Internet of Things Course Project

June, 2025, Bragança, Portugal

Anghelina Martiniuc(m321906), Beatriz Nahas(m320820)

1. Abstract

This project presents the simulation of an Internet of Things-based system to monitor and control chemical liquid levels in industrial laundry machines. While these machines often meet production needs, they typically lack modern automation and efficient maintenance tools. To address this, a prototype was created to simulate an automated refill system with remote control capabilities. The system uses an ESP8266 microcontroller to read data from a potentiometer (simulating liquid level) and control a LED (representing the refill pump). Data is transmitted via MQTT to Node-RED, which processes and forwards it to an InfluxDB database optimized for time-series data. Node-RED also provides a dashboard for real-time monitoring and manual refill commands. Core functions in the ESP8266 manage Wi-Fi and MQTT connectivity, automatic refill logic, and manual override based on user input. The system simulates refill events based on thresholds and allows remote interaction through a simple interface. InfluxDB enables the creation of visual dashboards to analyze historical trends, such as average, minimum, and maximum liquid levels over time. Although the solution is tested in a simulated environment, it offers a practical model for integrating similar systems into real industrial machines. The project demonstrates how affordable IoT tools and open-source platforms can improve machine maintenance through real-time data, remote access, and automation.

Key-words: Internet of Things, Remote control, Industrial laundry machine, Data analysis, Device simulation.

2. Introduction

This project aims to address a machine-related problem in the field of industrial cleaning equipment, with a specific focus on large-scale laundry machines. Despite technological advancements in recent years, some industrial sectors have remained outdated. These machines often meet production demands but suffer from poor maintainability.

To address part of this problem, the project proposes the development of a monitoring device capable of overseeing the chemical liquid levels used by the

machine's motor, automatically and manually refilling it as needed, and sending the collected data to a database to facilitate easier analysis and decision-making.

The entire development will be based on a fully simulated environment, without integration with a real industrial machine. Instead, alternative sensors and simulated inputs will be used to replicate typical machine conditions and behaviors, allowing for functional testing and validation of the proposed solution.

3. Development

In the following section, it will develop all the aspects that contain the Internet of Things device development process, from its physical and logical architecture to how the data generated will be shown to the user.

3.1. Physical architecture

The device will be composed of a central component, the ESP8266 microcontroller, which will be responsible for sending and receiving data. To simulate the operating environment, two supporting components will be used: a potentiometer, which will simulate the soap level sensor, and a led to simulate a proper motor, which will represent the chemical liquid pump responsible for refilling the tank. Figure 1 illustrates the physical architecture of the system, highlighting the ESP8266 as the central controller, along with the potentiometer and the led.

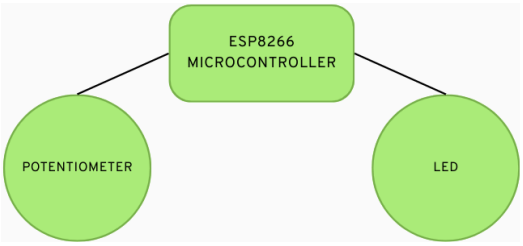

Figure 1: Physical architecture of the prototype.

The diagram illustrates the core components of the simulated device. The ESP8266 microcontroller serves as the central unit, responsible for processing data and controlling the system. It receives simulated soap level input from a potentiometer and acionate the led that represents the chemical pump responsible for refilling the liquid container.

3.2. Software connections

To collect and store the data generated by the microcontroller, the system will use Node-RED as the central communication interface. The ESP8266 will send sensor data to Node-RED using the MQTT protocol, which provides lightweight and efficient real-time messaging. Once the data is received, Node-RED can process it and store it in an InfluxDB database, ideal for managing time-series data such as liquid levels and pump activity.

In addition to data collection, Node-RED also enables bidirectional communication. Control signals, such as commands to activate or deactivate the chemical pump, can be sent from Node-RED back to the ESP8266 using MQTT protocol, allowing for remote operation and dynamic system behavior based on live data or user input. Figure 2 illustrates how the communication will work between the physical device and the software.

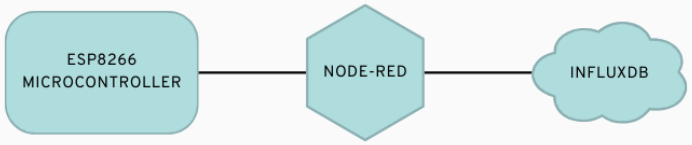

Figure 2: System communication architecture.

The diagram illustrates the data flow between the ESP8266 microcontroller, Node-RED, and the InfluxDB database. The ESP8266 sends data to Node-RED using the MQTT protocol, which allows for efficient and real-time communication. Node-RED acts as an intermediary, processing incoming data and forwarding it to the InfluxDB database for storage and future analysis.

3.3. Configuration and programing

To address the central issue of the machine, the code was developed and tested within a simulated environment. Figure 3 illustrates two key functions that manage network communication between the microcontroller and the MQTT broker[4].

```
void setup_wifi(){
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.print("\nIP Address: ");
  Serial.println(WiFi.localIP());
}
```

```
void reconnect(){
  while(!client.connected()){
    String clientId = "ESP8266Client-" + String(random(0xffff), HEX);
    if(client.connect(clientId.c_str())){
      Serial.println("MQTT connected");
      client.subscribe("IoT/ESP8266/Response");
    } else{
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Try again in 5 sec...");
      delay(5000);
    }
  }
}
```

Figure 3: setup_wifi() and reconnect() functions[4].

The setup_wifi() function connects the ESP8266 to the specified Wi-Fi network and prints the IP address once connected. The reconnect() function maintains the MQTT connection, retrying if disconnected and subscribing to the topic "IoT/ESP8266/Response" upon reconnection. This use of MQTT protocol provides an efficient and lightweight method for real-time communication between IoT devices [1], [2]. In Figure 4 is presented the function responsible for handling the MQTT topic income messages[4].

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

  if(String(topic) == "IoT/ESP8266/Response"){
    if(message == "true"){
      if(!autoRefillRequested){
        mqttRefillRequested = true;
        digitalWrite(led_pin, HIGH);
        Serial.println("MQTT refill started.");
      } else{
        Serial.println("MQTT refill ignored: Auto refill in progress.");
      }
    } else if(message == "false"){
      mqttRefillRequested = false;
      digitalWrite(led_pin, LOW);
      Serial.println("MQTT refill stopped.");
    }
  }
}
```

Figure 4: callback() function[4].

This function checks if the received message indicates a manual refill request. If so, and if an automatic refill is not already in progress, it triggers the manual refill by setting a flag and turning on the led. In Figure 5 the function checks whether the received MQTT message requests a manual refill. If the message is "true" and no automatic refill is currently active, it initiates the refill process by setting a flag and turning on the led indicator[4].

```
void readVolumeSensor(){
  int rawValue = analogRead(potentiometer);
  volumeML = map(rawValue, 0, 1023, 0, 100);
  Serial.print("Sensor volume: ");
  Serial.print(volumeML);
  Serial.println(" ml");

  if(volumeML <= 25 && !mqttRefillRequested && !autoRefillRequested){
    autoRefillRequested = true;
    digitalWrite(led_pin, HIGH);
    Serial.println("Auto refill triggered due to low volume.");
  }
  publishVolume();
}
void gradualRefill(){
  if(millis() - lastRefillTime >= refillInterval){
    lastRefillTime = millis();

    if(volumeML < 100){
      volumeML += refillStep;
      if(volumeML > 100) volumeML = 100;

      Serial.print("Refilling... ");
      Serial.print(volumeML);
      Serial.println(" ml");

      publishVolume();
    } else{
      Serial.println("Refill complete.");

      if(mqttRefillRequested) mqttRefillRequested = false;
      if(autoRefillRequested) autoRefillRequested = false;

      digitalWrite(led_pin, LOW);
      refillCooldownTime = millis();
    }
  }
}
```

Figure 5: readVolumeSensor() and gradualRefill() functions[4].

These functions handle soap volume monitoring and automatic refilling. readVolumeSensor() reads the volume level from a potentiometer and triggers an automatic refill if the volume drops below a threshold. gradualRefill() gradually increases the volume in steps until the maximum level is reached, simulating a refill process while updating the status and volume data. In Figure 6 the loop() function continuously checks and maintains the MQTT connection, ensuring the device stays connected to the broker[4].

```
void loop(){
  if(!client.connected()){
    reconnect();
  }
  client.loop();
  unsigned long now = millis();

  // Read sensor if not refilling and cooldown passed
  if((now - lastSensorRead >= sensorReadInterval) &&
     !mqttRefillRequested && !autoRefillRequested &&
     (now - refillCooldownTime >= cooldownDuration)){
    lastSensorRead = now;
    readVolumeSensor();
  }
  // Handle refill
  if(mqttRefillRequested || autoRefillRequested){
    gradualRefill();
  }
}
```

Figure 6: loop() function[4].

It periodically reads the soap volume sensor, provided no refill is in progress and the cooldown period has passed. If a refill is requested, it calls the gradualRefill() function to handle the refill process step by step.

As the main features are established in the microcontroller, and also how it will communicate with the Node-RED. FIgure 7 presents the Node-RED flow responsible for managing the communication between the ESP8266 and the database, as well as enabling user interaction through a dashboard[4].

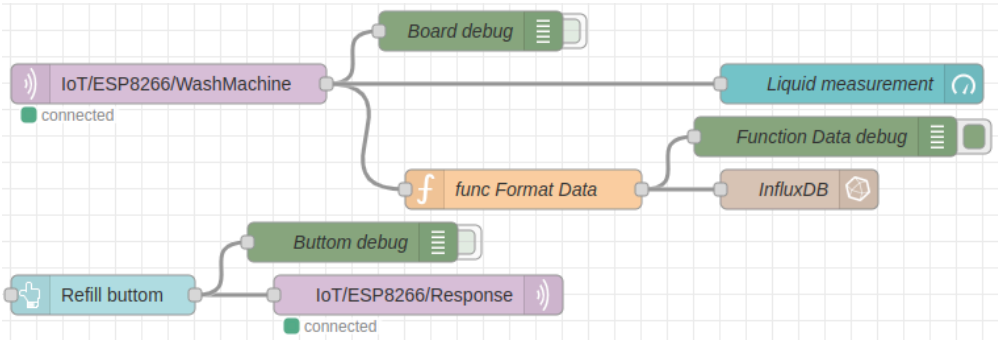

Figure 7: Node-RED flow[4].

This flow integrates the MQTT protocol for real-time device communication [1], InfluxDB for efficient time-series data storage [3], and the Node-RED Dashboard for real-time visualization and manual control. Together, these technologies create a responsive and extensible framework for monitoring and managing the simulated system remotely.

3.4. Data display

In the Node-RED dashboard, the user is able to visualize in real-time the soap volume in milliliters and manually refill the machine if necessary. In Figure 8 we are able to visualize the Node-RED dashboard[4].

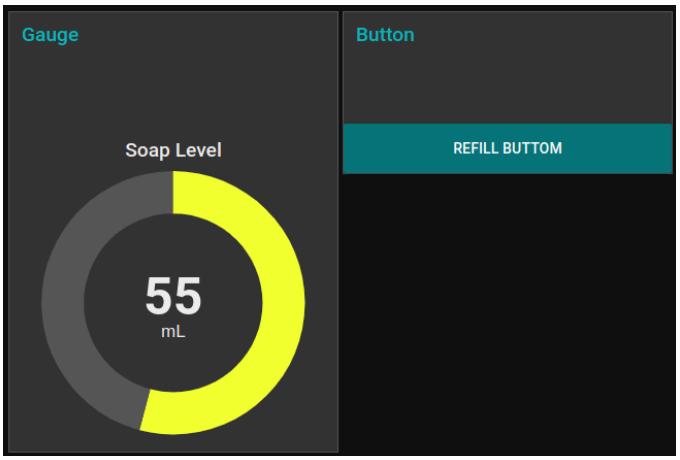

Figure 8: Node-RED control dashboard[4].

The dashboard provides real-time monitoring and control of the soap system. It features a gauge that displays the current soap volume, updated live via MQTT messages from the ESP8266. A refill button allows users to manually trigger the refill process by sending a command to the microcontroller.

Inside the InfluxDB environment, we can create dashboards to better visualize and make a complete analysis about the received data. Figure 9 exposes the dashboard created inside the database.

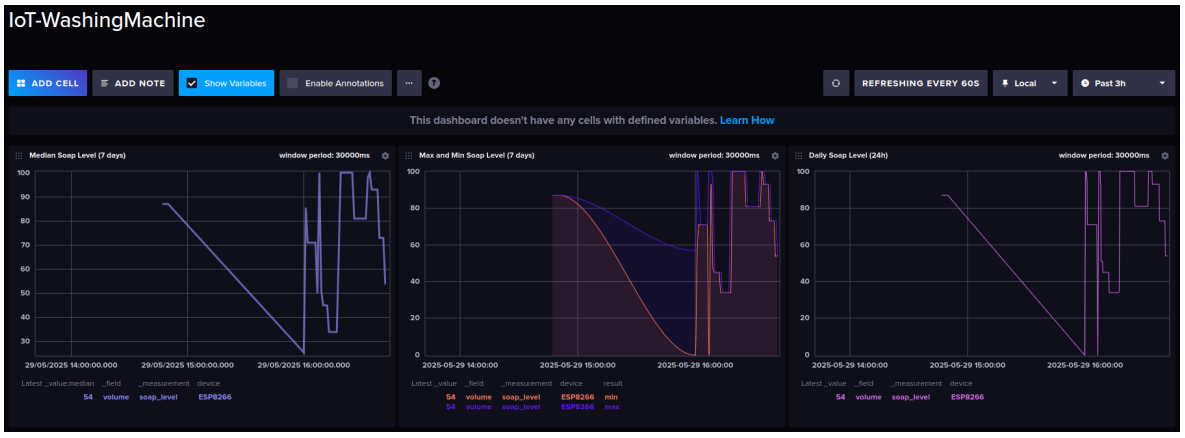

Figure 9: InfluxDB visualization dashboard.

In the dashboard, we can visualize three types of organized data: the median value of the soap volume from the past seven days, the maximum and minimum

value of the soap volume from the past seven days, and the soap volume from the last 24 hours.

4. Conclusion

This project successfully demonstrates the development of an IoT-based monitoring and control system for simulating the chemical liquid refill process in industrial laundry machines. Using the ESP8266 microcontroller, MQTT protocol, Node-RED, and InfluxDB, the system provides real-time monitoring, automatic and manual refilling, and efficient data visualization. Although tested in a simulated environment, the project lays a solid foundation for future integration with real industrial equipment, aiming to improve operational efficiency and maintenance management through smart automation.

5. References

[1] HiveMQ, “MQTT on Arduino with NodeMCU ESP8266 and HiveMQ Cloud”, HiveMQ Blog, Feb. 16, 2021. [Online]. Available: https://www.hivemq.com/blog/mqtt-on-arduino-nodemcu-esp8266-hivemq-cloud/.

[2] P. Macheso, T. D. Manda, S. Chisale, N. Dzupire, J. Mlatho, and D. Mukanyiligira, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”, IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9396027.

[3] InfluxData, “IoT made easy with Node-RED and InfluxDB”, InfluxData Blog, Aug. 4, 2020. [Online]. Available: https://www.influxdata.com/blog/iot-easy-node-red-influxdb/.

[4] A. Martiniuc, B. Nahas. “IoT-FinalProject-IPB”. Github Repository, May 15, 2025. [Online]. Available: https://github.com/nahasBeatriz/IoT-FinalProject-IPB/tree/main.