UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Dipartimento di Ingegneria e Scienze
dell'Informazione e Matematica

Università degli Studi dell'Aquila

Automated Gas Monitoring System

Department of Information Engineering, Computer Science and Mathematics

University Of L'Aquila, Italy

Professor Davide Di Ruscio

Software Engineering for Autonomous Systems

Members
Sergo Kashuashvili
Adam Bouafia

Contents

Introduction

The ESP32 Sensor Monitoring and MQTT Communication project aim to create a comprehensive system for monitoring environmental conditions, gas levels, and motion using an ESP32 microcontroller. The project leverages various sensors, including DHT22 for temperature and humidity, ultrasonic for distance measurement, gas sensor for gas levels, and a motion sensor for motion detection. The collected data is then transmitted to a remote MQTT broker for further analysis and monitoring.

Objectives

- Real-time monitoring of environmental conditions.
- Gas level measurement for safety and environmental awareness.
- Motion detection for security and automation.
- Integration with MQTT for remote data transmission and monitoring.
- Utilizing MAPE-K Framework for effective handling of emergency situations.

1. Requirements

a. Functional Requirements

The functional requirements for the gas monitoring system include:

Functional Requirements of the Gas Monitoring System		
Identifier	**Name**	**Description**
GM-FR001	Data Generation	The system must collect real-time data from gas sensors, including gas concentration levels, and provide continuous updates within a specified range.
GM-FR002	Data Collection	The system should process and store the gas concentration data, allowing for historical analysis and trend monitoring.

GM-FR003	Monitoring Interface	The system should feature a user interface for monitoring environmental conditions, displaying real-time gas concentration levels, and providing alerts for critical levels.
GM-FR004	Taking Action	Apart from alerts, the system should have functionality to take action in case of an emergency.
Non-Functional Requirements of the Gas Monitoring System		
Identifier	Name	Description
GM-NFR001	Scalability	The system should be designed to accommodate the addition of more gas sensors, ensuring scalability to handle an increasing number of monitored areas and diverse gas sources.
GM-NFR002	Reliability	The system should consistently provide accurate gas concentration readings, ensuring reliable detection and timely response to any variations or anomalies.
GM-NFR003	Monitoring Interface	The system's monitoring interface should maintain high availability, allowing users to access real-time data and receive alerts without significant downtime during operation.

2. Component Description

a. Managed System

The managed system consists of an ESP32 microcontroller, DHT22 sensor, ultrasonic sensor, gas sensor, motion sensor, and a stepper motor (AccelStepper). The ESP32 acts as the central controller, managing sensor readings and communication with the MQTT broker.

b. Architecture Diagram (MAPE-K)

The project utilizes the MAPE-K loop, incorporating advanced tools like InfluxDB, Grafana, and Telegram via Node-RED for comprehensive environmental management. Monitoring is done through sensors, capturing real-time data on temperature, humidity, and gas levels. Analysis involves assessing this data against set thresholds in Node-RED. Planning includes

determining actions like ventilation, controlling the valves coupled with alerts based on this analysis. Execution is carried out by stepper motors for ventilation and alerts via Telegram. Knowledge consists of the system's predefined thresholds and is enhanced by historical data stored in InfluxDB, visualized through Grafana panels, ensuring a dynamic and responsive environmental control system. We elaborate on technologies used in the next section.

3. Technologies

Programming Language: C++ (Arduino)

The project is implemented using the C++ programming language, specifically tailored for the Arduino platform. Arduino's simplified syntax and extensive library support make it an ideal choice for microcontroller-based projects. C++ allows efficient and low-level control over hardware, making it well-suited for programming the ESP32 microcontroller in this context.

Communication Protocol: MQTT

MQTT (Message Queuing Telemetry Transport) is employed as the communication protocol. MQTT is a lightweight and efficient messaging protocol designed for lowbandwidth, highlatency, or unreliable networks. It operates on a publish-subscribe model, facilitating the exchange of messages between devices. In this project, MQTT is utilized for transmitting sensor data from the ESP32 to a remote broker, enabling real-time monitoring and control.

Sensor Libraries: DHTesp, AccelStepper DHTesp

Library:

The DHTesp library is utilized for interfacing with the DHT22 sensor, which measures temperature and humidity. This library provides convenient functions for initializing the sensor, reading data, and handling errors. It ensures accurate and reliable data retrieval from the DHT22 sensor.

AccelStepper Library:

The AccelStepper library is included for stepper motor control, although the provided code does not explicitly utilize it. This library simplifies the management of stepper motors, offering features like acceleration and deceleration profiles, which can be beneficial in future expansions or modifications to the project. d. Sensor Interfaces: I2C, GPIO I2C (InterIntegrated Circuit):

I2C is a serial communication protocol used for connecting multiple devices on the same bus. It is employed in this project to interface with sensors that support I2C communication, enhancing the flexibility and scalability of the system.

GPIO (General Purpose Input/Output):

GPIO pins are used to interface with various sensors and actuators. Digital pins are configured as either input or output based on the requirements of each sensor. For instance, motion sensor input, gas sensor input, and ultrasonic sensor trigger and echo pins are managed using GPIO.

Database: InfluxDB

InfluxDB is a high-performance, distributed, and scalable open-source time-series database. It is designed to handle large volumes of time-stamped data efficiently.

Role in the System: InfluxDB is used to store time-series data generated by the simulation, such as blinds position, light intensity, and movement count. This data can be queried for analysis and visualization.

Alerts: TelegramBot

The Telegram bot in Node-RED is facilitated by the node-red-contrib-telegrambot module, which is installed as part of the Node-RED setup.

This integration allows Node-RED to interact with the Telegram Bot API, enabling automated messaging and interaction capabilities within Telegram chats .

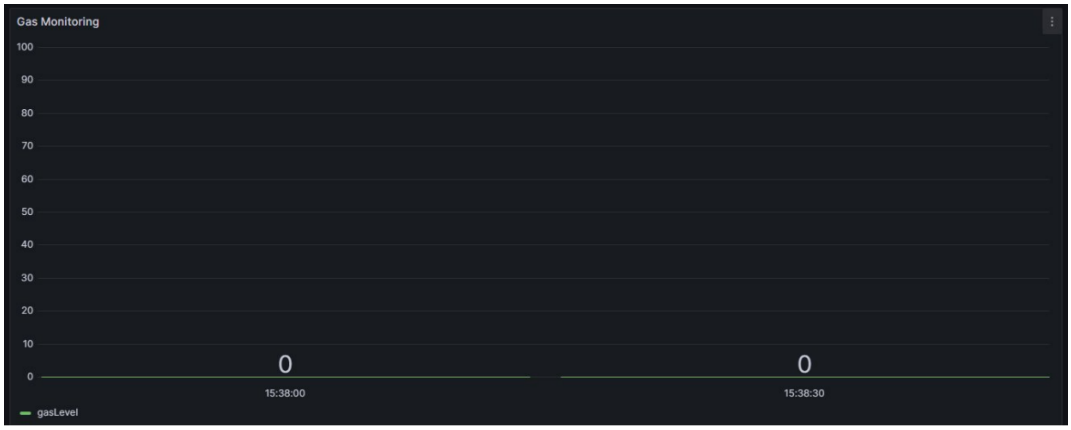
Containerization: Docker

Docker is a containerization platform that allows applications to be packaged with their dependencies into lightweight, portable containers. Containers ensure consistency across different environments and simplify deployment.

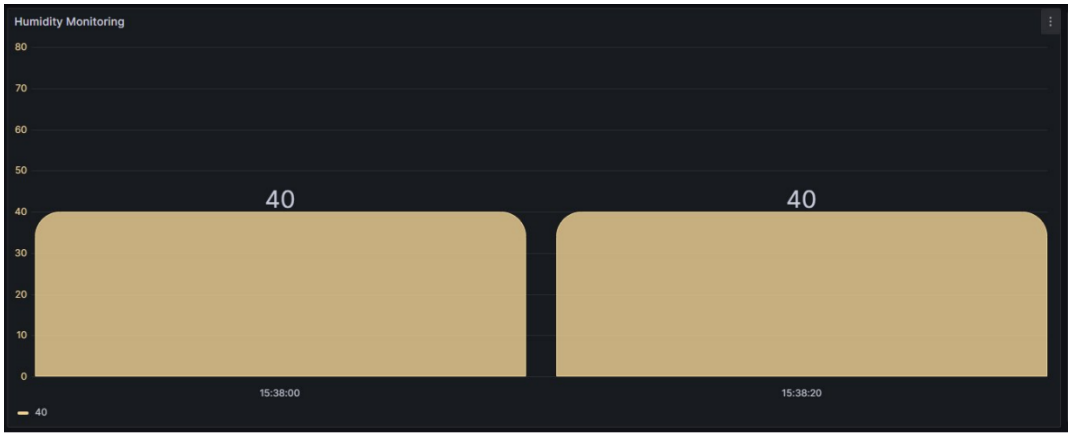
Role in the System: Docker is employed to containerize the entire Automated Blinds System, including Python scripts, dependencies, and services. It streamlines deployment, making the system easy to run on various platforms without compatibility issues.

We utilized the MQTT messaging protocols to allow for data communications between the system and InfluxDB. The main instrument for this task was Grafana, which is a graphical interface and an interaction system tool. Here are the Grafana panels for various sensors:

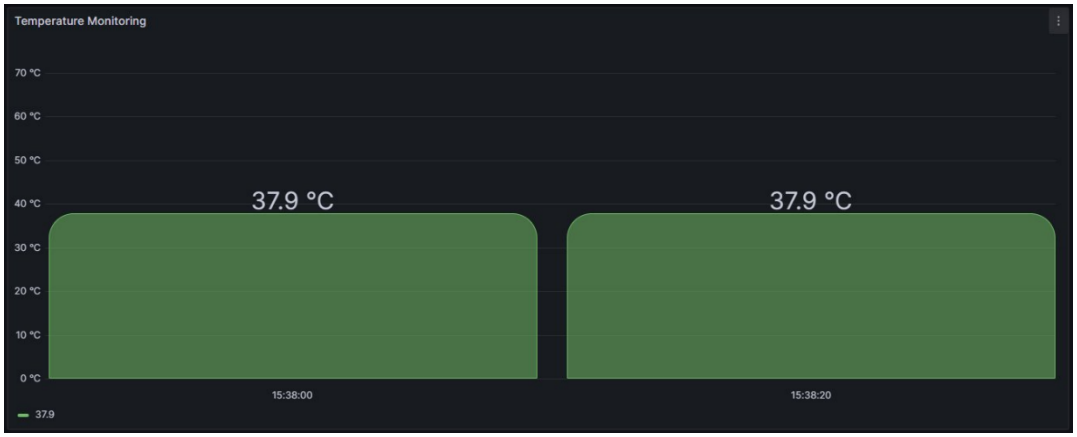
Gas Sensor:



Humidity Sensor:



Temperature Sensor:



Tank Capacity Sensor:

4. System Architecture

The system architecture for the Gas Monitoring Project is designed to efficiently integrate various sensors, process data, and establish communication with an MQTT broker. The key components and their interactions are described below:

Components:

1. **ESP32 Microcontroller:**
    a. Serves as the central processing unit, responsible for managing sensor data and communication.
    b. Interfaces with gas sensors, DHT22 for temperature and humidity, ultrasonic sensor for distance measurement, and a motion sensor.
    c. Utilizes GPIO pins for sensor connections and I2C for sensors that support this communication protocol.
2. **MQTT Communication Module:**
    a. Manages the communication between the ESP32 and the remote MQTT broker.
    b. Utilizes the EspMQTTClient library to establish and maintain a connection with the MQTT broker.
    c. Publishes gas concentration data, temperature, humidity, and motion events to specific MQTT topics for remote monitoring.
3. **Sensor Modules:**
    a. Gas Sensor: Measures gas concentration levels in the environment.
    b. DHT22 Sensor: Captures temperature and humidity data.
    c. Ultrasonic Sensor: Measures distance for tank capacity calculation.
    d. Motion Sensor: Detects motion events.
4. **Data Processing:**
    a. The ESP32 processes sensor readings, calculates tank capacity based on ultrasonic sensor data, and prepares the data for transmission.

b. Utilizes the DHTesp library for accurate management of DHT22 sensor data.

c. Converts analog gas sensor readings to a percentage scale for better interpretability.

5. System Functionality

1. **Sensor Readings:**
   a. The ESP32 triggers the sensors (gas, DHT22, ultrasonic, motion) to obtain real-time environmental data.
   b. Data is processed locally to derive relevant information, such as tank capacity and gas concentration.
2. **MQTT Communication:**
   a. The ESP32 establishes a connection to the MQTT broker using the EspMQTTClient library.
   b. Sensor data (temperature, humidity, gas concentration, tank capacity) is published to specific MQTT topics, such as "univaq/gas."
3. **Monitoring Interface:**
   a. A remote monitoring interface (not explicitly detailed in the provided code) subscribes to MQTT topics to receive real-time updates.
   b. Users can access the monitoring interface to visualize current environmental conditions, including gas levels, temperature, and humidity.
4. **Interrupt Handling (Motion Sensor):**
   a. The motion sensor triggers an interrupt when motion is detected, leading to the execution of the detectsMovement function. This event is reported to the serial monitor.
5. **Emergency Action:**
   a. Fan Control:
      i. activateFan: Turns on the fan by setting the pin (FAN_RELAY_PIN) to HIGH, activating the relay and consequently the fan.
      ii. deactivateFan: Turns off the fan by setting the pin to LOW, deactivating the relay and stopping the fan.
      iii. These functions control the state of the fan by manipulating the relay connected to the designated pin. They provide a way to activate and deactivate the fan based on system requirements, such as environmental conditions or user commands.
   b. Valve Control:

   i. openGasValve: Opens the gas valve by setting the pin (VALVE_RELAY_PIN) to HIGH, activating the relay and allowing gas flow.
   ii. closeGasValveCommand: Closes the gas valve by setting the pin to LOW, deactivating the relay and shutting off the gas flow.
6. activateVentilation: Publishes a JSON-formatted message to the "univaq/actuatorCommands" MQTT topic, commanding the system to activate ventilation.
7. closeGasValve: Publishes a JSON-formatted message to the same MQTT topic, commanding the system to close the gas valve.
8. These functions handle the control of the gas valve, allowing for both manual control (openGasValve and closeGasValveCommand) and remote control through MQTT messages (activateVentilation and closeGasValve). They provide a means to ensure gas safety within the system by allowing for controlled gas flow and emergency shutdown.

Conclusion

The ESP32 Sensor Monitoring and MQTT Communication project successfully achieve its objectives by creating a robust system for real-time monitoring and remote communication. The integration of various sensors enhances its versatility, making it applicable in environmental monitoring, security, and automation applications. The adherence to MAPE-K architecture ensures efficient data flow and processing, contributing to the overall success of the project.