**Picture of Hardware, Physics, and Transfer Function**



*Fig 1. Homemade capacitive touch sensor*

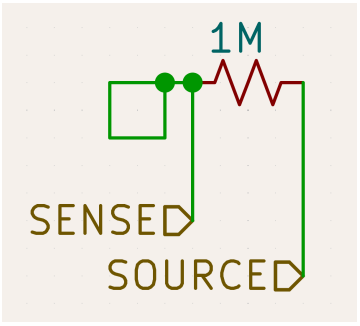



*Fig 2. Schematic of capacitive touch sensor*

The capacitive sensor is made from a 1 MΩ resistor and an unshielded metal wire (Fig 1 ), connected in series between two GPIOs (charge source and sense node), forming an RC 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, $\Delta C$. 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 $V_{IH}$ ($\approx 0.6 \cdot V_{CC}$ for the ATmega328P). Solving for that crossing time: $t_{th} = RC \cdot ln[\ V_{CC} / (V_{CC} - V_{IH})\ ]$

A counter was implemented in the code whose value increases in proportion to $t_{th}$. Rather than converting this counter value into a time value, it can be used directly to find a threshold that separates a touch from no touch. Thus, the counter value stays dimensionless. The counter values were monitored through UART logs. It stayed at a baseline of 29 with no touch. With a touch, the counter rises close to 500. Since the no-touch baseline always stays at 29, the rise above baseline is a usable touch indicator. Transfer function:

*Sensor ON if $N > 29$*

*Sensor OFF otherwise ($N \leq 29$)*

This is the system's actual experimental transfer function: a unimodal response where touch can be safely detected above the baseline. To verify the range was good, an I/O pin was set HIGH when the sensor was touched and set LOW when the sensor was not touched. After repeated testing, the I/O pin goes HIGH and LOW virtually instantly proving that the experimental transfer function correctly maps the capacitive sensor. The RC model predicts one touch band, and reality shows one: a single high branch, consistent with what the RC model predicts on its own.

**Signal Processing for Capacitive Sensor**

```
[21:14:39] Touch time val: 29
[21:14:39] Touch time val: 29
[21:14:39] Touch time val: 29
[21:14:39] Touch time val: 29
[21:14:39] Touch time val: 29
[21:14:39] Touch time val: 29
```

*Fig 3. Counter value with no touch*

```
[21:15:19] Touch time val: 73
[21:15:19] Touch time val: 98
[21:15:19] Touch time val: 188
[21:15:19] Touch time val: 265
[21:15:19] Touch time val: 226
[21:15:19] Touch time val: 90
```

*Fig 4. Counter value with touch*

No filtering or averaging is applied. The raw counter value from each charge cycle is used directly, and detection is done with a simple threshold decision on that count. Some capacitive sensors respond to a

finger merely being near the sensing element, since even an air gap couples some extra capacitance in. This sensor was tested for that behavior by bringing a finger close without contact, and no change in the count was observed. The count only moves once the finger physically touches the wire. So this design is a contact capacitive sensor. With no touch, the counter consistently reads 29 (Fig 3). Touching it was found to move the count in a single direction, rising to nearly 500 (Fig 5). Rather than applying any signal conditioning, the threshold was set a little above the 29 baseline so small changes in the resting value would not be misread as a touch. The sensor is therefore read as touched when the count rises above 40, and untouched otherwise.

**Second Pre-Packaged Sensor**

The second pre-packaged sensor was a hall effect sensor. To characterize its trip distance, a test fixture was built on a plastic ruler (0.1 cm precision) with the sensor mounted vertically at the 0 cm mark and a pivoting rod anchored at the 6 cm line. A permanent magnet was attached to the rod at varying distances while a logic analyzer monitored the sensor's output pin. The sensor has a single trip point at d = 1.1 cm.

To calibrate the sensor, the 1.1 cm trip point was verified across supply voltages of 2.7, 3.3, 3.7, 4.2, and 5.0 V to simulate battery droop. The threshold held fixed at 1.1 cm across every tested voltage, and the sensor switched cleanly between HIGH and LOW with negligible hysteresis. Detection was equally clean across a range of magnet sweep speeds, with zero data loss or distortion, so neither supply voltage nor sweep speed affects the trip point.

Since the trip point does not change with supply voltage, the 4.2 V battery being used can droop due to continuous use without affecting the tilt limits. And because it does not drift at higher sweep speeds, the limits are enforced regardless of how fast the motor drives the panel.

To validate this behavior experimentally, the sensor was integrated into the light tracker's tilt-limit system. A wooden test rig holds a motor with a magnet mounted on a ziptie arm. There are two Hall effect sensors positioned on either side of the motor marking the CW and CCW rotation limits. On the breadboard, there are four LDRs: two control the panel motor and two control the base motor. Together, they steer the light tracker.

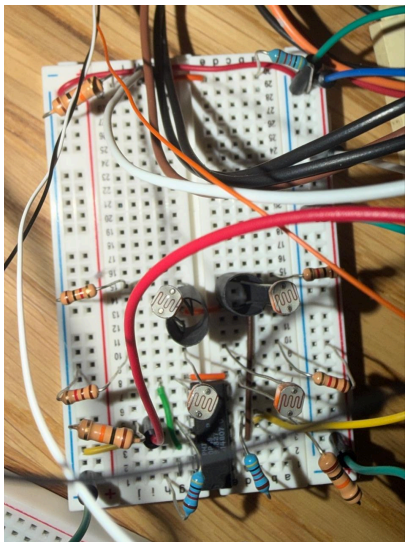


*Fig 6. 4 LDRs on breadboard*

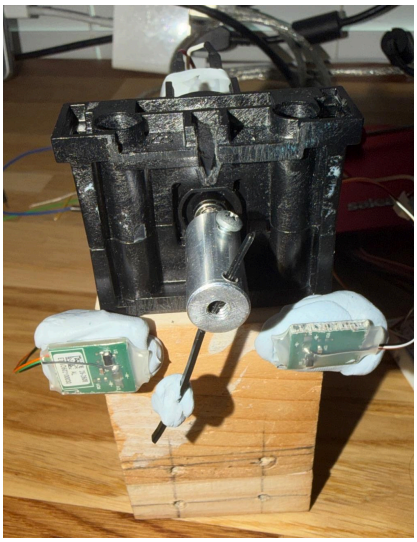


*Fig 7. Motor, magnet on ziptie, and two hall effect sensors*

The motor turns right (CW) when the right LDR receives more light and left (CCW) when the left LDR receives more light. The tilt limits were implemented via software polling that stops the motor once the magnet reaches a Hall effect sensor. The system was tested entirely through the LDRs driving the panel

motor: covering the left LDR made the motor spin CW and covering the right LDR made it spin CCW. In each case, the motor stopped exactly where the Hall effect sensor was, confirming the sensors correctly enforce the tilt limits and prevent overrotation. This was verified through visual observation and logic analyzer waveforms. The attached video shows this behavior: Hall effect experimental results

**What I Learned This Week**

This week, I learned how to make a capacitive sensor. Initially, I thought you would need to build something like a homemade capacitor with two plates, but after a little googling and asking AI, I found the design was a lot simpler. I also learned how to switch between several ADC channels. To keep the code simple and optimized, every sensor that requires an ADC can reuse the same function. And if you want a specific channel, you just pass the MUX bits as an input parameter to that function. This keeps the code neater as you no longer need separate ADC functions for each channel, which means less instructions and compiles down to smaller code.

**AI Interactions**

I used AI throughout the capacitive touch sensor work. I asked AI questions of how the capacitive touch sensor works, and then asked about different ways to build one. This led to a simple design consisting of a 1 MΩ resistor and an unshielded wire. Additionally, since I am programming in bare metal using the AVRIO library, I could not rely on the easy-to-use CapacitiveSensor library available in the Arduino IDE. After programming it myself, I used AI to verify my code.