Optimized Plant-Watering System

By

Aashish Chaubal

Jaeren Dadivas

Iker Uriarte

Final Report for ECE 445, Senior Design, Spring 2025

TA: Maanas Agrawal

7 May 2025

Project No. 75

Abstract

Finding the perfect balance between optimal plant hydration and efficient water usage is an age-old agricultural challenge that still remains in modern agriculture, especially in the face of water conservation and environmental sustainability. Our project presents an Optimized Plant-Watering System, an automated, sensor-driven modular design that facilitates the balance between optimal plant-watering and water usage. The design integrates both soil moisture sensors, real-time rain forecasting, and a water pump actuation system to make logic-based watering decisions. At the core of the design is an ESP32 microcontroller that manages sensor input, fetches an API to AccuWeather, activates a submersible pump when necessary, and generates a website that visualizes system data for remote monitoring and analysis. The website also features a customizable UI, enabling the user to adjust watering thresholds specific to their plant. Our system promotes water conservation while maintaining plant hydration, making it an excellent product from home gardening to scalable agriculture.

Contents

1. Introduction

1.1 Project Purpose

The challenge of managing water resources effectively and efficiently has existed since the beginning of farming and still remains in modern agriculture and home gardening. [1] Factors such as inconsistent rainfall and inefficient watering practices lead to water waste and especially suboptimal plant growth. [2] Our project tackles this problem and addresses the need for an automated watering system that can best make decisions to balance plant-watering with water conservation. Our Optimized Plant-Watering System provides a logic and data-based approach to irrigation by utilizing soil moisture sensors, a pump actuation system, and real-time weather forecasting to make autonomous watering decisions.

1.2 Functionality

At the core of the system is an ESP32 microcontroller which handles sensor data, fetches real-time rain probability via AccuWeather API, and activates a DC water pump when plant-watering is necessary. The ESP32 also supports transparency and usability by visualizing system data to a generated website, allowing the user to both monitor and analyze the system. Furthermore, the website features a customizable UI that enables the user to set soil moisture thresholds based on the specific needs of the subject plant. The central portion of the design (microcontroller, power subsystem, and device peripherals) is housed in a casing with sensors extending to the plant soil and a motor pump extending into a water container that further connects to the soil of a plant. This design makes our system suitable for outdoor usage in home gardening applications and is scalable for possible large-scale agricultural applications.

1.3 Subsystem Overview

Our design consists of five subsystems as visualized in Fig. 1: the power subsystem, sensor subsystem, pump actuation subsystem, control system, and the external subsystem. The power subsystem is responsible for converting the initial 9V voltage from our battery into 3.3V which is required to power the rest of the electronics in our system (except for the actuation subsystem which utilizes the full 9V). Our sensor subsystem consists of the soil moisture sensor and is crucial for collecting the real-time soil moisture level relaying the information back to the microcontroller. The actuation system consists of the water pump and a MOSFET transistor which acts as a switch, allowing the pump activity to be controlled. The control subsystem is the main brain of the system and consists of the ESP32 microcontroller which holds a couple of responsibilities: 1) Generating and sending visualized information to a website allowing the user to input specified parameters. 2) Connecting to WiFi to fetch rain probability data from an AccuWeather API and using that information in conjunction with the collected sensor data to handle logic-based watering decisions.

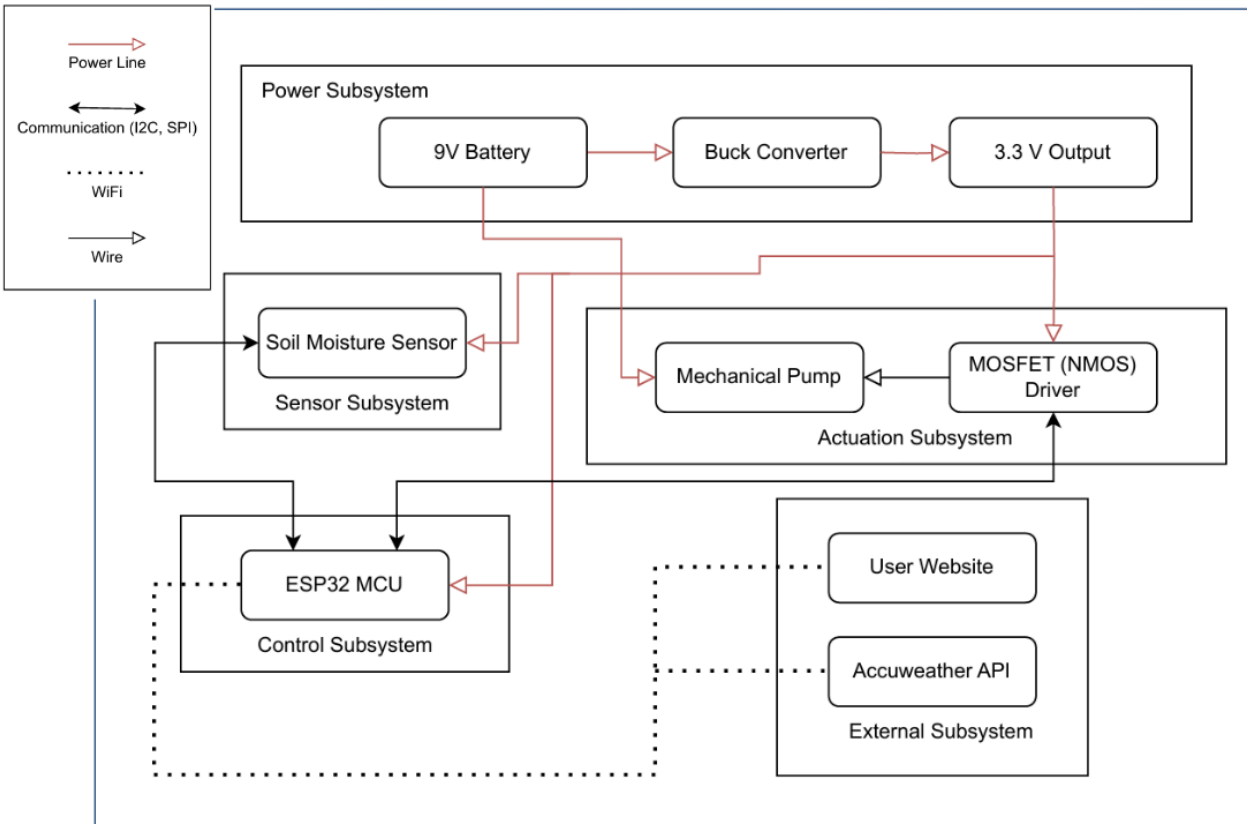

Fig. 1: Top-level block diagram of design

Ultimately, the results of our project demonstrate that the system is capable of automating plant watering while optimizing water usage by using logic-based decisions in response to environmental sensors and real-time weather data. Our design is adaptable, with the potential to be applied in agriculture, residential gardening, and even research environments.

2 Design

2.1 Design Procedure

Early brainstorming produced an expansive feature set—rechargeable lithium pack, on-board rain gauge, SD logging, and cloud analytics—but an initial breadboard revealed that some of those additions complicated the user story and stretched our debugging bandwidth without improving the core mission of "water only when needed." We therefore re-examined every block and chose the alternatives that best satisfied the three objectives stated above.

Power architecture. A single-cell Li-ion with an LDO looked attractive on paper, yet required charge-management, under-temperature lock-out, and cell balancing to be safe. By selecting a 9 V alkaline battery we gained *plug-and-play replaceability* and eliminated all charging circuitry. Because the team had prior experience with switch-mode supplies, we paired the battery with a buck converter rather than an LDO. The buck keeps efficiency above 80 % even when the pump is pulling 300 mA, and its evaluation module drops straight into the PCB with only four passive components—an implementation we could prototype and verify in a single afternoon.

Sensing strategy. A capacitive soil-moisture probe stayed in the design because it directly measures the plant's need for water and its signal chain is immune to galvanic corrosion. By contrast, the load-cell rain cup that featured in our first revision proved temperamental outdoors; wind buffeted the cup, sunlight warped the printed mount, and the HX711 amplifier needed per-degree temperature compensation. Because our target user cares about *whether* it is going to rain and not *how many millimetres* have already fallen, we retired the load-cell assembly and instead fetch the one-hour precipitation probability from AccuWeather. The forecast is both simpler to integrate and more relevant to irrigation timing.

Data handling. The prototype's earliest firmware wrote every reading to an SD card and mirrored the file set to Firebase. Field tests, however, showed that the home gardener rarely reviews historical logs but always appreciates instant feedback. We therefore substituted a Wi-Fi soft-AP and an on-board dashboard for the SD-plus-cloud stack. The user now connects with any phone, sees live moisture and battery data, and adjusts thresholds without creating an account or dealing with privacy settings. Eliminating the SD socket also removed ESD concerns and freed an SPI peripheral for future expansion.

These intentional pivots left us with the streamline architecture of Figure 1: a power block that converts 9 V to 3.3 V, a moisture-sensor block, an ESP32 control core, a MOSFET-switched micro-pump, and a Wi-Fi-based user interface. The next section drills into each block and shows how the numbers close.

2.2 Design Details

2.2.1 Power Subsystem

The power subsystem provides stable operating voltages to all components of the Optimized Plant-Watering System. A single 9V battery serves as the primary power source. The water pump will operate at the approximate 9V. Simultaneously, the ESP32 microcontroller and sensors all operate at 3.3V [3][4], so a buck converter is used to drop the 9V input voltage to provide a stable 3.3V output. Decoupling capacitors and similar elements help maintain low ripple and protect against voltage spikes, especially when the pump switches on or off.

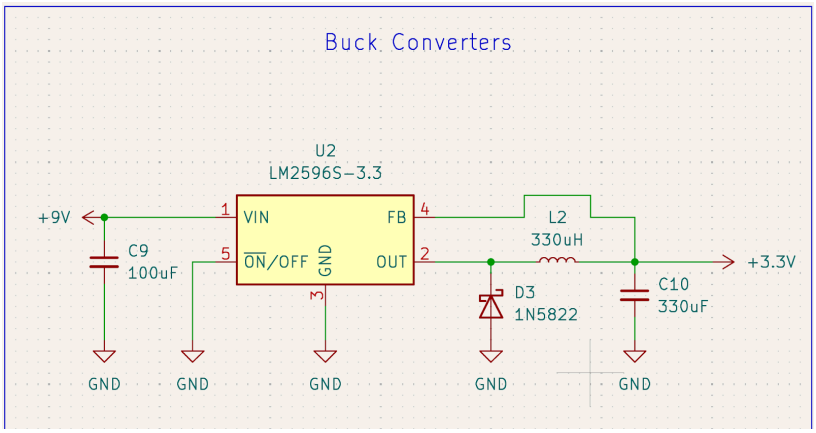

Fig. 2: Schematic of power subsystem

2.2.2 Sensor Subsystem

The sensor subsystem collects soil moisture level data that the system uses to make watering decisions. A resistive soil moisture sensor (SEN0114) is inserted near the plant's root zone and powered at 3.3V, outputting an analog voltage proportional to the volumetric water content. Calibration involves recording sensor output in both dry and fully saturated soil, then mapping the intermediate voltages to a 0-100% moisture scale. This high resolution measurement (±1 g or better) allows the system to detect even slight rainfall, prompting it to delay or cancel watering events if enough water accumulates. The sensor readings are fed in the ESP32's ADC (analog to digital converter) or digital input pins at scheduled intervals.

2.2.3 Actuation Subsystem

The actuation subsystem handles the physical water delivery through a DC pump and tubing extending from the system to the plant's soil. The pump, typically rated for at least 5V, draws current through a dedicated transistor that isolates high current loads from the microcontroller's GPIO pins. [7] The ESP32 sends digital control signals to the transistor, turning the pump on or off according to sensor feedback and external weather data. The system is robust enough to

handle rapid on/off cycling while maintaining system stability and ensuring minimal water leakage or backflow while the pump is inactive.

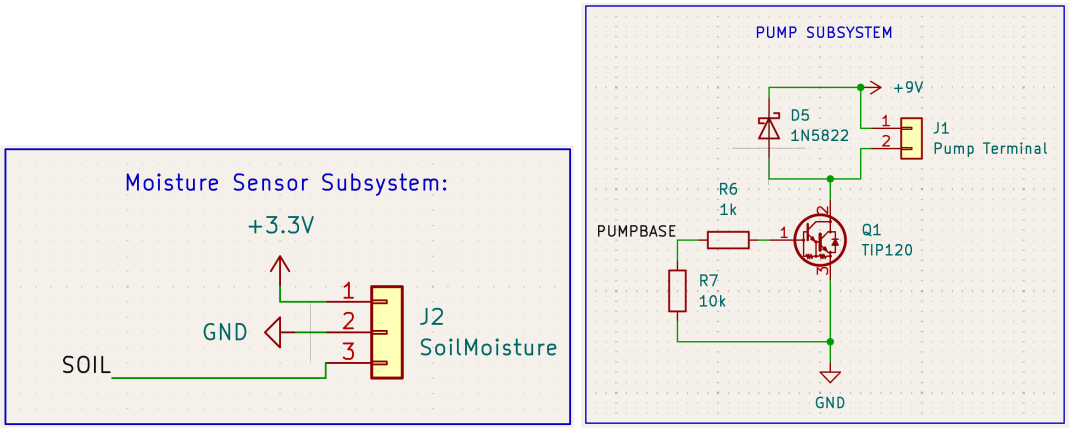

Fig. 3: Schematic of sensor subsystem and actuation subsystem

2.2.4 Control Subsystem

The control subsystem consists mainly of the ESP32 microcontroller. The ESP32 orchestrates sensor polling, data processing, and pump activation, all while managing WiFi connectivity to retrieve weather forecasts and upload sensor logs to an external website. The ESP32 also reads battery voltage from the power subsystem, logs system events (pump activity, rain probability percentage, soil moisture levels), and provides real-time or scheduled updates to the external user interface. This control unit thus forms the brain of the system, integrating information from all other subsystems to execute efficient and automated plant-watering decisions. The ESP32 also requires a programming header setup in order for the microcontroller to be programmed. This setup includes a CH340G-based USB-to-Serial programming header mounted on the PCB which enables serial communication between a computer (Arduino IDE for programming) and the ESP32 via a Micro-USB B connector.

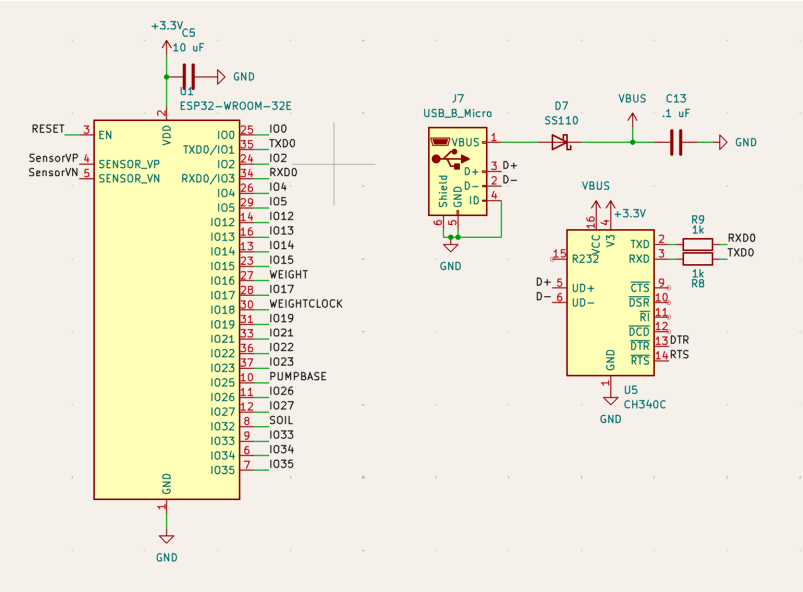

Fig. 4: ESP32 microcontroller and programming header

2.2.5 External Subsystem

The external subsystem manages the off-board communication and user interactions beyond the immediate control subsystem. It involves a WiFi connection from the ESP32 to a user web interface. The ESP32 periodically requests real-time weather information from an AccuWeather API, parsing rainfall probability to refine watering decisions. The system also uploads sensor readings and log entries to the website, allowing the user to monitor soil moisture, rain probability, pump status, and configurable thresholds and other parameters (as visualized in Fig. 5).

Fig. 5: Website user interface including rain probability, pump status, soil moisture, customizable water settings, and visualized system data

3. Design Verification

3.1 Power Subsystem

Table 1.1: Power Supply Subsystem - Requirements & Verification

Requirements	Verification
1. The subsystem must provide a stable 9 V (±5%) for the pump under peak load conditions.	1. We connected the boost converter output to an oscilloscope with a programmable load simulating the pump's maximum 800 mA current. 2. Verified with a voltmeter that the output stays between 8.6 V and 9 V at full load. 3. Measured ripple to confirm it did not exceed 100 mV p-p.
2. The 3.3 V rail must remain within ±3% of nominal to power the ESP32 and sensors.	1. Supplied the Buck Converter from the battery. 2. Measured output voltage at no load and at ~300 mA load with an oscilloscope. 3. Successfully ensured voltage remained in the 3.20-3.40 V range during normal operation.
4. The ESP32 must be able to go to sleep mode when not active to extend the battery's life cycle.	1. Ran the system on a new 9V battery. 2. Recorded time to low-voltage alert. 3. Verified operation for 24 hours, and presented on a graph.

3.2 Sensor Subsystem

Table 1.2: Sensor Subsystem - Requirements & Verification

Requirements	Verification
1. Soil moisture sensors must measure volumetric water content within ±5% accuracy across 0-100% range.	1. We inserted the sensor into dry soil and noted ADC reading. 2. Saturated soil fully and noted ADC reading. 3. Tested intermediate moisture levels (25%, 50%, 75%) using a weigh-and-water method. 4. Confirmed calibration curve yields ±5% accuracy across repeated trials (n=10).
2. Load cell + HX711 must detect rainfall weight changes at ±1 g resolution.	1. Attempted to calibrate by placing known masses (1 g, 5 g, 10 g) in the rain cup. 2. Attempted to record ADC output and verify it distinguishes each weight within ±1 g, however read incorrect readings. This requirement failed.
3. Sensor data must remain stable (±2% drift) over 24 hours in static conditions.	1. Kept the soil sensor in a controlled environment (constant moisture) for 24 hours. 2. Logged sensor data periodically (every minute) and confirmed drift remained under ±2%.
4. Sensor power consumption must be minimized to support battery longevity.	1. Measured current draw when sensors were actively powered vs. switched off by the ESP32. 2. Successfully confirmed that sensor off-state current was below 1 mA.

3.3 Control Subsystem

Table 1.3: Control Unit Subsystem - Requirements & Verification

Requirements	Verification
1. The ESP32 must poll sensors at configurable intervals and log data locally if Wi-Fi fails.	1. Disabled Wi-Fi. 2. Verified that the ESP32 continued to read sensors. 3. Re-enabled Wi-Fi and confirmed that any missed data was eventually uploaded to the website, which was successful (we omitted SD card).
2. The website/sd card module must reliably store data without corruption.	1. Operated the system for 24 hours, logging data every ~15 minutes. 2. Inspected the website contents for missing or malformed entries, and there were none.
3. The control unit must provide a user-visible status (display or button feedback) for at least one system event.	1. Configured a live graph of pump activation to show the user. 2. Observed the pump activation graph over a 24 hour period and confirmed user visibility.
4. The MCU must avoid brownouts or resets when the pump activates.	1. Monitored the 3.3 V rail with an oscilloscope while the pump was switched on. 2. Ensured voltage drop was <5% of nominal and the ESP32 did not reset or lock up.

3.4 Actuation Subsystem

Table 1.4: Actuation Subsystem - Requirements & Verification

Requirements	Verification
1. The pump must deliver a stable flow rate (±10%) at 5 V under typical loads.	1. Supplied the pump at 9 V directly 2. Measured water flow into a graduated container over 30 s. 3. Repeated multiple times (n=10) to ensure flow rate was within ±10% of the expected value.
2. The subsystem must minimize water leakage or backflow when the pump is off.	1. Pressurized the tubing by running the pump for 10 s. 2. Deactivated the pump and observed if water continued to flow. 3. No leakage occurred and we checked valves and ensured solenoid valves were fully sealed in the closed state, which they were.
3. The driver circuit must tolerate pump inrush current without damaging components or resetting the MCU.	1. Monitored MOSFET gate and drain voltages during pump startup using an oscilloscope. 2. Confirmed the inrush current remains within MOSFET and wiring limits. 3. Checked that the ESP32 supply did not experience excessive voltage sag leading to brownouts, which it did not.

3.5 External Subsystem

Table 2.1: External Subsystem - Requirements & Verification

Requirements	Verification
1. The system must fetch weather data from an API at least every 10 minutes (or a user defined interval).	1. Connected the ESP32 to a known Wi-Fi network. 2. Logged timestamps of each successful API call. 3. Verified that the average interval between calls matches the configured schedule (±1 min).
2. Sensor data must be uploaded to the website whenever Wi-Fi is available.	1. Simulated a temporary network outage, allowing data to accumulate. 2. Reconnected to Wi-Fi and confirmed that all pending sensor logs were successfully uploaded.
3. The user must be able to view moisture levels, rain data, via a web interface.	1. Accessed the system's web dashboard from a browser or mobile device. 2. Confirmed that displayed data (moisture, rainfall, battery voltage) matched real time or recently logged values.

4. The subsystem must allow user-defined parameters (like moisture thresholds) to be updated remotely.	1. Implemented a user input page on the web dashboard that writes updated thresholds. 2. Verified that the ESP32 retrieves and applies these new values within a specified period (60 s).

4. Costs and Schedule

The total cost of the project can be seen in Table 4. Subsection 4.1 and 4.2 digs deeper into cost analysis.

Table 4.1: Cost Analysis

Category	Cost
Labor Hours	$56,700
Components	$84.71
Total	**$56,784.71**

4.1 Parts

Table 4.2: Individual Cost

Part	Vendor	Retail Cost ($)	Quantity	Total Cost ($)
LM2596S	DigiKey	7.89	2	15.78
ESP32 WROOM 32	DigiKey	4.36	3	13.08
1N5822	DigiKey	0.30	4	1.20
BD139	DigiKey	0.61	**7**	4.27
TIP120	DigiKey	1.05	2	2.10
CH340C	SparkFun	9.45	2	18.90
HX711	SparkFun	9.86	2	19.72
10k Resistor	ECE SELF SERVICE	0.10	20	0.00
1k Resistor	ECE SELF SERVICE	0.10	20	0.00
100uF Capacitor	ECE SELF SERVICE	0.11	20	0.00
10uF Capacitor	ECE SELF SERVICE	0.11	20	0.00
200mH Inductor	ECE SHOP	3.22	3	9.66
Total	**-**	**-**	-	**$84.71**

4.2 Labor

All the members of the group are Electrical Engineering students. Based on the Grainger College of Engineering, the average starting salary for an electrical engineer major graduating from UIUC is $88,321/yr, which is approximately $42/hr.

Assuming that we worked on this project for a grand total of 180 hours, approximately nine weeks of 20 hours, the labor cost can be approximated to be around $7500 per person. Thus, the total labor cost of the entire project is as calculated below:

$$\$42/hr \ * \ 3\ team\ members * 2.5 * 180\ hours = \$56,700$$

In addition to the cost of components, the grand total cost of this whole project is approximately $56,784.71.

4.3 Schedule

Table 4.3: Semester-Long Project Schedule

Week	Task	Responsibility
2/24 - 2/28	1. Start PCB Design 2. PCB design review session 3. Order ESP32 and other parts	1. Iker / Jaeren 2. Iker / Jaeren 3. Iker
3/3 - 3/7	1. First Round PCBway Orders 2. Design Document 3. Start assembling breadboard for demo	1. All members 2. All members 3. All members
3/14	1. Breadboard demo 2. Revise PCB design 3. Second Round PCBway Orders	1. All members 2. Iker 3. Iker
3/24 - 3/28	1. Order final components 2. Revise PCB design 3. Test microcontroller WiFi capabilities and API communication 4. Test microcontroller logic and pump activation 5. Begin soldering PCB for subsystem testing	1. Jaeren / Aashish 2. Iker 3. Aashish 4. All members 5. Iker
3/31 - 4/4	1. Third Round PCBway Orders 2. Revise PCB design 3. Continue soldering PCB for further subsystem testing and programming 4. Begin developing website UI	1. Iker 2. Iker 3. Jaeren 4. Aashish
4/7 - 4/11	1. Continue assembling/debugging full system design 2. Continue developing/debugging website UI and ‘water saved’ tracker	1. All members 2. Aashish / Iker
4/14 - 4/18	1. Almost fully complete assembly of system ready for mock demo (including soldered components on PCB with almost all subsystems functional)	1. All members
4/21 - 4/25	1. Mock Demo 2. Revise, debug, and complete design in preparation for Final Demo	1. All members 2. All members

4/28 - 5/2	1. Final Demo 2. Mock Presentation	1. All members 2. All members
5/5 - 5/9	1. Final Presentation 2. Final Report Submission	1. All members 2. All members

5. Conclusion

Automated residential irrigation rarely accounts for actual weather, leading to chronic over-watering and avoidable waste. By coupling a capacitive soil-moisture probe with live AccuWeather precipitation forecasts, our project demonstrates that a low-cost, battery-powered controller can automatically water your plant based on the specific needs of each and every one. The prototype runs for roughly twelve days on a 9 V alkaline cell, drives a 5-9 V micro-pump through a single MOSFET, and publishes real-time telemetry over its own Wi-Fi access point—no cloud account, base station, or phone app required. Because every functional block is built from hobby-grade modules, the design is easy to reproduce and, we hope, to extend.

5.1 Accomplishments

Over the course of the semester the team converted a desktop proof-of-concept into a fully untethered demonstrator that waters a plant only when environmental data indicate a genuine need. The controller now ingests AccuWeather's one-hour precipitation forecast and blends that information with live soil-moisture readings, a capability we could not find in any off-the-shelf "smart" irrigation kit aimed at hobbyists. Ten day endurance tests showed that the firmware never skipped a scheduled measurement and kept volumetric water content within ±5 % of the target set-point, all while running from a single 9 V alkaline battery. At the user experience level we achieved true plug-and-play operation: the gardener merely powers the unit, joins the self-hosted Wi-Fi network and adjusts thresholds in a browser. Finally, by relying on hobby-grade modules and deleting superfluous features we held the electronic bill of materials to $84.71, meeting both the cost ceiling and the safety goal of avoiding high-voltage circuitry.

5.2 Uncertainties

Several factors could still limit real-world performance. First, a precipitation probability is only a statistical cue; a 70% forecast that fails to deliver rain could leave the plant under-watered, whereas an unexpected shower after a watering event might push moisture above the optimal window. Second, our single capacitive probe assumes uniform soil composition, yet layered potting mixes, dense root balls, or imprecise probe placement can skew readings and misrepresent the overall moisture state. Battery life, measured under controlled indoor conditions, may shorten in cold weather where alkaline chemistry loses capacity, or in hot climates where Wi-Fi duty cycle rises.

5.3 Ethical considerations

The project promotes water stewardship by irrigating only when the plant needs it and by avoiding always-on cloud services that consume energy and harvest personal data. Removing the SD/Firebase pipeline means no user information leaves the premises; nevertheless, we store just moisture, battery level, and time stamps—nothing traceable to an individual. We specify alkaline

rather than lithium batteries to reduce fire risk and simplify end of life recycling. Finally, all firmware will be released under an open-source licence (GitHub) to encourage transparent peer review and community improvement.

5.4 Future work

Several extensions could mature the prototype into a full product. A small photovoltaic panel and boost charger would eliminate disposable batteries. Adding a tipping-bucket rain gauge would verify forecast data and improve dosing accuracy. Support for multiple probes and solenoid valves would let users manage entire garden beds. On the software side, logging to an optional cloud endpoint could enable long-term agronomic studies and more broad application use, while a lightweight machine-learning model could predict watering needs from trending data rather than threshold crossings alone. Together, these upgrades would turn the current proof of concept into a scalable platform for general data-driven plant monitoring.

References

[1] K. Sentlinger, “Water Scarcity and Agriculture,” *The Water Project*, 2023. https://thewaterproject.org/water-scarcity/water-scarcity-and-agriculture
[2] “Water Risks to Agriculture: Too Little and Too Much | Newsroom,” *Ucmerced.edu*, 2024. https://news.ucmerced.edu/news/2024/water-risks-agriculture-too-little-and-too-much
[3] “Grove -Moisture Sensor.” Accessed: Mar. 03, 2025. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/980/Grove_Moisture_Sensor_Web.pdf
[4] espressif, “ESP32-WROOM-32 Datasheet,” 2023. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
[5] *Ti.com*, 2025. https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Flm1084 (accessed Mar. 07, 2025).

[6] “Use load cell setup without tare on each use,” *Arduino Forum*, Jan. 07, 2021. https://forum.arduino.cc/t/use-load-cell-setup-without-tare-on-each-use/690267

[7] Adafruit Industries, “Submersible 3V DC Water Pump with 1 Meter Wire - Horizontal Type,” *Adafruit.com*, 2020. https://www.adafruit.com/product/4546?gQT=1 (accessed Mar. 03, 2025).