

LPG GAS DETECTOR

WITH

ALERT MESSAGE

AYUSH KUMAR

REGISTRATION NUMBER: 12510643

ROLL NO: 14

ADITYA RAJ GUPTA

REGISTRATION NUMBER: 12515599

ROLL NO: 30

PRATIK MAHAPATRA

REGISTRATION NUMBER: 12514948

ROLL NO: 28

TUHIN KANTI GARAI

REGISTRATION NUMBER: 12514960

ROLL NO: 29

Abstract— This project aims to design and implement a low-cost, real-time LPG gas leakage detection system suitable for homes, restaurants, and industrial kitchens. The system uses a gas sensor MQ-2 to detect the presence of LPG in the environment. When the gas concentration exceeds a predefined threshold, the system triggers an alarm buzzer and can optionally send SMS alerts.

This project demonstrates how **embedded systems and sensors** can be integrated to solve real-world safety challenges. It promotes awareness of **gas leak hazards** and showcases the potential of **IoT-based preventive solutions**.

Keywords— LPG, Arduino UNO, MQ2 Sensor, Breadboard .

INTRODUCTION

In modern households and industrial environments, **Liquefied Petroleum Gas (LPG)** is widely used for cooking, heating, and energy applications. While LPG is efficient and economical, it poses a serious safety risk due to its **highly**

flammable nature and potential for leakage-related accidents.

To mitigate these risks, this project introduces a **smart LPG Gas Detection System** equipped with **real-time alert messaging**.

The primary goal is to design a **low-cost, reliable, and responsive system** that can:

- **Detect LPG gas leaks** using a gas sensor (e.g., MQ-2)
- **Trigger immediate alerts** through buzzer, LED, and/or SMS notifications
- **Enhance safety** in kitchens, laboratories, and industrial zones

MATERIALS AND

METHODOLOGY

The components used in making the LPG GAS DETECTOR ALERT MESSAGE are as follows-

MATERIALS:

1. Arduino UNO:

*Arduino Uno is a small computer board used for creating electronic projects. It's **user-friendly** and great for beginners. With various input and output pins, it can connect to sensors, lights, and other components. Programmed using a computer, it enables you to **control and automate** various tasks. Arduino Uno is widely used for learning and prototyping in the world of electronics and programming.*

2. MQ2 SENSOR :

*The MQ2 gas sensor is a widely used, low-cost sensor designed to detect combustible gases and smoke in the air. It **basically detects LPG**, smoke.. Its resistance changes when exposed to target gases.*

3. ESP-32 Module :

The ESP-32 module is a powerful, low-cost microcontroller with built-in Wi-Fi and Bluetooth, widely used for IoT, automation, and embedded systems projects.

4. Relay Module:

A relay module is an electronic switching device that allows low-power control signals (like from a microcontroller) to switch high-power electrical loads (like fans, lights, or pumps) on and off.

5. Buzzer:

*A buzzer is an audio signalizing device that produces **sound when powered**,*

commonly used for alerts, alarms, and notifications in electronic systems

6. DC MOTOR.

A DC motor is an electromechanical device that converts direct current (DC) electrical energy into mechanical rotation. It's widely used in robotics, vehicles, and industrial systems due to its controllability and efficiency.

7. SERVO MOTOR:

A servo motor is a precision-controlled motor that uses feedback to regulate its position, speed, and torque—making it ideal for applications requiring accurate motion control like robotics, CNC machines, and IoT devices.

8. LED RED AND GREEN

***Red LED:** May indicate a triggered alert (e.g., gas leak detected).*

***Green LED:** May show that the system is functioning normally or that no hazard is present.*

9. BREADBOARD:

A breadboard is a reusable platform for building and testing electronic circuits without soldering. It's essential for prototyping, especially in DIY, Arduino, and IoT projects.

10. LCD DISPLAY:

An LCD (Liquid Crystal Display) is a flat-panel technology that uses liquid crystals and polarized light to produce images with low power consumption and high clarity.

METHODOLOGY:

1. System Overview

The proposed system is designed to detect leakage of LPG gases such as propane and butane and trigger alerts to prevent hazards. It integrates gas sensors, microcontrollers, and communication modules to ensure real-time detection and response.

2. Sensor Module

MQ2 Gas Sensor is used for detecting LPG concentrations in the air.

- The sensor outputs an analog signal proportional to gas concentration.
- Calibration is performed to set threshold levels for safe and unsafe conditions.

3. Microcontroller Unit

- **ESP-32** is chosen for its dual-core processing and built-in Wi-Fi/Bluetooth capabilities.
- It reads analog input from the MQ2 sensor and compares it against predefined thresholds.
- If the concentration exceeds the threshold, the controller initiates alert protocols.

4. Alert Mechanisms

Visual Alerts: Red LED indicates danger; Green LED indicates normal status.

- **Auditory Alerts:** A buzzer is activated upon detection of leakage.
- **Relay Module:** Automatically cuts off power to connected appliances (e.g., solenoid valve) to prevent further leakage.
- **IoT Notification:** Sends real-time alerts to Discord app using Wi-Fi also display on LCD.

5. Power Supply

- The system operates on a regulated 5V DC supply.
- Backup battery or UPS integration is recommended for uninterrupted operation.

6. IoT Integration

- Data is transmitted to a cloud server or to Discord app.
- Users can monitor gas levels remotely and receive alerts.

7. Safety and Fail-Safe Design

- System includes watchdog timers and error-handling routines.
- Redundant alert paths (LED + buzzer + IoT) ensure reliability.
- Enclosure design prevents environmental interference with sensor readings.

8. Testing and Validation

- Simulated leakage scenarios are used to test sensor response time and accuracy.
- System is validated against known LPG concentrations.
- Performance metrics include detection latency, false positive rate, and alert reliability.

- *ESP 32 Module*
- *MQ2 Sensor*
- *Relay Module*
- *DC Motor*
- *Servo Motor*
- *LCD Display*
- *10 k Potentiometer*
- *Buzzer*
- *LED Green and Red*
- *Breadboard*
- *Jumping Wires*

POSITIONING THE COMPONENTS

1. **Arduino Uno** - Down -Right side of the Box
2. **ESP 32** - Left side of Arduino Uno
3. **MQ2 Sensor**- Front - Right side of the Box
4. **Relay Module**- Right side of the motor
5. **DC Motor**- Left- Front side Out of the box
6. **Servo Motor**- Right side Out of the Box
7. **Buzzer and LED (Green and Red)**- top of the box
8. **10 k Potentiometer** -Front - Left side of the Box

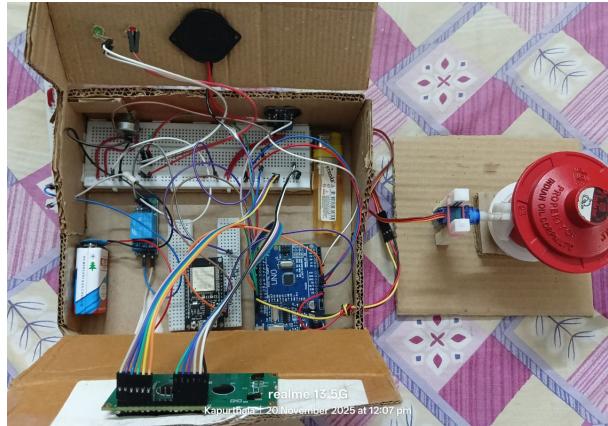
CONSTRUCTION:

Constructing a LPG Gas leakage Detection and alert system involves integrating multiple sensors , components , and functionalities into a single device. While the specific steps can vary depending on the design and technology used , here's general outline of the construction process:

GATHERING THE COMPONENTS

- *Arduino Uno ATMega328P*

9. LCD Display- At the Top of the Box



PROGRAMMING CODE FOR ARDUINO-

```
#include <LiquidCrystal.h>
```

```
#include <Servo.h>
```

```
// LCD pins: RS, EN, D4, D5, D6, D7
```

```
LiquidCrystal lcd(12, 11, 10, 9, 8, 7);
```

```
Servo valve;
```

```
// Pin setup
```

```
const int mq2Pin = A0;
```

```
const int relayPin = 6;
```

```
const int buzzerPin = 4;
```

```
const int redLed = 3;
```

```
const int greenLed = 2;
```

```
const int servoPin = 13;
```

```
// Motor is controlled by relay
```

```
int gasThreshold = 400;
```

```
void setup() {
```

```
Serial.begin(9600); // JSON output to  
ESP32 WiFi module
```

```
// LCD setup
```

```
lcd.begin(16, 2);
```

```
lcd.print(" Gas Detector ");
```

```
delay(1500);
```

```
lcd.clear();
```

```
valve.attach(servoPin);
```

```
pinMode(relayPin, OUTPUT);
```

```
pinMode(buzzerPin, OUTPUT);
```

```
pinMode(redLed, OUTPUT);
```

```
pinMode(greenLed, OUTPUT);
```

```
pinMode(mq2Pin, INPUT);
```

```
// Default safe state
```

```
digitalWrite(relayPin, LOW);
```

```
noTone(buzzerPin);
```

```
digitalWrite(redLed, LOW);
```

```
digitalWrite(greenLed, HIGH);
```

```
valve.write(0); // close valve
```

```
}
```

```

void loop() {
    int gasValue = analogRead(mq2Pin);

    // Show gas value
    lcd.setCursor(0, 0);
    lcd.print("Gas: ");
    lcd.print(gasValue);
    lcd.print("  "); // clear trailing chars

    // GAS LEAK DETECTED
    if (gasValue > gasThreshold) {
        lcd.setCursor(0, 1);
        lcd.print("!! GAS LEAK !! ");

        digitalWrite(redLed, HIGH);
        digitalWrite(greenLed, LOW);
        digitalWrite(relayPin, HIGH); // turn
        ON relay
        tone(buzzerPin, 1000);
        valve.write(90); // open valve

        // Send JSON alert to ESP32
        Serial.print("{\\"gas\\":");
        Serial.print(gasValue);
        Serial.print(",\\"alert\\":1}\\n");

    } else {
        // SAFE CONDITION
        lcd.setCursor(0, 1);
        lcd.print("SAFE      ");
        digitalWrite(redLed, LOW);
        digitalWrite(greenLed, HIGH);
        digitalWrite(relayPin, LOW);
        noTone(buzzerPin);
        valve.write(0); // close valve

        Serial.print("{\\"gas\\":");
        Serial.print(gasValue);
        Serial.print(",\\"alert\\":0}\\n");
    }
}

delay(500);
}

```

CODE FOR ESP32

Module-

```

#include <WiFi.h>
#include <WiFiClientSecure.h>

const char* WIFI_SSID = "Real me 13 5G";
const char* WIFI_PASS = "PASSWORD";

const char* DISCORD_WEBHOOK =
"https://discord.com/api/webhooks/
14406772914402100034/
wsBIV_J0wriqVO89kr2baT62G-

```

```

HTCyQZj9VDdmQ1gCApfleQf8mDl8CEaz
ZuBvpofqZc";

const int SERIAL1_RX = 16;
const int SERIAL1_TX = 17;
WiFiClientSecure client;

// Serial buffer for incoming JSON
char serialBuf[256];
int bufPos = 0;

// cooldown so we don't spam Discord
unsigned long lastDiscordMillis = 0;
const unsigned long
DISCORD_COOLDOWN_MS = 60000UL; // 60 seconds

// Forward declarations
bool postToDiscord(const String
&content);
void connectWiFiDebug();

// Parse a single line like
{"gas":400,"alert":1}

// returns true if at least gas parsed; sets
// gas and alert (alert default 0)
bool parseLine(const char *line, int &gas,
int &alert) {

    gas = -1;
    alert = 0;

    int r = sscanf(line, "{\\ \"gas\\ \" : %d,
\\ \"alert\\ \" : %d}", &gas, &alert);

    // Sometimes Arduino may print
    // without alert field; accept r >= 1
    return (r >= 1);
}

// Escape text so it can safely be placed
// inside a JSON string value.
// Escapes: \ " control chars (newline,
// carriage return, tab, backspace,
// formfeed)

// Leaves UTF-8 multibyte bytes alone
// (they are valid in JSON strings).

String escapeJson(const String &s) {
    String out;
    out.reserve(s.length() + 16);
    for (size_t i = 0; i < (size_t)s.length(); +
+i) {
        char c = s[i];
        switch (c) {
            case '\\': out += "\\\\"; break;
            case '\\': out += "\\\\\"; break;
            case 'b': out += "\\b"; break;
            case 'f': out += "\\f"; break;
            case 'n': out += "\\n"; break;
            case 'r': out += "\\r"; break;
            case 't': out += "\\t"; break;
            default:
                // control characters (0x00 - 0x1F)
                // must be escaped as \u00XX
                if ((unsigned char)c <= 0x1F) {

```

```

char buf[8];

sprintf(buf, "\u00%04x", (unsigned
char)c);

out += buf;

} else {
    out += c;
}

}

return out;
}

void setup() {
    Serial.begin(115200);
    delay(50);
    Serial.println();
    Serial.println("==>Discord bridge (JSON-escaped) ==");
}

// Start Serial1 to read Arduino at 9600
Serial1.begin(9600, SERIAL_8N1,
SERIAL1_RX, SERIAL1_TX);

Serial.println("Serial1 started @9600
(RX=GPIO16 TX=GPIO17)");

// Connect to WiFi with debug
connectWiFiDebug();

Serial.println("Ready. Waiting for
Arduino JSON lines...");

}

void loop() {
    // Ensure WiFi stays connected; attempt
    // reconnect if lost
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi lost.
Reconnecting...");
        connectWiFiDebug();
    }

    // Read from Serial1 buffer
    while (Serial1.available()) {
        char c = (char)Serial1.read();
        // Also echo to USB Serial for
        // debugging
        Serial.write(c);

        if (c == '\r') continue;
        if (c == '\n' || bufPos >=
(int)sizeof(serialBuf) - 1) {
            serialBuf[bufPos] = '\0';
            if (bufPos > 0) {
                // Trim leading whitespace
                int start = 0;
                while (serialBuf[start] &&
isspace((unsigned char)serialBuf[start]))
start++;
                char *line = serialBuf + start;
                // Debug print the received line
                Serial.print("Received line: ");
                Serial.println(line);
            }
            int gas = -1, alert = 0;
        }
    }
}

```

```

if(parseLine(line, gas, alert)) {
    Serial.print("Parsed -> gas=");
    Serial.print(gas);
    Serial.print(" alert=");
    Serial.println(alert);

    unsigned long now = millis();
    bool shouldPost = false;
    if(alert == 1) {
        if(now - lastDiscordMillis >=
DISCORD_COOLDOWN_MS) shouldPost =
true;
        else Serial.println("Alert received
but in cooldown. Skipping Discord post.");
    }

    if(shouldPost) {
        // Build plain message; avoid
        unescaped raw newlines by using \n
        inside string

        String rawMessage = "⚠ GAS
ALERT\\nGas Value: " + String(gas) +
(Arduino)";

        // rawMessage currently contains
        the literal backslash-n; but to be safe we
        will use a nicer message:

        String message = "⚠ GAS
ALERT\\nGas Value: " + String(gas) +
(Arduino)";

        // Escape for JSON

        String esc = escapeJson(message);
        Serial.print("Escaped message: ");
        Serial.println(esc);
        Serial.println("Posting to Discord
(debug)...");

        bool ok = postToDiscord(esc); //
postToDiscord expects unquoted/escaped
content (we will wrap again)

        if(ok) lastDiscordMillis = now;
        else Serial.println("Discord post
failed (see HTTP details).");
    }
} else {
    Serial.print("Unrecognized JSON
line: ");
    Serial.println(line);
}
}

bufPos = 0; // reset buffer
} else {
    if(bufPos < (int)sizeof(serialBuf) - 1)
        serialBuf[bufPos++] = c;
}
}

delay(5);
}

// Connect to WiFi and print status codes
// for debugging

void connectWiFiDebug() {

```

```

Serial.print("Connecting to WiFi SSID:");
}

Serial.print(WIFI_SSID);
Serial.println(" ...");

WiFi.mode(WIFI_STA);
WiFi.disconnect(true);
delay(100);
WiFi.begin(WIFI_SSID, WIFI_PASS);

unsigned long start = millis();
int tries = 0;
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    if (++tries % 8 == 0) {
        int st = WiFi.status();
        Serial.print(" status=");
        Serial.print(st);
        Serial.println("(0-idle,4-no-ssid,5-failed,6-disconnected)");
    }
    if (millis() - start > 30000UL) { // 30s timeout
        Serial.println();
        Serial.print("Timed out connecting.");
        Last status=");
        Serial.println(WiFi.status());
        break;
    }
}

if (WiFi.status() == WL_CONNECTED) {
    Serial.println();
    Serial.print("WiFi connected. IP=");
    Serial.println(WiFi.localIP());
    Serial.print("MAC=");
    Serial.println(WiFi.macAddress());
} else {
    Serial.println("WiFi not connected.");
    Check SSID/Password and 2.4GHz.");
}

// Debug-safe postToDiscord() that prints full HTTP status, headers and body

bool postToDiscord(const String &escapedContent) {
    if (strlen(DISCORD_WEBHOOK) < 10) {
        Serial.println("ERROR: DISCORD_WEBHOOK not set.");
        return false;
    }
    // Build host and path
    String webhook =
String(DISCORD_WEBHOOK);
    if (webhook.startsWith("https://"))
webhook = webhook.substring(8);
    else if (webhook.startsWith("http://"))
webhook = webhook.substring(7);
}

```

```

int slash = webhook.indexOf('/');
if (slash < 0) {
    Serial.println("ERROR: Invalid
webhook URL format.");
    return false;
}

String host = webhook.substring(0,
slash);

String path =
webhook.substring(slash); // includes
leading '/'

// Build JSON payload (we already
escaped the content)

String json = String("{\"content\": \"") +
escapedContent + "\"}");

const char* payload = json.c_str();
size_t payloadLen = strlen(payload); // 
correct byte length in bytes

client.setInsecure(); // skip cert
validation (ok for quick testing)

Serial.print("Connecting to ");
Serial.print(host);
Serial.print(" ... ");

if (!client.connect(host.c_str(), 443)) {
    Serial.println("connection failed");
    return false;
}

// Build and send request headers

String req = "POST " + path + " HTTP/
1.1\r\n";
req += "Host: " + host + "\r\n";
req += "User-Agent: ESP32-Discord-
Debug\r\n";
req += "Accept: application/json\r\n";
req += "Content-Type: application/
json\r\n";
req += "Content-Length: " +
String((unsigned long)payloadLen) +
"\r\n";
req += "Connection: close\r\n\r\n";
client.print(req);

client.write((const uint8_t*)payload,
payloadLen);

// Read and print response status line

String statusLine =
client.readStringUntil('\n');

statusLine.trim();
Serial.print("HTTP <- ");

Serial.println(statusLine); //1.1 400 Bad
Request

// Read and print headers

String header;
int retryAfter = -1;

while (client.connected()) {

    header = client.readStringUntil('\n');
    header.trim();

    if (header.length() == 0) break; // end
of headers

    Serial.print("H: ");
}

```

```

Serial.println(header);

if (header.startsWith("Retry-After:")) {
    retryAfter =
    header.substring(12).toInt();
}

// Read body if any
String body = "";
unsigned long start = millis();
while (client.available() && millis() - start < 3000) {
    body += client.readString();
}
if (body.length()) {
    Serial.println("Body:");
    Serial.println(body);
} else {
    Serial.println("No response body.");
}
client.stop();
// Check for 2xx (success)
if (statusLine.indexOf(" 2 ") >= 0) {
    Serial.println("Discord POST success.");
    return true;
} else {
    Serial.println("Discord POST failed
(non-2xx).");
    if (retryAfter > 0) {
        Serial.print("Server asked to retry
after ");
    }
}

Serial.print(retryAfter);
Serial.println(" seconds.");
return false;
}
}

```

WORKING MODEL:

When LPG gas leaks:

- 1. MQ-2 detects gas**
- 2. Arduino reads value**
- 3. If value > threshold:**
 - Red LED ON
 - Green LED OFF
 - Buzzer ON
 - Fan ON (DC motor)
 - Relay ON
 - Servo rotates to turn OFF the knob
 - ESP32 sends alert to phone (Discord)

4. LCD shows "LEAK DETECTED"

5. If no leak:

System stays normal (green LED ON, fan OFF, buzzer OFF)

RESULTS-

1. Successful Detection of LPG Gas

- The MQ-2 gas sensor accurately detected LPG leakage.

- The readings increased sharply whenever LPG was released near the sensor, proving that the sensor responded correctly to changes in gas concentration.

- “**Gas Detector**” at first 2 second
- Real-time gas value
- “**SAFE**” during normal condition
- “**Leak Detected**” during leakage

2. Automatic Alarm Activation

- When the gas concentration went above the set threshold:

- Red LED turned ON 
- Green LED turned OFF 
- Buzzer activated immediately

- This shows that the alert system worked perfectly



3. Automatic Ventilation System Worked

- The DC fan turned ON automatically during gas leakage.
- This helped to remove leaked gas from the surrounding area, showing that:
 - Ventilation system worked
 - Motor and relay operated correctly

4. Automatic Knob Control with Servo Motor

- The servo motor successfully rotated to close the gas knob during leakage.
- This proved that:
 - The system can prevent accidents
 - Automatic control mechanism is effective

5. Real-Time Display on LCD

- The LCD Display showed:

- This confirmed that the user interface was working properly.

6. Wireless Alert System (ESP32) Functioning

 GAS ALERT
Gas Value: 475 (Arduino)
 GAS ALERT
Gas Value: 461 (Arduino)
 GAS ALERT
Gas Value: 460 (Arduino)

- The ESP32 module was able to send:
- Notification to mobile via **Discord**
- This proved that the system can notify the user remotely even if they are not near the device.

7. All Components Responded Without Delay

- The Arduino UNO processed all sensor readings and output actions instantly.
- The system responded in less than 5 second, making it suitable for real-time safety use

8. Improved Safety and Reliability

- The project successfully:
 - Detected gas leakage
 - Warned the user
 - Controlled the environment
 - Prevented possible accidents
- The model worked reliably during repeated testing.

FUTURE SCOPE:

Integration with Smart Home Systems

- The detector can be connected with **Alexa, Google Home, or Home Automation Systems**.
- In the future, gas leakage can trigger **automatic fan ON, window opening, or main gas valve shutoff** through relays or servo motors.

Battery Backup & Low-Power Operation

- Future versions can include **rechargeable Li-ion batteries, solar charging, and ultra-low-power ESP32 modes** to keep the

system active even during power cuts.

GPS/Location Tracking for Outdoor Use

- In industries or transport (like LPG tankers), GPS can be added.
- If leakage is detected, the system can send **location data** to authorities or safety control rooms

Fire Detection + Gas Detection Combo

- By integrating flame sensors and temperature sensors, the system becomes a **multi-hazard detection solution** for homes and industries.

CONCLUSION:

Integrating an LPG gas leakage detection system with IoT capabilities using ESP-32 and MQ-2 sensor is a powerful step toward smarter safety solutions. By combining real-time sensing, local alerts (via buzzer and LED), and remote monitoring (via platforms like Discord), you create a device that can prevent accidents and save lives.

This project not only strengthens your technical foundation in sensor calibration, microcontroller programming, and cloud integration — it also opens doors to innovation and intellectual

property. With thoughtful design, calibration, and user experience enhancements, your system can evolve into a patent-worthy product that contributes meaningfully to public safety.

REFERENCES:

1. [*IoT Based Gas Leakage Detection System - IJCRT*](#)
2. [*Design of an LPG Leak Detection System Using IoT-Based MQ-2 Sensor - ResearchGate*](#)
3. [*LPG Gas Leakage Detection using ESP32 - ISJEM Journal*](#)
4. [*ESP32IO: ESP32 Gas Sensor Tutorial*](#)
5. [*SriTu Hobby: Gas and Smoke Leakage Detection with ESP32 + Discord*](#)