Smart Fertigation System for Precision Agriculture

LEOW KIEN ZEN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Electronics Engineering with honours

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2026

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : LEOW KIEN ZEN

ID No. : 2202372

Date : 11/5/2026

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"Smart Fertigation System for Precision Agriculture"** was prepared by **LEOW KIEN ZEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to thank everyone who contributed to the successful completion of this project. My deepest gratitude goes to my research supervisor, Dr. TEH PEH CHIONG, for his invaluable advice, continuous guidance, and patience throughout the development of this research. His encouragement and expertise were essential in shaping this project.

I would also like to express my heartfelt appreciation to my loving parents for their unwavering support, understanding, and encouragement during the entire course of my study. Their motivation has always been my greatest source of strength.

Special thanks are also extended to my friends who provided assistance, shared ideas, and offered moral support throughout this journey

Lastly, I am grateful to the university and the Faculty of Engineering for providing the necessary resources and facilities that made this project possible.

Without the support and contributions from all these individuals, the completion of this project titled “Smart Fertigation System for Precision Agriculture” would not have been possible.

Smart Fertigation System for Precision Agriculture

ABSTRACT

The increasing demand for sustainable and efficient agriculture has driven the adoption of smart farming technologies. This project focuses on the development of a Smart Fertigation System for Precision Agriculture, applying both IoT and industrial automation approaches for the cultivation of ginger plants.

For the indoor setup, an ESP32 microcontroller was used to build a prototype fertigation system. Sensors such as pH, electrical conductivity (EC), and DS18B20 temperature sensors were employed to monitor the growth environment, while a water pump controlled via a relay module regulated the delivery of nutrients to the plants. This setup demonstrates how a low-cost microcontroller can automate fertilizer application in a controlled environment.

In addition, an outdoor system was developed using an AMX-FX3U-M26MR-E Programmable Logic Controller (PLC) to manage agricultural processes in the field for ginger cultivation. The PLC was integrated with an ESP32 over Modbus TCP/IP via the UTARSF2 Wi-Fi network, enabling remote access, real-time monitoring, and flexible management through a web-based MQTT dashboard.

By implementing both an ESP32-based prototype and a PLC-based control system, this project highlights two complementary approaches to smart agriculture tailored for ginger farming. The systems improve precision in fertilizer application, reduce manual labour, and provide remote accessibility, contributing to more efficient and sustainable agricultural practices.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AMX-FX3U-M26MR-E	Model designation of the PLC used in the outdoor system
C/C++	C and C++ programming languages
CPU	Central Processing Unit
EC	Electrical Conductivity
ESP32	Espressif Systems 32-bit microcontroller
GPIO	General Purpose Input/Output
HMI	Human-Machine Interface
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
LoRa	Long Range (wireless communication protocol)
LoRaWAN	Long Range Wide Area Network
MQTT	Message Queuing Telemetry Transport
NPK	Nitrogen, Phosphorus, Potassium (fertilizer)
pH	Potential of Hydrogen
PLC	Programmable Logic Controller
RAM	Random Access Memory
SDG	Sustainable Development Goal
TCP/IP	Transmission Control Protocol / Internet Protocol
TDS	Total Dissolved Solids
TP-Link MR105	Model of Wi-Fi access point used in the outdoor system
UTARSF2	UTAR Smart Farming 2 (Wi-Fi network name)

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Agriculture has always been a vital sector for sustaining human life and supporting economic development. Traditional farming practices face challenges such as inefficient use of water and fertilizer, heavy labour requirements, and limited monitoring of crop conditions. Malaysia's agricultural farms still depend heavily on manpower to manually monitor farm conditions. In order to solve these issues, many universities have started to research precision agriculture and found that the best solution lies in smart farming technologies — systems that integrate sensors, automation, and communication to optimise farming processes, improve productivity, reduce wastage, and support sustainable practices.

One area of development is the use of IoT-based systems for small-scale or controlled environment agriculture. Such systems are typically built around an ESP32 microcontroller, which costs approximately RM20–25, and can connect to sensors to monitor parameters like pH, electrical conductivity (EC), and temperature. With these sensors, relay modules, and pumps, it is possible to control when and how nutrients are delivered. This approach is particularly useful in research or indoor farming where precise fertigation is required.

In larger or outdoor environments, more robust and reliable solutions are often needed. Programmable Logic Controllers (PLCs) are widely used in industrial automation and can be applied to agriculture for controlling equipment in the field. At

UTAR's agricultural park, which is fully exposed to sunlight, PLCs are the preferred choice due to their resilience to heat and harsh environmental conditions. With the integration of modern communication protocols such as MQTT and Modbus TCP/IP, PLCs can be connected to existing networks via an ESP32 bridge, allowing remote monitoring and control from distant locations.

This project therefore explores both approaches: an indoor ESP32-based smart fertigation prototype designed to demonstrate the fundamentals of the fertigation process, and an outdoor PLC-based control system implemented at the agricultural park with full IoT remote access via ESP32.

This project is aligned with the United Nations Sustainable Development Goals (SDGs), particularly SDG 2 (Zero Hunger), SDG 9 (Industry, Innovation and Infrastructure), and SDG 12 (Responsible Consumption and Production). SDG 2 calls for sustainable food production systems and the adoption of resilient agricultural practices that increase productivity. By automating fertilizer and water delivery through a smart fertigation system, this project directly supports more productive and efficient ginger cultivation. SDG 9 encourages the development of quality, reliable, and sustainable infrastructure through the adoption of technology and innovation. The integration of IoT-based monitoring using ESP32 and industrial-grade PLC automation with MQTT and Modbus TCP/IP communication represents a practical application of modern engineering in the agricultural sector. SDG 12 promotes the responsible management of natural resources. By delivering nutrients and water only when required — based on real-time pH, EC, and temperature sensor readings — this system minimises fertilizer wastage and reduces water overconsumption, contributing to more sustainable agricultural resource management.

1.2 Importance of the Study

Agriculture today requires more efficient methods to reduce labour, save resources, and increase crop yield. This study is important because it explores two different approaches to smart farming. The indoor ESP32-based fertigation system

demonstrates how low-cost sensors and automation can optimise fertilizer delivery for controlled environment agriculture. Meanwhile, the outdoor PLC system shows how industrial automation can be connected through MQTT and Modbus TCP/IP communication over the UTARSF2 Wi-Fi network, enabling remote monitoring and control via an ESP32.

By addressing both small-scale prototyping and larger-scale applications, this project highlights practical solutions that improve efficiency, reduce manual intervention, and support the adoption of smart agriculture technologies in Malaysia.

1.3 Problem Statement

To cultivate ginger, proper control of water and nutrient supply is needed to increase crop quality and ensure even growth. Traditional farming methods depend on manual labour for irrigation and fertilizer application, requiring operators to physically monitor key parameters such as pH, electrical conductivity (EC), temperature, and water levels. These parameters are critical for healthy plant growth, and without automated monitoring, resources are wasted and productivity is reduced.

Another problem is that with traditional farming methods, manual monitoring of agricultural equipment is time-consuming and limits operational flexibility. This issue is particularly significant when dealing with large-scale crops in outdoor environments such as agricultural parks. Without remote access, farmers and operators must remain physically present at the site, making the entire farming process less efficient and increasing the risk of delayed responses to abnormal conditions.

Therefore, a smart agriculture system with the ability to monitor, control, and automate the farming process is needed for ginger cultivation. For indoor or small-scale environments, an ESP32-based system can automate the fertigation process. For large-scale outdoor environments, PLCs provide the necessary industrial-grade control. Integrating a PLC with ESP32 using Modbus TCP/IP and MQTT communication provides reliable remote control. Solving these problems reduces

manual intervention, improves precision in nutrient management, and enhances overall farming efficiency.

1.4 Aim and Objectives

The aim of this project is to design and implement smart fertigation and control systems for ginger cultivation by utilising both IoT-based and industrial automation approaches, with the goal of improving precision, efficiency, and remote accessibility in agriculture.

The objectives of this project are as follows:

i. To develop an indoor smart fertigation system using an ESP32 microcontroller, integrated with pH, EC, and DS18B20 temperature sensors, to automate nutrient and water delivery for ginger plants.
ii. To implement and test an outdoor PLC-based control system for managing agricultural processes in a field environment.
iii. To establish and evaluate remote communication for the PLC system using Modbus TCP/IP and the MQTT protocol via the UTARSF2 Wi-Fi network.
iv. To assess the overall performance and effectiveness of both systems in supporting precision agriculture practices for ginger cultivation.

1.5 Scope and Limitation of the Study

This project focuses on the application of automation and communication technologies in ginger cultivation. The scope is divided into two parts:

i. An indoor smart fertigation system using an ESP32 microcontroller, integrating pH, EC, and DS18B20 temperature sensors to monitor nutrient and water conditions, with a pump and relay module to automate nutrient delivery to ginger plants.

ii. An outdoor control system using an AMX-FX3U-M26MR-E PLC, connected through Modbus TCP/IP and the MQTT protocol over the UTARSF2 Wi-Fi network via a TP-Link MR105 access point and an ESP32, enabling remote monitoring and control from a web-based dashboard.

Several limitations were identified in the course of this study:

i. The systems were developed and tested specifically for ginger plants and may require modification for other crops.
ii. The ESP32 system is limited in scale and intended for indoor or small-scale environments.
iii. The outdoor PLC system depends on the availability and stability of the UTARSF2 Wi-Fi network for MQTT communication.
iv. Only selected sensors (pH, EC, and DS18B20 temperature) were used; additional parameters such as humidity or soil nutrient content were not included.
v. Long-term field testing across multiple growth cycles was beyond the project timeframe.

1.6 Contribution of the Study

This study contributes to the advancement of smart agriculture by demonstrating two distinct approaches for ginger cultivation. The first contribution is the development of a low-cost ESP32-based fertigation system. In small-scale or indoor farming applications where affordability and flexibility are important, the ESP32 provides a practical and cost-effective solution, demonstrating automation in monitoring and regulation of nutrient delivery in a controlled environment.

The second contribution is the implementation of a PLC-based outdoor control system with ESP32 IoT integration. By integrating the AMX-FX3U-M26MR-E PLC with the ESP32 using Modbus TCP/IP and the MQTT protocol, the system can be monitored in real time and remotely controlled through the UTARSF2 Wi-Fi network. This highlights the practical use of industrial automation in agricultural processes and

demonstrates how modern communication technologies can enhance the reliability and accessibility of agricultural control systems.

In addition, this study provides a practical application for ginger farming by showing how both IoT-based and industrial-grade solutions can improve efficiency and reduce manual labour. The outcomes support the broader concept of precision agriculture, offering a scalable foundation for future research and development that can be adapted to other crops and larger farming operations.

1.7 Outline of the Report

This report is organised into five main chapters. Chapter 1 presents the introduction, which includes the background of the study, importance of the research, problem statement, aim and objectives, scope and limitations, contributions, and the overall structure of the report. Chapter 2 provides a literature review, discussing previous studies and theoretical foundations relevant to the project. Chapter 3 explains the methodology and work plan. Chapter 4 presents the results and discussion, covering implementation outcomes, sensor data collected, and a comparative analysis of both systems. Chapter 5 presents the conclusions and recommendations for future work. References and appendices are provided at the end of the report.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The literature review provides an overview of past studies and research findings related to smart agriculture and fertigation systems. By examining past work, we can understand the current trends, challenges, and technological advancements in the field. This chapter reviews existing research on automated fertigation, the integration of Internet of Things (IoT) technologies in agriculture, and the use of sensors and controllers for precision farming. The advantages and limitations of different approaches are highlighted, forming the basis for developing the proposed Smart Fertigation System for ginger plants.

2.2 ESP32 Indoor Systems

2.2.1 IoT-Based Agricultural Monitoring with ESP32

In recent technology development, low-cost smart farming systems have become possible using IoT devices. Because the ESP32 microcontroller has built-in Wi-Fi and Bluetooth capabilities and is affordable and versatile, it has been widely used in smart agriculture projects. Sneineh and Shabaneh (2023) designed a hydroponics monitoring

system that automatically controls water and nutrient delivery, providing real-time monitoring and remote control through the Blynk application. The system uses an ESP32 integrated with sensors such as pH, water level, and TDS. Based on this research, it is clear that the ESP32 is a cost-effective and scalable smart agriculture solution.

2.2.2 IoT-Enabled Smart Drip Irrigation Using ESP32

Pereira, Chaari, and Daroge (2023) built an ESP32-based smart drip irrigation system, connecting multiple sensors including soil moisture, temperature, air humidity, and water flow sensors to enable real-time monitoring and automated irrigation control. The ESP32 acts as a central processing unit and connects to the Blynk IoT mobile and web dashboard for user interaction, allowing farmers to visualise environmental parameters and remotely control the irrigation process.

The logic of the ESP32 is to automatically activate the drip irrigation system when the soil moisture drops below a user-defined threshold. The water flow sensor regulates the duration and amount of irrigation, ensuring precise delivery of water to the plants. Field testing confirmed the reliability of the system, demonstrating that an ESP32-based IoT system can maintain healthy plant conditions while conserving water.

2.3 PLC and MQTT Integration

MQTT is a popular protocol for industrial IoT that enables PLCs to communicate with modern IoT devices. Chen et al. (2024) developed a web-based IoT environmental and lighting control system that bridges traditional PLC devices with an MQTT broker hosted on a Raspberry Pi 4B embedded Linux platform. This system uses Python-based programs to control data exchange between Modbus TCP and MQTT, ensuring seamless interaction between industrial controllers and IoT services.

The study adopted Node-RED as a visual programming tool to design a cross-platform Human-Machine Interface (HMI), enabling real-time monitoring and control of laboratory conditions through standard web browsers. The experimental results demonstrated that the proposed system could successfully transform existing PLC-based control setups into IoT-enabled supervisory control systems. This work highlights how MQTT and Node-RED can act as middleware to extend the functionality of PLC-controlled systems into modern IoT ecosystems — a key inspiration for the outdoor system in this project.

2.4 Comparison: IoT Microcontrollers vs. PLCs in Agriculture

Programmable Logic Controllers (PLCs) have long been the backbone of automation in agriculture and industry due to their robustness, reliability, and ability to operate in harsh environments. They are widely used to control pumps, motors, and irrigation systems because of their proven stability and industrial-grade standards. However, PLCs are often costly, less flexible, and require specialised programming knowledge, which can limit their adaptability in rapidly evolving applications such as smart farming (Saban et al., 2023).

In contrast, IoT-based microcontrollers such as the ESP32 provide low-cost, power-efficient, and easily programmable platforms that can integrate a wide variety of sensors for nutrient and environmental monitoring. These devices are highly compatible with cloud services, MQTT brokers, and web dashboards, enabling real-time data visualisation and remote control.

Saban et al. (2023) proposed a hybrid approach by integrating LoRa/LoRaWAN communication with existing PLC infrastructure, allowing legacy PLCs to communicate with IoT devices and cloud platforms. This solution leveraged the reliability of PLCs for critical farm operations while adding the flexibility of IoT for data collection, cloud storage, and remote monitoring. From the comparison, a combined IoT–PLC system offers the best outcome: PLCs ensure operational

reliability of irrigation and machinery, while IoT devices provide intelligent decision-making through data analytics and cloud-based control.

2.5 Smart Agriculture and Precision Farming

2.5.1 IoT in Agriculture: Challenges and Opportunities

Farooq and Akram (2021) conducted a Systematic Literature Review (SLR) on the application of IoT in agriculture, analysing 80 research papers published between 2006 and 2019. Their study highlights the increasing role of IoT as a transformative technology in modernising agricultural practices. By contrast, IoT enables real-time monitoring of soil temperature, humidity, moisture, atmospheric conditions, fertilization, and crop growth, improving decision-making and resource management.

One of the key contributions of this article is its emphasis on the challenges associated with IoT adoption, including environmental variability, scalability of sensor networks, and the need for reliable connectivity in rural areas. Nonetheless, the findings demonstrate that IoT has the potential to significantly improve crop productivity, sustainability, and precision farming practices.

2.5.2 Precision Farming and Its Role in Sustainable Agriculture

Finger et al. (2019) provide a comprehensive review of precision farming (PF), highlighting its role in bridging agricultural productivity and environmental sustainability. Unlike traditional large-scale mechanised farming, which treats fields uniformly, PF enables spatially and temporally tailored management practices, allowing farmers to treat fields as heterogeneous systems. This site-specific approach

reduces waste, enhances efficiency, and mitigates environmental impacts such as soil degradation and excessive agrichemical use.

The review notes that PF adoption has been most prominent in developed countries due to the high costs of technology and infrastructure requirements. However, its environmental benefits — such as reduced greenhouse gas emissions, minimised pollution, and biodiversity preservation — justify broader public and private sector incentives to encourage wider adoption.

2.6 Ginger Plant Studies

2.6.1 Effect of NPK 15:15:15 Fertilizer on Ginger Growth and Yield

Several studies have highlighted the importance of nutrient management in enhancing the productivity of ginger (Zingiber officinale). Ekwugha et al. (2021) evaluated the growth performance of ginger and turmeric under different cropping systems using NPK 15:15:15 fertilizer. The research demonstrated that higher application rates, particularly 300 g of NPK 15:15:15 per plot, significantly improved plant height, leaf number, biomass, and rhizome yield compared to lower rates. These findings provide empirical support for the suitability of NPK 15:15:15 in ginger production.

2.6.2 Water Stress, Growth, and Yield Responses of Ginger in Tropical Climates

Gatabazi et al. (2019) evaluated the growth, yield, and water use efficiency (WUE) of commercial ginger under varying water stress conditions. Results indicated that reduced water stress significantly enhanced plant height, stem number, and stomatal conductance. Moderate water stress treatments achieved higher WUE compared to

fully irrigated conditions, suggesting that controlled water management can optimise productivity while conserving resources.

In tropical climates such as Malaysia, similar challenges of water availability and excess rainfall-induced waterlogging are encountered (Ghasemzadeh et al., 2010). Since ginger is highly sensitive to waterlogging, proper irrigation scheduling guided by sensor monitoring is essential to balance water supply, maintain high yields, and ensure quality rhizome production.

2.7 Summary

This chapter reviewed the application of IoT-based microcontrollers, PLCs, and precision farming technologies in agriculture. The ESP32 microcontroller has been highlighted as a cost-effective and scalable solution for integrating pH, EC, and temperature sensors, with IoT platforms such as Blynk providing real-time monitoring and user-friendly interfaces (Sneineh & Shabaneh, 2023). PLCs, on the other hand, offer industrial-grade reliability, and when integrated with IoT protocols such as MQTT and Modbus TCP/IP, they extend functionality for secure and remote monitoring (Chen et al., 2024).

The literature further demonstrated the role of smart agriculture and precision farming in enhancing sustainability. These technologies improve resource efficiency, optimise input usage, and support environmentally friendly practices (Finger et al., 2019). Nevertheless, issues such as infrastructure, cost, and technical expertise continue to pose challenges for wider adoption, especially in smallholder farming systems (Farooq & Akram, 2021).

In terms of ginger cultivation, balanced fertilization and efficient irrigation were identified as critical factors influencing growth and yield. Studies confirmed that NPK 15:15:15 fertilizer supports better plant development and rhizome production (Ekwugha et al., 2021), while moderate irrigation regimes improve water use efficiency without compromising yield (Gatabazi et al., 2019). In tropical regions like

Malaysia, sensor-guided irrigation helps avoid drought stress and waterlogging, reinforcing the importance of adopting smart fertigation systems for sustainable ginger production (Ghasemzadeh et al., 2010).

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

The methodology of this project outlines the systematic approach taken to design and implement a Smart Fertigation System for precision agriculture, specifically focusing on ginger plants. The system integrates both IoT-based monitoring and control using the ESP32 microcontroller and industrial automation through a PLC. This hybrid approach ensures scalability, cost-effectiveness, and reliability.

For the indoor prototype, an ESP32 microcontroller was employed as the central unit due to its low cost, built-in Wi-Fi capability, and compatibility with IoT platforms such as Blynk. The ESP32 was interfaced with three sensors:

i. pH sensor — to monitor the acidity/alkalinity of the nutrient solution.
ii. Electrical Conductivity (EC) sensor — to measure nutrient concentration in the solution.
iii. DS18B20 temperature sensor — to monitor the temperature of the nutrient solution.

These sensor readings are processed by the ESP32 and sent to the Blynk IoT platform, which provides a mobile and web-based dashboard for real-time monitoring and remote control. The outdoor system employs an AMX-FX3U-M26MR-E PLC to demonstrate industrial-scale agricultural automation, communicating via Modbus TCP/IP and relayed via MQTT for remote accessibility.

3.2 ESP32 System Development Model

This project adopts the V-Model software development methodology due to its structured approach that ensures each development phase is directly linked to a corresponding testing phase. The V-Model emphasises verification and validation at every stage, which is essential for ensuring the reliability of the Smart Fertigation System.

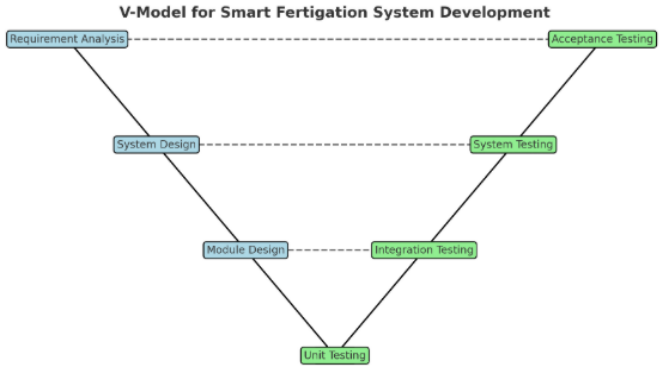

Figure 3.1: V-Model of Smart Fertigation System Development

3.2.1 Requirement Analysis

In this phase, the functional and non-functional requirements were defined: monitoring pH, EC, and temperature levels, controlling irrigation through a water pump via a relay module, and enabling remote monitoring and control via ESP32 and Blynk.

3.2.2 System Design

A high-level design was created to show the overall architecture: sensor integration with ESP32, actuator control (water pump via relay), and communication with the Blynk cloud platform and Google Sheets.

3.2.3 Module Design

Each subsystem was broken into modules:

i. Sensor module: pH, EC, and DS18B20 temperature sensors.
ii. Control module: water pump via relay module.
iii. Communication module: ESP32 with Wi-Fi connection to Blynk and Google Sheets.
iv. User interface module: Blynk mobile application for monitoring and manual control.

3.2.4 Coding / Implementation

The system was programmed on the ESP32 using the Arduino IDE. Sensor readings were acquired through analog/digital pins, while the pump was controlled using GPIO pins via a relay module. Data was transmitted to Blynk for visualisation and to Google Sheets via HTTP POST for data logging.

3.2.5 Unit Testing

Each sensor and actuator was tested individually. pH and EC sensor calibration was validated against known reference solutions, the DS18B20 temperature sensor output was verified, and the pump relay switching was tested for correct operation.

3.2.6 Integration Testing

Modules were integrated to verify data flow and functionality. Sensor readings were transmitted through the ESP32 to Blynk and Google Sheets, and actuator responses were tested based on the Blynk control switch.

3.2.7 System Testing

The entire fertigation system was tested as a whole to ensure correct operation based on nutrient solution conditions and manual override through the Blynk interface.

3.2.8 Acceptance Testing

The final system was validated against the initial requirements, ensuring it could reliably monitor nutrient solution conditions, deliver nutrients, and allow remote control.

3.3 Hardware Design ESP32

The hardware design of the Smart Fertigation System consists of sensing units, a processing unit, and actuators integrated together to automate irrigation and nutrient delivery. Figure 3.2 shows the pin connection for each sensor to the ESP32.

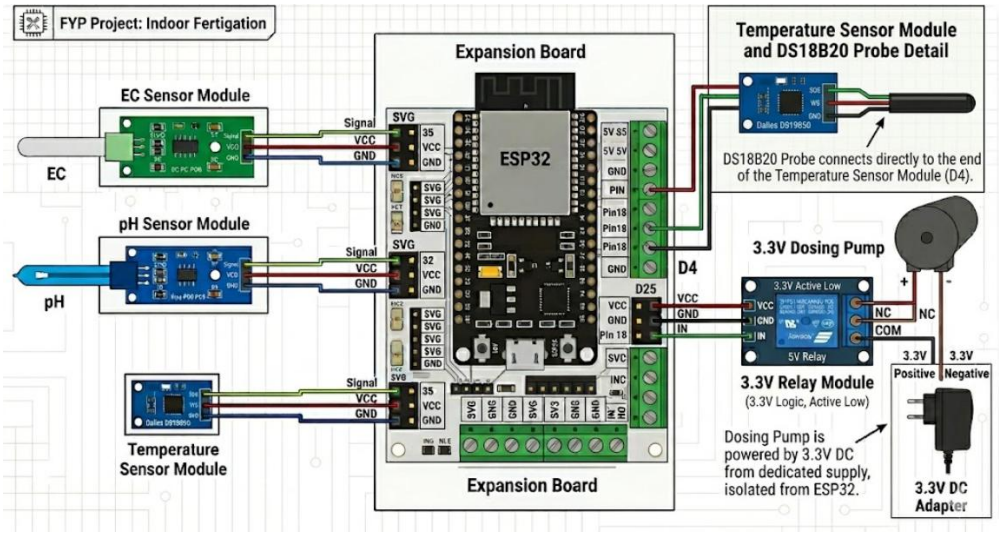

Figure 3.2: Pin Diagram Sensor to ESP32

3.3.1 Processing Unit

The ESP32 microcontroller is the core of the system, selected due to its dual-core processor, built-in Wi-Fi, low power consumption, and cost-effectiveness. It handles all sensor readings, decision-making, and communication with Blynk and Google Sheets.

3.3.2 Sensors

The DS18B20 temperature sensor measures the temperature of the nutrient solution using a digital one-wire interface, connected to GPIO pin 4. The pH sensor (connected to GPIO 32) monitors the acidity or alkalinity of the nutrient solution. The EC sensor (connected to GPIO 35) measures the electrical conductivity of the solution, which correlates with the nutrient concentration.

3.3.3 Actuators

The water pump is controlled via a relay module connected to GPIO pin 25 of the ESP32. The relay module jumper is set to 3.3V to match the ESP32 GPIO output logic level, as the module uses active-low logic where a LOW signal activates the relay and turns the pump ON.

3.3.4 IoT Integration

The ESP32 communicates with the Blynk platform via Wi-Fi, enabling remote monitoring and control. Sensor values are sent to Blynk virtual pins V1 (pH), V2 (EC), and V3 (temperature) every 2 seconds. The pump is controlled via virtual pin V4 from

the Blynk dashboard. All readings are also logged to Google Sheets via HTTP POST to a Google Apps Script endpoint.

3.4 PLC to MQTT Integration Methodology

3.4.1 MQTT Configuration

The MQTT broker and topics were configured to enable communication between the ESP32 and the web-based dashboard. Essential parameters such as broker address (broker.emqx.io), port number, username, and password were defined. Individual MQTT topics were created for each sensor reading and each actuator command.

3.4.2 Wi-Fi Connection Setup

The PLC was connected to the UTARSF2 Wi-Fi network through a TP-Link MR105 access point via its built-in Ethernet port. This allows the ESP32 to access both the PLC (via Modbus TCP/IP) and the MQTT broker (via Wi-Fi) through the same network.

3.4.3 C/C++ Firmware on ESP32

The ESP32 firmware, written in C/C++ using the Arduino IDE, uses the PubSubClient library for MQTT operations and a Modbus TCP client library for communication with the PLC. Publish and subscribe operations were established to send and receive messages over the UTARSF2 Wi-Fi network.

3.4.4 Command Transmission via Modbus TCP/IP

The ESP32 initiates Modbus TCP/IP read requests to the PLC's IP address at regular intervals, polling the specific registers that hold each sensor reading. Once values are retrieved, the ESP32 publishes them to the corresponding MQTT topics. For actuator control, when an operator presses a control button on the dashboard, a command is published to the corresponding MQTT topic. The ESP32 receives the message and translates it into a Modbus TCP/IP write operation, writing to the corresponding Modbus coil address in the PLC to activate or deactivate the target motor or valve.

3.4.5 PLC Response Handling

The PLC was programmed to respond to changes in its Modbus coil and register values. When the ESP32 writes to a specific coil address via Modbus TCP/IP, the PLC executes the corresponding action in its ladder logic — such as switching a motor or valve ON or OFF based on the updated coil state.

3.4.6 System Testing and Debugging

The system was tested by sending multiple MQTT commands through the ESP32 firmware and observing the PLC's response. Debugging was performed to resolve issues in message delivery, Wi-Fi connectivity, and PLC execution reliability.

3.5 Work Plan

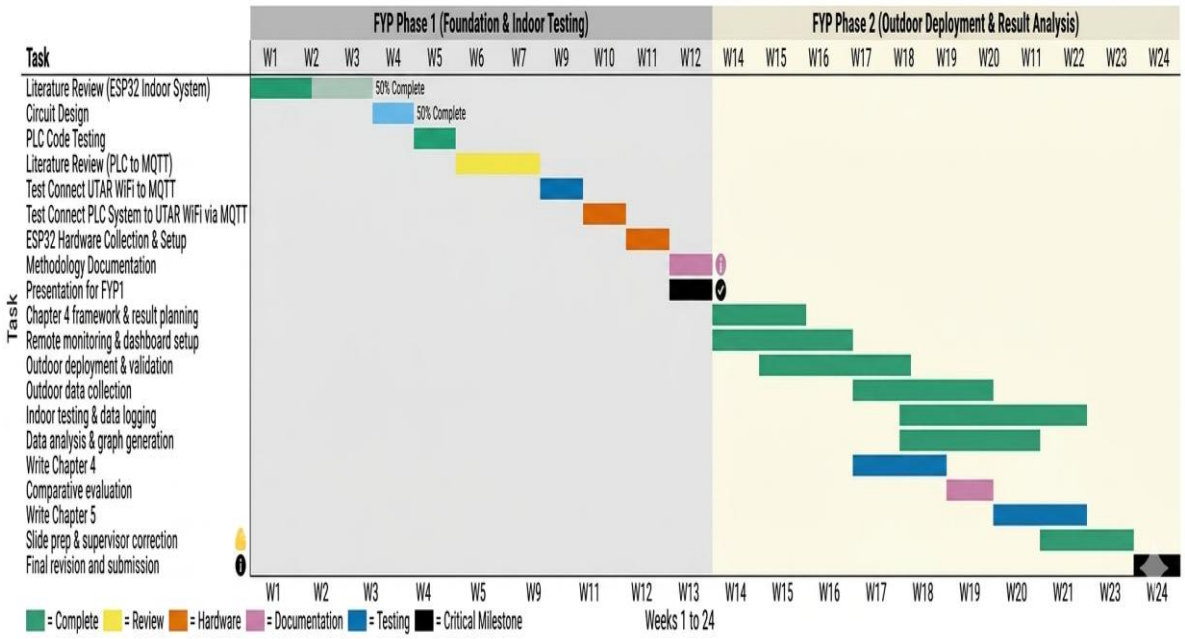

Figure 3.3: Gantt Chart of the Project

3.6 Summary

The methodology of this project combines IoT-based control using ESP32 and industrial automation through a PLC, ensuring a hybrid approach that balances scalability, reliability, and cost-effectiveness. For the ESP32 system, the V-Model software development process was applied, covering requirement analysis, system design, coding, and thorough testing phases. The ESP32 was integrated with pH, EC, and DS18B20 temperature sensors to monitor nutrient solution conditions, while the pump was controlled manually via the Blynk IoT platform.

For the PLC-based outdoor system, the methodology focused on integrating Modbus TCP/IP communication with the MQTT protocol over the UTARSF2 Wi-Fi network via the TP-Link MR105. The ESP32 reads sensor data from the PLC via Modbus TCP/IP and publishes it to the MQTT broker, while subscribing to command topics and writing Modbus coil values back to the PLC for actuator control. System

testing and debugging were carried out to ensure stable communication and reliable execution of commands.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the results obtained from the implementation of the Smart Fertigation System for Precision Agriculture. The system comprises two distinct sub-systems: an outdoor PLC-based fertigation system deployed at an agricultural park, and an indoor ESP32-based prototype developed in a controlled indoor environment for ginger plant cultivation. The discussion covers the physical implementation of both systems, the process of transitioning the outdoor system from a standalone manual operation to a fully automated and IoT-enabled platform, the remote monitoring infrastructure established, and the sensor data collected during the observation period. A comparative analysis between the indoor and outdoor systems is also presented, followed by a conclusion on the suitability of each approach for ginger cultivation.

4.2 System Overview and Physical Implementation

4.2.1 Outdoor PLC-Based System

The outdoor fertigation system was deployed at an agricultural park and serves as the primary implementation of this project. The system was originally operated as a

standalone, manually-controlled setup, where operators were required to be physically present to trigger irrigation cycles, monitor tank levels, and adjust fertigation schedules. This approach was labour-intensive and prone to human error, particularly in managing fertilizer concentrations and irrigation timing.

The hardware centre of the outdoor system is an AMX-FX3U-M26MR-E Programmable Logic Controller (PLC), an industrial-grade controller that handles all control logic for the motors, solenoid valves, and sensor monitoring at the agricultural park. The AMX-FX3U-M26MR-E is equipped with a built-in Ethernet port, which was connected to a TP-Link MR105 access point. This access point bridges the PLC into the UTARSF2 Wi-Fi network, enabling network-level communication between the PLC and other networked devices on site.

To convert the PLC system into a fully IoT-enabled platform, an ESP32 microcontroller was integrated into the same Wi-Fi network via the TP-Link MR105. The ESP32 communicates with the PLC over Modbus TCP/IP, reading sensor registers and writing coil addresses for actuator control. All data is then relayed via MQTT to the web-based dashboard. This combination of industrial PLC hardware with an IoT-capable ESP32 created a hybrid system that retained the reliability and deterministic ladder logic control of the PLC while gaining the flexibility and remote accessibility of an IoT platform.

The complete list of actuators and sensors monitored and controlled through the outdoor system is as follows. On the actuator side, the system controls four stirring motors (Stirrer 1–4) that agitate the nutrient solution in each fertilizer tank, four fertilizer dosing motors (Dosing Motor 1–4) that inject fertilizer concentrates into the irrigation lines, and one main tank pump. In addition, the system controls the main tank inlet valve and the fertilizer tank inlet valves, which regulate the flow of water pumped from the lake source into the main tank and each of the four fertilizer tanks respectively. These valves allow the system to automatically refill the tanks when water levels drop below the required threshold, without requiring manual intervention. On the sensor side, the system continuously monitors EC, pH, and temperature of the nutrient solution, as well as individual water levels for five tanks: the main tank and

Fertilizer Tanks 1 through 4. All of these readings and controls are accessible remotely through the web-based HTML dashboard via MQTT.

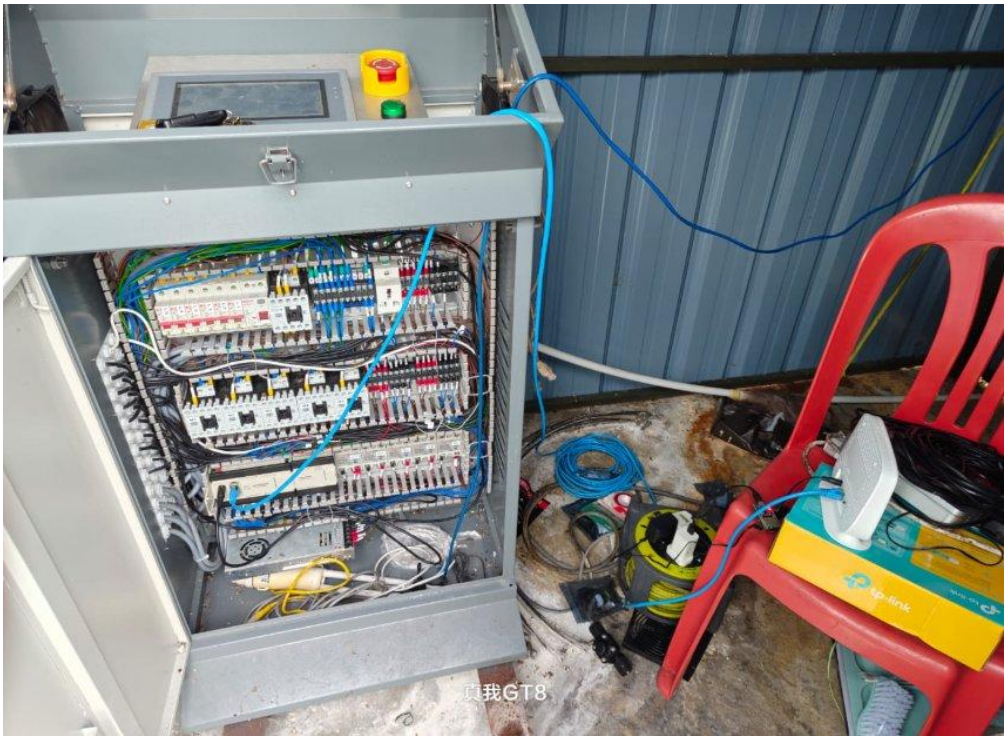

Figure 4.1: Outdoor PLC control panel with AMX-FX3U-M26MR-E PLC and TP-Link MR105 access point at the agricultural park

4.2.2 Indoor ESP32-Based Prototype

The indoor system was designed as a complementary prototype to validate the concept of IoT-based fertigation control in a controlled indoor environment. Unlike the outdoor system which required industrial-grade PLC hardware, the indoor prototype was built around an ESP32 microcontroller, significantly reducing system cost and complexity.

The indoor system integrates a pH sensor, an EC sensor, and a DS18B20 temperature sensor. A water pump controlled via a relay module provides the actuation mechanism for nutrient delivery. The ESP32 reads all sensor values at regular intervals and transmits the data to the Blynk IoT cloud platform for real-time monitoring and

remote pump control. The system also logs all sensor readings to Google Sheets via HTTP POST requests to a Google Apps Script endpoint.

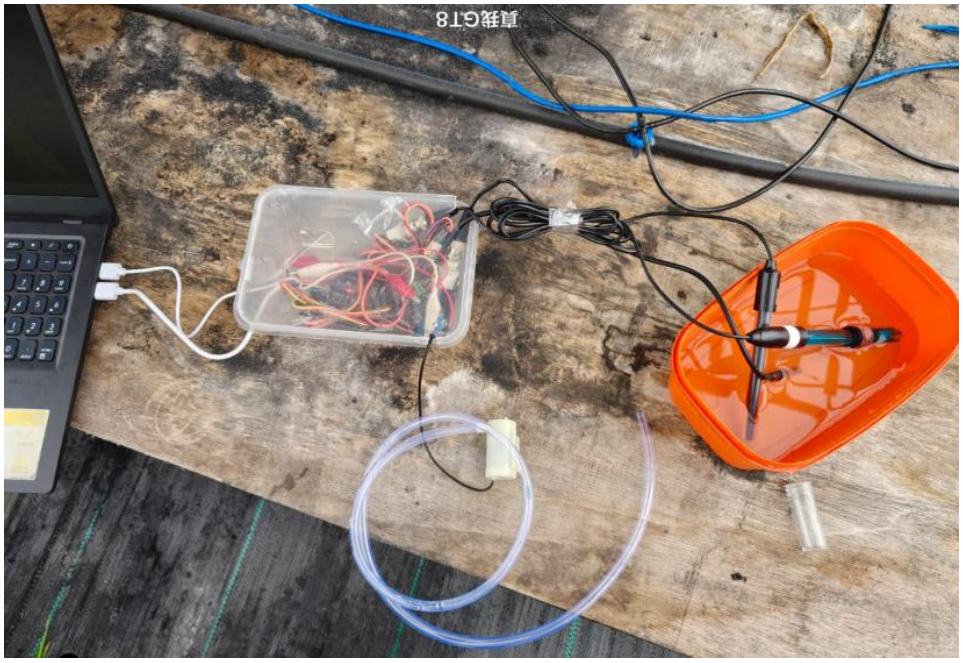

Figure 4.2: Indoor ESP32-based fertigation prototype

4.3 Transition from Manual Standalone System to Automated IoT Platform

4.3.1 Original Manual Operation

Prior to this project, the outdoor fertigation system at the agricultural park operated as a traditional standalone system. The PLC was programmed with fixed irrigation schedules and required on-site manual intervention for any changes. Operators had to physically visit the site to check tank water levels, verify pump operation, read sensor values from local displays, and manually activate or deactivate actuators when required. There was no mechanism for remote monitoring, and no sensor data was recorded for historical analysis. This manual approach introduced significant

operational inefficiencies and made it difficult to respond promptly to abnormal conditions such as low water levels or out-of-range pH values.

4.3.2 Integration of ESP32 with the PLC for IoT Capability

The core contribution of this project's outdoor implementation was the seamless integration of an ESP32 microcontroller with the existing AMX-FX3U-M26MR-E PLC infrastructure to convert the standalone system into a fully IoT-enabled platform. The integration followed a layered communication architecture, as illustrated in Figure 4.3.

Figure 4.3: Communication flowchart — AMX-FX3U-M26MR-E PLC (Ethernet) → TP-Link MR105 → ESP32 (Wi-Fi) → MQTT Broker → Web Dashboard

The AMX-FX3U-M26MR-E PLC communicates over its built-in Ethernet port using the Modbus TCP/IP protocol. Sensor values such as EC, pH, temperature, water level, and flow rate are stored in designated Modbus registers within the PLC memory. The TP-Link MR105 access point, connected to the PLC via Ethernet, bridges the PLC

into the UTARSF2 Wi-Fi network, making these Modbus registers accessible to any device on the same network.

The ESP32 was programmed using the Arduino IDE with the PubSubClient MQTT library. It connected to the UTARSF2 Wi-Fi network through the TP-Link MR105 access point. At regular intervals, the ESP32 initiates a Modbus TCP/IP request to the PLC's IP address, polling the specific registers that hold each sensor reading. Once the values are retrieved, the ESP32 immediately publishes them to the corresponding MQTT topics on the broker for display on the web dashboard. MQTT topics were defined for each sensor parameter and each actuator command, including individual topics for the four stirring motors, four dosing motors, five inlet valves, the main tank pump, and all five tank water level sensors.

For actuator control, the process works in reverse. When an operator presses a control button on the web dashboard, a command is published to the corresponding MQTT topic. The ESP32 receives the message and translates it into a Modbus TCP/IP write operation directed at the PLC. Specifically, the ESP32 writes to the corresponding Modbus coil address in the PLC, which directly sets the output state of the target motor or valve.

This architecture effectively decoupled the control logic (handled by the PLC) from the communication and remote access layer (handled by the ESP32 and MQTT), ensuring that the industrial reliability of the PLC was preserved while adding the flexibility of remote IoT control. The PLC continued to execute its ladder logic autonomously for safety-critical operations, while the ESP32 layer provided the remote override and monitoring capability.

4.3.3 Benefits Gained from the Conversion

The transition from a manual standalone system to an automated IoT platform delivered several measurable operational benefits:

i. Remote Monitoring: All sensor readings including water level, pH, temperature, EC, and flow rate can now be viewed in real time from any location with internet access through the web-based dashboard.
ii. Remote Actuation: Operators can now send ON/OFF commands to the PLC-controlled pumps, motors, and valves directly from a mobile phone or computer without being physically present at the site.
iii. Automated Data Logging: Sensor data is continuously recorded with timestamps, enabling historical analysis, trend observation, and early detection of anomalies.
iv. Reduced Labour Requirement: The need for routine on-site inspections was significantly reduced, as the system proactively provides all operational data remotely.

4.4 Remote Monitoring and Control Infrastructure

4.4.1 Outdoor System — MQTT and Web Dashboard

The remote monitoring infrastructure for the outdoor system was established using the MQTT protocol over the UTARSF2 Wi-Fi network. An MQTT broker (broker.emqx.io) was used as the message relay between the ESP32 at the agricultural park and the web-based control dashboard. The dashboard, developed in HTML and JavaScript, connects to the broker via WebSocket and displays live sensor readings including water level for all five tanks, pH, temperature, EC, and flow rate. The dashboard also provides dedicated actuator control panels for all actuators: Stirrer 1–4, Dosing Motor 1–4, inlet valves, and the main tank pump. Each actuator has individual ON/OFF buttons that transmit MQTT commands to the ESP32, which forwards them to the AMX-FX3U-M26MR-E PLC for execution. The dashboard is mobile-responsive, allowing operators to monitor and control the entire outdoor fertigation system from a smartphone.

4.4.2 Indoor System — Blynk and Google Sheets

For the indoor prototype, remote monitoring was implemented using the Blynk IoT platform. The ESP32 transmits sensor readings to Blynk virtual pins every two seconds, enabling near real-time display of pH, EC, and temperature on both the Blynk mobile application and the Blynk web console. The pump relay is exposed through a Blynk button widget mapped to virtual pin V4, allowing direct remote control of the pump from any device running the Blynk app.

In addition to Blynk, the indoor system logs all sensor data to a Google Sheets spreadsheet via HTTP POST requests directed to a deployed Google Apps Script web application. This provides a permanent, timestamped record of all sensor readings that can be exported and analysed at any time.

4.5 Sensor Data Collection Results

4.5.1 Outdoor System Data

Continuous sensor data was collected from the outdoor PLC-based system from 27 March 2026 to 17 April 2026, spanning a total of approximately 22 days. A total of 15,272 data records were logged during this period. No significant system downtime or data transmission failure was recorded throughout the monitoring period, demonstrating the stability and reliability of the MQTT-based communication architecture.

Table 4.1: Summary of Outdoor System Sensor Data (27 March 2026 – 17 April 2026)

Parameter	Minimum	Maximum	Average
Temperature (°C)	28.0	45.7	34.6
pH	7.09	7.76	7.29
EC (µS/cm)	6	2987	870

The outdoor temperature readings ranged from 28.0°C to 45.7°C with an average of 34.6°C, consistent with Malaysian tropical outdoor conditions. The elevated maximum temperature of 45.7°C was recorded during peak afternoon hours, reflecting direct sun exposure at the agricultural park. The pH values remained relatively stable, ranging from 7.09 to 7.76 with an average of 7.29, indicating a near-neutral to slightly alkaline growing medium within the acceptable range for ginger cultivation. EC values exhibited a wider variation from 6 to 2987 µS/cm, reflecting the dynamic nature of the outdoor fertigation cycle where nutrient concentration fluctuates as irrigation events occur and the system replenishes the nutrient tanks.

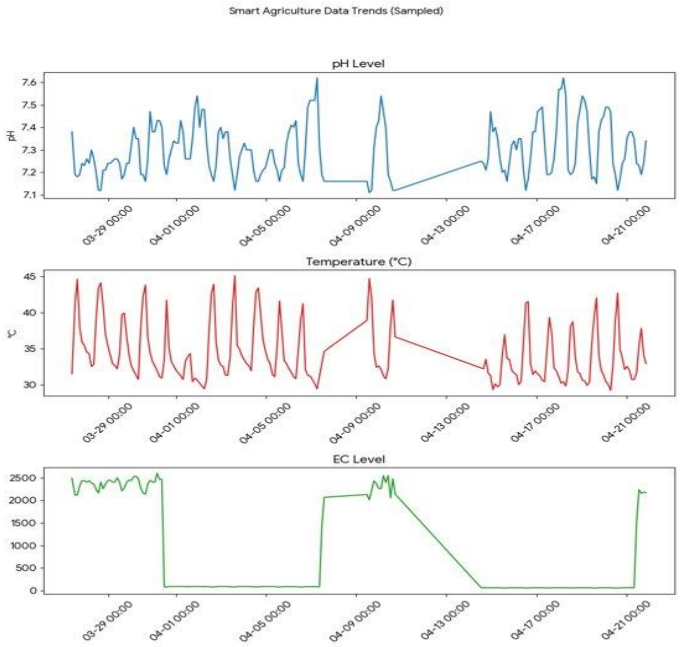

Figure 4.4: Outdoor smart agriculture sensor trends — pH, Temperature, and EC (27 March 2026 – 17 April 2026)

4.5.2 Indoor System Data

Sensor data collection for the indoor ESP32-based system was conducted continuously from 19 April 2026 to 21 April 2026, spanning approximately 2 days and 20 hours. A total of 170 valid data readings were recorded during this period, with readings logged approximately every 4 to 5 seconds via the Google Sheets logging integration.

Table 4.2: Summary of Indoor System Sensor Data (19 April 2026 – 21 April 2026)

Parameter	Minimum	Maximum	Average
Temperature (°C)	28.31	29.31	28.93
pH	3.90	7.09	6.01
EC (mS/cm)	0.00	0.98	0.16

The indoor temperature exhibited a gradual upward trend, rising from approximately 28.44°C on 19 April 2026 to a peak of 29.31°C by 21 April 2026. This increase of less than 1°C over nearly three days is attributed to natural ambient temperature variation within the laboratory room. The pH readings showed a general declining trend from approximately 6.50 on 19 April 2026, gradually decreasing to around 6.20 by 20 April 2026, before exhibiting wider fluctuations on 21 April 2026 with values ranging from 3.90 to 7.09. The wider pH spread on 21 April 2026 is likely attributable to nutrient depletion as the system operated without a solution refresh. The EC readings declined steadily from 0.89 mS/cm on 19 April 2026 to near zero by 20 April 2026, indicating progressive nutrient depletion over time.

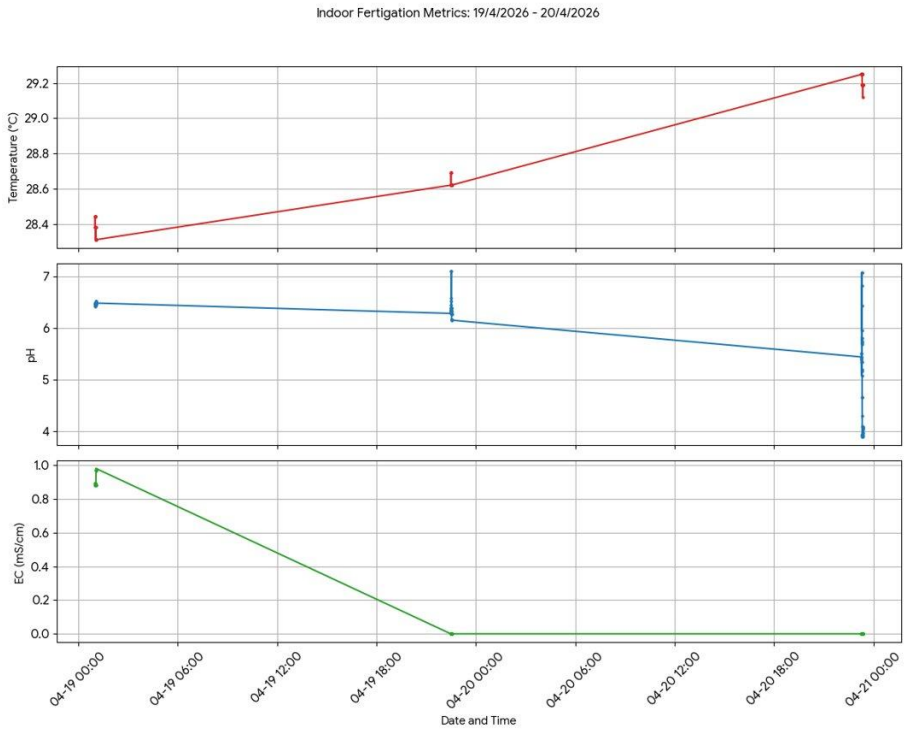

Figure 4.5: Indoor fertigation sensor trends — Temperature, pH, and EC (19 April 2026 – 21 April 2026)

4.6 Comparative Analysis: Outdoor vs Indoor System

Table 4.3 presents a structured comparison between the outdoor PLC-based system and the indoor ESP32-based prototype across key performance and implementation dimensions.

Table 4.3: Comparative Analysis of Outdoor and Indoor Fertigation Systems

Aspect	Outdoor (PLC-Based)	Indoor (ESP32-Based)
Hardware Platform	AMX-FX3U-M26MR-E PLC + ESP32 (IoT bridge)	ESP32 microcontroller
Communication Protocol	Modbus TCP/IP + MQTT over Wi-Fi	Blynk cloud + HTTP (Google Sheets)
Remote Control	Web dashboard (MQTT WebSocket)	Blynk mobile app
Sensors Monitored	EC, pH, Temperature, Water level (5 tanks), Flow rate	pH, EC, Temperature
Actuators Controlled	4 Stirrers, 4 Dosing motors, 5 Inlet valves, Main pump	1 Water pump (relay)
Data Logged	15,272 records over 22 days	170 records over ~3 days
Temperature Range	28.0°C – 45.7°C (avg 34.6°C)	28.31°C – 29.31°C (avg 28.93°C)
pH Range	7.09 – 7.76 (avg 7.29)	3.90 – 7.09 (avg 6.01)
System Cost	Higher (industrial PLC)	Lower (microcontroller ~RM25)
Complexity	High (PLC + IoT bridge)	Moderate (Arduino IDE + Blynk)
Environment	Outdoor, variable, sun-exposed	Indoor, controlled

The outdoor system demonstrated superior scalability and robustness for real-world agricultural deployment. The PLC provided deterministic and fail-safe control logic that continued to operate independently even during temporary MQTT connectivity losses. The integration of the ESP32 as an IoT gateway layer allowed remote monitoring and control to be added without replacing or modifying the existing PLC infrastructure, making it a cost-effective upgrade path.

The indoor system, while more limited in scale, demonstrated that the fundamental IoT monitoring and control concept could be implemented at significantly lower cost using a single ESP32 microcontroller. The simplicity of the Blynk platform reduced development time and allowed rapid prototyping. However, the indoor system is constrained by its dependence on the Blynk cloud service and lacks the industrial resilience of the PLC-based outdoor system.

4.7 Discussion and Conclusion on System Suitability for Ginger Cultivation

Based on the results and comparative analysis, both the outdoor and indoor systems successfully fulfilled their respective roles within the Smart Fertigation System for Precision Agriculture project. However, for the specific cultivation of ginger plants (Zingiber officinale), the outdoor system is clearly the more suitable long-term deployment environment for the following agronomic and technical reasons.

Ginger is a tropical plant that requires abundant sunlight for photosynthesis to drive the metabolic processes that contribute to rhizome development and biomass accumulation. While ginger is often described as shade-tolerant at the seedling stage, full productive growth and commercially viable rhizome yield require exposure to natural solar radiation. Indoor growing environments, even with supplemental lighting, cannot fully replicate the spectral composition and intensity of natural sunlight, which limits the photosynthetic rate and ultimately constrains plant productivity.

The outdoor temperature data recorded in this project averaged 34.6°C with peak values of 45.7°C, which aligns closely with the optimal growing temperature range of 25°C to 35°C for ginger as reported in the literature. The tropical Malaysian climate therefore naturally provides the thermal conditions that ginger requires without the need for active climate control.

In conclusion, the outdoor PLC-based IoT fertigation system represents a significant advancement over the original manual standalone operation. The integration of ESP32 as a Modbus TCP/IP and MQTT bridge successfully converted a traditional industrial PLC system into a remotely accessible, data-driven IoT platform without compromising the reliability of the core control logic. The indoor ESP32 prototype successfully demonstrated the feasibility of low-cost IoT fertigation monitoring and serves as a valuable proof-of-concept for future development.

4.8 Limitations of the System

4.8.1 Outdoor System Limitations

The AMX-FX3U-M26MR-E PLC is equipped with only one RJ45 Ethernet port supporting Modbus TCP, alongside one RS485 port supporting Modbus RTU. These two communication interfaces share the same PLC CPU scan cycle, meaning the PLC processes communication requests sequentially rather than in parallel. When the HMI panel is operated rapidly within a short time interval — triggering frequent RS485 communication requests — the PLC's CPU becomes occupied processing HMI transactions, causing the ESP32's concurrent Modbus TCP read and write requests over Ethernet to time out or return errors. This limitation was observed during system testing and is a constraint of the PLC hardware architecture rather than a software bug. A potential mitigation is to implement retry logic in the ESP32 firmware, or to reduce the HMI polling rate to minimise contention.

4.8.2 Indoor System Limitations

Several limitations were identified in the indoor ESP32-based prototype. First, the pH and EC sensors were not fully calibrated against laboratory-grade reference solutions. As a result, the sensor readings, while consistent in trend, may carry absolute measurement errors that reduce their precision for making exact fertigation decisions.

Second, the indoor system relies entirely on the Blynk cloud platform for remote monitoring and control. If the Blynk server experiences downtime or the free-tier usage limit is reached, the remote monitoring and pump control functionality would be unavailable. Unlike the outdoor PLC system where control logic continues to execute autonomously during network disruption, the indoor ESP32 system has no offline fallback control mechanism.

Third, the nutrient solution in the indoor prototype was not automatically replenished during the monitoring period. As observed in the sensor data, the EC value declined steadily from 0.89 mS/cm to near zero over approximately 24 hours, indicating nutrient depletion. Without an automated dosing or refill mechanism, the system cannot maintain consistent nutrient concentration over extended operation periods.

Fourth, the indoor growing environment lacks natural sunlight and relies only on ambient room lighting, which is insufficient to sustain the full photosynthetic requirements of ginger plants. This constraint limits the indoor prototype to short-term monitoring and proof-of-concept demonstrations rather than full crop production.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The development of the Smart Fertigation System for Precision Agriculture has successfully demonstrated the viability of integrating IoT-enabled microcontrollers and industrial-grade PLCs to support ginger cultivation. This project deployed a dual-system architecture: a PLC-based outdoor system for robust, field-scale control, and an ESP32-based indoor prototype for real-time monitoring and proof-of-concept validation. Together, they establish a functional foundation for smart agricultural management.

The system objectives were met regarding the capability to monitor key parameters including pH, EC, and temperature. The outdoor system facilitates PLC-based control via Modbus TCP/IP, while the indoor prototype utilises the Blynk platform to provide real-time data visualisation and manual pump actuation for remote irrigation control. This hybrid approach bridges the gap between low-cost prototyping and industrial-grade reliability.

Despite technical challenges regarding communication protocol contention and sensor calibration, this project serves as a comprehensive proof-of-concept. It confirms that IoT-based systems can significantly reduce manual labour, provide an accessible interface for managing critical agricultural variables, and support sustainable farming practices in tropical climates like Malaysia.

From a broader perspective, this project contributes meaningfully to the United Nations Sustainable Development Goals (SDGs). The Smart Fertigation System supports SDG 2 (Zero Hunger) by improving the precision and efficiency of food crop production, enabling more consistent ginger yields through automated nutrient and water management. It aligns with SDG 9 (Industry, Innovation and Infrastructure) by demonstrating how IoT-based technologies and industrial automation can be applied to modernise agricultural infrastructure, even within a university research context. Furthermore, the system's ability to reduce fertilizer overconsumption and prevent unnecessary water usage through sensor-guided control directly supports SDG 12 (Responsible Consumption and Production). These contributions reinforce the relevance of smart agriculture not only as an engineering achievement, but as a step towards more sustainable and responsible farming practices in Malaysia and the broader tropical agricultural context.

5.2 Recommendations for Future Work

5.2.1 Transition to Fully Automated Fertigation

The current indoor system relies on manual pump activation via the Blynk app. Future work should integrate peristaltic pumps controlled by the ESP32 to transition to fully automated nutrient dosing. A closed-loop PID control algorithm could be implemented to automatically adjust pH and EC levels to specific setpoints, maintaining optimal nutrient concentrations without constant user oversight.

5.2.2 Expansion of Offline Control and Edge Computing

The indoor system currently depends on the Blynk cloud platform for remote control. Future iterations should incorporate a local edge computing gateway such as a

Raspberry Pi, mitigating risks associated with third-party server downtime and allowing the system to execute irrigation schedules even if the external network is disrupted.

5.2.3 Hardware Upgrade for Parallel Communication

As identified in the system limitations, the AMX-FX3U-M26MR-E PLC is restricted in handling concurrent TCP and RS485 communication requests. Future deployments should upgrade to a PLC model that supports multi-client TCP/IP connections, or implement a dedicated communication gateway, allowing the PLC to interface with multiple monitoring and control devices simultaneously without command timeouts.

5.2.4 Long-Term Sensor Calibration and Durability

The sensors used in this project require periodic calibration to maintain accuracy. Future research should explore the integration of self-calibrating industrial sensors or software-based drift-compensation algorithms to ensure consistency over long durations. Enclosure designs should also be improved for field deployment to protect against high humidity and direct solar radiation.

5.2.5 Scalability and Wireless Mesh Networking

To scale the system for larger agricultural plots, the current communication method should be upgraded to a wireless mesh network such as ESP-NOW or LoRaWAN. This would allow multiple distributed sensor nodes to relay data back to a central controller, significantly extending the system’s coverage area while reducing wiring complexity across the farming site.

REFERENCES

Akham, H., Singh, N.I., & Sharma, R.K. (2010). Effect of integrated nutrient management on growth, yield and economics of ginger (Zingiber officinale Rosc.). Crop Research, 40(1/3), 152–156.

Chen, C-Y., Wu, S-H., Huang, B-W., Huang, C-H., & Yang, C-F. (2024). Web-based Internet of Things on environmental and lighting control and monitoring system using Node-RED, MQTT and Modbus communications within embedded Linux platform. Internet of Things, 25, 101305. https://doi.org/10.1016/j.iot.2024.101305

Ekwugha, U.E., Moses, O., Anyaegbu, P.O., & Jennifer, O.C. (2021). Growth performance of ginger and turmeric as influenced by cropping system and NPK fertilizer 15:15:15. International Journal of Research and Scientific Innovation (IJRSI), 8(2), 30–43.

Farooq, M.S., & Akram, S. (2021). IoT in agriculture: Challenges and opportunities. Journal of Agricultural Research, 59(1), 63–76.

Finger, R., Swinton, S.M., El Benni, N., & Walter, A. (2019). Precision farming at the nexus of agricultural production and the environment. Annual Review of Resource Economics, 11(1), 313–335. https://doi.org/10.1146/annurev-resource-100518-093929

Gatabazi, A., et al. (2019). Irrigation regimes for sustainable ginger cultivation: Water use efficiency and yield response. Journal of Agricultural Science and Technology, 21(3), 55–67.

Ghasemzadeh, A., Jaafar, H.Z., & Rahmat, A. (2010). Effects of soil moisture and irrigation on the growth and rhizome production of ginger in tropical climates. Journal of Agricultural Science, 2(3), 125–132.

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

Saban, M., Bekkour, M., Amdaouch, I., El Gueri, J., Ait Ahmed, B., Chaari, M.Z., Ruiz-Alzola, J., Rosado-Muñoz, A., & Aghzout, O. (2023). A smart agricultural system based on PLC and a cloud computing web application using LoRa and LoRaWan. Sensors, 23(5), 2725. https://doi.org/10.3390/s23052725

Sneineh, M.A., & Shabaneh, A.A.A. (2023). Design of a smart hydroponics monitoring system using an ESP32 microcontroller and the Internet of Things. MethodsX, 11, 102401. https://doi.org/10.1016/j.mex.2023.102401

APPENDICES

Appendix A: ESP32 Source Code

Library .cpp

```
#include "SimpleModbusTCP.h"

SimpleModbusTCP::SimpleModbusTCP(const char* ip, uint16_t port)
{
   _serverIP.fromString(ip);
   _port = port;
   _transactionID = 0;
}

bool SimpleModbusTCP::connect()
 {
   return _client.connect(_serverIP, _port);
}

void SimpleModbusTCP::disconnect()
 {
   _client.stop();
}

bool SimpleModbusTCP::isConnected()
{
   return _client.connected();
```

```
}

uint16_t SimpleModbusTCP::_readUInt16(uint8_t* data, uint8_t index)
 {
   return (data[index] << 8) | data[index + 1];
}

void SimpleModbusTCP::_sendRequest(uint8_t* request, uint8_t length)
{
   request[0] = _transactionID >> 8;
   request[1] = _transactionID & 0xFF;
   _transactionID++;
   while (_client.available()) { _client.read(); }
   _client.write(request, length);
}

bool SimpleModbusTCP::_waitResponse(uint8_t* response, uint8_t
expectedBytes, uint16_t timeout)
{
   unsigned long start = millis();
   while (_client.available() < expectedBytes) {
      if (millis() - start > timeout) {
         Serial.println("[Modbus] Timeout waiting for response");
         // NOTE: Do NOT call _client.stop() here
         // FX3U-ENET-ADP only supports 1 TCP connection
         // Closing it means waiting 30-60 sec for PLC to release the slot
         return false;
      }
      delay(5);
   }
   _client.read(response, expectedBytes);
   return true;
}
```

```
uint16_t SimpleModbusTCP::readHoldingRegister(uint16_t address)
{
   uint8_t request[12] = {
      0, 0, 0x00, 0x00, 0x00, 0x06,
      0x01, 0x03,
      (uint8_t)(address >> 8), (uint8_t)(address & 0xFF),
      0x00, 0x01
   };
   _sendRequest(request, 12);
   delay(20);
   uint8_t response[11];
   if (_waitResponse(response, 11)) {
      if (response[7] == 0x03 && response[8] == 2) {
         return ((uint16_t)response[9] << 8) | response[10];
      }
   }
   return 0;
}

bool SimpleModbusTCP::writeHoldingRegister(uint16_t address, uint16_t value)
{
   uint8_t request[12] = {
      0, 0, 0x00, 0x00, 0x00, 0x06,
      0x01, 0x06,
      (uint8_t)(address >> 8), (uint8_t)(address & 0xFF),
      (uint8_t)(value >> 8), (uint8_t)(value & 0xFF)
   };
   _sendRequest(request, 12);
   delay(20);
   uint8_t response[12];
   if (_waitResponse(response, 12))
   {
```

```
        return (response[7] == 0x06);
    }
    return false;
}

bool SimpleModbusTCP::readMultipleRegisters(uint16_t startAddress, uint16_t
quantity, uint16_t* buffer)
{
    if (quantity > 125) return false;
    uint8_t request[12] = {
        0, 0, 0x00, 0x00, 0x00, 0x06,
        0x01, 0x03,
        (uint8_t)(startAddress >> 8), (uint8_t)(startAddress & 0xFF),
        (uint8_t)(quantity >> 8), (uint8_t)(quantity & 0xFF)
    };
    _sendRequest(request, 12);
    delay(20);
    uint8_t byteCount = quantity * 2;
    uint8_t totalBytes = 9 + byteCount;
    uint8_t response[totalBytes];
    if (_waitResponse(response, totalBytes))
    {
        if (response[7] == 0x03 && response[8] == byteCount)
{
            for (int i = 0; i < quantity; i++) {
                buffer[i] = ((uint16_t)response[9 + i * 2] << 8) | response[10 + i * 2];
            }
            return true;
        }
    }
    return false;
}
```

```
bool SimpleModbusTCP::readCoil(uint16_t address)
{
   uint8_t coilBuf[1];
   if (readMultipleCoils(address, 1, coilBuf))
   {
      return coilBuf[0] & 0x01;
   }
   return false;
}

bool SimpleModbusTCP::readMultipleCoils(uint16_t startAddress, uint16_t
quantity, uint8_t* buffer)
{
   if (quantity > 2000) return false;
   uint8_t request[12] = {
      0, 0, 0x00, 0x00, 0x00, 0x06,
      0x01, 0x01,
      (uint8_t)(startAddress >> 8), (uint8_t)(startAddress & 0xFF),
      (uint8_t)(quantity >> 8),     (uint8_t)(quantity & 0xFF)
   };
   _sendRequest(request, 12);
   delay(20);
   uint8_t byteCount = (quantity + 7) / 8;
   uint8_t totalBytes = 9 + byteCount;
   uint8_t response[totalBytes];
   if (_waitResponse(response, totalBytes))
   {
      if (response[7] == 0x01 && response[8] == byteCount)
{
         for (int i = 0; i < byteCount; i++)
{
            buffer[i] = response[9 + i];
         }
```

```
            return true;
        }
    }
    return false;
}

bool SimpleModbusTCP::writeCoil(uint16_t address, bool value)
{
    uint8_t request[12] = {
        0, 0, 0x00, 0x00, 0x00, 0x06,
        0x01, 0x05,
        (uint8_t)(address >> 8), (uint8_t)(address & 0xFF),
        value ? 0xFF : 0x00, 0x00
    };
    _sendRequest(request, 12);
    delay(20);
    uint8_t response[12];
    if (_waitResponse(response, 12)) {
        return (response[7] == 0x05);
    }
    return false;
}
```

Library .h

```
#ifndef SIMPLE_MODBUS_TCP_H
#define SIMPLE_MODBUS_TCP_H

#include <WiFi.h>

class SimpleModbusTCP
```

```
{
public:
    SimpleModbusTCP(const char* ip, uint16_t port = 502);

    bool connect();
    void disconnect();
    bool isConnected();

    uint16_t readHoldingRegister(uint16_t address);
    bool     writeHoldingRegister(uint16_t address, uint16_t value);
    bool     readMultipleRegisters(uint16_t startAddress, uint16_t quantity,
uint16_t* buffer);

    bool     readCoil(uint16_t address);
    bool     writeCoil(uint16_t address, bool value);
    bool     readMultipleCoils(uint16_t startAddress, uint16_t quantity, uint8_t*
buffer);

private:
    IPAddress  _serverIP;
    uint16_t   _port;
    WiFiClient _client;
    uint16_t   _transactionID;

    uint16_t _readUInt16(uint8_t* data, uint8_t index);
    void     _sendRequest(uint8_t* request, uint8_t length);
    bool     _waitResponse(uint8_t* response, uint8_t expectedBytes, uint16_t
timeout = 5000);
};

#endif
```

ESP32 main code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <HTTPClient.h>          // For Google Sheets logging
#include "SimpleModbusTCP.h"

const char* WIFI_SSID     = "UTARSF2";     // <-- Change to your B310 SSID
const char* WIFI_PASSWORD = "utarsf102023";  // <-- Change to your B310
password

// Paste your Google Apps Script Web App URL here after deploying
const char* SHEETS_URL = "
https://docs.google.com/spreadsheets/d/1Vjbz6piZ7SSzaENMIZ-p-
lv5AZtoypW56mF8BKk5CWY/edit?gid=0#gid=0"; // <-- CHANGE THIS

const char* MQTT_BROKER    = "broker.emqx.io";
const int   MQTT_PORT      = 1883;  // Plain MQTT
const char* MQTT_USER      = "";              // No auth needed
const char* MQTT_PASS      = "";              // No auth needed
String mqttClientId = "ESP32_AgriPLC_" + String((uint32_t)ESP.getEfuseMac(),
HEX);  // Must be unique

const char*    PLC_IP   = "192.168.1.18";
const uint16_t PLC_PORT = 502;

const unsigned long POLL_INTERVAL    = 90000; // 90 seconds
const unsigned long RECONNECT_DELAY  = 30000; // 30 sec between MQTT
retries on slow internet

#define T_SENSORS          "plc/sensors"
#define T_DOSING           "plc/dosing"
```

```
#define T_STATUS          "plc/status"
#define T_ALARMS          "plc/alarms"
#define T_CTRL_MOTOR      "plc/control/motor"
#define T_CTRL_VALVE      "plc/control/valve"
#define T_CTRL_DOSING     "plc/control/dosing"
#define T_CTRL_REGISTER   "plc/control/register"

SimpleModbusTCP modbus(PLC_IP, PLC_PORT);
WiFiClient      wifiClient;
PubSubClient    mqtt(wifiClient);

unsigned long lastPollTime    = 0;
int modbusTimeoutCount        = 0;  // Count consecutive timeouts
const int MAX_TIMEOUTS        = 5;  // Force reconnect after 5 timeouts

bool getBit(uint8_t* buf, uint16_t bitIndex) {
   return (buf[bitIndex / 8] >> (bitIndex % 8)) & 0x01;}
float readFloat(uint16_t addr) {
   uint16_t regs[2];
   if (modbus.readMultipleRegisters(addr, 2, regs)) {
      uint32_t combined = ((uint32_t)regs[1] << 16) | regs[0];
      float f;
      memcpy(&f, &combined, sizeof(f));
      return f;
   }
   return -999.0f;
}

void mqttCallback(char* topic, byte* payload, unsigned int length) {
   char msg[256];
   if (length >= sizeof(msg)) length = sizeof(msg) - 1;
   memcpy(msg, payload, length);
   msg[length] = '\0';
```

```
Serial.printf("[MQTT IN] %s : %s\n", topic, msg);

StaticJsonDocument<128> doc;
if (deserializeJson(doc, msg) != DeserializationError::Ok) {
   Serial.println("[MQTT] Bad JSON");
   return;
}

// {"name":"stirrer1","val":1}
if (strcmp(topic, T_CTRL_MOTOR) == 0) {
   const char* name = doc["name"];
   bool val = (int)doc["val"] != 0;

   // Map name → Modbus coil address
   struct { const char* n; uint16_t addr; } motors[] = {
      {"stirrer1",  55},
      {"stirrer2",  56},
      {"stirrer3",  57},
      {"stirrer4",  58},
      {"pump_circ", 61}
   };
   for (auto& m : motors) {
      if (strcmp(name, m.n) == 0) {
         bool ok = modbus.writeCoil(m.addr, val);
         Serial.printf("[PLC] %s (M%d) = %s  %s\n",
            name, m.addr, val ? "ON" : "OFF", ok ? "OK" : "FAILED");
         if (ok) {
            delay(1000); // Wait 1s for PLC ladder logic to settle
            publishStatus();
         }
         return;
      }
```

```
    }
    Serial.printf("[MQTT] Unknown motor: %s\n", name);
}

// {"name":"main_valve","val":1}
else if (strcmp(topic, T_CTRL_VALVE) == 0) {
    const char* name = doc["name"];
    bool val = (int)doc["val"] != 0;

    struct { const char* n; uint16_t addr; } valves[] = {
        {"main_valve",   50},
        {"tank1_valve",  51},
        {"tank2_valve",  52},
        {"tank3_valve",  53},
        {"tank4_valve",  54},
        {"loop_valve",   59},
        {"output_valve", 60},
        {"flush_valve",  62}
    };
    for (auto& v : valves) {
        if (strcmp(name, v.n) == 0) {
            bool ok = modbus.writeCoil(v.addr, val);
            Serial.printf("[PLC] %s (M%d) = %s  %s\n",
                name, v.addr, val ? "ON" : "OFF", ok ? "OK" : "FAILED");
            if (ok) {
                delay(1000); // Wait 1s for PLC ladder logic to settle
                publishStatus();
            }
            return;
        }
    }
    Serial.printf("[MQTT] Unknown valve: %s\n", name);
}
```

```
// {"name":"dosing1_start","val":1}
else if (strcmp(topic, T_CTRL_DOSING) == 0) {
   const char* name = doc["name"];
   bool val = (int)doc["val"] != 0;

   struct { const char* n; uint16_t addr; } dosingBits[] = {
      {"dosing1_start", 26},
      {"dosing2_start", 27},
      {"dosing3_start", 28},
      {"dosing4_start", 29},
      {"dosing1_stop",  31},
      {"dosing2_stop",  34},
      {"dosing3_stop",  37},
      {"dosing4_stop",  40}
   };
   for (auto& d : dosingBits) {
      if (strcmp(name, d.n) == 0) {
         bool ok = modbus.writeCoil(d.addr, val);
         Serial.printf("[PLC] %s (M%d) = %s  %s\n",
            name, d.addr, val ? "ON" : "OFF", ok ? "OK" : "FAILED");
         if (ok) {
            delay(1000); // Wait 1s for PLC ladder logic to settle
            publishStatus();
            publishDosing();
         }
         return;
      }
   }
   Serial.printf("[MQTT] Unknown dosing bit: %s\n", name);
}

// {"addr":199,"val":100}
```

```
    else if (strcmp(topic, T_CTRL_REGISTER) == 0) {
        if (!doc.containsKey("addr") || !doc.containsKey("val")) {
            Serial.println("[MQTT] Missing addr or val");
            return;
        }
        uint16_t addr = doc["addr"];
        uint16_t val  = doc["val"];
        bool ok = modbus.writeHoldingRegister(addr, val);
        Serial.printf("[PLC] Write D%d = %d  %s\n", addr, val, ok ? "OK" :
"FAILED");
    }
}

void logToSheets(float ph, float temp, float ec, float flow, uint16_t waterLevel) {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("[Sheets] WiFi not connected - skip logging");
        return;
    }

    HTTPClient http;
    http.begin(SHEETS_URL);
    http.addHeader("Content-Type", "application/json");
    http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);

    // Build JSON payload
    StaticJsonDocument<200> doc;
    doc["water_level"] = waterLevel;
    doc["ph"]          = ph;
    doc["temperature"] = temp;
    doc["ec"]          = ec;
    doc["flow_rate"]   = flow;

    char payload[200];
```

```
    serializeJson(doc, payload);

    int httpCode = http.POST(payload);
    if (httpCode == HTTP_CODE_OK || httpCode == 302) {
        Serial.println("[Sheets] Data logged OK");
    } else {
        Serial.printf("[Sheets] Failed, HTTP code: %d\n", httpCode);
    }
    http.end();
}

void publishSensors() {
    StaticJsonDocument<200> doc;
    uint16_t wl = modbus.readHoldingRegister(30);
    if (wl == 0 && !modbus.isConnected()) { modbusTimeoutCount++; return; }
    modbusTimeoutCount = 0; // Reset on success
    float ph   = readFloat(90);
    float temp = readFloat(92);
    float ec   = readFloat(94);
    float flow = readFloat(114);

    doc["water_level"] = wl;
    doc["ph"]          = serialized(String(ph,   2));
    doc["temperature"] = serialized(String(temp, 2));
    doc["ec"]          = serialized(String(ec,   2));
    doc["flow_rate"]   = serialized(String(flow, 2));

    char buf[200];
    serializeJson(doc, buf);
    mqtt.publish(T_SENSORS, buf, true);
    Serial.printf("[MQTT OUT] %s : %s\n", T_SENSORS, buf);

    // Log to Google Sheets
```

```
   logToSheets(ph, temp, ec, flow, wl);
}
void publishDosing()
 {
   StaticJsonDocument<384> doc;
   doc["dosing1_freq"] = modbus.readHoldingRegister(170);
   doc["dosing2_freq"] = modbus.readHoldingRegister(172);
   doc["dosing3_freq"] = modbus.readHoldingRegister(174);
   doc["dosing4_freq"] = modbus.readHoldingRegister(176);
   doc["dosing1_vol"]  = serialized(String(readFloat(200), 2));
   doc["dosing2_vol"]  = serialized(String(readFloat(202), 2));
   doc["dosing3_vol"]  = serialized(String(readFloat(204), 2));
   doc["dosing4_vol"]  = serialized(String(readFloat(206), 2));
   doc["set_vol1"]     = modbus.readHoldingRegister(199);
   doc["set_vol2"]     = modbus.readHoldingRegister(196);
   doc["set_vol3"]     = modbus.readHoldingRegister(197);
   doc["set_vol4"]     = modbus.readHoldingRegister(198);

   char buf[384];
   serializeJson(doc, buf);
   mqtt.publish(T_DOSING, buf, true);
   Serial.printf("[MQTT OUT] %s : %s\n", T_DOSING, buf);
}

void publishStatus() {
   // Bulk coil reads (same as working read-only version)
   uint8_t coilA[1] = {0}; bool okA = modbus.readMultipleCoils(16, 4,  coilA); //
M16-M19
   uint8_t coilB[2] = {0}; bool okB = modbus.readMultipleCoils(26, 15, coilB); //
M26-M40
   uint8_t coilC[2] = {0}; bool okC = modbus.readMultipleCoils(50, 13, coilC); //
M50-M62
```

```
StaticJsonDocument<512> doc;

// Valves (Bulk C, offset from M50)
doc["main_valve"]   = okC ? getBit(coilC, 0)  : -1;
doc["tank1_valve"]  = okC ? getBit(coilC, 1)  : -1;
doc["tank2_valve"]  = okC ? getBit(coilC, 2)  : -1;
doc["tank3_valve"]  = okC ? getBit(coilC, 3)  : -1;
doc["tank4_valve"]  = okC ? getBit(coilC, 4)  : -1;
doc["loop_valve"]   = okC ? getBit(coilC, 9)  : -1;
doc["output_valve"] = okC ? getBit(coilC, 10) : -1;
doc["flush_valve"]  = okC ? getBit(coilC, 12) : -1;

// Motors (Bulk C)
doc["stirrer1"]     = okC ? getBit(coilC, 5)  : -1;
doc["stirrer2"]     = okC ? getBit(coilC, 6)  : -1;
doc["stirrer3"]     = okC ? getBit(coilC, 7)  : -1;
doc["stirrer4"]     = okC ? getBit(coilC, 8)  : -1;
doc["pump_circ"]    = okC ? getBit(coilC, 11) : -1;

// Dosing ON/OFF (Bulk A)
doc["dosing1_on"]   = okA ? getBit(coilA, 0) : -1;
doc["dosing2_on"]   = okA ? getBit(coilA, 1) : -1;
doc["dosing3_on"]   = okA ? getBit(coilA, 2) : -1;
doc["dosing4_on"]   = okA ? getBit(coilA, 3) : -1;

// Dosing start/stop (Bulk B, offset from M26)
doc["dosing1_start"] = okB ? getBit(coilB, 0)  : -1;
doc["dosing2_start"] = okB ? getBit(coilB, 1)  : -1;
doc["dosing3_start"] = okB ? getBit(coilB, 2)  : -1;
doc["dosing4_start"] = okB ? getBit(coilB, 3)  : -1;
doc["dosing1_stop"]  = okB ? getBit(coilB, 5)  : -1;
doc["dosing2_stop"]  = okB ? getBit(coilB, 8)  : -1;
doc["dosing3_stop"]  = okB ? getBit(coilB, 11) : -1;
```

```
    doc["dosing4_stop"]  = okB ? getBit(coilB, 14) : -1;

    char buf[512];
    serializeJson(doc, buf);
    mqtt.publish(T_STATUS, buf, true);
    Serial.printf("[MQTT OUT] %s : %s\n", T_STATUS, buf);
}

void publishAlarms() {
    uint8_t coilD[1] = {0};
    bool okD = modbus.readMultipleCoils(400, 5, coilD); // M400-M404
    bool masterAlarm = modbus.readCoil(450);            // M450

    StaticJsonDocument<128> doc;
    doc["alarm_master"] = masterAlarm;
    doc["alarm1"] = okD ? getBit(coilD, 0) : -1;
    doc["alarm2"] = okD ? getBit(coilD, 1) : -1;
    doc["alarm3"] = okD ? getBit(coilD, 2) : -1;
    doc["alarm4"] = okD ? getBit(coilD, 3) : -1;
    doc["alarm5"] = okD ? getBit(coilD, 4) : -1;

    if (masterAlarm) Serial.println("[PLC] !! ALARM TRIGGERED !!");

    char buf[128];
    serializeJson(doc, buf);
    mqtt.publish(T_ALARMS, buf, true);
    Serial.printf("[MQTT OUT] %s : %s\n", T_ALARMS, buf);
}

void connectWiFi() {
    Serial.printf("Connecting to WiFi: %s", WIFI_SSID);

    // Set static DNS before connecting - fixes rc=-2 on slow internet
```

```
    WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE,
                IPAddress(8,8,8,8),   // Primary DNS: Google
                IPAddress(8,8,4,4));  // Secondary DNS: Google

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500); Serial.print(".");
    }
    delay(2000); // Wait 2 sec for DNS to stabilize after connect
    Serial.printf("\nWiFi OK  IP: %s\n", WiFi.localIP().toString().c_str());
    Serial.printf("Gateway : %s\n", WiFi.gatewayIP().toString().c_str());
    Serial.printf("DNS     : 8.8.8.8 (Google)\n");
}

bool connectMQTT() {
    // Test DNS resolution first
    IPAddress brokerIP;
    Serial.print("Resolving broker.emqx.io ... ");
    if (WiFi.hostByName("broker.emqx.io", brokerIP)) {
        Serial.printf("OK -> %s\n", brokerIP.toString().c_str());
    } else {
        Serial.println("DNS FAILED");
        return false;
    }

    // Test TCP connection first before MQTT
    Serial.printf("Testing TCP port %d ... ", MQTT_PORT);
    WiFiClient testClient;
    testClient.setTimeout(30000); // 30 sec for very slow internet
    if (!testClient.connect(brokerIP, MQTT_PORT)) {
        Serial.println("TCP FAILED - internet too slow, will retry");
        testClient.stop();
        return false;
```

```
    }
    Serial.println("TCP OK");
    testClient.stop();
    delay(1000);

    Serial.printf("Connecting to MQTT: %s:%d ...", MQTT_BROKER,
MQTT_PORT);
    if (mqtt.connect(mqttClientId.c_str(), MQTT_USER, MQTT_PASS)) {
        Serial.println(" OK");
        mqtt.subscribe(T_CTRL_MOTOR);
        mqtt.subscribe(T_CTRL_VALVE);
        mqtt.subscribe(T_CTRL_DOSING);
        mqtt.subscribe(T_CTRL_REGISTER);
        Serial.println("[MQTT] Subscribed to control topics");
        return true;
    }
    Serial.printf(" FAILED rc=%d\n", mqtt.state());
    return false;
}

void setup() {
    Serial.begin(115200);
    delay(1000);
    Serial.println("\n=== Smart Agriculture ESP32 MQTT Bridge ===");
    Serial.printf("MQTT Client ID: %s\n", mqttClientId.c_str());

    connectWiFi();

    // Increase TCP timeout for slow internet
    wifiClient.setTimeout(30000); // 30 seconds TCP timeout
    mqtt.setServer(MQTT_BROKER, MQTT_PORT);
    mqtt.setCallback(mqttCallback);
    mqtt.setBufferSize(512);
```

```
  mqtt.setSocketTimeout(30);   // 30 sec timeout for slow 4G
  mqtt.setKeepAlive(60);       // Send keepalive every 60 sec
  connectMQTT();

  Serial.print("Connecting to PLC...");
  if (modbus.connect()) Serial.println(" OK");
  else                  Serial.println(" FAILED - will retry in loop");
}

void loop() {

  // Keep Wi-Fi alive
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("[WiFi] Lost. Reconnecting...");
    connectWiFi();
  }

  // Keep MQTT alive - this also calls mqttCallback for incoming messages
  if (!mqtt.connected()) {
    Serial.println("[MQTT] Lost. Reconnecting...");
    delay(5000); // Wait 5s before retry on slow internet
    if (!connectMQTT()) {
      delay(RECONNECT_DELAY);
    }
    return;
  }
  mqtt.loop();

  // Keep PLC alive - ONLY reconnect if truly disconnected
  // FX3U-ENET-ADP supports only 1 TCP connection
  // Frequent reconnects will lock out the PLC for 30-60 seconds
  if (!modbus.isConnected()) {
```

```
        Serial.println("[PLC] Connection lost. Waiting 30s for PLC to release TCP
slot...");
        modbusTimeoutCount = 0;
        delay(30000); // Wait 30 sec for FX3U-ENET-ADP to release old connection
        Serial.println("[PLC] Reconnecting...");
        if (!modbus.connect()) {
            Serial.println("[PLC] Reconnect failed - will retry");
            delay(RECONNECT_DELAY);
            return;
        }
        delay(500);
        Serial.println("[PLC] Reconnected OK");
        return;
    }

    // Too many timeouts but still connected - flush and wait
    if (modbusTimeoutCount >= MAX_TIMEOUTS) {
        Serial.println("[PLC] Too many timeouts - flushing and pausing 5s...");
        modbusTimeoutCount = 0;
        delay(5000); // Just wait, don't disconnect
    }

    // Publish all data every 10 seconds
    unsigned long now = millis();
    if (now - lastPollTime >= POLL_INTERVAL) {
        lastPollTime = now;

        // Always verify PLC connection before polling
        if (!modbus.isConnected()) {
            Serial.println("[PLC] Lost before poll. Reconnecting...");
            if (!modbus.connect()) {
                Serial.println("[PLC] Reconnect failed - skipping this poll");
                return;
```

```
        }
        delay(500); // Give PLC time to settle after reconnect
        Serial.println("[PLC] Reconnected OK");
      }

      Serial.println("\n--- Publishing PLC data ---");
      publishSensors();
      delay(100); // Small gap between publish functions
      publishDosing();
      delay(100);
      publishStatus();
      delay(100);
      publishAlarms();
   }
}
```

Appendix B: Sensor Data Logs link

https://docs.google.com/spreadsheets/d/1Vjbz6piZ7SSzaENMIZ-p-lv5AZtoypW56mF8BKk5CWY/edit?gid=0#gid=0