DEVELOPMENT OF A BLUETOOTH-BASED SMART IRRIGATION SYSTEM USING ESP32 FOR OFFLINE AGRICULTURAL APPLICATIONS

A Mini Research Project
In partial fulfillment of the Requirements for the course
COE 346 A - Methods of Research
Negros Oriental State University - Main Campus 1

Submitted by:
Amorin, Christian Meden G.
Gravador, Godfrey Angelou B.
Repollo, Reheca
Velez, Jera Dinah F.

June 2025

ADVISER'S CERTIFICATION

This is to certify that the mini-research project entitled DEVELOPMENT OF A BLUETOOTH-BASED SMART IRRIGATION SYSTEM USING ESP32 FOR OFFLINE AGRICULTURAL APPLICATIONS

Authored by:

Amorin, Christian Meden G.
Gravador, Godfrey Angelou B.
Repollo, Reheca
Velez, Jera Dinah F.

Has been reviewed and found acceptable for **FINAL PRINTING.**

ENGR. MARLON RAGUSTA
Content Adviser

Date Signed: ________

Republic of the Philippines
NEGROS ORIENTAL STATE UNIVERSITY
Main Campus 1 & 2, Dumaguete City
COLLEGE OF ENGINEERING AND ARCHITECTURE

APPROVAL SHEET

This mini-research project entitled:

DEVELOPMENT OF A BLUETOOTH-BASED SMART IRRIGATION SYSTEM USING ESP32 FOR OFFLINE AGRICULTURAL APPLICATIONS

Prepared and Submitted by:

Velez, Jera Dinah F.

Has been reviewed and approved by the research committee.

CRAIG N. REFUGIO, PhD
Chairman
Date Signed: ________

ENGR. CHRIST C. QUINICOT
Contents Adviser
Date Signed: ________

ACCEPTED by the Dean of the College of Engineering and Architecture, NORSU MC2 as a final requirement for the course COE 346 - Methods of Research.

JOSEF VILL S. VILLANUEVA, PhD
Dean, College of Engineering and Architecture
Date Signed: ________

ABSTRACT

This project addressed the challenge of inefficient manual irrigation practices in small-scale or offline agricultural settings, where overwatering and underwatering frequently lead to resource waste and poor plant health. The main objective in creating this research is to create an ESP32 system that is also controlled through Bluetooth with the goal of low-power, low-cost and an efficient irrigation watering system.

To achieve this goal, the researchers created a prototype that consists of an **ESP32 microcontroller**, **capacitive soil moisture sensor, relay** and **water pumps**. The Capacitive Soil Moisture Sensors sends value from the soil to the ESP32 and sends that data via Bluetooth to an application in a smartphone created through **Android Studio** and interprets that data in the app. In the App, users can choose what profiles they should choose **(Small or Medium)**, in which it automatically triggers the water pump. Water delivery is determined and pre-calibrated to 5mL and 10mL for small and medium pots respectively.

As testing was done, it demonstrated that the **Bluetooth Communication range** remained stable **up to 8 meters** and readings from the Soil Moisture Sensors were consistently accurate. The irrigation system achieved water delivery to pots within a **±1 mL approximation** and showed a **30-50% efficiency increase in water usage** compared to traditional methods of irrigation. The total cost of implementing this prototype was approximately **₱1,441** which confirms its validity for use in low-resource environments.

These findings suggest that the developed system offers a reliable, scalable, and affordable solution for precision irrigation in offline and rural applications. Future work may include expanding plant profiles, adding weather-based automation, or enabling multi-pot scalability.

Keywords: Smart irrigation, ESP32, Bluetooth, soil moisture sensor, offline automation, low-cost engineering solution

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who contributed to the success of this research project.

First and foremost, we extend our heartfelt appreciation to **Engr. Marlon Ragusta**, our **Content Adviser** and instructor, for his valuable insights, encouragement, and feedback throughout the development of the system. His guidance played a crucial role in shaping the direction and clarity of our research.

We are equally grateful to **Dr. Craig N. Refugio**, our **Methods Adviser**, for his expert support in refining our methodology, validating our research process, and continuously challenging us to improve the scientific rigor of our work.

We also thank the **College of Engineering and Architecture at Negros Oriental State University - Main Campus II** for providing us with the tools, facilities, and academic support needed to complete this project.

To our family, friends, and classmates, we are grateful for your unwavering love, support, and appreciated your understanding, sacrifices, and encouragement. Your belief in us kept us going and made this process lighter and meaningful. We are truly grateful for those who helped make this research a success in whatever manner. We really appreciate your assistance.

Above all, we want to thank God from the bottom of our hearts for giving us courage, insight, and direction during this research process. Without His grace, this achievement would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

Abbreviation / Symbol	Meaning
ESP32	Embedded System Platform 32-bit
DC	Direct Current
mL	Milliliters
IDE	Integrated Development Environment
IoT	Internet of Things
GSM	Global System for Mobile Communications
₱	Philippine Peso (currency symbol)
BT	Bluetooth

CHAPTER I

INTRODUCTION

1.1 Background of the Study

Due to water's direct influence on mitosis, translocation, and photosynthesis, it became essential in plant development. Inappropriate irrigation practices—excessive or insufficient watering—can result in poor crop yield, plant stress, and inefficient water use (Gutierrez et al., 2014). Maintaining an optimal irrigation schedule is still difficult in rural and underdeveloped areas because of limited access to real-time environmental data and erratic internet connectivity (Pereira, Chaari, & Daroge, 2023).

With the advancement of smart agriculture technologies, embedded systems such as the ESP32 microcontroller offer innovative and cost-effective solutions. To achieve automation in environments with unstable internet connectivity, the ESP32 is a powerful, economical microcontroller; it is equipped with Bluetooth and Wi-Fi capabilities, making it well-suited for this project(Mehta et al., 2023). Because they rely on internet availability, the offline setting application limits the generally used Wi-Fi-enabled irrigation systems. In order to maintain the advantages of automation even in areas that are inaccessible, this Bluetooth-based irrigation is a possible alternative (Pereira et al., 2023).

This study is focused on developing an affordable and energy-efficient automatic watering system with the use of ESP32 microcontroller and Bluetooth to enable automatic watering depending on real-time soil moisture data. The system uses capacitive soil moisture sensors along with a water pump to provide water exactly when it is required, making the system energy responsive and efficient. Furthermore, the system introduces a basic plant profile feature allowing users to assign plant types (e.g., small, medium, or large) that control the volume of water released—for instance, 5mL for small plants and 20mL for large ones. This personalized irrigation method addresses the unique needs of different plant types, contributing to better water resource management (Tyagi et al., 2024).

1.2 Statement of the Problem

Traditional irrigation methods are prone to inconsistency and human error, often resulting in over- or under-watering, especially in remote or rural settings. With this in mind, the study aims to answer the following research questions:

1. How can soil moisture be effectively monitored using a sensor-based system for irrigation?
2. How can Bluetooth communication be integrated with the ESP32 to automate irrigation in offline environments?

3. Can the implementation of plant-specific watering profiles improve water conservation and plant health?
4. Will the proposed Bluetooth-based system demonstrate improved efficiency compared to manual or conventional irrigation methods?

1.3 Objectives of the Study

The main goal of this study is to design and implement a Bluetooth-based automated irrigation system using the ESP32 microcontroller. The specific objectives are:

1. To monitor soil moisture levels in real-time using capacitive soil moisture sensors.
2. To automate the activation of a water pump when soil moisture drops below a predefined threshold.
3. To enable Bluetooth-based manual control and configuration through a mobile application.
4. To incorporate basic plant profiles to regulate the amount of water dispensed based on plant size.
5. To develop a cost-effective, offline, and energy-efficient irrigation solution suitable for home and small-scale agricultural use.

1.4 Scope and Delimitations of the Study

Scope: This study focuses on the design, implementation, and testing of an offline automated irrigation system using an ESP32 microcontroller, capacitive soil moisture sensors, a water pump, and Bluetooth communication. It includes a basic plant profile system to adjust watering volumes.

Delimitations: The system is limited to small-scale setups such as home gardens or indoor plants. It does not include cloud storage, Wi-Fi-based monitoring, weather integration, or advanced nutrient sensing. The plant profiles are based on size categories only and do not account for specific plant species or soil types.

1.5 Significance of the Study

This study contributes to the field of smart agriculture by offering a practical solution for irrigation in areas with limited connectivity. It showcases the potential of the ESP32 microcontroller in creating low-cost, accessible automation systems for water conservation (Gutiérrez et al., 2014; Mehta et al., 2023).

The project is significant to the following groups:

- Home gardeners who need a low-maintenance irrigation system.
- Small-scale farmers in rural regions lacking internet infrastructure.
- Students and educators in engineering and agricultural programs focusing on IoT and embedded systems.
- Environmental advocates promote sustainable resource use through efficient irrigation.

CHAPTER 2

REVIEW OF RELATED LITERATURE

This part of the study highlights existing literature that focuses on automatic irrigation systems using microcontrollers and environmental inputs. Traditional agricultural methods often lead to water waste that results in inconsistent plant health. Automatic irrigation systems provide a possible solution especially for offline agricultural setups. By exploring existing research, gaps can be identified and addressed.

2.1 Smart Irrigation Systems and Water Conservation

As the world's population is on the rise, food concerns are also increasing. Agriculture is one of the solutions to aid these demands, a reason why water conservation is essential. Freshwater resources are being consumed at a fast-moving rate. Efficient and accurate water management is vital not only to sustain food crops but also to maintain an effective long-term sustainable farming. Manual methods take time and often lead to water-wastage because of the lack of control in water distribution. In response, an automatic watering system can be a possible solution to manage water use by controlling the precision of water through monitoring soil moisture, this setup can help reduce water waste and improve agriculture.

These modern setup use technologies such as soil moisture sensors, and microcontrollers to ensure that water is distributed well and efficiently. These systems prevent water waste and ensure that plants receive the right amount of water needed for optimal growth, meanwhile supporting and enhancing plant health and agricultural practices.

A study by Gutierrez et al. (2014) showed that developing a wireless sensor network into watering setups allows continuous monitoring of the soil and its moisture, therefore enabling water distribution when moisture levels drop below the set threshold and stopping water distribution when the soil moisture is above a certain threshold. These systems help areas that are experiencing water scarcity and help maintain water correctly, offering advantages environmentally and in food demands.

One of the components of a smart irrigation system is the soil moisture sensor which detects soil moisture and triggers the irrigation process. Sahu and Mohanty (2020) have studied that microcontrollers, specifically the ESP32, combined with the soil moisture sensor can offer effective and accurate changes in the system, preventing over and under-watering and ensures a steady distribution. While the cost of an automatic watering system is expensive compared to manually watering the system, it can serve as an investment for it helps users in the long-term and it is hassle-free compared to the traditional methods. Bhat and Prasad (2023) pointed out that automatic watering systems can reduce water waste by approximately 5%, while also reducing the need for manual labor because it is embedded automatically.

The system's automatic behavior is one of the advantages in smart irrigation setups. Pereira et al. (2023) found that by using ESP32 microcontrollers combined with Bluetooth, users can control the irrigation process via a mobile application and is very convenient for offline agriculture. This remote accessibility is beneficial in rural areas where traditional setups are inefficient.

In addition, adding weather forecasts in automatic watering systems can enhance and ensure how water is used. By adjusting water distribution through scheduling based on predicted rainfall, these systems can help avoid unnecessary irrigation. Based on the study of Tyagi et al. (2021), with the use of machine learning algorithms, automatic irrigation systems can be more effective and efficient. They can predict future water needs in real-time, responding to current environmental conditions. The growing availability of microcontrollers such as the ESP32 which is used in this system can make automatic watering more practical for users, especially in offline and areas with unlimited resources. Gupta and Aggarwal (2020) showed that Bluetooth-based irrigation systems can offer an affordable solution that doesn't rely on internet connectivity making it easier for users to adopt this technology.

In summary, automatic watering systems offer a convenient and powerful solution to growing challenges of water conservation and agricultural practices. With the combination of tools like soil moisture sensors, microcontrollers, and communication via Bluetooth, these systems help users use more water efficiently, reducing manual efforts and advancing from outdated irrigation methods, As water shortages and climate change affecting the global food demands, adapting a smart irrigation systems are significant and essential, not just for today's agricultural needs but for sustaining a more secured food production in future generations.

2.2 Microcontroller-Based Irrigation Technologies

Technologies using microcontrollers offer essential and significant solutions in manual agricultural practices. It is affordable as well as it reduces manual efforts. ESP32 microcontrollers, also Arduino and Raspberry Pi are the key parts of smart irrigation systems as they serve as the backbone of the system, offering flexibility and compatibility with many communication tools such as Bluetooth and WiFi. They act as the brains of an automatic irrigation system, with their compatibility with sensors that can detect soil moisture sensors and collect real time data to trigger water distribution to plants. This system helps users, which are the farmers, to conserve water, and ensure that plants receive the right amount of care.

2.2.1 The Role of Microcontrollers in Smart Irrigation

Microcontrollers serve as the key part of automatic systems, the heart of the system which is responsible for gathering and collecting data from sensors and controlling water distribution in sprinklers and other outputs. When sensors are triggered, irrigation is then activated. This technology not only reduces water waste but enhances plant health.

Microcontrollers like the ESP32 are the key part of many automated technologies, most specifically smart irrigation systems. These microcontrollers serve as the heart that gathers and collects data with the use of environmental sensors that control the distribution of water, sprinklers, etc. By analyzing real time data from sensors such as the soil moisture sensor, the system then triggers and irrigation is being activated. This process not only conserves water but it also eliminates the risk of over and under watering.

These microcontrollers become an important component in automatic agricultural practices because of their affordability and their compatibility with certain communication tools. It is ideal in areas with limited access to the internet, with the development of the system automatic irrigation system can be possible without internet access.

ESP32 microcontroller is ideal for two core processing units. It has the capability to have compatibility in both Bluetooth and WiFi communication tools, and it is advantageous for it consumes a small amount of power. With this, communication between the system and the software, which is the mobile app that allows users to control and monitor irrigation processes through mobile apps become possible. According to Mehta et al. (2023), automatic irrigation systems have been successfully implemented not just in small scale farming, but also beneficial for large scale farming.

2.2.2 Microcontroller-Based Irrigation Systems with Sensors

The primary function of a microcontroller-based irrigation system is to gather, collect, and process data from environmental sensors like the soil moisture sensor. The system triggers the irrigation process when the soil moisture level drops on a certain threshold, ensuring that water is distributed only when it is needed. Soil moisture sensors are important for measuring the amount of water present in the soil, this enables the system to turn on automatically when moisture levels drop below a certain threshold and stop when the set threshold is reached.

There are types of soil moisture sensors, resistive sensors and capacitive sensors. Using capacitive soil moisture sensors are ideal together with the ESP32 microcontroller as these sensors can resist corrosion unlike the normal sensors. Sahu and Mohanty (2020) demonstrated the effectiveness of combining capacitive sensors with microcontrollers for managing irrigation processes efficiently. The system then activates the water pump when the sensor is triggered, specifically when the data from the soil moisture drops at the set threshold, this function ensures that the plant receives the right amount of water and avoids overwatering. Not only soil moisture sensors, but also other environmental sensors, such as temperature, humidity, and rain sensors, can be implemented into these systems for ensuring and optimizing more accurate irrigation processes. As an example, a temperature sensor can adjust the irrigation schedule depending on environmental temperature. On the other hand, a rain sensor can automatically stop the irrigation process if rainfall is detected, preventing water wastage.

2.3 Offline Smart Irrigation Using Bluetooth Technology

Water wastage is one of the issues in traditional agricultural methods. In recent years, smart irrigation systems have become an efficient and effective solution. This system can help users conserve water by controlling its precision and reducing water consumption. However, much of this innovation depends on cloud based infrastructure which needs internet connectivity. Reliance on the internet can limit the use of these systems especially in offline areas.

As a solution, the use of Bluetooth is a possible alternative. Bluetooth can enable offline connection with the mobile app. With the communication between the irrigation system and mobile devices, Bluetooth offers an affordable, low energy consumption, and local solution, eliminating the need for constant internet connectivity. This is particularly useful for farmers in remote areas, where internet connectivity is an issue, users can still manage their irrigation systems remotely without relying on an internet connection. Bluetooth-powered smart irrigation systems are especially advantageous and convenient in areas with limited resources.

2.3.1 Bluetooth-Based Smart Irrigation System Design

With the innovation of a smart irrigation system with the use of Bluetooth, together with environmental sensors such as the soil moisture sensors and also the key component which is the micro controllers, the system can automate the irrigation process efficiently.

- Soil Moisture Sensors: These sensors measure soil moisture, and the measurement, which is the data triggers the activation process. It triggers the system to automatically allo water distribution to plants and stop when it reaches the set soil moisture or the predefined threshold.
- Microcontrollers: Microcontrollers are the brain of the smart irrigation system, these chips decide the activation of the irrigation process with the data from environmental sensors.
- Bluetooth Communication: Bluetooth is an ideal alternative to WiFi. It allows users to monitor irrigation processes in their devices via this communication tool without the need of internet connectivity.

This system is made to automate irrigation processes in agricultural areas without the need of internet connectivity, and Bluetooth as an alternative communication tool. Users can monitor the watering process via Bluetooth. Gupta and Aggarwal (2020) illustrated this approach in their study, where they implemented an irrigation system controlled by this communication tool and also with the ESP32 acting as the brain of the system and the sensors that collect real time data. Their system enabled farmers to check moisture levels, activate irrigation, and adjust settings directly from a mobile app, making it an efficient, accurate, and user-friendly solution particularly in areas with limited internet access.

2.4 Plant Profiling and Customization of Watering

Manual watering methods can be very time consuming and it treats the plant the same, doing the irrigation process the same with different plants. This method can lead to some plants being over watered and under watered, compromising plant health. This feature adds an additional innovation of the system

Water conservation is very important especially in areas with limited water resources. Plant profiling and customized irrigation schedules offer a solution to this challenge. By tailoring irrigation to the size, type, growth stage, and environmental conditions of each plant, these systems ensure that each one receives the right amount of water at the right time, not being over watered and under watered at the same time.

Plant profiling feature involves collecting data from environmental sensors such as soil moisture sensors, temperature sensors, etc. to gather information and these information being gathered to adjust watering schedule to avoid water wastage and ensure that specific plants receive the right amount of water.

2.4.1 The Role of Sensors in Plant Profiling

Sensors are important components in plant profiling as through these sensors, data can be collected and gathered and these gathered data are used for accurate customization of plant profiles. Plant needs can be customized accurately with the help of environmental sensors, ensuring that plants receive the right amount of water needed, avoiding over watering and under watering to enhance plants' health. Different environmental sensors work together to provide an efficient and accurate data of each plant's water requirement that may be used for customization for irrigation processes. Together with the ESP32 microcontrollers, the system can automatically trigger watering events based on information collected from these sensors, ensuring that plants receive the appropriate moisture for healthy growth.

2.4.2 Custom Irrigation Based on Plant Profiles

Irrigation systems can be tailored, allowing water distribution based on plant profiles. These customized irrigation systems adjust watering distribution based on plant requirements. Several factors can be the type and size of those plants because different plants have different water needs. Factors such as temperature and wind can also affect plants, the reason why customizing plant profiles is important. The combination of microcontrollers like the ESP32 and environmental sensors can adjust the automation process and ensure that plants receive the right amount of water for their specific needs.

2.4.3 Benefits of Plant Profiling and Custom Irrigation

Plant profiling with the help of environmental sensors is beneficial in the irrigation process. Some of the benefits include;

1. **Water Efficiency:** Water wastage is reduced as water is distributed only when necessary, reducing overwatering.
2. **Improved Plant Health:** Different plants require different water levels, with this feature, plants can't receive the appropriate and right amount of water for their specific needs, ensuring plant health.
3. **Environmental Sustainability:** This feature helps reduce overall water consumption, making them more environmentally friendly.

2.4.4 Challenges of Implementing Custom Irrigation Systems

Plant profiling features for customized irrigation processes have advantages, but there are also challenges including sensors' accuracy, incorrect reading collected from sensors can result in over watering or under watering. Sensors must be properly maintained and calibrated to ensure accuracy. System complexity can also be a challenge for implementing the overall automatic irrigation system can be complicated especially for users with no technician expertise, making the setup simpler and having tutorials can be a solution. Also, cost is one of the challenges, microcontrollers are affordable but some of the high quality sensors are expensive. This is particularly true for smallholder farmers in low-resource areas, where affordability remains a key barrier to implementation.

2.5 Benefits of Smart Irrigation Systems with Plant Profiling

Smart irrigation systems that integrate plant profiling bring several important benefits, offering the potential to transform water usage and boost agricultural productivity. Benefits include water conservation and efficiency, for instance, a study by Sahu and Mohanty (2020) demonstrated that their ESP32-based irrigation system cut water usage by 50% compared to traditional irrigation methods. Another benefit is that it improves plant health and growth. According to Sharma and Garg (2020), implementing profiles in specific plants can help users adjust watering schedules and distributions. This approach led to healthier plants.

2.6 Summary of Related Works

Existing research and studies are very important for the development and improvement of automatic irrigation systems, with their help in water conservation, its affordability, and this system ensures and enhances plant health. According to Gutiérrez et al. (2014) and Sahu and Mohanty (2020), microcontrollers such as the ESP32 and other sensor networks significantly improved water efficiency. Pereira et al. (2023) and Gupta and Aggarwal (2020) demonstrated the use of Bluetooth communication for offline irrigation management, demonstrating that offline irrigation systems can be highly effective and accurate, particularly for smallholder farmers in remote areas with limited access to internet connectivity. Despite these advancements, there are still challenges that need to be addressed for a better version of the system.

2.7 Research Gap

Although offline agricultural areas benefit from automatic watering systems, there are still a number of research gaps that need to be addressed for a better optimization of the system including Bluetooth range, although it works well for small farms, investigating alternative long-range communication technologies are essential to scaling these systems for larger agricultural operations. Large-scale implementation, sensor accuracy and calibration, and the combination of AI and machine learning.

CHAPTER 3
METHODOLOGY

This chapter explains, step by step, **how** we designed, built, and tested the Bluetooth‑based smart irrigation system powered by the ESP32. It is organised into six sections: research design, materials, experimental procedures, data‑collection methods, analytical techniques, and illustrative diagrams.

3.1 Research Design

We adopted a **developmental–experimental** (design‑based) approach. Developmental work focused on rapid prototyping—iteratively refining hardware and firmware—while the experimental phase gathered **quantitative data** to answer four performance questions:

1. How precise is the capacitive soil‑moisture sensor?
2. How accurate is the calibrated water delivery for each plant profile?
3. How much water can the system save versus manual watering?
4. How stable is Bluetooth communication in an offline environment?

3.2 Materials

All components were chosen for **affordability**, **local availability**, and **low power consumption**.

3.2.1 Hardware

Item	Description	Qty	Unit (₱)	Total (₱)
ESP32 Dev Board	Dual‑core MCU with Bluetooth	1	175	175
Capacitive Soil‑Moisture Sensor	Corrosion‑resistant	2	85	170
4‑Channel Relay Module	Opto‑isolated, 5 V	1	150	150
5 V Submersible Pump	Mini DC pump	2	48	96
USB Power Bank (5 V)	10 000 mAh	1	300	300
External 5 V Adapter	Bench power	1	400	400
Tubing (5/16 in)	PVC hose	1 m	17	17

Breadboard + Jumpers	Prototyping	—	50	50
AWG-22 Stranded Wire	Pump leads	—	16	16
1 L Recycled Bottle	Water reservoir	1	0	0
TOTAL				**1,441**

Table 3.1: Lists all hardware components used in building the Bluetooth-based smart irrigation system. Each item includes the quantity used and its corresponding cost in Philippine Pesos (PHP). This breakdown demonstrates the system's low-cost nature, making it suitable for small-scale and resource-limited applications.

3.2.2 Software

Tool	Purpose
Arduino IDE	ESP32 firmware (C/C++)
Android Studio	Mobile app: Bluetooth pairing, live moisture display, profile buttons

Table 3.2: Identifies the software tools utilized for programming and interfacing the system components. It highlights the use of open-source platforms such as Arduino IDE and Android Studio, chosen for their accessibility, flexibility, and compatibility with the ESP32 microcontroller and mobile application development.

3.3 Experimental Procedures

1. **Prototype Assembly** • Wire sensors to GPIO 34/35; relays to GPIO 26/27.
   • Mount pumps and route tubing to each pot.
   • Power ESP32 via power bank; pumps via 5 V adapter.
2. **Firmware Upload**
   • Code reads moisture every 5 s, compares value to a **40 % threshold**, and activates the correct pump for **3 s (5 mL)** or **6 s (10 mL)** depending on Bluetooth command *(POT:SMALL / POT:MEDIUM)*.
3. **Mobile App Pairing & Control**
   • User pairs phone, selects profile, and monitors live moisture.
4. **Pump Calibration**
   • Use a graduated cylinder; adjust runtime until deviation ≤ ±1 mL.
5. **Indoor Test Runs**
   • Operate system for five consecutive days on two potted plants.
   • Log moisture before/after watering, pump runtime, and Bluetooth RSSI.

• Wire sensors to GPIO 34/35; relays to GPIO 26/27.
• Mount pumps and route tubing to each pot.
• Power ESP32 via power bank; pumps via 5 V adapter.

6. **Firmware Upload**
   • Code reads moisture every 5 s, compares value to a **40 % threshold**, and activates the correct pump for **3 s (5 mL)** or **6 s (10 mL)** depending on Bluetooth command *(POT:SMALL / POT:MEDIUM)*.
7. **Mobile App Pairing & Control**
   • User pairs phone, selects profile, and monitors live moisture.
8. **Pump Calibration** (*Figure 3-3*)
   • Use a graduated cylinder; adjust runtime until deviation ≤ ±1 mL.
9. **Indoor Test Runs**
   • Operate system for five consecutive days on two potted plants.
   • Log moisture before/after watering, pump runtime, and Bluetooth RSSI.

3.4 Data-Collection Methods

- **Automatic Logs** (CSV): timestamp, moisture %, pump state, profile ID.
- **Manual Validation**: Gravimetric oven-dry method for spot-checking sensor accuracy.
- **Observation Notes**: Bluetooth dropouts, power interruptions, environmental anomalies.

3.5 Analytical Techniques

Metric	Formula / Criterion
Sensor Precision (%)	(|Actual − Sensor Reading| ÷ Actual) × 100
Delivery Deviation (mL)	Actual – Target (± 1 mL acceptable)
Water-Savings (%)	(Manual – Smart) / Manual × 100
Bluetooth Reliability (%)	Successful sessions / total sessions × 100

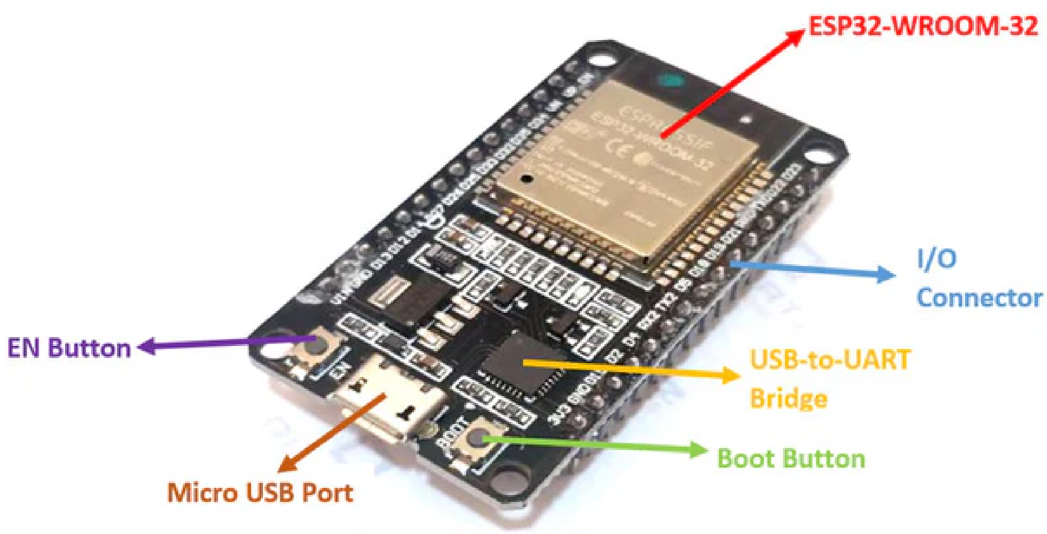
Table 3.3: Outlines the core metrics used to evaluate the performance of the Bluetooth-based smart irrigation system. Each metric includes the specific formula or criterion used to quantify system accuracy, reliability, and efficiency. These metrics were essential in assessing sensor precision, water delivery consistency, water conservation impact, and Bluetooth communication reliability—ensuring a comprehensive evaluation of both hardware and software components in real-world testing. A precision rubric was applied (Precise ≥ 85 %, Moderate 70–84.99 %, Inaccurate < 70 %).

CHAPTER 4
DESIGN AND IMPLEMENTATION

This chapter presents the overall design and implementation process of the ESP32 Bluetooth-Controlled Irrigation System. It outlines the systematic approach taken to develop the prototype, including the selection of components, circuit configuration, and programming logic.

4.1 Detailed Description of the System Design

Figure 1: ESP32 Microcontroller



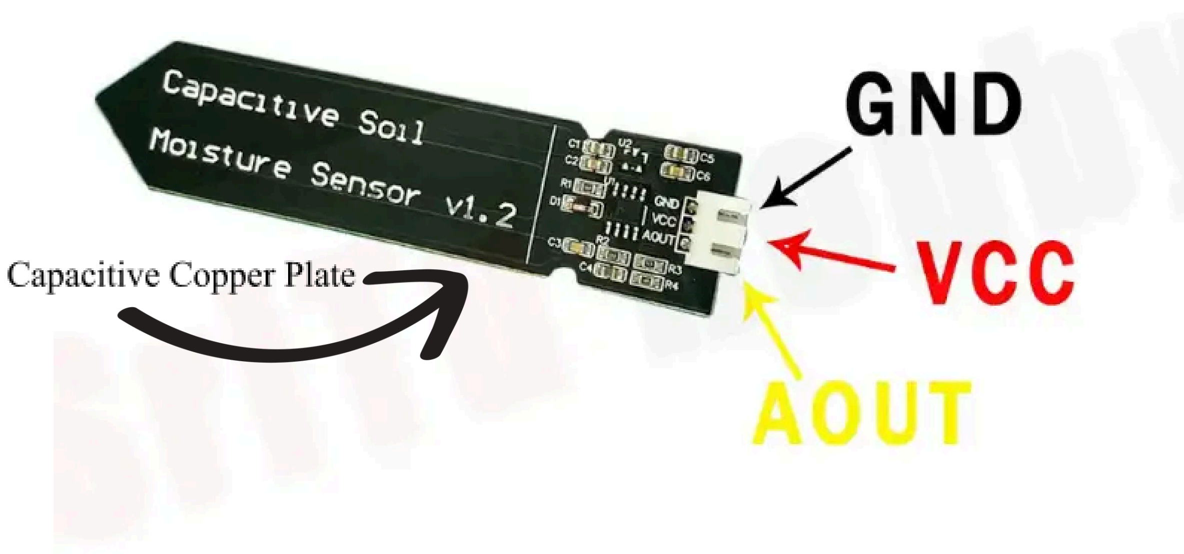
The ESP32 Microcontroller is a low-cost, flexible and energy-efficient microcontroller that integrates Wi-Fi and Bluetooth capabilities to create prototypes that utilize Wi-Fi and Bluetooth. The ESP32 is selected for its low power consumption, integrated Bluetooth, and affordability, aligning with Mehta et al. (2023) and Pereira et al. (2023), who used ESP32 in smart irrigation systems due to its IoT-friendly architecture.

PARTS:

- **Micro USB Port** - Used to connect and communicate the ESP32 Microcontroller to a computer for programming the ESP32 and also supply power.
- **ESP32-WROOM-32** - It is the brains of the ESP32 Microcontroller and also contains Wi-Fi and Bluetooth modules inside for wireless capabilities.
- **USB to UART Bridge** - This bridge acts like the translator for sending/ receiving data to and from the Microcontroller and the Computer.

- **I/O Pins** - Used to connect Sensors and other I/O Devices and communicate with the ESP32 Microcontroller. The GPIO Pins may be used as Digital Pins or be used as Analog Pins.
- **Boot Button**- A button used for when in the event the auto- uploading feature in the Arduino IDE in uploading your code from the computer to the ESP32 Microcontroller does not work properly.
- **EN Button (reset)** - Resets the ESP32 Microcontroller in the event that your code does not work properly or crashes the Microcontroller.

Figure 2: Capacitive Soil Moisture Sensor

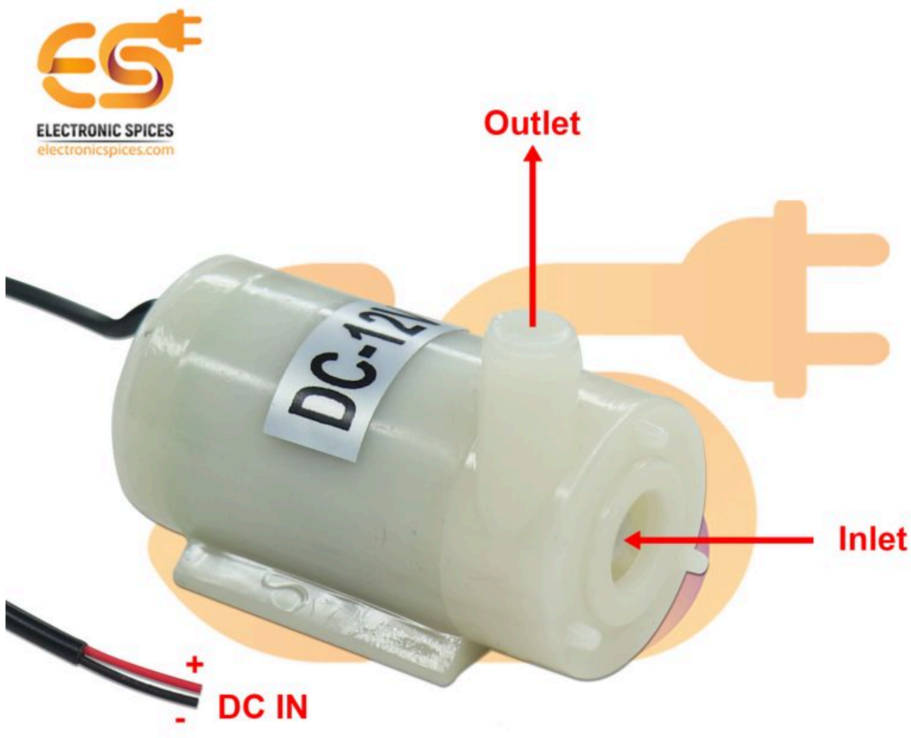


A Capacitive Soil Moisture Sensor is a Device used to measure the Soil Moisture Level or water content in a soil (or other materials). The Capacitive Soil Moisture Sensor is commonly used for automated water and soil irrigation systems. A capacitive moisture sensor is used to avoid corrosion and provide long-term accuracy. It gives an analog signal proportional to the moisture level in the soil. Provides data coming from the soil and sends that data to the ESP32 to see if the soil is wet or dry.

PARTS:

- **Capacitive Copper Plate**- Used to detect soil moisture levels when put into soil. When soil is moist or has water, it conducts more electricity when it has little to no soil.
- **Wires** (Red, Black, Yellow)
    - Red (VCC) - Connects 5VDC power to the Sensor to function.
    - Black (GND) - Connect to ground connection.
    - AOUT (Data) - Connects to the I/O Pins in the ESP32 Microcontroller for data sensing.

Figure 3: Submersible Water Pump

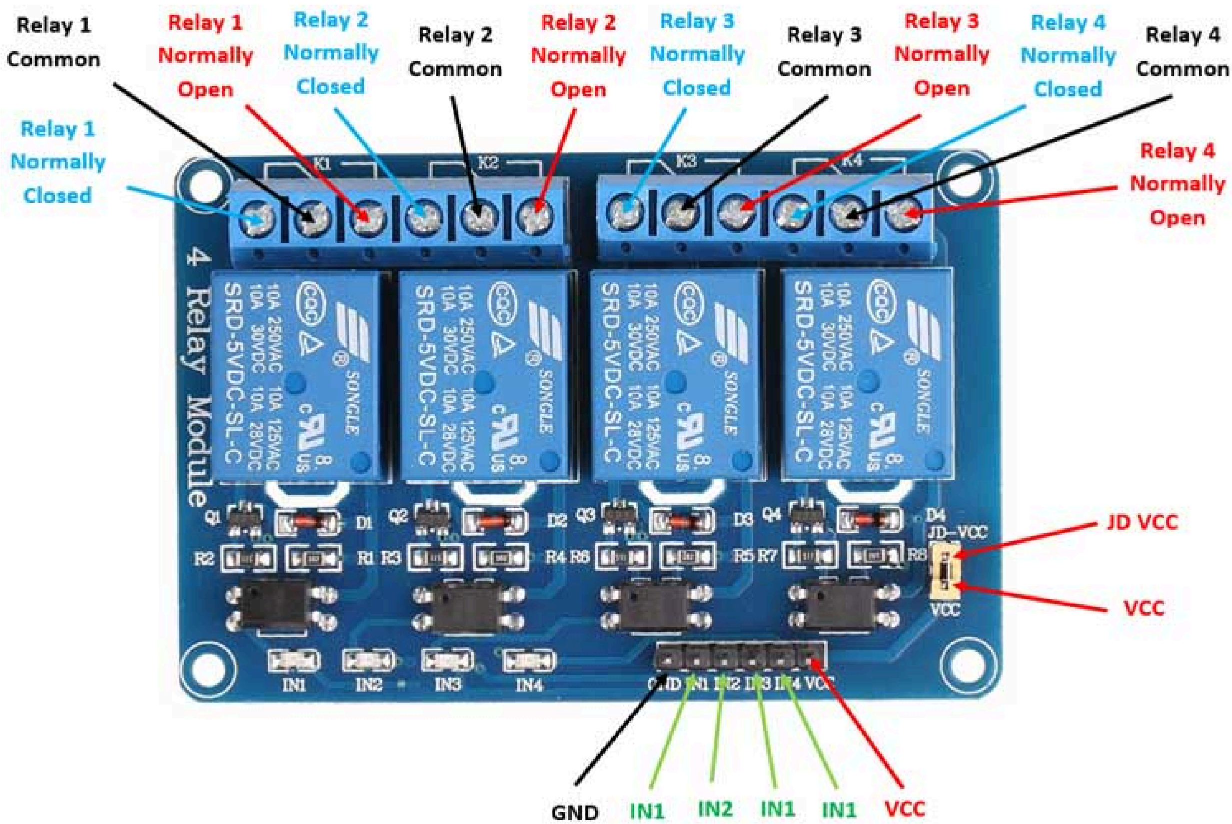


The Submersible water pump is a pump that is designed to be submerged in a water source or a container and its purpose is to pump water from a container or a water source to a destination like a flower pot.

PARTS

- **Inlet-** Water from a container or from a water tank enters this part of the submersible water pump.
- **Outlet** - Water exits here to be distributed to where you put your water outlet to.
- **DC In** - Provides power to the submersible water pump in order for the pump to work and function properly.
    - **Red (+)** - Positive end of the submersible water pump and to be connected to the positive (VCC) part in the power supply.
    - **Black (-)** - Negative end of the submersible water pump and to be connected to the negative (GND) part in the power supply.

Figure 4: 4 Relay Module

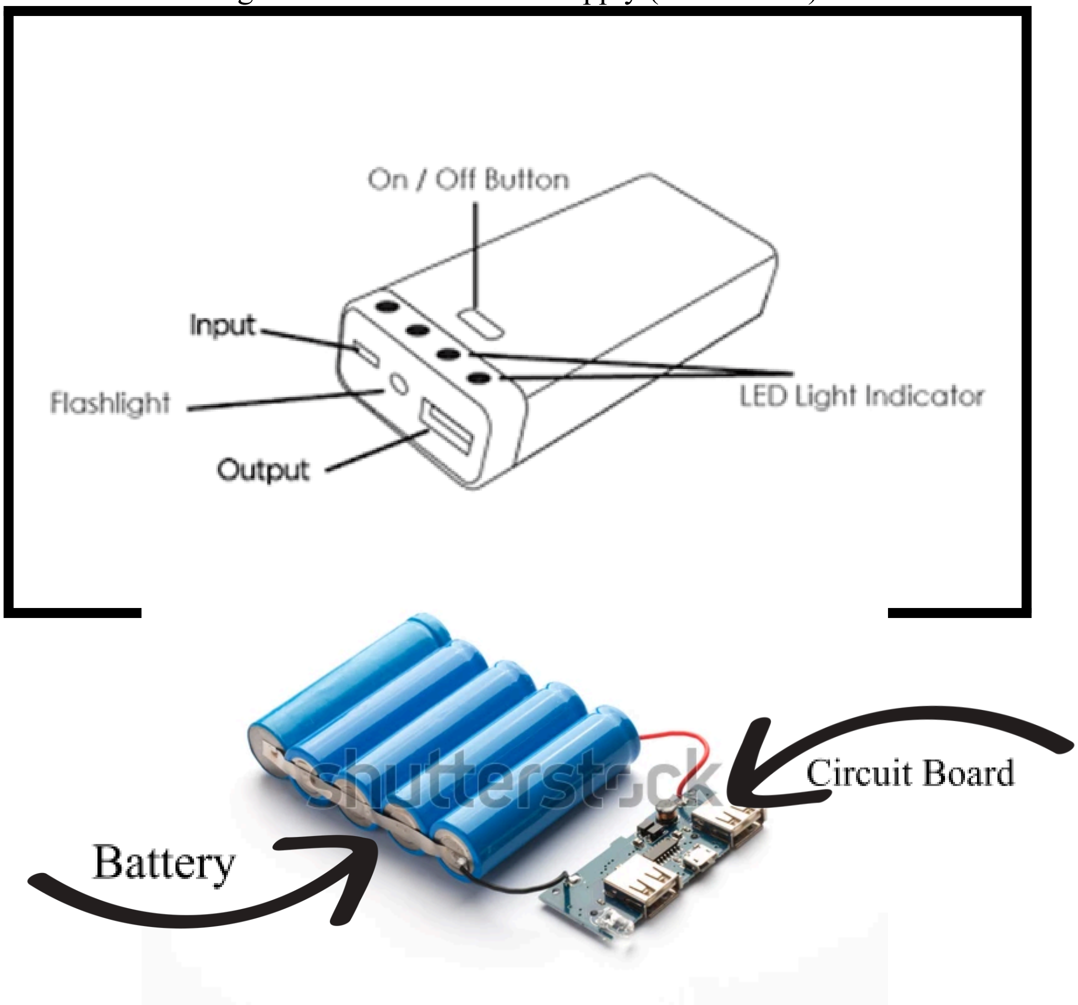


This is a 4 relay module and it is used for controlling the submersible water pumps on when to turn them on based on the capacitive soil moisture sensor. The relay stays open based on how many mL and what profile did you choose for your pot (Small and Medium).

PARTS:

- **VCC and GND** - Connects to Power and Ground respectively of the 4 relay modules in order to function. Properly.
- **IN 1 to IN 4 (Input 1-4)** - Signal pins that are triggered when the ESP32 Microcontroller gives a signal.
- **Normally Closed (NC)** - devices connected to the Normally Closed end receive power from a power source not until the relay switches to the Normally Open end, where your devices will not receive power.
- **Normally Open (NO)** - devices connected to the Normally Open will not receive any power from a power source not until the relay switches from Normally Closed to Normally Open when it receives a signal from the IN1-IN4 Pins.
- **Common (COM)** - A terminal where current flows in or out in the relay.
- **JD VCC** - it is used to power the relay coils inside the relay and jumpered together if you want to power the relay from only one power source.

Figure 5 : External Power Supply (PowerBank)



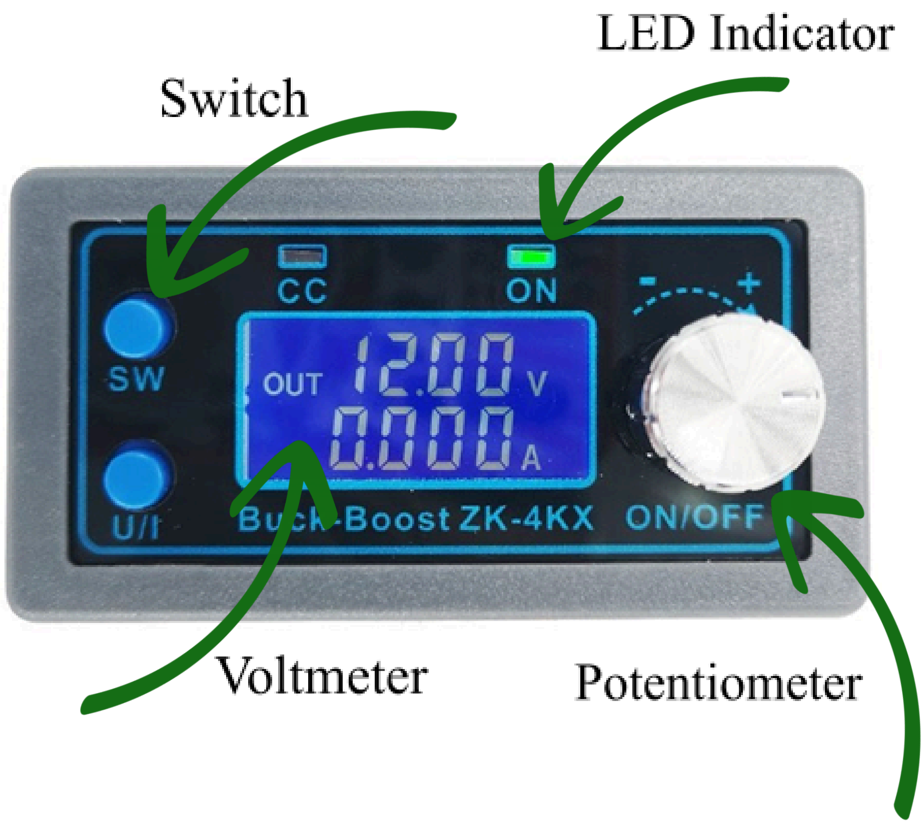
A 5V Power Supply (Powerbank) is used to give a constant power to the ESP32 Microcontroller and also the prototype can be accessible in offline environments

PARTS:

- **LED Light Indicator -** A LED indicator to let the user know if how much power is left in the PowerBank
- **Output** - Connects to the ESP32 Microcontroller or any other devices with USB support to power or charge the device .
- **Flashlight** - A LED light to be used in an event if you are in a dark place and are in need of light.
- **Battery** - Stores energy in the PowerBank in order to be used in outdoor and offline environments.
- **Input** - This is where you connect your PowerBank to an outlet that is connected to the electricity grid in order for the PowerBank to be charged fully to be used in outdoor and offline environments.

- **Circuit Board** - Controls how much power will it supply as giving your device too much to too little voltage may cause your device to not charge at the appropriate rate or destroy your device.

Figure 6: External Power Supply (Variable)

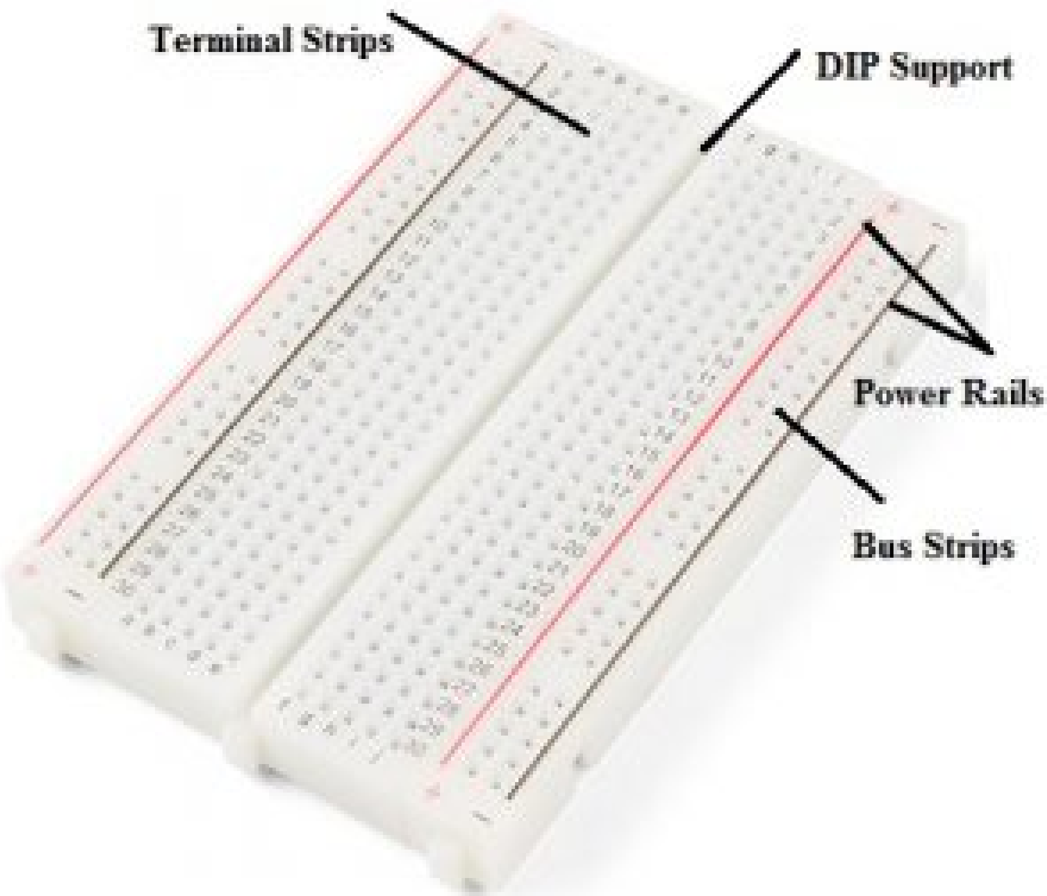


A variable Power Supply is used to give the COM Terminal in the relay power in order for the relay to function properly. It is needed as if the relay is turned on by the ESP32 Microcontroller, the current needed for the water pump to function is too high for the ESP32 Microcontroller to handle.

PARTS:

- **Potentiometer** - Is used to adjust the voltage levels in the power supply for accurate voltage outputs to give to devices.
- **Voltmeter**- Gives the user an indication on whether what voltage it is set on for your devices to work in that set voltage.
- **LED Indicator** - Lets the user know if the Power Supply is turning on when connected to an outlet.
- **Switch** - Turns on and Off the Power Supply in order for the user not to waste power when not in use.

Figure 7: Breadboard

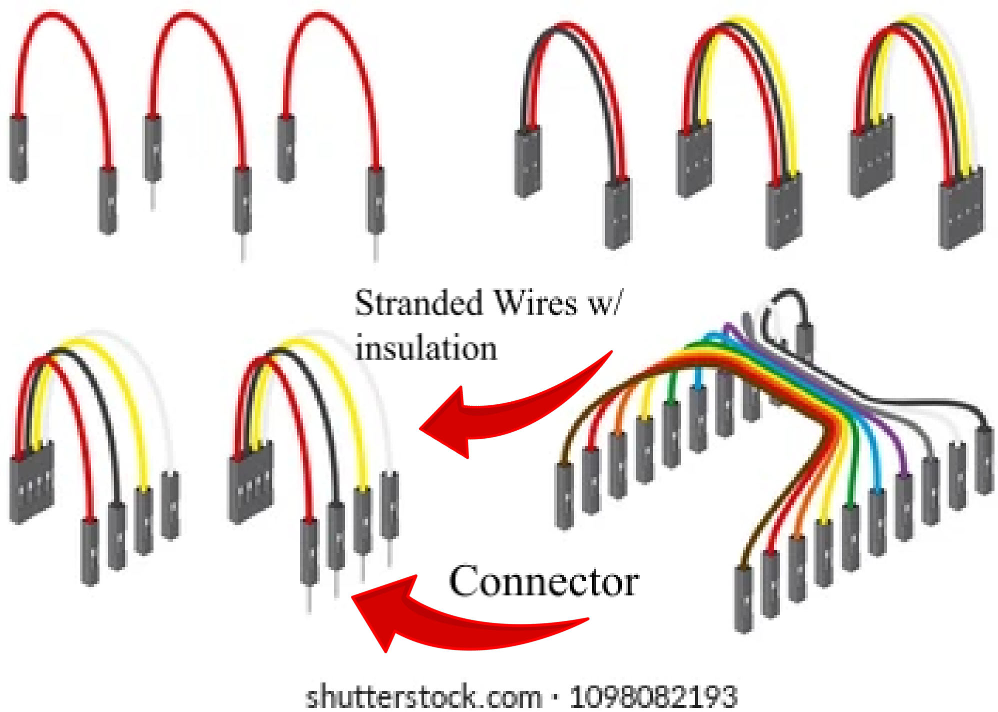


A Breadboard is used to connect the necessary connections in a neat and tidy manner without the mess of wires. Also extends the dedicated power and ground Pins found in the ESP32 microcontroller for more sensors and devices to be used.

PARTS:

- **Power Rails:** Connects and adds more devices that connect to either the positive (+) or negative (-) of the power rails.
    - Red (+) - Connects to the Positive end of a power source or a battery.
    - Black (-) - Connects to the Negative end of a power source or a battery to make sure your devices are working properly.
- **Terminal Strips** - Connects to devices like the ESP32 Microcontroller. These Terminal Strips are connected vertically and these terminal strips usually contain 5 holes per side in a breadboard.
- **DIP Support** - Divides the breadboard in two separate strips and its purpose is to connect devices like ESP32 Microcontroller properly while not connecting other I/O pins together.

Figure 8: Jumper Wires



Jumper Wires are used in the making of prototype as this prototype is for school works only and was not required to spend money on making the prototype compact. These Jumper Wires are connected to and from the ESP32 Microcontroller and other devices and sensors like the Capacitive Soil Sensor and 4 Module relay.

PARTS:

- **Connector** - Located at each end of a Jumper wire, its purpose is to connect one end of a device or a breadboard to the other end of a breadboard like connecting a jumper wire from the capacitive soil moisture sensor to the ESP32 Microcontroller.
- **Stranded Wires w/Insulation** - Located inside the insulation layer of a jumper wire (colored) and its purpose is to properly transfer energy from one end to the other. The insulation's purpose is to not get shocked by the jumper wire when transferring power but also you can properly find where your jumper wire is connected to as when connecting many jumper wires in a breadboard can be very confusing.

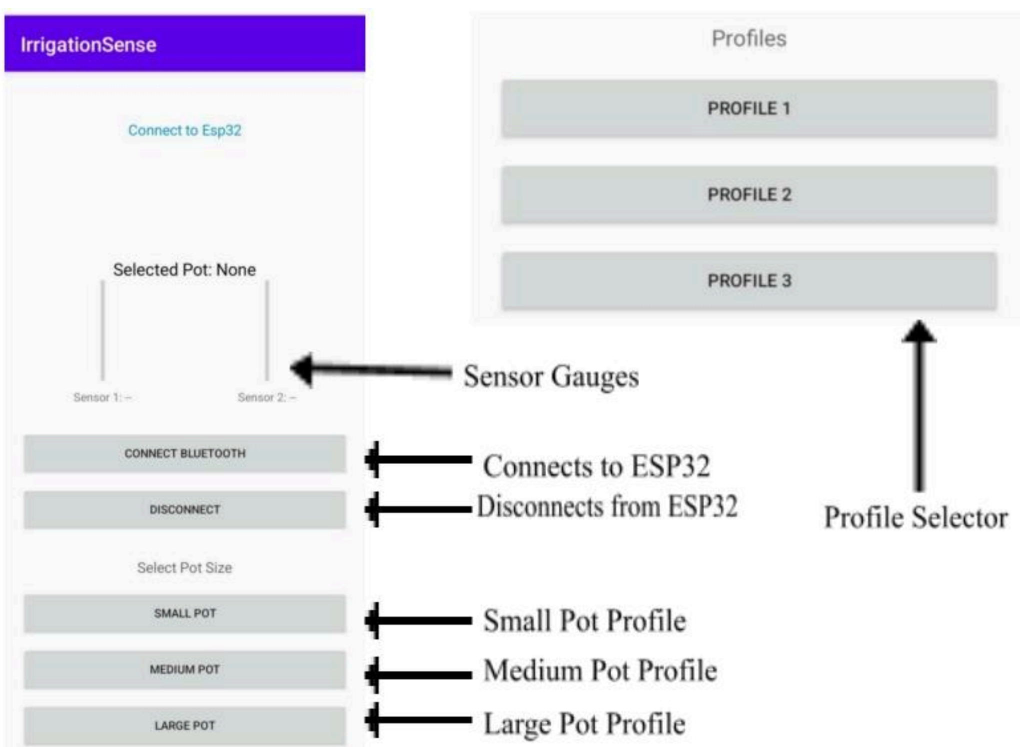
4.2 Software Development

Mobile Application

Figure 9: Android Studio Logo

Android Studio is a software used to code and create programs exclusively for Android devices. The **IrrigationSense App** was created and developed with the **Android Studio** software to simplify pairing and control with the hardware (ESP 32 Microcontroller). The interface includes:

Figure 10 : IrrigationSense Application



PARTS:

- **Sensor Gauges** - Two Gauges are found on the app and its purpose is to let the user know if the moisture of the soil is low (Dry) or High (Wet) with a set value threshold.
- **Connects of ESP32** - This button's purpose is to connect with ESP32 Microcontroller and its Bluetooth functionality to your device like a smartphone in order to interact with the hardware on the ESP32.
- **Disconnects from ESP32** - This button's purpose is to disconnect from the ESP32 Microcontroller when it is not in use. It also gives the ESP32 into sleep mode in order to conserve power.
- **Small Pot Profile** - The purpose of this is to let the ESP32 Microcontroller you are using a small pot and that pressing this button turns on the water pump at a desired mL (5mL).
- **Medium Pot Profile** - The purpose of this is to let the ESP32 Microcontroller you are using a medium pot and that pressing this button turns on the water pump at a desired mL (10mL).
- **Large Pot Profile** - The purpose of this is to let the ESP32 Microcontroller you are using a large pot and that pressing this button turns on the water pump at a desired mL (20mL).
-

4.3 Hardware Logic

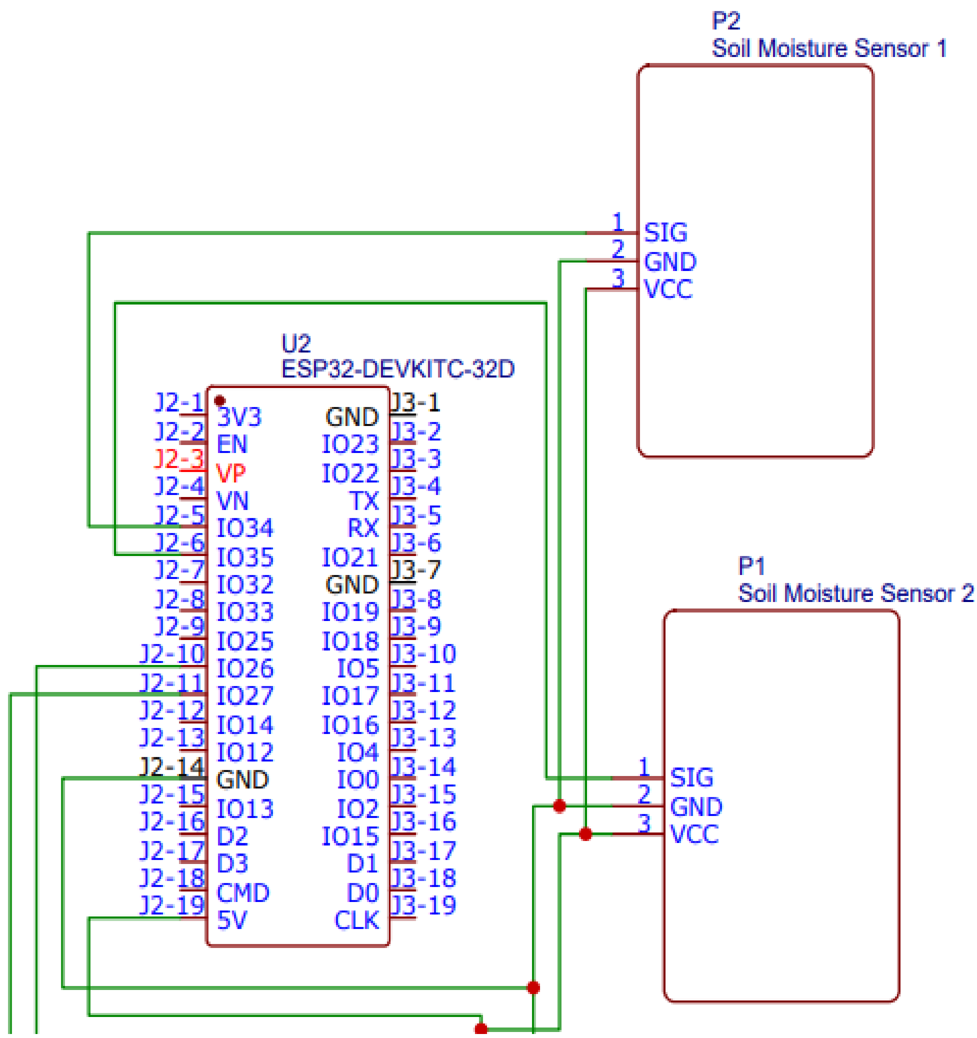
Figure 11: Arduino IDE Logo

The implementation for the Hardware functionality and Logic is created with the Arduino IDE Software as the ESP32 Microcontroller is compatible with the Arduino IDE Software in creating and implementing code for our hardware. We set what pin number found on the ESP32 Microcontroller will be used in order for the Devices (Soil Moisture

Sensor and Relay) to function. Here we also implement the Bluetooth capabilities of our Microcontroller in order to interact and send data values to the IrrigationSense Application.

4.4 Circuit Schematic

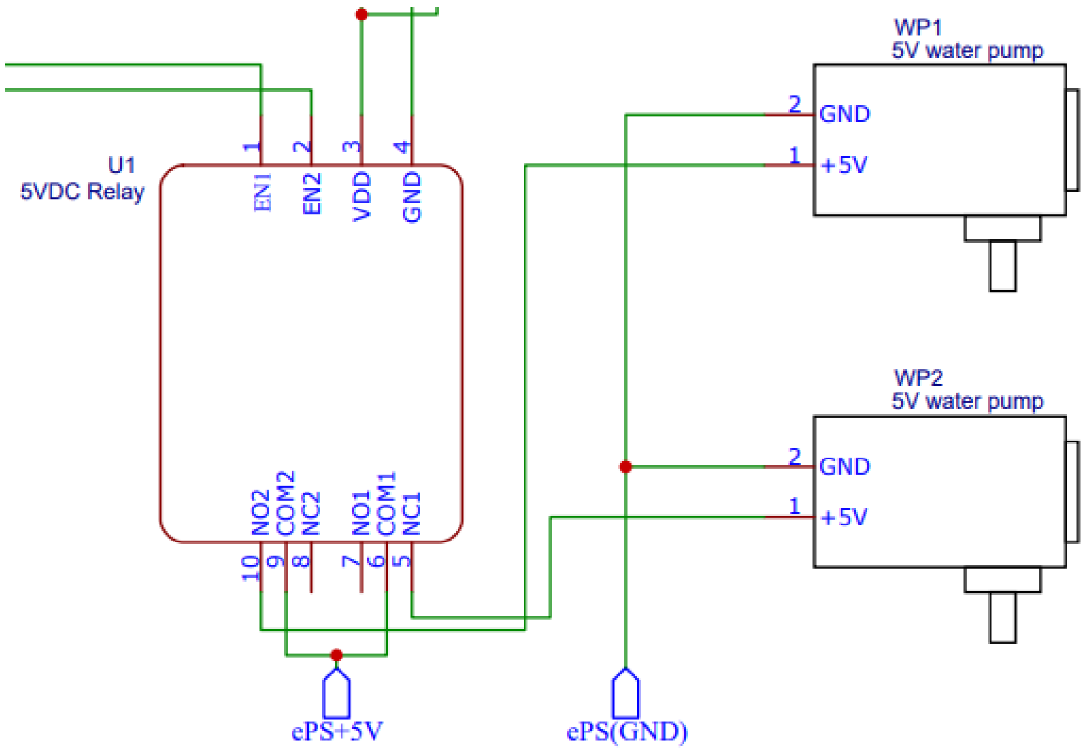
Figure 12: Connections for Soil Moisture Sensor



Legend:

U2 = ESP32 Microcontroller	SIG = Signal	IO34 = Pin 34
P1 = Soil Moisture Sensor 2	5V= Power	IO 35 = Pin 35
P2 = Soil Moisture Sensor 1	GND = Ground	

Figure 12.1: Water pump and relay connections



Legend:

ePS+5V = ExternalVariable PowerSupply(5V)
ePS(GND) = ExternalVariable PowerSupply(GND)
NC1 = Normally Closed 1
NC2 = Normally Closed 2
COM1 = Common 1
COM2 = Common 2
WP2 = 5V Water Pump
WP1 = 5V Water Pump

EN1 = Relay Input 1
EN2 = Relay Input 2
VDD = 5V
GND = Ground
U1 = 5VDC Relay
IO27 = Pin 27
IO26 = Pin 26

4.5 Implementation Process

4.5.1 Circuit Assembly

As shown in Figure 11, prepare all the necessary connections from each components as shown:

- Connect the **ESP32 Microcontroller** to the breadboard for ease of installation when adding devices to the pins of the ESP32.

- The **ESP32 Microcontroller** is connected to power via the USB micro-B port and is connected to a Power supply (Power bank).
- Connect **5V and Ground pins** of the ESP32 to the power rails found on the breadboard for expandability and usability of the pins for more devices.
- For the **capacitive soil moisture sensor**, connect the **Red (VCC) and Black (GND)** pins to power rails of the breadboard and the **Yellow (Data)** pin to **Pins 34 and 35** of the ESP32 Microcontroller.
- For the **4 relay module**, Connect **VCC and GND** pins to the **power rails** found on the breadboard and for the Input (IN), **connect IN1 and IN2** to **Pin 26 and 27** in the ESP 32 Microcontroller respectively.
- For the **water pumps**, **connect the Red wire (VCC)** to the **Normally Closed** port of the relay and the **Black wire (GND)** to the **external power supply ground (Variable).**
- Make sure the **COM ports** found on the 4 module relay are **connected to the Positive end (VCC)** of the external Power supply (Variable).

4.5.2 Plant Profile Calibration

- Using a measuring cylinder, the pump's runtime was calibrated to approximately deliver 5, 10, and 20 mL.

- Moisture thresholds were determined through trial and error, set at 40% as the trigger point.

4.5.3 Testing Environment

- Tests were conducted indoors using **two types of potted plants** to simulate **small and medium** irrigation needs. The large plant profile was excluded from testing due to the experiment's scope and limitations.
- Bluetooth performance was evaluated in various indoor locations. The system maintained stable connectivity at distances up to **8 meters**, confirming reliable offline control using Bluetooth communication.

Note on Large Plant Profile Testing

While the system was designed to support three plant profiles—**small (5 mL)**, **medium (10 mL)**, and **large (20 mL)**—only the **small** and **medium** profiles were included in the experimental trials.

The **large plant profile** was excluded due to the following limitations:

- The **DC water pump** used in the prototype had a limited flow rate, which made it difficult to consistently deliver 20 mL within the programmed runtime.

- The available test setup did not include sufficiently large containers or soil volumes representative of actual large-plant conditions.

- Time constraints and hardware limitations during prototype calibration restricted the scope of profile testing.

Despite this, the **system logic for large-profile watering (20 mL)** was fully implemented in the software, including Bluetooth command handling and pump activation. Future testing with higher-capacity pumps and larger plant containers is recommended to validate the large-profile setting in real-world scenarios.

4.6 Design Challenges and Solutions

Challenge	Solution Implemented
Moisture sensor readings fluctuated	An averaging algorithm was implemented to stabilize the sensor data and reduce noise.
Inconsistent water output from pumps	The pump runtime was calibrated through trial and error, and delay control was added.
Limited testing for large-profile plants	Testing was focused on small and medium profiles due to flow rate and container size constraints. The logic for large profile was still implemented in the app and system.

Table 4: This table summarizes key issues encountered during the design and testing phases, along with the corresponding solutions applied to improve accuracy and system performance.

4.7 Summary

This chapter outlined the comprehensive design and implementation process of the Bluetooth-based smart irrigation system using the ESP32 microcontroller. The system integrates capacitive soil moisture sensors, a relay-controlled water pump, and an Android-based mobile application for Bluetooth communication.

Key milestones include:

- The successful assembly of all hardware components, including sensors, pumps, and relay modules.
- The development of a custom Android app (IrrigationSense) for remote control and profile selection.
- The calibration of pump runtimes for **small** and **medium** plant profiles, based on precise water delivery (5 mL and 10 mL respectively).
- Reliable indoor Bluetooth operation up to 8 meters.

Although the **large plant profile** was not tested physically due to system constraints, the software logic for its operation was fully developed and validated. The system's modular design ensures it can be extended or scaled for future enhancements.

By focusing on **affordability**, **offline control**, and **plant-specific water efficiency**, the system offers a practical irrigation solution for small-scale agricultural users and home gardeners, especially in areas without stable internet connectivity.

CHAPTER 5
RESULTS AND DISCUSSION

A quantitative-experimental approach was employed to evaluate the performance of the low-cost Bluetooth-based smart irrigation system using ESP32. This chapter presents, analyzes, and interprets all data collected in relation to the questions posed in the study, focusing on sensor precision, watering accuracy, water-savings impact, and overall system reliability.

5.1 Results

Basis for Tables 5.1 and 5.2

PRECISION (%)	REMARK
85 – 100	Precise
70 – 84.99	Moderate
Below 70	Inaccurate

Table 5.0: This table provides the classification criteria used to evaluate sensor precision in Tables 5.1 and 5.2. Sensor precision was computed using the formula:

- Precision (%) = (1 – |Actual – Sensor Reading| ÷ Actual) × 100

Based on the resulting percentage, sensor performance was categorized into three ranges:

- Precise (85–100%): Sensor readings are very close to the actual moisture levels.
- Moderate (70–84.99%): Sensor shows acceptable but less accurate results.
- Inaccurate (below 70%): Readings deviate significantly from actual values.

This classification scale ensured objective evaluation of the ESP32 moisture sensor's accuracy in the field trials.

Table 5.1 Sensor Precision in Measuring Soil Moisture (Small-Profile Plant)

TRIAL	Actual Moisture (%)	Sensor Reading (%)	Precision (%)	Remarks
1	18	20	90	Precise
2	20	22	90.91	Precise

3	20	23	86.96	Precise
4	23	24	95.83	Precise
5	23	24	95.83	Precise
6	23	24	95.83	Precise
7	25	26	96.15	Precise
8	25	26	96.15	Precise
9	25	26	96.15	Precise
10	26	28	92.86	Precise
Average	**22.8**	**24.3**	**93.83**	**Precise**

Table 5.1: Presents the results of ten soil moisture measurements for a small-profile plant, comparing actual moisture levels obtained through the oven-dry method with the readings from the ESP32-connected capacitive sensor. Precision was calculated using the standard formula to assess how closely the sensor aligned with true moisture content. The average precision of 93.83% demonstrates the sensor's strong accuracy and suitability for monitoring moisture in small-plant environments.

Legend.
Actual Moisture (%): reference value measured by the oven-dry method.
Sensor Reading (%): value reported by the capacitive sensor via ESP32.
Precision (%): closeness of sensor reading to actual value.
Remarks: evaluation based on the precision scale above.

Interpretation. Precision ranged from 86.96 % to 96.15 %, with an average of 93.83 %, indicating that the sensor produced highly reliable readings for the small-profile plant.

Table 5.2 Sensor Precision in Measuring Soil Moisture (Medium-Profile Plant)

TRIAL	Actual Moisture (%)	Sensor Reading (%)	Precision (%)	Remarks
1	31	32	96.88	Precise
2	26	27	96.3	Precise
3	25	27	92.59	Precise
4	25	27	92.59	Precise

5	24	25	96	Precise
6	24	25	96	Precise
7	24	26	92.31	Precise
8	23	26	88.46	Precise
9	23	24	95.83	Precise
10	23	24	95.83	Precise
Average	**24.8**	**26.3**	**94.3**	**Precise**

Table 5.2: Shows precision analysis for the medium-profile plant across ten trials. Like Table 5.1, it compares the actual and sensor-reported values to evaluate the system's consistency. With an average precision of 94.30%, the results confirm that the sensor maintains a high degree of reliability for medium-sized plants, consistently surpassing the threshold for "Precise" performance.

Legend.
Actual Moisture (%): reference value measured by the oven-dry method.
Sensor Reading (%): value reported by the capacitive sensor via ESP32.
Precision (%): closeness of sensor reading to actual value.
Remarks: evaluation based on the precision scale above.

Interpretation. All ten trials achieved at least 88 % precision, with an overall average of 94.30 %, confirming consistent sensor accuracy for the medium-profile plant.

5.1.1 Water-Delivery Precision

Profile	Target Output (mL)	Actual Output (mL)	Deviation (mL)	Remarks
Small	5	5.2	0.2	Precise
Medium	10	9.7	–0.3	Precise

Table 5.3: Summarizes the calibrated water output of the system for small and medium plant profiles. Target volumes were set at 5 mL and 10 mL, respectively, and the actual water dispensed was measured using a graduated cylinder. Deviations were all within ±1 mL, affirming that the motor-pump timing logic effectively achieves dosage-based irrigation accuracy.

Pump runtime calibration achieved deviations below ±1 mL, meeting the precision standard for plant-specific watering.

Figure 13: Water-Savings Impact

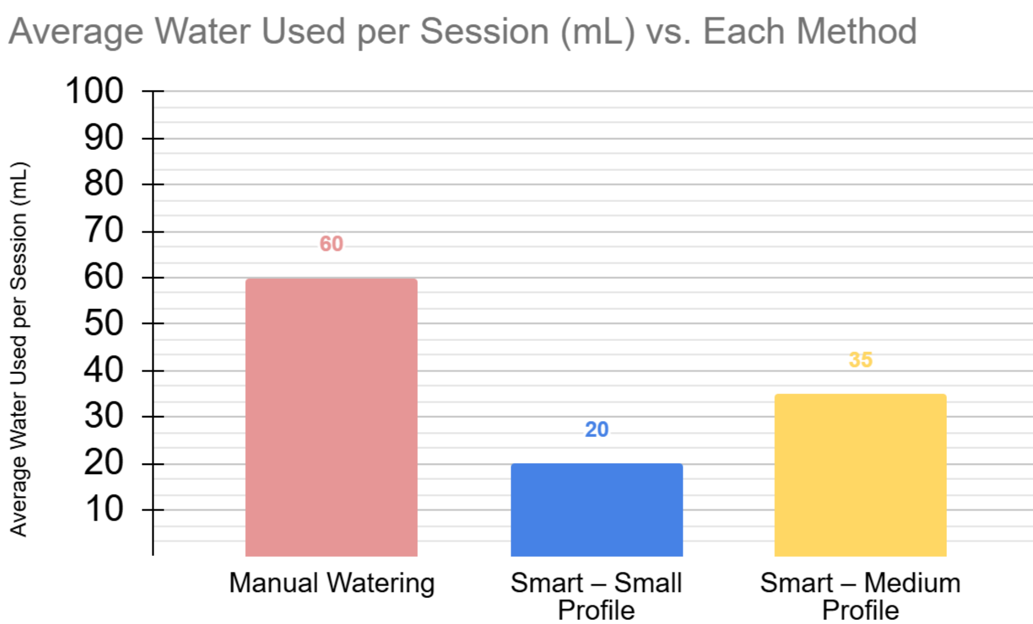


Figure 13: Presents a comparative bar chart illustrating the reduction in water consumption between manual and automated watering methods. The smart irrigation system achieved savings of **35–50%**, supporting its effectiveness as a sustainable and resource-efficient alternative, especially for areas with limited water availability.

5.2 Discussion of Findings

1. **Sensor Precision.** Tables 5.1 and 5.2 show overall precisions of **93.83 %** (Small) and **94.30 %** (Medium). All readings exceeded the 85 % threshold, confirming that the capacitive sensor is precise and reliable.
2. **Water‑Delivery Accuracy.** Deviation values of **±0.2 mL** (Small) and **±0.3 mL** (Medium) indicate that the system dispenses water volumes very close to the calibrated targets, preventing both over‑ and under‑watering.
3. **Water Conservation.** The bar‑graph comparison confirms up to **50 % water savings**. Such efficiency aligns with the objective of developing a sustainable irrigation solution for offline or low‑resource environments.
4. **System Reliability.** Bluetooth communication remained stable within an 8‑meter indoor range, and the system triggered watering precisely at the 40 % moisture threshold in all observed cases.

5. **Overall Effectiveness.** Combining high sensor accuracy, precise water delivery, and significant water savings, the system demonstrates high effectiveness and practical relevance for small-scale agricultural users lacking internet connectivity.

5.3 Summary

The quantitative results confirm that the Bluetooth-based smart irrigation system:

- Measures soil moisture with **> 93 % precision**.
- Delivers water volumes with **< ±1 mL deviation**.
- Achieves **35–50 % water savings** over manual methods.
- Operates reliably within an **8 m indoor Bluetooth range**.

These findings collectively validate the system's viability as a low-cost, offline solution for precision irrigation.

CHAPTER 6
CONCLUSION AND RECOMMENDATION

This chapter presents the conclusions derived from the findings of the study and offers informed recommendations based on the system's performance. The research explored the effectiveness and practicality of a Bluetooth-based smart irrigation system using ESP32 designed for offline agricultural applications.

6.1 Conclusions

The following conclusions are drawn based on the data analysis and the fulfillment of the research objectives:

1. Precision in Sensing Soil Moisture

The system demonstrated a high degree of precision in soil moisture detection, with an average accuracy of **93.83%** for the small-plant profile and **94.30%** for the medium-plant profile. These results confirm the reliability and consistency of the capacitive sensor used with the ESP32 microcontroller.

2. Precision in Preventing Overwatering and Underwatering

Based on moisture readings before and after irrigation, the system triggered watering accurately only when moisture fell below the threshold (40%), and the calibrated water outputs (5 mL for small, 10 mL for medium) were successfully achieved. This ensured precise water delivery based on need.

3. Effectiveness in Terms of Convenience and Moisture Detection

The combination of real-time sensor feedback, Bluetooth control, and simple plant profile selection made the system user-friendly. The plant-specific calibration enabled convenient and accurate moisture detection and response.

4. Water Conservation Impact

The system achieved a water usage reduction of **35–50%** compared to manual watering practices. This supports its application in water-scarce and resource-limited settings.

5. Suitability for Offline and Low-Cost Agricultural Use

With a total cost of only **₱1,441** and no need for internet connectivity, the system proved viable for home gardens and small-scale farms, particularly in rural areas with limited infrastructure.

6.2 Recommendations

Based on the study's results and conclusions, the following recommendations are offered for future development and implementation:

1. Broaden Calibration Across More Soil Types

Test and calibrate the system on various soil textures (e.g., loam, sandy, clay) to maintain precision in diverse environments.

Basis: Although the current results were precise, they were based on a limited soil type.

2. Integrate Environmental Sensors

Add sensors for rain or humidity to enhance watering decisions and avoid redundancy during natural rainfall.

Basis: The system relies solely on soil moisture, which may not reflect sudden weather changes.

3. Improve App Interface and Control Options

Include scheduling, real-time monitoring, and customizable thresholds in the mobile app to enhance usability and control.

Basis: Users could benefit from more flexible control options and feedback.

4. Add Visual Feedback for System Status (Optional LED Indicator)

Although LEDs were not used in this version, future implementations may consider adding a visual indicator for Bluetooth connection or moisture alerts.

Basis: This improves user interaction, especially in outdoor or low-tech scenarios.

5. Develop User Support Materials

Create user manuals, setup guides, and tutorial videos to improve adoption and ensure ease of use for non-technical users.

Basis: Accessibility and acceptance can be strengthened through better user education.

6.3 Final Remarks

The Bluetooth-based smart irrigation system using ESP32 successfully addressed the problem of inefficient manual watering in offline, low-resource agricultural settings. It proved to be reliable, cost-effective, and easy to use, and it demonstrated high accuracy in both sensing and water delivery. This project serves as a practical foundation for future work in smart farming technologies aimed at sustainability and rural innovation.

References:

Ameer Baig, M., & Khan, S. (2023). Smart irrigation system using IoT. *International Journal of Civil Engineering and Technology, 1*(1), 13–19. Retrieved from https://www.espjournals.org/IJCEET/2023/Volume1-Issue1/IJCEET-V1I1P103.pdf

Gutiérrez, J., Villa-Medina, J. F., Nieto-Garibay, A., & Porta-Gándara, M. Á. (2014). Automated irrigation system using a wireless sensor network and GPRS module. *IEEE Transactions on Instrumentation and Measurement, 63*(1), 166–176. https://doi.org/10.1109/TIM.2013.2276487

Mehta, K. R., Naidu, J., Baheti, M., Parmar, D., & Sharmila, A. (2023). Internet of Things based smart irrigation system using ESP WROOM 32. *Journal on Internet of Things, 5*, 45–55. https://doi.org/10.32604/jiot.2023.043102

Pereira, G. P., Chaari, M. Z., & Daroge, F. (2023). IoT-enabled smart drip irrigation system using ESP32. *IoT, 4*(3), 221–243. https://doi.org/10.3390/iot4030012

Tyagi, S., Anand, R., Sabharwal, A., & Reddy, S. R. N. (2024). Plant recommendation system using smart irrigation integrated with IoT and machine/deep learning. *Communications in Soil Science and Plant Analysis.* https://doi.org/10.1080/00103624.2024.2367035

Gupta, H., & Aggarwal, R. (2020). Bluetooth-based irrigation system: An affordable solution for offline agriculture. *International Journal of Computer Applications, 975*, 1–6. https://doi.org/10.5120/ijca2020920201

Sahu, S., & Mohanty, S. P. (2020). Design and development of an IoT-based smart irrigation system using ESP32. *Materials Today: Proceedings, 21*, 1056–1061. https://doi.org/10.1016/j.matpr.2019.11.104

Tyagi, S., Singh, R., & Sharma, S. (2021). Machine learning-based smart irrigation system. *Journal of Agricultural Informatics, 12*(2), 45–52.

Sahu, P., & Mohanty, S. (2020). Smart irrigation system using capacitive soil moisture sensor and microcontroller. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, 9*(4), 1234–1240.

Gupta, A., & Aggarwal, R. (2020). Bluetooth-controlled irrigation system using ESP32. *International Journal of Engineering and Technology, 12*(3), 234–239.

Appendices

Appendix A: Complete Arduino IDE Code (ESP32)

TERM:

BLOCK is a group of lines of code that are written together to perform a specific task.

Library Inclusion (BLOCK 1)

```
#include "BluetoothSerial.h"
#include "esp_sleep.h"
```

Purpose:

These lines tell the program to include additional functionality for Bluetooth communication and deep sleep (power-saving mode).

Declaring Bluetooth and Pins (BLOCK 2)

```
BluetoothSerial SerialBT;
const int soilSensor1Pin = 34;
const int soilSensor2Pin = 35;
const int relay1Pin = 26;
const int relay2Pin = 27;
```

Purpose:

Sets up Bluetooth functionality (SerialBT).

Defines pins:

- Two pins for moisture sensors (**soilSensor1Pin**, **soilSensor2Pin**).
- Two pins for relays that activate water pumps (**relay1Pin, relay2Pin**).

Variables for Moisture and Timing (BLOCK 3)

```
const int moistureThreshold = 2500;
bool isConnected = false;
bool wateringEnabled = false;
unsigned long lastSendTime = 0;
const unsigned long sendInterval = 2000;
int pumpDurationSeconds = 3;
```

Purpose:

moistureThreshold: Moisture level that decides if watering is needed.

isConnected: Checks if Bluetooth is currently connected.

wateringEnabled: Whether the system should activate pumps.

lastSendTime and **sendInterval**: Control how often moisture readings are sent via Bluetooth.

pumpDurationSeconds: Determines how many seconds to run the pumps.

Bluetooth Connection Handling (Callback) (BLOCK 4)

```
void btCallback(esp_spp_cb_event_t event, esp_spp_cb_param_t *param) {
  if (event == ESP_SPP_SRV_OPEN_EVT) {
    isConnected = true;
    Serial.println("Bluetooth connected");
  } else if (event == ESP_SPP_CLOSE_EVT) {
    isConnected = false;
    Serial.println("Bluetooth disconnected");
  }
}
```

Purpose:

This block responds whenever a Bluetooth device connects or disconnects:

- Sets the connection status.
- Prints status messages to the console.

Sleep Function (Power Saving) (BLOCK 5)

```
void goToSleep() {
  Serial.println("Disabling Bluetooth and going to deep sleep...");
  SerialBT.end();
  delay(100);
  esp_deep_sleep_start();
}
```

Purpose:
When called, this function:

- Turns off Bluetooth.
- Waits briefly.
- Puts the device into deep sleep (a power-saving mode).

Setup Function (Initial Configuration) (BLOCK 6)

```
void setup() {
  Serial.begin(19200);
  SerialBT.begin("ESP32_MoistureMonitor");
  SerialBT.register_callback(btCallback);

  pinMode(relay1Pin, OUTPUT);
  pinMode(relay2Pin, OUTPUT);
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, HIGH);

  Serial.println("ESP32 Ready");
  esp_sleep_enable_timer_wakeup(10ULL * 1000000);
```

Purpose:

This setup does several things when you first power on the device:

- Starts serial communication (for console messages).
- Activates Bluetooth communication with the device name "ESP32_MoistureMonitor".
- Assigns the Bluetooth event handler (btCallback).
- Sets relay pins to output mode (ready to activate pumps).
- Prints "ESP32 Ready" to the console.
- Sets a timer so the device can wake itself from sleep after 10 seconds if it goes to sleep.

Main Loop (Core Functionality)

This section repeats continuously.

A. Bluetooth Command Handler (BLOCK 7)

```
if (SerialBT.available()) {
  String cmd = SerialBT.readStringUntil('\n');
  cmd.trim();
  Serial.println("Received: " + cmd);


  if (cmd.equalsIgnoreCase("DISCONNECT")) {
    SerialBT.println("Disconnecting and going to sleep...");
    SerialBT.flush();
    delay(100);
    goToSleep();

  } else if (cmd.startsWith("POT:")) {
    wateringEnabled = true;
    if (cmd.equalsIgnoreCase("POT:SMALL")) {
      pumpDurationSeconds = 3;
    } else if (cmd.equalsIgnoreCase("POT:MEDIUM")) {
      pumpDurationSeconds = 6;
    } else if (cmd.equalsIgnoreCase("POT:LARGE")) {
      pumpDurationSeconds = 9;
    }
    Serial.println("Pump duration set to " + String(pumpDurationSeconds) + " seconds");
  }
}
```

```
if (moisture2 > moistureThreshold) {
    Serial.println("Soil 2 dry, watering for " + String(pumpDurationSeconds) + " seconds");
    digitalWrite(relay2Pin, LOW);
    delay(pumpDurationSeconds * 1500);
    digitalWrite(relay2Pin, HIGH);
  } else {
    digitalWrite(relay2Pin, HIGH);
  }
} else {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, HIGH);
}
```

Purpose:

Checks if watering is activated **(wateringEnabled).**

If the soil is dry **(moisture > moistureThreshold):**

Turns on pumps (relay goes LOW).

Keeps pumps running for a set duration (pumpDurationSeconds).

Turns pumps off again after watering.

If watering is off, keeps pumps deactivated.

Waits 1.5 seconds before checking again.

Purpose:

This main loop does the core functions repeatedly:

- Checks for Bluetooth commands from a phone or app:
  - **"DISCONNECT"**: shuts down and goes into power-saving sleep mode.
  - **"POT:SMALL/MEDIUM/LARGE":** enables watering and sets pump duration.
- Ensures pumps stay off if Bluetooth is not connected.
- Reads moisture sensor values from two soil sensors.
- Sends these moisture readings over Bluetooth every 2 seconds.
- If watering is enabled:
  - Checks if soil is too dry:
    - Activates pump briefly if needed.
  - Otherwise, keeps pumps turned off.
- Waits for a short delay before repeating the loop again.

B. Managing Connection State (BLOCK 8)

```
if (!isConnected) {
  digitalWrite(relay1Pin, HIGH);
  digitalWrite(relay2Pin, HIGH);
  return;
}
```

Purpose:

Ensures pumps remain off when no device is connected via Bluetooth.

Appendix B: Android App Code (Kotlin

NOTE: Lines starting with // are comments. These explain what each line or section of code does, making it easier to understand.

EXAMPLE:
COMMENT:
// The package name for your app (should be the first line of the file)
CODE:
packagecom.example.irrigationsense

ACTUAL KOTLIN CODE:
// The package name for your app (should be the first line of the file)

package com.example.irrigationsense

// Import statements: these allow you to use code from other libraries in your project

// AppCompatActivity is the main base class for screens in Android apps

import androidx.appcompat.app.AppCompatActivity

// Lets you request permissions at runtime (e.g., for Bluetooth)

import androidx.core.app.ActivityCompat

// Lets you access app resources and colors safely

import androidx.core.content.ContextCompat

// Used for specifying permissions (e.g., Bluetooth, Location)

import android.Manifest

```
// Lets you suppress lint warnings in code (like permission checks)
import android.annotation.SuppressLint

// For displaying alert dialogs/pop-up messages
import android.app.AlertDialog

// For controlling Bluetooth hardware on the phone
import android.bluetooth.BluetoothAdapter     // Represents Bluetooth radio on device
import android.bluetooth.BluetoothDevice      // Represents a paired Bluetooth device
import android.bluetooth.BluetoothSocket      // Used for direct Bluetooth connections

// For checking device's Android version
import android.os.Build

// Used to pass info between activities and handle activity lifecycle
import android.os.Bundle

// Widgets for building your app's user interface
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast
```

```
// For reading incoming data from Bluetooth connection
import java.io.InputStream

// For UUIDs and utility classes (needed for Bluetooth)
import java.util.*

@SuppressLint("MissingPermission")  // Ignores lint warnings about missing Bluetooth permissions
class MainActivity : AppCompatActivity() {
    // A text label for showing connection or app status
    private lateinit var tvStatus: TextView

    // Text labels for showing the current moisture readings from sensor 1 and sensor 2
    private lateinit var tvSensor1: TextView
    private lateinit var tvSensor2: TextView

    // Button to connect to a Bluetooth device
    private lateinit var btnConnect: Button

    // Button to disconnect from a Bluetooth device
    private lateinit var btnDisconnect: Button

    // Buttons for choosing the pot size, which affects watering duration
    private lateinit var btnSmallPot: Button
```

```kotlin
private lateinit var btnMediumPot: Button

private lateinit var btnLargePot: Button


// Progress bars (gauges) to visually show the moisture level for each sensor

private lateinit var gaugeSensor1: ProgressBar

private lateinit var gaugeSensor2: ProgressBar


// Text label to show which pot size is currently selected

private lateinit var tvSelectedPot: TextView


// Buttons to quickly save or load preferred watering profiles

private lateinit var btnProfile1: Button

private lateinit var btnProfile2: Button

private lateinit var btnProfile3: Button


// Keeps track of which pot size the user has selected (SMALL, MEDIUM, LARGE, or NONE)

private var selectedPotSize: String = "NONE"


// Stores saved pot size profiles; lets users save/retrieve their favorite settings

private val profiles = mutableMapOf<Int, String>()


// This provides access to the phone's Bluetooth system; initialized only when needed ("lazy")

private val bluetoothAdapter: BluetoothAdapter? by lazy {
```

// Gets the Bluetooth manager service from the system

val manager = getSystemService(BLUETOOTH_SERVICE) as? android.bluetooth.BluetoothManager

// Returns the Bluetooth adapter, or gets a default adapter if manager is null

manager?.adapter ?: BluetoothAdapter.getDefaultAdapter()

}

// This holds the current Bluetooth connection (the "socket" is the communication channel)

private var bluetoothSocket: BluetoothSocket? = null

// This will be the background thread that keeps listening for data from the Bluetooth device

private var readThread: Thread? = null

// This is the starting point when the app screen is created

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

// Sets the layout of the screen using activity_main.xml

setContentView(R.layout.activity_main)

// These lines connect the code variables to the actual UI elements in your app layout

tvStatus = findViewById(R.id.tvStatus) *// Finds the TextView for showing status messages*

tvSensor1 = findViewById(R.id.tvSensor1) *// Finds the TextView for sensor 1 reading*

tvSensor2 = findViewById(R.id.tvSensor2) *// Finds the TextView for sensor 2 reading*

```
btnConnect = findViewById(R.id.btnConnect)         // Finds the "Connect" button
btnDisconnect = findViewById(R.id.btnDisconnect)   // Finds the "Disconnect" button

btnSmallPot = findViewById(R.id.btnSmallPot)       // Finds the button for "Small Pot"
btnMediumPot = findViewById(R.id.btnMediumPot) // Finds the button for "Medium Pot"
btnLargePot = findViewById(R.id.btnLargePot)   // Finds the button for "Large Pot"

gaugeSensor1 = findViewById(R.id.gaugeSensor1)   // Finds the progress bar for sensor 1
gaugeSensor2 = findViewById(R.id.gaugeSensor2) // Finds the progress bar for sensor 2

tvSelectedPot = findViewById(R.id.tvSelectedPot)   // Finds the TextView for displaying the selected pot size

btnProfile1 = findViewById(R.id.btnProfile1)       // Finds the first profile button
btnProfile2 = findViewById(R.id.btnProfile2)       // Finds the second profile button
btnProfile3 = findViewById(R.id.btnProfile3)       // Finds the third profile button

// Checks if the Android version is 12 (S) or newer
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // If yes, requests Bluetooth permissions from the user at runtime
    ActivityCompat.requestPermissions(
```

```
                this, // context (this activity)
                arrayOf(
                    Manifest.permission.BLUETOOTH_CONNECT, // permission to connect with Bluetooth devices
                    Manifest.permission.BLUETOOTH_SCAN     // permission to scan for Bluetooth devices
                ),
                1 // request code (any unique integer)
            )
        }


        // When the "Connect" button is pressed, open the list of paired Bluetooth devices
        btnConnect.setOnClickListener { showDevicePicker() }


        // When the "Disconnect" button is pressed, show a confirmation dialog
        btnDisconnect.setOnClickListener {
            AlertDialog.Builder(this)
                .setTitle("Confirm Disconnect")   // Dialog title
                .setMessage("Are you sure you want to disconnect the Bluetooth device?") // Dialog message
                .setPositiveButton("Yes") { _, _ -> disconnectBluetooth() } // If user selects "Yes", disconnect
                .setNegativeButton("Cancel", null) // If user selects "Cancel", do nothing
                .show() // Display the dialog
        }
```

```
    // When the "Small Pot" button is pressed:
    btnSmallPot.setOnClickListener {
        selectedPotSize = "SMALL"                    // Save the selected pot size
        sendBluetoothCommand("POT:SMALL")            // Send the command via Bluetooth
        tvSelectedPot.text = "Selected Pot: Small"       // Update the display
        Toast.makeText(this, "Selected Small Pot", Toast.LENGTH_SHORT).show()
 // Show a quick popup message
    }


    // When the "Medium Pot" button is pressed:
    btnMediumPot.setOnClickListener {
        selectedPotSize = "MEDIUM"               // Set selected pot size to "MEDIUM"
            sendBluetoothCommand("POT:MEDIUM")  // Send Bluetooth command for medium pot
        tvSelectedPot.text = "Selected Pot: Medium"     // Update displayed selected pot
        Toast.makeText(this, "Selected Medium Pot", Toast.LENGTH_SHORT).show()
// Show popup message
    }


    // When the "Large Pot" button is pressed:
    btnLargePot.setOnClickListener {
        selectedPotSize = "LARGE"                    // Set selected pot size to "LARGE"
          sendBluetoothCommand("POT:LARGE")  // Send Bluetooth command for large pot
```

```
        tvSelectedPot.text = "Selected Pot: Large"          // Update displayed selected pot
        Toast.makeText(this, "Selected Large Pot", Toast.LENGTH_SHORT).show()
// Show popup message
    }

        // When a profile button is pressed, call handleProfile with the profile number (1, 2, or 3)
    btnProfile1.setOnClickListener { handleProfile(1) }
    btnProfile2.setOnClickListener { handleProfile(2) }
    btnProfile3.setOnClickListener { handleProfile(3) }
    }

    // This function saves or loads a profile, depending on if it exists
    private fun handleProfile(profileId: Int) {
    val existing = profiles[profileId]  // Check if this profile already has a saved pot size

    if (existing == null && selectedPotSize != "NONE") {
        // If there's no saved pot size and a pot size is selected, save it to this profile
        profiles[profileId] = selectedPotSize
        Toast.makeText(this, "Saved $selectedPotSize to Profile $profileId", Toast.LENGTH_SHORT).show()
    } else if (existing != null) {
        // If this profile already has a pot size, load it, send it to the device, and update display
        selectedPotSize = existing
```

```
        sendBluetoothCommand("POT:$selectedPotSize")

        tvSelectedPot.text = "Selected Pot: ${selectedPotSize.capitalize()}"

        Toast.makeText(this, "Loaded Profile $profileId: $selectedPotSize", Toast.LENGTH_SHORT).show()

    } else {

        // If nothing is selected and the profile is empty, show a warning message

        Toast.makeText(this, "No pot size selected to save to Profile $profileId", Toast.LENGTH_SHORT).show()

    }

    }


    private fun showDevicePicker() {

    // Get the list of Bluetooth devices already paired with the phone

    val pairedDevices = bluetoothAdapter?.bondedDevices


    // If there are no paired devices, update the status message and stop

    if (pairedDevices.isNullOrEmpty()) {

        tvStatus.text = "No paired Bluetooth devices found."

        return

    }


    // Create an array of device names (or addresses if the name isn't available)

    val deviceNames = pairedDevices.map { it.name ?: it.address }.toTypedArray()

    // Turn the set of paired devices into a list (so we can access by index)

    val deviceList = pairedDevices.toList()
```

```
    // Show a pop-up dialog listing all paired devices
    AlertDialog.Builder(this)
        .setTitle("Select a device") // Dialog title
        .setItems(deviceNames) { _, which ->
            // When a device is selected, connect to it
            connectToDevice(deviceList[which])
        }
        .show() // Display the dialog
}

private fun connectToDevice(device: BluetoothDevice) {
    // Update the status label to show that a connection attempt is starting
    tvStatus.text = "Connecting to ${device.name}..."

    // If there is already a background thread reading data, stop it
    readThread?.interrupt()

    // Start a new background thread for connecting and communicating with the device
    Thread {
        try {
            // Bluetooth SPP (Serial Port Profile) UUID; used for most Bluetooth devices
            val uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
```

```
// Create a Bluetooth socket for communication with the selected device
val socket = device.createRfcommSocketToServiceRecord(uuid)
// Cancel device discovery (speeds up connection)
bluetoothAdapter?.cancelDiscovery()
// Attempt to connect to the Bluetooth device
socket.connect()

// Save this socket for later communication
bluetoothSocket = socket

// Update the status on the main screen (must run on UI thread)
runOnUiThread {
tvStatus.text = "Connected to ${device.name}"
}

// Prepare to read incoming data from the Bluetooth device
val inputStream: InputStream = socket.inputStream
val buffer = ByteArray(1024) // Buffer for incoming bytes

// Start a new background thread to keep reading data from the Bluetooth device
readThread = Thread {
try {
// This loop runs as long as the thread is not interrupted
```

```
while (!Thread.currentThread().isInterrupted) {
        val bytes = inputStream.read(buffer) // Read data into the buffer
        if (bytes > 0) {
        // Convert the received bytes into a string, trimming any extra spaces
        val data = String(buffer, 0, bytes).trim()

        // Check if the data contains readings from both sensors
        if (data.contains("Sensor1:") && data.contains("Sensor2:")) {
        // Split the string by commas (example: "Sensor1:123,Sensor2:456")
        val parts = data.split(",")
        // Find and extract the value for Sensor 1
        val sensor1 = parts.find { it.startsWith("Sensor1:") }
          ?.substringAfter("Sensor1:")?.trim() ?: "0"
        // Find and extract the value for Sensor 2
        val sensor2 = parts.find { it.startsWith("Sensor2:") }
          ?.substringAfter("Sensor2:")?.trim() ?: "0"
// Update the user interface on the main thread with the new sensor values
        runOnUiThread {
                // Show the latest sensor readings as text
          tvSensor1.text = "Sensor 1: $sensor1"
          tvSensor2.text = "Sensor 2: $sensor2"

    // Convert the sensor values to integers (default to 0 if conversion fails)
                val s1 = sensor1.toIntOrNull() ?: 0
```

```
val s2 = sensor2.toIntOrNull() ?: 0
val threshold = 2000 // Set threshold for soil moisture

// Calculate a "moisture percent" for each sensor (for the progress bar)
val percent1 = (100 - (s1.coerceIn(0, 4095) * 100 / 4095))
val percent2 = (100 - (s2.coerceIn(0, 4095) * 100 / 4095))

// Update the progress bars to reflect soil moisture percentage
gaugeSensor1.progress = percent1
gaugeSensor2.progress = percent2

// Resource IDs for progress bar colors (blue and red)
val blue = R.drawable.circular_progress_blue
val red = R.drawable.circular_progress_red

// Change gauge color depending on the soil moisture value
gaugeSensor1.progressDrawable = ContextCompat.getDrawable(
    this@MainActivity,
    if (s1 < threshold) red else blue // Red if below threshold, blue if above
)
gaugeSensor2.progressDrawable = ContextCompat.getDrawable(
    this@MainActivity,
```

```
                    if (s2 < threshold) red else blue
                    )
                }
            }
        }
    }

    // If there is a problem reading from the Bluetooth device, catch the error
    } catch (e: Exception) {
    e.printStackTrace()  // Print the error for debugging
    runOnUiThread {
        tvStatus.text = "Disconnected" // Update status to show disconnected
    }
    }
    }
    readThread?.start() // Start the thread that listens for incoming sensor data

// If there's a problem connecting to the Bluetooth device, catch the error
} catch (e: Exception) {
    runOnUiThread {
    tvStatus.text = "Connection failed: ${e.message}" // Show error message to user
    }
    e.printStackTrace() // Print the error for debugging
```

```kotlin
        }
    }.start()   // Start the background thread that manages the connection
    }

    // Sends a command to the connected Bluetooth device (like "POT:SMALL" or "DISCONNECT")
    private fun sendBluetoothCommand(command: String) {
    try {
        // Get the output stream from the Bluetooth connection and send the command
        bluetoothSocket?.outputStream?.apply {
            write("$command\n".toByteArray()) // Write the command, ending with a newline
            flush() // Make sure all data is sent out right away
        }
    } catch (e: Exception) {
        // If there is an error, update the status and print the error for debugging
        runOnUiThread {
            tvStatus.text = "Error sending command"
        }
        e.printStackTrace()
    }
    }

    // Safely disconnects from the Bluetooth device and resets the app's state
    private fun disconnectBluetooth() {
```

```
try {
    // Send a "DISCONNECT" command to the Bluetooth device (if possible)
    bluetoothSocket?.outputStream?.write("DISCONNECT\n".toByteArray())
    bluetoothSocket?.outputStream?.flush()
} catch (_: Exception) {} // Ignore errors here, since we're disconnecting anyway

// Stop the background thread that listens for sensor data
readThread?.interrupt()
readThread = null

try {
    // Close the Bluetooth connection/socket
    bluetoothSocket?.close()
} catch (_: Exception) {} // Ignore errors here too

// Clear the socket variable (no longer connected)
bluetoothSocket = null
  // Update the app's display to show that it is now disconnected
runOnUiThread {
    tvStatus.text = "Disconnected"      // Show status message
    tvSensor1.text = "Sensor 1: --"     // Clear Sensor 1 display
    tvSensor2.text = "Sensor 2: --"     // Clear Sensor 2 display
    gaugeSensor1.progress = 0           // Reset progress bar for Sensor 1
    gaugeSensor2.progress = 0           // Reset progress bar for Sensor 2
```

```
    }
    }

    // This function runs automatically when the app screen is closed or destroyed
    override fun onDestroy() {
    super.onDestroy()
    disconnectBluetooth() // Make sure Bluetooth is safely disconnected when app closes
    }
}
```

Code for Application Design

NOTE: Lines starting with <!-- are comments. These explain what each line or section of code does, making it easier to understand.

EXAMPLE:

COMMENT:

<!--This layout describes the design for your main Android app screen.

A ScrollView allows the entire content to scroll if it doesn't fit on one screen.-->

CODE:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
         android:padding="16dp">
```

ACTUAL CODE:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This layout describes the design for your main Android app screen.
```

```xml
A ScrollView allows the entire content to scroll if it doesn't fit on one screen.
-->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
            android:padding="16dp">

            <!-- Main vertical layout that holds all the UI components -->
            <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:gravity="center_horizontal"
        android:paddingBottom="16dp">

        <!-- Shows the connection status at the top -->
            <TextView
            android:id="@+id/tvStatus"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="32dp"
            android:layout_marginBottom="16dp"
            android:padding="90dp"
            android:text="Connect to Esp32"
```

```
android:textAppearance="@style/TextAppearance.AppCompat.Medium"
android:textColor="@android:color/holo_blue_dark" />

<!-- Displays the currently selected pot size -->
<TextView
android:id="@+id/tvSelectedPot"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Selected Pot: None"
android:textSize="21sp"
android:textColor="@android:color/black"
android:layout_marginTop="12dp"
android:layout_gravity="center_horizontal" />

<!-- Horizontal layout for two progress bars and their sensor readings -->
<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:gravity="center"
android:layout_marginBottom="24dp">

<!-- Sensor 1: Gauge + label -->
<LinearLayout
```

```
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
android:orientation="vertical"
android:gravity="center_horizontal">

    <!-- Moisture level gauge for Sensor 1 -->
    <ProgressBar
  android:id="@+id/gaugeSensor1"

style="@android:style/Widget.DeviceDefault.Light.ProgressBar.Horizontal"
  android:layout_width="100dp"
  android:layout_height="100dp"
  android:indeterminate="false"
    android:max="100"
  android:progress="0"
  android:rotation="-90"
  android:layout_marginBottom="8dp" />

    <!-- Text label for Sensor 1 -->
    <TextView
  android:id="@+id/tvSensor1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
```

```
            android:text="Sensor 1: --" />
    </LinearLayout>

    <!-- Sensor 2: Gauge + label -->
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical"
        android:gravity="center_horizontal">

            <!-- Moisture level gauge for Sensor 2 -->
            <ProgressBar
          android:id="@+id/gaugeSensor2"

style="@android:style/Widget.DeviceDefault.Light.ProgressBar.Horizontal"
          android:layout_width="100dp"
          android:layout_height="100dp"
          android:indeterminate="false"
            android:max="100"
          android:progress="0"
          android:rotation="-90"
          android:layout_marginBottom="8dp" />
```

```
        <!-- Text label for Sensor 2 -->
        <TextView
      android:id="@+id/tvSensor2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
        android:text="Sensor 2: --" />
  </LinearLayout>
  </LinearLayout>

<!-- Button for connecting to Bluetooth -->
  <Button
  android:id="@+id/btnConnect"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Connect Bluetooth"
  android:layout_marginBottom="8dp" />

   <!-- Button for disconnecting Bluetooth -->
  <Button
  android:id="@+id/btnDisconnect"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Disconnect"
  android:layout_marginBottom="24dp" />
```

```
<!-- Label for the pot size selection section -->

<TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Select Pot Size"

android:textAppearance="?android:attr/textAppearanceMedium"

android:layout_marginBottom="12dp" />


<!-- Buttons for selecting pot size -->

<Button

android:id="@+id/btnSmallPot"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:text="Small Pot"

android:layout_marginBottom="8dp" />


<Button

android:id="@+id/btnMediumPot"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:text="Medium Pot"

android:layout_marginBottom="8dp" />
```

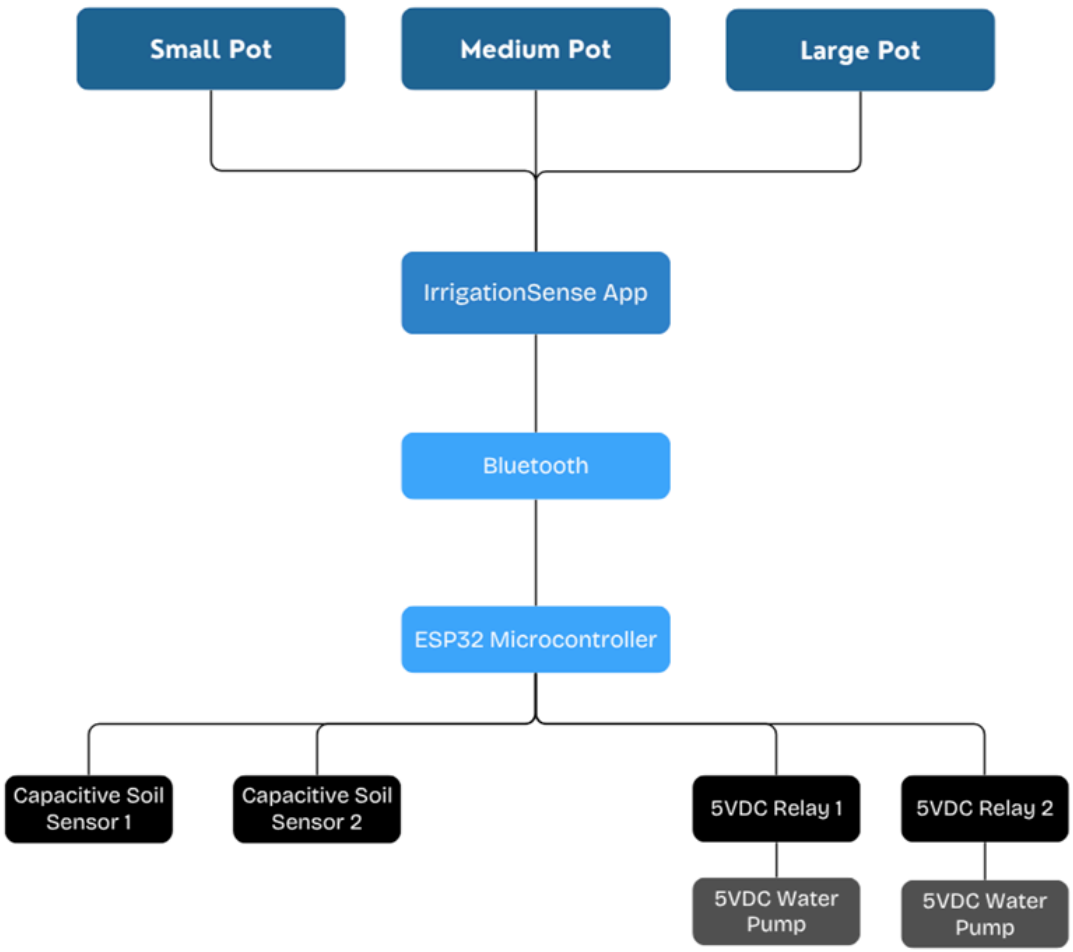
```xml
<Button
android:id="@+id/btnLargePot"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Large Pot"
android:layout_marginBottom="8dp" />


<!-- Label for profiles section -->
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Profiles"
android:textAppearance="?android:attr/textAppearanceMedium"
android:layout_marginBottom="12dp" />

<!-- Buttons for user profiles -->
<Button
android:id="@+id/btnProfile1"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Profile 1"
android:layout_marginBottom="8dp" />
```

```
        <Button
        android:id="@+id/btnProfile2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Profile 2"
        android:layout_marginBottom="8dp" />


    <Button
        android:id="@+id/btnProfile3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Profile 3"
        android:layout_marginBottom="8dp" />

        </LinearLayout>
    </ScrollView>
```

Appendix C: Schematic Diagram and System Diagram

System Diagram

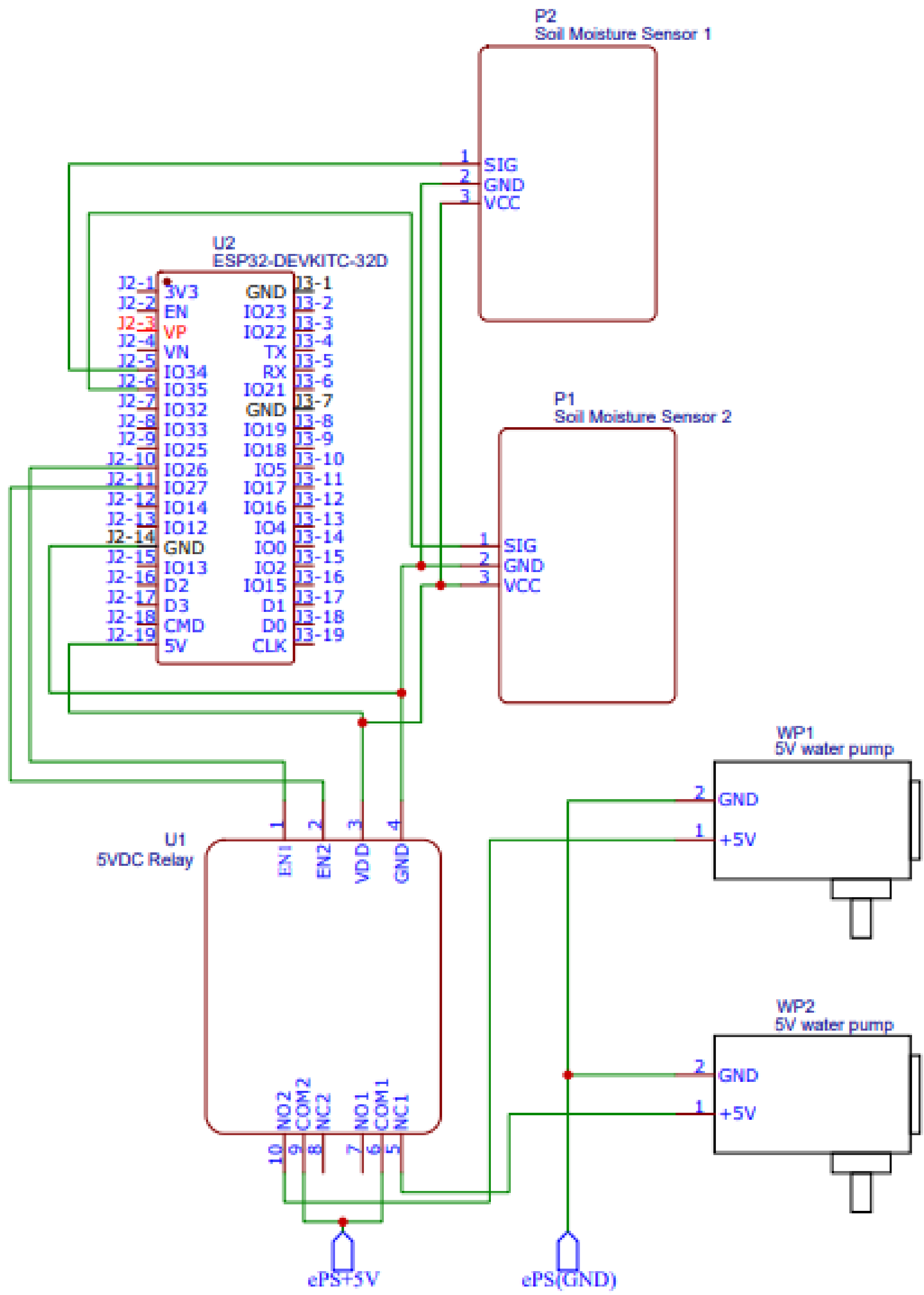
Figure 14: Hardware and Software block diagram



This block diagram shows a step-by-step breakdown on when you select a pot size (Small, Medium or Large) through the Application (IrrigationSense), and sends that data via Bluetooth and received in the hardware prototype (ESP32 Microcontroller) and the Capacitive Soil Sensor reads the value of the soil.

Schematic Diagram

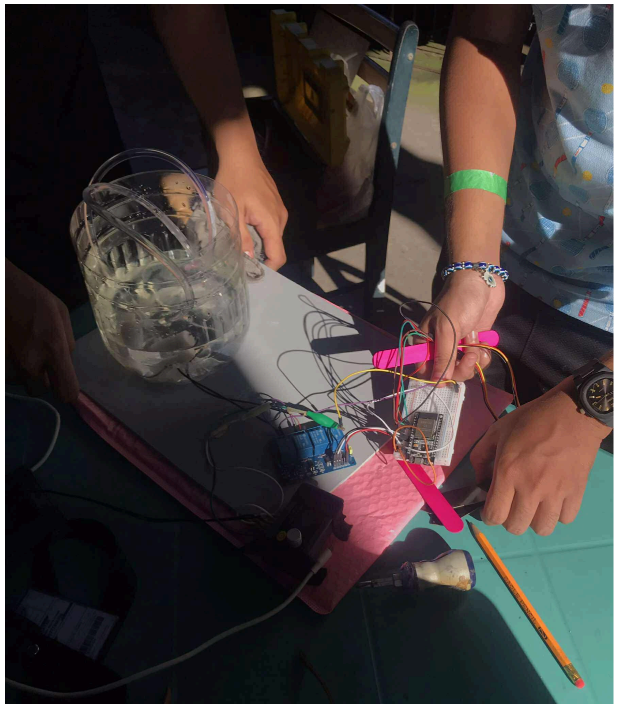
Figure 15: Schematic diagram for the ESP32 hardware prototype



The schematic shown in the picture above shows how the components like the 5VDC Water Pump (WP1/WP2), Relay (U1), and the Capacitive Soil Moisture Sensor (P1/P2) connect to the appropriate GPIO Pins in the ESP32 Microcontroller (Pins 26, 27,34 and 35) to function properly as what it is intended.

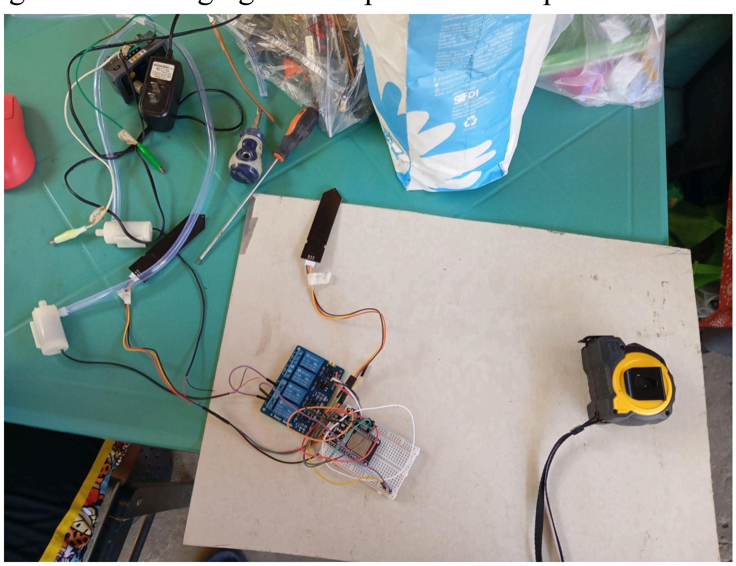
Project Documentation

Figure 16: Creating and Testing the Hardware Prototype



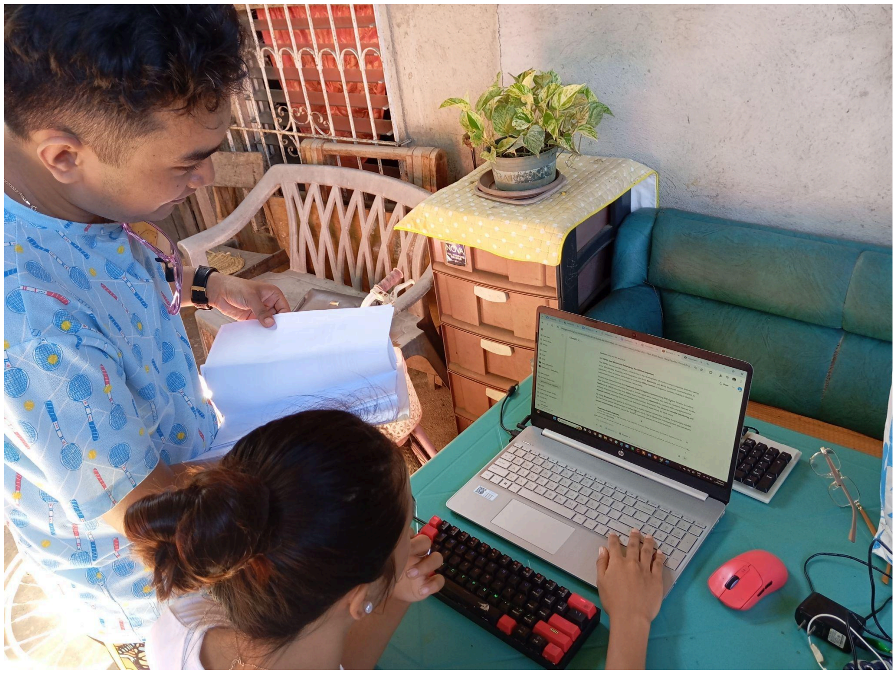
The figure shows the researcher preparing the hardware/components needed to be put on a board for support.

Figure 17: Arranging the components to be put on the board



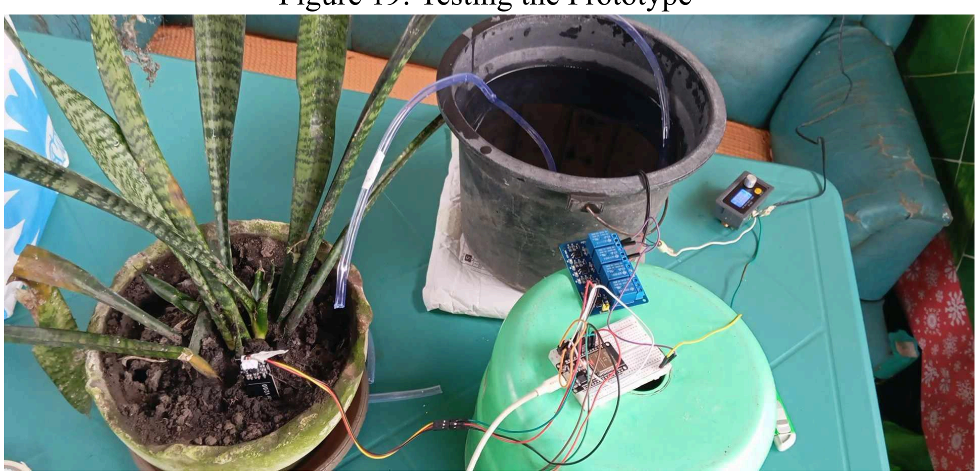
The figure shows the researchers preparing the board to have the components to be put in the board.

Figure 18: Collecting and gathering data



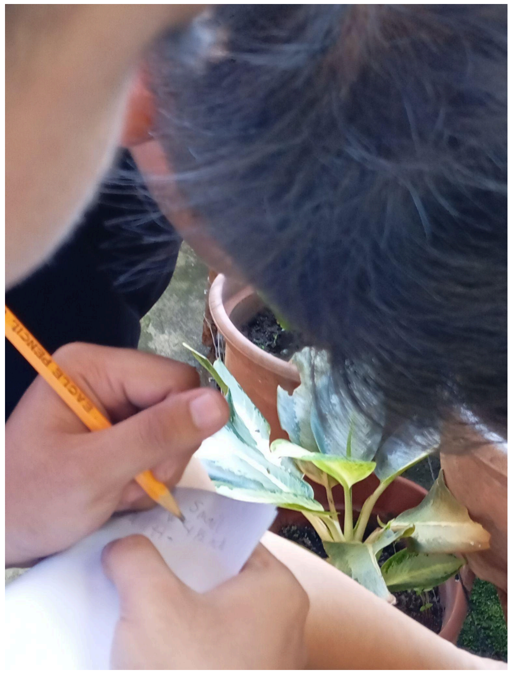
In this figure, the researchers are collecting and gathering data based on the results on the hardware to be put and interpreted in the research manuscript.

Figure 19: Testing the Prototype



In this figure, the researchers are testing the components of the prototype to make sure that all components are working as intended before putting all the components on a board.

Figure 20: Writing the dimensions and sizes for flower pot



In this figure, the researchers are calculating the dimensions of the flower pot that will later be designated as Small and Medium flower pots.

Figure 21: Measuring the dimensions for flower pots

In this figure, the researchers are measuring using measuring tape, the dimensions of Small and Medium flower pots.

CURRICULUM VITAE

Gravador, Godfrey Angelou B.

Address: Purok Molave, Tubtubon, Sibulan, Negros Oriental

Contact No: 09056003365

Email Address: godfreygravador33@gmail.com

Birthdate: December 30, 2003

Age: 21 **Sex:** Male

Educational Background

- **Tertiary Education**

  Negros Oriental State University Main Campus I & II
  Dumaguete City, Negros Oriental, Philippines
  Bachelor of Science in Computer Engineering
  S.Y. 2020-2025

- **Secondary Education**

  Senior High School:
  Ajong National High School
  Ajong, Sibulan, Negros Oriental
  SY: 2019-2021

  Junior High School:
  Ajong National High School
  Ajong, Sibulan, Negros Oriental
  SY: 2015-2019

- **Primary Education**

  Sibulan Central Elementary School
  Yapsutco St, Sibulan, Negros Oriental
  SY: 2009-2015

- **Trainings Attended**

  N/A

- **Experiences**

  N/A

- **Skills**

  Computer Hardware and Software Literate

CURRICULUM VITAE

Amorin, Christian Meden G.

Address: Purok Kalubihan, Daro, Dumaguete City

Contact No: 09268263558

Email Address: camorin23@gmail.com

Birthdate: April 4, 2001

Age: 24 **Sex:** Male

Educational Background

- **Tertiary Education**

  Negros Oriental State University Main Campus I & II
  Dumaguete City, Negros Oriental, Philippines
  Bachelor of Science in Computer Engineering
  S.Y. 2020-2025

- **Secondary Education**

  Senior High School:
  Ramon Teves Pastor Memorial- Dumaguete Science High School
  Ma. Asuncion Village, Barangay Daro, Dumaguete City, Negros Oriental
  SY: 2018-2020

  Junior High School:
  Catherina Cittadini School
  Calindagan, Dumaguete City, Negros Oriental
  SY: 2015-2018

- **Primary Education**

  Catherina Cittadini School
  Calindagan, Dumaguete City, Negros Oriental
  SY: 2008-2014

- **Trainings Attended**

  N/A

- **Experiences**

  N/A

- **Skills**

  Computer Hardware and Software literate

CURRICULUM VITAE

Repollo, Reheca

Address: Poblacion, Pamplona, Negros Oriental

Contact No: 09284116047

Email Address: rehecaaaaaaaa11@gmail.com

Birthdate: November 27, 2003

Age: 21 **Sex:** Female

Educational Background

- **Tertiary Education**

  Negros Oriental State University Main Campus I & II
  Dumaguete City, Negros Oriental, Philippines
  Bachelor of Science in Computer Engineering
  S.Y. 2021-2025

- **Secondary Education**

  Senior High School:
  La Consolacion College,
  Bais City, Negros Oriental
  SY: 2019-2021

  Junior High School:
  Pamplona National High School
  Cambalud, Pamplona, Negros Oriental
  SY: 2015-2019

- **Primary Education**

  Pamplona Central Elementary School
  Poblacion, Pamplona, Negros Oriental
  SY: 2009-2015

- **Trainings Attended**
  N/A
- **Experiences**
  N/A
- **Skill/s**
  Computer Literate

CURRICULUM VITAE

Velez, Jera Dinah F.

Address: Purok Everlasting,Taclobo, Dumaguete City

Contact No: 09662307558

Email Address: vjeradinah@gmail.com

Birthdate: November 1, 2002

Age: 22 **Sex:** Female

Educational Background

- **Tertiary Education**

  Negros Oriental State University Main Campus I & II
  Dumaguete City, Negros Oriental, Philippines
  Bachelor of Science in Computer Engineering
  S.Y. 2020-2025

- **Secondary Education**

  Senior High School:
  Jose B. Cardenas Memorial High School - Main Campus
  Exodus Avenue, Panubigan, Canlaon City, Negros Oriental
  SY: 2018-2020

  Junior High School:
  Saint Francis High School Incorporated
  Poblacion, Vallehermoso, Negros Oriental
  SY: 2015-2018

- **Primary Education**

  Don Vicente Lopez Sr. Memorial
  Elementary School
  Bagawines, Vallehermoso, Negros
  Oriental
  SY: 2008-2014

- **Trainings Attended**
  N/A
- **Experiences**
  Crew at Spincam Duma (2024-Present)
  - **Skills**
    N/A