Gas Leak Detection and Control System Using ESP8266 and Telegram Integration

A REPORT

submitted by

Ankan Routh (21BPS1182)

Sanjay Rao Puthli (21BPS1187)

Saket Gupta (21BPS1611)

Ameiy Acharya (21BPS1104)

B. Tech. Computer Science and Engineering

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2024

Abstract

This report presents the design and implementation of an embedded gas leak detection and control system leveraging the Internet of Things (IoT) paradigm. The system integrates an ESP8266 microcontroller, MQ135 gas sensor, servo motor, online web server, and Telegram bot to enable real-time monitoring, control, and alerting capabilities. The primary objectives are to detect and quantify gas concentrations, trigger alarms for leak scenarios, provide manual override control, and deliver instant notifications via the Telegram messaging platform. The methodology involved hardware setup, software development for data acquisition and processing, and integration of the Telegram bot API. Successful implementation yielded a robust system capable of accurate gas sensing, automated leak response mechanisms, remote monitoring, modular design for multi-scenario use.

1. Objectives

The primary objectives of this project are as follows:

1) Design and implement a gas sensing module capable of accurately detecting and quantifying gas concentrations in the environment.

2) Develop a robust algorithm to distinguish between safe and hazardous gas levels, triggering appropriate alarms and responses.

3) Integrate a manual control mechanism to allow for remote overriding of the system in case of emergencies or maintenance requirements.

4) Establish an online web server to provide real-time monitoring and control capabilities, accessible from any internet-connected device.

5) Incorporate a Telegram bot to deliver instant notifications and alerts regarding gas leaks or system status changes.

2. Introduction

The risk of gas leaks poses significant safety hazards in residential, commercial, and industrial settings. Gas accumulation can lead to explosions, fires, and exposure to toxic compounds, making timely detection and mitigation critical. Traditional gas detection systems often lack real-time monitoring, remote access, and seamless integration with modern communication channels. This project aims to develop an intelligent, IoT-enabled gas leak detection and control system that leverages cutting-edge technologies to address these limitations.

3. Methodology

The project methodology comprised several key stages, including hardware setup, software development, and Telegram bot integration.

Hardware Setup:

- ESP8266 microcontroller: Chosen for its Wi-Fi capabilities, allowing for internet connectivity and communication with the web server and Telegram bot.
- MQ135 gas sensor: Selected for its ability to detect a wide range of gases, including ammonia, nitrogen oxides, benzene, and smoke.
- Servo motor: Incorporated to demonstrate manual control and actuation capabilities, such as opening or closing vents or valves.

Software Development:

- Firmware programming for the ESP8266 using the Arduino IDE, enabling data acquisition from the MQ135 sensor, servo motor control, and communication with the web server and Telegram bot.
- Algorithms for gas concentration calculation, threshold determination, and decision-making logic for alarm triggering and control actions.
- Integration of the Telegram Bot API using the node-telegram-bot-api library, allowing for real-time notifications and user interaction via the Telegram messaging platform.

Telegram Bot Integration:

- Leveraged the Telegram Bot API to create a dedicated bot for the gas leak detection and control system.
- Programmed the bot to send notifications and alerts to subscribed users upon detecting gas leaks or system status changes.

- Implemented commands and conversational interfaces for users to query system information, trigger manual overrides, and perform other control actions via the Telegram bot.

4. Implementation

The implementation phase involved the seamless integration of the hardware components, firmware programming, web server deployment, and Telegram bot configuration.

Hardware Integration:

- The ESP8266 microcontroller was connected to the MQ135 gas sensor and servo motor through the appropriate pin mappings and circuitry.
- Power supply and necessary protections were implemented to ensure safe and reliable operation.

Firmware Programming:

- The ESP8266 was programmed using the Arduino IDE, with custom libraries for sensor data acquisition, servo control, and Wi-Fi connectivity.
- Gas concentration calculation algorithms were implemented, considering sensor characteristics and environmental factors.
- Decision logic was developed to trigger alarms and control actions based on predefined gas concentration thresholds.
- Wi-Fi connectivity was established, enabling communication with the web server and Telegram bot.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <Servo.h>  // Include the Servo library
#include <Wire.h>   // Include the Wire library for I2C devices

// Wi-Fi credentials
const char* ssid = "Ankan's S20 FE";
const char* password = "entq3186";

// Server details
```

```
const char* serverAddress = "http://192.168.200.3:8000";
WiFiClient wifiClient;

// Servo configuration
Servo myservo;
const int servoPin = D1;  // Pin to which the servo is connected

// MQ-135 sensor configuration
const int mq135SensorPin = A0;  // Pin to which the MQ-135 sensor is connected
const unsigned long requestInterval = 5000;  // Interval between requests (milliseconds)
unsigned long lastRequestTime = 0;

// URL parameters
const char* urlParametersGasValue = "?gasvalue=";
const char* urlParameterValveStatusRead = "&valvestatusread=true";
const char* urlParameterValveStatus = "&valvestatus=";
const char* urlParameterManualOverrideRead = "&manualoverrideread=true";

void setup() {
  myservo.attach(servoPin);  // Attach servo to the specified pin
  Serial.begin(9600);        // Start serial communication
  WiFi.begin(ssid, password);  // Connect to Wi-Fi network
}

void loop() {
  unsigned long currentTime = millis();

  if (currentTime - lastRequestTime >= requestInterval) {
    lastRequestTime = currentTime;

    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      int mq135Value = analogRead(mq135SensorPin);  // Read value from the MQ-135 sensor

      // Construct the URL with the gas value
      char url[100];
      sprintf(url, "%s%d%s", serverAddress, mq135Value, urlParametersGasValue);
      strcat(url, urlParameterValveStatusRead);

      // Send HTTP GET request
      http.begin(wifiClient, url);
      int httpCode = http.GET();

      if (httpCode > 0) {
        String payload = http.getString();
        Serial.println(payload);

        if ((payload == "1") || (mq135Value > 710)) {
          myservo.write(90);  // Open the valve
          sprintf(url, "%s%d%s", serverAddress, 1, urlParameterValveStatus);
          http.begin(wifiClient, url);
          httpCode = http.GET();
        } else {
          sprintf(url, "%s%s%s", serverAddress, urlParameterManualOverrideRead,
urlParameterValveStatus);
          http.begin(wifiClient, url);
          httpCode = http.GET();

          if (httpCode > 0) {
            String payload1 = http.getString();
            if (payload1 == "0") {
              myservo.write(0);  // Close the valve
              sprintf(url, "%s%d%s", serverAddress, 0, urlParameterValveStatus);
              http.begin(wifiClient, url);
              httpCode = http.GET();
            }
          }
        }
      } else {
        Serial.printf("HTTP request error: %s\n", http.errorToString(httpCode).c_str());
      }

      http.end();  // Close the connection
    }
  }
}
```

Web Server Deployment:

- The Python-based web server was deployed on a cloud hosting platform or local server, accessible via the internet.
- Secure communication channels were established between the ESP8266 and the web server for data exchange and control commands.

```python
import json
import requests
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib.parse

# Define variables to store data
gasvalue = None
manualoverride = None
valvestatus = None

# Telegram Bot configuration
telegram_bot_token = '6543149823:AAHNchsL62Stg3ZGllj0T1Kd4uJ-u8ZbACo'
telegram_chat_id = '6319653242'
threshold = 750  # Set your threshold value here

def send_telegram_message(message):
    url = f"https://api.telegram.org/bot{telegram_bot_token}/sendMessage"
    data = {'chat_id': telegram_chat_id, 'text': message}
    return requests.post(url, data=data).json()

def save_data_to_file(data, filename='data.json'):
    with open(filename, 'w') as file:
        json.dump(data, file)

def load_data_from_file(filename='data.json'):
    try:
        with open(filename, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return None

class RequestHandler(BaseHTTPRequestHandler):
    data = {'gasvalue': gasvalue, 'manualoverride': manualoverride, 'valvestatus': valvestatus}
    save_data_to_file(data)
    def do_GET(self):
        parsed_url = urllib.parse.urlparse(self.path)
        query_string = parsed_url.query

        if query_string:
            query_dict = urllib.parse.parse_qs(query_string)
            data = load_data_from_file()

            if "gasvalue" in query_dict:
                gasvalue = query_dict["gasvalue"][0]
                data['gasvalue'] = gasvalue
                if float(gasvalue) > threshold:
                    send_telegram_message(f"Warning: Gas value is above the threshold! Current value: 
{gasvalue}")

            if "manualoverride" in query_dict:
                manualoverride = query_dict["manualoverride"][0]
                data['manualoverride'] = manualoverride

            if "valvestatus" in query_dict:
                valvestatus = query_dict["valvestatus"][0]
                data['valvestatus'] = valvestatus

            save_data_to_file(data)

            if "givedata" in query_dict and query_dict["givedata"][0] == 'true':
                response = f"gasvalue: {data['gasvalue']}, valvestatus: {data['valvestatus']}"
            elif "gasvalueread" in query_dict and query_dict["gasvalueread"][0] == 'true':
                response = f"{data['gasvalue']}"
            elif "manualoverrideread" in query_dict and query_dict["manualoverrideread"][0] == 'true':
                response = f"{data['manualoverride']}"
            elif "valvestatusread" in query_dict and query_dict["valvestatusread"][0] == 'true':
                response = f"{data['valvestatus']}"
            else:
                response = "Data received successfully!"

            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(bytes(response, "utf8"))
        else:
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write(bytes("No Data Received Yet!", "utf8"))

if __name__ == "__main__":
    PORT = 8000
    server_address = ('', PORT)
    httpd = HTTPServer(server_address, RequestHandler)
    print(f"Server running on port {PORT}")
    httpd.serve_forever()
```

Telegram Bot Configuration:

- A dedicated Telegram bot was created and configured using the Bot API provided by the Telegram platform.
- The bot was programmed to receive and respond to user commands, queries, and control actions.
- Integration with the gas leak detection system was established, enabling the bot to receive real-time notifications and alerts.
- Subscribed users were added to the bot, allowing them to receive instant updates and interact with the system via the Telegram messaging interface.

```
from telegram.ext import Updater, CommandHandler
import requests
import json

# Telegram Bot configuration
TELEGRAM_BOT_TOKEN = '6543149823:AAHNchsL62Stg3ZGllj0T1Kd4uJ-u8ZbACo'  # Replace with
your actual bot token
GET_URL = 'http://localhost:8000/?givedata=true'

# Command handlers for the Telegram bot
def update(update, context):
    response = requests.get(GET_URL)
    data = response.text.split(',')
    gas_value = data[0].split(':')[1].strip()
    valve_status = data[1].split(':')[1].strip()
    update.message.reply_text(f"Gas value: {gas_value}\nValve status: {valve_status}")

def check_valve(update, context):
    read_url = 'http://localhost:8000/?valvestatusread=true'  # Replace with your actual URL
    response = requests.get(read_url)
    data = response.text

    if data == '0':
        update.message.reply_text("Valve is open.")
    else:
        update.message.reply_text("Valve is shut.")

def shut_valve(update, context):
    write_url = f'http://localhost:8000/?valvestatus=1&manualoverride=1'
    response = requests.get(write_url)
    if response.status_code == 200:
        update.message.reply_text("Valve shut command sent")
    else:
        update.message.reply_text("Failed to send valve shut command")

def open_valve(update, context):
    write_url = f'http://localhost:8000/?valvestatus=0&manualoverride=0'
    response = requests.get(write_url)
    if response.status_code == 200:
        update.message.reply_text("Valve open command sent")
    else:
        update.message.reply_text("Failed to send valve open command")

def main():
    # Create the Updater and pass it your bot's token
    updater = Updater(TELEGRAM_BOT_TOKEN, use_context=True)

    # Get the dispatcher to register handlers
    dp = updater.dispatcher

    # Register command handlers
    dp.add_handler(CommandHandler("update", update))
    dp.add_handler(CommandHandler("shutvalve", shut_valve))
    dp.add_handler(CommandHandler("checkvalve", check_valve))
    dp.add_handler(CommandHandler("openvalve", open_valve))

    # Start the bot
    updater.start_polling()

    # Run the bot until you press Ctrl-C
    updater.idle()

if __name__ == '__main__':
    main()
```

5. Results

Upon successful implementation, the gas leak detection and control system demonstrated the following key results:

Gas Sensing and Leak Detection:

- Accurate and reliable detection of gas concentrations within the environment, with the MQ135 sensor providing real-time data to the system.
- Effective algorithms for distinguishing between safe and hazardous gas levels, triggering appropriate alarms and responses based on predefined thresholds.

Manual Control and Override:

- The integrated servo motor allowed for manual control and actuation of vents, valves, or other mechanical components related to gas mitigation or system maintenance.
- Remote control capabilities were provided through the Telegram bot, enabling authorized users to override the system and initiate specific actions as needed.

Instant Notifications and Alerts:

- The integrated Telegram bot delivered real-time notifications and alerts to subscribed users upon detecting gas leaks or system status changes.

- Users could interact with the bot to query system information, trigger manual overrides, and perform other control actions through a conversational interface.

6. Discussion

The implemented gas leak detection and control system demonstrated robust performance and successful integration of various components and technologies. However, it is essential to address potential limitations and explore opportunities for further enhancements.

Limitations:

- The MQ135 gas sensor, while versatile, may exhibit cross-sensitivity to other gases or environmental factors, potentially affecting the accuracy of gas concentration measurements.
- The system's reliance on internet connectivity could pose challenges in areas with poor or intermittent network coverage, potentially affecting real-time monitoring and control capabilities.
- Scalability and resource constraints might become considerations when deploying the system in large-scale or complex environments with multiple monitoring points or control mechanisms.

Future Enhancements:

- Incorporate additional gas sensors or sensor arrays to improve selectivity and accuracy in gas detection, enabling identification of specific gas types.
- Explore machine learning techniques for advanced data analysis and pattern recognition, potentially enhancing the system's ability to predict and prevent gas leaks proactively.
- Implement failover mechanisms and redundancy measures to ensure system reliability and continuous operation in case of hardware failures or network disruptions.
- Integrate the system with building automation systems, industrial control systems, or other existing infrastructure for seamless integration and centralized monitoring and control.

- Develop mobile applications or extend the Telegram bot functionality to provide enhanced user experiences and additional control options.

7. Conclusion

The gas leak detection and control system developed in this project successfully demonstrated the potential of integrating embedded systems, Internet of Things technologies, and modern communication channels for enhanced safety and control applications. By leveraging the capabilities of the ESP8266 microcontroller, MQ135 gas sensor, servo motor, online web server, and Telegram bot, the system achieved accurate gas sensing, automated leak response mechanisms, remote monitoring.