Smart Gas Leakage Monitoring System

Advanced Embedded System Lab

Summer Term 2023

1st Shihab Ud Doula *dept. Electronic Engineering*
Hochschule Hamm-Lippstadt
Lippstadt, Germany
shihab-ud.doula@stud.hshl.de
and 2nd Neaz Mahmud *dept. Electronic Engineering*
Hochschule Hamm-Lippstadt
Lippstadt, Germany
Neaz.mahmud@stud.hshl.de
and 3rd Younsuk Choi *dept. Electronic Engineering*
Hochschule Hamm-Lippstadt
Lippstadt, Germany
younsuk.choi@stud.hshl.de

I. Introduction

The term ”smart devices” encompasses the Internet-connected devices commonly utilized in our daily lives. These devices enable the establishment of smart homes and facilitate remote management and control from any location. With the increasing adoption of Internet of Things (IoT) devices, there is a growing potential for their implementation in various contexts. However, this expansion also brings forth concerns for humanity, as the integration of wireless networks becomes more intricate yet standardized. In many developing and underdeveloped countries, gas leakage is a prevalent occurrence, where the detection of accumulated gas from pipelines or cylinders is often beyond the capabilities of ordinary individuals. Unfortunately, it is these individuals who bear the ultimate consequences. The invisible nature of gas renders it imperceptible to the naked eye, thereby impeding its identification and leading to unfortunate incidents and casualties annually. Addressing such challenges, we have developed the ”Smart Gas Leakage System,” an exceptional fusion of hardware and software. This system boasts advanced hardware components and incorporates sophisticated software solutions, all aimed at mitigating gas leaks worldwide.

II. Project Description

The objective of this project is to develop an IoT-based gas leakage monitoring system that can detect gas leaks and fires in real time. The system will send alerts to users when a gas leak or fire is detected, and it will also be able to activate the sprinkler in case of fire. The system could be used in homes to detect gas leaks from appliances such as stoves and ovens. The system could also be used in businesses to detect gas leaks from industrial equipment. Additionally, the system could be used in public areas such as schools and hospitals to detect gas leaks that could pose a safety hazard.

The system will work by using a gas sensor to detect gas leaks and flame sensor to detect fire. The gas sensor and the flame sensor will be connected to an Arduino board, which will send data to the Raspberry Pi. The Raspberry Pi will act as a MQTT broker, which will forward the data to the MQTT client app or Adafruit IO. The MQTT client app or Adafruit IO will then send an alert to the user.

The system will also contain actuators such as Piezo Buzzer and will be connected to sprinkler mechanism. Upon detecting abnormal level of gas, the piezo buzzer will be turned on and whenever fire is detected by the system, the sprinkler mechanism will be activated.

This system will help to prevent gas leaks and fires, which can lead to serious injuries or death. The system will also provide peace of mind to users, knowing that they are alerted to gas leaks and fires in real time. Additionally, the system can help to save money by preventing damage to property and equipment.

A Block diagram to describe the overview of the IoT system.

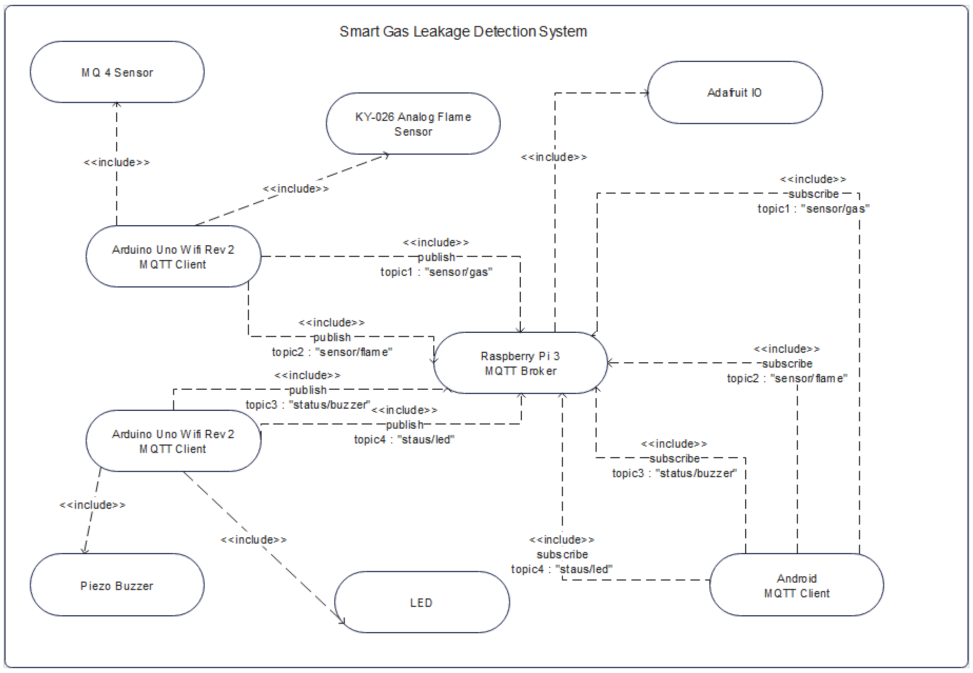

Smart Gas Leakage Detection System
MQ 4 Sensor
KY-026 Analog Flame Sensor
Adafuit IO
<<include>>
<<include>>
<<include>>
<<include>> subscribe topic1 : "sensor/gas"
Arduino Uno Wifi Rev 2 MQTT Client
<<include>> publish topic1 : "sensor/gas"
<<include>> publish topic2 : "sensor/flame"
Raspberry Pi 3 MQTT Broker
<<include>> subscribe topic2 : "sensor/flame"
<<include>> publish topic3 : "status/buzzer"
<<include>> publish topic4 : "staus/led"
Arduino Uno Wifi Rev 2 MQTT Client
<<include>> subscribe topic3 : "status/buzzer"
<<include>>
<<include>>
<<include>> subscribe topic4 : "staus/led"
Piezo Buzzer
LED
Android MQTT Client

Fig. 1. Block Diagram

Here are some specific scenarios in which the system could be used:

- A gas leak is detected in a home. The system turns on the piezo buzzer (alarm) and sends an alert to the user, who then takes appropriate action to shut off the gas supply and evacuate the home.
- A fire has been detected in a home. The system activated the sprinklers and sends an alert to the user, who then takes appropriate action to extinguish the fire and evacuate the home.

III. Project and Team Management

A. *Agile Method*

In the execution of our project, we employed the Agile methodology, a widely recognized and adopted approach for software development and project management. Agile methodologies emphasize iterative and incremental development, collaborative teamwork, adaptability, and continuous improvement.

By embracing Agile principles, our project team was able to navigate the complex landscape of IoT-based gas leakage monitoring system development effectively. The iterative nature of Agile allowed us to break down the project into manageable sub-parts, facilitating a step-by-step approach towards the ultimate goal. This iterative process enabled us to gather feedback, make necessary adjustments, and ensure that the evolving requirements were met throughout the project's life-cycle.

B. *Task/Goal Breakdown*

The implementation of the system was broken down into the following sub-parts:

Sensor Reading: The first stage focuses on sensor reading, where the gas sensor and flame sensor are implemented to detect gas leaks and fires, respectively. These sensors are connected to an Arduino board, which is responsible for reading data from the sensors.

Actuators: The next stage involves the integration of actuators. A Piezo Buzzer is incorporated as an actuator to generate an audible alarm when an abnormal level of gas is detected. Additionally, the sprinkler mechanism is connected as an actuator, and the system is programmed to activate it upon detecting a fire.

Communication Protocol: To enable communication and data transfer, a MQTT (Message Queuing Telemetry Transport) communication protocol is implemented. A Raspberry Pi is set up as an MQTT broker to facilitate communication. The Arduino board is configured to send sensor data to the MQTT broker, and an MQTT client app or Adafruit IO is configured to receive data from the broker. The app or Adafruit IO is programmed to send alerts to the user based on the received data.

Lastly, a mobile client app is developed to provide a user-friendly interface for receiving alerts. The MQTT client functionality is integrated into the app, allowing it to receive alerts and display them to the user. The app also enables the user to take appropriate actions in response to the alerts.

C. *Task Distribution*

Shihab Ud Doula: Project Idea, Use Case Diagram, Class Diagram, Sensor Reading from Arduino
Neaz Mahmud: Sensor Reading, Raspberry pi setup, Mqtt communication
Younsuk Choi: Block Diagram, Actuators, Adafruit IO

IV. TECHNOLOGIES

A. *Sensors*

- **MQ4 Methane Gas Detecting Sensor**: The MQ4 gas sensor utilizes a heated Tin Dioxide (SnO2) material to detect flammable gases like methane. When exposed to the target gas, the sensor's resistance changes, which is measured by an Arduino or microcontroller. Calibration and a calibration curve are typically used to convert the resistance readings into gas concentration values. This enables the system to monitor gas levels and trigger appropriate actions like sending alerts or activating safety measures.

Fig. 2. Mq4 sensor

- **Analog Flame sensor**: The analog flame sensor is a fire detection component that utilizes an infrared (IR) receiver and emitter. It works by emitting IR rays and detecting the infrared radiation emitted by flames. When a flame is present, the sensor translates the detected radiation into an analog voltage signal. This signal's magnitude correlates with the intensity of the flame. By comparing the analog signal to a predefined threshold, the sensor can determine if a flame is detected. The output from the sensor can be used to trigger actions like activating alarms, sending alerts, or initiating fire suppression systems. Proper placement and calibration are crucial to minimize false alarms and ensure accurate flame detection in fire safety applications.

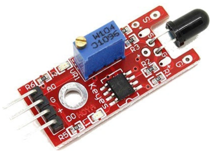


Fig. 3. Analog Flame Sensor

- **Piezo buzzer**: The piezo buzzer in the gas leakage monitoring system functions as an audible alarm. It operates based on the piezoelectric effect, where an electrical signal causes a piezoelectric material to vibrate, generating sound waves. When triggered by a gas leak or specific conditions, the buzzer emits a loud and distinctive sound, alerting users to the potential danger. Its compact size, low power consumption, and ease of integration make it an effective component for providing audible alerts in the system.

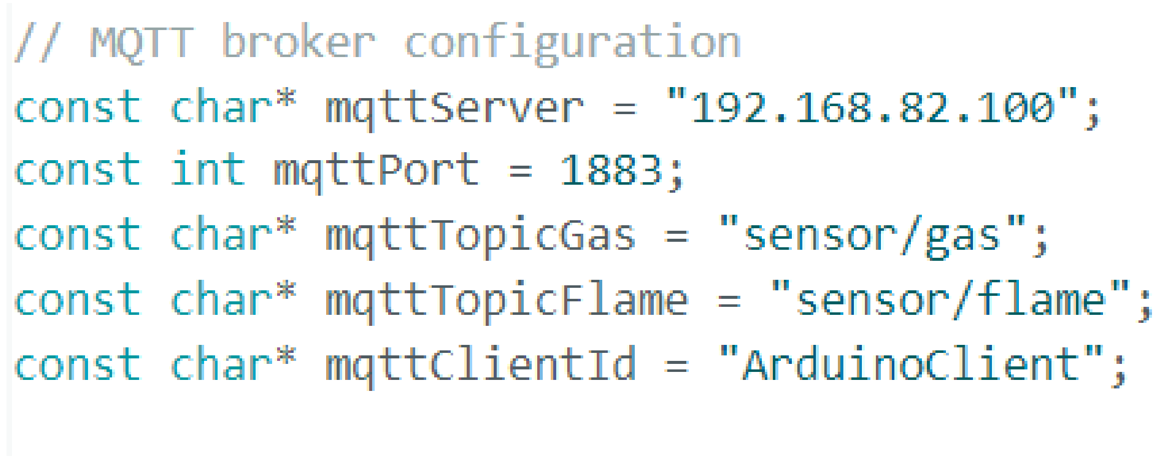
Fig. 4. Piezo Buzzer Sensor

- **LED**: The LED (Light-Emitting Diode) plays a crucial role in the gas leakage monitoring system. It serves as a visual indicator to provide important status information to users. The LED can be used to display various conditions such as system readiness, gas detection, and system activation. For example, the LED can be programmed to illuminate when the system is powered on and ready for operation. It can also blink or change color when a gas leak or fire is detected, providing immediate visual feedback to users. The LED enhances the system's user interface and assists in quickly identifying potential hazards.

Fig. 5. LED

B. *Communication Protocol*

The MQTT protocol is employed to establish communication between the gas and flame sensors connected to the Arduino board and the MQTT client app or Adafruit IO. This protocol ensures seamless and lightweight messaging, allowing the system to send real-time updates and alerts to the users. A Raspberry Pi is configured as an MQTT broker. It acts as a central hub, receiving data from the sensors via the Arduino board and forwarding it to the appropriate destination, i.e., the MQTT client app or Adafruit IO. The MQTT client app or Adafruit IO is configured to subscribe to specific topics on the MQTT broker, enabling it to receive the forwarded sensor data. This data contains information about gas leaks and fire detection from the respective sensors. The MQTT client app or Adafruit IO processes this data to generate alerts based on predefined criteria.



```
// MQTT broker configuration
const char* mqttServer = "192.168.82.100";
const int mqttPort = 1883;
const char* mqttTopicGas = "sensor/gas";
const char* mqttTopicFlame = "sensor/flame";
const char* mqttClientId = "ArduinoClient";
```

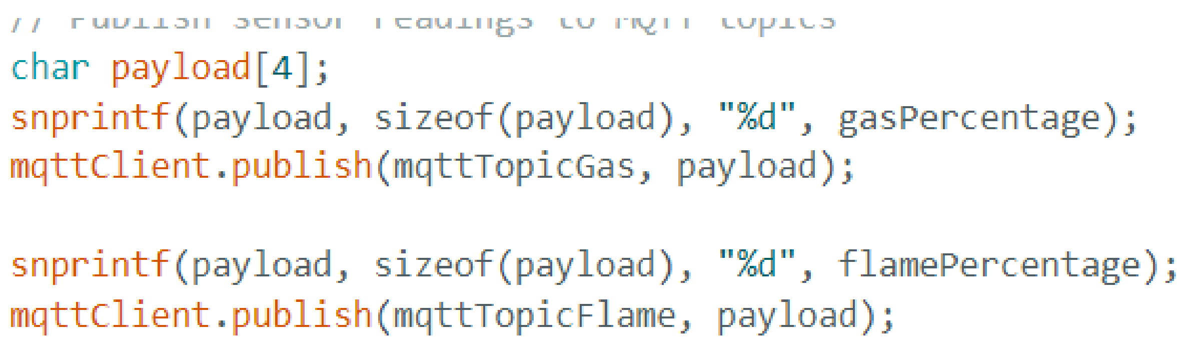
Fig. 6. Mqtt Configuration

C. *Programming language*

C++ was employed as the programming language to implement the project. The main functions within the code serve essential purposes, as described below in an academic style:

The **'setup()'** function, executed during the initialization phase of the Arduino board, performs crucial initializations. It initializes the serial communication interface, allowing for communication with external devices. Additionally, it establishes a connection with the specified Wi-Fi network using the provided credentials ('ssid' and 'password'), enabling subsequent communication over the network. The function also configures the MQTT server and port settings required for communication with the MQTT broker. Lastly, it initiates the I2C communication protocol, facilitating communication with compatible devices.

The **'loop()'** function represents the core operational logic of the program, executed repeatedly after the setup phase. It first checks if the MQTT client is currently connected to the MQTT broker. In the event of a disconnection, the function calls the 'reconnect()' routine to reestablish the connection. Subsequently, the 'loop()' function continuously runs the MQTT client loop to handle both incoming and outgoing messages, ensuring a seamless flow of data.



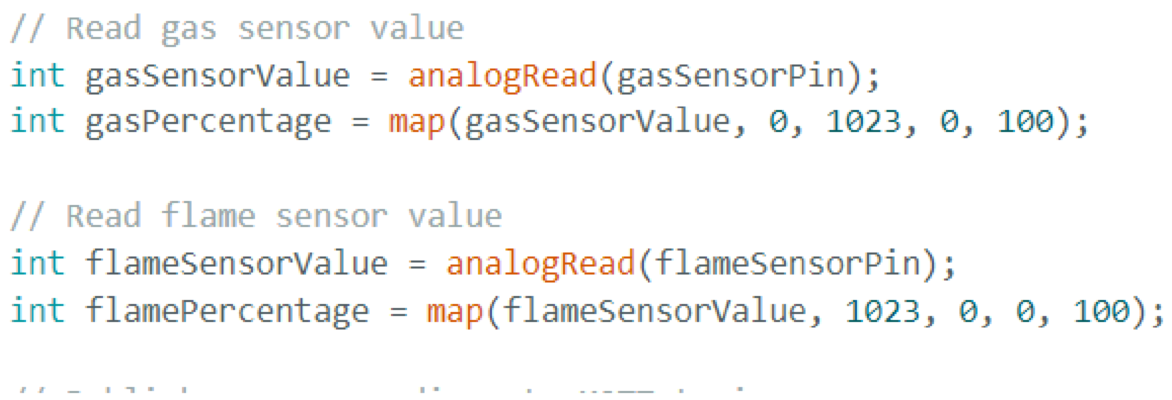
```
// Publish sensor readings to MQTT topics
char payload[4];
snprintf(payload, sizeof(payload), "%d", gasPercentage);
mqttClient.publish(mqttTopicGas, payload);

snprintf(payload, sizeof(payload), "%d", flamePercentage);
mqttClient.publish(mqttTopicFlame, payload);
```

Fig. 7. Publishing Message to Broker

The **‘readGasSensorValue()‘** function retrieves the analog value from the MQ-4 gas sensor pin (‘gasSensorPin‘) using the ‘analogRead()‘ function. This value, representing the detected gas concentration, is then mapped to a percentage range (0 to 100) using the ‘map()‘ function. The resulting gas percentage is stored in the ‘gasPercentage‘ variable, providing valuable information about the gas levels present in the environment.

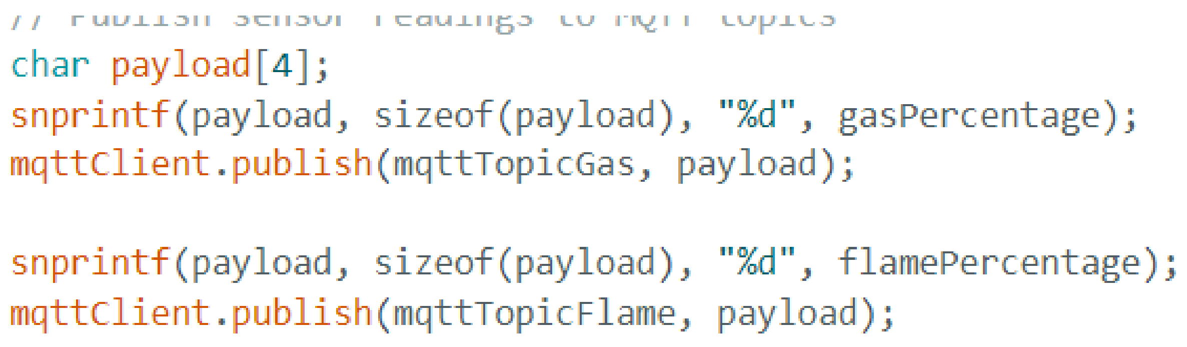
Similarly, the **‘readFlameSensorValue()‘** function employs the ‘analogRead()‘ function to obtain the analog value from the KY-026 flame sensor pin (‘flameSensorPin‘). This value, indicative of the detected flame intensity, is mapped to a percentage range (0 to 100) using the ‘map()‘ function. The calculated flame percentage is stored in the ‘flamePercentage‘ variable, enabling the monitoring of potential fire hazards.

Following the acquisition of the sensor readings, the code utilizes the **‘mqttClient.publish()‘** function to publish the gas and flame percentage values to their respective MQTT topics (‘mqttTopicGas‘ and ‘mqttTopicFlame‘). The ‘snprintf()‘ function is employed to convert the sensor readings to strings, which are then stored in the ‘payload‘ variable. The subsequent publishing of the payload ensures that the gas and flame sensor data is available for further analysis and dissemination.



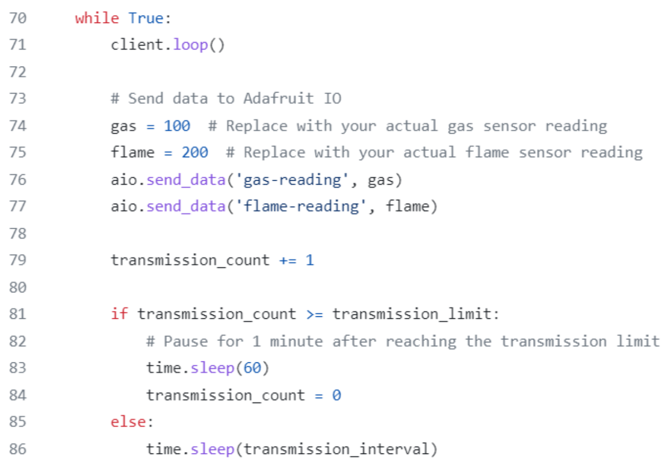
```
// Read gas sensor value
int gasSensorValue = analogRead(gasSensorPin);
int gasPercentage = map(gasSensorValue, 0, 1023, 0, 100);

// Read flame sensor value
int flameSensorValue = analogRead(flameSensorPin);
int flamePercentage = map(flameSensorValue, 1023, 0, 0, 100);
```

Fig. 8. Sensor reading



```
// Publish sensor readings to MQTT topics
char payload[4];
snprintf(payload, sizeof(payload), "%d", gasPercentage);
mqttClient.publish(mqttTopicGas, payload);

snprintf(payload, sizeof(payload), "%d", flamePercentage);
mqttClient.publish(mqttTopicFlame, payload);
```

Fig. 9. Publishing Message to Broker



```python
    while True:
        client.loop()

        # Send data to Adafruit IO
        gas = 100  # Replace with your actual gas sensor reading
        flame = 200  # Replace with your actual flame sensor reading
        aio.send_data('gas-reading', gas)
        aio.send_data('flame-reading', flame)

        transmission_count += 1

        if transmission_count >= transmission_limit:
            # Pause for 1 minute after reaching the transmission limit
            time.sleep(60)
            transmission_count = 0
        else:
            time.sleep(transmission_interval)
```

Fig. 10. Adafruit Python Code

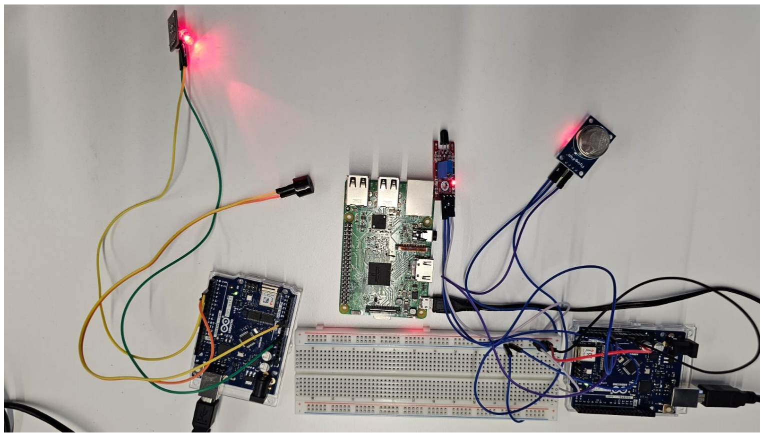


Fig. 11. Visual representation of the system

V. Implementation

The primary components of the system include gas and flame sensors, an Arduino board, a Raspberry Pi acting as an MQTT broker, and MQTT client app or Adafruit IO. These components form the foundation of the system's static structure. Gas sensors are employed to detect gas leaks from appliances such as stoves and ovens in homes, as well as from industrial equipment in businesses. Flame sensors are used to detect fires in the environment. Both sensors are connected to an Arduino board, which serves as an interface to collect data from the sensors. The collected sensor data is then transmitted to a Raspberry Pi, which acts as an MQTT broker. The Raspberry Pi facilitates the communication and data exchange between the sensors and the MQTT client app or Adafruit IO. It receives the data from the Arduino board and forwards it to the appropriate destination. The MQTT client app or Adafruit IO receives the forwarded data from the MQTT broker. These applications are responsible for processing the data and generating alerts based on predefined thresholds or conditions. The alerts are then sent to the users, notifying them about the detected gas leak or fire in real time.

To take immediate actions in response to detected hazards, the system includes actuators such as a Piezo Buzzer and a sprinkler mechanism. When an abnormal level of gas is detected, the Piezo Buzzer is activated to emit an audible alarm. In the event of a fire detection, the sprinkler mechanism is triggered to suppress the fire and minimize potential damage.

A. *Class Diagram*

A Class diagram to describe the classes in our whole system in the actual environment.

B. *Use Case Diagram*

A Use Case diagram shows how users can use the system and what can be done with the system.

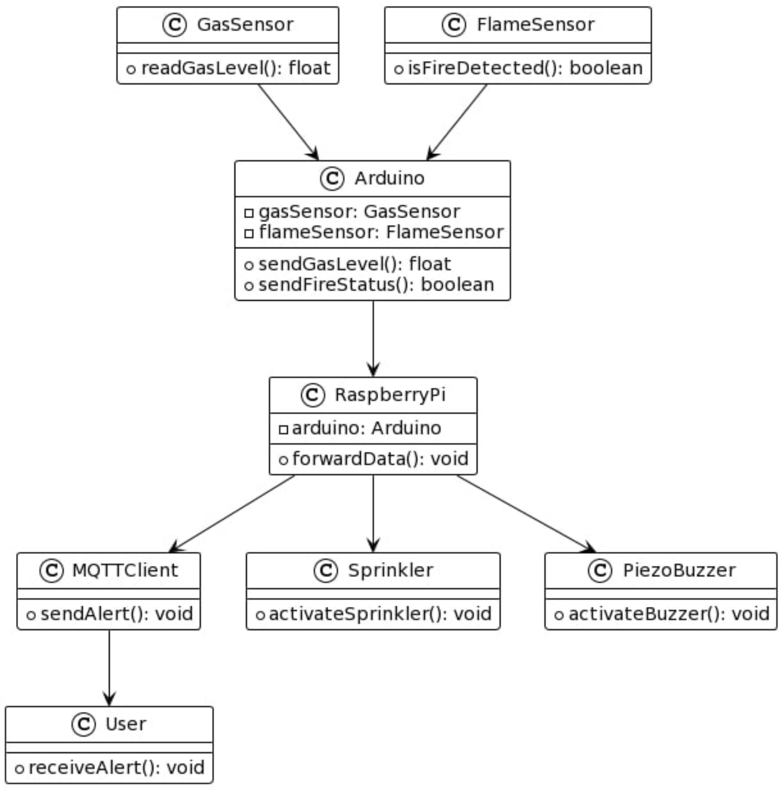


Fig. 12. Class Diagram

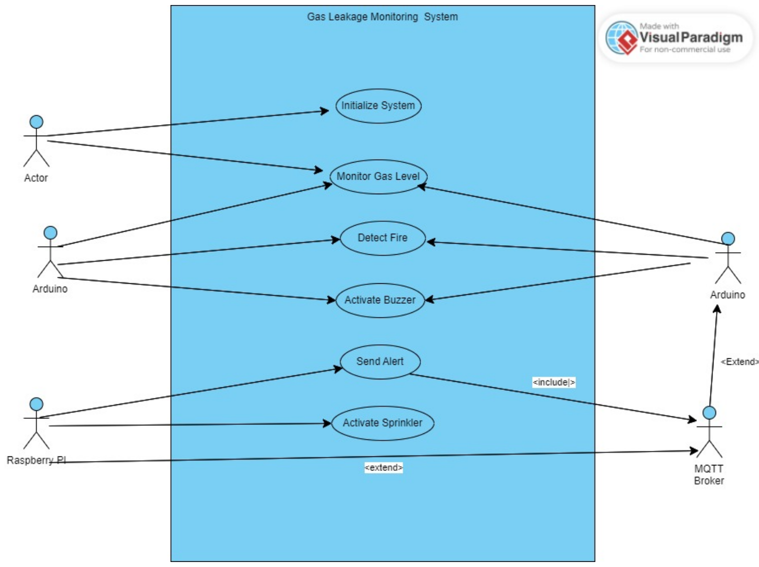


Fig. 13. Use Case Diagram

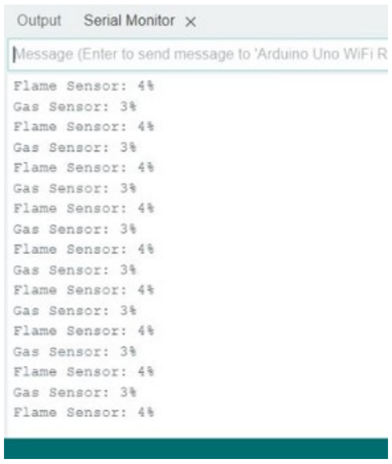
VI. Use Case

The Gas Leakage Monitoring System we implemented can be further observed in the use case perspective.

A. *Arduino and Raspberry Pi 3*

As it can be seen from figure 14, the gas readings and flame readings are observable on serial monitor of the Arduino Uno module that is connected to the MQ 4 sensors and Flame Sensors. These data can be read, used, saved, and sent.

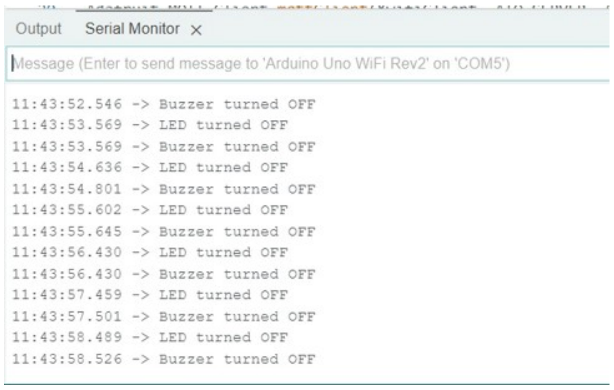
Also, figure 15 shows that the states of buzzer and LED are observable on serial monitor of the Arduino Uno module connected with these actuators. Not only the states of these actuators can be read, saved and sent but also these actuators states can be changed by commanding them to operate.


Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno WiFi R

```
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
Gas Sensor: 3%
Flame Sensor: 4%
```

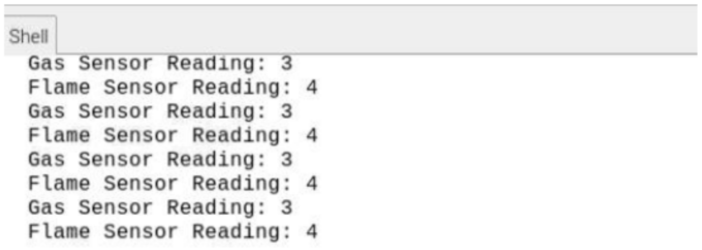
Fig. 14. Sensor Reading value


Output Serial Monitor ×

Message (Enter to send message to 'Arduino Uno WiFi Rev2' on 'COM5')

```
11:43:52.546 -> Buzzer turned OFF
11:43:53.569 -> LED turned OFF
11:43:53.569 -> Buzzer turned OFF
11:43:54.636 -> LED turned OFF
11:43:54.801 -> Buzzer turned OFF
11:43:55.602 -> LED turned OFF
11:43:55.645 -> Buzzer turned OFF
11:43:56.430 -> LED turned OFF
11:43:56.430 -> Buzzer turned OFF
11:43:57.459 -> LED turned OFF
11:43:57.501 -> Buzzer turned OFF
11:43:58.489 -> LED turned OFF
11:43:58.526 -> Buzzer turned OFF
```

Fig. 15. Actuators State


Shell

```
Gas Sensor Reading: 3
Flame Sensor Reading: 4
Gas Sensor Reading: 3
Flame Sensor Reading: 4
Gas Sensor Reading: 3
Flame Sensor Reading: 4
Gas Sensor Reading: 3
Flame Sensor Reading: 4
```

Fig. 16. Python

The figure 16 shows that the readings of the sensors are sent and read on Raspberry Pi 3 module.

B. *Mobile Client and Adafruit IO*

Figure 17 shows that the users can observe the accurate sensor readings and the status of the actuators on the mobile client. They can also activate or deactivate the actuators using their mobile.

Figure 18 shows that the users can access to the information of the system without using their mobiles but on the Adafruit IO. This can further developed so that the remote control of the actuators on this cloud server can be realized as well.

C. *Conclusion for the Use Case of the system*

In conclusion, the system has the following use cases.

- Reading the environmental data : the gas readings and flame readings can be accessed on Arduino Uno, Raspberry Pi, user's mobile client and on cloud server.

- Alarming the users : once the gas readings and flame readings exceed the preset threshold value, the system autonomously notifies the users by activating the actuators. In this alarming scenario, users are notified on their mobile clients. It could be further developed so that the cloud can represent this state as well.

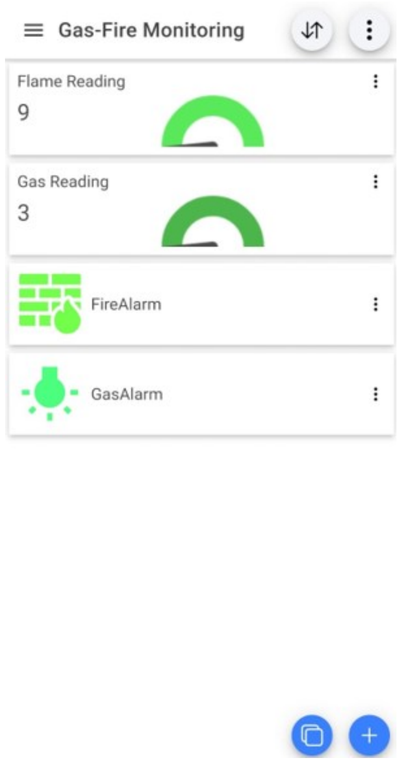
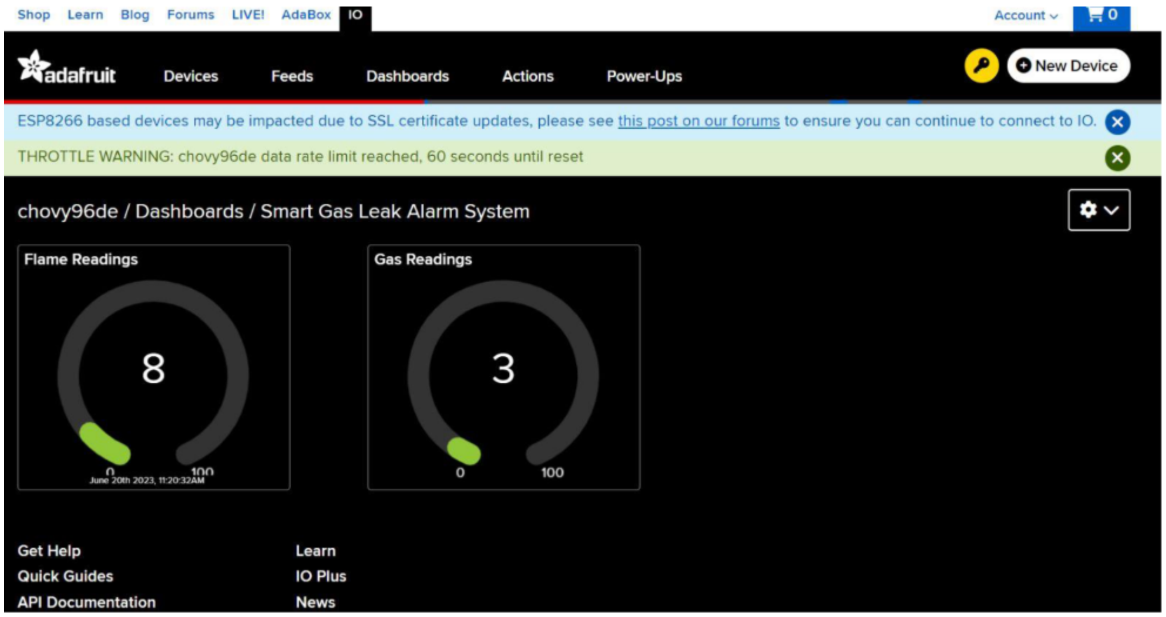


Fig. 17. Mobile Client



ESP8266 based devices may be impacted due to SSL certificate updates, please see this post on our forums to ensure you can continue to connect to IO.

THROTTLE WARNING: chovy96de data rate limit reached, 60 seconds until reset

chovy96de / Dashboards / Smart Gas Leak Alarm System

Fig. 18. Adafruit IO

VII. Conclusion

As discussed in this paper, we have successfully developed an IoT system for Gas Leakage and Flame Detection. To recapitulate, the gas and flame reading data, which was measured with sensors and Arduino, are sent and managed by Raspberry

Pi 3, so that users can observe this information using the user's mobile device or the Adafruit IO cloud server. If the gas and flame reading data exceeds the threshold value, the system alarms the users with piezo buzzer and LED and notifies the user on their mobile device and the cloud. With this system, we managed to achieve our goals that users can be notified of imminent danger of fire.

However, this system can surely be developed further. A possible feature we can develop further is to control the actuators remotely. For example, once the user is notified of the possible fire on the app, the user shall take appropriate actions and turn off the Piezo buzzer or LEDs. We can develop our system further so that deactivation of these actuators can also be done remotely.

Another possible improvement can be made regarding the alarming system. To ensure better safety, provided that the user does not respond to the the alarm for a certain period of time, the system can be developed further to notify the situation to the nearby fire authority.

VIII. References

All of our work related to this project can be found in the GitHub Repository.
https://github.com/Shihab-007/Advanced-Embedded-System-Hardware-Enginnering
https://solwit.com/en/posts/how-iot-is-changing-the-world-around-us-the-future-of-the-internet-of-things/

Methane Gas Sensor MQ4 (analog/digital). (n.d.). Future Electronics Egypt. https://store.fut-electronics.com/products/methane-gas-sensor-mq4-analog-digital

ArduinoModules. (2021). KY-026 Flame Sensor Module. ArduinoModulesInfo. https://arduinomodules.info/ky-026-flame-sensor-module/

Dabrowska, M. (2023). How IoT is Changing the World Around Us — the Future of the Internet of Things. Solwit — Software Development — Tests — Cloud Services. https://solwit.com/en/posts/how-iot-is-changing-the-world-around-us-the-future-of-the-internet-of-things/

Figure II: IoT-Based Fire Alarm System [6]. (n.d.). ResearchGate. https://www.researchgate.net/figure/IoT-Based-Fire-Alarm-System-6$_{fig}2_3$66590302

ARDUINO UNO WiFi REV2. (n.d.). Arduino Official Store. https://store.arduino.cc/products/arduino-uno-wifi-rev2
MQTT Essentials - All Core Concepts explained. (n.d.). https://www.hivemq.com/mqtt-essentials/
Santos, R., Santos, R. (2023). Testing Mosquitto broker and client — Random nerd tutorials. Random Nerd Tutorials. https://randomnerdtutorials.com/testing-mosquitto-broker-and-client-on-raspbbery-pi/
Woolsey, J. (2020). Adafruit IO: Connecting Your Raspberry Pi To The Outside World. Woolsey Workshop.https://www.woolseyworkshop.
io-connecting-your-raspberry-pi-to-the-outside-world/