IoT based Air Quality Index Monitoring System

Project ID: 22303

B.Tech. Project Report
submitted for fulfillment of
the requirements for the
Degree of Bachelor of Technology
Under Biju Pattnaik University of Technology

Submitted By

Ramya Pandey **ROLL NO. ECE201811298**

Shiv Prasad Mishra **ROLL NO. ECE201812285**

2021 – 2022

Under the guidance of

Prof. Manoj Kumar Senapati

NIST INSTITUTE OF SCIENCE & TECHNOLOGY (Autonomous)
Palur Hills, Berhampur, Odisha – 761008, India

BONAFIDE CERTIFICATE

This is to certify that the Project entitled "**IoT based Air Quality Index Monitoring System** "is a bonafide record by **Ramya Pandey** (Roll No. BTECH 201811298) & **Shiv Prasad Mishra** (Roll No. BTECH 201812285) under my supervision and guidance, in partial fulfilment of the requirements for the award of Degree of Bachelor of technology from National Institute of Science and Technology under Biju Pattnaik University of Technology for the year 2022.

Prof. Manoj Kumar Senapati
(Faculty Advisor)
Asst. Professor
Dept. of Electronics and Communication Engineering

ACKNOWLEDGEMENT

We would like to take this opportunity to thank all those individuals whose invaluable contribution directly or indirectly has gone into the making of this project a tremendous learning experience for us.

It is our proud privilege to epitomize our deepest sense of gratitude and indebtedness to our faculty guide, **Mr. Manoj Kumar Senapati** for his valuable guidance, keen and sustained interest, intuitive ideas, and persistent endeavor. His guidance and inspirations enabled us to complete our report work successfully.

We give our sincere thanks to **Mr. Rakesh Roshan**, Dept. B. Tech Project Coordinator, **Dr. M Suresh**, Head of Department, ECE and **Mr. Rajesh Kumar Dash**, B. Tech Project Coordinator for giving us the opportunity and motivating us to complete the project within the stipulated period and providing a helping environment.

We acknowledge with immense pleasure the sustained interest, encouraging attitude and constant inspiration rendered by **Prof. Sukanta Mohapatra** (Chairman) and **Dr. Priyadarshi Tripathy** (Principal) N.I.S.T. Their continued drive for better quality in everything that happens at N.I.S.T. and selfless inspiration has always helped us to move ahead.

Ramya Pandey

Shiv Prasad Mishra

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Smart monitoring of the environment has been an essential area of research where the decision-making process is inevitable. Due to the development of technology and increased human activities for sustained life, different environmental parameters are changing day by day. These environmental parameters are temperature, humidity, amount of Carbon monoxide, methane, smoke, etc. Due to different technological and industrial activities, the values of all these parameters are increasing day by day. In this project, by using different sensors and IoT, different environmental parameters have been monitored regularly. Based on the monitoring the sensor values of a particular region can classify into different zone using a machine learning approach. In this project, we have used both simulation and testbed experiments for the validation of the proposed methodology. Here, we made an IoT Based Air Quality Index Monitoring System in which have monitored the Air Quality Index over a Thingspeak server using the internet. We have used MQ135 Air Quality Sensor that can detect the level of various air pollutant. The MQ-135 gas sensor senses the gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulphide and smoke. This is a simple prototype for an Environmental IoT Air Pollution/Quality Monitoring System for monitoring the concentrations of major air pollutant gases. We have also used the BME680 sensor which is an environmental sensor that combines gas, pressure, humidity and temperature sensors. The gas sensor can detect a broad range of gases like volatile organic compounds (VOC). For this reason, the BME680 can be used in indoor air quality control.

1. INTRODUCTION

Now there is advancement of every technology in every field but till now there is no field of advancement in checking environmental condition parameters and taking necessary steps according to the condition of the environment. So, our project is basically collecting different environmental conditions using different sensors and analyzing it and taking necessary steps for controlling the adverse effect of the environmental conditions. The solution used in our project are cheap and can be installed in any place and can be used by any layman.

In this project have collected various parameters of environment like temperature, air quality (like Co2 level, smoke level, carbon monoxide level) of a particular area and pushed the data in cloud.

The data from the sensors have further been used to analyze the air quality of a particular area at a particular time. Our work has been followed in 3 major phases: data acquisition phase, uncertainty handling phase, & decision-making phase [4].

The AQI is an index for reporting daily air quality. It tells us how clean or polluted our air is, and what associated health effects might be a concern for us. The AQI focuses on health affects we may experience within a few hours or days after breathing polluted air.

EPA calculates the AQI for five major air pollutants regulated by the Clean Air Act: ground-level ozone, particle pollution (also known as particulate matter), carbon monoxide, sulfur dioxide, and nitrogen dioxide. For each of these pollutants, EPA has established national air quality standards to protect public health. Ground-level ozone and airborne particles are the two pollutants that pose the greatest threat to human health in this country [4].

2. PROBLEM STATEMENT & OBJECTIVE

Air pollution is one of the environmental issues that can't be ignored. Inhaling pollutants for a long time cause a damage to human health.

The main objective is to inform and caution the public about the risk of exposure to daily pollution levels.

It also comprises of

- Comparing air quality conditions at different situations.
- Analysing the change in air quality (improvement or degradation).

Day to day, the level of Air pollution is increasing rapidly due to increase industries, factories, vehicle use which affect human health. So here we have designed a device/system which can measure air quality around it and monitor air pollution levels and also indicates and warns us when the air quality goes down beyond a certain level. The AQI is an index that tells you how clean or polluted your air is, and what associated health effects might be a concern for us. This system can sense NH3, NOx, alcohol, Benzene, smoke, CO2, and some other gases, these gases are harmful to human health. It has a small display that will show the air quality value in the PPM unit. So, this system is perfect for Air Quality Monitoring. This is a small portable device, we can use it at our home, office, classroom, and factory. It can save us from harmful gases[1].

The level of pollution has increased with times by a lot of factors like the increase in population, increased vehicle use, industrialization and urbanization which results in harmful effects on humans by directly affecting the health of the population exposed to it. So, we need to monitor the Air Quality Index. In this project, have made an IoT Based Air Quality Index Monitoring System in which we will monitor the Air Quality Index over a Thingspeak server using the internet. We have used MQ135 Air Quality Sensor that can detect the level of various air pollutant. We have also used the BME680 sensor which is an environmental sensor that combines gas, pressure, humidity and temperature sensors. The gas sensor can detect a broad range of gases like volatile organic compounds (VOC). For this reason, the BME680 can be used in indoor air quality control.

3. METHODOLOGY

3.1 Requirements:

The key components of the Air Quality Index Monitoring System are ESP8266 Wi-Fi Module, MQ135, OLED Display. The MQ135 is one type of gas sensor that can sense NH3, NOx, alcohol, Benzene, smoke, CO2, and some other gases, these gases are harmful to human health. Arduino is the main microcontroller board of this system. The gas sensor continuously measures air quality and sends data to the Arduino board. Then Arduino Code prints air quality value on the OLED display in the PPM unit. We will monitor the air quality on Thing Speak server using the internet.

All used components:

- Wi-Fi module ESP8266 (NodeMCU)
- MQ135 Gas sensor
- BME680 Module
- 0.96-inch OLED Display
- Connecting Wires
- LEDs (White, Green, Blue and Red)
- 220-ohm resistor
- Buzzer
- Breadboard
- Veroboard

Phases followed:

- Phase - 1: Detection of air pollutant level.
- Phase - 2: Creating the interface.
- Phase - 3: Execution and testing.
- Phase - 4: Analysing the data.
- Phase - 5: Giving conditions as per situations.

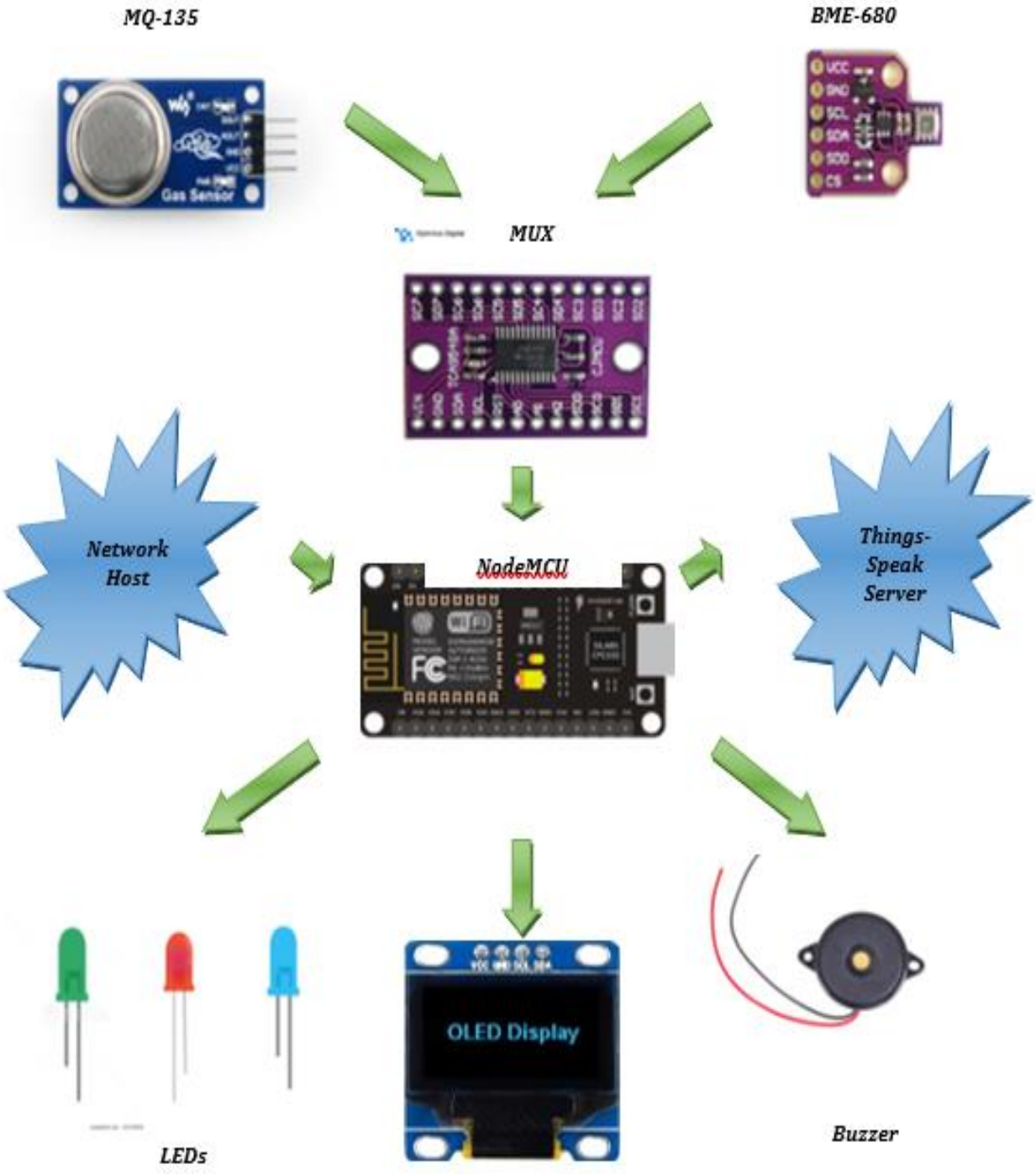


Figure 1 : Block Diagram of Air Quality Index Monitoring System

4. OVERVIEW

- We have interfaced the MQ-135 Sensor and BME680 with NodeMCU (ESP8266 Wi-Fi module), written the code in Arduino Platform and compiled it to feed the results to the OLED Display to show the instant Air Quality Index of a particular place.
- The sensor will gather the data of various environmental parameters and send it to the ThingSpeak server which displays the data online after a particular time interval.
- It acts as a smart solution that can help to mitigate air pollution outdoors and achieve a cleaner and safer environment.
- This is a realization of a scheme to be implemented in other projects of greater level such as weather forecasting, temperature updates.
- This project can prove useful to determine the bad levels or bad quality of air in a real time aspect and warn the people around to take pre-cautionary measures.
- This prototype is only intended to gather data of environment at a particular place at regular time intervals and can be later analysed to deduce conclusive conditions and prepare required precautionary measures.

Air Quality Index Levels of Health Concern	Numerical Value	Meaning
Good	0-50	Air quality is considered satisfactory, and air pollution poses little or no risk.
Moderate	51-100	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
Unhealthy for Sensitive Groups	101-150	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
Unhealthy	151-200	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
Very Unhealthy	201-300	Health alert: everyone may experience more serious health effects.
Hazardous	> 300	Health warnings of emergency conditions. The entire population is more likely to be affected.

Table 1 : Different levels of Air Quality Index (in PPM)

5. TOOLS AND SOURCE PROGRAMS

5.1. Hardware Components

5.1.1. Node MCU (ESP8266 Wi-Fi Module)

Figure 2 : ESP8266 Wi-Fi Module

NodeMCU is a low-cost open source. IoT platform It initially included firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which was based on the ESP-12 module. Later, support for the ESP32 32-bit MCU was added.

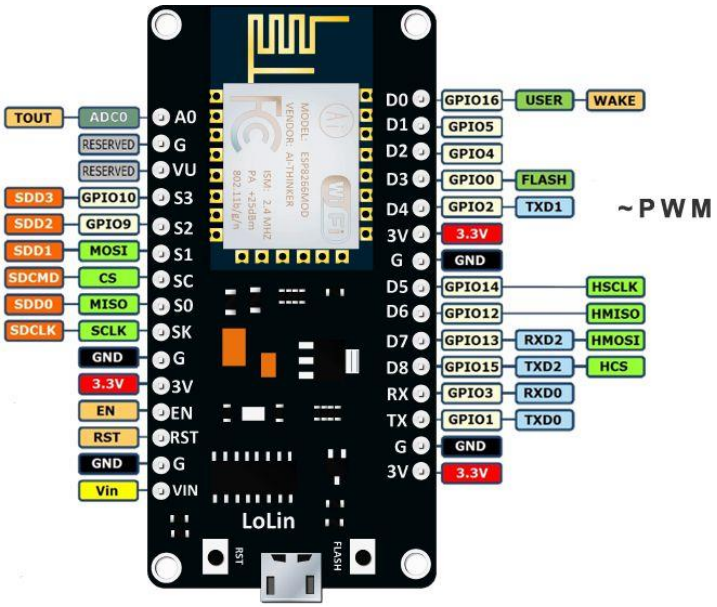


Figure 3 : NodeMCU Pinout

5.1.2. MQ-135 Air Quality Sensor

Figure 4 : MQ-135 Sensor

The MQ-135 gas sensor senses the gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulfide and smoke. The MQ-3 gas sensor has a lower conductivity to clean the air as a gas sensing material. In the atmosphere, we can find polluting gases, but the conductivity of the gas sensor increases as the concentration of polluting gas increases. MQ-135 gas sensor can be implemented to detect the smoke**,** benzene**,** steam and other harmful gases. It has the potential to detect different harmful gases. It is with low cost and particularly suitable for Air quality monitoring application [2].

The MQ135 sensor is a signal output indicator instruction. It has two outputs: analog output and TTL output. The TTL output is low signal light which can be accessed through the IO ports on the Microcontroller. The analog output is a concentration, i.e., increasing voltage is directly proportional to increasing concentration. This sensor has a long life and reliable stability as well [2].

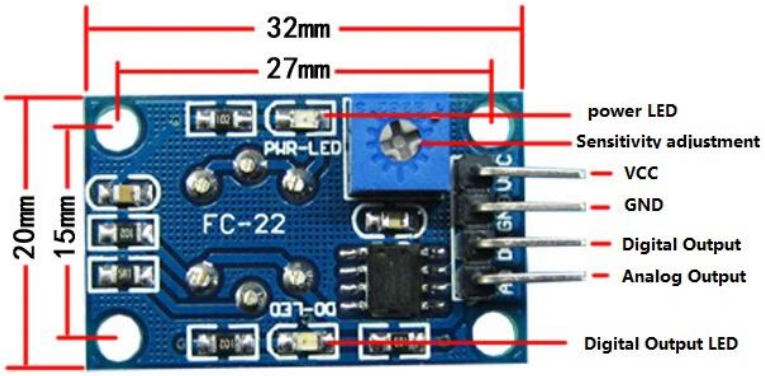


Figure 5 : MQ-135 Pinout

5.1.4. BME680

The BME680 is an environmental sensor that combines gas, pressure, humidity and temperature sensors. The gas sensor can detect a broad range of gases like volatile organic compounds (VOC). For this reason, the BME680 can be used in indoor air quality control.

Figure 6 : BME680 Sensor

5.1.4.1. BME680 Measurements

The BME680 is a 4-in-1 digital sensor that measures:

- Temperature
- Humidity
- Barometric pressure
- Gas: Volatile Organic Compounds (VOC) like ethanol and carbon monoxide
-

5.1.4.2. Gas Sensor

The BME680 contains a MOX (Metal-oxide) sensor that detects VOCs in the air. This sensor gives you a qualitative idea of the **sum of VOCs/contaminants** in the surrounding air – **it is not specific** for a specific gas molecule [3].

MOX sensors are composed of a metal-oxide surface, a sensing chip to measure changes in conductivity, and a heater. It detects VOCs by adsorption of oxygen molecules on its sensitive layer. The BME680 reacts to most VOCs polluting indoor air (except CO2).

When the sensor comes into contact with the reducing gases, the oxygen molecules react and increase the conductivity across the surface. As a raw signal, the BME680 outputs resistance values. These values change due to variations in VOC concentrations:

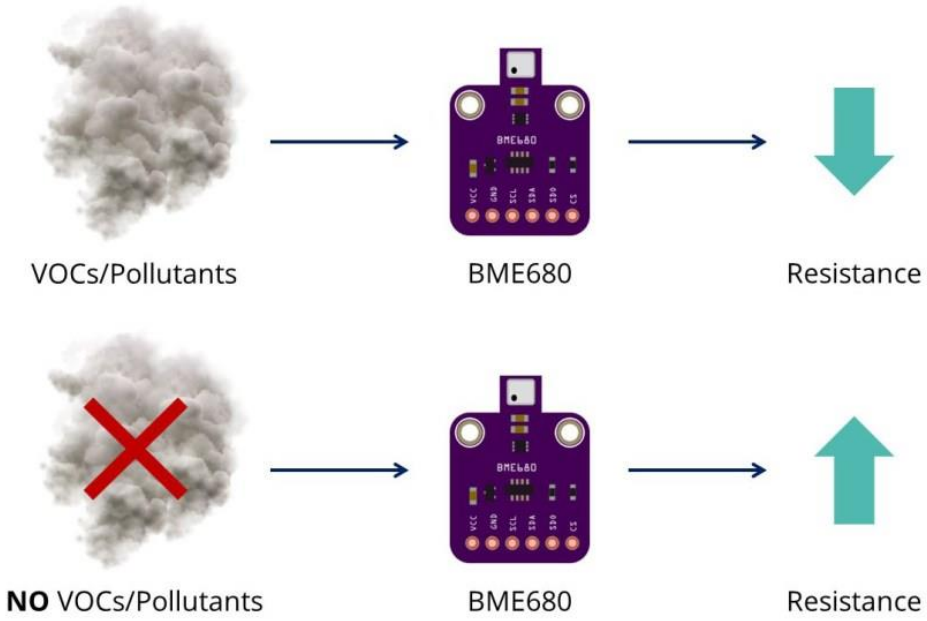


Figure 7 : Working of BME680

- **Higher** concentration of VOCs » **Lower** resistance
- **Lower** concentration of VOCs » **Higher** resistance

The reactions that occur on the sensor surface (thus, the resistance) are influenced by parameters other than VOC concentration like temperature and humidity.

5.1.4.3. Relevant Information Regarding Gas Sensor

The gas sensor gives you a qualitative idea of VOCs gasses in the surrounding air. So, we can get trends, compare your results and see if the air quality is increasing or decreasing. To get precise measurements, we need to calibrate the sensor against knows sources and build a calibration curve [3].

When we first get the sensor, it is recommended to run it for 48 hours after start collecting "real" data. After that, it is also recommended to run the sensor for 30 minutes before getting a gas reading.

5.1.4.4. BME680 Operation Range

The following table shows the operation range and accuracy for the temperature, humidity and pressure sensors for the BME680.

Sensor	Operation Range	Accuracy
Temperature	-40 to 85 ºC	+/- 1.0ºC
Humidity	0 to 100 %	+/- 3%
Pressure	300 to 1100 hPa	+/- 1 hPa

Table 2 : Operation Range and Accuracy for BME680

5.1.4.5. BME680 Pinout

Here's the BME680 Pinout:

VCC	Powers the sensor
GND	Common GND
SCL	SCL pin for I2C communication SCK pin for SPI communication
SDA	SDA pin for I2C communication SDI (MISO) pin for SPI communication

Table 3 : Pinout for BME680

5.1.5. TCA9548A Multiplexer

The I2C communication protocol allows you to communicate with multiple I2C devices on the same I2C bus as long as all devices have a unique I2C address. However, it will not work if you want to connect multiple I2C devices with the same address.

The TCA9548A I2C multiplexer allows you to communicate with up to 8 I2C devices with the same I2C bus. The multiplexer communicates with a microcontroller using the I2C communication protocol. Then, you can select which I2C bus on the multiplexer you want to address.

To address a specific port, you just need to send a single byte to the multiplexer with the desired output port number.

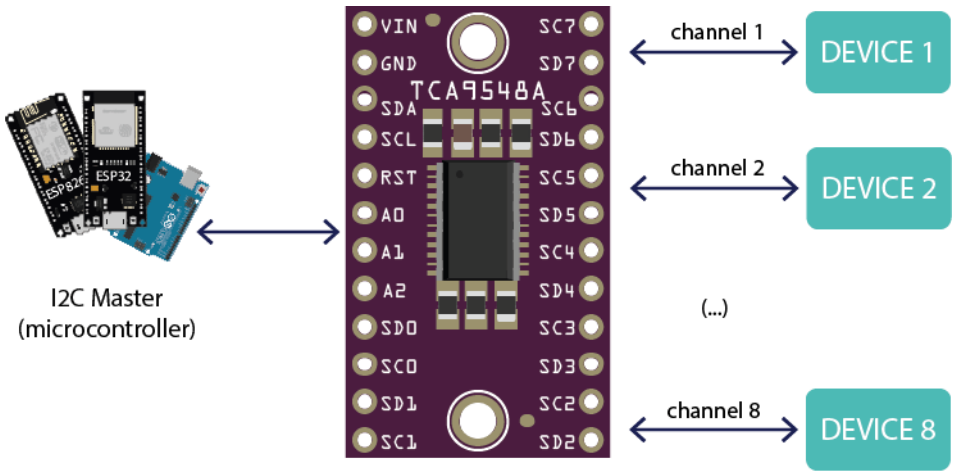


Figure 8 : TCA9548A Multiplexer Working

5.1.5.1. TCA9548A Multiplexer Features

Here's a summary of its main features:

- 1 to 8 bidirectional translating switches.
- Active-low reset input
- Three address pins—up to 8 TCA9548A devices on the same I2C bus
- Channel selection through an I2C bus
- Operating power supply voltage range: 1.65V to 5.5V
- 5V tolerant pins.

5.1.6. 0.96″ I2C OLED Display

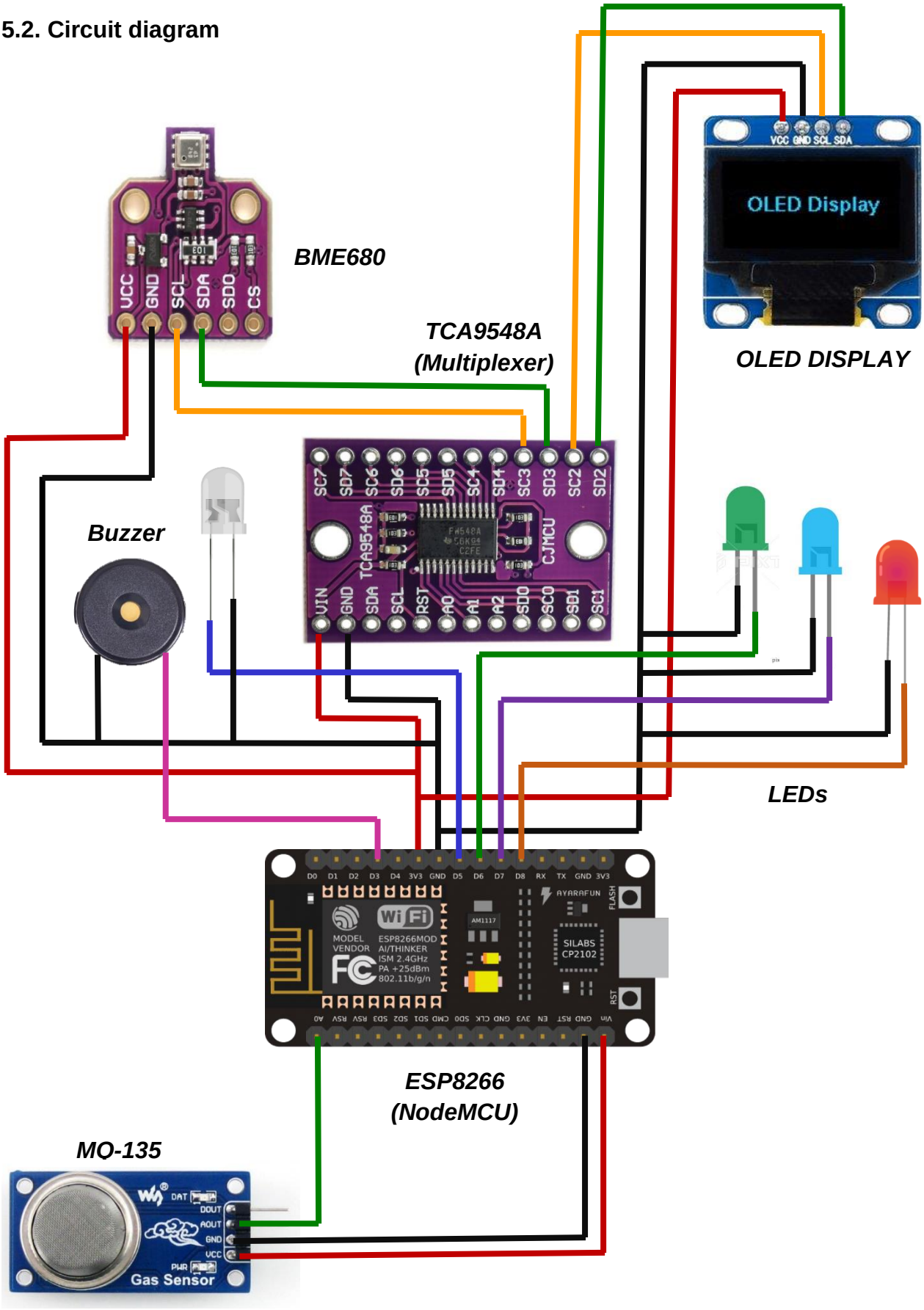
This is a 0.96-inch blue OLED display module. The display module can be interfaced with any microcontroller using SPI/IIC protocols. It is having a resolution of 128x64. The package includes display board, display,4 pin male header pre-soldered to board.

Figure 9 : OLED Display Pinout

OLED (Organic Light-Emitting Diode) is a self-light-emitting technology composed of a thin, multi-layered organic film placed between an anode and cathode.

In contrast to LCD technology, OLED does not require a backlight. OLED possesses high application potential for virtually all types of displays and is regarded as the ultimate technology for the next generation of flat-panel displays.

5.2. Circuit diagram



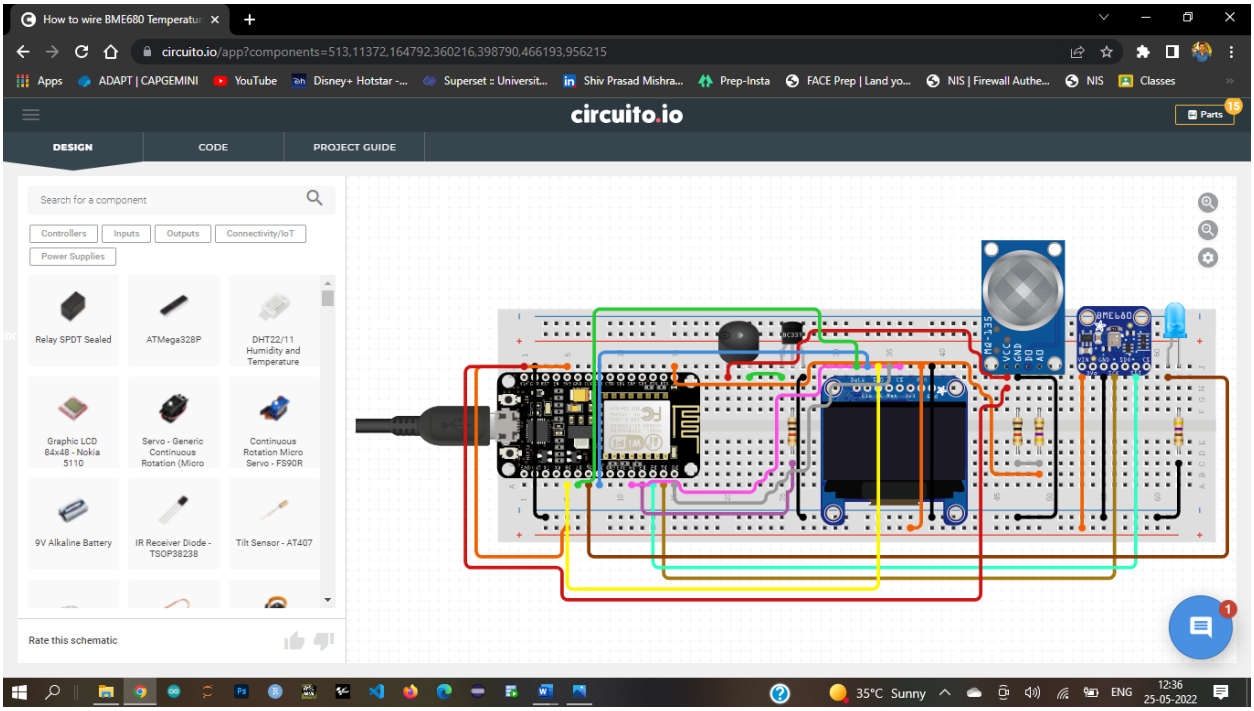


Figure 10 : Circuit Diagram

We have interfaced MQ135 Air Quality Sensor and BME680 with NodeMCU ESP8266 Board and 0.96″ I2C OLED Display. The circuit diagram is given above.

- We have assembled the circuit in a breadboard. First, we have connected the MQ135 Analog input pin to A0 of NodeMCU.

- Then we have connected its VCC and GND to NodeMCU Vin & GND respectively. Similarly, 0.96″ OLED Display and BME680 are I2C Module.

- So, we have connected their SDA & SCL Pin to NodeMCU D2 & D1 Pins and finally connected its VCC to 3.3V GND to GND.

- We have also interfaced the Multiplexer TCA9548A with BME680 and OLED Display to safely transfer data to the Wi-Fi module as they might have different I2C addresses.

6. ARDUINO CODE FOR ESP8266 AND OLED DISPLAY

If our OLED doesn't have a RESET pin, we should set the OLED_RESET variable to -1 as shown below:

#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

Note: if our OLED has a RESET pin, we should connect it to a different GPIO than GPIO.

6.1. Importing libraries

First, we need to import the necessary libraries. The Wire library to use I2C and the Adafruit libraries to write to the display: Adafruit_GFX and Adafruit_SSD1306.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

6.2. Initialize the OLED display

Then, you define our OLED width and height. In this example, we're using a 128×64 OLED display. If you're using other sizes, we can change that in the SCREEN_WIDTH, and SCREEN_HEIGHT variables.

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

Then, initialize a display object with the width and height defined earlier with I2C communication protocol (&Wire).

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

The (-1) parameter means that our OLED display doesn't have a RESET pin. If our OLED display does have a RESET pin, it should be connected to a GPIO. In that case, we should pass the GPIO number as a parameter.

In the setup(), initialize the Serial Monitor at a baud raute of 115200 for debugging purposes.

```
Serial.begin(115200);
Initialize the OLED display with the begin() method as follows:
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println("SSD1306 allocation failed");
  for(;;); // Don't proceed, loop forever
}
```

This snippet also prints a message on the Serial Monitor, in case we're not able to connect to the display.

Serial.println("SSD1306 allocation failed");

In case you're using a different OLED display, you may need to change the OLED address. In our case, the address is 0x3C.

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))

If this address doesn't work, we can run an I2C scanner sketch to find your OLED address.
After initializing the display, add a two second delay, so that the OLED has enough time to initialize before writing text:

delay(2000);

6.3. Clear display, set font size, color and write text

After initializing the display, clear the display buffer with the clearDisplay() method:

display.clearDisplay();

Before writing text, we need to set the text size, color and where the text will be displayed in the OLED.
Set the font size using the setTextSize() method:

display.setTextSize(1);

Set the font color with the setTextColor() method:

display.setTextColor(WHITE);

WHITE sets white font and black background.
Define the position where the text starts using the setCursor(x,y) method. In this case, we're setting the text to start at the (0,10) coordinates.

display.setCursor(0,10);

Finally, we can send the text to the display using the println() method, as follows:

display.println("Hello, world!");

Then, we need to call the display() method to actually display the text on the screen.

display.display();
cause that pin is being used for I2C communication in the EPS8266.

7. ARDUINO CODE FOR ESP8266 AND BME680

7.1. Libraries

The code starts by including the needed libraries: the wire library to use I2C, the SPI library (if we want to use SPI instead of I2C),the Adafruit_Sensor and Adafruit_BME680 libraries to interface with the BME680 sensor.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
```

7.2. SPI communication

We prefer to use I2C communication protocol with the sensor.

```
/*#define BME_SCK 14
#define BME_MISO 12
#define BME_MOSI 13
#define BME_CS 15*/
```

7.3. Sea level pressure

A variable called SEALEVELPRESSURE_HPA is created.

```
#define SEALEVELPRESSURE_HPA (1013.25)
```

This variable saves the pressure at the sea level in hectopascal (is equivalent to milibar). This variable is used to estimate the altitude for a given pressure by comparing it with the sea level pressure. This example uses the default value, but for accurate results, replace the value with the current sea level pressure at your location.

7.4. I2C

This example uses I2C communication protocol by default. The following line creates an Adafruit_BME680 object called bme on the default ESP8266 I2C pins: GPIO 5 (SCL), GPIO 4 (SDA).

```
Adafruit_BME680 bme; // I2C
```

To use SPI, we need to comment this previous line and uncomment the following line.

```
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI
```

7.5. Setup()

In the setup() start a serial communication.

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

7.6. Init BME680 Sensor

Initialize the BME680 sensor:

```
if (!bme.begin()) {
  Serial.println(F("Could not find a valid BME680 sensor, check wiring!"));
  while (1);
}
```

Set up the following parameters (oversampling, filter and gas heater) for the sensor.

```
// Set up oversampling and filter initialization
bme.setTemperatureOversampling(BME680_OS_8X);
bme.setHumidityOversampling(BME680_OS_2X);
bme.setPressureOversampling(BME680_OS_4X);
bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
```

To increase the resolution of the raw sensor data, it supports oversampling. We'll use the default oversampling parameters, but we can change them.

- setTemperatureOversampling(): set temperature oversampling.
- setHumidityOversampling(): set humidity oversampling.
- setPressureOversampling(): set pressure oversampling.
-

The BME680 sensor integrates an internal IIR filter to reduce short-term changes in sensor output values caused by external disturbances. The setIIRFilterSize() method sets the IIR filter. It accepts the filter size as a parameter:

- *BME680_FILTER_SIZE_0 (no filtering)*
- *BME680_FILTER_SIZE_1*
- *BME680_FILTER_SIZE_3*
- *BME680_FILTER_SIZE_7*
- *BME680_FILTER_SIZE_15*
- *BME680_FILTER_SIZE_31*
- *BME680_FILTER_SIZE_63*
- *BME680_FILTER_SIZE_127*

The gas sensor integrates a heater. Set the heater profile using the setGasHeater() method that accepts as arguments:

- *the heater temperature (in degrees Centigrade)*
- *the time the heater should be on (in milliseconds)*

We'll use the default settings: 320 ºC for 150 ms.

7.6. Loop()

In the loop(), we'll get measurements from the BME680 sensor.

First, tell the sensor to start an asynchronous reading with bme.beginReading(). This returns the time when the reading would be ready.

```
// Tell BME680 to begin measurement.
unsigned long endTime = bme.beginReading();
if (endTime == 0) {
  Serial.println(F("Failed to begin reading :("));
  return;
}
Serial.print(F("Reading started at "));
Serial.print(millis());
Serial.print(F(" and will finish at "));
Serial.println(endTime);
```

Then, call the endReading() method to end an asynchronous reading. If the asynchronous reading is still in progress, block until it ends.

```
if (!bme.endReading()) {
  Serial.println(F("Failed to complete reading :("));
  return;
}
```

After this, we can get the readings as follows:

- bme.temperature: returns temperature reading
- bme.pressure: returns pressure reading
- bme.humidity: returns humidity reading
- bme.gas_resistance: returns gas resistance

```
Serial.print(F("Temperature = "));
Serial.print(bme.temperature);
Serial.println(F(" *C"));

Serial.print(F("Pressure = "));
Serial.print(bme.pressure / 100.0);
Serial.println(F(" hPa"));

Serial.print(F("Humidity = "));
Serial.print(bme.humidity);
Serial.println(F(" %"));

Serial.print(F("Gas = "));
Serial.print(bme.gas_resistance / 1000.0);
Serial.println(F(" KOhms"));

Serial.print(F("Approx. Altitude = "));
Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
Serial.println(F(" m"));
```

8. ARDUINO CODE FOR ESP8266 AND TCA9548A

First, import the required libraries to control the OLED display: Adafruit_GFX and Adafruit_SSD1306. The Wire library is needed to use the I2C communication protocol.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Define the OLED width and height.

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
```

Create an Adafruit_SSD1306 instance to communicate with the OLED display.

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

8.1. Select the I2C Channel

The TCA9548A() function can be called to select the bus that you want to communicate with. It sends a byte to the multiplexer with the port number.

```
// Select I2C BUS
void TCA9548A(uint8_t bus){
  Wire.beginTransmission(0x70);  // TCA9548A address
  Wire.write(1 << bus);          // send byte to select bus
  Wire.endTransmission();
  Serial.print(bus);
}
```

You must call this function whenever you want to select the I2C port.

8.2. Setup()

In the setup(), initialize a serial communication for debugging purposes.

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

Start I2C communication on the default I2C pins with the I2C multiplexer.

```
Wire.begin();
```

Then, initialize each display. The following lines show an example for the first OLED display (it is connected to bus number 2).

```
//Init OLED display on bus number 2

TCA9548A(2);
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;);
}

// Clear the buffer
display.clearDisplay();
```

Initializing the other displays is similar, but you need to call the TCA9548A() function with the corresponding I2C bus number.

9. TESTING

9.1. ThingSpeak

ThingSpeak™ is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams. Once you send data to ThingSpeak from our devices, we can create instant visualizations of live data without having to write any code. With MATLAB® analytics inside ThingSpeak, we can write and execute MATLAB code to perform more advanced preprocessing, visualizations, and analyses. Get started with building our IoT systems without setting up servers or developing web software.

The Integration with The Things Network allows us to seamlessly forward data from The Things Network to ThingSpeak for analysis and visualization.

9.2. Setup ThingSpeak

- Create a free MathWorks account or sign into ThingSpeak using an existing account.
- Select the ThingSpeak channel you want your data to stream into. See Collect Data in a New Channel for help creating a new channel.
- Record the following for the selected channel:
- Channel ID, which is listed at the top of the channel view.
- Write API key, which can be found on the API Keys tab of your channel view.

9.3. Create Integration on The Things Network

- In The Things Network Console, go to your application and click on Integrations > add an integration > ThingSpeak.
- Enter your write API key in the Authorization field and your channel ID in the Channel ID field.

9.4. Payload Format

In The Things Network Console use the **Payload Formats** tab in the application for The Things Network to control the payload sent to ThingSpeak.

ThingSpeak can accept up to 8 fields of data per channel, a status field, and three position fields including latitude, longitude, and elevation. Your payload must be a JSON formatted object with at least one of the allowed fields.

As an example of how to work with payload formats, see the sample code for a decoder provided below. This example decodes a 20-character packet and converts it to the format compatible with a ThingSpeak channel. The names of the payload fields must be field1, field2, … as in the sample code else ThingSpeak will get null values.

9.5. Visualize, Analyze, and Act

Use your channel view to display a summary of the channel fields. Add specialized visualizations using the MATLAB Visualizations App or pre-built widgets. Use MATLAB analysis, React, TalkBack, and TimeControl to act on your data.

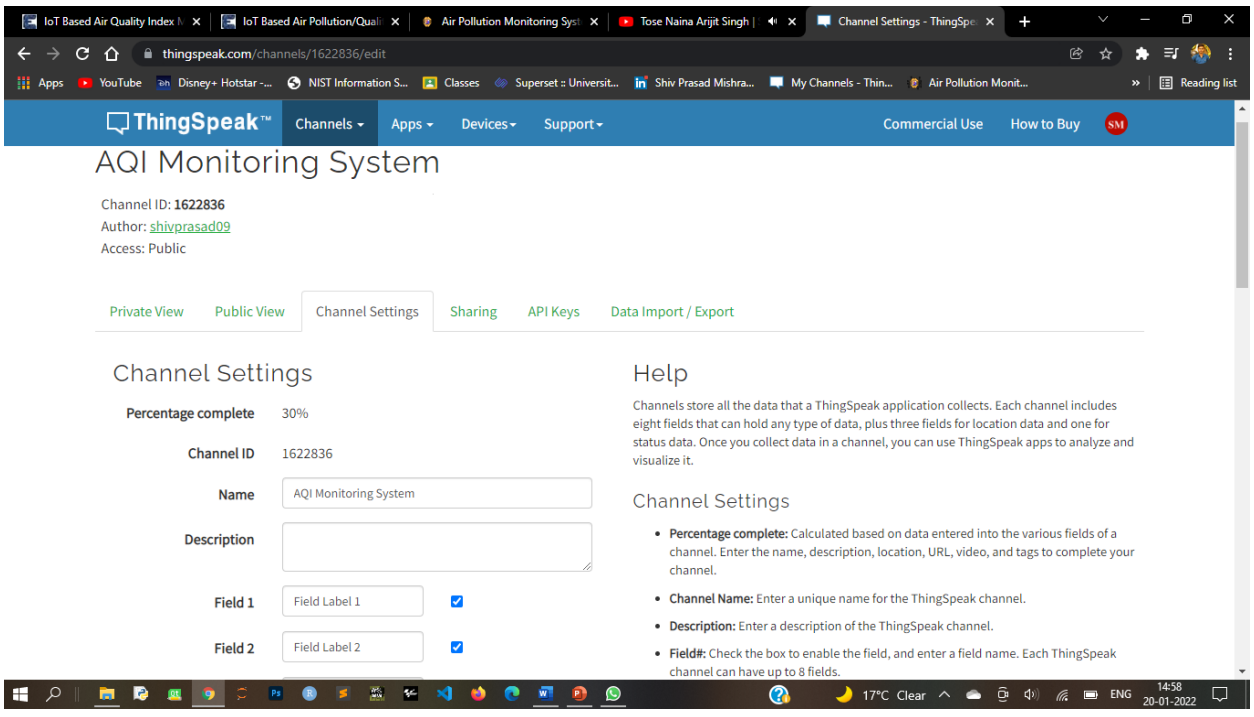


Figure 11 : Setting up Thingspeak Server

10. FINAL OBSERVED RESULTS

In Arduino IDE, click Tools > Board and select ESP8266 NodeMCU.

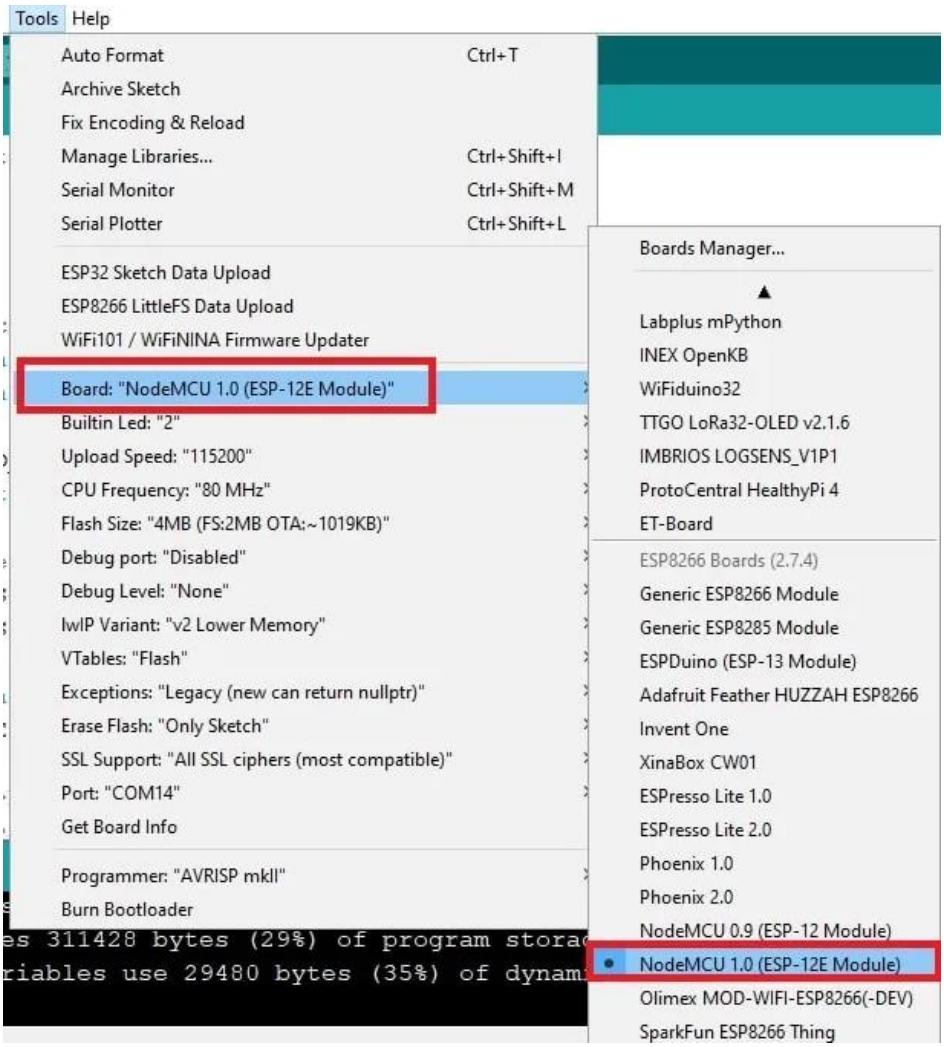


Figure 12 : Selecting Board for Code Upload

Now, click Tools > Port and choose the port which you are using. Now, upload the code by clicking on the upload button.

After you have uploaded the following code on your ESP8266 NodeMCU development board, press the ENABLE button as follows:

Figure 13 : Reset Button of NodeMCU

Once the code is uploaded, we can open serial monitor. The NodeMCU will first start connecting to Wi-Fi network. All the happening can be observed on Serial Monitor.

Open the Serial Monitor at a baud rate of 115200, press the on-board RST button. The sensor measurements will be displayed.

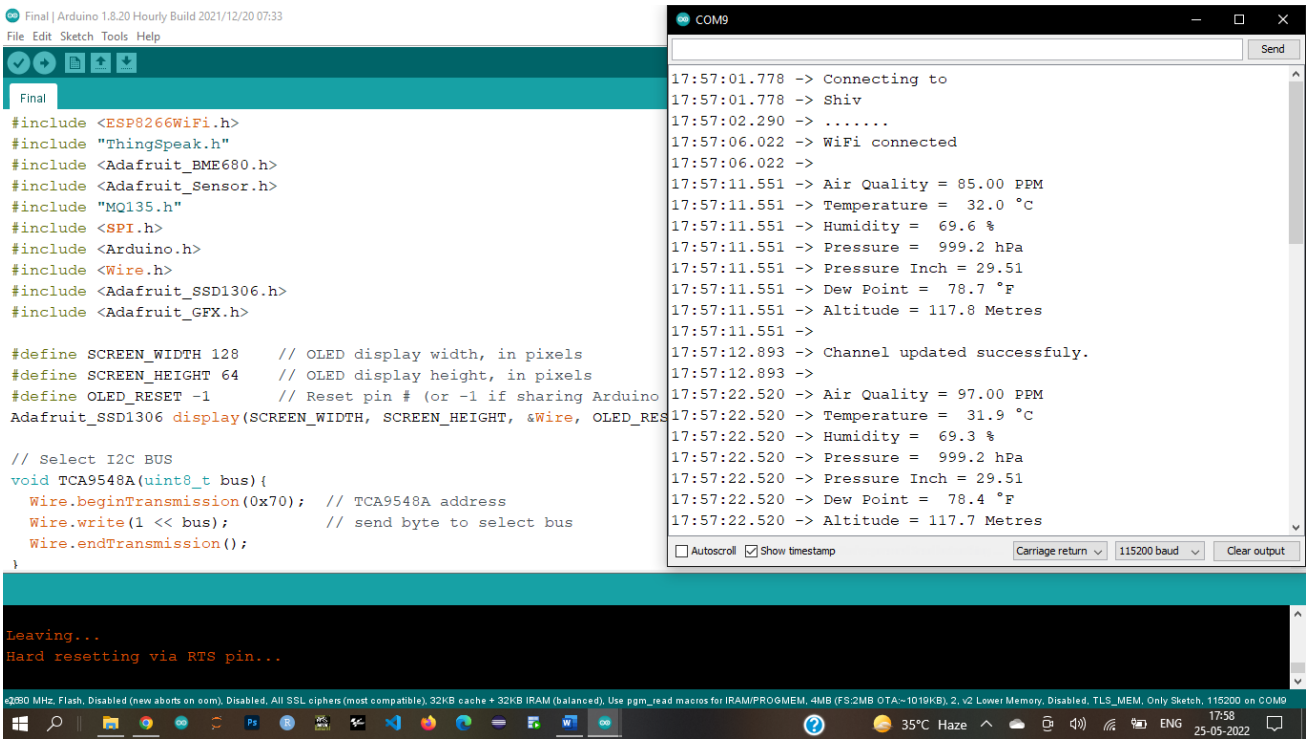


Figure 14 : Observations on Serial Monitor

Once connected to a Wi-Fi network, the sensor will read the value and the value will be displayed on OLED Screen.

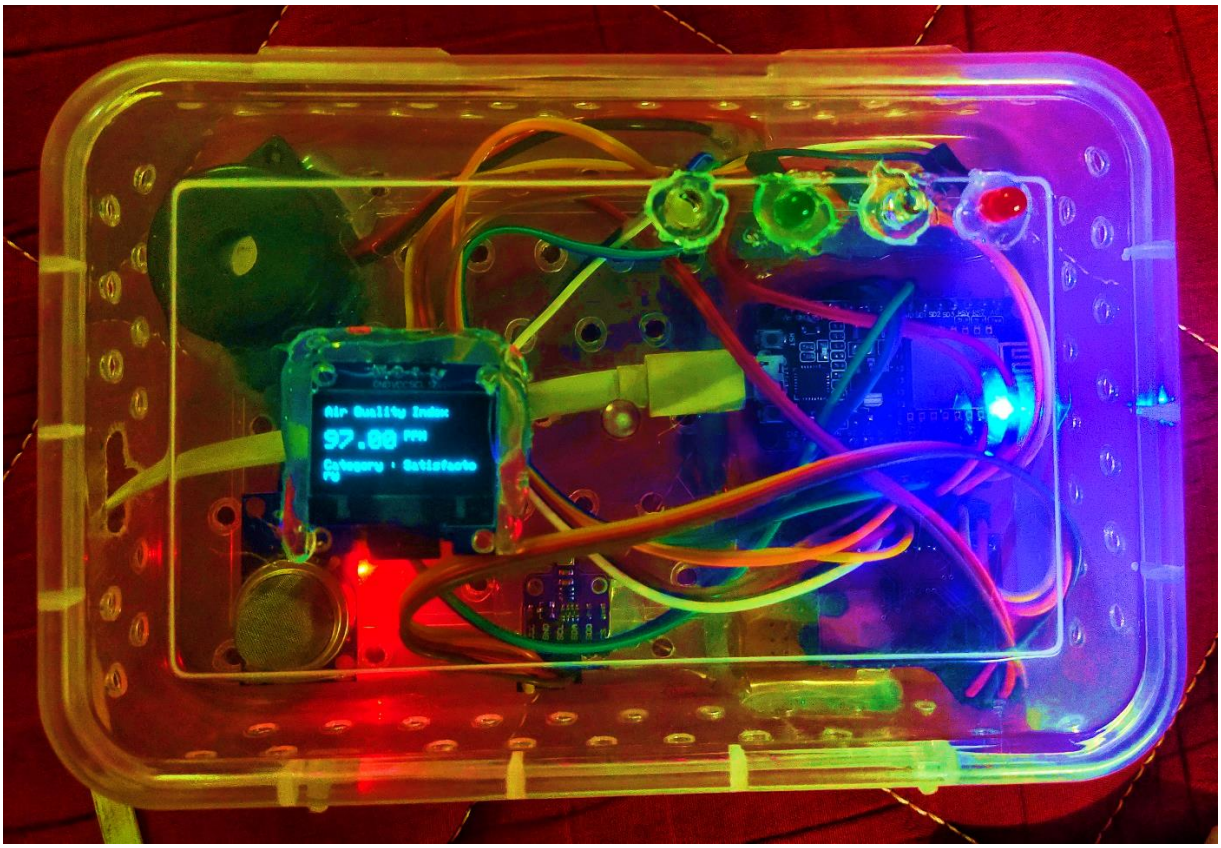


Figure 15 : Displaying AQI & Other Parameters on OLED Display

Similarly, we can see the online data of Air Quality Index on Thingspeak Server. We can just go to Thingspeak Private view and check the data being uploaded after the interval of 15 seconds.

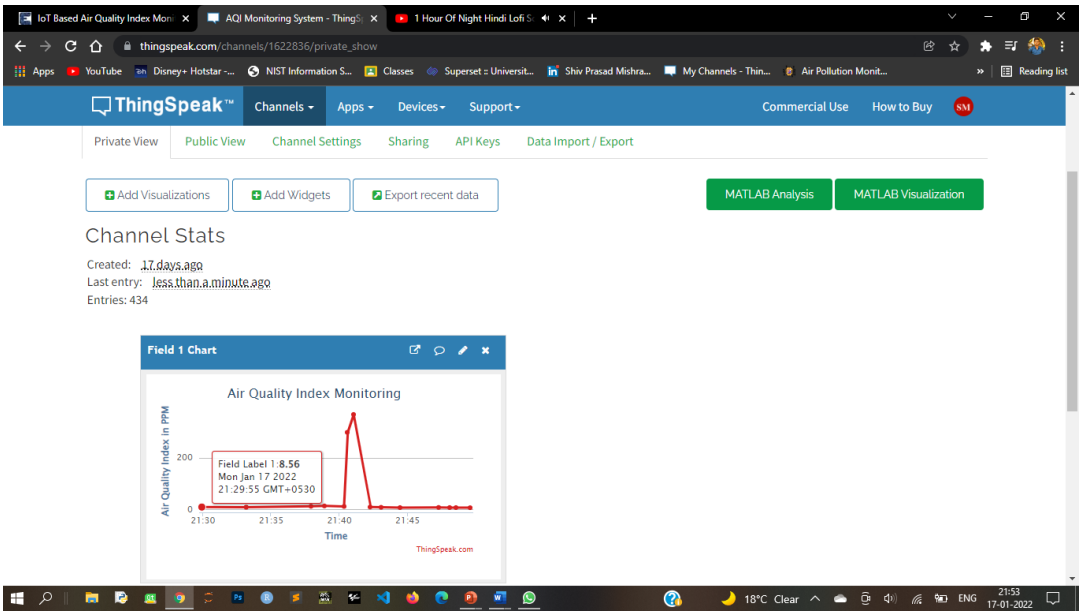


Figure 16 : Results on ThingSpeak Server

The spike in the graph clearly shows that there has been a detection of toxic gas (or one of the five major air pollutants regulated by the Clean Air Act: ground-level ozone, particle pollution (also known as particulate matter), carbon monoxide, sulfur dioxide, and nitrogen dioxide).

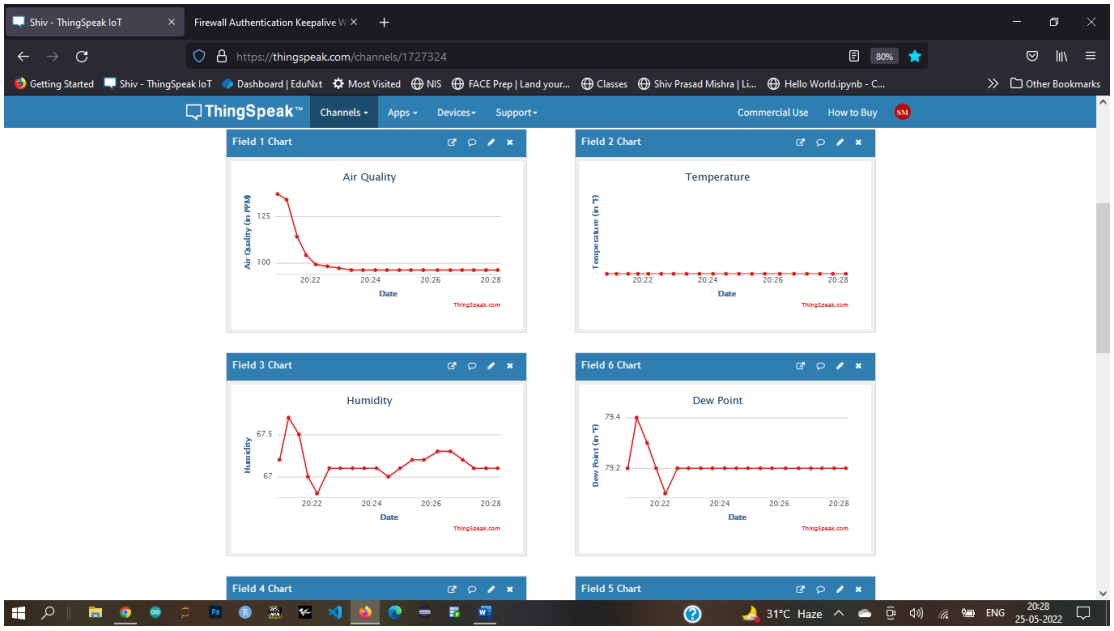


Figure 17 : More Parameters on Thingspeak

11. MATLAB VISUALIZATIONS ON THINGSPEAK

11.1. MATLAB Visualizations App

Visualize data from a ThingSpeak™ channel using MATLAB® functions and toolboxes listed in Access MATLAB Add-On Toolboxes. You can view and explore data using interactive or static visualizations. You can also make the visualizations public and use the URL to embed them on websites. Create these interactive visualizations using the MATLAB Visualizations app:

- Area plot
- Line plot
- Scatter plot
- Stem plot
- Plot with two stacked Y-axes

You also have access to many more display options from the MATLAB Plot Gallery.

11.2. Visualize Data with MATLAB

1. Click **Apps** > **MATLAB Visualizations**.
2. Click **New** to start your visualization.
3. Select a template or an example with sample code, which you can run and explore the results.
4. Click **Create**.

11.3. MATLAB Visualizations Settings

- **Name**: Enter a unique name for your visualization. Press enter, or click outside the name box anytime you change the name and the stored name of your visualization is automatically updated.
- **MATLAB Code**: Enter custom code, or modify the sample code with your data.
- **Create a public URL**: Check this box to make this visualization public and generate a public URL to share your MATLAB visualization. If this box is

checked, you can also add the visualization to the public view of your public channels.

- **Auto Update**: Execute MATLAB code and update currently visible visualizations every five minutes. The visualizations you can automatically update include:
    - Visualizations added to the channel view
    - Visualizations opened in view mode

  This check box is visible only to users with a paid ThingSpeak license.
- **Save and Run**: Click to save and run your visualization.
- **Save**: Click to save your visualization without running the code. An asterisk on this button indicates unsaved changes.
- **MATLAB Plot Output**: This field displays your visualization.
- **Output**: This field displays the output of your code. Use it to debug and modify your code.
- **Clear Output**: Click to clear the visualization and the output.
- **Display Settings**: You can add the visualization to your channel view. Expand **Add/Edit this Visualization to a Channel** to see a list of your channels.
    - To add the visualization to the private channel view, select **Private View**.
    - To show the visualization on a public view of a channel, make sure that the channel is public, and enable **Create a public URL**. Then select **Public View** for the channels you want to see the visualization on.
    - To update your selections, click **Save Display Settings**. All selected channels are shown in the section **Show on Channel(s)**.
- **Delete**: Click to delete the visualization. Deleting a visualization also deletes it from your channel view.

- **My Channels** (right side of page) See information about your saved channels, including:
    - Channel name
    - Channel ID
    - Access
    - Write and Read API Keys
    - Channel fields

11.4. VISUALIZATION 1

Use Histogram to Understand Variation in Data for temperature values read from a public channel. This example reads 10 hours of temperature data and shows the variation in a histogram.

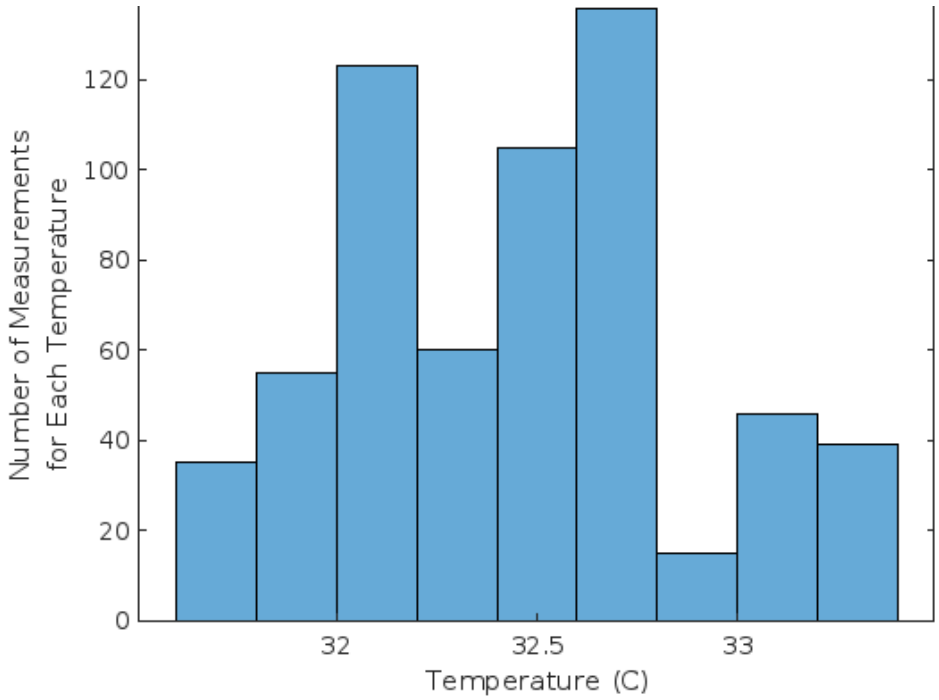


Figure 18 : Histogram of Temperature Variation

11.5. VISUALIZATION 2

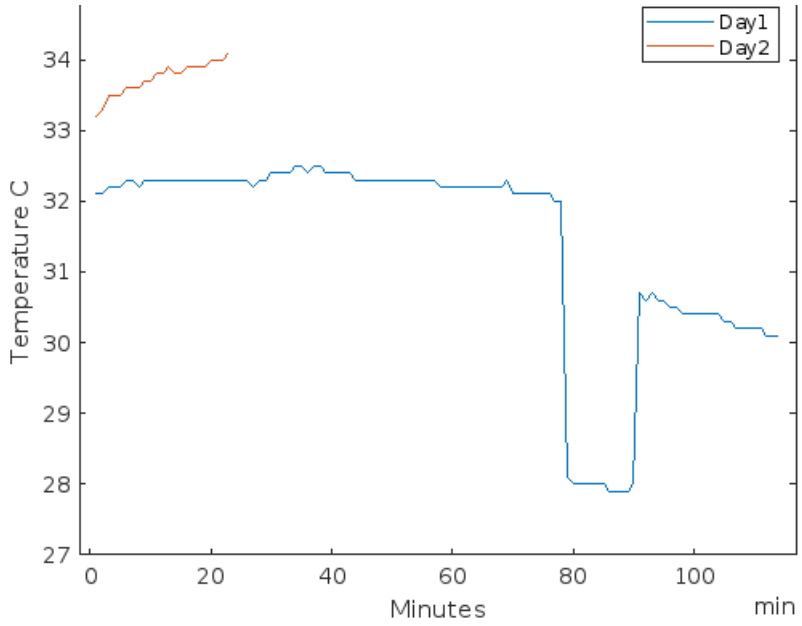


Figure 19 : 2 - Day Temperature Comparison

11.6. VISUALIZATION 3

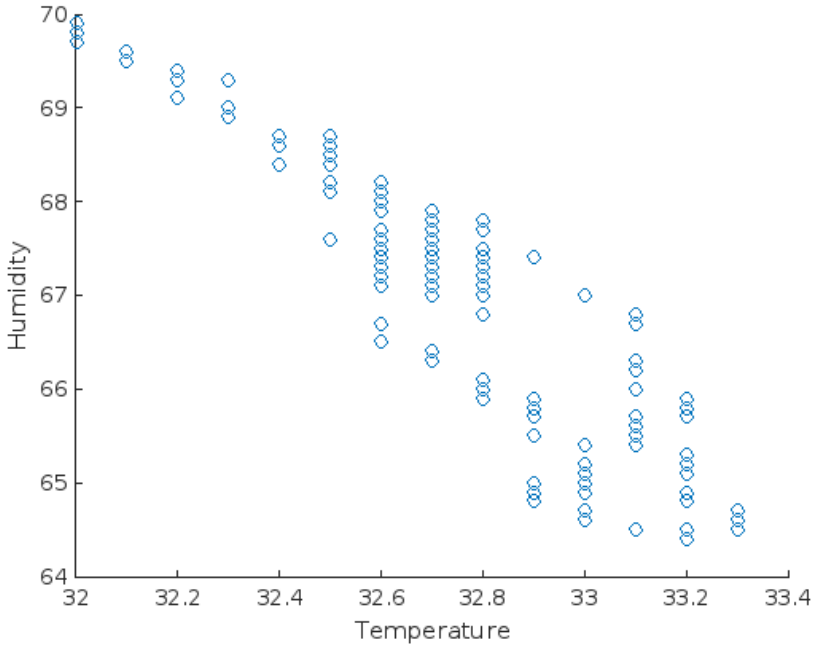


Figure 20 : Visualize Correlation between Temperature and Humidity 3

12. CONCLUSION

The sensors all combined together will gather the data of various environmental parameters and send it to the Thing Speak server which displays the data online after a particular time interval. It is a smart and cheap solution that can help to mitigate air pollution outdoors and achieve a cleaner and safer environment. It is a realization of a scheme to be implemented in other projects of greater level such as weather forecasting, temperature updates [2].

Day to day, the level of Air pollution is increasing rapidly due to increase industries, factories, vehicle use which affect human health. So here we have designed a device/system which can measure air quality around it and monitor air pollution levels and also indicates and warns us when the air quality goes down beyond a certain level. This system can sense NH3, NOx, alcohol, Benzene, smoke, CO2, and some other gases, these gases are harmful to human health. It has a small display that will show the air quality value in the PPM unit. So, this system is perfect for Air Quality Monitoring. This is a small portable device, we can use it at our home, office, classroom, and factory. It can save us from harmful gases.

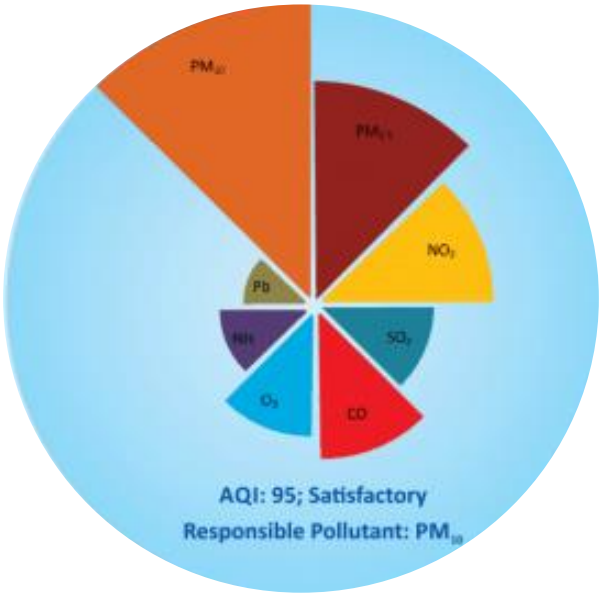


Figure 21 : A pi-chart showing the AQI of a place.

13. REFERENCES

[1] IoT Based Air Pollution Monitoring System, Anand Jayakumar , Praviss Yesyand T, Venkstesh Prashanth K ,International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 ,Volume: 08 Issue: 03 | Mar 2021 www.irjet.net p-ISSN: 2395-0072.

[2] IOT Based Air Quality Monitoring System Using MQ135, Kinnera Bharath Kumar Sai, Somula Ramasubbareddy And Ashish Kr. Luhach, Scalable Computing: Practice and Experience, Volume 20, Number 4, pp. 599–606.

[3] Galadima, A.A., "Arduino as a learning tool," in Electronics, Computer and Computation (ICECCO), 2014 11th International Conference on , vol., no., pp.1-4, Sept. 29 2014-Oct. 1 2014 doi: 10.1109/ICECCO.2014.6997577

[4] Air Quality Measurement in Buildings, Attila LAKATOS, Magazine of Hydraulics, Pneumatics, Tribology, Ecology, Sensorics, Mechatronics, ISSN 1453 – 7303 “HIDRAULICA” (No. 4/2021).

[5] https://www.arduino.cc/reference/en/libraries/dht-sensor-library/

[6] https://components101.com/sensors/mq135-gas-sensor-for-air-quality

[7] https://www.arduino.cc/en/reference/libraries

14. APPENDICES

14.1. APPENDIX A

```
#include <ESP8266WiFi.h>
#include "ThingSpeak.h"
#include <Adafruit_BME680.h>
#include <Adafruit_Sensor.h>
#include "MQ135.h"
#include <SPI.h>
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

void TCA9548A(uint8_t bus){
 Wire.beginTransmission(0x70); // TCA9548A address
 Wire.write(1 << bus);
 Wire.endTransmission();
}

const char* ssid = "Shiv";
const char* pass = "14091999";
WiFiClient client;
unsigned long myChannelNumber = 1727324;
const char * myWriteAPIKey = "B33EX8ABC85NY7U7";
float h, tC, tF, p, prin, dp, alt;
```

```
float air_quality, air_quality_a;
char temperatureString[6];
char dpString[6];
char altString[6];
char humidityString[6];
char pressureString[7];
char pressureInchString[6];
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME680 bme;
int buzz = 2;
int LED1 = 14;
int LED2 = 12;
int LED3 = 13;
int LED4 = 15;
void setup() {

 Wire.begin();
 TCA9548A(3);
 Serial.begin(115200);

 if (!bme.begin()) {
 Serial.println(F("Could not find a valid BME680 sensor, check wiring!"));
 for(;;);
 }

 bme.setTemperatureOversampling(BME680_OS_8X);
 bme.setHumidityOversampling(BME680_OS_2X);
 bme.setPressureOversampling(BME680_OS_4X);
 bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
 bme.setGasHeater(320, 150);
```

```
 WiFi.mode(WIFI_STA);
 ThingSpeak.begin(client);

 TCA9548A(2);
 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
 Serial.println(F("SSD1306 allocation failed"));
 for(;;);
 }

display.clearDisplay();

 display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
 display.clearDisplay();
 delay(10);
 display.clearDisplay();
 display.setTextSize(2);
 display.setTextColor(WHITE);
 display.setCursor(0,0);
 display.println("Project ID : 22303");
 display.display();
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,45);
 display.println("AQI Monitoring System");
 display.display();
 delay(7000);
 Serial.println("Connecting to ");
 Serial.println(ssid);

 display.clearDisplay();
 display.setCursor(0,0);
```

```
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.println("Connecting to ");
 display.setTextSize(2);
 display.print(ssid);
 display.display();

 WiFi.begin(ssid, pass);
 while (WiFi.status() != WL_CONNECTED)
 {
 delay(500);
 Serial.print(".");
 }
 Serial.println("");
 Serial.println("WiFi connected");
 Serial.println();
 display.clearDisplay();
 display.setCursor(0,0);
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.print("WiFi connected");
 display.display();
 delay(4000);
 pinMode(LED1, OUTPUT);
 pinMode(LED2, OUTPUT);
 pinMode(LED3, OUTPUT);
 pinMode(LED4, OUTPUT);
 pinMode(buzz, OUTPUT);
}
void loop() {
 if(WiFi.status() != WL_CONNECTED){
```

```
Serial.print("Attempting to connect");
while(WiFi.status() != WL_CONNECTED){
WiFi.begin(ssid, pass);
delay(5000);
}
Serial.println("\nConnected.");
}
MQ135 gasSensor = MQ135(A0);

air_quality_a = analogRead(A0);
air_quality = air_quality_a/1;
TCA9548A(3);
h = bme.readHumidity();
tC = bme.readTemperature();
tF = tC*1.8+32.0;
dp = tF-0.36*(100.0-h);
alt = bme.readAltitude(SEALEVELPRESSURE_HPA);
p = bme.readPressure()/100.0F;
prin = 0.02953*p;

dtostrf(tC, 5, 1, temperatureString);
dtostrf(h, 5, 1, humidityString);
dtostrf(p, 6, 1, pressureString);
dtostrf(prin, 5, 2, pressureInchString);
dtostrf(dp, 5, 1, dpString);
dtostrf(alt, 5, 1, altString);

Serial.print("Air Quality = ");
Serial.print(air_quality);
Serial.println(" PPM");
Serial.print("Temperature = ");
```

```
Serial.print(temperatureString);
Serial.println(" °C");
Serial.print("Humidity = ");
Serial.print(humidityString);
Serial.println(" %");
Serial.print("Pressure = ");
Serial.print(pressureString);
Serial.println(" hPa");
Serial.print("Pressure Inch = ");
Serial.println(pressureInchString);
Serial.print("Dew Point = ");
Serial.print(dpString);
Serial.println(" °F");
Serial.print("Altitude = ");
Serial.print(altString);
Serial.println(" Metres");
Serial.println();

ThingSpeak.setField(1, air_quality);
ThingSpeak.setField(2, temperatureString);
ThingSpeak.setField(3, humidityString);
ThingSpeak.setField(4, pressureString);
ThingSpeak.setField(5, pressureInchString);
ThingSpeak.setField(6, dpString);
ThingSpeak.setField(7, altString);

int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if(x==200){
Serial.println("Channel updated successfuly.");
Serial.println();
}
```

```
TCA9548A(2);
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Air Quality Index");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(air_quality);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" PPM");
display.display();
if (air_quality<=50)
{
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);

display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,45);
display.println("Category : Good");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);
```

```
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Temperature");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(tC);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" deg. C");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);

display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Humidity");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(h);
display.setTextSize(2);
display.setTextColor(WHITE);
```

```
display.println(" %");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
}

else if( air_quality>50 && air_quality<=100 )
{

digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);

display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,45);
display.println("Category : Satisfactory");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Temperature");

display.setCursor(0,20);
```

```
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(tC);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" deg. C");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
delay(1000);

display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Humidity");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(h);
display.setTextSize(2);
display.setTextColor(WHITE);
display.println(" %");
display.display();
digitalWrite(LED1, HIGH);
delay(1500);
digitalWrite(LED1, LOW);
}
else if( air_quality>100 && air_quality<=200 )
```

```
{

 digitalWrite(LED2, HIGH);
 delay(1000);
 digitalWrite(LED2, LOW);
 delay(750);

 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,45);
 display.println("Category : Moderate");
 display.display();
 digitalWrite(LED2, HIGH);
 delay(1000);
 digitalWrite(LED2, LOW);
 delay(750);
 display.clearDisplay();
 display.setCursor(0,0);
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.println("Temperature");

 display.setCursor(0,20);
 display.setTextSize(2);
 display.setTextColor(WHITE);
 display.print(tC);
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.println(" deg. C");
 display.display();
 digitalWrite(LED2, HIGH);
```

```
delay(1000);
digitalWrite(LED2, LOW);
delay(750);

display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Humidity");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(h);
display.setTextSize(2);
display.setTextColor(WHITE);
display.println(" %");
display.display();
digitalWrite(LED2, HIGH);
delay(1000);
digitalWrite(LED2, LOW);
}
else if( air_quality>200 && air_quality<=300 )
{

digitalWrite(LED3, HIGH);
digitalWrite(buzz, HIGH);
delay(750);
digitalWrite(LED3, LOW);
digitalWrite(buzz, LOW);
delay(750);
```

```
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,45);
display.println("Category : Poor");
display.display();
digitalWrite(LED3, HIGH);
digitalWrite(buzz, HIGH);
delay(750);
digitalWrite(LED3, LOW);
digitalWrite(buzz, LOW);
delay(750);
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Temperature");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(tC);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" deg. C");
display.display();
digitalWrite(LED3, HIGH);
digitalWrite(buzz, HIGH);
delay(750);
digitalWrite(LED3, LOW);
digitalWrite(buzz, LOW);
```

```
delay(750);

display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Humidity");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(h);
display.setTextSize(2);
display.setTextColor(WHITE);
display.println(" %");
display.display();
digitalWrite(LED3, HIGH);
digitalWrite(buzz, HIGH);
delay(750);
digitalWrite(LED3, LOW);
digitalWrite(buzz, LOW);
}
else if( air_quality>300 && air_quality<=400 )
{
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(500);
```

```
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0,45);
display.println("Category : Severe");
display.display();
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(500);
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Temperature");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(tC);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" deg. C");
display.display();
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(500);
```

```
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Humidity");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(h);
display.setTextSize(2);
display.setTextColor(WHITE);
display.println(" %");
display.display();
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
}
else if (air_quality>400 )
{
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(350);

display.setTextSize(1);
```

```
display.setTextColor(WHITE);
display.setCursor(0,45);
display.println("Category : Hazardous");
display.display();
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(350);
display.clearDisplay();
display.setCursor(0,0);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println("Temperature");

display.setCursor(0,20);
display.setTextSize(2);
display.setTextColor(WHITE);
display.print(tC);
display.setTextSize(1);
display.setTextColor(WHITE);
display.println(" deg. C");
display.display();
digitalWrite(LED4, HIGH);
digitalWrite(buzz, HIGH);
delay(500);
digitalWrite(LED4, LOW);
digitalWrite(buzz, LOW);
delay(350);
```

```
 display.clearDisplay();
 display.setCursor(0,0);
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.println("Humidity");

 display.setCursor(0,20);
 display.setTextSize(2);
 display.setTextColor(WHITE);
 display.print(h);
 display.setTextSize(2);
 display.setTextColor(WHITE);
 display.println(" %");
 display.display();
 digitalWrite(LED4, HIGH);
 digitalWrite(buzz, HIGH);
 delay(500);
 digitalWrite(LED4, LOW);
 digitalWrite(buzz, LOW);
 }

}
```

14.2. APPENDIX B

```
% Read temperature for the last 10 hours from a ThingSpeak channel and
% Visualize temperature variations using the MATLAB HISTOGRAM function.

% Channel ID to read data from
readChannelID = 1727324;

% Temperature Field ID
TemperatureFieldID = 2;

% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = '38996GR5FPNYAEZ2';


% Get temperature data from field 4 for the last 10 hours = 10 x 60 minutes.

tempF = thingSpeakRead(readChannelID,'Fields',TemperatureFieldID,...
'NumMinutes',10*60, 'ReadKey',readAPIKey);

histogram(tempF);
xlabel('Temperature (C)');
ylabel('Number of Measurements\newline for Each Temperature');
title('Histogram of Temperature Variation');
```

14.3. APPENDIX C

% Read temperature data from a ThingSpeak channel for three seperate days and visualize the data in a single plot using the PLOT function.

```
% Channel ID to read data from
readChannelID = 1727324;
% Temperature Field ID
myFieldID = 2;
% One day date range
oneDay = [datetime('yesterday') datetime('today')];

% Channel Read API Key
% If your channel is private, then enter the read API key between the '' below:
readAPIKey = '38996GR5FPNYAEZ2';

% Read Temperature Data. Learn more about the THINGSPEAKREAD function by
% going to the Documentation tab on the right side pane of this page.
temperatureDay1 = thingSpeakRead(readChannelID,'Fields',myFieldID, ...
                        'dateRange', oneDay, 'ReadKey',readAPIKey);
temperatureDay2 = thingSpeakRead(readChannelID,'Fields',myFieldID, ...
                        'dateRange',oneDay-days(1),'ReadKey',readAPIKey);
temperatureDay3 = thingSpeakRead(readChannelID,'Fields',myFieldID, ...
                       'dateRange', oneDay-days(2),'ReadKey',readAPIKey);

% Create array of durations
myTimes1 = minutes(1:length(temperatureDay1));
myTimes2 = minutes(1:length(temperatureDay2));
myTimes3 = minutes(1:length(temperatureDay3));

% Visualize the data
plot(myTimes1,temperatureDay1, myTimes2,temperatureDay2, myTimes3,
temperatureDay3);
legend({'Day1','Day2','Day3'});
xlabel('Minutes');
ylabel('Temperature C');
title('3-Day Temperature Comparison');
```

14.4. APPENDIX D

```
% Read temperature and humidity from a ThingSpeak channel and visualize the
relationship between them using the SCATTER plot


% Channel ID to read data from
readChannelID = 1727324;
% Temperature Field ID
TemperatureFieldID = 2;
% Humidity Field ID
HumidityFieldID = 3;


% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = '38996GR5FPNYAEZ2';


% Read Temperature and Humidity Data.


data = thingSpeakRead(readChannelID,'Fields',[TemperatureFieldID HumidityFieldID],
...
                                    'NumPoints',300, ...
                                    'ReadKey',readAPIKey);


temperatureData = data(:,1);


% Read Humidity Data
humidityData = data(:,2);


% Visualize the data
scatter(temperatureData,humidityData);
xlabel('Temperature');
```

14.5. APPENDIX E

Sl. No.	Item Description	Quantity	Item Price	Total
1.	ESP8266 (NodeMCU)	2	400	800
2.	MQ-135	1	180	180
3.	BME680	1	1700	1700
4.	OLED Display	1	280	280
5.	TCA9548A MUX	1	300	300
6.	Breadboard	1	80	80
7.	220-Ohm Resistors	10	1	10
8.	Jumper Wires	20	5	100
9.	Tape	1	30	30
10.	Plastic Box	2	25	50
11.	LEDs	6	1	6
12.	Buzzer	1	30	30
13.	Vero board (4'' x 2'')	1	40	40
14.	Solder	1	75	75
15.	Glue Stick	2	20	40
				Total : Rs. 3721/-