Final Project Report

Title: Environmental Detection Project

Team Members:

Member	Contact
Sofia Yang	syang78@hawk.iit.edu
Zara Moinuddin	zmoinuddin@hawk.iit.edu

Introduction :

Background

We recognize that climate change and air pollution are some of our generation's biggest challenges. It is critical for us to either adapt to changing environmental conditions, or develop a solution to mitigate these conditions.

Our team believes that by collecting valuable air pollution metrics, we can analyze the data and use it as the basis for further solutions to predict or mitigate the effects of climate change.

In addition, by continuously sensing air pollution metrics, we can create a detection system to alert when air quality conditions are unsafe. This would be extremely useful for public health and safety measures in areas of the world that are most impacted by climate change.

Problem Statement

The problem statement was to maintain the 'safety' of inhabitants in a closed-environment by detecting levels of air pollution.

Thus, the team decided to develop an automatic monitoring and notification system for environmental air quality and pollution metrics.

Through this project, our goal is to collect metrics for ppm-based levels of toxic gasses (such as carbon monoxide, ammonia, nitric oxide, etc) and visualize the trends of these gas concentrations as charts or graphs. The team's second goal is to create a notification system to send email alerts when the concentrations of these gasses are at an unsafe level.

Some challenges in the project would include being able to constantly collect metrics without any data loss, as well as instantaneously visualizing data after collection without any delay. Another challenge would include instantaneous notifications when the gas levels are detected as unsafe. If there were a delay in notifications, it could compromise the health of the user who is relying on the system as a safety indicator.

Research

The team conducted research on the following topics to develop an encompassing solution:

Internet of Things

Internet of Things (IoT) is a system of interconnected devices that communicate with each other, or to a processing server in the cloud. IoT's innovation is in vast data collection and analysis, as well as using collected data to automate services that were previously only manually completed.

One example of IoT applications in industries include smart farming, where sensors are used to analyze soil and crop conditions, and identify which crops are healthy.
A centralized server would determine which area of crops need watering / fertilizer, and then would direct sprinklers / fertilizer dispensers to turn on in that area of crops.

Sensor Research

DHT11

The DHT11 is a temperature and humidity sensor with 3 pins (to VCC, ground, and a data output pin). Each DHT11 sensor includes a NTC thermistor (a resistor sensitive to temperature), as well as a humidity measuring component that utilizes electrodes which decrease in resistance as the amount of humidity increases.

The DHT11 sensor supports a power supply from 3 - 5 V, and a max current of 2.5 amperes. The range of the sensor's measurements are from 40 - 80°C for temperature, and 0-100% humidity.

MQ135

The MQ135 is an air quality detection sensor with 4 pins (to VCC, ground, and 2 output data pins; 1 digital, an 1 analog). The digital pin returns a 'high' output signal when gas levels go over a set threshold level (which can be calibrated using a potentiometer or code). The MQ135's analog pin returns a numerical float value corresponding to the PPM levels of gasses.

Using a single analog output from the sensor, it is possible to derive the ppm levels for multiple gasses. This is because each gas has an individual characteristic curve (a function of the sensor's resistance as a function of ppm level). Thus, by defining a function to compare each datapoint to the corresponding characteristic curve, it is possible to read ppm levels for a range of gasses.

ThingSpeak IOT Platform

ThingsSpeak is a cloud-based IOT platform for logging and storing data from devices (such as sensors, microcontrollers, etc). Using ThingsSpeak, it is possible to develop applications, and use Rest API / HTTP requests to retrieve analyzed data from the platform as comma-separated value files, or JSON documents.

SMTP / SendGrid

SMTP (Simple mail transfer protocol) is a standardized protocol for sending email messages to a recipient. SMTP services are used to convert emails to the standardized protocol, and route it to the corresponding recipient. By using SMTP services such as SendGrid / MailerSend, it is

possible to send emails from a custom domain through making API requests to the service from a pre-existing application.

Front-End Web Development

The online ‘site maker’ Wix was what was used to create the website for showing the data in the project. It’s an easy to use platform that can help one build a website without having to know how to code. It has a drag and drop feature that makes it extremely user friendly.

iframe/HTML Embed:

- HTML embed will be able to send the data to the website and have it displayed in the website.
- It was put in by wix having an area where you can just simply copy and paste the HTML code and add it to the website.

Prototype Description

1. System for collecting air pollution metrics through the MQ135 and DHT11 sensors, and logging the metrics as standardized JSON files.

2. Front-end web application for visualizing collected data as graphs / charts of the sensor value trends over time.

3. A notification system to send email alerts whenever the pollutant values recorded are above a predefined ‘safe’ threshold.

This prototype would fulfill all the project specifications the team set out to fulfill in the problem statement.

The metrics collected by the prototype include temperature, humidity, CO, CO2, CH3, NH4, and (CH3)2CO.

Implementation:

Software

The sensor metrics collection and streaming system was implemented in C using the Arduino IDE.

The implementation utilizes the MQUnifiedsensor and DHTlib libraries.

Algorithm

The solution implementation first connects to the in-lab Wifi network using AT commands. Then, every 2000 milliseconds, the code reads the sensor metrics from the DHT11 and MQ135 sensors and streams it to ThingSpeak using HTTP requests.

After that, the code loops through each metric value and checks if the values are greater than or equal to its corresponding threshold value (which is stored in an array of floats). For each value that is over the threshold value, the code adds a descriptive string/line to a JSON document that summarizes the data point. When all values have been looped through, the code writes an HTTP request to a SMTP server and sends an email to the recipient.

The flowchart for the code is shown below:

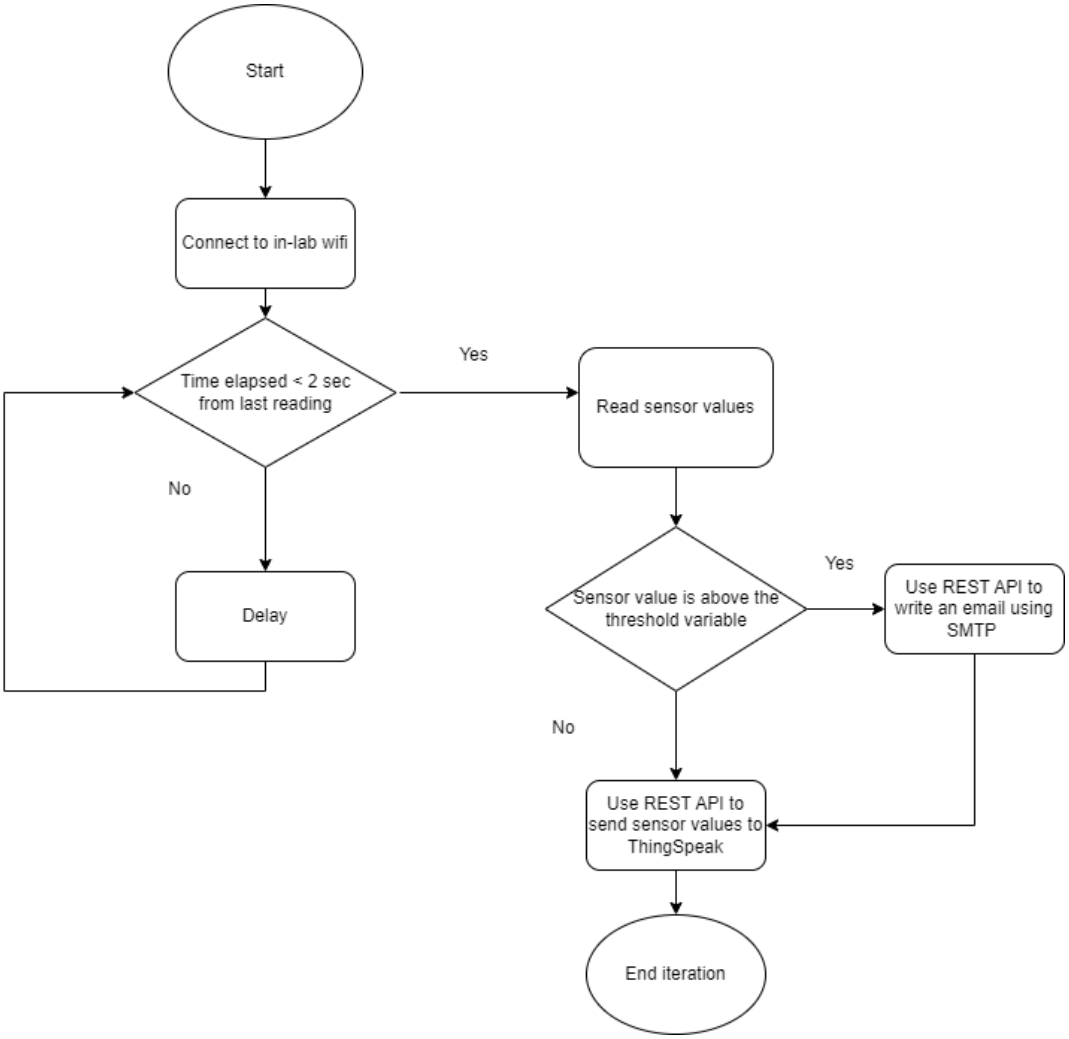


Figure 1: The software flowchart.

One obstacle that the team faced was having the emails instantaneously trigger. In the first implementation, the team utilized ThingsSpeak's interface for MATLAB scripting to trigger emails directly from the IoT platform. However, there was a 2-3 minute delay in sending the emails.

The team later switched to using HTTP requests to an SMTP server (similar to SendGrid), and sending emails from a custom domain. This lessened the delay between the trigger and sending the emails.

Hardware

The hardware component of the implementation uses an Arduino Uno microcontroller, as well as a Wifi Shield module. This is connected to a breadboard which is powered using a 5V battery pack.

The DHT11 temperature and humidity sensor and MQ135 air quality sensor are wired to power and ground terminals on the breadboard, and the output pins of each sensor are connected to the digital and analog pins of the Arduino board.

In addition, an LED was added to the breadboard to test whether the breadboard is working; the LED should be lit when the breadboard is functioning and the Arduino is on.

The final schematic is shown below:

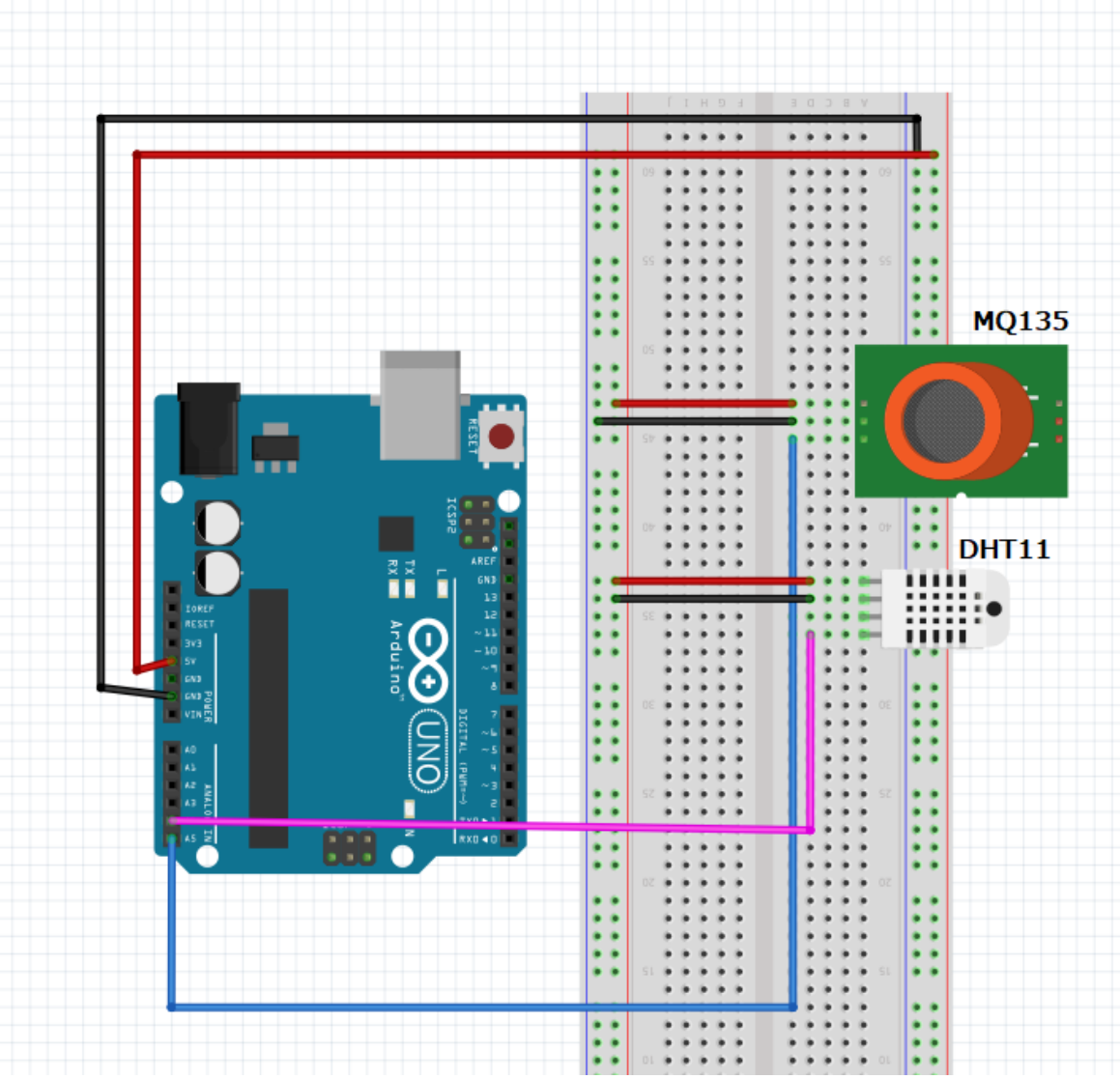


Figure 2: The breadboard wiring diagram.

One obstacle the team faced was that the pinouts in the datasheets were slightly different from the actual pinouts of the sensor modules used. It required a methodical debugging process to discover the errors in wiring the sensors, and rectify those errors accordingly.

Analysis

The prototype created was successful, and fulfilled the requirements the team set out to achieve in the problem statement.

One shortcoming of the prototype is that the system collects metrics only when the server / computer connected to the Arduino is powered on. This means that the system is incapable of operating individually from its computer. In future modifications, it would be useful to develop a solution to ensure that collected metrics can be streamed at all times.

The metrics visualization system works as expected, and fulfills the initially set requirements. There is very little delay between collecting and graphing each data point.

The notification system is also successful; the system sends emails to a specified recipient when pollution metrics are above a safe threshold. The delay between receiving metrics and sending alerts is very little. A way to improve this component would be to implement a back-end component of the web application to let users join a 'mailing list', and have the notification system send emails to all users in the mailing list, rather than manually defining recipients in the code.

Conclusion

Our project was successful; in the timeline of three to four weeks, the team created a working prototype that fulfills our chosen problem statement.

Future modifications to the project would include making improvements to the system as stated above in the ‘Analysis’ section.
A possible continuation of this project would be to develop a regression-based machine learning model and utilize the pollution metrics collected to forecast air quality trends. In this way, the system can predict when the air quality will drop, and warn in advance.

References

1. Arduino-Libraries. (n.d.). *WiFiNINA/api.md at master · arduino-libraries/Wifinina*. GitHub. Retrieved November 13, 2022, from https://github.com/arduino-libraries/WiFiNINA/blob/master/docs/api.md

2. *Write data to channel*. Write Data to Channel - MATLAB & Simulink. (n.d.). Retrieved November 13, 2022, from https://www.mathworks.com/help/thingspeak/write-data.html

3. DHT11 humidity & temperature sensor - mouser.com. (n.d.). Retrieved November 13, 2022, from https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf?origin=new

4. Arduino - temperature humidity sensor: Arduino tutorial. Arduino Getting Started. (n.d.). Retrieved November 13, 2022, from https://arduinogetstarted.com/tutorials/arduino-temperature-humidity-sensor

5. Osoyoo ESP8266 Wi-Fi Module Lesson 1: "Hello world" HTTP web server. OSOYOO ESP8266 Wi-Fi Module Lesson 1: "Hello World" HTTP web server " osoyoo.com. (n.d.). Retrieved November 20, 2022, from https://osoyoo.com/2020/12/20/osoyoo-esp8266-wi-fi-module-lesson-1-hello-world-http-web-server/

6. miguel5612. (n.d.). Miguel5612/MQSENSORSLIB_DOCS: Documentation about sensors MQ. GitHub. Retrieved November 20, 2022, from https://github.com/miguel5612/MQSensorsLib_Docs/

7. Technical Data MQ-135 gas sensor - olimex. (n.d.). Retrieved November 20, 2022, from https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf

8. miguel5612. (n.d.). Miguel5612/mqsensorslib: We present a unified library for MQ sensors, this library allows to read MQ signals easily from Arduino, Genuino, ESP8266, ESP-32 boards whose references are MQ2, MQ3, MQ4, MQ5, MQ6, MQ7, MQ8, MQ9, MQ131, MQ135, MQ136, MQ303A, MQ309A. GitHub. Retrieved November 20, 2022, from https://github.com/miguel5612/MQSensorsLib

9. The University. (1978). *What is ...* Amazon. Retrieved December 7, 2022, from https://aws.amazon.com/what-is/iot/

Appendix

Credits

Sofia Yang

- Help define problem statement and researched components needed for a solution
- Contributed to the hardware prototyping process
- Implemented the data collection and streaming functions
- Implemented the notification system

Zara Moinuddin

- Helped define and researched information about the problem statement
- Contributed to the hardware prototyping process
- Helped find resources/documentation during the software implementation process
- Created the final web application to showcase the completed prototype

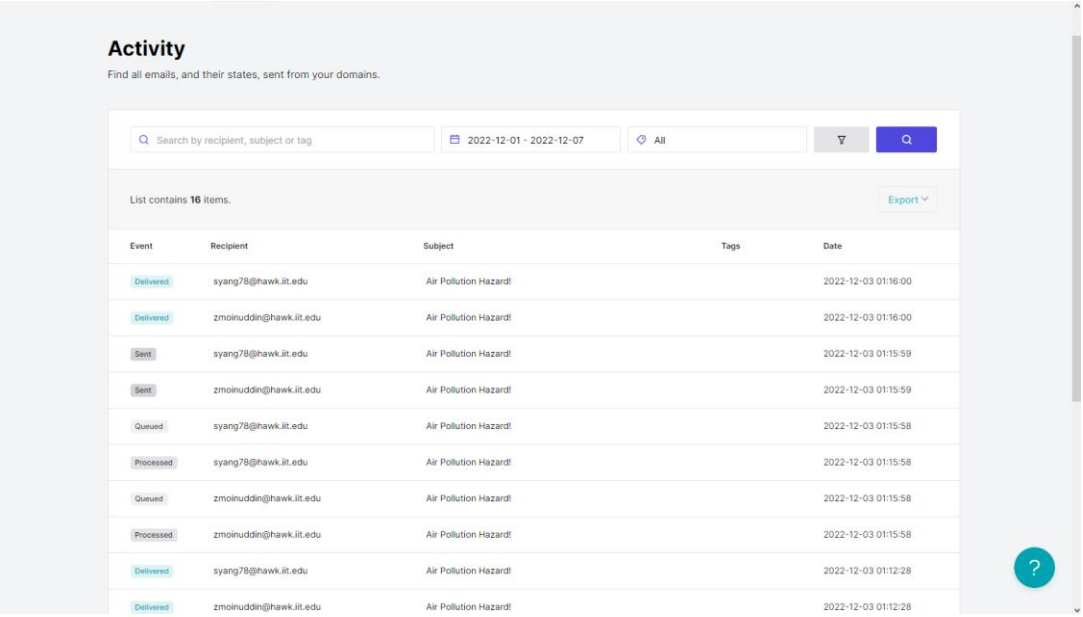
Demonstration

The final webapp created can be accessed here:
https://zmoinuddin.wixsite.com/environmentaldetect

Logs of the collected metrics are shown here:
https://thingspeak.com/channels/1921499

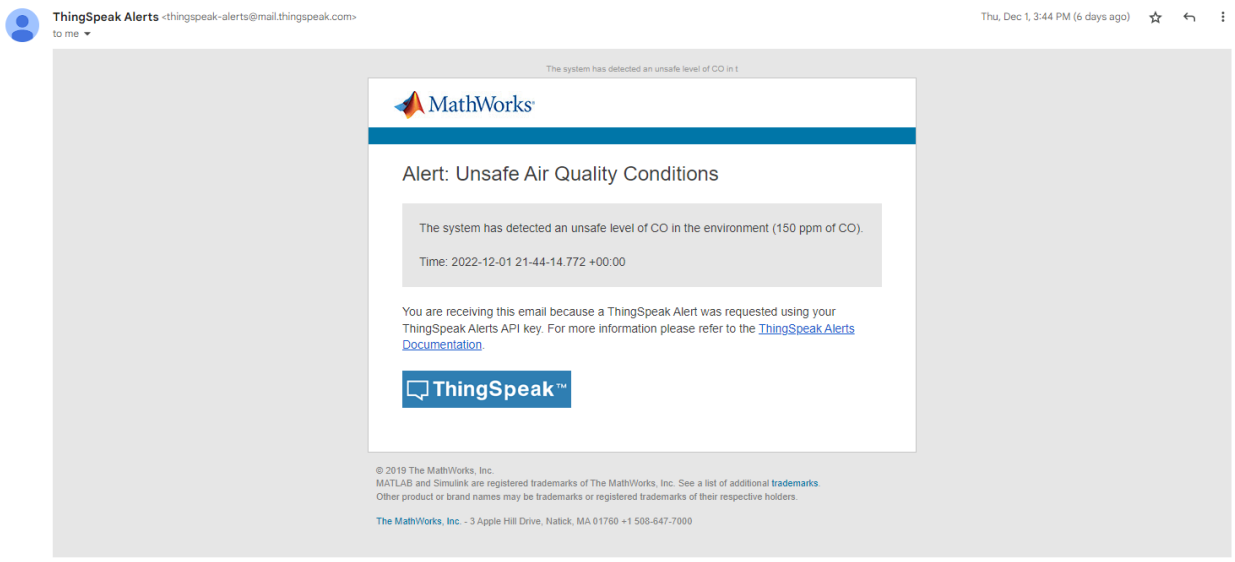
The CSV files can also be accessed from here:

A screenshot of the logged email notifications are shown below:

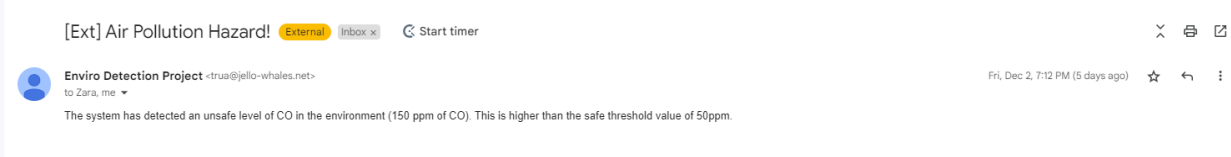


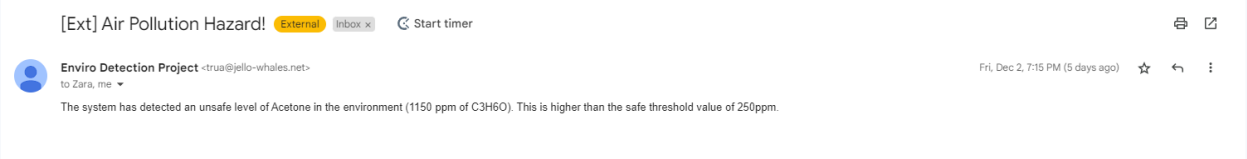
Attachment 1: MailerSend SMTP logged notifications

The emails sent are also shown below:



Attachment 1: The emails scripted using ThingsSpeak





Attachment 2, 3: The emails sent using the SMTP.

Source Code

```
#include <SoftwareSerial.h>
SoftwareSerial esp(4,5);

#include<dht.h>
dht DHT;

#define DHT11_PIN 3

#define DEBUG true
String TSIP = "184.106.153.149";// thingspeak.com ip
String MSIP = "https://api.mailersend.com/v1/email";
String Api_key = "GET /update?key=***";
String mailerSendBearerToken = "***";

int error;
const int sensor_pin = A0;
float temp;
float humi;
float output;
#include <MQUnifiedsensor.h>

//Definitions
#define placa "Arduino UNO"
#define Voltage_Resolution 5
#define pin A2
#define type "MQ-135" //MQ135
#define ADC_Bit_Resolution 10 // For arduino UNO/MEGA/NANO
#define RatioMQ135CleanAir 3.6//RS / R0 = 3.6 ppm
//#define calibration_button 13

//Declare Sensor
MQUnifiedsensor MQ135(placa, Voltage_Resolution, ADC_Bit_Resolution, pin, type);

void setup()
{
  Serial.begin(9600);
  esp.begin(9600);
   MQ135.setRegressionMethod(1); //_PPM =  a*ratio^b

  MQ135.init();
  pinMode(sensor_pin,INPUT);

  send_command("AT+RST\r\n", 2000, DEBUG); //reset module
  send_command("AT+CWMODE=1\r\n", 1000, DEBUG); //set station mode
  send_command("AT+CWJAP=\"***\",\"***\"\r\n", 2000, DEBUG);   //connect wifi network
  while(!esp.find("OK")) { //wait for connection
  Serial.println("Connected");}
   Serial.print("Calibrating please wait.");
  float calcR0 = 0;
  for(int i = 1; i<=10; i ++)
  {
```

```
    MQ135.update(); // Update data, the arduino will read the voltage from the analog pin
    calcR0 += MQ135.calibrate(RatioMQ135CleanAir);
    Serial.print(".");
  }
  MQ135.setR0(calcR0/10);
  Serial.println("  done!.");

  if(isinf(calcR0)) {Serial.println("Warning: Connection issue, R0 is infinite (Open circuit
detected) please check your wiring and supply"); while(1);}
  if(calcR0 == 0){Serial.println("Warning: Connection issue found, R0 is zero (Analog pin shorts
to ground) please check your wiring and supply"); while(1);}
}

void loop()
{
int chk = DHT.read11(DHT11_PIN);
temp = DHT.temperature;
humi = DHT.humidity;
  start: //label
  error=0;
    MQ135.update(); // Update data, the arduino will read the voltage from the analog pin

  MQ135.setA(605.18); MQ135.setB(-3.937); // Configure the equation to calculate CO concentration
value
  float CO = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b
values set previously or from the setup

  MQ135.setA(77.255); MQ135.setB(-3.18); //Configure the equation to calculate Alcohol
concentration value
  float Alcohol = MQ135.readSensor(); // SSensor will read PPM concentration using the model, a
and b values set previously or from the setup

  MQ135.setA(110.47); MQ135.setB(-2.862); // Configure the equation to calculate CO2
concentration value
  float CO2 = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b
values set previously or from the setup

  MQ135.setA(44.947); MQ135.setB(-3.445); // Configure the equation to calculate Toluen
concentration value
  float Toluen = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and
b values set previously or from the setup

  MQ135.setA(102.2 ); MQ135.setB(-2.473); // Configure the equation to calculate NH4
concentration value
  float NH4 = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b
values set previously or from the setup

  MQ135.setA(34.668); MQ135.setB(-3.369); // Configure the equation to calculate Aceton
concentration value
  float Aceton = MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and
b values set previously or from the setup
  String command = Api_key + "&field1=" + temp + "&field2=" + humi + "&field3=" + CO + "&field4="
+ Alcohol + "&field5=" + CO2 + "&field5=" + Toluen + "&field6=" + NH4 + "&field7=" + Aceton
+"\r\n" ;
  updatedata(TSIP, command);

  if (error==1){
    goto start; //go to label "start"
  }
  delay(1000);
}
```

```
void updatedata(String IP, String str){
  String command = "AT+CIPSTART=\"TCP\",\"";
  command += IP;
  command += "\",80";
  Serial.println(command);
  esp.println(command);
  delay(2000);
  if(esp.find("Error")){
    return;
  }
  //command = Api_key + "&field1=" + temp + "\r\n" ;
  command = str;
  Serial.print("AT+CIPSEND=");
  esp.print("AT+CIPSEND=");
  Serial.println(command.length());
  esp.println(command.length());
  if(esp.find(">")){
    Serial.print(command);
    esp.print(command);
  }
  else{

   Serial.println("AT+CIPCLOSE");
   esp.println("AT+CIPCLOSE");
    //Resend...
    error=1;
  }
  }


void updatedata2(String IP, String str){
  String command = "AT+CIPSTART=\"TCP\",\"";
  command += IP;
  command += "\",80";
  Serial.println(command);
  esp.println(command);
  delay(2000);
  if(esp.find("Error")){
    return;
  }
  command = str;
  Serial.print("AT+CIPSEND=");
  esp.print("AT+CIPSEND=");
  Serial.println(command.length());
  esp.println(command.length());
  if(esp.find(">")){
    Serial.print(command);
    esp.print(command);
  }
  else{

   Serial.println("AT+CIPCLOSE");
   esp.println("AT+CIPCLOSE");
    //Resend...
    error=1;
  }
  }

String send_command(String command, const int timeout, boolean debug)
{
  String response = "";
  esp.print(command);
```

```
  long int time = millis();
  while ( (time + timeout) > millis())
  {
    while (esp.available())
    {
      char c = esp.read();
      response += c;
    }
  }
  if (debug)
  {
    Serial.print(response);
  }
  return response;
}
```