Dashboard Communication Through MQTT

Nitin Lala, Tigo Goes, Toine Steehouwer and Thomas van Vliet

***Abstract*—This paper discusses the development of an omniwheel bot that can drive autonomously or manually using a MQTT-based dashboard for real-time communication and control, and is able to map its enviroment on a map while doing so. The system was designed with IoT applications in mind, offering dynamic data updates, low-latency interactions, and secure connections. We address challenges such as system scalability, reliability in data transmission, and the efficient visualization of real-time information. The results demonstrate the ability of this system to manage and visualize data, with the flexibility for future enhancements and improvements in its architecture.**

***Index Terms*—omniwheels, environment mapping, dashboard, MQTT, communication, control, IoT, real-time, security**

I. Introduction

For this TinLab project a project was designed to create a robot which can autonomously drive around and map its environment. The technology used for such a bot could be used in a wide assoertment of applications, such as in warehouses for the transport of goods, or for bots which can be used in a home environment to help with chores.

II. Preliminary investigation

III. research questions

Main question: How can visual mapping be implemnted on an omniwheel driven robot to map an environment? Subquestions:

- How can the real-time sensor values of the ultrasonic sensors be used to map the environment?
- How can the map the location data be displayed remotely to the user in a clear way?

IV. Opdrachtomschrijving

MQTT Broker: The system utilizes the Mosquitto broker for local testing and EMQX for scalability. Mosquitto is installed via the ‘sudo apt-get install mosquitto‘ command and is used to facilitate communication between the bot and the dashboard with low-latency message exchanges. For production-level scalability, EMQX ensures the broker can handle a larger number of devices and concurrent users.

Frontend Dashboard: The dashboard is built using React.js, with dynamic data visualization provided by Grafana. The dashboard subscribes to MQTT topics through MQTT.js, allowing for real-time updates on bot status, including metrics like speed, battery, and system diagnostics. The user interface includes interactive controls for bot movement and emergency stop functionalities. Additionally, system diagnostics and event logging are provided to help users track the bot’s behavior and troubleshoot any issues that arise.

A. Problem definition

B. Requirements

C. Quality risk analysis

V. Method

A. Movement of the explorer bot

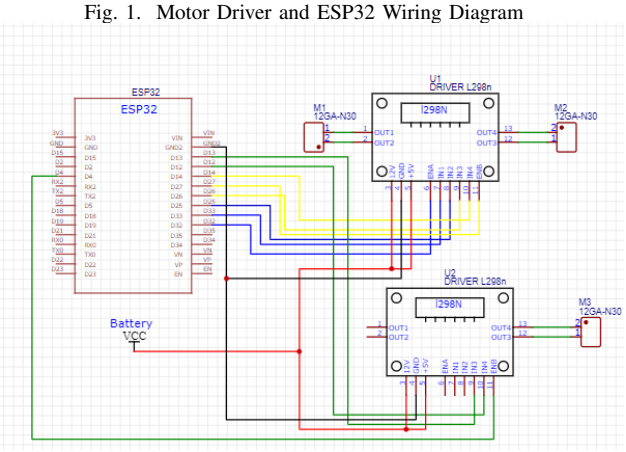
The explorer bot is designed to navigate autonomously in an indoor environment, utilizing omniwheels for movement. To evaluate the performance of omniwheels in robotic movement, a test platform was designed and constructed. The platform consists of the following components:

- **Omniwheels**: Three omniwheels arranged in a triangular configuration to enable omnidirectional movement.
- **Motors & Drivers**: Each wheel is powered by a DC motor, controlled via two **L298N (HW-95) motor drivers**, ensuring precise speed and direction control.
- **Microcontrollers & Communication**:
  - An **ESP32 UWB** serves as the **master controller**, responsible for high-level motion commands and data collection.
  - A secondary **ESP32** acts as a **slave controller**, handling motor control and executing movement instructions in real time.
- **Power Supply**: Two different battery packs. One is for the master controller and the other is for the slave controller and motors.

B. Hardware Design

The hardware design consists of the integration of omniwheels, motors, motor drivers, ESP32 microcontrollers, and a dual power supply system. The master ESP32 UWB handles high-level motion commands and data processing, while the slave ESP32 executes real-time motor control. The wiring connections between the ESP32 boards, motor drivers, and power supply are illustrated in Figure 1.

Fig. 1. Motor Driver and ESP32 Wiring Diagram



C. The display

The explorer bot uses a lcd display to show basic information to the user. The chosen display makes use of the i2c protocol to communicate with the microcontroller (I2c allows multiple devices to be connected to the same two wires which simplifies connections). The display was connected to an ESP32 microcontroller to test it in the same way it will be used in the final product. These connections are found in the electric schematic in the appendix. For testing purposes a test program was created which shows a text on the display with an number behind it which increments once every 500 miliseconds. To test the display the test code is uploaded to the ESP32 and the display was monitored until the number on the display reached to 10 to see if the correct text is displayed. Following the above procedure the display was tested and found to be working.

D. The gyroscope sensor

The explorer bot will need to know its orientation to correctly interpret sensor data (–see other subsection–). For this reason an i2c gyroscope sensor was chosen. The gyroscope was connected to the ESP32 microcontroller according to the electric schematic in the appendix (TODO) and testcode was written which prints the sensor data in the terminal. This test code was uploaded to the ESP32 and the sensor was rotated on the x, y and z axis to see if the sensor data changes accordingly. The sensor was found to be working correctly. Later the gyroscope sensor was replaced by a barometer sensor, See

The robot uses ultrasonic sensors to measure the distance to the nearest object. Using these values, we can draw lines on the map to define the room and obstacles. During testing, we encountered an issue we had not initially accounted for: when the robot rotates, the sensors rotate with it. This means we need to track the robot's rotation relative to its starting position. Using this rotation, we can adjust the sensor readings accordingly. For example, if the robot has turned 30 degrees, the measured distances must be transformed to match the new orientation. To perform these calculations, we use the Pythagorean theorem along with trigonometric functions to accurately determine the position of obstacles. The Pythagorean theorem is particularly useful for recalculating distances when translating between coordinate systems after rotation. We started testing by first verifying the accuracy of the ultrasonic sensors. This was done by placing an object at a predefined distance from the sensor and comparing the measured values with the expected results. We printed the raw sensor data to confirm the measurements were correct. Once we validated the sensor readings, we focused on determining the robot's rotation angle. Initially, we considered using a gyroscope, as it seemed like a suitable solution. However, after extensive testing, we discovered that the gyroscope primarily measures angular velocity around the X and Y axes rather than absolute rotation around the Z-axis. Despite this limitation, we used the gyroscope to test real-time recalculations of sensor data. By manually rotating the sensor and observing the calculated angle, raw sensor data, and recalculated distances, we verified the correctness of our approach. Once this was confirmed, we replaced the gyroscope with a magnetometer, which provided direct measurements of the robot's heading.

VI. Results

Extensive tests confirm the reliability and performance of the system. Real-time data transmission is achieved with an average latency of approximately 50ms, which is suitable for controlling devices in real-time applications. The system is capable of handling up to 100 simultaneous connections without any noticeable performance degradation. Stress testing demonstrates the system's ability to maintain message delivery with less than 0.5% message loss when optimized with QoS settings. Additionally, the dashboard remains responsive and stable, with updates occurring at a smooth 30 frames per second (FPS), even under heavy data load. In the event of network disruptions, the MQTT system's persistent sessions and QoS 2 delivery mechanism ensure that messages are reliably delivered once the connection is restored.

VII. Conclusion and Advice

A. Dashboard Communication Through MQTT

The MQTT-based dashboard provides an efficient and scalable solution for controlling and visualizing an omniwheel bot in real-time. The system demonstrates excellent performance in terms of scalability, reliability, and data security. Looking forward, future improvements

will focus on incorporating AI-driven analytics to predict bot behavior, enhancing security features to address potential vulnerabilities, and integrating live video streaming through WebRTC for real-time visualization of bot movements. The architecture of the system is also designed with scalability in mind, allowing for the integration of additional devices, making it suitable for industrial IoT applications and multi-bot environments.