Smart Plant Watering with ESP32 and MQTT

Paul-Christian Thoma(32436)-Elham Mohammadi(32475)-Deniz-Zeynep Adem(33784)

1. Introduction

By Elham Mohammadi

Maintaining optimal soil moisture is essential for plant health, especially in indoor or unattended environments. Manual watering is often inconsistent, which can lead to overwatering, underwatering, or resource waste. This project addresses these challenges by developing a semi-autonomous, Wi-Fi-enabled plant watering system that monitors soil moisture in real time and triggers irrigation when needed. This is especially helpful when leaving over a longer period of time but still wanting to make sure that the plants covered by the system are regularly watered. added by PT

The system consists of two ESP32-S3 microcontrollers communicating via Wi-Fi using the MQTT protocol. One ESP32 functions as a sensor node, measuring soil moisture and publishing readings to an MQTT broker. The second ESP32 acts as an actuator node (pump nodes), subscribing to control messages and operating a peristaltic pump through servo control.

A key feature of the system is that the decision to activate watering is not made directly on the sensor node. Instead, the control logic is handled externally using Node-RED, which processes incoming MQTT data and publishes appropriate commands. This approach allows the control logic to be easily modified without reprogramming the ESP32 devices.

Sensor data is also logged to Google Sheets via Google Apps Script, and email notifications are triggered when dry soil conditions are detected. The system integrates with Node-RED for real-time visualization and monitoring.

By combining local sensing, cloud-based logic, and remote monitoring tools, this system provides a scalable, modular approach to smart irrigation in home or laboratory environments.

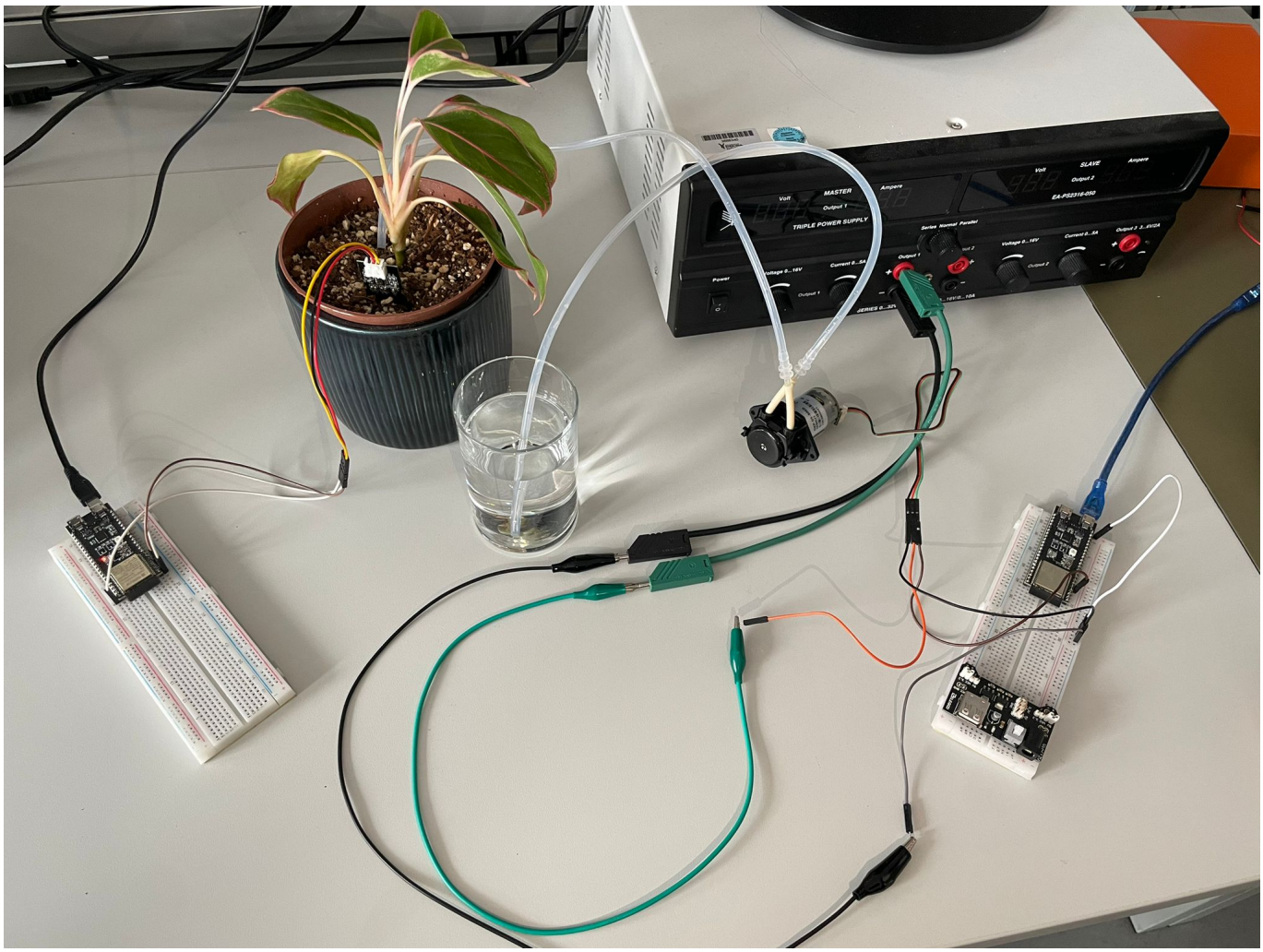


Figure 1. Automated Plant Watering System Prototype

2. Materials and Methods

2.1 Materials

By Elham Mohammadi

- **ESP32-S3-VROOM-1 (x2):** Used as the main microcontrollers for sensor and pump subsystems. These Wi-Fi-enabled boards handle data acquisition, communication, and actuation tasks.
- **Capacitive Soil Moisture Sensor:** Provides more reliable and corrosion-resistant measurements compared to resistive sensors. It measures soil moisture based on dielectric properties.
- **DFR0523 Peristaltic Pump:** A compact, DC-operated pump capable of controlled water delivery, ideal for small-scale irrigation.
- **Silicone Hose (x2):** Used to direct water flow from the peristaltic pump to the plant’s soil.
- **Breadboards (x2):** Prototyping platforms used to connect electronic components without soldering.
- **Jumper Wires:** Used for electrical connections between components on the breadboard and to the ESP32 boards.

- **Elegoo Breadboard Power Module:** Provides regulated 3.3V and 5V power outputs to the components on the breadboard.
- **9V Battery and Barrel Jack Connector:** Powers the pump subsystem. The barrel jack simplifies safe and secure connection to the power supply.
- **Water Source**: A container holding the water that will be pumped to the plant. It must be placed lower than the pump inlet to ensure smooth suction.

2.2 Software

By Elham Mohammadi and Deniz Zeynep Adem

Arduino IDE:

- Used to write, compile, and upload the firmware to both ESP32-S3 microcontrollers.
- Supported libraries for Wi-Fi, MQTT, HTTP, and analog sensor interfacing.
- Essential for debugging and serial monitoring during development.

Node-RED:

- A flow-based programming tool used to visualize MQTT data.
- Helpful for monitoring real-time soil moisture values and pump activity.
- Provides a user-friendly interface to create custom dashboards and automate responses.

Google Sheets:

- Serves as the cloud-based data logger for sensor readings, system status, and pump activity.
- Allows for timestamped tracking of system behavior.
- Easily shareable and can be exported for further analysis.

Google Apps Script:

- A backend script that receives HTTP POST requests from the ESP32.
- Logs moisture values, system status, and pump actions into Google Sheets.

MQTT Broker (HiveMQ Public Broker):

- Facilitates real-time communication between the sensor unit and pump unit.
- Allows the moisture sensor ESP32 to publish status and trigger messages.
- The pump ESP32 subscribes to control commands (`WATER`, `STOP`) and publishes its response (`Watered`, `Stopped`).

Tinkercad / Fritzing:

- Used to create circuit diagrams representing the sensor and pump setups.
- Helped document the physical wiring for presentations and documentation.

AI Tools:

- Assisted in problem solving and code debugging.

2.3 Setup Overview

by Elham Mohammadi and Deniz Zeynep Adem

The Sensor Node (ESP32-S3) and Pump Node (ESP32-S3) both connect to the local Wi-Fi network. The MQTT connection is established with the HiveMQ public broker.

Soil Moisture Reading:

- The Sensor Node reads analog soil moisture values using a sensor connected to one of its GPIO pins.
- It publishes the moisture value to a specific MQTT topic and determines the soil status (Moist, Dry, or Alert).
- At the same time, it sends a JSON payload via HTTP POST to a Google Apps Script, which logs the data in Google Sheets, including timestamp, status, moisture level, and any action taken.

Node-RED Evaluation and Decision:

- Node-RED runs on a separate device or server, subscribing to the MQTT topic from the Sensor Node.
- It continuously evaluates the incoming moisture values.
- If the moisture is below a defined threshold, Node-RED publishes a control message (`WATER`) to the pump topic.
- If the soil is moist, Node-RED sends a `STOP` message.

Pump Activation:

- The Pump Node subscribes to the control topic.
- When it receives a `WATER` message, it activates a peristaltic pump using a latching relay.
- If it receives a `STOP` message, it turns the pump off.
- It also publishes its status back to MQTT so the Sensor Node and Node-RED know the action was performed.

Logging and Alerts:

- Google Sheets logs every sensor reading and pump action.

If:

- The **moisture readings remain unchanged** for several cycles, or
- The **pump does not respond to control messages**,
- Then the Sensor Node logs an **“Alert ⚠ Maintenance Needed”** in the sheet.

- Additionally, Node-RED and Google Apps Scripts can send email alerts to notify the user that **soil is too dry** or **maintenance is required**.

To bring the designed system to life, the next step involves assembling the hardware components and ensuring proper electrical connections. The wiring setup includes the soil moisture sensor connected to the Sensor Node and a peristaltic pump controlled by the Pump Node via a latching relay. Accurate and safe wiring is essential to ensure reliable communication, sensor readings, and pump activation.

The following diagrams, created using Tinkercad, illustrate the complete hardware connections for both the Sensor and Pump nodes in the system.

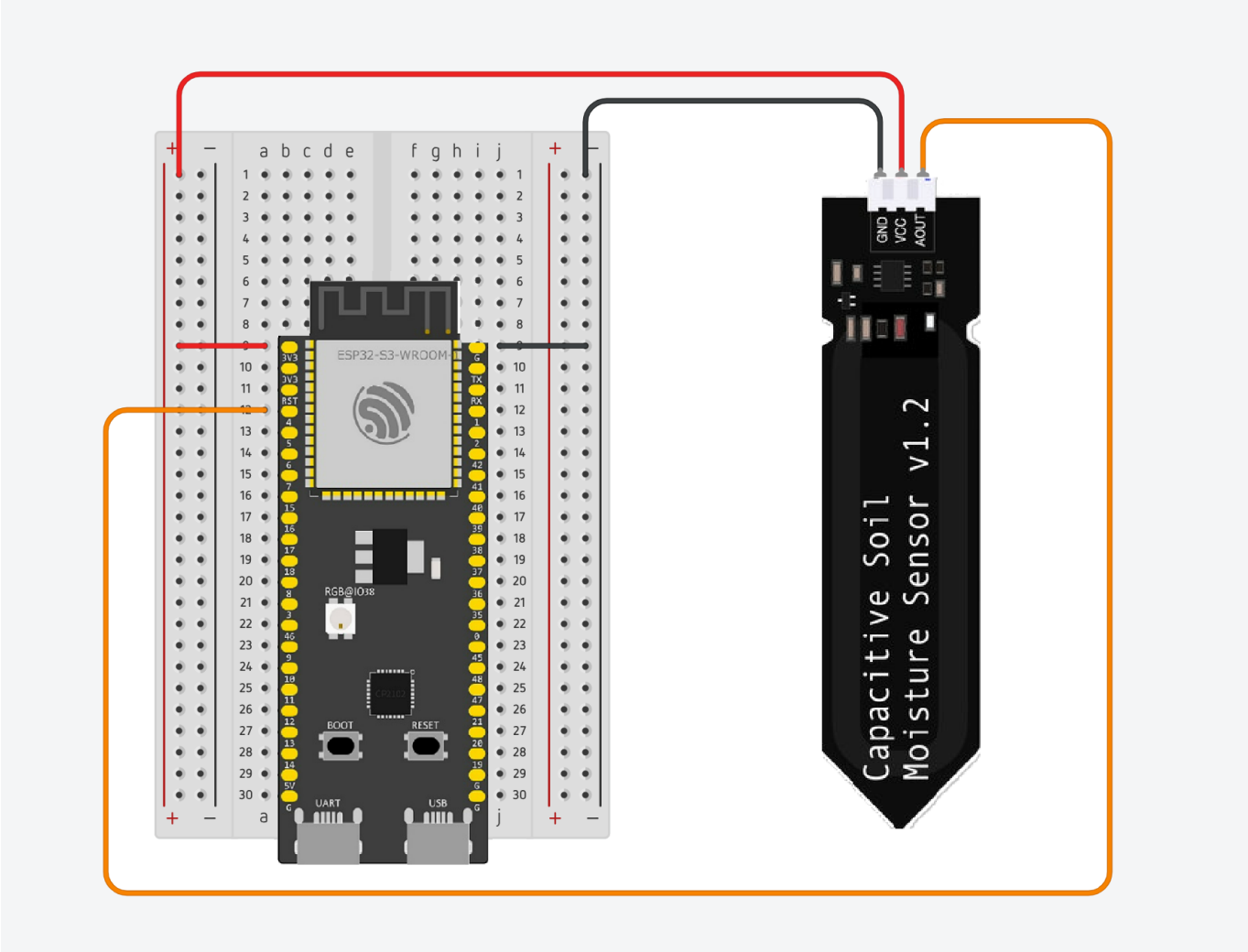


Figure 2. Schematic Diagram of Sensor Setup (designed by Deniz-Zeynep Adem)

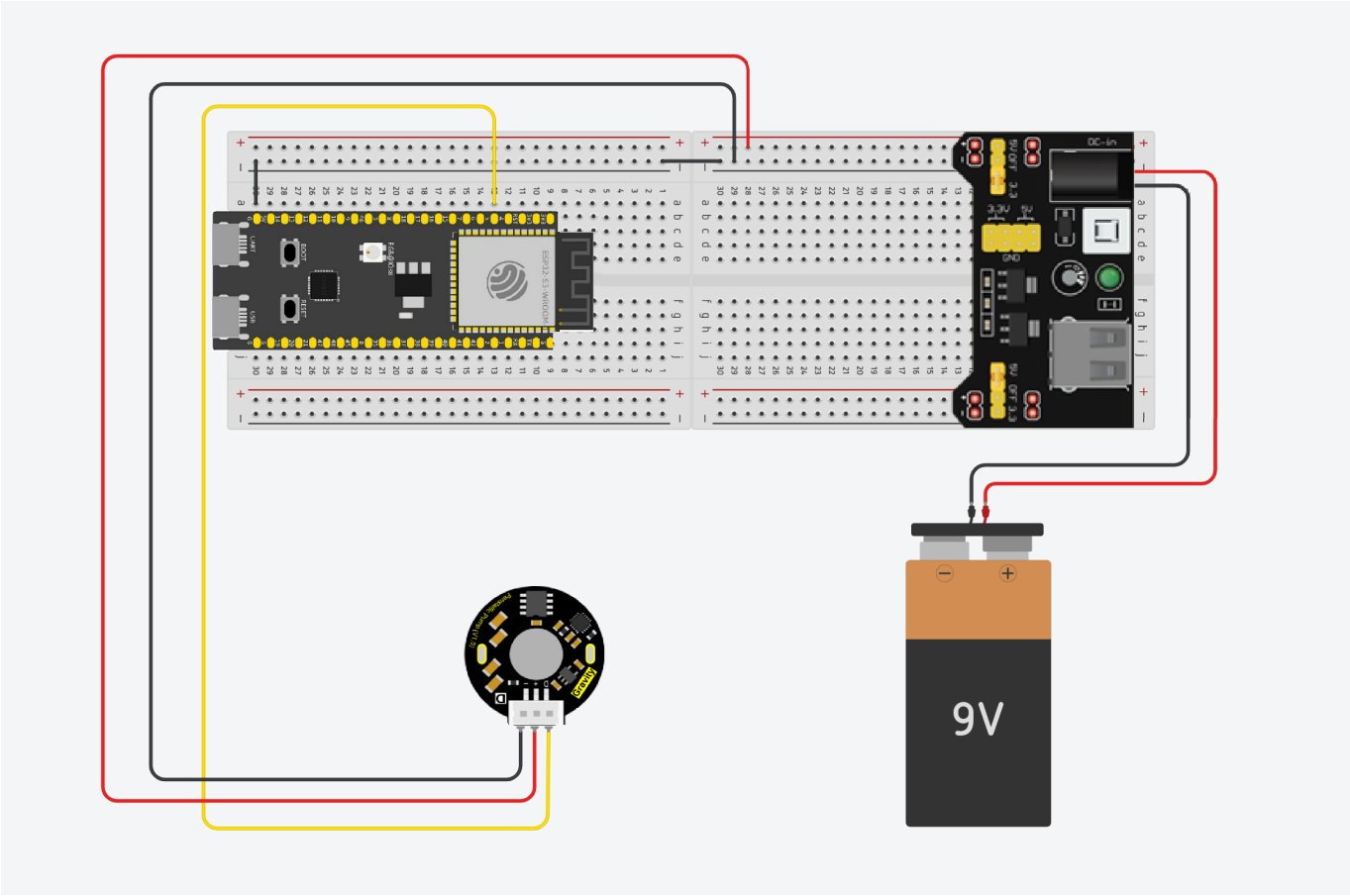


Figure 3. Schematic Diagram of Pump Setup (designed by Deniz-Zeynep Adem)

3. Results

3.1 Code Explanation

By Paul-Christian Thoma & Elham Mohammadi & Deniz-Zeynep Adem

Sensor Node Code:

```
#include <WiFi.h>              // Library for WiFi connection
#include <HTTPClient.h>        // Library for HTTP requests
#include <MQTTClient.h>        // Lightweight MQTT client

// Wi-Fi and MQTT credentials
const char WIFI_SSID[] = "iotlab";
const char WIFI_PASSWORD[] = "iotlab18";
const char MQTT_BROKER[] = "broker.hivemq.com";  // Public MQTT broker
const int MQTT_PORT = 1883;
const char MQTT_CLIENT_ID[] = "ESP32_Sensor_001"; // Unique client ID

// Google Apps Script webhook URL for sending data to Google Sheets
const char GOOGLE_SHEET_URL[] =
"https://script.google.com/macros/s/AKfycbwWx8W6ov2rufz6ODUtUngPOhJMdYazwkHo
```

```
zmzk8-3n6gyL-qNCGgX9YxKtBTL7Qm5Vfg/exec";

// MQTT Topics
const char PUBLISH_TOPIC_VALUE[] = "plant/moisture/value";   // Publishes
raw moisture reading
const char PUBLISH_TOPIC_STATUS[] = "plant/moisture/status"; // Publishes
human-readable status
const char COMMAND_TOPIC[] = "plant/pump/control";           // Used to send
watering command

// Sensor pin and threshold
const int sensorPin = 4;          // GPIO pin connected to moisture sensor
const int threshold = 2500;       // Dry threshold — adjust based on
calibration

WiFiClient net;                   // WiFi client object
MQTTClient mqtt(256);             // MQTT client with 256-byte buffer

unsigned long lastRead = 0;
const int readInterval = 5000;    // Time between measurements (5 seconds)

// Function to connect to MQTT broker
void connectToMQTT() {
  mqtt.begin(MQTT_BROKER, MQTT_PORT, net);     // Start MQTT with broker
info
  while (!mqtt.connect(MQTT_CLIENT_ID)) {      // Keep trying until
connected
    Serial.print(".");
    delay(500);
  }
  Serial.println("\nMQTT connected.");
}

// Function to send data to Google Sheets using HTTP POST
void postToGoogleSheet(int moisture, const String& status) {
  HTTPClient http;                             // Create HTTP client
  http.begin(GOOGLE_SHEET_URL);                // Set URL for POST
  http.addHeader("Content-Type", "application/json"); // Send as JSON

  // Format data as JSON payload
  String action;
  if (status == "Dry") {
    action = "Pump is ON □";
  } else if (status == "Moist") {
    action = "OK □";
  } else {
    action = "Alert ⚠□ Maintenance Needed";
  }

  String jsonPayload = "{\"moisture\":\"" + String(moisture) +
                       "\",\"status\":\"" + status +
```

```
                        "\",\"action\":\"" + action + "\"}";

  Serial.println("POSTing: " + jsonPayload);

  int responseCode = http.POST(jsonPayload);   // Send POST request
  Serial.println("HTTP Response: " + String(responseCode)); // Print result
  http.end();                                  // Close connection
}

void setup() {
  Serial.begin(115200);                  // Initialize serial monitor
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  // Connect to WiFi

  // Wait until WiFi is connected
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("\nWiFi connected.");

  connectToMQTT();                       // Connect to MQTT broker
}

void loop() {
  mqtt.loop();                           // Handle background MQTT tasks

  // Only read sensor and publish every 5 seconds
  if (millis() - lastRead > readInterval) {
    int value = analogRead(sensorPin);               // Read analog moisture
value
    String status = value > threshold ? "Dry 🏜" : "Moist 🌱"; // Determine
status

    // Publish raw value and status to MQTT topics
    mqtt.publish(PUBLISH_TOPIC_VALUE, String(value));
    mqtt.publish(PUBLISH_TOPIC_STATUS, status);

    // Print to Serial for debugging
    Serial.print("Soil Moisture: ");
    Serial.print(value);
    Serial.print(" → Status: ");
    Serial.println(status);

    // Send the data to Google Sheets
    postToGoogleSheet(value, status);

    // If soil is dry, send water command
    if (status.startsWith("Dry")) {
      mqtt.publish(COMMAND_TOPIC, "WATER"); // Triggers pump
    }
```

```
    lastRead = millis(); // Update time for next read
  }
}
```

Pump Node Code:

By Paul-Christian Thoma & Elham Mohammadi & Deniz-Zeynep Adem

```
#include <WiFi.h>              // WiFi library for ESP32
#include <PubSubClient.h>      // MQTT library
#include <Servo.h>             // For controlling the servo pump
#include <HTTPClient.h>        // For HTTP POST to Google Sheets

// Wi-Fi credentials
const char* ssid = "iotlab";
const char* password = "iotlab18";

// MQTT broker
const char* mqtt_server = "broker.hivemq.com";
const char* control_topic = "plant/pump/control";

// Google Apps Script Web App URL
const char* google_script_url =
"https://script.google.com/macros/s/AKfycbwWx8W6ov2rufz6ODUtUngPOhJMdYazwkHo
zmzk8-3n6gyL-qNCGgX9YxKtBTL7Qm5Vfg/exec";

WiFiClient espClient;
PubSubClient client(espClient);

Servo pumpServo;
const int servoPin = 5;

void setup_wifi() {
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
}

void postToGoogleSheet(const String& status, const String& action) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(google_script_url);
    http.addHeader("Content-Type", "application/json");

    // Build JSON payload
    String payload = "{\"status\":\"" + status + "\",\"moisture\":\"-
```

```
\",\"action\":\"" + action + "\"}";

    int httpResponseCode = http.POST(payload);
    Serial.print("Google Sheet POST response: ");
    Serial.println(httpResponseCode);
    http.end();
  } else {
    Serial.println("WiFi not connected. Could not send to Google Sheets.");
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String msg = "";
  for (unsigned int i = 0; i < length; i++) {
    msg += (char)payload[i];
  }
  msg.trim();

  Serial.print("Message received [");
  Serial.print(topic);
  Serial.print("]: ");
  Serial.println(msg);

  if (String(topic) == control_topic) {
    if (msg == "WATER" || msg == "ON") {
      pumpServo.write(180);
      Serial.println("Pump is ON □");
      postToGoogleSheet("Dry ⊗□", "Pump is ON");
    } else if (msg == "OFF") {
      pumpServo.write(90);
      Serial.println("Pump is OFF □");
      postToGoogleSheet("Moist □", "Pump is OFF");
    }
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32_Pump_Node")) {
      Serial.println("connected");
      client.subscribe(control_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

```
void setup() {
  Serial.begin(115200);
  setup_wifi();
  pumpServo.attach(servoPin);
  pumpServo.write(90); // Default OFF position
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

3.3 Code Logic/Overview:

by Paul Thoma

This project employs two ESP32 microcontroller systems to create a smart irrigation system capable of:

- Monitoring soil moisture levels
- Controlling a pump remotely using MQTT
- logging all measurements to Google Sheets for data tracking
- sending an email alert if irrigation is not working

For that it is divided into two key parts: 1. Soil Moisture Publishing node 2. Pump Control Subscriber node

How the code is achieving that and the logic behind that will be explained in the following text.

3.4 Soil Moisture Publishing Node

Measures soil and controls watering logic via MQTT.

This ESP32 is responsible for:

- Measuring soil moisture using a capacitive soil moisture sensor
- Publishing the data to an MQTT broker (in this case : broker.hivemq.com)
- Logging each measurement to a Google Sheet via a webhook
- Uploading the values so that node-red can send watering commands if values are above threshold and can send an alert email if watering does not work

Setup:

- When uploading the code for the first time make sure the correct COM Port is selected

- Connect Moisture Sensor to GPIO Pin and change that pin in the code (in this case GPIO 4)
- Google Apps Script used to connect to Google Sheets using 'HTTPClient.'
- change WiFi credentials
- insert what broker is being used

Key Libraries used:

- WiFi.h - Connects to WiFi
- MQTTClient.h - Publishes MQTT Messages
- HTTPClient.h - Sends POST request to Google Sheets

Loop Behavior:

- Reads soil moisture value
- Sends the value and status to MQTT topics
- Sends data to Google Sheets via Webhook
- If soil value is above set threshold in node-red it sends ON command to pump
- If moisture value has not decreased below set threshold within 10 seconds it triggers an email alert via node red
- If moisture value below 1400 sends OFF command to pump

3.5 Pump Control Subscriber Node

Controls pump based on MQTT messages.

This ESP32 handles the actuation based on MQTT control messages and is responsible for:

- Listening to the MQTT topic
- Controls pump via GPIO 5
- Turns the pump ON and OFF depending on commands

Key Libraries used:

- WiFi.h - Connects to WiFi
- PubSubClient.h - Subscribes to MQTT topics
- Servo.h - Operates pump

Setup:

- When uploading the Code for the first time make sure the correct COM Port is selected
- Connect pump to GPIO pin (in this case GPIO5)
- Connect pump to power source and ground

Callback Function:

- Interprets received MQTT messages
- ON → sets servo to 180° (pump ON)
- OFF → sets servo to 90° (pump OFF)
- Logs current state to serial monitor

Loop:

- Ensures Constant MQTT connection
- Waits for new MQTT messages

3.6 Google Apps Script and Google sheets

by Deniz Zeynep Adem

Google Sheets and Google Apps Script play a key role in the data logging and monitoring functionality of the project. Google Sheets acts as a live, cloud-based dashboard where all sensor readings, status updates, and system actions are recorded in real time. To enable this, a custom Google Apps Script was deployed as a web app, which allows external devices—like the ESP32 microcontroller—to send HTTP POST requests directly to the sheet.

The script receives and analyzes incoming JSON data containing the timestamp, moisture level, system status (such as “Moist □” or “Dry ☹”), and any actions taken (e.g., “Pump ON” or “Alert/Maintenance Needed”). This integration not only removes the need for manual logging but also allows for enhanced automation features, such as triggering email alerts when the moisture drops below a threshold or when abnormal behavior is detected.

Overall, the use of Google Sheets and Apps Script provides a simple yet powerful solution for real-time monitoring, data storage, and system feedback without the need for additional servers or complex backend infrastructure.

The saved data can be accessed through this link:

https://docs.google.com/spreadsheets/d/1N0_HnzShdQWwzD-0LYbjcm9xt-VcR1N9Uz19G0Y0euU/edit?usp=sharing

Step-by-Step Guide to Connect ESP32 to Google Sheets

1. Create a Google Sheet

- Open [Google Sheets](https://sheets.google.com) and create a new sheet.
- Name the sheet something like **“Automated Plant Watering Data”**.
- Rename the first sheet tab to **“Sheet1”** (or match it with the script).
- Add column headers in Row 1:
- **A1:** Timestamp
- **B1:** Action
- **C1:** Moisture
- **D1:** Status

2. Open Google Apps Script

- In your Google Sheet, go to the top menu:
- **Extensions → Apps Script**
- Delete any default code and paste the example code which is written down below.
- Rename the project (e.g., `SoilMoistureLogger`).

3. Deploy as a Web App

In the Apps Script editor:

- 1. Click **Distribute → Manage deployments** (or “New deployment”).
- 2. Choose **“Web app”** as the deployment type.
- 3. Set:
- **Description:** `Soil Moisture Logger`
- **Execute as:** `Me`
- **Who has access:** `Anyone`
- 4. Click **Distribute**.
- 5. **Authorize access** when prompted.
- 6. Copy the **Web App URL** – this is the endpoint your ESP32 will POST data to.

4. Integrate the Web App URL into Your ESP32 Code

In your ESP32 Arduino code, update the `GOOGLE_SHEET_URL`:

```
const char GOOGLE_SHEET_URL[] =
“https://script.google.com/macros/s/your_script_id/exec”;
```

Don't forget to replace `“your_script_id”` with the full Web App URL you copied!

5. Upload and Run the ESP32 Code

Upload your finalized ESP32 code via Arduino IDE or PlatformIO.

- **The ESP32 will:**
- Read the soil moisture sensor.
- Format the data into a JSON string.
- Send it via HTTP POST to your Google Apps Script Web App.
- Your script writes it into the top of the sheet automatically.

6. (Optional) Enable Email Alerts

- The script automatically sends emails if `“Alert”` is in the `action` string.
- You can customize the `MailApp.sendEmail()` line to notify multiple people or change message formatting.

With this setup:

- Your sensor system pushes real-time data into a Google Sheet.
- You receive email alerts for critical statuses like `“Maintenance Needed”`.

Google Apps Script Codes:

```
try {
  const sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Sheet1");
  const data = JSON.parse(e.postData.contents);
  const status = data.status || "";
  const moisture = data.moisture || "";
  const action = data.action || "";
  const timestamp = new Date();
  // Insert new row at top (after header)
  sheet.insertRows(2, 1);
```

```
  sheet.getRange(2, 1).setValue(timestamp);   // Column A: Timestamp
  sheet.getRange(2, 2).setValue(action);      // Column B: Action (e.g.,
Pump is ON)
  sheet.getRange(2, 3).setValue(moisture);    // Column C: Moisture value
  sheet.getRange(2, 4).setValue(status);      // Column D: Status (e.g.,
Moist )
  // Optional: Send email alert when there's a maintenance alert
  if (action.toLowerCase().includes("alert")) {
    const emailAddress = "youremail@example.com"; // Replace with your email
    const subject = "⚠ Plant System Alert";
    const message = `An alert has been logged in your system:\n\n` +
                    `Status: ${status}\nMoisture: ${moisture}\nAction:
${action}\nTime: ${timestamp}`;
    MailApp.sendEmail(emailAddress, subject, message);
  }
  return
ContentService.createTextOutput("Success").setMimeType(ContentService.MimeTy
pe.TEXT);
} catch (error) {
  Logger.log("Error: " + error);
  return ContentService.createTextOutput("Error: " + error)
                       .setMimeType(ContentService.MimeType.TEXT);
}
```

4. Node-RED Overview

Created by Paul-Christian Thoma & Documented by Elham Mohammadi

Node-RED is used as an external control system. It subscribes to MQTT topics published by the sensor node, applies threshold rules to determine soil dryness, and publishes control commands to the pump node. This separation allows the logic to be updated or expanded easily without changing ESP32 firmware.

Node-RED also supports visualization of sensor data and logging, which facilitates remote monitoring. Email notifications are triggered when dry conditions are detected but not improving within a set period of time, alerting users to take action or verify the system.

Below is a video showcasing the node-Red flow and showing the code and possible errors that came up and how they were fixed. Starting with the discussion part of this documentation.

https://student-wiki.eolab.de/lib/exe/fetch.php?w=120&h=120&tok=b20d52&media=amc:ss2025:group-f:whatsapp_image_2025-07-28_at_12.23.27_am.jpeg

Figure 4. Documentation of the data in note-red (designed by Paul-Christian Thoma)

5. Discussion and Conclusion

By Elham Mohammadi & Paul-Christian Thoma & Deniz-Zeynep Adem

The project successfully demonstrates an automated plant watering system based on real-time soil moisture monitoring and MQTT communication. Separating control logic from the sensor node to Node-RED allows flexible and remote decision-making. Integration with Google Sheets and email notifications adds monitoring and alert capabilities.

The setup of the hardware is quick and easy especially since the water pump had a build in relay. As long as the right amount of power in this case 9V can be supplied to the pump. Most of the effort was to program the software and setup the connection to a cloud based spreadsheet.

When setting up the email-Alert extra steps where needed to allow node-red to send an email. For example creating an app password for the gmail email and enabling 2FA authentication. If feasible it might be an option to rely on sms messaging or a telegram bot for communicating the alert.

A possible advantage would be real time communication with no delay. Additionally it was necessary to include a command to prevent the alert from being triggered multiple times during the same event. Otherwise if the pump would malfunction once the alert would continously be send every 10 seconds until it was fixed which would lead to the inbox being filled with e-mails.

As of right now the system is only suitable for indoor application of plants inside a living room. Since the current setup is not weatherproof/waterproof.

To make it self sustainable it could be considered to use a PV-module for supplying power and charging the battery which is powering the pump. It should be connected to a charging circuit to prevent overcharging of the battery. To safe energy a deep sleep function could be integrated so the sensor is not constantly reading values but rather every couple hours since the moisture will most likely not rapidly change. When setting it up outside a stable connection to a local WiFi network needs to be possible.

5.1 Limitations & Improvements

By Elham Mohammadi

While the system successfully demonstrates automated irrigation using MQTT and ESP32 microcontrollers, there are several limitations that affect its performance and scalability:

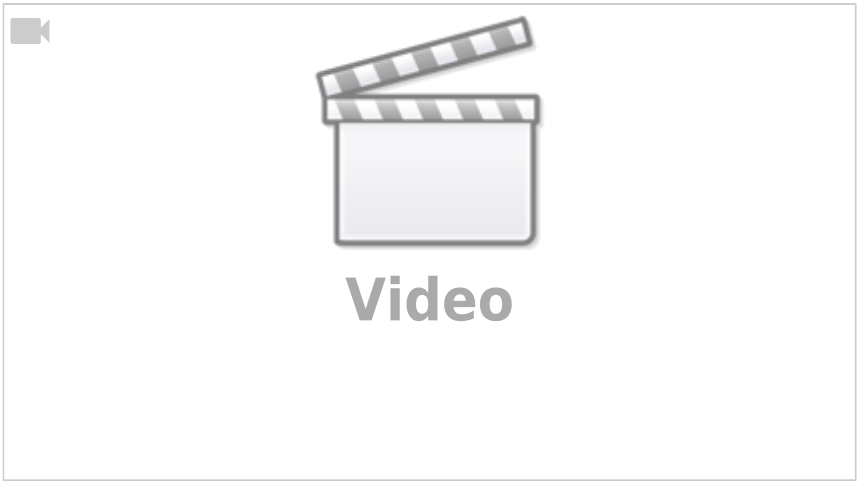
- **Sensor Calibration Issues:** The analog soil moisture sensor is sensitive to soil type, compaction, and mineral content. This makes it difficult to define a universal threshold for “dry” soil, and may require manual calibration for different plants or growing media.
- **Inconsistent Sensor Readings:** Over time, the captivated soil moisture sensor can degrade, oxidize, or produce noisy values depending on environmental conditions. Frequent recalibration or sensor replacement may be needed.
- **Limited Power Supply:** The system currently uses a 9V battery, which is not ideal for long-term operation. The battery drains quickly, especially when the pump is active, and would need

to be replaced or recharged often. A better alternative would be using a rechargeable battery with solar charging or USB power. To make sure it works efficiently a PV-Module would be great so it can be left in a greenhouse for example. You probably will not have outlets to power the system in a realistic setting.

- **Single Plant Design:** The current setup irrigates only one plant at a time. Extending it to multiple pots would require multiple sensors, more complex tubing, and pump control logic. The system can be scaled up relatively quickly due to the fact that everything is processed on node-RED.
- **No Local Display or Manual Control:** The system relies on cloud-based logic (via Node-RED) and MQTT communication. There is no physical interface for local control or status indication on the device itself (e.g., no screen or buttons).
- **Wi-Fi Dependency:** The system requires stable Wi-Fi to function properly. In areas with poor connectivity, MQTT communication and cloud logging (e.g., to Google Sheets) may fail, disrupting automatic watering and notifications.
- **Lack of Enclosure or Waterproofing:** The electronics are currently exposed on a breadboard. In a real-world indoor or greenhouse setup, a proper waterproof enclosure would be necessary for safety and durability.

6. Media and Demonstration

By Elham Mohammadi & Paul-Christian Thoma & Deniz-Zeynep Adem



7. References

By Elham Mohammadi & Paul-Christian Thoma & Deniz-Zeynep Adem

- Rui Santos. ESP32 Servo Motor Web Server Arduino IDE. Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/
- DFRobot. Gravity: Digital Peristaltic Pump (DFR0523). https://wiki.dfrobot.com/Gravity__Digital_Peristaltic_Pump_SKU__DFR0523
- MegaNano. Breadboard Power Module. https://meganano.uno/breadboard-power-module/

- DFRobot Store Page for DFR0523: https://www.dfrobot.com/product-1698.html
- AZ-Delivery Electronics Products. https://www.az-delivery.de/en/collections/alle-produkte

From:
https://student-wiki.eolab.de/ - **HSRW EOLab Students Wiki**

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2025:group-f:start&rev=1753728219

Last update: **2025/07/28 20:43**