

ELEE 4230/6230E: Sensors and Transducers
Course Project - Summer 2026
2-Axis Light Tracker
Nikhil Mahadevan

Table of Contents

ELEE 4230/6230E: Course Project……1
0. A Tour of the 2-Axis Light Tracker……...2
1. Problem Definition and Environmental Project Constraints……..3
2. Sensor Selection and Design Decisions…….5
3. RAW Sensor Dynamics and Signal Processing……7
4. Capacitive Sensor (User Input System).....11
5. Other Sensors Used……..15
6. Digital Logic + Algorithms……..17
7. Final Design and Experimental Results……...19
8. Failure Analysis……21
9. Physical Implementation……..22
10. Course Reflection……...23
11. References……..24
Appendix……..25

0. A Tour the 2-Axis Light Tracker

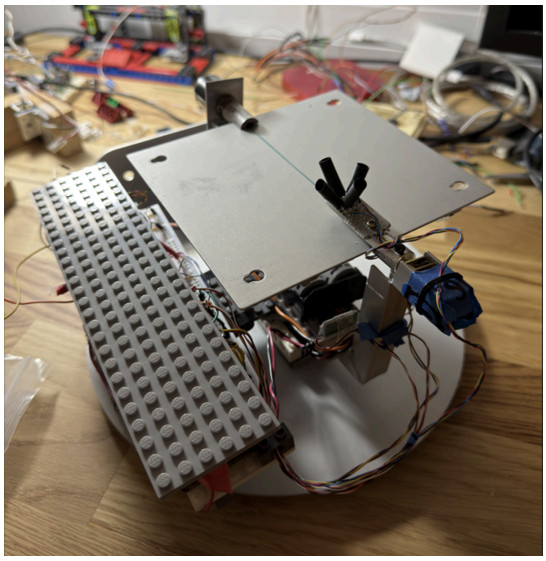


Fig 1. Picture of the system

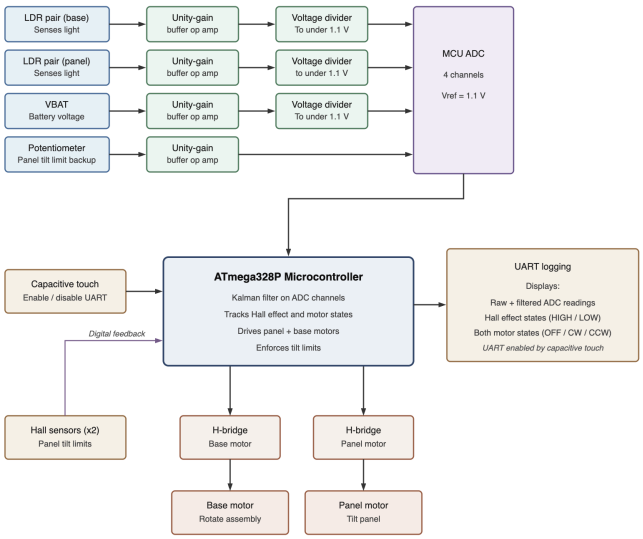


Fig 2. Block diagram of the system

There are two main inputs: LDRs, which measure light intensity, and VBAT, which measures battery voltage. Since the whole system is powered from the same battery, measuring VBAT is vital as it gives all other ADC channels a baseline to be read against. Both the LDR and VBAT voltages are first buffered through unity-gain op-amps to decouple current draw and prevent loading errors, and then scaled down to the ADC's 1.1 V reference before being handed off for processing. To keep the panel from overrotating, tilt limits are enforced through digital Hall effect sensors: magnets mounted on the panel pull the sensor output low as they approach. Software flags then use that signal to turn off the panel motor rotation when they go low. As a backup in case the Hall effect sensors fail, a mechanical potentiometer provides a redundant tilt limit. It rotates along with the panel and its output voltage changes with the angle. Like the LDR and VBAT, the pot's output runs through its own unity-gain buffer before being read on an ADC channel. Taken as a whole, the system works like this: when a bright light source is detected, the sensors send their readings to the microcontroller, which processes them and drives two motors - a base motor that rotates the whole assembly toward the source, and a panel motor that tilts it to the correct angle while staying within the tilt limits at all times.

1. Problem Definition and Environmental Project Constraints

Problem Definition

Static solar panels lose efficiency because they cannot adjust to the sun's movement. Since their angle is fixed, the panel is not always pointed in the direction of the sun which reduces its power output. The challenge is to design a system that can accurately locate the brightest light source and reliably reorient the panel towards it.

Environment Being Measured

Testing was conducted indoors under climate-controlled conditions (~68°F, normal humidity) using a flashlight to simulate light input (see Fig. 3). An environmental checklist was completed for all sensor subsystems (LDR array, Hall effect sensors, and potentiometer) and confirmed minimal risk from factors like temperature, humidity, RF, or corrosion (see Appendix 1.1). Primary safety concerns were strictly mechanical, presenting two main challenges: structural strain on the aluminum frame and the risk of over-rotation. To ensure structural integrity, the aluminum frame was reinforced at key stress points, while over-rotation was prevented using tilt limits provided by the Hall effect sensors and potentiometer (see Fig. 1 for Hall effect and potentiometer tilt limits).

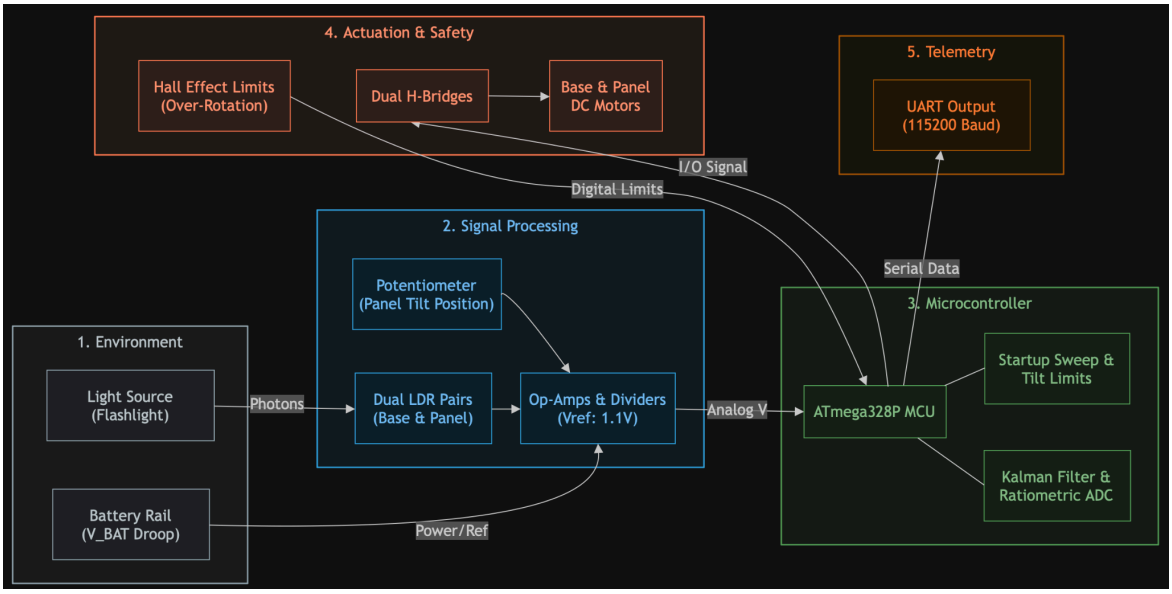


Fig. 3 System architecture

Constraints

- RAW sensor requirement: The 4 LDRs serve as the RAW sensor. A photoresistor's resistance-vs-lux curve is nonlinear so active signal processing and linearization is needed before reaching the ADC.
- Pre-packaged sensors: The 2 Hall effect switches satisfy the pre-packaged sensor requirement and are used for tilt-limit detection.

- Physics-branch redundancy: The Hall effect switches (magnetic sensing) and the potentiometer (mechanical, resistive-wiper sensing) both detect tilt limits, satisfying the redundancy across two different branches of physics.
- Capacitive user input: A homemade 1 MΩ RC capacitive touch sensor provides user input that toggles system logging.
- Battery-only power: Two Li-ion packs power the entire system with no wall supply - one powers the microcontroller and sensors while the other powers only the motors.
- Calibration to a known value: Each sensor is calibrated against a known reference - the potentiometer via a full-sweep startup calibration to determine tilt limits, the Hall effect switches against a known trip distance, and the LDR array against battery voltage rather than a fixed reference.

The table below summarizes the key constraints pulled from datasheets and characterization:

Parameter	**Constraint / Range**	**Source**
LDR resistance	~1kΩ (bright light) to 200 kΩ (dark)	Own characterization
LDR unit-to-unit variation	±50% or more	LDR datasheet
LDR spectral sensitivity	400–600 nm, peak ~520 nm	LDR datasheet
Hall effect operate point (B_{op})	9–26 Gauss (typ. 17)	Hall effect datasheet
Hall effect hysteresis	5 Gauss (typ.)	Hall effect datasheet
Hall effect supply voltage	1.8–5.5 V	Hall effect datasheet
Hall effect trip distance	1.1 cm	Own characterization
Potentiometer resistance	200 Ω–100 kΩ (standard line)	Own characterization
Potentiometer linearity	±0.25%	Pot datasheet
UART telemetry	Used at 115200 baud	9.216 MHz crystal
Total project cost	$249.55	Final BOM

Table 1 Key constraints

Modeling Limitation

The potentiometer's tilt limits are calibrated once at startup by sweeping the panel fully left and right, allowing the microcontroller to sample the bounding ADC values. Because these readings scale with battery voltage, this single calibration assumes short operating sessions or a sufficiently slow battery discharge rate. Mid-run recalibration is not performed. Consequently, during extended operation on a draining battery, the fixed potentiometer limits will gradually drift out of calibration even when there is battery droop calibration in other parts of the system like the LDR tracking logic.

2. Sensor Selection and Design Decisions

Sensor Role & Option	Metrics	Pros	Cons
Light Sensing: LDR Array(Selected)	Power: Minimal (< 1 mA) Interface: Resistor Divider, Op-Amp, and ADC	Low cost; simple resistance output; easy array layout.	Non-linear response curve; requires ADC compensation.
Light Sensing: Photodiodes	Power: Minimal (< 1 mA) Interface: Op-Amp	Fast response time; highly linear output.	Adds extra op-amp circuit complexity and board space.
Angle Sensing: 10-Turn Pot(Selected)	Power: Low (< 1 mA) Interface: Op-Amp + ADC	True absolute angle at startup; zero software drift; simple ADC reading.	Mechanical wiper wear over long-term use.
Angle Sensing: IMU (6-Axis)	Power: Medium (~5-10 mA) Interface: I2C/SPI	Contactless; measures acceleration and orientation directly.	Software integration drift over time; complex bus communication.
Limit Sensing: Hall Effect(Selected)	Power: Minimal (< 1 mA) Interface: GPIO pins	Contactless operation; zero moving parts; simple digital output.	Requires a magnet mounted on the moving frame.
Limit Sensing: Microswitches	Power: Zero Interface: GPIO pins	No magnets required; simple switch logic.	Mechanical contacts wear over time; prone to frame flex issues.

Table 2 Sensor comparison table

**Note: Full component datasheets are in Appendix 2.1 - 2.3*

Final Sensor Selection

- LDRs (RAW): Chosen for their simple resistance-based output, non-linear response (to fit project requirement), low cost, easy to use, and determines brightest light source.
- Hall Effect Sensors (Pre-Packaged): Chosen for contactless operation, simple mounting, zero moving parts, and direct digital MCU control to act as tilt limit switches.
- 10-Turn Potentiometer (Physics Requirement): Chosen for its simplicity, direct mechanical coupling, simple electrical interfacing, and low power draw to provide angular position feedback.
- Capacitive Touch Sensor: Homemade sensor chosen to satisfy the project requirement.

Design Decision Blocks

Decision 1: Light Detection Subsystem

- What was chosen: LDR array (Light-Dependent Resistors).
- Alternative considered: Cameras, photodiodes, phototransistors, and thermopiles.
- Why the alternative would not work as well: Cameras, photodiodes, and phototransistors add processing complexity and extra circuitry. Thermopiles measure heat/infrared rather than visible light. LDRs provide the simplest resistance-based output needed for a basic differential array.

Decision 2: Tilt Limit Detection

- What was chosen: Hall effect magnetic sensors.
- Alternative considered: Mechanical microswitches and optical interrupters.
- Why the alternative would not work as well: Mechanical limit switches wear down over time and require precise physical contact, while optical interrupters add wiring complexity. Hall effect sensors offer contactless operation with zero moving parts and feed directly into digital MCU pins.

Decision 3: Tilt Angle Feedback

- What was chosen: 10-turn mechanical potentiometer.
- Alternative considered: Inertial Measurement Units (IMUs).
- Why the alternative would not work as well: IMUs introduce software complexity, cost, reduced battery life (power hungry) and integration drift over time. A 10-turn potentiometer offers direct mechanical coupling, zero software drift, simple ADC interfacing, and minimal power draw.

3. RAW Sensor Dynamics and Signal Processing

Raw Sensor Physics and Modeling

A Light Dependent Resistor (LDR) is a photoresistor whose resistance decreases nonlinearly as light intensity increases. In a basic single-LDR voltage divider, the output voltage is given by:

$$V_{out} = V_{in} \cdot R_{fixed} / (R_{fixed} + R_{LDR})$$

Because the LDR varies nonlinearly with lux, it produces a nonlinear output voltage curve. Since I want to sense light direction, using a four-element differential divider network (R_1 - LDR_1 - LDR_2 - R_2) is helpful. Adding fixed resistors R_1 and R_2, where R_1 equals R_2, limits current draw under bright light conditions when the LDR's resistance drops significantly. Different light intensities can cause the voltage between the two LDRs to change.

To model the response, a constant-brightness light source was swept through five discrete angles (0°, 45°, 90°, 135°, and 180°), and the voltage output was measured by an ADC. The results show a linear relationship between incident light angle (Angle) and the microcontroller's ADC reading (ADC_{count}) and is modeled by the experimental transfer function:

$$ADC_{count} = -4.59 \times Angle + 903.40$$

The negative slope reflects the decreasing midpoint voltage (and ADC value) as the light sweeps across the sensors from 0° to 180°.

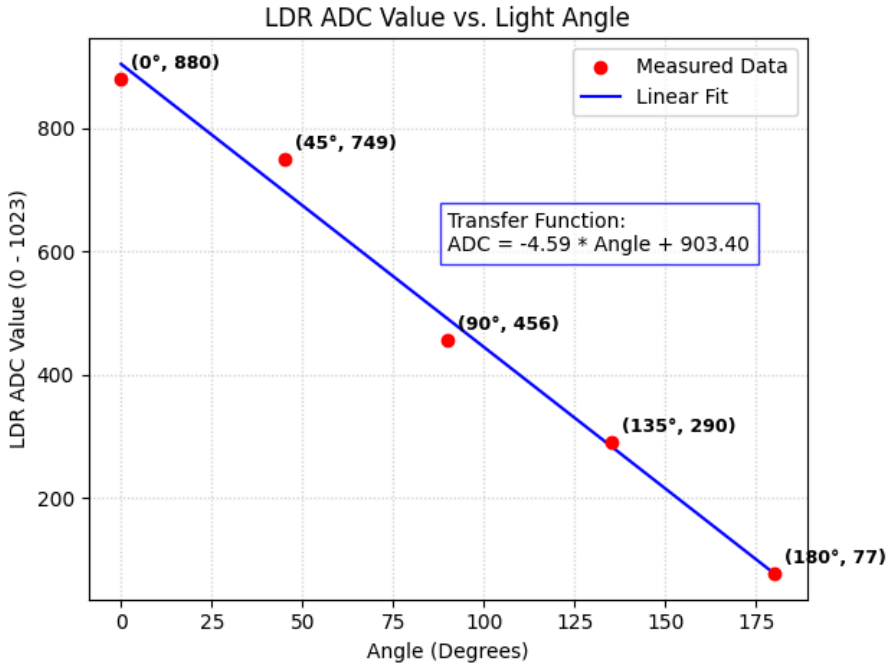


Fig. 4 Light angle vs. LDR ADC value function

While the dual-LDR network provides an effective linear model over the tested range, a few physical limitations exist:

- Angle Precision & Calibration Noise: Test angles were visually approximated rather than with a protractor, introducing alignment errors.
- Ambient Light Sensitivity: The model assumes a single dominant point source. Other light sources such as a laptop screen or room lighting can shift the DC offset.
- Non-identical Sensor Components: Manufacturing variances in resistance-versus-lux profiles between the LDRs can cause symmetry offsets
- Physical Placement: The system relies on all LDRs being mounted at identical angles with matching light-filtering tubes. Any physical misalignment or difference between the assemblies can cause a measurement offset.

Signal Processing Path ("Signal Processing Mountain")

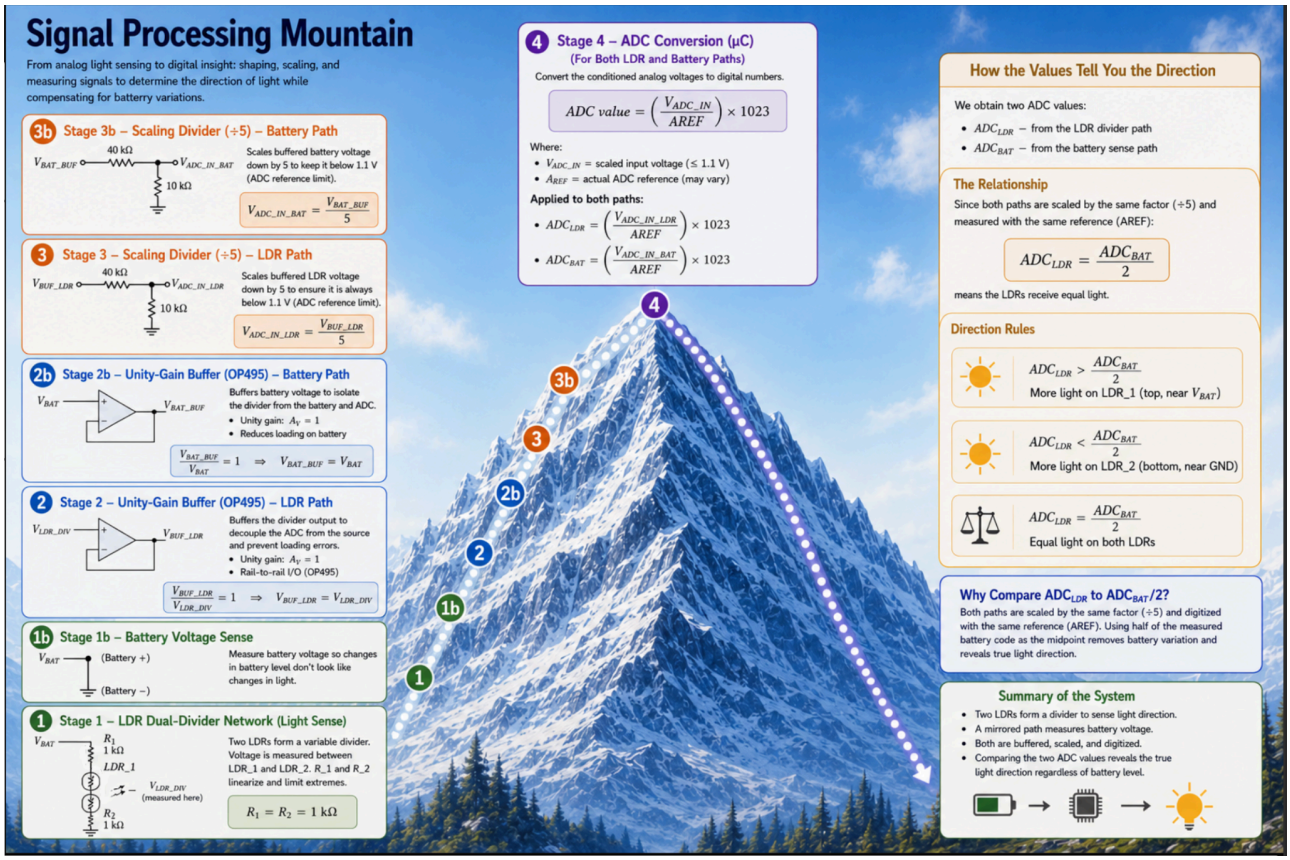


Fig 5. Signal processing mountain

The signal processing path conditions raw sensor voltages prior to digital quantization:

1. Fixed 1 kΩ resistors used to limit current draw
2. Op-amp buffers used to decouple current and prevent loading errors.
3. Resistor divider scales used to scale down the voltage by a factor of five to ensure the voltage is below the ADC reference (1.1 V)

4. Battery supply voltage (VBAT) goes through an identical buffer op-amp and voltage divider by a factor of five
5. Software evaluates sensor inputs relative to the battery reading accounting for battery discharge

Final Implementation (Circuit and Power)

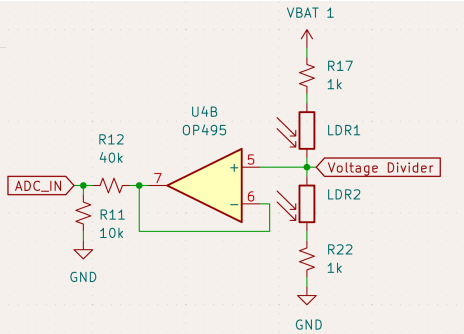


Fig 6. LDR pair ADC measuring circuit

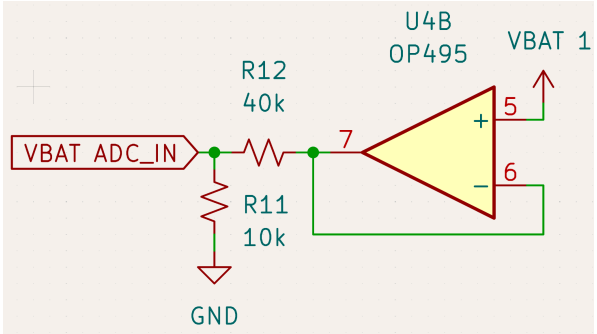


Fig 7. VBAT ADC measuring circuit

The LDR pair forms a four-element divider (1 kΩ - LDR_1 - LDR_2 - 1 kΩ) connected directly across VBAT and ground. The midpoint of the voltage divider goes through a unity gain buffer opamp that decouples the sensor from the ADC and is divided by a factor of five using another resistor divider to get the sensor voltage within the input voltage of the ADC (see Fig. 6).

Since the LDRs are powered directly by the battery, their output voltage will drop as the battery drains. To maintain accurate calibration, the battery voltage must also be measured to give a baseline. Thus, VBAT also goes through an identical unity gain buffer opamp and 40 kΩ/10 kΩ scaling divider which feeds into one of the microcontroller's ADC channels (see Fig. 7).

The software compares the LDR ADC readings against the battery reference ADC value. Here is one example of how it is used:

```
if (panel_filt_val > (3 * battery_adc) / 4)
```

Fig. 8 Comparing panel LDR ADC to the battery reading

**Note: Full source code is in Appendix 7.2*

Evaluating sensor readings as a dynamic fraction of the current battery voltage ensures motor decision thresholds stay calibrated even as VBAT changes over time.

Filtering and Kalman Implementation

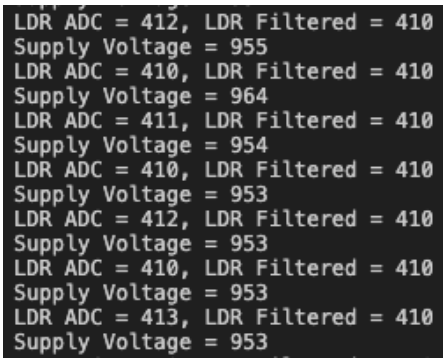


Fig. 9 LDR ADC values before and after filtering

A discrete 1D Kalman Filter using scaled integer arithmetic is implemented to smooth noisy ADC sensor readings (LDRs, battery level, and potentiometer position) in real time. The filter minimizes measurement noise without introducing significant lag by calculating a Kalman Gain (K) at each iteration. This gain dynamically weights the prior state estimate against the incoming raw ADC sample.

Internal variables are scaled by 65,536, allowing the ATmega328P to compute precise fractional decimals using fast integer math. Because the ATmega328P lacks a dedicated Hardware Floating-Point Unit (FPU), using integer arithmetic avoids the heavy CPU overhead of software floating-point emulation, reducing clock cycles per execution and boosting execution speed (see Appendix 3.1 for the Kalman filter code).

To implement the filter without floating-point math, state variables and parameters are tracked using a custom C structure (struct KalmanFilter), with internal values scaled up by a factor of 65,536 (2^{16}).

Parameter Definitions:

- x (State Estimate): Current filtered ADC value, initialized to 33,554,432 (512 x 65,536) to start at the midpoint of the 10-bit ADC range (0 - 1023).
- p (Estimate Covariance): Initial uncertainty in the estimate, set to 65,536 (1.0 descaled).
- q (Process Noise Covariance): Expected physical change of the sensor over time. LDRs and the potentiometer use q = 13,107 (≈ 0.2) for responsive tracking, while the battery supply uses a low q = 66 (≈ 0.001) to reflect a steady DC line.
- r (Measurement Noise Covariance): Electrical ADC noise tolerance, set to 655,360 (10.0 descaled) across all channels to eliminate jitter.

**Note: Picture of struct code is in Appendix 3.2*

4. Capacitive Sensor (User Input System)

Physics and Construction

The capacitive sensor is constructed using a 1 MΩ resistor connected in series with a piece of unshielded, coiled metal wire between two microcontroller GPIO pins, a charge source pin and a high-impedance sense node.



Fig. 10 Homemade capacitive touch sensor

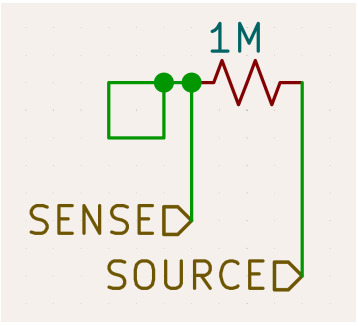


Fig. 11 Schematic of capacitive touch sensor

The sense node is a high impedance input charged through R by the source pin. Total node capacitance C is the sum of the wire's stray capacitance C_0 and, when touched, the capacitance a finger couples into the loop, ΔC. The node charges as a standard first-order RC circuit:

$$V_C(t) = V_{CC} \cdot (1 - e^{(-t/RC)})$$

The code measures the time for V_C to cross the input-high threshold V_{IH} (≈ 0.6 ·V_{CC} for the ATmega328P). Solving for that crossing time:

$$t_{th} = RC \cdot ln[\ V_{CC} / (V_{CC} - V_{IH})\ ]$$

A timer-based counter increments in proportion to t_{th}. Rather than converting this count to absolute units of time, the raw dimensionless counter value (N) is monitored directly to determine touch state.

Using the microcontroller's 9.216 MHz system clock (T_{clk} ≈ 108.5 ns), each polling loop tick corresponds to approximately 434 ns (4 clock cycles). Evaluating the RC threshold equation at V_{IH} ≈ 0.6 · V_{CC} yields: t_{th} ≈ 0.9163 · R · C

For R = 1 MΩ, the dimensionless counter values (N) translate to the following physical capacitance estimates calculated at the sensor node:

- Baseline Parasitic Capacitance (C_0 - Untouched): At the resting baseline count of N = 29, the stray wire and pin capacitance is calculated as: C_0 ≈ 13.7 pF

- Total Node Capacitance (C_{total} - Touched): When touched, the total capacitance present at the sense node increases, causing the count to rise to N ≈ 500: $C_{total} \approx 237.0$ pF
- Measured Delta Capacitance (ΔC): The touch event introduces an additional coupled capacitance ($\Delta C = C_{total} - C_0$) at the sensing node of: $\Delta C \approx 223.3$ pF

This measured increase in node capacitance significantly exceeds the baseline stray capacitance, providing a clear physical basis for the threshold-based touch detection mechanism.

Testing confirmed that bringing a finger close to the capacitive sensor without physical contact produces no measurable system response, verifying that the design operates strictly as a contact-based capacitive sensor rather than a proximity sensor. This behavior was verified by having the microcontroller send the sensor data via UART and observing the reported values. When a user makes direct physical contact with the wire, body capacitance couples into the node, increasing total node capacitance and extending the RC charging time constant. This causes the loop count to jump significantly above the baseline, registering a touch event.

Signal Processing and Implementation

The sensor converts physical node capacitance into a time-domain duration using an RC charge-cycle scheme. By driving a step-voltage high from the charge source pin (PB0), current flows through the 1 MΩ resistor to charge the receive node (PB4) toward VCC.
The rate of voltage rise is dictated by the total capacitance present at the node ($C = C_0 + \Delta C$). As capacitance increases during a touch event, the time constant ($\tau = R \cdot C$) extends, slowing the voltage rise. Once the voltage across PB4 crosses the microcontroller's digital input-high threshold ($V_{IH} \approx 0.6 \cdot VCC$), the pin transitions to a logical HIGH. This turns the physical capacitance change into a measurable time delay.

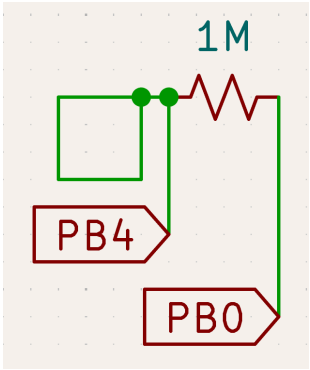


Fig. 12 Circuit of capacitive touch sensor

The measurement cycle relies on a software-polling execution sequence on the microcontroller:

1. The receive pin (PB4) is driven LOW as an output to completely drain residual charge from the node capacitance.
2. PB4 is immediately switched to an input and the internal pull-up resistor is disabled.

3. The source pin (PB0) is driven HIGH, starting the RC charging process.
4. A software while loop polls PINB every cycle, incrementing a counter variable (touch_time) until PB4 reads HIGH.

**Note: Capacitive touch sensor code is in Appendix 4.1*

Rather than using an ADC channel, the microcontroller reads the signal digitally via a standard GPIO pin. A software loop increments a counter (touch_time) for as long as the pin remains low. Once the pin reaches a logic-HIGH threshold and the while loop terminates, the accumulated count is evaluated against an experimentally determined software threshold (touch_time > 40). If the count exceeds 40, the system recognizes a valid touch event and executes capacitive_sensor().

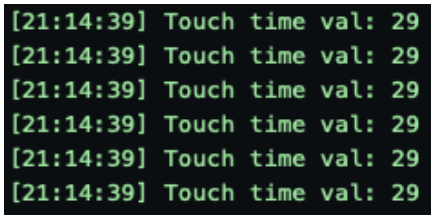


Fig. 13 Counter value with no touch

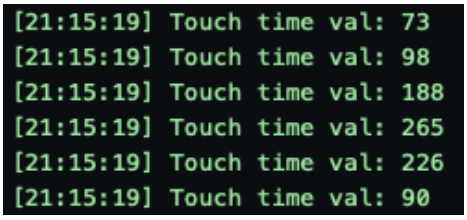


Fig. 14 Counter value with touch

From the data, the transfer function mapping the raw loop count (N) to the touch state is:

$$\textit{Sensor ON if } N > 29$$
$$\textit{Sensor OFF otherwise } (N \leq 29)$$

While the physical baseline measurement shows that the sensor is ON when $N > 29$ and OFF when $N \leq 29$, setting the active decision boundary to 40 adds a protective hysteresis buffer. This offset prevents false triggers caused by electrical fluctuations, noise, or jitter.

System Integration

The capacitive touch sensor acts as a UART logging switch for the light tracking system. It determines whether the system runs with active serial logging or in silent mode at boot:

- Default Mode (Untouched at Boot): If the sensor is not touched during startup, UART logging stays enabled. The microcontroller actively builds state messages and sends serial data such as sensor readings, motor status, and limit switch triggers to the computer.
- Silent Mode (Touched at Boot): If the sensor is held during power-on, UART logging is turned off. The system shuts down serial communication hardware and skips building/ sending log strings, freeing up processing time for the main control loop.

To track if UART is enabled, there is a software global flag (UART_enabled).

```
54    uint8_t UART_enabled = 1;
```

Fig. 15 Initialization of UART_enabled global flag

The capacitive sensor interacts with the rest of the system as a global control gate for diagnostic output. Rather than overriding the motor control or light-tracking algorithms directly, it modifies how every other component reports its status:

- Boot-Only Execution: The sensor logic executes exclusively during the startup sequence. By taking its reading on boot, it establishes the logging state before the main loop begins, ensuring zero CPU overhead or polling delay during active light tracking.
- Gating Diagnostic Output: Throughout the main loop, other sensors (such as the LDRs and Hall effect limit switches) routinely check a global logging flag before attempting to process or send diagnostic data.
- Bypassing Data Formatting: When silent mode is active, any logic block that handles status updates - like motor direction changes or Hall effect limit triggers - skips string construction and buffer management entirely. The tracking logic and limit checks continue running normally, but all background communication and message formatting tasks are bypassed.

**Note: Capacitive sensor function is in Appendix 4.2*

Limits and Performance

The main limitation of this design is its susceptibility to environmental noise across different hardware setups, as the unshielded wire node can shift baseline capacitance. Furthermore, checking the sensor exclusively at boot means the logging state cannot be toggled dynamically without power-cycling the system. During testing, the sensor demonstrated excellent noise immunity to proximity, showing no count change when a finger hovered near the loop and confirming it operates strictly on direct physical contact. Baseline resting counts hovered reliably around 29 across multiple boots, whereas direct physical contact triggered a massive jump to approximately 500.

5. Other Sensors Used - Hall Effect

A sensor used in this system was the Hall effect sensor, which acts as a magnetic limit switch to detect mechanical boundary limits. It operates on the principle that a magnetic field alters the flow of electricity. When a magnet attached to the pivoting panel arm moves near the sensor, its magnetic field pushes electrons toward one side of the internal sensor material. This buildup of electrons on one side creates a small voltage difference that increases as the magnet gets closer.

The Hall effect sensor produces a digital output that switches between VCC (HIGH) and 0 V (LOW). Powered directly from the battery rail, it outputs 0 V when a magnet is detected and VCC when no magnet is present. The sensor's spatial resolution is defined by a single trip point at a physical distance of 1.1 cm, transitioning states cleanly as the magnet crosses this boundary (see Appendix 5.1 for falling-edge and Appendix 5.2 for rising-edge scope traces). Testing across supply voltages ranging from 2.7 V to 5.0 V confirmed that this 1.1 cm detection threshold remains fixed, proving that position accuracy is immune to battery voltage droop. Controlled trials yielded a 100% detection rate at distances of 1.1 cm or closer and 0% 1.2 cm and beyond, exhibiting negligible hysteresis. Furthermore, sweep testing with the magnet moved past the sensor at varying speeds yielded 100% detection reliability.

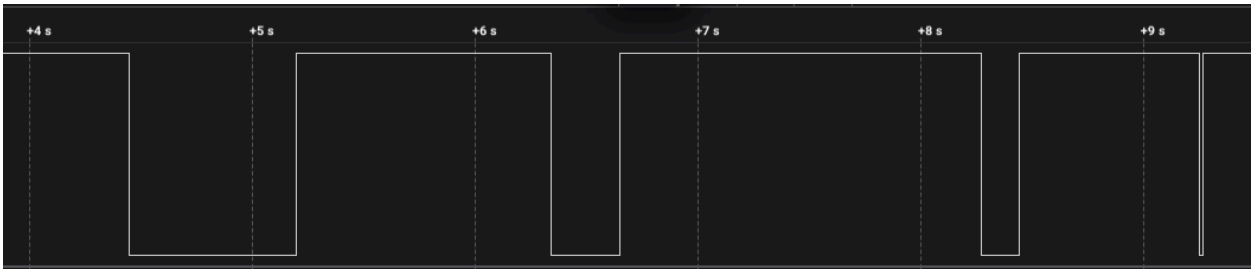


Fig. 16 I/O waveform at various speeds

In this system, two Hall effect sensors serve as tilt limits for the panel's range of motion. Magnets are mounted on either side of the panel, and the sensors were expected to drop their output signal LOW when a magnet reached a limit, signaling the microcontroller to stop the motors. During testing, the sensors detected the panel magnets reliably across all sweeps, and software state flags were implemented to ensure the panel does not go past the Hall effect sensors.
The internal processing may work by using a decoupling capacitor to smooth out any noise from the Hall effect's output voltage.

A limitation is that the sensor relies strictly on physical alignment and a short sensing distance. Since the sensor output is purely binary, if the magnet is shifted to more than 1.1 cm away at all times, the sensor will only output HIGH. As a result, the microcontroller receives no trigger signal and fails to stop the motor.

Other Sensors Used - Mechanical Potentiometer

A sensor used in this system was a mechanical potentiometer, which acts as a redundant tilt limit. It works by using an internal wiper arm that slides across a resistive track. As the wiper rotates, it changes the resistance, which creates a variable output voltage

The potentiometer produces a continuous analog voltage output spanning approximately 0 V to ~VCC across its full mechanical sweep. Its resolution is continuous, as the internal wiper slides along a smooth resistive element rather than discrete steps. And it is highly accurate because the voltage changes linearly with movement.

The potentiometer was expected to serve as a redundant tilt limit by outputting an analog voltage that changes as the panel tilts, signaling the microcontroller to stop the motors when a boundary threshold is reached. To account for battery voltage changes and mechanical shift, the system self-calibrates on boot by trusting the Hall effect sensors first: the panel sweeps to each Hall sensor limit, records the potentiometer's ADC values at both extremes, and sets those recorded limits (e.g. 0.55 V to 0.70 V at maximum VBAT) as the active threshold bounds. Because the output voltage remains low (under 1.1 V), the signal is routed through an OP495 unity gain buffer opamp directly into the microcontroller's ADC pin. During testing, this dynamic threshold approach worked as intended, reliably catching the physical limits and halting the motors if the primary Hall effect limit switches failed.

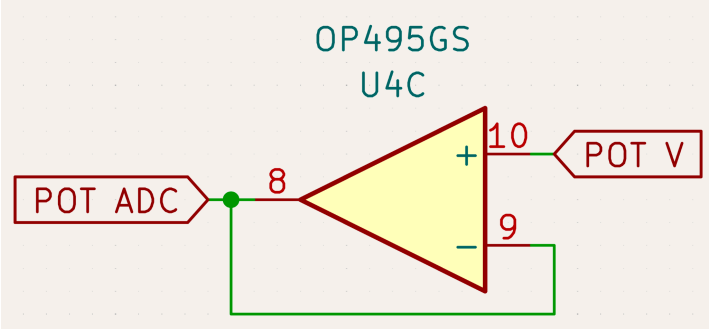


Fig. 17 Schematic of potentiometer's signal conditioning

A reasonable guess of the potentiometer's signal processing is there actually isn't any. It is a passive component that only relies on physical contact to divide voltage.

A key limitation of this design is that the potentiometer limits are only calibrated on boot. As the system runs on battery power over extended periods, battery voltage changes cause the potentiometer's output voltage range to shift. Because the limits are static after startup, this drift can cause the threshold checks to become inaccurate over time. A proposed software fix would be to implement an internal timer that tracks system runtime. After a set duration, the system would briefly pause operation, re-run the calibration sequence, and update the potentiometer's high and low ADC thresholds.

6. Digital Logic + Algorithms

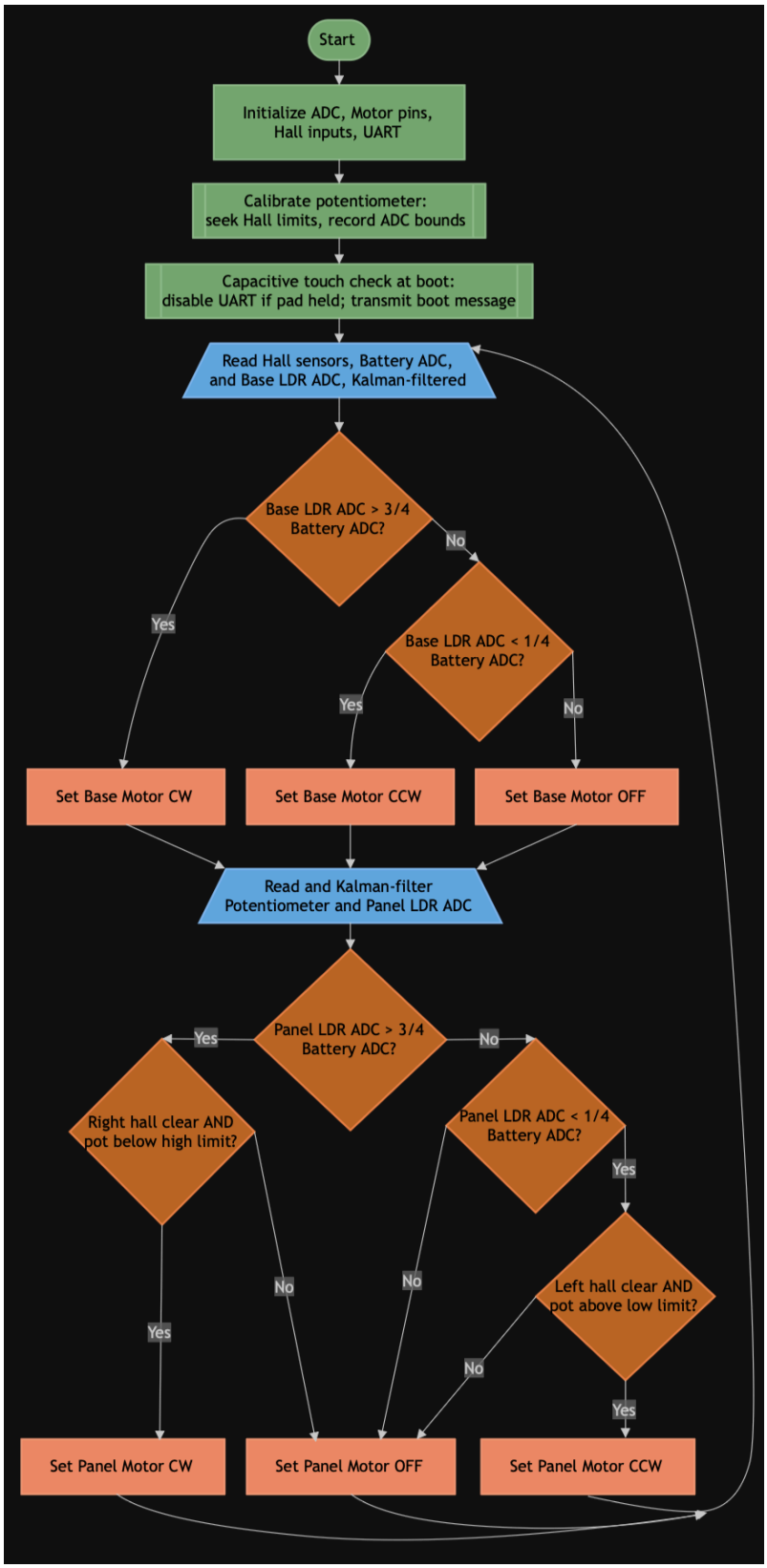


Fig. 18 Flowchart of system logic

Algorithm Descriptions

The sensor acquisition software provides the core control loop with digitized readings. To promote clean software architecture and code reuse, a single helper function encapsulates all analog reading logic (see Appendix 6.1 for ADC code). It accepts a channel index and applies a bitwise mask to safely swap multiplexer channel bits without corrupting peripheral configurations. The function then initiates conversion, polls until hardware completion, and returns a 10-bit digital result. For physical boundary monitoring, digital input routines evaluate the Hall-effect sensors using edge detection, storing previous pin states in memory so that system logging and control flags change state only when a physical limit is actively crossed (see Appendix 6.2 for Hall effect logic).

A fixed-point, 1-dimensional Kalman filter was implemented to reduce sensor noise and smooth out ADC readings before they reach the control logic (see Appendices 3.1 and 3.2 for Kalman filter algorithm). Raw sensor data fluctuates due to environmental light changes, electrical noise and ADC characteristics. While Kalman filters effectively clean up this noise, standard floating-point math can slow down an 8-bit microcontroller. To solve this, the filter scales internal variables by 65,536 so all state updates run using fast integer calculations instead of floating point numbers. Each ADC channel maintains its own filter variable with custom noise settings (q and r), delivering clean, stable data in real time without appreciably slowing down the main loop.

The "Go Beyond" algorithm continuously recalibrates the light-tracking thresholds using battery ADC readings (see Fig. 8 and Appendix 6.3 for battery-ratiometric tracking logic). This adjustment is necessary because the LDR sensor circuits run directly off the main battery rail and as the battery drains, the baseline voltage across the sensors change. Without dynamic adjustment, fixed threshold values might cause the system to misinterpret a discharging battery as a dark room and fail to operate correctly. To solve this, the algorithm calculates upper and lower ADC bounds for all LDRs on every loop proportional to the filtered battery reading.

Design Decision

The ATmega328P was selected as the primary microcontroller to learn bare-metal programming because its 8-bit architecture makes direct register manipulation, bitwise operations, and low-level byte structures easy to understand. Removing the DIP-package IC from an Arduino board onto a custom breadboard enabled complete hardware flexibility, including selecting a standalone 9.216 MHz crystal to achieve a theoretical 0% baud rate error for reliable UART communication across standard baud rates. While 32-bit platforms like the ESP32 or Raspberry Pi Pico which have superior computing power were considered, their complex register structures can make bare-metal programming difficult to learn.

7. Final Design and Experimental Results

Signal Conditioning and Kalman Filtering

To eliminate high-frequency electrical noise and sensor jitter without sacrificing execution speed, a 1D Kalman filter was implemented across all ADC channels including the LDR and battery voltage sense channels using fixed-point integer scaling.

The signal conditioning performance was previously verified in Fig. 9.

The filter successfully reduced peak-to-peak ADC measurement noise from 3 LSBs down to 0 LSBs at steady state while maintaining real-time execution. By dynamically weighting incoming raw ADC samples against prior state estimates across every channel, the filter stabilizes baseline LDR and supply voltage readings without introducing perceptible phase lag into the motor control loop. Applying this filter across all analog inputs satisfies the raw sensor signal conditioning requirement by delivering stable, noise-free data to both the directional decision logic and dynamic threshold calculations (see Appendices 3.1 and 3.2 for Kalman filter code).

Battery-Ratiometric Dynamic Tracking ("Go Beyond" Algorithm)

To maintain precise tracking as the battery voltage discharges, the microcontroller calculates light-detection thresholds relative to real-time supply voltage measurements.

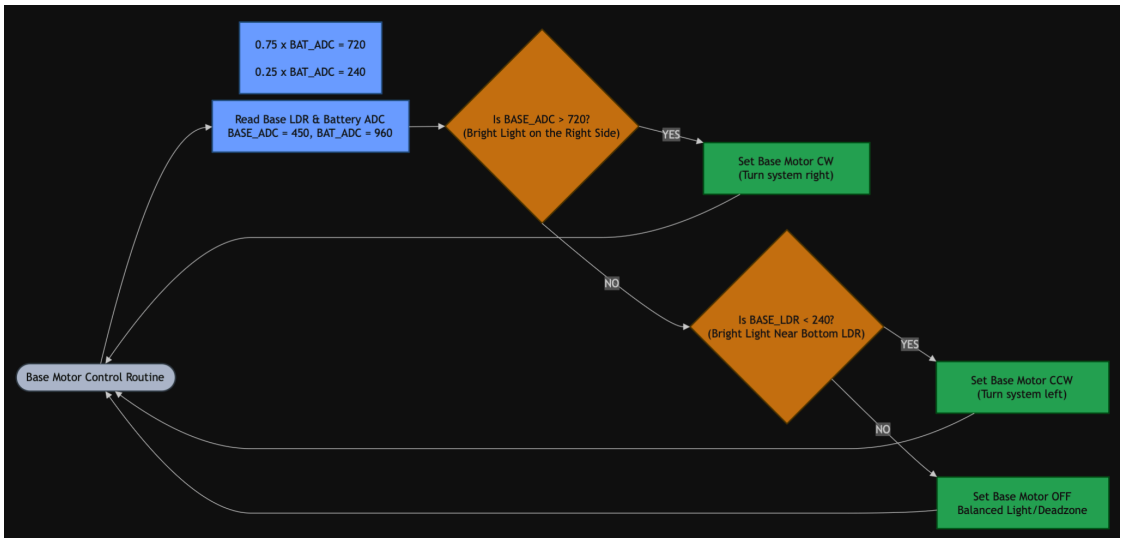


Fig. 19 Battery-ratiometric base motor decision flowchart

The system maintained 100% light-tracking decision accuracy across a full battery discharge window spanning 4.2 V down to 2.7 V. Evaluating LDR midpoint voltages as a dynamic fraction of the real-time filtered battery voltage (3/4VBAT and 1/4VBAT) prevents baseline threshold drift as the supply depletes, ensuring the system continuously identifies the brightest light direction. This directly satisfies the battery-only power constraint and dynamic calibration goal by ensuring

system tracking decisions remain immune to supply voltage droop (see Appendix 6.3 for battery ratiometric tracking logic code).

The flowchart in Fig. 19 illustrates the core ratiometric thresholding logic applied to the base motor routine. The panel motor utilizes this identical directional evaluation logic, with the addition of Hall effect limit sensor checks.

Hall Effect Tilt Limit Accuracy

To enforce physical rotation boundaries and protect the frame from over-rotation, digital Hall effect sensors were integrated to provide deterministic, contactless tilt-limit detection.
The I/O waveform and boundary detection performance was previously verified in Fig. 16.
The digital Hall effect sensor demonstrated a maximum physical trip distance of 1.1 cm, achieving a 100% detection success rate at ≤ 1.1 cm and 0% false triggers beyond 1.2 cm across all tested sweep speeds and supply voltages. Securely mounting the Hall effect sensors and panel magnets ensured consistent mechanical alignment, confirming that the hardware reliably detects proximity boundaries to halt motor motion. This directly satisfies the pre-packaged sensor requirement and primary physical safety goal by enforcing strict tilt limits to protect the mechanical structure.

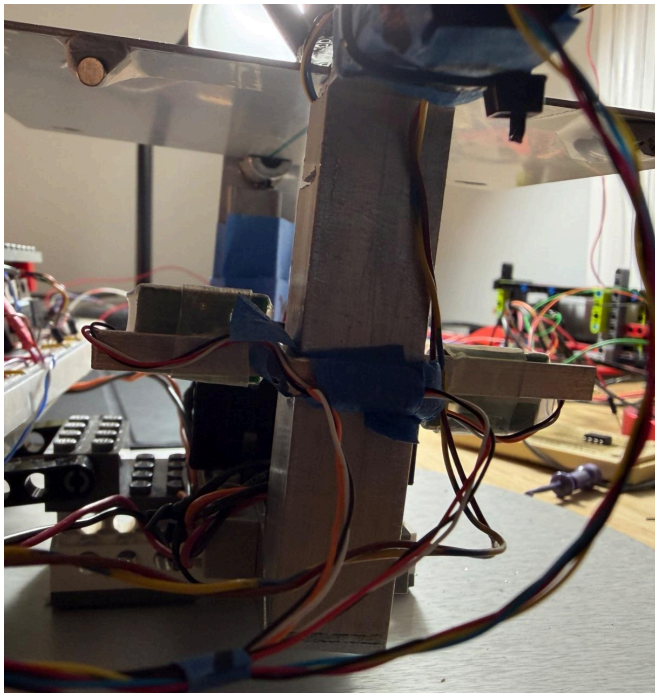


Fig 20. Hall Effect Sensors Mounted on Aluminum Bracket

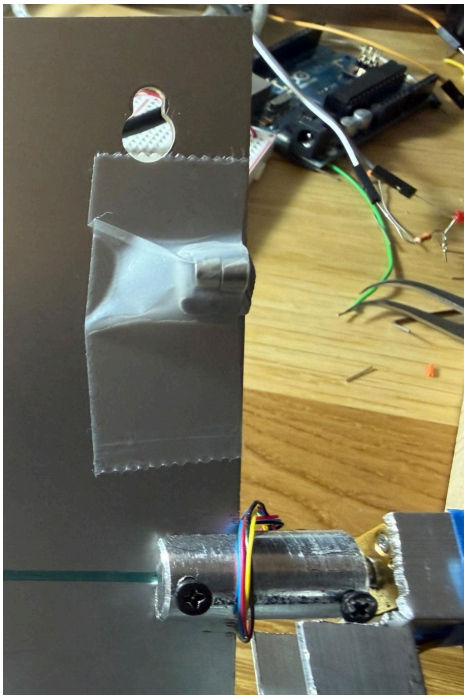


Fig 21. Magnet Mounted on Panel

**Note: Project Video is in Appendix 7.1 and source code is in Appendix 7.2*

8. Failure Analysis

During experimental testing, the Hall effect sensors were evaluated as limit switches using a physical setup: a zip tie was attached to the motor shaft with a magnet secured at the end. As the motor spun the zip tie toward the limit boundary, the magnet was expected to pass over the Hall effect sensor, triggering a LOW signal. The microcontroller was expected to read this transition, cut power to the motor, and stop rotation immediately at the boundary.

When the motor drove the zip tie toward the limit, the sensor output dropped LOW as expected, and the MCU cut motor power. However, due to rotational momentum, the zip tie coasted past the Hall effect and the output returned to HIGH. Because the loop polled the state of the pin, seeing a HIGH signal caused the microcontroller to infer that the limit was not reached and re-engage the motor in the same direction. This caused the system to overshoot and continuously spin past the Hall effect limit.

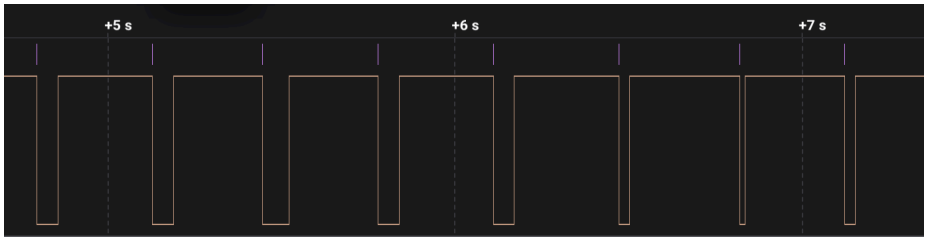


Fig. 22 I/O Waveform of Hall Effect Boundary Overshoot

The initial software design failed because it continuously polled the Hall effect sensors without keeping track of their states. I assumed that cutting motor power would result in an immediate stop without coasting. Because the Hall effect sensor had a narrow detection region, the coasting motor caused the magnet to overshoot the sensor.

Using a logic analyzer, I measured a ~15 ms window between receiving a LOW signal from the Hall effect sensor and turning off the motor. I then implemented directional software flags for the Hall effect sensors. When a sensor triggers LOW while the motor moves in a given direction, a software flag blocks further motion in that direction while still allowing the motor to stop or rotate in reverse (see Appendix 8.1 and 8.2 for the software flag code). Once the motor drives back in the opposite direction, it clears the directional flag to prepare for the next cycle.

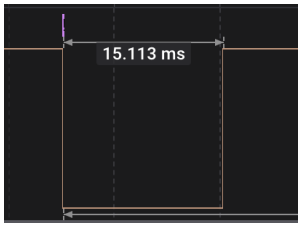


Fig. 23 I/O Waveform Showing ~15 ms Window

9. Physical Implementation

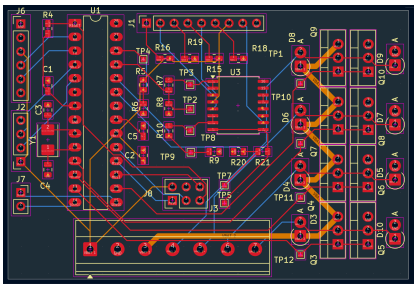


Fig 24. PCB Layout / Routing

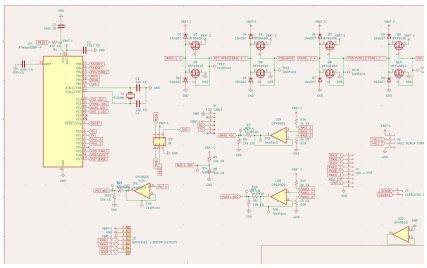


Fig 25. Circuit Schematic

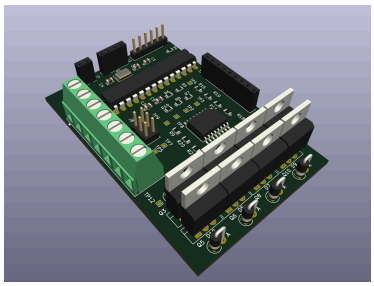


Fig 26. 3D Rendered Board

The PCB was designed in KiCad as a four-layer board, combining surface-mount components - resistors, capacitors, the OP495 quad op-amp, and crystal - with through-hole parts including the ATmega328P, MOSFETs, diodes, and all headers. THT MOSFETs were chosen so small heatsinks could be added later if needed. Headers were assigned by signal type: the potentiometer and LDRs share one female header, the Hall effect sensor gets its own two-pin header on the opposite side to ease routing, and the capacitive touch sensor has a separate header. Battery and motor lines run through screw terminals rather than pin headers for a more secure connection. The FTDI header sits above the Hall effect header, and the ISP programming header occupies the center of the board. Decoupling capacitors and the crystal sit close to the MCU to reduce noise, while analog circuitry sits on the right side of the MCU for easy routing. The H-bridge MOSFETs and diodes are grouped as an isolated cluster since only four traces tie them back to the MCU, and this section uses thicker 1.0 mm traces to reduce trace resistance. Test points were placed at nearly every node for debugging (see Appendix 9.1 for design files).

The light tracker sits on a rigid metal U-shaped base attached to a flat plate, forming a self-supporting frame. The assembly rotates on a LEGO turntable driven by a motor-and-wheel. Batteries mount at the base of the U to keep the center of gravity low, while the electronics ride on a breadboard above the rotation motor, housed in a LEGO enclosure that contains the wiring. Every subsystem is screwed or fastened to the frame with hand-cut brackets and custom couplings, and nothing structural relies on tape - resulting in a self-contained unit that can be picked up, rotated, and operated as one piece (see Fig. 1 for the full enclosure).

Power is split across two Li-ion packs to isolate noisy motor current from sensitive analog circuitry. V Bat 1 supplies the MCU and sensor circuitry, while V Bat 2 powers the H-bridge for the motors. Both are controlled by the same ON/OFF switch.

During testing, a physical problem occurred when the aluminum frame cracked due to cutting and bending. The fix was to attach aluminum brackets at the cracked joints to reinforce the frame for reliable operation.

10. Course Reflection

Project Reflections

- What assumption was most incorrect?
  Looking back, the assumption that was most incorrect was assuming that as soon as the Hall effect sensor triggered LOW and told the motor to stop, it would stop instantly. In reality, the motor kept coasting past the sensor due to momentum. I fixed this in software later on, but if I hadn't, the project wouldn't have worked.
- Where did theory deviate most from reality?
  Theory deviated most from reality with the capacitive sensor. When I plotted the counter value over time, the signal had huge, random spikes before dropping back down. I couldn't explain why it behaved like that, but it was a real physical observation that isn't captured by standard capacitive sensing math.

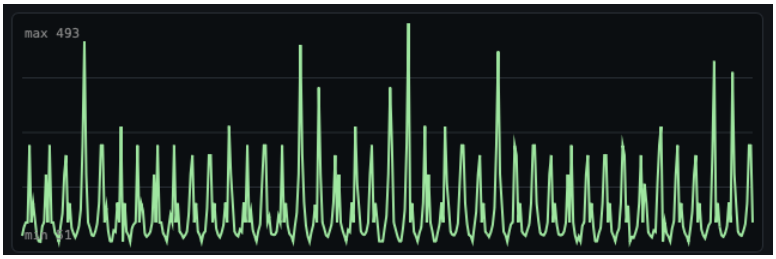


Fig. 27 Capacitive touch sensor's count output over time (X-axis: time, Y-axis: counts)

- What design decision limited your performance?
  Choosing the ATmega328P limited my performance the most due to its processing power and memory limits. To get around this, I had to optimize my code as much as possible, like switching to a fixed-point Kalman filter and replacing bit-shifting operations with direct bitwise AND masks to save CPU cycles. Even with those limits, if I were to start over, I would still pick the ATmega328P because it's a great chip for mastering bare-metal hardware fundamentals.
- What would you NOT do again?
  If I were to build this again, I would spend more time organizing and trimming my wiring. I could have placed components on the breadboard much more efficiently to keep wire runs short. Connecting so many external sensors with full-length wires made the setup messy, and I wasted a lot of time trying to cram all that extra wire length into place.

Lecture Reflections

- Any thoughts to share about the lectures?
  I really enjoyed the lectures and found the content helpful. If possible, I think adding bare-metal programming (or any low-level language) to the course would be awesome. Getting hands-on with the hardware directly made a huge difference in how I understand hardware operations and performance optimization

11. References

Human References: Ashok Mahadevan, Kai DeVito

AI References:

- Tool: Gemini
- Used for:
    - Checking code
    - Creating visual diagrams
    - Answering bare-metal questions
    - Coming up with ideas for debugging

Appendix

1. Problem Definition and Environmental Project Constraints

1.1 Sensors Checklist

2. Sensor Selection and Design Decisions

2.1 LDR Datasheet

2.2 Hall effect datasheet

2.3 Potentiometer datasheet

3. RAW Sensor Dynamics and Signal Processing

3.1 Code for Kalman filter

```
115  signed int kalman_update(struct KalmanFilter *select, unsigned int raw_adc_val)
116  {
117      // Scale the raw ADC value by 65536
118      int64_t raw_adc_val_multiplied = (int64_t)raw_adc_val * 65536;
119
120      // Predict error covariance (p)
121      select->p = select->p + select->q;
122
123      // Calculate Kalman gain (K) which is scaled up by 65536
124      select->k = ((select->p) * 65536) / (select->p + select->r);
125
126      // Update state estimate (x)
127      select->x = select->x + ((select->k * (raw_adc_val_multiplied - select->x)) / 65536);
128
129      // Update error covariance (p) which is used in the next iteration
130      select->p = ((65536 - select->k) * select->p) / 65536;
131
132      // Scale down value by 65536
133      signed int filtered_adc_val = (signed int)(select->x / 65536);
134
135      // Return filtered ADC value
136      return filtered_adc_val;
137  }
```

3.2 Picture of struct code

```
34   struct KalmanFilter
35   {
36       int64_t k; // Calculated kalman gain
37       int64_t x; // Filtered state estimate
38       int64_t p; // Estimate error covariance
39       int64_t q; // Process noise covariance
40       int64_t r; // Measurement noise covariance
41   };
42
43   struct KalmanFilter base_kalman_filter    = {0, 33554432, 65536, 13107, 655360}; // Base LDR
44   struct KalmanFilter panel_kalman_filter   = {0, 33554432, 65536, 13107, 655360}; // Panel LDR
45   struct KalmanFilter battery_kalman_filter = {0, 33554432, 65536, 66,    655360}; // Batery voltage
46   struct KalmanFilter pot_kalman_filter     = {0, 33554432, 65536, 13107, 655360}; // Potentiometer
```

4. Capacitive Sensor (User Input System)

4.1 Capacitive Sensor Timing Code

```
199     ////////////////////////////////////////////
200     /////////// Capacitive Touch Sensor ///////////
201     ////////////////////////////////////////////
202
203     // ** ONLY ENABLED ONCE ON BOOT
204
205     // Capacitive touch check (SEND = PB0 / Arduino pin 8, RECEIVE = PB4 / Arduino pin 12)
206     unsigned int touch_time = 0;
207
208     DDRB = (DDRB & 0b11101110) | SEND_OUTPUT; // Reset PB0 (SEND) PB4 (RECEIVE); set send = 1 (OUTPUT)
209     PORTB &= RECEIVE_PULL_UP_OFF; // Set RECEIVE pull-up LOW
210
211     DDRB |= RECEIVE_PIN; // Set RECEIVE to OUTPUT
212     PORTB &= 0b11101110; // SEND + RECEIVE pull-up LOW
213
214     DDRB &= ~RECEIVE_PIN; // Set RECEIVE to INPUT
215     PORTB |= SEND_OUTPUT; // Set OUTPUT pin pull-up HIGH
216
217     while (!(PINB & RECEIVE_PIN)) // wait for RECEIVE to go HIGH (charge time through mutual capacitance)
218     {
219         touch_time++;
220     }
221         if (touch_time > 40) // Determined experimentally, base reading is 14, can be different on other boards
222     {
223         capacitive_sensor(); // Call capacitive sensor function
224     }
225     touch_time = 0;
```

4.2 Capacitive Sensor Function

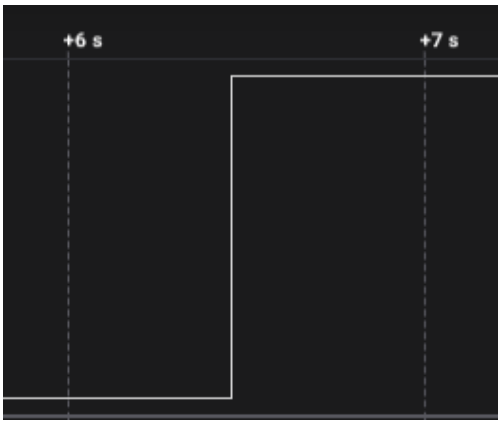
```
185     void capacitive_sensor(void)
186     {
187     UCSR0B = 0; // turns off TX
188     UART_enabled = 0;
189     }
```

5. Other Sensors Used

5.1 Falling-edge



5.2 Rising-edge



6. Digital Logic + Algorithms

6.1 ADC code

```
78  void ADC_Setup(void)
79  {
80      ADMUX |= VREF_1_1;
81      ADCSRA |= PRESCALER_ADEN;
82  }
83
84  unsigned int getADCValue(unsigned char channel)
85  {
86      ADMUX = (ADMUX & MUX_BITS_OFF) | channel;
87      ADCSRA |= (1 << ADSC);
88      while (ADCSRA & (1 << ADSC));
89      return (ADC);
90  }
```

6.2 Hall effect logic

```
238         ////////////////////////////////
239         /////// Hall Effect Check /////////
240         ////////////////////////////////
241         uint8_t right_hall_high = (PIND >> 2) & 1; // PD2 (Right)
242         uint8_t left_hall_high  = (PIND >> 3) & 1; // PD3 (Left)
243
244         if (right_hall_high)
245         {
246             if (right_last_message != 'A')
247             {
248                 if (UART_enabled)
249                 {
250                     UART_Transmit_Ptr("Right Hall effect is HIGH\n");
251                 }
252                 right_last_message = 'A';
253             }
254         }
255         else
256         {
257             if (right_last_message != 'B')
258             {
259                 if (UART_enabled)
260                 {
261                     UART_Transmit_Ptr("Right Hall effect is LOW\n");
262                 }
263                 right_last_message = 'B';
264             }
265         }
266
```

6.3 Battery ratiometric tracking logic

```
314         if (base_filt_val > (3 * battery_adc) / 4)
315         {
316             if ((PORTD & BASE_MOTOR_CW) == 0)
317             {
318                 setBaseMotor(BASE_MOTOR_CW);
319                 if (UART_enabled)
320                 {
321                     UART_Transmit_Ptr("Base motor is CW\n");
322                 }
323             }
324         }
325         else if (base_filt_val < battery_adc / 4)
326         {
327             if ((PORTD & BASE_MOTOR_CCW) == 0)
328             {
329                 setBaseMotor(BASE_MOTOR_CCW);
330                 if (UART_enabled)
331                 {
332                     UART_Transmit_Ptr("Base motor is CCW\n");
333                 }
334             }
335         }
336         else
337         {
338             if (PORTD & BASE_MOTOR_MASK)
339             {
340                 setBaseMotor(MOTOR_OFF);
341                 if (UART_enabled)
342                 {
343                     UART_Transmit_Ptr("Base motor is OFF\n");
344                 }
345             }
346         }
```

7. Final Design and Experimental Results

7.1 Project Video

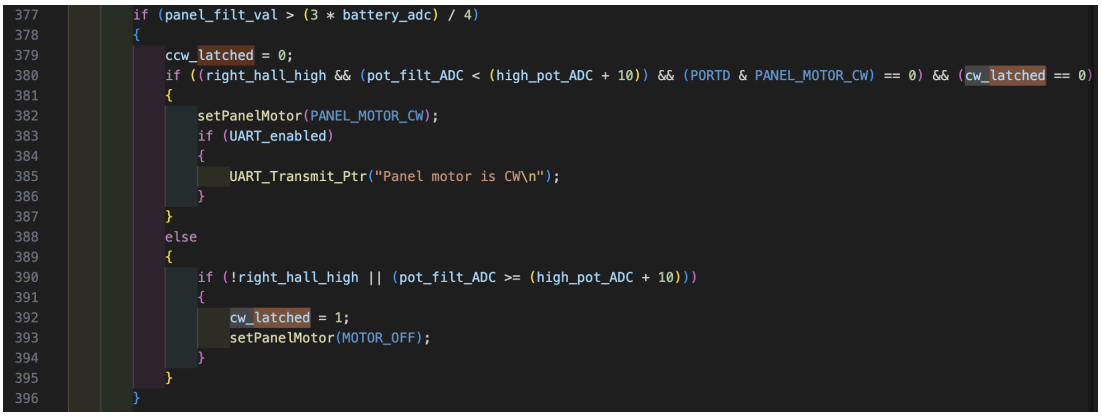
7.2 Github link for source code

8. Failure Analysis

8.1 Software flag initialization

```
236     // Motor state latches
237     int cw_latched = 0;
238     int ccw_latched = 0;
```

8.2 An example of software flag implementation



```
377         if (panel_filt_val > (3 * battery_adc) / 4)
378         {
379             ccw_latched = 0;
380             if ((right_hall_high && (pot_filt_ADC < (high_pot_ADC + 10)) && (PORTD & PANEL_MOTOR_CW) == 0) && (cw_latched == 0)
381             {
382                 setPanelMotor(PANEL_MOTOR_CW);
383                 if (UART_enabled)
384                 {
385                     UART_Transmit_Ptr("Panel motor is CW\n");
386                 }
387             }
388             else
389             {
390                 if (!right_hall_high || (pot_filt_ADC >= (high_pot_ADC + 10)))
391                 {
392                     cw_latched = 1;
393                     setPanelMotor(MOTOR_OFF);
394                 }
395             }
396         }
```

9. Physical Implementation

9.1 Github link for design files

10. Project Github

10.1 Github repository