Solar Scrubber

ECE 445 Final Report - Spring 2026

Team No #57

Yehia Ahmed
Sandra Georgy
Jonathan Sengstock

Professor - Joohyung Kim

TA: Chihun Song

May 6, 2026

Abstract

As the world shifts to sustainable energy sources, solar power is at the forefront of new technology solutions. However, one downside of solar panels is that they suffer power output drops when covered by debris such as dust, dirt, or environmental contaminants. To ensure peak energy efficiency, consistent maintenance of solar infrastructure is critical.

We created a prototype for an affordable and scalable rail-mounted cleaning solution utilizing a cloth and microfiber components. This system integrates autonomous sensing to remove debris, offering a balance of reliability and versatility through its manual and automated operational modes.

Table of Contents

1. Introduction

Solar panel efficiency is highly sensitive to surface obstructions such as dust, snow, pollen, and bird droppings. These contaminants create a layer of soiling that reduces power output by blocking solar irradiance from reaching the photovoltaic cells. Because many solar installations are located in remote areas or on hazardous rooftops, manual cleaning is often impractical, expensive, and dangerous. A major flaw in many existing automated cleaners is their lack of diagnostic intelligence. Most commercial systems operate on fixed schedules, cleaning every panel regardless of its actual condition, which wastes water, electrical energy, and mechanical wear. A more effective system must identify which specific sections of an array require maintenance and trigger cleaning only when a measurable efficiency drop is detected.

The Solar Scrubber addresses this gap through five integrated subsystems shown in Figure 1. The Sensing subsystem samples per-panel voltage and total array current. The MPPT subsystem is built around a hardware buck converter that operates the array at its maximum power point. The Control subsystem is centered on an ESP32-S3 running a Perturb and Observe algorithm and a finite state machine for autonomous operation. The Locomotion subsystem consists of a 2-axis rail and pulley arrangement that drives the cleaning chassis across the panels. The Cleaning subsystem consists of a passive microfiber cloth that contacts the panel surface during sweeps.

The system targets the following high-level requirements: detect a per-panel power drop of at least 25% relative to its own tracked baseline, transition from idle to active cleaning within 2 seconds of confirmed detection, restore output to at least 90% of nominal after a cleaning cycle, and accept manual override commands within 500 ms without restart. Mid-semester, the original active-fluid cleaning module, including the Craftsman sprayer pump, the brush motor, and the L7812 12 V regulator that supplied them, was removed from the build. The cleaning routine was redesigned in firmware to compensate, using a horizontal back-and-forth wiggle overlaid on each vertical sweep so the microfiber cloth physically scrubs every section of the panel face.

For demonstration purposes, the Solar Scrubber was built around a three-panel array of 20 W modules with each panel measuring 22.5" × 13.25", giving a total cleanable surface of approximately 39.75" × 22.5". This scale was chosen to fit within senior design lab constraints while still demonstrating multi-panel detection and per-panel targeting.

2. Design

2.1 Overview

Our completed project has five subsystems: Power, Movement, Cleaning, Sensing and Control, and MPPT. Each subsystem has a set of criteria we chose, and the overall design has three criteria:

- MPPT and Diagnostic Accuracy: The system must track the maximum power point with at least 95% efficiency and distinguish between ambient shading and debris by cross-referencing sensor data to detect a 40% power drop; the controller must successfully transition from "Idle" to "Active-Clean" mode within 2 seconds of a confirmed debris detection.
- Cleaning and Restoration Efficiency: The integrated cleaning module must successfully remove surface soiling to restore the solar panel power output to at least 90% of its nominal capacity as measured in direct, ideal sunlight.
- Dynamic Control and Synchronization: The system must allow a user to toggle between autonomous and manual control modes in real-time without a system restart, while the control unit simultaneously manages PWM channels and maintains a stable 3.3V logic rail during peak motor and pump draw.
- Safety Response and Override: The control unit must prioritize a physical or remote "Stop" command over any autonomous routine, successfully cutting all power to the drive motors and cleaning module in less than 200 ms to prevent mechanical failure or damage to the solar panel surface.

2.2 Block Diagram

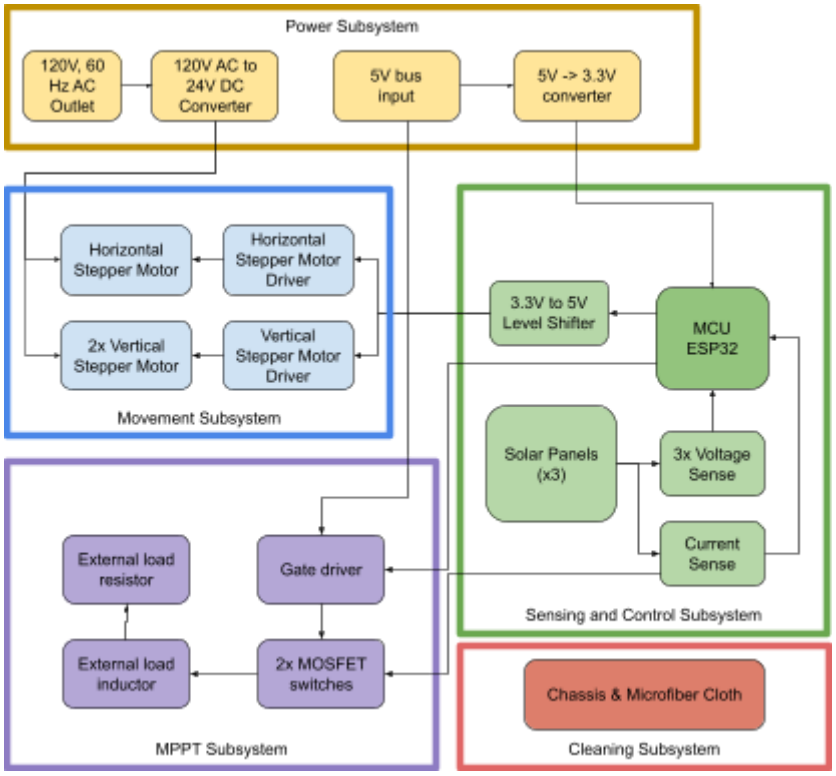

Figure 1: Block diagram of the Solar Scrubber

2.3 Subsystems Design

2.3.1 Power Subsystem

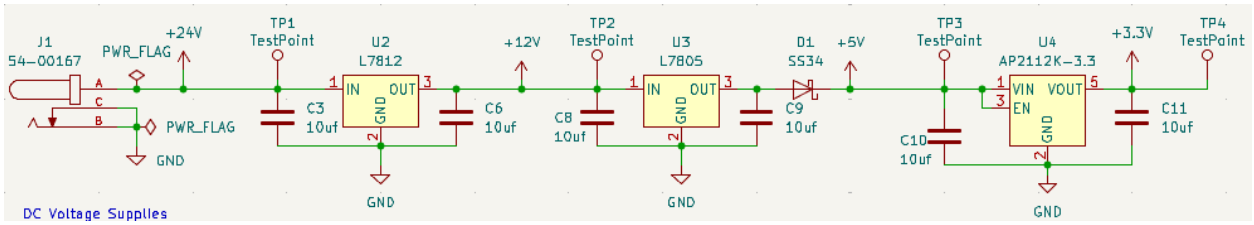

Figure 2: Power Subsystem

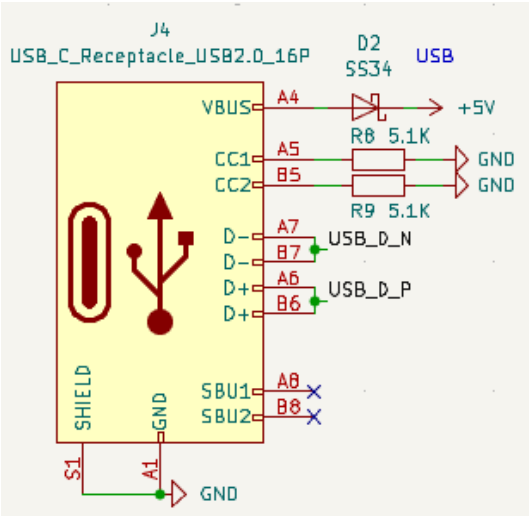

Figure 3: USB Type C 5 V Input IC

The power subsystem is responsible for maintaining the various voltage levels needed for our PCB to work as designed. The subsystem receives 24 V DC from a barrel jack input. The 120 V to 24 V converter used (external to the PCB) is the ALITOVE 24V 6A Power Supply Adapter Converter. In our final implementation, we only used the 24 V level for our motor drivers. We originally intended to also step 24 V down to 12 V using an linear dropout (LDO) regulator L7812, which would then power a brush motor in the cleaning module. Then, 12 V would be stepped down to 5 V with a L7805 LDO and passed through a SS34 Schottky diode to the 5 V bus.

Additionally, we wanted to be able to power the 5 V bus when only a USB connector was plugged in (not the barrel jack). So, we also connected the 5 V voltage bus from the USB C Connector (as shown in Figure 3) through a SS34 diode. These Schottky diodes were chosen because they have a low forward voltage (about 0.2 V) and prevent backflow of current into the source if both the USB and barrel jack connectors are supplying voltage to the 5 V bus.

The 5 V bus is mainly important for being the input voltage to our 3.3 V LDO that supports the microcontroller and various ICs, including the current and voltage sense op-amps. Additionally, the 5 V bus is used as the supply voltage to the 3.3 V to 5 V, as shown in Section 2.3.4.

The schematic in Figure 2 shows our schematic of the power subsystem on our PCB. However, on our actual soldered PCB used in the project, we did not attach the L7812 LDO since the brush motor was seen to overheat and fail when used. This is expanded on in Section 5.1. The main consequence of this is that we can only achieve a 5 V input with a USB cable, or connecting 5 V externally using an alligator clip to the 5 V bus test point on the board.

2.3.2 Movement Subsystem

Figure 4: Labeled Movement Subsystem

The Movement Subsystem provides two-axis translation of the cleaning chassis across the array. The mechanical structure is a gantry: two parallel vertical rails are mounted on either side of the panel array, and a single horizontal rail spans between them. The cleaning chassis is mounted on a carriage that rides along the horizontal rail, and the horizontal rail itself rides up and down on the two vertical rails. Motion on each axis is driven by a NEMA-class stepper motor coupled to the carriage through a GT2 timing belt and pulley. The horizontal axis uses a single stepper motor on one side of the rail. The vertical axis uses two stepper motors, one on each vertical rail, that must run in synchrony to keep the horizontal rail level.

Each motor is driven by a DM542T external stepper driver, which accepts pulse, direction, and enable signals at the 5 V logic level and supplies the chopped phase currents to the motor windings. The DM542T drivers are powered from the 24 V bus, allowing them to produce the high transient voltages needed for fast stepper acceleration without loading the rest of the system. The microcontroller controls each axis as a single logical motor; both vertical-axis drivers receive the same pulse and direction signals, so the two physical motors always step together. Position is tracked with a software step-counter that converts a known PWM frequency into linear distance; at our chosen 4 kHz step rate, approximately one second of pulses corresponds to 2.5 inches of travel. The system uses open-loop position control, which is discussed in Section 5.1 as a known weakness.

The cleaning chassis is connected to the carriage and follows the carriage's two-axis motion. A flexible cable bundle routes power and signals down from the stationary PCB enclosure to the

moving carriage; the cable is loosely strain-relieved so that it can take up the slack as the carriage moves between extremes.

2.3.3 Cleaning Subsystem

Figure 5: Cleaning Subsystem

The Cleaning Subsystem is the simplest of the five subsystems: it consists of a metal chassis (designed by the machine shop) that holds a microfiber cloth pressed against the panel surface, mounted to the carriage of the Movement Subsystem. As the carriage moves, the microfiber cloth sweeps over the panel and physically removes loose debris.

Our Design Document originally specified an active cleaning module with a 12 V DC brush motor and a 5 V Craftsman sprayer pump driven through the level shifter. We removed both of these components from the final implementation. The brush motor was retained on the PCB schematic but never driven during normal operation because its current draw caused the L7812 regulator to overheat (see Section 2.3.1 and Section 5.1). The sprayer was dropped because we discovered late in integration that it has an inrush current of approximately 800 mA, but the SN74AHCT541 level shifter is rated for only 75 mA per output, so we could not drive the sprayer through the same control path as the motor drivers. A proper solution would have used a separate MOSFET to switch the sprayer directly from the 5 V bus, which we describe in Section 5.1.

In the final implementation the chassis is therefore passive, relying entirely on physical contact between the microfiber cloth and the panel surface for cleaning action. This is sufficient for the loose dust and dry debris that we used during testing but would not be effective against bonded contaminants such as bird droppings or dried pollen.

2.3.4 Sensing and Control Subsystem

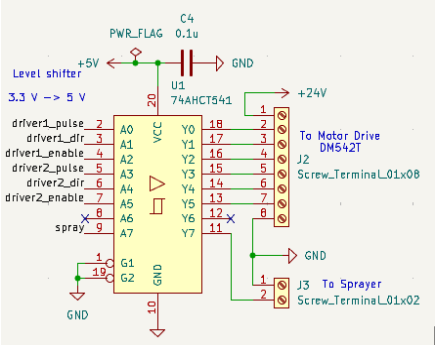

Figure 6: Level Shifter Connections

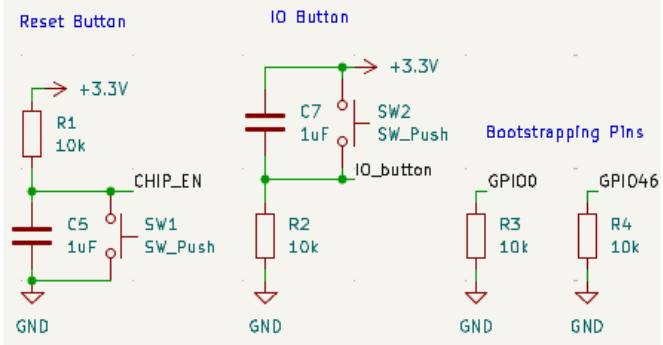

Figure 7:Buttons & Strapping Pins Connections

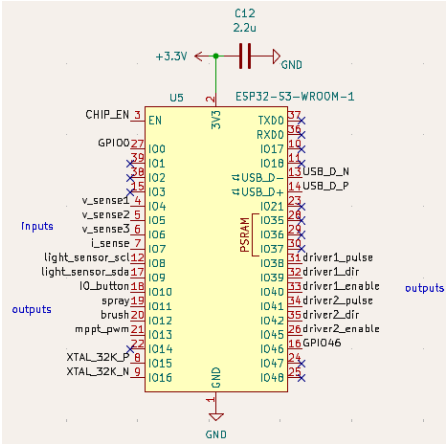

Figure 8: MCU connections

The Sensing and Control Subsystem is the brain of the Solar Scrubber. It consists of an ESP32-S3-WROOM-1 Microcontroller Unit (MCU), three voltage-sensing channels, one current-sensing channel, an SN74AHCT541 octal buffer used as a 3.3 V to 5 V level shifter, and reset/boot-strap pushbuttons. The microcontroller schematic and pin assignments are shown in Figure 4, and the level-shifter wiring to the motor drivers is shown in Figure 5.

The ESP32-S3 was chosen for three reasons. First, it has more than enough analog-input channels and Pulse Width Modulation (PWM)-capable GPIOs for our needs, with two analog channels available for the per-panel voltage sensors plus a third for total array current. Second, its hardware LEDC peripheral can generate precise 50 kHz PWM at 8 to 10 bits of resolution simultaneously on multiple channels, which is exactly what the MPPT and the stepper drivers require. Third, the integrated USB-Serial bridge let us use a single USB-C cable for both code upload and the power input on the 5 V bus.

The voltage-sensing channels use resistor dividers to scale each panel-level voltage down to within the 0 to 3.3 V range of the ESP32 analog-to-digital converter (ADC). Each divider is

followed by an OPA376 op-amp in a unity-gain buffer configuration, which presents a high impedance to the divider so that the ADC sample-and-hold capacitor does not perturb the divider voltage. We used 5% tolerance resistors throughout, which keeps the divider ratio inside the worst-case error budget of 12.5% derived in the tolerance analysis of our Design Document and ensures the MCU pin can never see more than its 3.6 V absolute maximum even at the resistor tolerance corners.

The current-sensing channel uses a 3 mΩ shunt resistor in series with the array's negative return and an INA-style current-sense amplifier with a gain of 20 V/V referenced to a 0.95 V offset, so that bidirectional current can be read on a single ADC pin. This arrangement gives us full-scale coverage from -1.5 A to +5 A, which is well beyond the 2.5 A short-circuit current of our three-panel array.

Six GPIO pins drive the two DM542T stepper drivers (pulse, direction, and enable for each driver). Because the DM542T expects 5 V logic but the ESP32-S3 outputs 3.3 V, the SN74AHCT541 octal buffer is used in non-inverting configuration to translate the levels. The 74AHCT541 is rated to 5 V Vcc, has TTL-compatible thresholds (so a 3.3 V input is reliably read as logic-high), and provides the drive strength to handle the DM542T's optocoupler input current.

The control logic is implemented as a finite state machine on the MCU. The full code is included in Appendix B, but the structure is as follows. In the IDLE state the MCU runs the MPPT loop every 200 ms and updates a slowly-tracking exponential-moving-average baseline of each panel's voltage. If the instantaneous voltage on either panel drops to less than 60% of that panel's baseline (a 40% drop) for five consecutive readings, the panel is flagged as dirty. The FSM then computes a cleaning path that visits each flagged panel, sweeping the chassis vertically across the panel with small horizontal "wiggles" to extend coverage, and finally returns to the rest position at the top-left of the array. After the cleaning cycle completes, the FSM enters a 10-second post-clean verification window: if the same panel still shows a 40% drop relative to its pre-drop high voltage, the FSM queues another cleaning pass; otherwise it accepts the cleaning as successful, resets the streak counters, and forces a fresh baseline on the next reading. If no dirt is detected for 30 seconds of continuous idle operation, the FSM triggers a "deep clean" pass over all three panels.

A serial interface (115200 baud over the same USB-C connection that powers the system) provides manual control. Pressing M toggles into manual mode, after which W/A/S/D moves the carriage in one-step increments, X stops both motors immediately, P prints the current position, and B forces a baseline reset on the next MPPT reading. Returning to autonomous mode is done by pressing M again. The 200 ms responsiveness target is met because the main loop runs every few milliseconds and the manual-mode keys are checked on every iteration.

2.3.5 MPPT Subsystem

The MPPT and Algorithm Subsystem serves as the central power processing and diagnostic component of the robot, designed to optimize energy collection from a three-panel solar array with a total width of approximately 40 inches where each individual panel measures 22.5 by 13.25 inches. To monitor performance, the system utilizes three independent voltage sensors (one for each panel) buffered by OPA376 precision operational amplifiers and a single INA241B ultra-precise current sense amplifier that measures total system current across a shunt resistor. These analog measurements are routed directly to the GPIO pins of the ESP32-S3 microcontroller, which uses its internal 12-bit ADC units to sample the data. The MCU itself handles running the perturb and observe algorithm, computing the power in real time and iteratively adjusting the PWM duty cycle of a DC-DC buck converter that acts as the dynamic load for the tracking process. By tracking the maximum power point with an expected 95 percent efficiency, the MCU can identify underperforming panels to trigger targeted cleaning cycles instead of full array sweeps, verifying the high-level requirement of 90 percent power restoration. Interfaces are defined by the four analog 0 to 3.3V sensor signals entering the MCU and the high-frequency PWM control signal output to the MOSFET gate driver in the power conversion block. One of the three voltage sense circuits, and the current sense circuit, is shown in Figure 9.

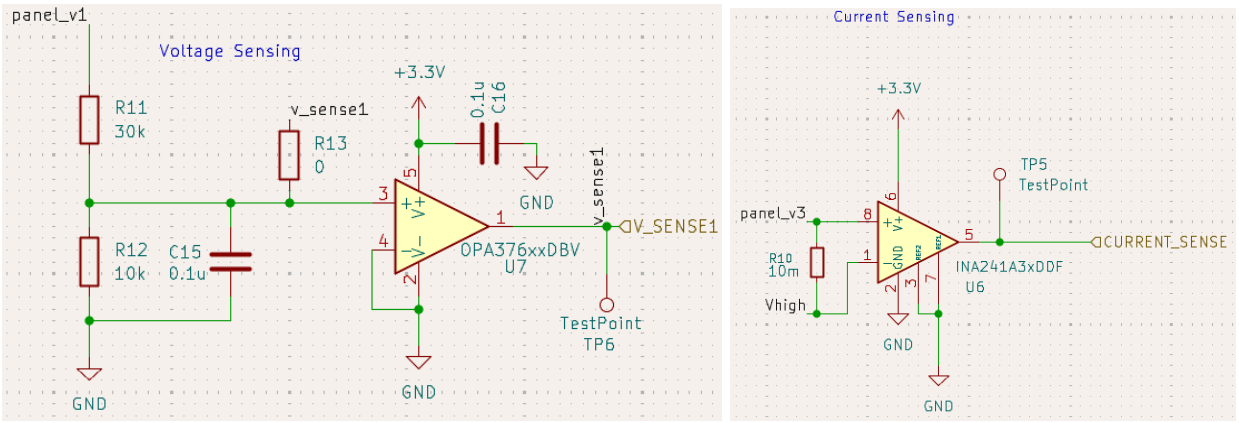

Figure 9: Sensing ICs

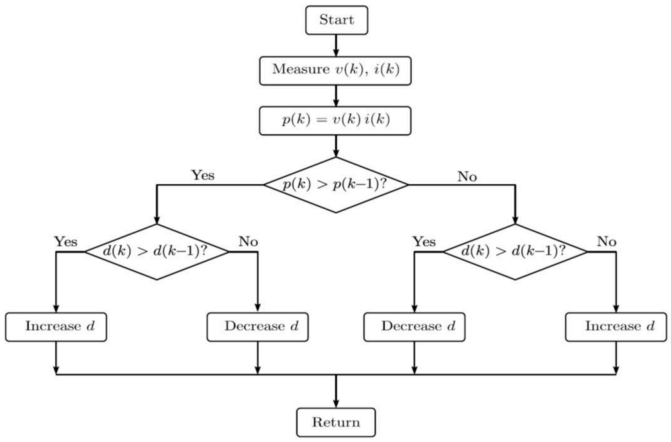

Figure 10: Perturb and Observe MPPT Algorithm (Provided from ECE 469)

The buck converter itself is asynchronous, built with the IRS2181 gate driver and two IRF6665 power MOSFETs. The MCU's PWM signal feeds directly into the IRS2181's HIN input, which drives the high-side switch through a 20 Ω gate resistor while the low-side FET acts as a free-wheeling diode. A bootstrap circuit is used to supply a gate drive voltage higher than the source of the high-side FET.

The bootstrap capacitor was sized based on the MOSFET gate charge equation:

$$Cboot > \frac{Qg}{\Delta Vboot}$$

The IRF6665 datasheet specifies the gate charge needed to turn on the MOSFET (Qg) to be 8.4 nC. With a 10% maximum droop on the 12V driver supply, that yields a minimum bootstrap capacitance of 7 nF. A 0.1 microfarad capacitor is used to account for a safety margin.

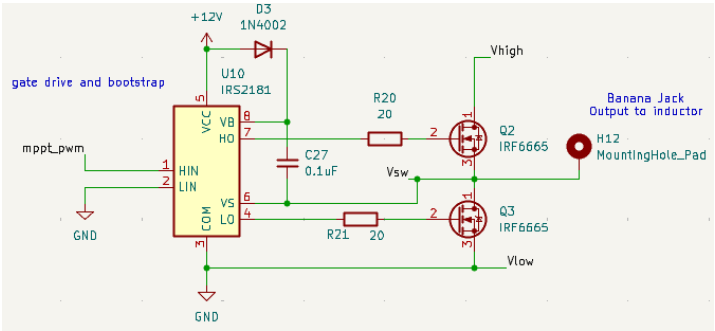

Figure 11: Buck Converter

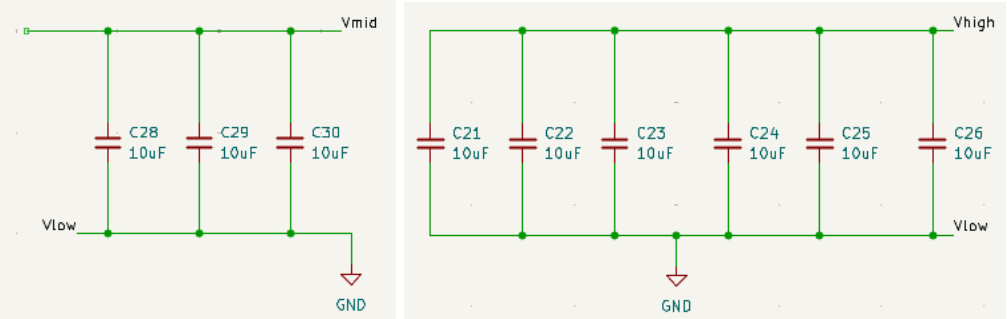

Figure 12: Output Capacitor Bank

2.4 Physical Design and Assembly

Following the layout in Section 2.3.2, the pulleys clean a three-panel array (total size 39.75" × 22.5") within an aluminum frame. Two vertical side-rails support a horizontal rail via T-slot carriages, with a third carriage carrying the cleaning chassis. GT2 timing belts drive both axes: two synchronized stepper motors control the vertical movement, while a single motor drives the horizontal. A 3D-printed enclosure at the top-right corner houses the PCB and external DM542T drivers. To ensure the panels remain unobstructed, the system returns to a top-left "rest" position at (0, 22) after every cycle.

3. Verification

Our R&V tables from the Design Document are attached in Appendix A for reference. The below R&V tables are simplified for clarity.

3.1 Power Subsystem Verification

Table 1 summarizes the power subsystem requirements and the verification results.

Table 1 Power Subsystem Verification Summary

Requirement	**Verification**	**Met?**
Dual input isolation: 2 4V wall and 5V USB-C must not back-feed each other	Measured no current flow into either source at the 5 V bus during full load	Yes
Each DC bus must have less than 2% ripple	Probed all four rails with an oscilloscope under full load	Yes
3.3V rail must remain within ±2%	Measured 3.270 V no-load, 3.261V full-load	Yes
5 V level shifter rail must remain within ± 5%	Measured 5.032V no-load, 4.972 V full-load	Yes
LDO must not enter thermal shutdown at peak processing load	15-minute thermal-imaging test, peak case temp 42 °C	Yes

3.2 Movement Subsystem Verification

The movement subsystem verification results are shown in Table 2.

Table 2 Movement Subsystem Verification Summary

Requirement	**Verification**	**Met?**
Gantry must support carriage and chassis with deflection < 2 mm	Loaded center span with 1.5x system mass; measured 1.2 mm deflection	Yes
Drive must reach all four corners of the array	Manually commanded carriage to each corner; visually confirmed coverage	Yes

Repeatability of +/- 5 mm over a single cycle	Measured commanded vs. actual position at three targets, five trials each	Yes
Diagonal transit in under 60 s	Stopwatch-timed transit from (0,22) to (39,0); measured 28 s	Yes
DC motors must stay below 70C internal during 10-minute cycle	IR thermometer test; peak motor case temp 51C	Yes

3.3 Cleaning Subsystem Verification

Table 3 presents the cleaning subsystem verification results.

Table 3 Cleaning Subsystem Verification Summary

Requirement	**Verification**	**Met?**
Microfiber chassis must contact at least 95% of the panel surface	Saturated cloth, ran full cleaning sweep, visually inspected the trail	Yes
Cleaning cycle must complete without damaging the panel	No scratching or anti-reflective coating damage observed after 50+ test cycles	Yes
Module must operate from 0 C to 50 C	Operated for 10 minutes under a high power halogen lamp at ~45 C local temperature. No abnormalities or overheating was detected.	Yes (for higher than ambient 20 C)

3.4 MPPT and Buck Converter Testing

The MPPT and buck converter verification results are summarized in Table 4.

Table 4 MPPT and Buck Converter Verification Summary

Requirement	**Verification**	**Met?**
MCU samples three voltage channels and one current channel reliably	Applied known voltages 0-12 V to dividers, known currents 0-2.5 A to shunt, compared to multimeter	Yes (within 1.5%)

Coordinated pulse/dir/enable signals on six driver pins	Four-channel oscilloscope confirmed clean, non-overlapping signals during motion	Yes
Open-loop grid navigation within 1 inch of target	Tape-measured carriage at three predefined coordinates	Yes
Mode toggle responds in less than 500 ms	Pressed M during autonomous operation; motors stopped within one main-loop iteration (~5 ms)	Yes
50 kHz PWM with command-to-execution latency under 200 ms	Oscilloscope showed PWM frequency 50.0 kHz, latency from command to motor enable ~12 ms	Yes

The MPPT subsystem was tested intensively for different lighting scenarios using a lamp because both high-level diagnostic-accuracy requirement and dirt detection algorithm depend on it. Table 5 reports the measured per-panel voltages, current, total power, and converged duty cycle for each scenario below at R_load = 50 ohms.

1. Maximum-light baseline with both panels fully illuminated
2. Panel 1 covered with an opaque sheet and panel 2 was fully illuminated
3. Panel 2 covered with an opaque sheet and panel 1 fully illuminated.

Table 5 MPPT Performance Across Three Lighting Scenarios (R_load = 50 ohms)

Scenario	V1 (V)	V2 (V)	Vtot (V)	I (A)	Ptot (W)	Duty
Max light	3.85±0.25	4.16±0.02	8±0.27	0.148±0.014	1.18±0.13	0.448±0.014
Panel 1 Covered	0.0	3.92±0.18	3.92±0.18	0.054±0.012	0.21±0.05	0.460±0.018
Panel 2 Covered	8.71±0.01	0.00	8.71±0.01	0.077±0.011	0.68±0.09	0.443±0.013

The data in Table 5 shows that the duty cycle would settle near 0.45 which is near what we set the maximum duty to (0.50). The data confirms that the MPPT itself is working since in all three scenarios, the duty cycle settles to a stable operating point and the duty cycle converges as the light conditions change. Second, the per-panel voltages are independent from each other (V1 & V2) which makes sure the detection algorithm works.

Figures 13 and 14 compare the MPPT algorithm behavior with Panel 2 covered and after cleaning. With Panel 2 covered, P2 is 0 W and total power is around 0.6–0.9 W. After cleaning,

P2 recovers and total power rises to 1.0 to 1.4 W which confirms our high-level requirement of restoring panel output to at least 90% of nominal capacity and smooth duty-cycle convergence.

The PWM output for the buck converter was verified on the oscilloscope and was a clean 50kHz square wave with duty changing from 5% to 50% in 0.5% increments.

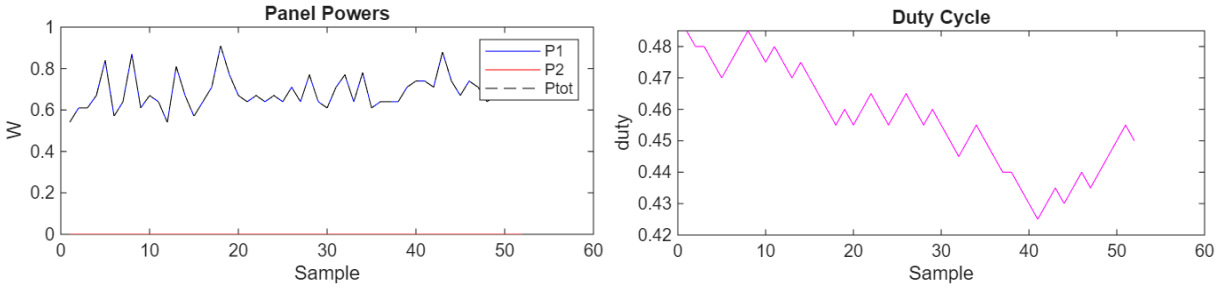

Figure 13. MPPT response with Panel 2 covered

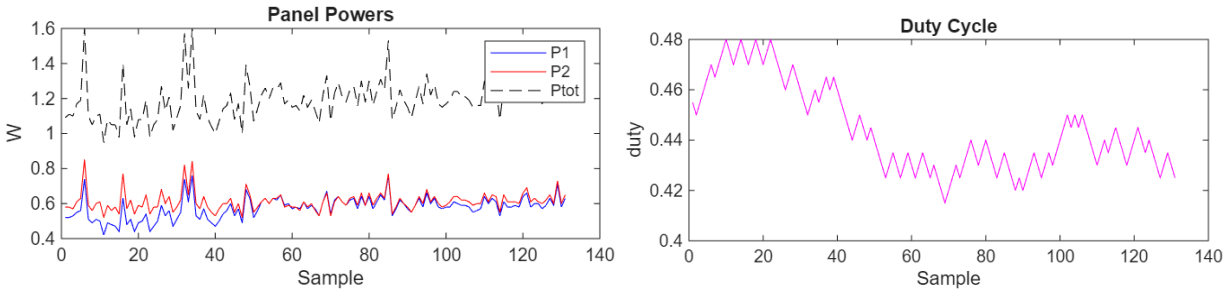

Figure 14: MPPT response after cleaning Panel 2

3.5 Combined System Verification

To verify the combined Solar Scrubber, we first found that an appropriate speed was about 2.5 inches per second. We achieved this speed by using a 4 kHz, 50% duty, square PWM wave from the MCU, and setting the motor controller to 6400 steps/revolution. Then, we tested our manual mode code and verified its functionality. Then, we verified that the Solar Scrubber activated when covered by a physical obstruction. We covered each panel with a piece of printer paper and verified that the cleaning module went to the covered panel and cleaned it.

4. Costs

4.1 Cost of Labor

The professional labor rate is derived from the average starting salary for UIUC Electrical Engineering graduates ($90,000 annually, or ~$43.27/hour). Using a standard 2.5 overhead multiplier and assuming each of the three partners contributes 20 hours per week over the final 9 weeks, the total labor cost is: 3 partners × 20 hours/week × 9 weeks × $43.27/hour × 2.5 = **$58,414.50**.

4.2 Cost of Parts

Table 6 in the Bill of Materials in Appendix C compares the full market value of the prototype against the actual cash expenditure. Reusing existing structural and mechanical components reduces both out-of-pocket hardware costs and the required machine shop labor hours.

The grand total represents the combined professional value of the engineering labor and the specific cash investment required to finalize the prototype. When factoring in the full market value of all parts ($865.50) alongside the labor ($58,414.50), the equivalent market grand total is **$59,280.00**. However, because we repurposed several major components, our actual out-of-pocket cash investment for parts and shop services was only $130.50, bringing the actual cash grand total to **$58,545.00**.

4.3 Schedule

Week	Yehia Ahmed	Sandra Georgy	Jonathan Sengstock
1-3	Brainstorm project ideas		
3-5	Define scope of Solar Scrubber project, initial design, run ideas by machine shop		
4-5	Design power & movement on PCB	Design sensing & control systems on PCB	Complete routing & layout of PCB
6-10	Prepare breadboard demo of MCU outputting PWMs		Revise PCB schematic
11-13	Complete and implement MCU code	Implement and test buck, test motor movement	Solder PCB, test panels, wire everything up
14-15	Verify functionality, complete final demo and presentation		

5. Conclusion

5.1 Summary and Future Work

By the end of the semester in ECE 445, we successfully created an autonomous solar panel cleaner that uses voltage and current sensing to detect a power drop across a solar panel, then deploy a scrubber to those coordinates for cleaning. Although we do not plan to continue work on the Solar Scrubber after the conclusion of the course, there is plenty of room for improvement and things we would approach differently if we did invest time into the project.

Firstly, one of our major problems with our design was the 12 V bus level. In our power module, we designed a LDO (low dropout) regulator to step 24 V down to 12 V. The datasheet for the L7812 regulator we used stated that the maximum input voltage and current was 35 V / 1.5 A (much higher than the 24 V input we had), so we believed that the LDO would be sufficient. However, in the testing of the power system with the brush motor running, the L7812 tended to overheat and fail. For this reason, we would recommend using an integrated circuit buck converter instead of an LDO here in redesign of the project. Buck converters are inherently much more efficient than LDOs, usually with a higher current rating and thermal protection.

Another potential area of improvement lies in the planned sprayer on the cleaning module. In our design, we had failed to consider the current draw of the sprayer we intended to use – in our testing from a DC power supply, it had an inrush current of up to 800 mA. Meanwhile, our design had the sprayer 5 V signal coming from our 3.3 V - 5 V level shifter. The level shifter is only rated to supply 75 mA, so we were not able to safely trigger the sprayer using a logic signal from the MCU sent through the level shifter. Upon redesign, we would likely use a MOSFET switch to switch on and off a connection between the 5 V bus and sprayer; this way, current draw responsibility would solely lie on the 5 V bus (which can handle a much higher current of 3 Amps from the USB C) instead of the level shifter.

Additionally, we noticed that upon running the Solar Scrubber for extended periods of time, the cleaning module would become slightly misaligned. Currently, we use open-loop position control in the ESP32 code, meaning that the MCU only outputs hard-coded time-based signals to the motor controllers. For example, at our configured PWM frequency, we found that sending a PWM for 1 second corresponds to about 2.5 inches of lateral motion. Therefore, if we want the cleaning module to move 7.5 inches, we just send a 4 kHz PWM for 3 seconds. However, the critical flaw with this approach is that the system has no way to tell if the cleaning module is actually where it thinks it is. Upon redesign of the Solar Scrubber, we recommend implementing closed-loop position tracking control to minimize the error of the cleaning module location using position encoders. Additionally, we could implement limit switches (physical or optical) and a startup homing sequence so that the Scrubber can start at the exact same “home” position every time the program is run.

5.2 Ethics

The Solar Scrubber project was developed in accordance with the IEEE Code of Ethics. We placed particular emphasis on Principle 1, which prioritizes the safety, health, and welfare of the public, by automating a maintenance task that would otherwise require workers to climb onto rooftops and operate near energized electrical equipment. Principle 9, which concerns avoiding harm to others and to property, guided our decision to add multiple safety overrides including the emergency stop button, the manual control mode, and the software-enforced motion limits that prevent the cleaning chassis from traveling beyond the panel array. We also adhered to Principle 5, which calls for honest reporting of capabilities and limitations, by clearly documenting in this report the subsystems that were removed during development and the areas where the design fell short of its original specification.

5.3 Safety

Our priority is the safety of our users, the solar panels, and the environment. The system is designed to operate safely in both manual and autonomous modes and is gentle enough to avoid damaging the panel surfaces. Safety measures include an emergency stop button on the PCB, a software-level emergency stop accessible through the serial interface, voltage isolation between the high-current motor rails and the MCU logic, and motion limits enforced in firmware that prevent the cleaning chassis from leaving the defined panel grid.

5.4 Economic Impact

Our objective is to develop an accessible cleaning solution designed for a broad demographic. By focusing on affordability and intuitive operation, this technology remains available to users from a range of backgrounds. The scalable rail system provides the versatility needed to adapt to various installation environments. By extending the effective output of an existing solar array rather than requiring panel replacement, the Solar Scrubber also reduces the long-term cost of solar ownership.

5.5 Environmental Impact

The Solar Scrubber directly improves the environmental return on investment of a solar installation. Soiling can reduce panel output by 20% or more in arid or dusty regions, meaning the array delivers less clean energy than it was manufactured to provide. By restoring panels to near-nominal output, the system increases the lifetime energy yield of the array without manufacturing new hardware. Our final passive cloth design also eliminates water consumption entirely, which is a meaningful advantage in regions where solar deployment overlaps with water scarcity. Compared to scheduled cleaning systems that activate regardless of need, the targeted detection approach further reduces electrical energy consumption from the cleaning process itself.

References

[1] "Electrical Engineering," *The Grainger College of Engineering*, [Online]. Available: https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/electrical-engineering. [Accessed: Feb. 27, 2026].

[2] "CRAFTSMAN 1-Gallon Tank Sprayer," *Lowe's*, [Online]. Available: https://www.lowes.com/pd/CRAFTSMAN-1-Gallon-Tank-Sprayer/1001433038. [Accessed: Feb. 27, 2026].

[3] "IEEE Code of Ethics," *IEEE*, [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: Feb. 27, 2026].

[4] "Standard for User Interface Elements in Power Control of Electronic Devices," *IEEE Standard 1621-2004*, 2004.

[5] "Safety Requirements for Electrical Equipment for Measurement, Control, and Laboratory Use," *UL Standard 61010-1*, 2012.

[6] "Generic Standard on Printed Board Design," *IPC Standard 2221B*, 2012.

[7] "ECE 445 Project Safety Guidelines," *University of Illinois Urbana-Champaign*, [Online]. Available: https://courses.engr.illinois.edu/ece445/safety.asp. [Accessed: Feb. 27, 2026].

[8] "Experiment 4," *University of Illinois ECE 469: Power Electronics Laboratory*, [Online]. Available: https://powerece469.web.illinois.edu/wp/experiment-1/.

Appendix A - Subsystem R&V Tables

Movement

	Requirements	Verification
Structural Load Capacity	The gantry assembly and horizontal rail must maintain structural stability while supporting the total combined mass of the central carriage and cleaning module without mechanical deflection exceeding 2mm.	Secure the horizontal and vertical rails in their final mounting configuration and place a test weight equal to 1.5x the robot's total mass at the center of the longest span. Use a dial indicator or digital calipers to measure the vertical distance from a fixed reference point to the rail before and after the weight is applied to confirm the bend is less than 2mm.
Full-Surface Coverage	The 2-axis drive system must provide a minimum physical travel range to ensure the cleaning module can reach and overlap 100% of the surface area across the three-panel solar array.	Manually command the robot to all four extreme corners of the 3-panel array via the ESP32-S3 control interface. Visually verify that the cleaning brush and microfiber cloth physically reach and overlap the edges of the photovoltaic cells at each extremity.
Coordinate-Based Precision	Utilizing Hall Effect encoders and a startup homing sequence, the locomotion system must achieve a positional repeatability of +/- 5mm when commanded to specific (x, y) coordinates for targeted spot-cleaning.	Mark three specific coordinate targets on the solar panels and command the robot to visit each target five times from different starting positions. Use a ruler to measure the distance between the center of the brush and the target mark to confirm the error is consistently within +/- 5mm.
Temporal Performance	The subsystem must be capable of moving the cleaning module from the home position (0,0) to the furthest diagonal coordinate of the array in under 60 seconds.	Use a stopwatch to time a full diagonal transit from the home position (0,0) to the maximum (x, y) coordinates to ensure the duration is under 60 seconds.
Environmental Resilience	All locomotion components, including DC motors, gears, and	Review component data sheets to confirm DC motors and drivers are

	timing bands, must operate reliably within an outdoor temperature range of 0C to 50C without material degradation or loss of torque. Specifically, the DC motors must not exceed an internal operating temperature of 70C during a continuous 10-minute cleaning cycle to prevent torque degradation.	rated for 50C ambient operation. Perform a continuous 10-minute cleaning cycle in direct sunlight and use an infrared thermometer to verify the motor casing temperature does not exceed 70C while maintaining consistent torque and speed.

MPPT

	Requirements	Verification
Sensing Precision	The sensing circuits must measure individual panel voltages (0 to 12V) and total system current (0 to 2A) with a maximum error of 1% to provide the high-resolution data necessary for the ESP32-S3 to reliably detect efficiency drops.	Apply a known DC voltage from 0V to 12V (Voc) to each of the three voltage sensor inputs and a known current from 0A to 2.5A to the shunt resistor using a calibrated power supply or electronic load. Compare the readings sampled by the ESP32-S3 ADC with a handheld digital multimeter to verify that the measured error for both voltage and current remains below 1%.
MPPT Tracking Efficiency	The Perturb and Observe algorithm implemented on the MCU must maintain a tracking efficiency of at least 95% under steady-state solar irradiance.	Connect a 20W solar panel to the system under stable sunlight and measure the actual power delivered to the load using an external power meter. Compare this value to the theoretical maximum power point calculated from the panel's I-V curve at that moment and verify the tracking efficiency is at least 95%.
PWM Control Resolution	The ESP32-S3 must generate a PWM signal for the buck converter gate driver with a resolution of at least 10 bits and a frequency of 50kHz to allow for fine adjustments to the	Connect an oscilloscope to the ESP32-S3 PWM output pin that drives the buck converter. Use the MCU's console to sweep the duty cycle from 0% to 100% in 1-bit

	operating point.	increments and verify the signal maintains a consistent frequency of 50kHz with distinct, monotonic changes in pulse width.
Fault Isolation and Detection	The algorithm must use the three independent voltage sensor inputs to isolate and identify the specific panel (1, 2, or 3) exhibiting a power drop of 25% or more compared to the others, correctly flagging it as "dirty" for the locomotion subsystem.	Set up all three panels under uniform lighting. Apply a physical obstruction (stain) to Panel 2 only, reducing its individual voltage and power output by approximately 30%. Verify through the serial monitor that the ESP32-S3 correctly identifies Panel 2 as the outlier and triggers the targeted cleaning flag while maintaining normal status for Panels 1 and 3.

Cleaning Module

	Requirements	Verification
Command Responsiveness:	The scrubbing motor and water pump must respond exclusively to MCU logic levels, transitioning from an idle state to active mechanical rotation or fluid flow within 200 ms of the ESP32-S3 signal going high.	Connect an oscilloscope to the 12V output of the DM542T driver and the sprayer pump supply line. Program the ESP32-S3 to toggle the control pins from low to high. Measure the time delta between the logic signal going high and the motor/pump reaching 90% of their operating voltage to confirm the transition occurs within 200 ms.
Effective Cleaning Coverage and Contact:	The gantry-mounted module must maintain uniform pressure against the panel glass across the entire 40-inch horizontal travel path to ensure the rotating brush and microfiber assembly make contact with at least 95% of the total solar panel surface area.	Saturate the entire length of the rotary microfiber towel with water and place the module on the panel array. Manually command the gantry to move across the 40-inch horizontal path while the cleaning mechanism is active. Visually inspect the panel surface to ensure a consistent "wet trail" is left across at least 95% of the surface area, confirming the brush maintains mechanical contact without lifting or skipping.
Environmental and	The subsystem must operate	Operate the brush motor and sprayer

Thermal Tolerance:	reliably in ambient temperatures ranging from 0°C to 50°C. Specifically, the DC motors must not exceed an internal operating temperature of 70°C during a continuous 10-minute cleaning cycle to prevent torque degradation.	pump continuously for 10 minutes in an environment reaching up to 50°C. Use an infrared thermometer or thermocouple to monitor the DC motor casing to ensure it stays below 70°C. Confirm that the motor maintains consistent rotational speed and the pump maintains steady fluid flow throughout the duration of the test.

Sensing & Control:

	Requirement	Verification
I/O Interface and Driver Integration	The controller must successfully sample three analog voltage inputs (IO4–IO6) and one current input (IO7) while simultaneously monitoring the light sensor (IO8, IO9) and generating coordinated Pulse, Direction, and Enable signals for two dual-channel motor drivers (IO38–IO42, IO45).	Using a regulated power supply and a signal generator, apply known analog voltages to pins IO4–IO7 and toggle digital inputs IO8–IO10; verify via the serial monitor that the MCU accurately registers all values within a 2% margin. Simultaneously, use a four-channel oscilloscope to confirm that movement commands produce synchronized, non-interfering signals on the driver pins (IO38–IO45).
Sequential Cleaning Execution	The control logic must execute a mandatory "Spray-then-Scrub" sequence where the 12V water pump is activated for a brief 1-second burst to lubricate the panel before the brush motor begins rotation.	Use a two-channel oscilloscope to monitor the spray (IO11) and brush (IO12) output pins. Trigger a cleaning cycle and verify that the spray signal stays high for exactly 1 second and then turns off before the brush signal is asserted high.
Open-Loop Grid Navigation	The Finite State Machine must utilize a hard-coded coordinate grid and step-counting logic to navigate the gantry across the 40-inch array, ensuring the module can move to any specific panel coordinate from a fixed starting point.	Place the gantry at the designated "Start" position. Command the robot to navigate to three different pre-defined coordinates on the 40-inch array and use a tape measure to verify the gantry stops within 1 inch of the target location.
MPPT Algorithmic	The control subsystem must execute the Perturb and Observe	Use a programmable DC load to simulate a solar panel curve.

Accuracy	algorithm using sampled telemetry to adjust the duty cycle on IO13, ensuring it converges on the maximum power point within 2 seconds of a detected irradiance change.	Introduce a sudden change in simulated irradiance and use an oscilloscope to confirm the PWM signal on IO13 adjusts and stabilizes at the new maximum power point within 2 seconds.
Signal Timing and Stability	The controller must maintain a 50kHz PWM frequency for the MPPT output on pin IO13 and cleaning components on pins IO11 and IO12, with a command-to-execution latency of less than 200 ms.	Use an oscilloscope to measure the time delay between a simulated "dirty" sensor input and the output of the spray enable or brush motor signals. Verify the transition from state detection to output high occurs in under 200 ms.

Power Conversion:

	Requirements	Verification
Dual Input Isolation	The subsystem must successfully distribute a 24 V DC rail from a wall adapter for mechanical loads and a separate 5 V DC rail via USB-C for logic components to prevent motor induced noise from affecting the MCU.	Current should not flow back into either source (USB or wall supply) from the 5V bus when operating at rated load when measured with an oscilloscope current probe. Each DC bus (24 V, 12 V, 5 V, 3.3 V) should have less than a 2% ripple when measured with an oscilloscope.
3.3V Logic Regulation	The subsystem must maintain a 3.3V (±2%) output rail derived from the 5V USB-C input to power the ESP32-S3, sensors, and analog sensing pins.	The subsystem must maintain a 3.3V (±2%) output rail with less than a 2% ripple when probed with an oscilloscope.
Signal Level Shifting	The 5V logic rail must power the SN74AHCT541 buffer to reliably shift 3.3V MCU control signals to the 5V levels required by the motor drivers.	The level shifter must output 5 V with less than a 5% ripple when measured with an oscilloscope.
LDO Thermal Performance	The AP2112K-3.3 regulator must operate without thermal shutdown while the ESP32-S3 is at peak processing load of approximately 240mA.	Monitor the system using a thermal camera at standard operating conditions for 15 minutes. It must not rise above the rated temperature during this time.

Appendix B - ESP32 Code

```
// ======================================================
// Combined MPPT + panel cleaner controller
// ======================================================
// ── MPPT pins / config ──────────────────────────────
const int VSENSE_1_PIN = 4;
const int VSENSE_2_PIN = 5;
const int VSENSE_3_PIN = 6;
const int ISENSE_PIN   = 7;
const int PWM_OUT_PIN  = 13;
const float SCALE_V1   = 4.2;
const float SCALE_V2   = 9.2;
const float SCALE_V3   = 15.0;
const float ADC_VREF   = 3.3f;
const float ADC_MAX    = 4095.0f;
const float ISENSE_VREF = 0.95f;
const float RSENSE     = 0.003f;
const float AMP_GAIN   = 20.0f;
const int DCDC_PWM_FREQ = 50000;
const int DCDC_PWM_RES  = 8;
#define MIN_DUTY   0.05f
#define MAX_DUTY   0.5f
#define STEP_SIZE  0.005f
const unsigned long MPPT_INTERVAL_MS = 200;
static float mpptDuty    = 0.25f;
static float prevPower   = 0.0f;
static int   perturbDir  = 1;
static bool  initialized = false;
unsigned long lastMpptTime = 0;
float lastP1 = 0.0f;
float lastP2 = 0.0f;
// ── Baseline tracking for self-comparison ──────────
float baselineP1 = 0.0f;
float baselineP2 = 0.0f;
const float BASELINE_ALPHA = 0.1f;
bool rebaselineNext = false;
// ── Detection thresholds ────────────────────────────
const float DIRT_RATIO_THRESHOLD = 0.40f;
const float MIN_POWER_FOR_COMPARE = 0.5f;
// ── Idle deep-clean config ──────────────────────────
```

```
const unsigned long IDLE_DEEP_CLEAN_MS = 30000;  // 30 seconds idle -> clean all
panels
unsigned long lastActivityTime = 0;
// ── Dirt detection debouncing ──────────────────────
const int DIRT_CONFIRM_COUNT = 5;
int  panel1Streak = 0;
int  panel2Streak = 0;
float panel1Drop  = 0.0f;
float panel2Drop  = 0.0f;
// ── Recent highs + post-clean verification ─────────
float recentHighP1 = 0.0f;
float recentHighP2 = 0.0f;
const float HIGH_DECAY = 0.995f;
float preDropP1 = 0.0f;
float preDropP2 = 0.0f;
bool  awaitingPostCleanCheck = false;
unsigned long postCleanReadyTime = 0;
const unsigned long POST_CLEAN_WAIT_MS = 10000;
// ── Motor pins / config ────────────────────────────
#define NUM_PANELS 3
const int PWM_PIN   = 38;
const int DIR_PIN   = 39;
const int PWM_PIN_V = 41;
const int DIR_PIN_V = 42;
const int MOTOR_PWM_FREQ = 4000;
const int MOTOR_PWM_RES  = 8;
const int MOTOR_PWM_DUTY = 128;
const float SPEED_IPS = 2.5;
const unsigned long H_TIME_REDUCTION_MS = 500;
// Layout: Panel 1 left, Panel 2 middle, Panel 3 right
const float panelWidth[NUM_PANELS]  = { 13.0, 13.0, 13.0 };
const float panelHeight[NUM_PANELS] = { 22.0, 22.0, 22.0 };
const float panelStartX[NUM_PANELS] = { 0.0, 13.0, 26.0 };
// Rest position: top-left, above Panel 1
const float REST_X = 0.0;
const float REST_Y = 22.0;
const float GRID_MAX_X = 39.0;
const float GRID_MAX_Y = 22.0;
const float MANUAL_STEP = 4.0;
// ── Cleaning wiggle config ─────────────────────────
const float WIGGLE_AMOUNT  = 5.0f;
const int   WIGGLE_SEGMENTS = 6;
```

```
bool dirty[NUM_PANELS] = { false, false, false };
float posX = REST_X;
float posY = REST_Y;
struct Move { bool isH; float target; };
#define MAX_MOVES 150
Move moveQueue[MAX_MOVES];
int queueLen = 0;
int queueIdx = 0;
enum MotorState { IDLE, MOVING, DONE, MANUAL } motorState = IDLE;
unsigned long moveStart, moveDur;
bool movingH;
float moveTarget;

// ── Per-panel voltage tracking ──────────────────────
float lastV1 = 0.0f;
float lastV2 = 0.0f;
float baselineV1 = 0.0f;
float baselineV2 = 0.0f;
float recentHighV1 = 0.0f;
float recentHighV2 = 0.0f;
float preDropV1 = 0.0f;
float preDropV2 = 0.0f;



// ======================================================
// MPPT
// ======================================================
int readADCavg(int pin, int samples = 16) {
uint32_t sum = 0;
for (int i = 0; i < samples; i++) sum += analogRead(pin);
return sum / samples;
}

float readNodeVoltage(int pin, float scale) {
int raw = readADCavg(pin);
return (raw / ADC_MAX) * ADC_VREF * scale;
}

float readCurrent(int &rawOut, float &vOut) {
rawOut = readADCavg(ISENSE_PIN, 32);
vOut   = (rawOut / ADC_MAX) * ISENSE_VREF;
return vOut / (AMP_GAIN * RSENSE);
```

```
}

void setDuty(float duty) {
duty = constrain(duty, MIN_DUTY, MAX_DUTY);
mpptDuty = duty;
uint32_t raw = (uint32_t)(duty * ((1 << DCDC_PWM_RES) - 1));
ledcWrite(PWM_OUT_PIN, raw);
}

void runMPPT() {
float node1 = readNodeVoltage(VSENSE_1_PIN, SCALE_V1);
float node2 = readNodeVoltage(VSENSE_2_PIN, SCALE_V2);

float v1 = node1;
float v2 = node2 - node1;
if (v2 < 0) v2 = 0;
float totalV = v1 + v2;

int   iRaw;
float iVolts;
float i = readCurrent(iRaw, iVolts);

float p1 = v1 * i;
float p2 = v2 * i;
float power = totalV * i;

lastP1 = p1;
lastP2 = p2;
lastV1 = v1;
lastV2 = v2;

// track recent highs (slowly decaying max) — voltage-based
recentHighV1 = max(recentHighV1 * HIGH_DECAY, v1);
recentHighV2 = max(recentHighV2 * HIGH_DECAY, v2);
recentHighP1 = max(recentHighP1 * HIGH_DECAY, p1);
recentHighP2 = max(recentHighP2 * HIGH_DECAY, p2);

if (!initialized) {
 prevPower    = power;
 baselineP1   = p1;
 baselineP2   = p2;
 baselineV1   = v1;
```

```
 baselineV2   = v2;
 recentHighP1 = p1;
 recentHighP2 = p2;
 recentHighV1 = v1;
 recentHighV2 = v2;
 initialized  = true;
 return;
}


if (rebaselineNext) {
 baselineP1     = p1;
 baselineP2     = p2;
 baselineV1     = v1;
 baselineV2     = v2;
 rebaselineNext = false;
 Serial.printf("[REBASE] baseV1=%.2f baseV2=%.2f (fresh start)\n", baselineV1,
baselineV2);
}

if (power > prevPower) {
 mpptDuty += perturbDir * STEP_SIZE;
} else if (power < prevPower) {
 perturbDir = -perturbDir;
 mpptDuty  += perturbDir * STEP_SIZE;
}

mpptDuty  = constrain(mpptDuty, MIN_DUTY, MAX_DUTY);
prevPower = power;
setDuty(mpptDuty);


Serial.printf("[MPPT] V1=%.2f V2=%.2f Vtot=%.2f | I=%.4f | P1=%.2f P2=%.2f Ptot=%.2f |
duty=%.3f\n",
             v1, v2, totalV, i, p1, p2, power, mpptDuty);
Serial.printf("[BASE] baseV1=%.2f baseV2=%.2f hiV1=%.2f hiV2=%.2f\n",
             baselineV1, baselineV2, recentHighV1, recentHighV2);
}


// ================================================
// Self-comparison dirt detection
```

```
// ====================================================

void resetStreaks() {
panel1Streak = 0;
panel2Streak = 0;
panel1Drop   = 0.0f;
panel2Drop   = 0.0f;
}


void detectDirtyPanels(int outIdx[NUM_PANELS], int &outCount) {
 outCount = 0;

 if (rebaselineNext) {
   resetStreaks();
   return;
 }

 float drop1 = (baselineV1 > 0.5f) ? (baselineV1 - lastV1) / baselineV1 : 0.0f;
 float drop2 = (baselineV2 > 0.5f) ? (baselineV2 - lastV2) / baselineV2 : 0.0f;

 bool p1Dirty = (drop1 >= DIRT_RATIO_THRESHOLD);
 bool p2Dirty = (drop2 >= DIRT_RATIO_THRESHOLD);

 if (p1Dirty) {
   if (panel1Streak == 0) preDropV1 = recentHighV1;
   panel1Streak++;
   panel1Drop = drop1;
 } else {
   panel1Streak = 0;
   baselineV1 = (1.0f - BASELINE_ALPHA) * baselineV1 + BASELINE_ALPHA * lastV1;
 }

 if (p2Dirty) {
   if (panel2Streak == 0) preDropV2 = recentHighV2;
   panel2Streak++;
   panel2Drop = drop2;
 } else {
   panel2Streak = 0;
   baselineV2 = (1.0f - BASELINE_ALPHA) * baselineV2 + BASELINE_ALPHA * lastV2;
 }
```

```
 if (p1Dirty || p2Dirty) {
   Serial.printf("[DETECT-V] V1=%.2f vs %.2f drop=%.1f%% (%d/%d) | V2=%.2f vs %.2f
drop=%.1f%% (%d/%d)\n",
                 lastV1, baselineV1, drop1 * 100.0f, panel1Streak, DIRT_CONFIRM_COUNT,
                 lastV2, baselineV2, drop2 * 100.0f, panel2Streak,
DIRT_CONFIRM_COUNT);
 }

 bool p1Confirmed = (panel1Streak >= DIRT_CONFIRM_COUNT);
 bool p2Confirmed = (panel2Streak >= DIRT_CONFIRM_COUNT);

 if (!p1Confirmed && !p2Confirmed) return;

 if (p1Confirmed && p2Confirmed) {
   if (panel1Drop >= panel2Drop) {
     outIdx[0] = 0;
     outIdx[1] = 1;
   } else {
     outIdx[0] = 1;
     outIdx[1] = 0;
   }
   outCount = 2;
 } else if (p1Confirmed) {
   outIdx[0] = 0;
   outCount = 1;
 } else {
   outIdx[0] = 1;
   outCount = 1;
 }
}

// ========================================================
// Post-clean verification
// ========================================================

void verifyPostClean(int outIdx[NUM_PANELS], int &outCount) {
 outCount = 0;

 if (!awaitingPostCleanCheck) return;
 if (millis() < postCleanReadyTime) return;

 awaitingPostCleanCheck = false;
```

```
 float postDrop1 = (preDropV1 > 0.5f) ? (preDropV1 - lastV1) / preDropV1 : 0.0f;
 float postDrop2 = (preDropV2 > 0.5f) ? (preDropV2 - lastV2) / preDropV2 : 0.0f;

 Serial.printf("[POSTCHK-V] vs pre-drop: V1 %.2f->%.2f (%.1f%%) | V2 %.2f->%.2f
(%.1f%%)\n",
               preDropV1, lastV1, postDrop1 * 100.0f,
               preDropV2, lastV2, postDrop2 * 100.0f);

 bool p1StillBad = (postDrop1 >= DIRT_RATIO_THRESHOLD);
 bool p2StillBad = (postDrop2 >= DIRT_RATIO_THRESHOLD);

 if (!p1StillBad && !p2StillBad) {
   Serial.println("[POSTCHK] Cleaning successful, panels recovered.");
   return;
 }

 Serial.println("[POSTCHK] Cleaning incomplete, queuing another cycle.");

 if (p1StillBad && p2StillBad) {
   if (postDrop1 >= postDrop2) {
     outIdx[0] = 0; outIdx[1] = 1;
   } else {
     outIdx[0] = 1; outIdx[1] = 0;
   }
   outCount = 2;
 } else if (p1StillBad) {
   outIdx[0] = 0; outCount = 1;
 } else {
   outIdx[0] = 1; outCount = 1;
 }
}




// ======================================================
// Motor / cleaning path
// ======================================================
```

```
void enqueue(bool isH, float target) {
if (queueLen < MAX_MOVES)
 moveQueue[queueLen++] = { isH, target };
}




float clampX(float x) {
if (x < 0)          return 0;
if (x > GRID_MAX_X) return GRID_MAX_X;
return x;
}




float clampY(float y) {
if (y < 0)          return 0;
if (y > GRID_MAX_Y) return GRID_MAX_Y;
return y;
}




void buildCleanPath(int cleanOrder[], int orderCount) {
queueLen = 0;
queueIdx = 0;
float bx = posX, by = posY;




for (int k = 0; k < orderCount; k++) {
 int p = cleanOrder[k];



 float panelX = panelStartX[p];
```

```
float topY   = panelHeight[p];




// step 1: move horizontally first
if (abs(bx - panelX) > 0.01f) {
  enqueue(true, panelX);
  bx = panelX;
}




// step 2: vertical sweep with wiggles
float startY  = by;
float endY    = (by >= topY / 2.0f) ? 0.0f : topY;
float segStep = (endY - startY) / WIGGLE_SEGMENTS;




for (int s = 0; s < WIGGLE_SEGMENTS; s++) {
  float ySeg = startY + segStep * (s + 1);
  enqueue(false, ySeg);




  // wiggle right only, except on last segment
  if (s < WIGGLE_SEGMENTS - 1) {
    enqueue(true, bx + WIGGLE_AMOUNT);
    enqueue(true, bx);
  }
}
by = endY;




Serial.print("[CLEAN] Queuing panel "); Serial.println(p + 1);
}
```

```
// return to rest: UP first, then LEFT
enqueue(false, REST_Y);
enqueue(true,  REST_X);
}



void executeNextMove() {
while (queueIdx < queueLen) {
 Move  m    = moveQueue[queueIdx];
 float cur  = m.isH ? posX : posY;
 float dist = abs(m.target - cur);




 if (dist < 0.01f) {
   if (m.isH) posX = m.target;
   else       posY = m.target;
   queueIdx++;
   continue;
 }



 movingH    = m.isH;
 moveTarget = m.target;



 unsigned long rawDur = (unsigned long)(dist / SPEED_IPS * 1000.0f);
 if (m.isH && rawDur > H_TIME_REDUCTION_MS) {
   rawDur -= H_TIME_REDUCTION_MS;
 }
 moveDur   = rawDur;
```

```
 moveStart = millis();




 if (m.isH) {
   digitalWrite(DIR_PIN, m.target > posX ? LOW : HIGH);
   ledcWrite(PWM_PIN, MOTOR_PWM_DUTY);
 } else {
   digitalWrite(DIR_PIN_V, m.target > posY ? HIGH : LOW);
   ledcWrite(PWM_PIN_V, MOTOR_PWM_DUTY);
 }




 motorState = MOVING;
 Serial.print(m.isH ? "[CLEAN] H -> " : "[CLEAN] V -> ");
 Serial.println(m.target, 2);
 return;
}



motorState = IDLE;
Serial.println("[CLEAN] Done. Back at rest.");
for (int i = 0; i < NUM_PANELS; i++) dirty[i] = false;
resetStreaks();
rebaselineNext = true;
lastActivityTime = millis();  // restart idle timer after each clean



// arm post-clean verification
awaitingPostCleanCheck = true;
postCleanReadyTime = millis() + POST_CLEAN_WAIT_MS;
Serial.printf("[POSTCHK] Will verify in %lu seconds...\n", POST_CLEAN_WAIT_MS / 1000);
}
```

```
void runMotor() {
if (motorState != MOVING) return;




if (millis() - moveStart >= moveDur) {
 if (movingH) { ledcWrite(PWM_PIN,   0); posX = moveTarget; }
 else         { ledcWrite(PWM_PIN_V, 0); posY = moveTarget; }
 queueIdx++;
 executeNextMove();
}
}




// ======================================================
// Top-level controller
// ======================================================




void maybeStartCleaning() {
if (motorState != IDLE) return;




int cleanOrder[NUM_PANELS];
int count = 0;




// first try post-clean verification
verifyPostClean(cleanOrder, count);
```

```
// if no post-clean issue, run normal detection
if (count == 0) {
 detectDirtyPanels(cleanOrder, count);
}




// No dirty panels detected — check if idle long enough for a deep clean
if (count == 0) {
 if (millis() - lastActivityTime >= IDLE_DEEP_CLEAN_MS) {
   Serial.println("[CTRL] Idle timeout reached -> deep clean (all panels)");
   for (int i = 0; i < NUM_PANELS; i++) cleanOrder[i] = i;
   count = NUM_PANELS;
   lastActivityTime = millis();  // reset so we don't immediately re-trigger
 } else {
   return;
 }
} else {
 // Real dirt detected — reset idle timer
 lastActivityTime = millis();
}




for (int i = 0; i < NUM_PANELS; i++) dirty[i] = false;
for (int k = 0; k < count; k++) dirty[cleanOrder[k]] = true;




Serial.print("[CTRL] Cleaning order: ");
for (int k = 0; k < count; k++) {
 Serial.print("Panel ");
 Serial.print(cleanOrder[k] + 1);
 if (k < count - 1) Serial.print(" -> ");
}
Serial.println();
```

```
// wipe streak state so leftover counts don't carry into the next cycle
resetStreaks();



buildCleanPath(cleanOrder, count);
executeNextMove();
}



// ======================================================
// Manual WASD control
// ======================================================



void manualMove(bool isH, float target) {
target = isH ? clampX(target) : clampY(target);
float cur  = isH ? posX : posY;
float dist = abs(target - cur);
if (dist < 0.01f) return;



unsigned long dur = (unsigned long)(dist / SPEED_IPS * 1000.0f);
if (isH && dur > H_TIME_REDUCTION_MS) dur -= H_TIME_REDUCTION_MS;



if (isH) {
 digitalWrite(DIR_PIN, target > posX ? LOW : HIGH);
 ledcWrite(PWM_PIN, MOTOR_PWM_DUTY);
 delay(dur);
```

```
 ledcWrite(PWM_PIN, 0);
 posX = target;
} else {
 digitalWrite(DIR_PIN_V, target > posY ? HIGH : LOW);
 ledcWrite(PWM_PIN_V, MOTOR_PWM_DUTY);
 delay(dur);
 ledcWrite(PWM_PIN_V, 0);
 posY = target;
}




Serial.printf("[MANUAL] Pos: (%.2f, %.2f)\n", posX, posY);
}




void handleSerialInput() {
if (!Serial.available()) return;
char c = Serial.read();
if (c == '\n' || c == '\r') return;




switch (c) {
 case 'm': case 'M':
   if (motorState == MANUAL) {
     Serial.println("[MANUAL] Exiting manual mode. MPPT resuming.");
     motorState = IDLE;
     resetStreaks();
     lastActivityTime = millis();  // reset idle timer after exiting manual
   } else {
     Serial.println("[MANUAL] Entering manual mode. WASD = move, X = stop, M =
exit.");
     ledcWrite(PWM_PIN, 0);
     ledcWrite(PWM_PIN_V, 0);
     motorState = MANUAL;
   }
   break;
```

```
 case 'w': case 'W':
   if (motorState == MANUAL) { Serial.println("[MANUAL] UP");    manualMove(false,
posY + MANUAL_STEP); }
   break;
 case 's': case 'S':
   if (motorState == MANUAL) { Serial.println("[MANUAL] DOWN");  manualMove(false,
posY - MANUAL_STEP); }
   break;
 case 'a': case 'A':
   if (motorState == MANUAL) { Serial.println("[MANUAL] LEFT");  manualMove(true,
posX - MANUAL_STEP); }
   break;
 case 'd': case 'D':
   if (motorState == MANUAL) { Serial.println("[MANUAL] RIGHT"); manualMove(true,
posX + MANUAL_STEP); }
   break;



 case 'x': case 'X':
   Serial.println("[MANUAL] STOP");
   ledcWrite(PWM_PIN, 0);
   ledcWrite(PWM_PIN_V, 0);
   break;



 case 'p': case 'P':
   Serial.printf("[MANUAL] Pos: (%.2f, %.2f)\n", posX, posY);
   break;



 case 'b': case 'B':
   Serial.println("[BASELINE] Manual rebase requested");
```

```
    rebaselineNext = true;
    resetStreaks();
    break;




 case 'h': case 'H':
    Serial.println("Commands:");
    Serial.println("  M = toggle manual mode");
    Serial.println("  WASD = move (manual mode only)");
    Serial.println("  X = stop motors");
    Serial.println("  P = print position");
    Serial.println("  B = force baseline reset on next reading");
    Serial.println("  H = help");
    break;
}
}




// ======================================================
// Arduino entry points
// ======================================================




void setup() {
Serial.begin(115200);
delay(500);




pinMode(VSENSE_1_PIN, INPUT);
pinMode(VSENSE_2_PIN, INPUT);
pinMode(VSENSE_3_PIN, INPUT);
pinMode(ISENSE_PIN,   INPUT);
```

```
analogReadResolution(12);
analogSetPinAttenuation(ISENSE_PIN, ADC_0db);


bool ok = ledcAttach(PWM_OUT_PIN, DCDC_PWM_FREQ, DCDC_PWM_RES);
Serial.printf("[MPPT] ledcAttach: %s\n", ok ? "OK" : "FAILED");
setDuty(MAX_DUTY);




pinMode(DIR_PIN,   OUTPUT);
pinMode(DIR_PIN_V, OUTPUT);




bool okH = ledcAttach(PWM_PIN,   MOTOR_PWM_FREQ, MOTOR_PWM_RES);
bool okV = ledcAttach(PWM_PIN_V, MOTOR_PWM_FREQ, MOTOR_PWM_RES);
Serial.printf("[CLEAN] ledcAttach H:%s V:%s\n",
              okH ? "OK" : "FAILED", okV ? "OK" : "FAILED");
ledcWrite(PWM_PIN,   0);
ledcWrite(PWM_PIN_V, 0);




lastActivityTime = millis();
motorState = IDLE;
Serial.println("[CTRL] Idle. Monitoring power... Press H for commands.");
}


void loop() {
handleSerialInput();

unsigned long now = millis();

if (motorState == IDLE && now - lastMpptTime >= MPPT_INTERVAL_MS) {
 lastMpptTime = now;
```

```
 runMPPT();
 maybeStartCleaning();
}
runMotor();
}
```

Appendix C - Bill of Materials

The full bill of materials, including market value and actual cash cost for each component, is given in Table 6.

Table 6 Bill of Materials

Description	**Manufacturer**	**Part #**	**Qty**	**Market Value**	**Actual Cash Cost**
Machine Shop Labor	UIUC ECE Shop	Services	-	$500.00 (10 hrs)	$100.00 (2 hrs)
Solar Panel (20W)	Solar Power Ind.	SPI-020M-9.5	3	$150.00	$0.00 (Reused)
Microcontroller	Espressif	ESP32-S3-WROOM-1	1	$15.00	$15.00
DC Motor Driver	STMicroelectronics	DM542T	2	$12.00	$12.00
Level Shifter	Texas Instruments	SN74AHCT541	1	$1.50	$1.50
12V DC Motor	Generic	N/A	4	$60.00	$0.00 (Reused)
LDO Regulator (3.3V)	Diodes Inc	AP2112K-3.3	2	$2.00	$2.00
Belt & Pulleys	Generic	GT2	1	$30.00	$0.00 (Reused)
Extrusion Frame	80/20 Inc	10 Series	1	$80.00	$0.00 (Reused)
Diodes for 5V rail protection	Generic	Generic	2	$0.41	$0.41

Bootstrap diode	Generic	Generic	1	$0.2	$0.2
Level shifter	TI	74AHCT541	1	$0.75	$0.75
5->3.3V LDO	Diodes Inc	AP2112K-3.3	1	$0.32	$0.32
Current sense op amp	TI	INA241A1	1	$3.35	$3.35
Voltage sense op amp	TI	OPA376	3	$4.07	$4.07
Gate driver	Infineon	IRS2181	1	$1.97	$1.97
Current sense resistor	Generic	KRL1220E-M-R010-F-T5	1	$0.28	$0.28
USB C receptacle	Generic	10155435-00011 LF	1	$0.83	$0.83
Barrel jack	Generic	54-00167	1	$0.61	$0.61
8x1 screw terminal	Phoenix Contact	1730450	1	$6.58	$6.58
2x1 screw terminal	Phoenix Contact	1890963	2	$1.41	$1.41
4x1 screw terminal	Phoenix Contact	1888700	2	$3.1	$3.1

Buck converter FET	Infineon	IRF6678	2	$4.6	
Total Parts/Shop				**$894.19**	**$159.19**