throughout.

9. Ni Labview Monitoring Implementation

The LabVIEW implementation is a virtual instrument (VI) built from a front panel and a block diagram. Unlike the MATLAB dashboard, its displayed LDR, error, voltage, current, and power values come from the virtual-sun model of Section V; it additionally contains an HTTP client that polls the Flask endpoint.

Front Panel Figure 7 shows the front panel in Manual operating mode. The controls are two knobs, Sun Vertical Angle (0–90°) and Sun Horizontal Angle (0–180°), a numeric Tracking Threshold (set to 1), an enumerated Operating Mode control, and a STOP button. The indicators are four numeric LDR displays (TL, TR, BL, BR), Horizontal and Vertical Error displays, two gauges for Panel Vertical Angle and Panel Horizontal Angle, numeric Solar Voltage, Solar Current, and Solar Power displays, and a waveform chart titled “Solar Performance.” The screenshot shows the Operating Mode control set to Manual. The front panel contains no soil-moisture or pump indicators; the irrigation display exists only in the MATLAB dashboard.

10. Block Diagram

The block diagram of Figure 8 has the following structure.

- While Loop. All processing sits inside one While Loop that runs until the STOP SYSTEM Boolean becomes true.
- Shift registers. Two shift registers carry the panel angles from one iteration to the next; they are initialised with the constants 90 and 45.
- Virtual sun and LDR model. Subtract, multiply, and add nodes with the constants 600 and 3 implement (6).
- Errors and stepping. Sums and differences form EH and EV as in (3); absolute-value and comparison nodes test them against the Tracking Threshold; Select nodes with the constants 1, 0, and –1 choose the step that is added to the shift register.
- Efficiency and solar values. Arithmetic nodes with the constants 100, 135, 18, and 2 produce a tracking-

VII. EXPERIMENTAL METHODOLOGY AND EVALUATION

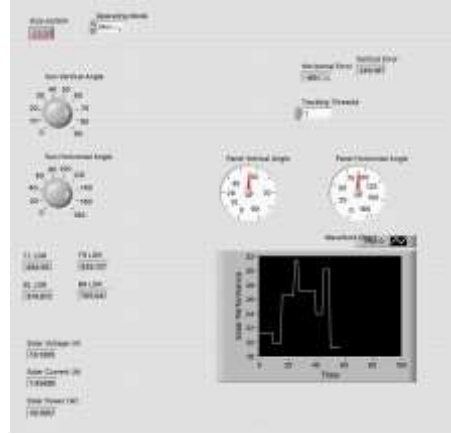


Figure 7: LabVIEW front panel in Manual mode (screenshot). The LDR, error, voltage, current, and power values are produced by the virtual-sun model, not measured

This section defines how the prototype should be evaluated. Only the observation window of Section XI was actually obtained; everything else in Table 5 is a protocol with status “not yet performed.”

1. Metrics

For n paired voltage readings, with module reading V_i and reference reading V_i^{ref} ,

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |V_i - V_i^{\text{ref}}| \quad (9)$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (V_i - V_i^{\text{ref}})^2} \quad (10)$$

Serial and communication quality are summarised by the loss rate

$$\rho_{\text{loss}} = 1 - \frac{N_{\text{valid}}}{N_{\text{sent}}} \quad (11)$$

where N_{sent} is taken from a firmware frame counter and

N_{valid} is the number of frames passing the nine-field check, and by the request latency

$L_j = t_{resp,j} - t_{req,j}$ (12) reported as median and 95th percentile over $j = 1, \dots, m$

requests. For the tracking comparison, the energy over an observation interval $[0, T]$ is

$$E = \int_0^T v(t) i(t) dt, \quad (13)$$

the normalised gain of the tracking panel over the fixed panel is

$$G = \frac{E_{track} - E_{fixed}}{E_{fixed}}, \quad (14)$$

and, following the energy-accounting argument of [2], the net gain subtracts the energy consumed by the servos and controller:

$$G_{net} = \frac{E_{track} - E_{act} - E_{fixed}}{E_{fixed}}. \quad (15)$$

The comparison must hold irradiance, time, temperature, orientation, and shading equal for both panels; swapping the two panels' roles between days and using a calibrated irradiance sensor remove module mismatch and sky variation

12. Labview As An Instrumentation Environment

The front panel gives an instrument-like interface with knobs, gauges, numeric displays, and a chart, with no display code, and dataflow wiring makes the relationship between controls and indicators directly visible. Together with built-in HTTP client functions [46], this allows a monitoring panel to be assembled quickly, which is the usual motivation for virtual instrumentation around Arduino acquisition [18, 29]. In the present VI these strengths are used for the simulation front end; connecting the HTTP response to the indicators is the missing step (Section XIII).

The comparison must hold irradiance, time, temperature, orientation, and shading equal for both panels; swapping the two panels' roles between days and using a calibrated irradiance sensor remove module mismatch and sky variation.

VIII. RESULTS AND DISCUSSION

1. Observation Window

Table 6 lists eight samples read from the live MATLAB dashboard of Figure 6 at 20 s intervals. The project report gives the channel values as approximate to about ± 10 counts, because they were read from the plotted traces. The errors in Table 6 were recomputed from the listed channel values with (3).

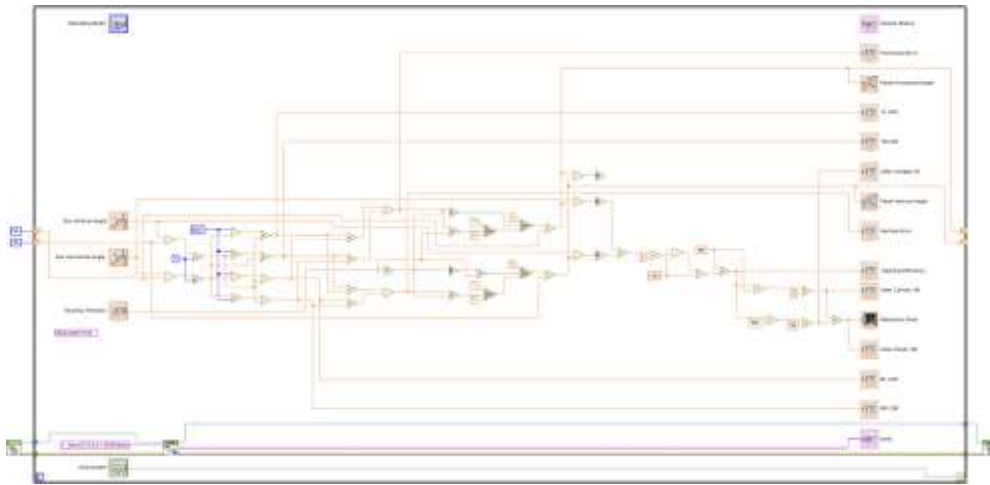


Figure 8: LabVIEW block diagram of the solar tracker VI (screenshot). The HTTP Open/GET/Close chain with the URL constant `http://127.0.0.1:5000/data` runs along the bottom of the While Loop; the body string indicator is at the lower right.

Table 5: Evaluation Protocol and Status

#	Test	Procedure	Metrics	Status
1	Sensor consistency	Uniform diffuse illumination; record ≥ 1000 frames; repeat with LDRs swapped between quadrants	Mean, std. per channel; inter-channel mismatch; offset to subtract	Not performed
2	Directional response	Place a lamp left, right, above, and below at fixed distance; log EH, EV and servo angle	Sign agreement; successful-response rate; time to $ E \leq \tau$	Not performed
3	Servo response	Command $\pm$ steps, measure the resulting angle with a protractor or camera; reverse direction to expose backlash	Settling time; steady-state error; backlash	Not performed
4	Voltage accuracy	Compare module reading with a reference multimeter at 10 or more voltages	MAE, RMSE (Eqs. (9),(10))	Not performed
5	Serial integrity	Add a frame counter to the firmware; log for ≥ 1 h	Valid-frame ratio; loss rate (Eq. (11))	Not performed
6	Endpoint latency and refresh	Time each GET at the client; log client refresh interval	Median and 95th-percentile latency (Eq. (12)); interval	Not performed
7	Fixed vs. tracking energy	Two identical panels, identical loads, simultaneous operation, panels swapped on alternate days; calibrated irradiance reference per IEC 61724-1 [13]; log current and voltage	Energy, normalised gain, net gain with servo energy (Eqs. (13)–(15))	Not performed (no current sensor fitted)

Table 6: Values Read From the Live Dashboard (Approximate, ± 10 Counts) and Recomputed Errors

Time (s)	TL	TR	BL	BR	EH	EV	Soil	Pump
550	372	1022	315	200	−535	+879	895	ON
570	376	1022	312	198	−532	+888	893	ON
590	373	1022	313	200	−536	+882	893	ON
610	368	1020	318	212	−546	+858	892	ON
630	365	1015	318	212	−544	+850	850	ON
650	345	1005	312	216	−564	+822	892	ON
670	362	1018	316	221	−561	+843	891	ON
690	381	1022	320	223	−544	+860	891	ON
Mean	367.8	1018.2	315.5	210.2	−545.2	+860.2	—	—
Std. dev.	11.0	5.9	3.0	9.8	11.8	22.3	—	—

2. Tracking Behaviour

Across the window, TR remains between 1005 and 1022 counts, within about 1.7 % of the 1023-count full scale, so the quadrant error is compressed by saturation of one channel. The errors are $E_H = -545 \pm 12$ and $E_V = +860 \pm 22$ (mean and sample standard deviation of the eight readings), both far larger than any plausible deadband, and both indicate the light is strongest on the right and top. The status frame of the snapshot, however, reports the tracker state as WAITING and a servo angle of 26° . Only a single frame's angle is available and no servo-angle trajectory was logged, so the captured data show that the sensing and visualisation chain works and that the error signals have the expected sign for a bright top-right quadrant.

They do not show a panel converging on the light, nor which axis the 26° refers to. The cause of the WAITING state under persistently large errors cannot be determined from the captured data alone; a logged servo angle on both axes would be needed. Consequently, the tracking function remains a hardware and logic implementation whose closed-loop performance is not yet demonstrated.

The standard deviations of the individual channels, 3 to 11 counts, are comparable to the stated reading uncertainty and therefore include digitisation of the

plot; they are an upper bound on the true static noise and should be re-measured at the data source (test 1 of Table 5).

3. Panel Voltage, Soil, And Pump

The panel voltage baseline is about 0.2 V with transient spikes near 1.7 and 1.9 V, which are discussed qualitatively because the light conditions of the capture are not recorded. No current is measured, so physical power and energy are unavailable; the only power value in the system, 19.157 W in the LabVIEW panel, is a model output. The soil reading sits near 890 for the whole window, classified as dry, with a single dip to about 850 near 630 s that is much smaller than the 240-count gap to the dry threshold.

The pump trace therefore stays ON, no wet or moderate transition occurs, and the hysteresis behaviour of (8) was not exercised in the captured

record. The pump is a DC motor, so no water is delivered. Resistive soil probes require calibration and drift [37], so the absolute soil values should not be interpreted as moisture percentages.

4. Data Sources Of The Two Interfaces

The two interfaces do not currently consume the same software data source. The MATLAB dashboard reads the Arduino serial port directly. The LabVIEW VI computes its displayed values from a model and polls the Flask endpoint, whose response it displays only as text. Both are clients of the same Arduino only in the sense that the Flask bridge, if running, would be fed by the same serial stream; simultaneous operation was not demonstrated, and Section VII explains why a single serial port limits it. Table 7 summarises what is implemented, simulated, and proposed.

Table 7: Implementation Status

Item	Status
Four-LDR pan-tilt prototype	Implemented (hardware; Figure 2)
Panel-voltage channel	Implemented; voltage only
Soil probe, relay, pump load	Implemented; no water delivered; no transitions captured
MATLAB live dashboard	Implemented; live capture (Figure 6)
LabVIEW virtual-sun VI	Implemented; values simulated
LabVIEW HTTP GET of /data	In block diagram; response shown as text only
Flask bridge	Referenced; source not reproduced
Closed-loop tracking convergence	Not demonstrated
Fixed vs. tracking energy	Not measured
Current and power	Simulated in LabVIEW only
Latency, loss, refresh interval	Not measured
IoT/cloud logging	Proposed only

IX. MATLAB AND LABVIEW COMPARISON

Table 8 compares the two implementations as they exist in this project, not the two environments in general. They are best understood as separate clients: one reads the serial line, the other is a simulation front end with an HTTP client. Having both is useful because the MATLAB script is the more direct observation of the real hardware, while the LabVIEW VI lets the tracking logic and the error identities of (7) be explored without hardware and gives a graphical instrument panel; this complementarity is a design observation, not a measured result. MATLAB could also consume the same JSON endpoint with webread [47], which would make the two clients truly parallel.

X. LIMITATIONS AND FUTURE WORK

1. Limitations

- Quantitative validation. Only an eight-point window of approximate, digitised values is available; no track-ing accuracy, energy gain, latency, or voltage error is established.
- Sensing. LDR mismatch, thermal drift, saturation of the TR channel, shadows, and cloud transients degrade the differential error [24].
- Mechanics. Servo resolution, backlash, frame flexibility, and cable drag limit the angular accuracy, and hobby servos give no position feedback [25].
- Electrical measurement. The prototype measures panel voltage only. Without current and a calibrated irradiance reference, neither power nor efficiency can be stated.

Communication. The Flask bridge is not characterised in this paper, the LabVIEW client does not parse its response, and a single serial port constrains simultaneous clients.

Table 8: MATLAB Versus LabVIEW Implementations in This Project

Aspect	MATLAB dashboard	LabVIEW VI
Development model	Text script; one file	Graphical dataflow; front panel and block diagram
Data retrieval	Direct serialport read of the nine-field frame	HTTP client GET of the Flask endpoint; response shown as a string
Data displayed	Real LDR, voltage, soil, pump, status	Model-generated LDR, errors, angles, V, I, P
Visualisation	Tiled line, bar, and stair plots; status text	Knobs, gauges, numeric indicators, waveform chart
Instrument interface	Serial port	HTTP; serial through VISA was not used
Graphing	150 s rolling window with threshold lines	Scrolling waveform chart of one performance value
Error handling	Discards malformed lines; cleanup closes the port; no reconnection	Error wire through HTTP open/GET/close
Integration complexity	Low for serial; port name is platform-specific	Low for the model; HTTP parsing step missing
Reusability	Script portable wherever MATLAB runs; edit the port name	VI reusable for simulation and training; needs LabVIEW
Limitations here	No reconnection; no logging to file	Simulated values only; no soil or pump indicators

- Irrigation. The pump load is a DC motor, soil values are raw and uncalibrated, and hysteresis transitions were not captured.
- Duration. No long-duration or outdoor test was per-formed.

2. Future Work

Each item below is proposed and not implemented. (i) Add a calibrated current sensor, for example the INA219 already identified in the project's bill of materials, and measure PV power and energy. (ii) Compare the tracking panel against an identical fixed panel following the pro-tocol of Section X, including servo energy [2]. (iii) Log frames, including the servo angle of both axes and the tracker state, to file with a frame counter and timestamps.

(iv) Wire the LabVIEW HTTP response into parsed numeric indicators, and let MATLAB consume the same endpoint [47]. (v) Make the bridge robust with port re-opening and stale-data flags. (vi) Normalise the quadrant error by the channel sum to reduce sensitivity to satu-ration and overall brightness, and calibrate per-channel offsets. (vii) Improve the mechanical design and evaluate under varying weather. (viii) Replace the DC-motor load with a real pump and a calibrated soil sensor.

XI. CONCLUSION

This paper described a low-cost Arduino Uno proto-type that combines four-LDR dual-axis sensing, a panel-voltage channel, and threshold-based pump switching, and two monitoring clients built on its nine-field serial frame. The MATLAB dashboard reads the serial stream directly and displays real LDR, voltage, soil, and pump data; the LabVIEW VI provides a virtual-sun model, for which we showed that the quadrant errors are $-12 \Delta\phi$ and $+12 \Delta\theta$, and an HTTP client that polls a local Flask endpoint. A captured eight-point window showed the ex-pected sign of the errors for a bright top-right quadrant, with the TR channel near ADC saturation, and a dry soilreading that kept the pump ON.

What has been demonstrated is the acquisition and vi-sualisation chain on real hardware and the logic of the tracking model in simulation. Closed-loop tracking con-vergence, energy gain, panel power, endpoint latency,

and simultaneous operation of both clients have not been measured and are left for the protocol of Section X. We make no claim of improved efficiency.

REFERENCES

1. Mousazadeh, H., Keyhani, A., Javadi, A., Mobli, H., Abrinia, K., & Sharifi, A. (2009). A review of principle and sun-tracking methods for maximizing solar systems output. *Renewable and Sustainable Energy Reviews*, 13(8), 1800–1818. <https://doi.org/10.1016/j.rser.2009.01.022>
2. Lazariou, G.C., Longo, M., Roscia, M., & Pagano, M. (2015). Comparative analysis of fixed and sun tracking low power PV systems considering energy consumption. *Energy Conversion and Management*, 92, 143–148. <https://doi.org/10.1016/j.enconman.2014.12.046>
3. Eke, R. & S, entürk, A. (2012). Performance comparison of a double-axis sun tracking versus fixed PV system. *Solar Energy*, 86(9), 2665–2672. <https://doi.org/10.1016/j.solener.2012.06.006>
4. Sungur, C. (2009). Multi-axes sun-tracking system with PLC control for photovoltaic panels in Turkey. *Renewable Energy*, 34(4), 1119–1125. <https://doi.org/10.1016/j.renene.2008.06.020>
5. Wang, J-M. & Lu, C-L. (2013). Design and implementation of a sun tracker with a dual-axis single motor for an optical sensor-based photovoltaic system. *Sensors*, 13(3), 3157–3168. <https://doi.org/10.3390/s130303157>
6. Chin, CS., Babu, A., & McBride, W. (2011). Design, model-ing and testing of a standalone single axis active solar tracker using MATLAB/Simulink. *Renewable Energy*, 36(11), 30753090<https://doi.org/10.1016/j.renene.2011.03.026>
7. Kaur, T., Mahajan, S., Verma, S., Priyanka, & Gambhir, J. (2016). Arduino based low cost active dual axis solar tracker. In 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), pp. 1–5. <https://doi.org/10.1109/ICPEICES.2016.785339>

8. Zolkapli, M., Al-Junid, SAM., Othman, Z., Manut, A., & Mohd Zulkifli, MA. (2013). High-efficiency dual-axis so-lar tracking developement using Arduino. In 2013 Interna-tional Conference on Technology, Informatics, Management, Engineering and Environment (TIME-E), pp. 43–47. <https://doi.org/10.1109/TIME-E.2013.6611961>
9. Othman, N., Manan, MIA., Othman, Z., & Al Junid, SAM. (2013). Performance analysis of dual-axis solar tracking sys-tem. In 2013 IEEE International Conference on Control Sys-tem, Computing and Engineering (ICCSCE), pp. 370–375. <https://doi.org/10.1109/ICCSCE.2013.6719992>
10. Nguyen, BT. & Ho, H-XT. (2020). Design, implementation and performance analysis of a dual axis solar tracking system. *Advances in Science, Technology and Engineering Systems Journal*, 5(3), 41–45. <https://doi.org/10.25046/aj050306>
11. Chenchireddy, K., Mulla, GB., Jagan, V., Sultana, W., Sydu, SA., & Giddalur, E. (2025). Solar tracker using Arduino microcontroller and light dependent resistor. *International Jour-nal of Power Electronics and Drive Systems*, 16(1), 70–75. <https://doi.org/10.11591/ijpeds.v16.i1.pp70-75>
12. Bajpai, A., Tolani, M., & Balodi, A. (2023). Performance analysis of modified dual axis solar tracking system using LDR's. In 2023 IEEE 5th International Conference on Advances in Electronics, Computers and Communications (ICAEC), pp. 1–5. <https://doi.org/10.1109/ICAEC59324.2023.10560246>
13. International Electrotechnical Commission (2021). Photo-voltaic system performance – Part 1: Monitoring (IEC 61724-1:2021). IEC.
14. Ahmed, OA., Sayed, HK., Jalal, KA., Mahmood, DY., & Habeeb, WH. (2019). Design and implementation of an indoor solar emulator based low-cost autonomous data logger for PV system monitoring. *International Journal of Power Electronics and Drive Systems*, 10(3), 1645–1654. <https://doi.org/10.11591/ijpeds.v10.i3.pp1645-1654>
15. Muñoz, JV., Nieto, LM., Canalejo, JF., Montes-Romero, J., Gonzalez-Rodriguez, AG., & Gulkowski, S. (2024). Novel indoor educational I-V tracer for photovoltaic modules. *Electronics*, 13(24), 4932. <https://doi.org/10.3390/electronics13244932>
16. Rashid, F. & Nour, MI. (2025). A low-cost prototype I-V and P-V curve tracer for PV modules. In *Integrating Sus-tainability to Education, Business, and Environmental Energy Solutions* (pp. 141–149). Springer. https://doi.org/10.1007/978-3-031-90046-4_14
17. Oladimeji, I., Adediji, YB., Akintola, JB., Afolayan, MA., Ogunbiyi, O., Ibrahim, SM., & Olayinka, SZ. (2020). Design and construction of an Arduino-based solar power parameter-measuring system with data logger. *Arid Zone Journal of Engi-neering, Technology and Environment*, 16(2), 255–268.
18. Matijević, M. & Cvjetković, V. (2016). Overview of architec-tures with Arduino boards as building blocks for data acquisition and control systems. In 13th International Conference on Re-mote Engineering and Virtual Instrumentation (REV), pp. 56–63. <https://doi.org/10.1109/REV.2016.7444440>
19. Nichols, D. (2017). Arduino-based data acquisition into Excel, LabVIEW, and MATLAB. *The Physics Teacher*, 55(4), 226–227. <https://doi.org/10.1119/1.4978720>
20. Allamehzadeh, H. (2016). Solar energy overview and maximizing power output of a solar array using sun trackers. In 2016 IEEE Conference on Technologies for Sustainability (SusTech), pp. 14–19. <https://doi.org/10.1109/SusTech.2016.7897136>
21. Baouche, FZ., Abderezzak, B., Ladmi, A., Arbaoui, K., Suci, G., Mihaltan, TC., Raboaca, MS., Hudis,teanu, SV., & T, urcanu, FE. (2022). Design and simulation of a solar tracking system for PV. *Applied Sciences*, 12(19), 9682. <https://doi.org/10.3390/app12199682>
22. Aquino Larico, ER. (2021). Intelligent dual-axis solar tracking system in the Peruvian highlands: Implementation and performance. In 2021 IEEE Engineering International Re-search Conference (EIRCON), pp. 1–4.

- <https://doi.org/10.1109/EIRCON52903.2021.9613436>
23. Otieno, OR. & Abdulhamid, M. (2021). On the design of solar energy tracker system. *Journal of Siberian Federal University. Engineering & Technologies*, 14(6), 689–702. <https://doi.org/10.17516/1999-494X-0343>
 24. Román-Raya, J., Ruiz-García, I., Escobedo, P., Palma, AJ., Guirado, D., & Carvajal, MA. (2020). Light-dependent resistors as dosimetric sensors in radiotherapy. *Sensors*, 20(6), 1568. <https://doi.org/10.3390/s20061568>
 25. Hwang, Y., Minami, Y., & Ishikawa, M. (2018). Virtual torque sensor for low-cost RC servo motors based on dynamic system identification utilizing parametric constraints. *Sensors*, 18(11), 3856. <https://doi.org/10.3390/s18113856>
 26. Blaifi, S., Moulahoum, S., Colak, I., & Merrouche, W. (2018). Monitoring and enhanced dynamic modeling of battery by genetic algorithm using LabVIEW applied in photovoltaic system. *Electrical Engineering*, 100(2), 1021–1038. <https://doi.org/10.1007/s00202-017-0567-6>
 27. Srividya, P. & Sankonatti, J. (2018). Fault monitoring system for photovoltaic modules in solar panels using LabVIEW. In 2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE), pp. 23662369. <https://doi.org/10.1109/ICRIEECE44171.2018.9009282>
 28. Mezouari, A., Mateur, K., Alareqi, M., Hlou, L., & Elgouri, R. (2017). Development of an integrated data-acquisition system for photovoltaic blocks mutualization monitoring using LabVIEW. In 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), pp. 14. <https://doi.org/10.1109/WITS.2017.7934645>
 29. Ishafit, Indratno, TK., & Prabowo, YD. (2020). Arduino and LabVIEW-based remote data acquisition system for magnetic field of coils experiments. *Physics Education*, 55(2), 025003. <https://doi.org/10.1088/1361-6552/ab5ed6>
 30. Eteiba, MB., El Shenawy, ET., Shazly, JH., & Hafez, AZ. (2013). A photovoltaic (cell, module, array) simulation and monitoring model using MATLAB/GUI interface. *International Journal of Computer Applications*, 69(6), 14–28. <https://doi.org/10.5120/11845-7579>
 31. Chand, K. & Khosla, A. (2022). MATLAB-based real-time data acquisition tool for multimodal biofeedback and Arduino-based instruments. *Journal of Information Technology Research*, 15(1), 1–20. <https://doi.org/10.4018/jitr.299922>
 32. Allafi, I. & Iqbal, T. (2017). Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring. In 2017 IEEE Electrical Power and Energy Conference (EPEC), pp. 1–5. <https://doi.org/10.1109/EPEC.2017.8286184>
 33. Hamied, A., Mellit, A., Benghanem, M., & Boubaker, S. (2023). IoT-based low-cost photovoltaic monitoring for a greenhouse farm in an arid region. *Energies*, 16(9), 3860. <https://doi.org/10.3390/en16093860>
 34. Z., Yang, C., & Zhang, Y. (2014). A design of cross-terminal web system based on JSON and REST. In 2014 IEEE 5th International Conference on Software Engineering and Service Science (ICSESS), pp. 904–907. <https://doi.org/10.1109/ICSESS.2014.6933711>
 35. Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259). IETF. <https://www.rfc-editor.org/rfc/rfc8259>
 36. Fielding, R., Nottingham, M., & Reschke, J. (2022). HTTP Semantics (RFC 9110). IETF. <https://www.rfc-editor.org/rfc/rfc9110>
 37. Adla, S., Rai, NK., Karumanchi, SH., Tripathi, S., Disse, M., & Pande, S. (2020). Laboratory calibration and performance evaluation of low-cost capacitive and very low-cost resistive soil moisture sensors. *Sensors*, 20(2), 363. <https://doi.org/10.3390/s20020363>
 38. Olujimi, A., Aaron, I., Adebayo, O., Afolarin, A., & Jonathan, E. (2022). Smart solar powered irrigation system. *Journal Européen des Systèmes Automatisés*, 55(4), 535–540. <https://doi.org/10.18280/jesa.550413>
 39. Arduino S.r.l. (2024). Arduino UNO R3 – User Manual (SKU A000066), Rev. 3. Retrieved October 16, 2026, from <https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>

40. Arduino (n.d.). Servo library reference. Arduino Reference. Retrieved October 16, 2026, from <https://reference.arduino.cc/reference/en/libraries/servo/>.
41. Pallets (n.d.). Quickstart – Flask documentation (3.1.x). Retrieved October 16, 2026, from <https://flask.palletsprojects.com/en/stable/quickstart/>.
42. Liechti, C. (n.d.). pySerial documentation, version 3.5. Retrieved October 16, 2026, from <https://pyserial.readthedocs.io/en/latest/>.
43. MathWorks (n.d.). serialport – Connection to serial port. MATLAB Documentation, R2026b. Retrieved October 16, 2026, from <https://www.mathworks.com/help/matlab/ref/serialport.html>.
44. MathWorks (n.d.). tiledlayout – Create tiled chart layout for displaying subplots. MATLAB Documentation, R2026b. Retrieved October 16, 2026, from <https://www.mathworks.com/help/matlab/ref/tiledlayout.html>.
45. National Instruments (2026). GET VI. LabVIEW API Reference (updated July 27, 2026). Retrieved October 16, 2026, from <https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/vi-lib/httpclient/get-vi.html>.
46. National Instruments (2026). HTTP Client. LabVIEW API Reference (updated July 27, 2026). Retrieved October 16, 2026, from https://www.ni.com/docs/en-US/bundle/labview/page/lvcomm/http_client.html.
47. MathWorks (n.d.). webread – Read content from RESTful web service. MATLAB Documentation, R2026b. Retrieved October 16, 2026, from <https://www.mathworks.com/help/matlab/ref/webread.html>.