NOISE POLLUIION MONITORING

ABSTRACT

Modernization has brought the world technological advancements, but it has also brought with it a slew of problems. In today's Malaysia, air and noise pollution are becoming more of a concern, along with a rise in occupational disease. A monitoring system is needed to address these issues. This paper describes the development of a real-time IoT-based air and noise pollution monitoring system that can provide monitoring and alert the user to the pollution levels. This monitoring system was built using IoT technology, which included the use of an ESP8266 Wi-Fi Module NodeMCU as a microcontroller to communicate with the chosen IoT analytics platform, ThingSpeak. A gas sensor MQ9 was used to measure carbon monoxide concentrations, and a sound sensor LM393 was used to measure noise levels in the surrounding area. The measured values were displayed on the Arduino software's serial monitor, then sent to the ThingSpeak server and graphically displayed in real time on a screen. The results of the electronic sensors were compared to the results of the stand-alone carbon monoxide meter and digital sound level meter for validation. The proposed monitoring system produced promising results, with 91.12 % and 97.86 % accuracy for gas and sound levels shown by the gas sensor MQ9 and sound sensor LM393, respectively. The framework also provides ThingSpeak server warning messages. When the calculated conditions exceeded the user's defined cap, the server sent the user an email update with the gas and noise limit status. This has made the system more useful and convenient

INTRODUCTION

Malaysia's industrial growth has accelerated in recent years. This raises pollution levels, especially in industrial areas. Major industries affecting the air quality are the iron and steel industry, nonferrous metal industry, non-metallic

(mineral) industry, oil and gas industry, petrochemical industry, pulp and paper, power plant, and waste incineration sector [1]. Each of the pollutant has different trait. For instance, Particulate matter (PM) can cause lung cancer and cardiopulmonary deaths while ozone (O3) can reduce lung cancer function and induce coughing and choking [2]. Carbon monoxide (CO) can cause mortal growth in pregnant women as well as affect tissue development of young children [2]. The concentration of carbon monoxide and the duration of exposure will determine the probability of health risk. The toxic effects range from subtle cardiovascular and neurobehavioral effects at low concentrations to unconsciousness and death after prolonged exposures to high concentrations of carbon monoxide [3]. A non-irritating gas such as nitric oxide (NO) may irritate respiratory infections with symptoms such as cough, sore throat, nasal congestion, and fever while sulfur dioxide (SO2) can cause shortness of breath in people with asthma [4]. According to the Ministry of Health Malaysia (MoH), respiratory system diseases were one of the leading causes of hospitalization in MoH hospitals in 2011, accounting for 10.36% of all hospitalizations (MoH, 2012) [5]. Noise-induced hearing loss (NIHL), in addition to respiratory disease, is also one of public health issues [9]. It is, however, regarded as the most preventable cause of hearing loss in the workplace [6]. Every year, the number of occupational diseases in the heavy machine industry rises. In Malaysia, based on the industrial noise control module, cases regarding occupational noise related Hearing Disorders (HD) increased from 2876 cases in 2016 to 4787 cases in 2017 [7]. Machinery, manufacturing, and automobiles are the three most harmful causes of noise emissions in an industrial environment [4]. Excessive noise causes low performance and absenteeism among workers, as well as a loss of productivity due to hearing loss [9]. It also induces sleep disorders and a variety of health problems, including hypertension and human psychosis. An individual can tolerate 90 decibels (dB) of noise for 8 hours, 115 dB for a short time, and noise levels higher than that are not recommended [7]. According to other research, peak sound levels of 85dB to 90dB can cause hearing loss [9]. While many systems exist to track air pollution levels and detect toxic gases, many businesses need a real-time IoT monitoring device. The smart city model, which is rapidly becoming the benchmark for both developing and developed countries [10], includes air quality monitoring and control. The IoT-based industrial plant safety gas leakage detection system is an existing air pollution monitoring system equipped with a IoT system specially designed

to detect gas leakages in power plant sites to avoid pollution, explosions and maintain the safety of employees. The proposed leakage detector is able to detect various types of gases and immediately alert workers via Short Message Service (SMS) notifications. Despite its simplicity, the centralized warning protocol may be ideal for alerting powerplant personnel in circumstances where cell phones are not permitted. Indoor air quality system is an IoT system that is innovatively integrated into the use of Raspberry Pi, Fuzzy logic, and Cloud Server applications [12]. This system can detect carbon dioxide gas and particulate matter (PM10) [12]. Message Queuing Telemetry Transport (MQTT) is the main con of this system as it only functions in a low bandwidth device. K. Gayathri proposed a device has high sensitivity towards more gases such as ammonia, sulphide, and benzene steam in a study [13]. At the same time, the device will determine the air quality index and noise levels. These systems, including the indoor air quality system, use Message Queuing Telemetry Transport (MQTT) to monitor outputs and send messages [13], which is the con of this system. Another system provides additional features such as air, sound, temperature, and humidity monitoring [14]. The inputs of this system are from the MQ135 and MQ7 gas sensors, sound sensor module mic, andDHT11 temperature humidity sensor. This system can detect hazardous gas and display pollution levels on an LCD display monitor. It is not, unfortunately, designed for wireless centralized monitoring [1]. The proposed framework in this paper, on the other hand, integrates gas and noise pollution measurement and provides centralized monitoring from a website, as well as the ability to warn the user when pollution levels reach alarming levels. This will ultimately provide impetus to address the pollution problem in heavy industry while still ensuring that pollution levels remain within safe limits

SYSTEM SPECIFICATION

HARDWARE REQUIEMENTS:

Sensor **:Sound Sensor LM393 & GAS Sensor MQ9.**

System **:Pentium IV 2.4GHz.Hard.**

Disk **:500 GB.**

Monitor **:15 VGA Colour.**

Ram **:4 GB.**

SOFTWARE REQUIREMENTS:

Operating System **:Windows 10.**

Front End **:Phyton**

Back End **Json**

METHODOLOGY:

The proposed IoT Based Air and Noise Pollution monitoring system block diagram is presented in Figure 1. A NodeMCU ESP8266 Wi-Fi module is used as the main microcontroller as it can connect to the selected IoT server using Wi-Fi connection. Wireless communication between ESP8266 Wi-Fi Module NodeMCU and ThingSpeak was enabled. ThingSpeak was the cloud system used to analyze and store air quality and noise level data. In this proposed system, hotspot mode was used to connect ESP8266 to the internet. A mobile phone was used as a hotspot to make it easier for ESP8266 to connect to the Internet without installing a separate wireless router. A NodeMCU board is used as it offered a simpler and direct conversion from analog to digital. Gas sensor MQ9 was used to measure carbon

monoxide concentrations and a sound sensor LM393 was used to measure noise levels in the surrounding area. Gas sensor MQ9 has a range of detection of 10-1000 ppm for Carbon Monoxide gas and 100-10000 ppm for combustible gas. Figure 2 shows the hardware setup when measuring the carbon monoxide concentration and noise level using the MQ9 gas sensor and LM393 sound sensor. Figure 3 (a) shows the Benetech carbon monoxide meterand the (b) SMART SENSOR Intell Instruments Pro digital sound level meter. Both are stand-alone measuring devices used to measure carbon monoxide concentration and noise levels for validation purpose. The Benetech carbon monoxide meter used to measure carbon monoxide concentrations has a measuring range from 0 – 1000 ppm carbon monoxide and a basic error rate of ±5 to ±10 ppm. A digital sound level meter manufactured by SMART SENSOR Intell Instruments Pro used to measure the noise level in the surrounding area has a basic error rate of ±1.5 dB and is able to measure noises ranging from 30 dB – 130 dB. The values obtained from both devices were used as true values.

Naïve Bayes Algorithm Used –

A naive Bayes classifier (NBC) is a simple probabilistic based method that may expect the magnificence club possibilities. A naive Bayes classifier can easily manage missing characteristic values by absolutely omitting the corresponding probabilities for those attributes while calculating the probability of club for every class. It also calls for the magnificence conditional independence, i.e., a characteristic on a given magnificence is independent of these of different attributes.

Data : 'Training dataset: D = $X_1, X_2, \ldots X_n$ // Training dataset, D, which contains a set of training instances and their associated classlabels.

Result : noise list: $noise_{list}$

for each class, $C_i \varepsilon D$ **do**

Find the prior probabilities, $P(C_i)$

end

for each attribute values, $A_i \varepsilon D$ **do**

Find the class of conditional probabilities, $P(A_i \mid C_i)$

end

for each class, $X_i \varepsilon D$ **do**

Find the conditional probabilities, $P(X_i \mid C_i)$*if* $P(X_i \mid C_i) == 0$ **then**

// Use Laplacian Estimator recalculate the conditional probability $P(X_i \mid C_i)$ using Laplacian Estimator

end

if X_i is misclassified **then**

$misClass_{list} <-- X_i$

$misPro_{list} <-- P(X_i \mid C_i)$ *// Store all the probabilities for all misclassified*

end

else

$pureClass_{list} <-- X_i$

$purePro_{list} <-- P(X_i \mid C_i)$ *// Store the probabilities for all purely classified instances.*

end

end

*Tnoise = find**MIN** (purePro_{list}) // Use as noise threshold*

for each instance x_i ε *misClass$_{list}$* **do**

Find the conditional probablities $P(X_i \mid C_i)$ *misPro$_{list}$*

if $P(X_i \mid C_i) < Tnoise$ **then**

noise$_{list}$ <-- X_i // *Store the instance as noise*

end

end

return *noise$_{list}$*

Let D be a training set of data instances and their associated class labels. Each instance is represented by an n-dimensional attribute vector, X ={x1,x2,...,xn}, depicting n measurements made on the instance from n attributes, respectively,{A1,A2,...,An}.

Suppose that there are m classes,{C1,C2,...,Cm}. For a test instance, X, the classifier will predict that X belongs to the class with the highest conditional probability, conditioned on X. That is, the naive Bayes classifier predicts that the instance X belongs to the class Ci, if and only if P(Ci|X) > P(Cj|X) for $1 \le j \le m$, $j \ne i$.The class Ci for which P(Ci|X) is maximized is called the Maximum Posterior $P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$

In Bayes theorem shown in Equation (1), as P(X) is a constant for all classes, only P(X|Ci)P(Ci) needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are likely equal, that is, P(C1) = P(C2) = ... = P(Cm), and therefore we would maximize P(X|Ci).

Otherwise, we maximize P(X|Ci)P(Ci). The class prior probabilities are calculated by P(Ci) = |Ci,D|/|D|, where |Ci,D| is the number of training instances of class Ci in D. To compute P(X|Ci) in a dataset with many attributes is extremely computationally expensive. Thus, the naive assumption of class-conditional independence is made in order to reduce computation in evaluating P(X|Ci). This presumes that the attributes' values are conditionally independent of one another, given the class label of the instance, i.e., there are no dependence relationships among attributes. Thus, Equation (2) and (3) are used to produce P(X|Ci).

$$P(X|C_i) = \prod_{k=1} P(x_k|C_i)$$

$$P(X|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i) \quad \ldots \text{Eq.2}$$

P(X|Ci) = P(x1|Ci)×P(x2|Ci)×...×P(xn|Ci) (3) In Equation (2), xk refers to the value of attribute Ak for instance X. Therefore, these probabilities P(x1|Ci),P(x2|Ci),...,P(xn|Ci) can be easily estimated from the training instances.

If the attribute value, Ak, is categorical, then P(xk|Ci) is the number of instances in the class Ci ∈ D with the value xk for Ak, divided by |Ci,D|, i.e., the number of instances belonging to the class Ci ∈ D. To predict the class label of instance X,P(X|Ci)P(Ci) is evaluated for each class Ci ∈ D. The naive Bayes classifier predicts that the class label of instance is the class Ci, if and only if P(X|Ci)P(Ci) > P(X|Cj)P(Cj) for 1 ≤ j ≤ m and j 6= i In other words, the predicted class label is the class Ci for which P(X|Ci)P(Ci) is the maximum.

Laplacian Estimation -

As in naive Bayes classifier discussed above, we calculate **P(X|Ci)** as the product of the probabilities **P(x1|Ci)×P(x2|Ci)×...×P(xn|Ci),** based on the independence assumption and class conditional probabilities, and end up with a actual probable value of zero for some **P(x|Ci).** This may happen if the attribute value x is never observed in the training data for a particular class **Ci.** Therefore, the generated Equation 3 becomes zeros for such attribute value regardless the values of other attributes. Thus, naive Bayes classifier is unable to predict the class of such test instances. In order to resolve such issues in our model, we use Laplace estimator *Cestnik (1990)* to scale up the values by smoothing factor. In Laplace-estimate, the class probability is defined as *Cestnik (1990)*:

$$P(C = c_i) = \frac{n_c + k}{N + n * k} \quad \ldots \text{Eq.3}$$

Where nc is the number of instances satisfying C=ci, N is the number of training instances, n is the number of classes and k=1. Let's consider a phone call behavior example, for the behavior class 'reject' in the training data containing 1000 instances, we have 0 instance with relationship is qual to unknown, 990 instances with relationship=friend, and 10 instances with relationship is equal to mother.

The probabilities of these contexts are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively. On the alternative hand in keeping with equation 2 the probabilities of those contexts could be as follows:

In this way we obtain the above non zero possibilities rounded up to a three decimal places severally victimization Laplace calculator outlined on top of these.

$$\frac{1}{1003} = 0.001, \frac{991}{1003} = 0.988, \frac{11}{1003} = 0.011$$

The "new" probability estimates are close to their "previous" counterparts, and these values can be used for further processing in our model.

- ***Noise Detection Technique -***

In this section, to discuss the noise detection technique in order to detect phone call, Machine running, behavior. Figure 1 shows the complete execution of noise detection system block and technique used. In order to detect noise, The implemented system use Naive Bayes classifier (NBC)[9] as the basis for noise identification. Using nbc we tend to rst calculate the probability for every attribute by scanning the coaching knowledge, The portable dataset Each example contains 4 attribute values, time, area and relationship and corresponding noise level behavior. For each attribute value, respectively for this dataset. Using these possibilities we calculate the conditional possibility for each example as nbc was implemented beneath the independence.

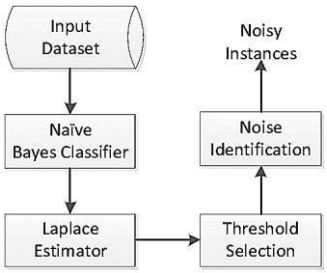


Fig. 8 : Input Phase

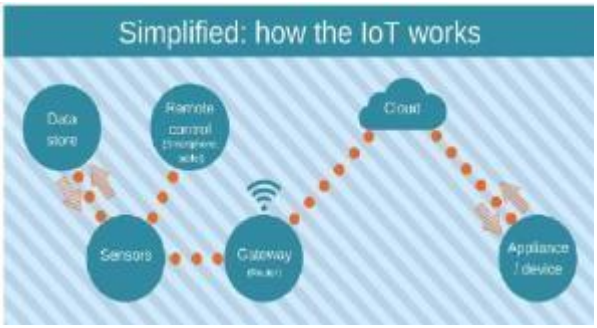
Assumption, it estimates zero possibilities if the probability for one attribute is zero attribute is zero. In such cases, we use Laplace- estimator [10] estimate the conditional probability of any of the attribute value. After the match value prediction it shows the "Number of record scans" and show the probability generated. Once we have calculated conditional probability for each instance, we differentiate between the purely classified instances and misclassified instances using machine learning.

PROJECT DESCRIPTION:

Noise pollution has become a very big issue around the world. The adverse effects of this pollution include hearing impairment, negative social behavior annoyance, sleep disturbance and intelligibility to understand people's speech. In learning context, noise can affect understanding and behavior of people and places with high noise level are not suitable for learning and teaching process. Internet of Things (IoT) technology is one of the best choices to monitor the noise or sound intensity in the environment for the safety of human being.

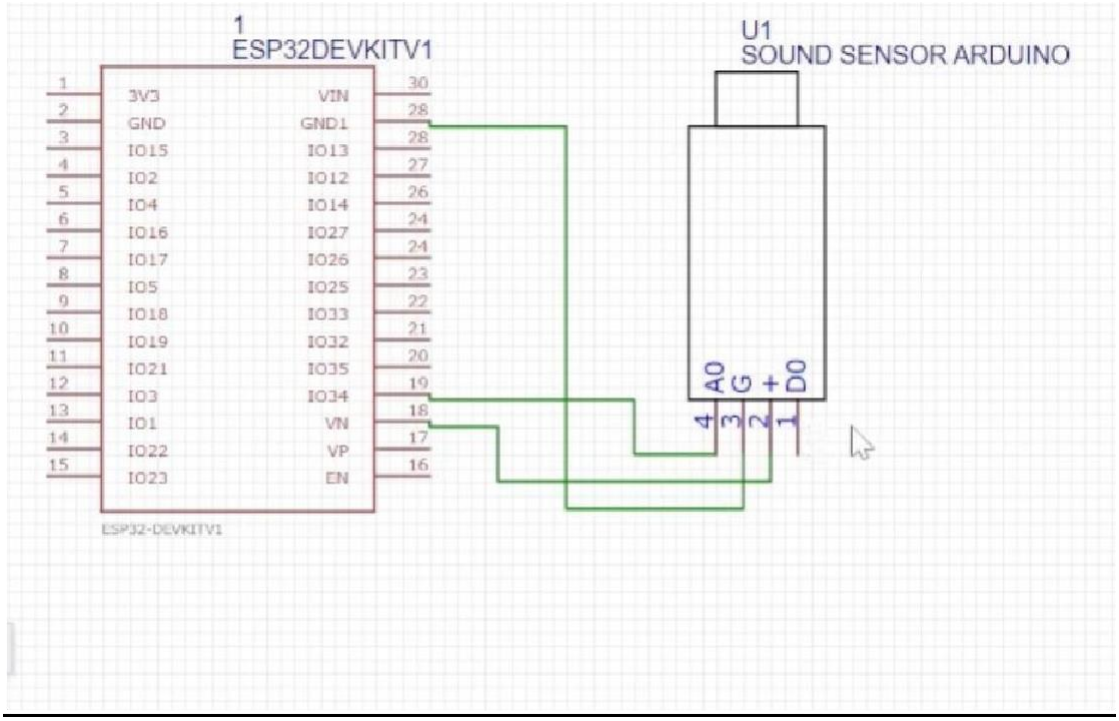
The aim of this paper is to deliver a development of an IoT based noise monitoring system comprises of a sound sensor, an IoT platform called NodeMCU, LCD and LEDs. The system will provide a real-time alert if the noise exceeds the threshold noise limit set byEnvironmental Department of Health standard. Equipped with an Android application, the data from the sound sensor will be transferred into the cloud server and subsequently transferred into the app for display and to enable remote monitoring.

IMPLEMENTATION:

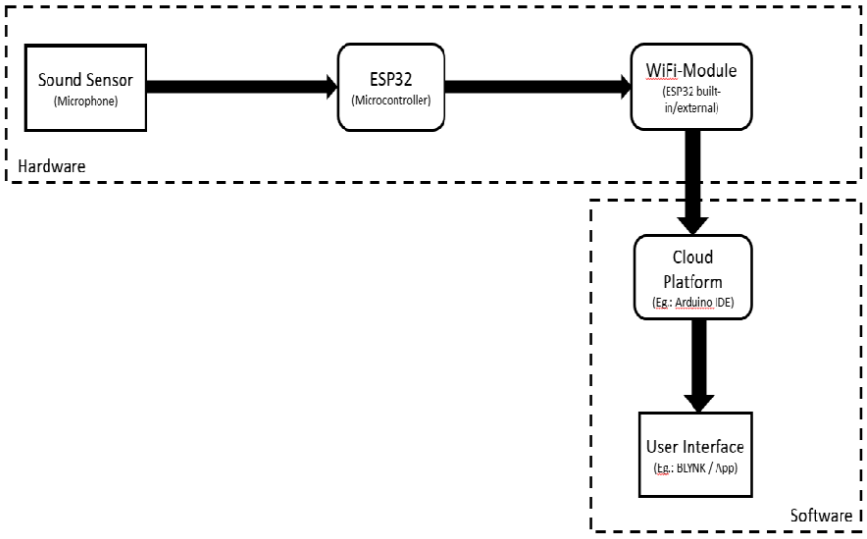


Sound sensor or mic sensor provides digital output and it detects sound from atmosphere. A WiFi module is also connected to Arduino and it is used to transfer data from the sensors to cloud server. ESP8266 WiFi module is used to store the data to online server. The data from sensor are basically analog signal so analog to digital converter (ADC) is used to convert the data. 16 x 2 Liquid crystal display (LCD) is used to display the measured value from the sensors. It can display two lines and each line has 16 characters.

Circuit diagram:



Block diagram:



Source code:

4.1 Front End code:

```
import SwiftUI

struct ContentView: View {
   @State private var isMonitoring =
   false @State private var noiseLevel
   = 0.0

   var body: some View
      {VStack {
```

```
        Text("Current Noise Level: \(noiseLevel) dB")
            .font(.largeTitle)

        Button(action: {
            // Start/stop monitoring code
            hereself.isMonitoring.toggle()
        }) {
            Text(isMonitoring ? "Stop Monitoring" : "Start Monitoring")
                .padding()
                .background(Color.blue)
                .foregroundColor(.white)
                .cornerRadius(10)
        }
    }
  }
}

@main

struct NoiseApp: App {
  var body: some Scene
  {
    WindowGroup
      {
      ContentView
      ()
    }
  }
}

xml
<!-- activity_main.xml layout file -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:id="@+id/noiseLevelTextVie
        w"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content
        " android:text="Current Noise Level:
        0 dB"android:textSize="24sp" />

    <Button
        android:id="@+id/startStopButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start Monitoring" />

 </LinearLayout>

 kotlin
 // MainActivity.kt
import     android.os.Bundle
import    android.view.View
import
android.widget.Button
 import android.widget.TextView
 import androidx.appcompat.app.AppCompatActivity

 class MainActivity : AppCompatActivity() {
    private    lateinit    var    noiseLevelTextView:
    TextView private  lateinit  var  startStopButton:
    Button
```

```
    private var isMonitoring = false

    override fun onCreate(savedInstanceState:
        Bundle?) { super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        noiseLevelTextView =
        findViewById(R.id.noiseLevelTextView) startStopButton =
        findViewById(R.id.startStopButton)

        startStopButton.setOnClickListener {
            // Start/stop monitoring code
            here isMonitoring =
            !isMonitoring updateUI()
        }
    }

    private fun updateUI()
        {if (isMonitoring) {
            startStopButton.text = "Stop Monitoring"
        } else {
            startStopButton.text = "Start Monitoring"
        }
    }
}
```

4.2.Back End code:

```
    import sounddevice as
sdimport numpy as np
import paho.mqtt.client as
mqttimport time

# MQTT settings
```

```
mqtt_broker_address =
"your_broker_address"mqtt_port = 1883
mqtt_topic = "noise_level"

# Sampling parameters
sample_rate = 44100 # Samples per
second duration = 10 # Recording
duration in seconds

# Function to calculate dB from audio
datadef calculate_db(audio_data):
  rms = np.sqrt(np.mean(np.square(audio_data))) # Root Mean
  Square db = 20 * np.log10(rms / 0.0002) # Reference sound
  pressure level return db

# MQTT callback functions
def on_connect(client, userdata, flags,
  rc):if rc == 0:
    print("Connected to MQTT
  broker")else:
    print("Connection to MQTT broker failed")

def on_publish(client, userdata,
  mid): print("Data published to
  MQTT")

# Create MQTT
client client =
mqtt.Client()
client.on_connect =
on_connect
client.on_publish =
on_publish

# Connect to MQTT broker
```

```
client.connect(mqtt_broker_address,
mqtt_port)

# Start recording and publishing
with sd.InputStream(callback=calculate_db, channels=1,
  samplerate=sample_rate):print("Monitoring noise level...")
  while True:
    db = calculate_db(np.random.random_sample(44100)) # Simulated audio data for testing
    client.publish(mqtt_topic, str(db))
    time.sleep(60) # Publish data every minute (adjust as needed)
```

Sample output screenshot:

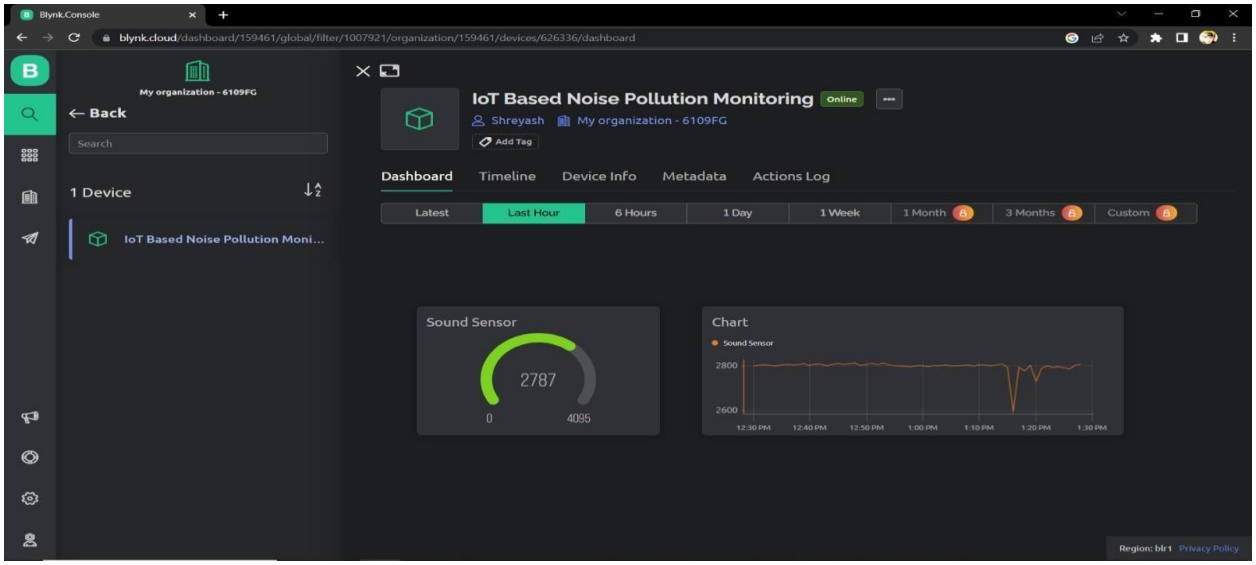


Fig4.1.1initial state

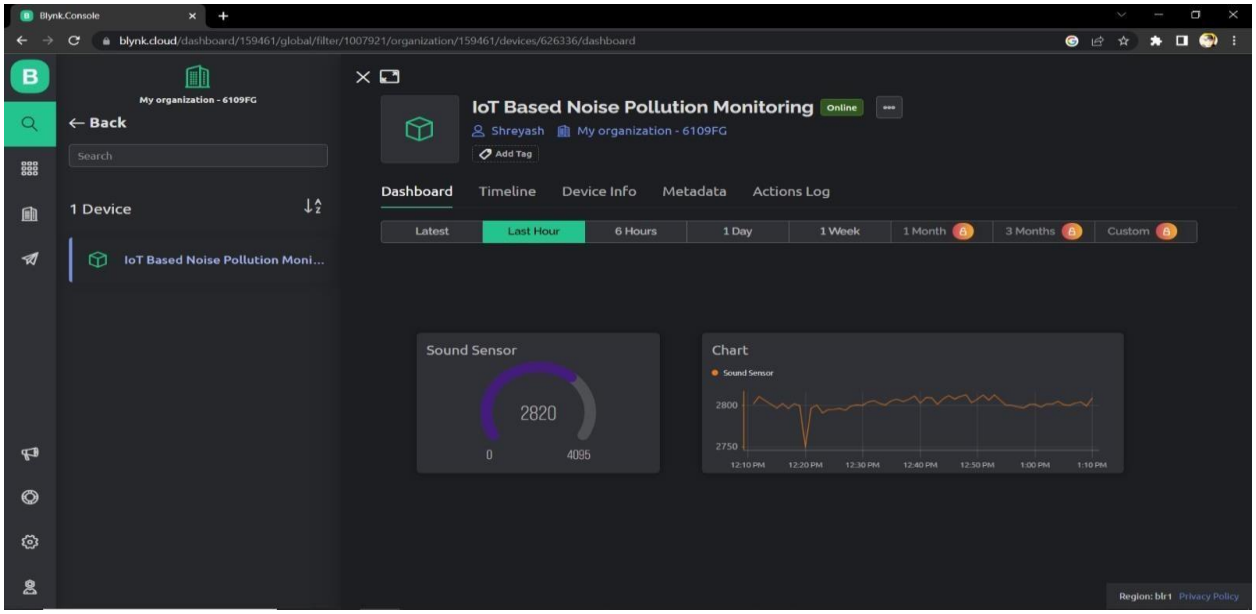


Fig 4.1.2 processing

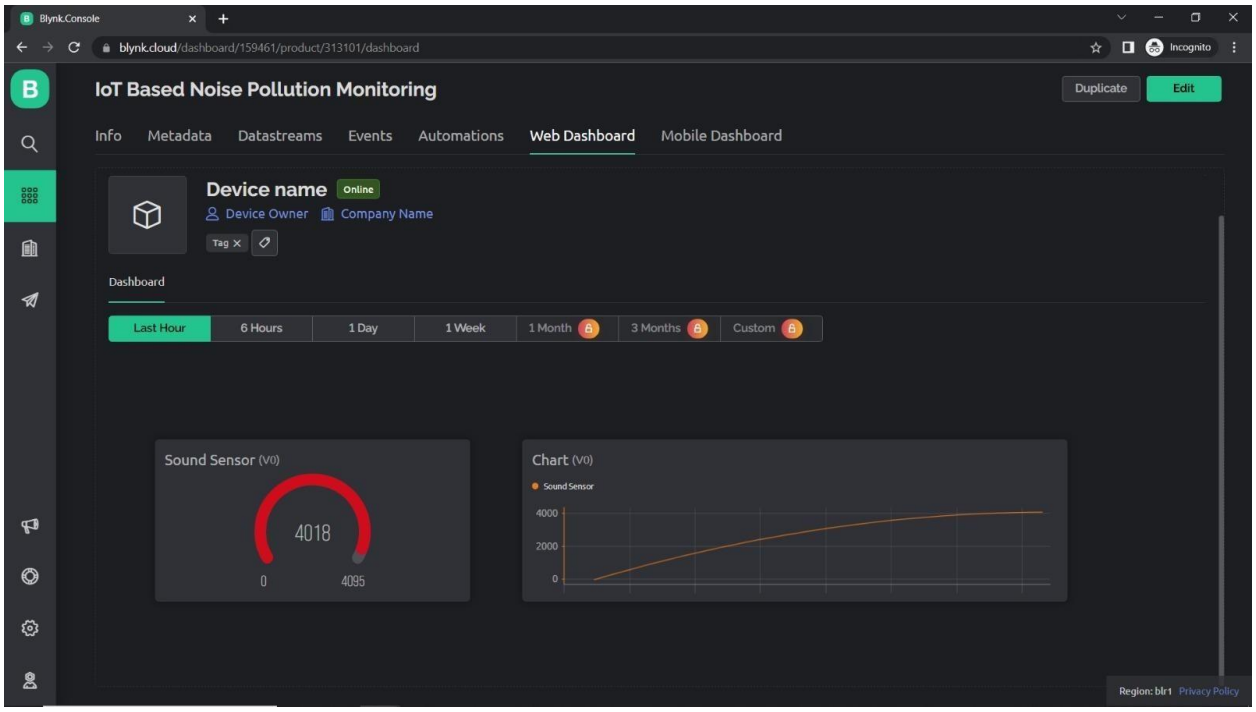


Fig.4.1.3.End of the process

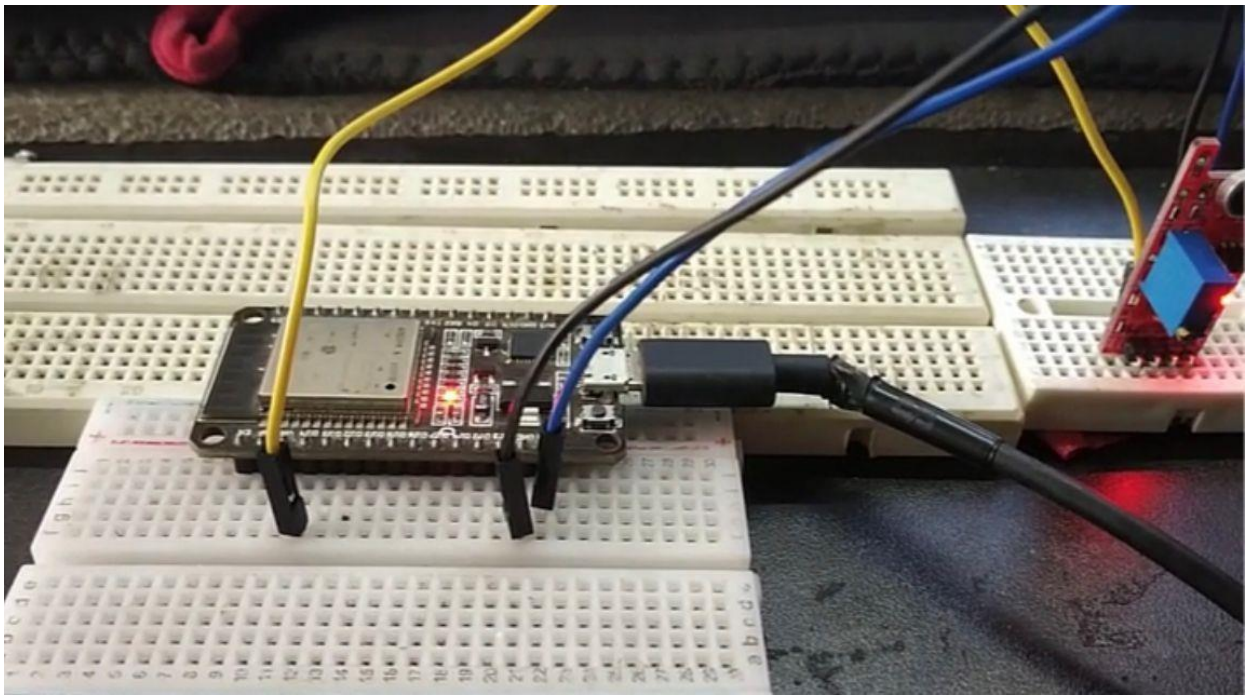


Fig.4.1.4. sensor board

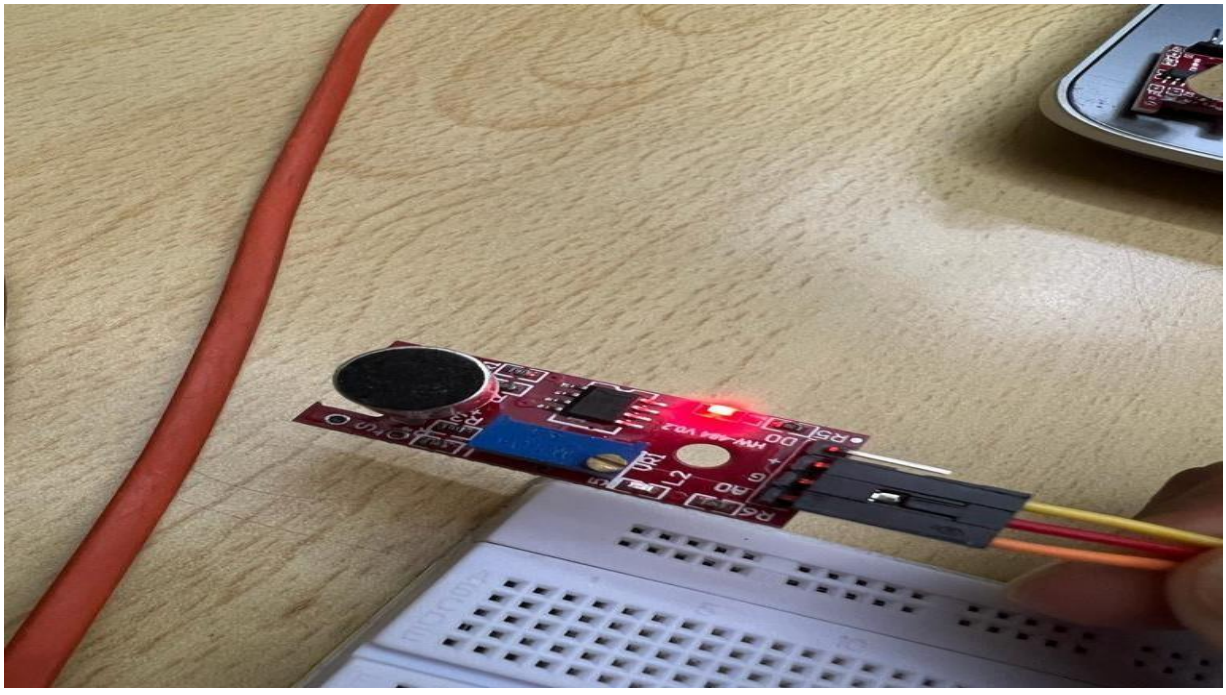


Fig.4.1.5 circuit board

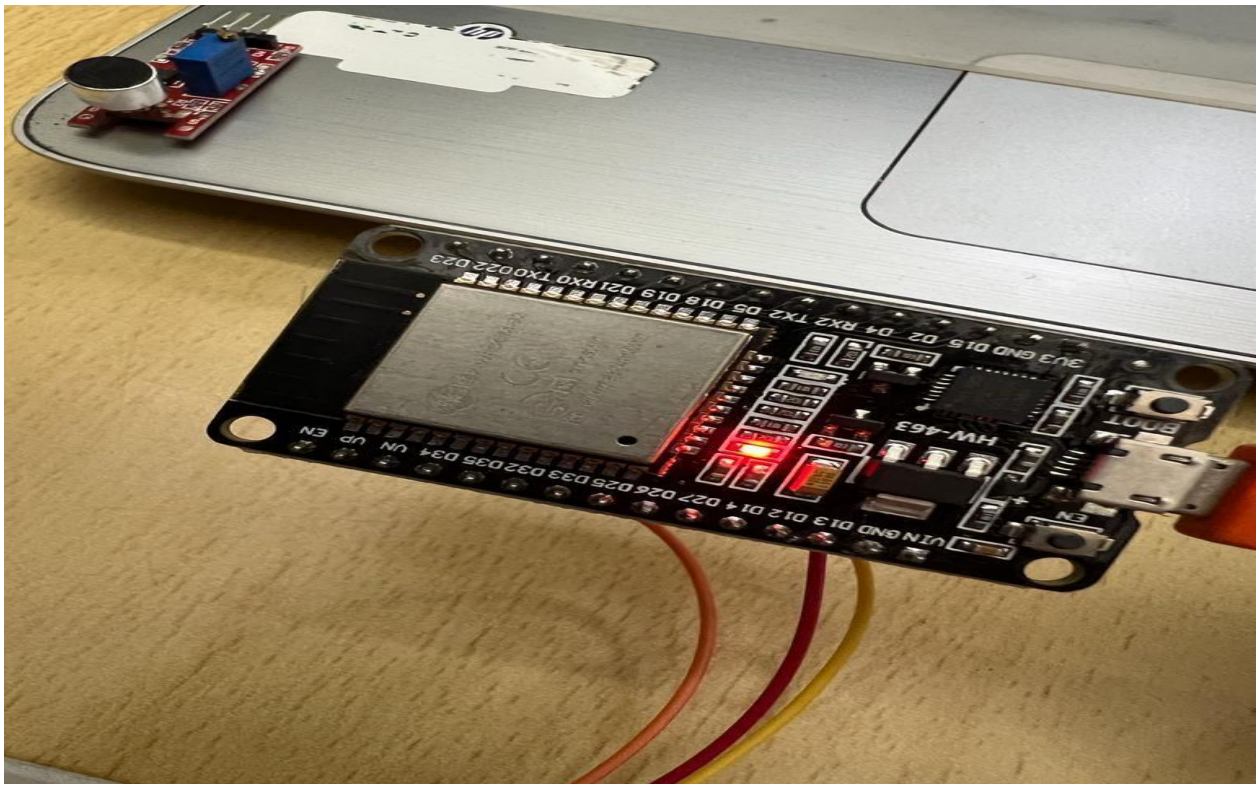
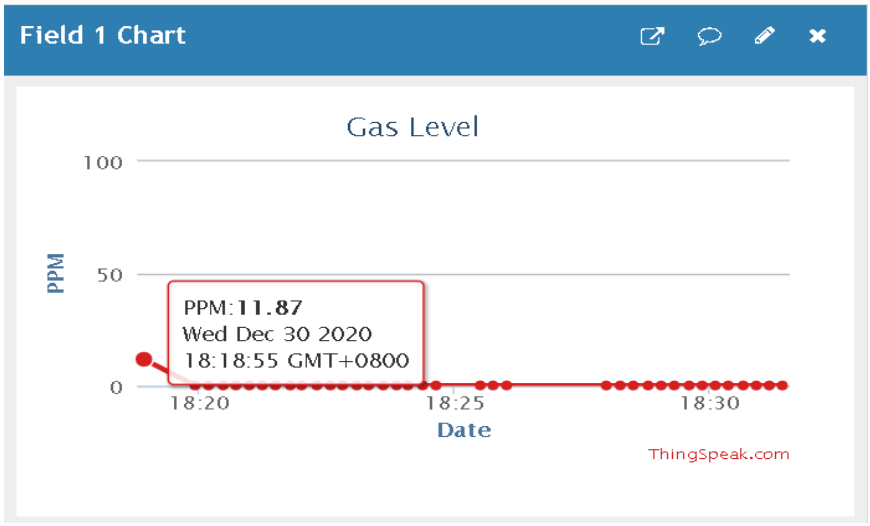


Fig.4.1.6 Over all circuit board

Noise pollution monitoringSystem analysis:

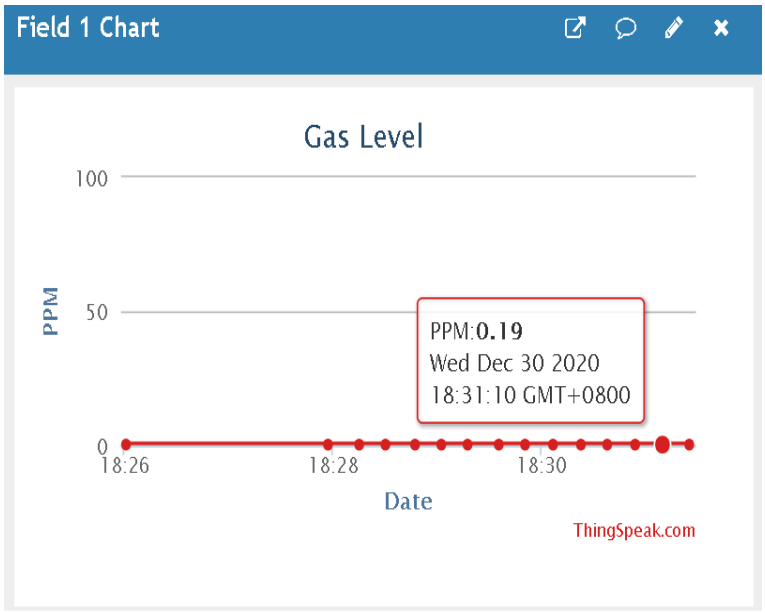
The value obtained by the gas sensor MQ9 and sound sensor LM393 were both analog inputs. Both sensors were connected to the analog pin of ESP8266 Wi-Fi Module NodeMCU. The value obtained from analog input wasconverted to into digital output. The measured ppm value and noise level in decibel (dB) were displayed on the Arduino serial monitor of the selected COM port. The data was sent to the ThingSpeak server and the output displayed in graphical analytic form.

Referring to Figure 5 (a) and (b), it shows the ThingSpeak graphical representation ofgas level. Figure 5 (a) shows the highest carbon monoxide concentration, while Figure 5 (b) shows the lowest concentration value obtained from the gas sensor MQ9.



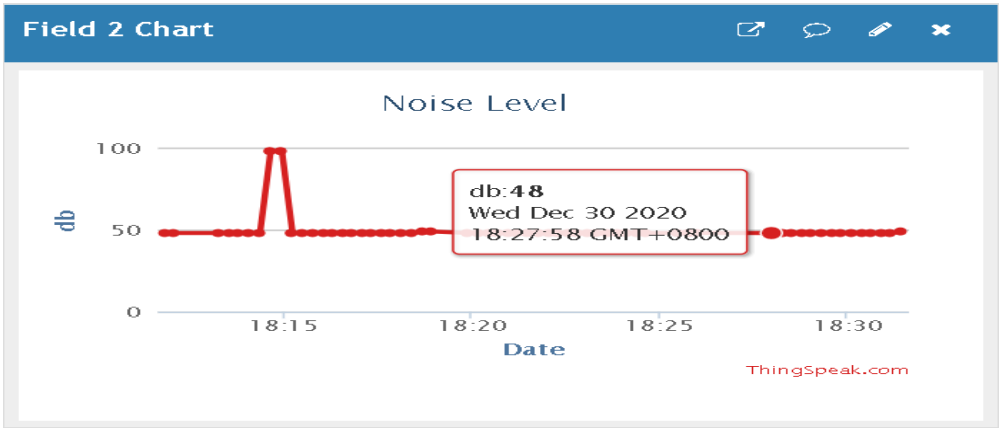
(a)

Fig 1: Showing the ThingSpeak graph of the (a) maximum concentration of Carbon monoxide, CO



(b)

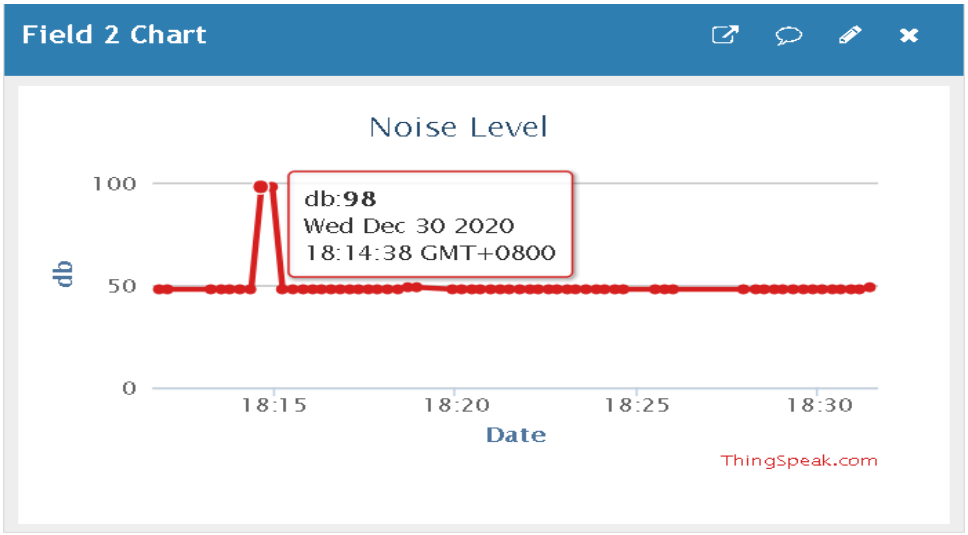
fig2: Showing the ThingSpeak graph of the (b) minimum concentration of Carbon monoxide, CO in ppm (parts per million)



(a)

Fig 3:Showing the Thingspeak graph of the (a) maximum noise level

The ThingSpeak graphical representation of noise levelin decibels (dB) is shown in Figures6 (a) and (b). Figure 6 (a) shows the highest level of noise while Figure 6 (b) shows the lowest level of noise recorded from the sound sensor LM393.To activate the alerting state, a Bluetooth speaker wasturned up to its maximum volume to provide loud sound from a song



(b)

Fig 4: Showing the Thingspeak graph of the (b) minimum noise level in decibel (dB)

The validation results are shown in Figures 7 and 8. Running ten trials and averaging the findings completed the validation process for carbon monoxide concentration. Figure 7 shows that the average carbon monoxide concentration for the gas sensor MQ9 and the carbon monoxide meter are identical, though the average valuesvaried at times. This might be due to the tolerance level of the sensor and carbon monoxide meter. The ability of gas sensor MQ9 and

carbon monoxide meter to measure the carbon monoxide concentrations was also hampered by variations in the direction of the wind and the condition of the surrounding room.

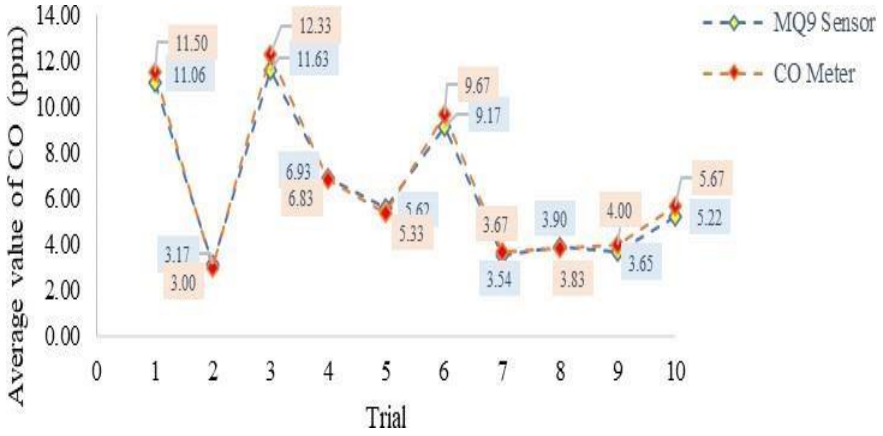


Fig 5: Showing the comparison graph of carbon monoxide concentrations in ppm using gas sensor MQ9 and carbon monoxidemeter

meter are shown in Figure8. To confirm the noise values measured by the sound sensor LM393, three The noise level values obtained from the sound sensor LM393 and the digital sound level experiments were carried out. The same part of the song was played for 60 seconds. The trials were conducted in a controlled environment with a Bluetoothspeaker acting as a noise source. From the results, theoverall noise level from both instruments was comparable. Between the sound sensor LM393 and the digital sound level meter, there were minor variations in value ranging from 0.1 to 0.3 decimal points. This may be due to the sensor's and instrument's tolerance levels.

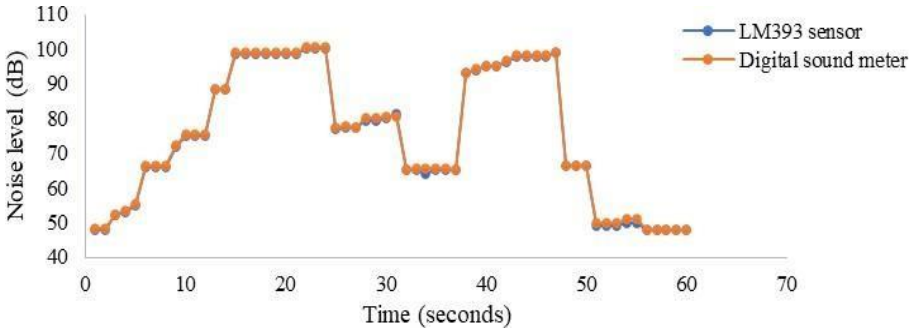


Fig 6: Showing the comparison graph of noise levels in dB using sound sensor LM393 and digital sound level meter