

Air Quality Monitoring(AQM)

IoT Based Air Pollution Monitoring System in which we will monitor the Air Quality over a webserver using internet and will trigger an alarm when the air quality goes down beyond a certain level, means when there are sufficient amount of harmful gases are present in the air like CO₂, smoke, alcohol, benzene and NH₃.

It will show the air quality in PPM on the LCD and as well as on webpage so that we can monitor it very easily.

We have used MQ135 sensor as the air quality sensor which is the best choice for monitoring Air Quality as it can detect most harmful gases and can measure their amount accurately. In this IOT project, you can monitor the pollution level from anywhere using your computer or mobile. We can install this system anywhere and can also trigger some device when pollution goes beyond some level, like we can switch on the Exhaust fan or can send alert SMS/mail to the us.

*First of all we will connect the **ESP8266 with the Arduino**. ESP8266 runs on 3.3V and if you will give it 5V from the Arduino then it won't work properly and it may get damage. Connect the VCC and the CH_PD to the 3.3V pin of Arduino. The RX pin of ESP8266 works on 3.3V and it will not communicate with the Arduino when we will connect it directly to the Arduino.*

Source Code:

Before beginning the coding for this project, we need to first Calibrate the MQ135 Gas sensor. There are lots of calculations involved in converting the output of sensor into PPM value.

```
#ifndef MQ135_H
#define MQ135_H
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
```

```
#define RLOAD 10.0

/// Calibration resistance at atmospheric CO2 level

#define RZERO 76.63

/// Parameters for calculating ppm of CO2 from sensor resistance

#define PARA 116.6020682

#define PARB 2.769034857


/// Parameters to model temperature and humidity dependence

#define CORA 0.00035

#define CORB 0.02718

#define CORC 1.39538

#define CORD 0.0018


/// Atmospheric CO2 level for calibration purposes

#define ATMOCO2 397.13


class MQ135 {

private:

    uint8_t _pin;


public:

    MQ135(uint8_t pin);

    float getCorrectionFactor(float t, float h);

    float getResistance();

    float getCorrectedResistance(float t, float h);
```

```

float getPPM();
float getCorrectedPPM(float t, float h);
float getRZero();
float getCorrectedRZero(float t, float h);
};
#endif

To get PPM value directly,
}

void loop MQ135 gasSensor = MQ135(A0);

```

```
float air_quality = gasSensor.getPPM();
```

To calibrate the MQ135, let it run for 12 to 24 hours and RZERO value.

```

#include "MQ135.h"

void setup () {
  Serial.begin (9600);

  () {

    MQ135 gasSensor = MQ135(A0); // Attach sensor to pin A0

    float rzero = gasSensor.getRZero();

    Serial.println (rzero);

    delay(1000);

  }

```

After getting the RZERO value.

Now we can begin the actual code for our Air quality monitoring project.

In the code, first of all we have defined the libraries and the variables for the Gas sensor and the LCD. By using the Software Serial Library, we can make any digital pin as TX and RX pin. In this code, we have made Pin 9 as the RX pin and the pin 10 as the TX pin for the ESP8266. Then we have included the library for the LCD and have defined the pins for the same. We have also defined two more variables: one for the sensor analog pin and other for storing air quality value.

```
#include <SoftwareSerial.h>
```

```
#define DEBUG true
```

```
SoftwareSerial esp8266(9,10);
```

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

```
LiquidCrystal lcd(12,11, 5, 4, 3, 2);
```

```
const int sensorPin= 0;
```

```
int air_quality;
```

Then we will declare the pin 8 as the output pin where we have connected the buzzer.

lcd.begin(16,2) command will start the LCD to receive data and then we will set the cursor to first line and will print the 'circuitdigest'. Then we will set the cursor on the second line and will print 'Sensor Warming'.

```
pinMode(8, OUTPUT);
```

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

```
lcd.setCursor (0,0);
```

```
lcd.print ("circuitdigest ");
```

```
lcd.setCursor (0,1);
```

```
lcd.print ("Sensor Warming ");
```

```
delay(1000);
```

Then we will set the baud rate for the serial communication. Different ESP's have different baud rates so write it according to your ESP's baud rate. Then we will send the commands to set the ESP to communicate with the Arduino and show the IP address on the serial monitor.

```
Serial.begin(115200);
```

```
esp8266.begin(115200);
```

```
sendData("AT+RST\r\n",2000,DEBUG);
```

```

sendData("AT+CWMODE=2\r\n",1000,DEBUG);

sendData("AT+CIFSR\r\n",1000,DEBUG);

sendData("AT+CIPMUair_quality=1\r\n",1000,DEBUG);

sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG);

pinMode(sensorPin, INPUT);

lcd.clear();

```

For [printing the output on the webpage](#) in web browser, we will have to use **HTML programming**. So, we have created a string named webpage and stored the output in it. We are subtracting 48 from the output because the read() function returns the ASCII decimal value and the first decimal number which is 0 starts at 48.

```

if(esp8266.available())
{
    if(esp8266.find("+IPD,"))
    {
        delay(1000);

        int connectionId = esp8266.read()-48;

        String webpage = "<h1>IOT Air Pollution Monitoring System</h1>";

        webpage += "<p><h2>";

        webpage+= " Air Quality is ";

        webpage+= air_quality;

        webpage+=" PPM";

        webpage += "<p>";
    }
}

```

The following code will call a function named sendData and will send the data & message strings to the webpage to show.

```

sendData(cipSend,1000,DEBUG);

sendData(webpage,1000,DEBUG);

```

```

cipSend = "AT+CIPSEND=";

cipSend += connectionId;

cipSend += ",";

cipSend += webpage.length();

cipSend += "\r\n";

```

The following code will print the data on the LCD. We have applied various conditions for checking air quality, and LCD will print the messages according to conditions and buzzer will also beep if the pollution goes beyond 1000 PPM.

```

lcd.setCursor (0, 0);

lcd.print ("Air Quality is ");

lcd.print (air_quality);

lcd.print (" PPM ");

lcd.setCursor (0,1);

if (air_quality<=1000)

{

lcd.print("Fresh Air");

digitalWrite(8, LOW);

```

Finally the below function will send and show the data on the webpage. The data we stored in string named 'webpage' will be saved in string named 'command'. The ESP will then read the character one by one from the 'command' and will print it on the webpage.

```

String sendData(String command, const int timeout, boolean debug)

{

String response = "";

esp8266.print(command); // send the read character to the esp8266

long int time = millis();

while( (time+timeout) > millis())

{

```

```

while(esp8266.available())
{
    // The esp has data so display its output to the serial window

    char c = esp8266.read(); // read the next character.

    response+=c;

}

}

if(debug)
{
    Serial.print(response);
}

return response;
}

```

Alternative Python Code:

1. *boot.py* file:- The *boot.py* file has the code that only needs to run once on boot. This includes importing libraries, network credentials, instantiating pins, connecting to your network, and other configurations. We create our web server using sockets and the Python socket API.

The official documentation imports the socket library as follows:

try:

```
import usocket as socket
```

```
client = MQTTClient("umqtt_client", SERVER)
```

except:

```
import socket
```

We need to import the *Pin* class from the *machine* module to be able to interact with the GPIOs.

```
from machine import Pin
```

After importing the socket library, we need to import the network library. The network library allows us to connect the ESP32 or ESP8266 to a Wi-Fi network.

```
import network
```

The MQTT protocol is supported in a built-in library in the Micropython binaries -- this protocol is used to send data from your ESP8266, over WIFI, to a free cloud database.

We used umqtt.simple library, and knowing our SERVER ID, it is possible to create our MQTT client object :

```
from umqtt.simple import MQTTClient
```

```
SERVER = "mqtt.thingspeak.com"
```

The following lines turn off vendor OS debugging messages:

```
import esp
```

```
esp.osdebug(None)
```

Then, we run a garbage collector:

```
import gc
```

```
gc.collect()
```

A garbage collector is a form of automatic memory management. This is a way to reclaim memory occupied by objects that are no longer used by the program. This is useful to save space in the flash memory.

The following variables hold network credentials:

```
ssid = '<your_ssid>'
```

```
password = '<your_network_password>'
```

Then, setting the ESP32 or ESP8266 as a Wi-Fi station:

```
station = network.WLAN(network.STA_IF)
```

After that, activating the station:

```
station.active(True)
```


Finally, the ESP32/ESP8266 connects to the router using the SSID and password defined earlier:

```
station.connect(ssid, password)
```

The following statement ensures that the code doesn't proceed while the ESP is not connected to the network.

```
while station.isconnected() == False:  
    pass
```

After a successful connection, print network interface parameters like the ESP8266 IP address – using the `ifconfig()` method on the station object.

```
print('Connection successful')  
print(station.ifconfig())
```

`from time import sleep` Create a Pin object called `led` that is an output, that refers to the ESP8266 GPIO2:

```
led = Pin(2, Pin.OUT)
```

2. `main.py` file:- The `main.py` file contains the code that runs the webserver to serve files and perform tasks based on the requests received by the client.

First, we have to import the libraries for modules, sensors (MQ135, DHT) and for IoT Platform Thing-speak (MQTT Client):

```
from machine import Pin
```

```
import dht, math, time, network, micropython, esp
```

```
from machine import ADC
```

```
from umqttsimple import MQTTClient
```

```
import ubinascii
```

```
from MQ135 import MQ135
```

To initialise the sensors by creating a DHT and MQ135 instance as follows :

```
sensor_dht = dht.DHT11(Pin(5))
```

```
sensor_mq135 = MQ135(0)
```

Then create MQTT client instance and write your channel ID , write API key from channel created on ThingSpeak. Then create MQTT "Topic"

```
SERVER = "mqtt.thingspeak.com"
```

```
client = MQTTClient("umqtt_client", SERVER)
```

```
CHANNEL_ID = "<your_channel_ID>"
```

```
WRITE_API_KEY = "<your_channel_write_API_key>"
```

```
topic = "channels/" + CHANNEL_ID + "/publish/" + WRITE_API_KEY
```

Run the loop to measure the temperature, humidity, air quality with the help of sensors as follows :

```
while True:
```

```
    sensor_dht.measure()
```

```
    temperature = sensor_dht.temperature()
```

```
    humidity = sensor_dht.humidity()
```

```
    rzero = sensor_mq135.get_rzero()
```

```
    corrected_rzero = sensor_mq135.get_corrected_rzero(temperature, humidity)
```

```
    resistance = sensor_mq135.get_resistance()
```

```
    ppm = sensor_mq135.get_ppm()
```

```
    corrected_ppm = sensor_mq135.get_corrected_ppm(temperature, humidity)
```

With those variables updated, we created our "MQTT Payload" and that's it! We are ready to send data to ThinsSpeak :

```
    payload =  
    "field1="+str(temperature)+"&field2="+str(humidity)+"&field3="+str(corrected_ppm)
```

```
    client.connect()
```

```
client.publish(topic, payload)
```

```
print("Payload published successfully")
```

```
#print(publish_check)
```

```
client.disconnect()
```

```
time.sleep(30)
```