# AIR POLLUTION MONITORING

## PHASE 1: PROBLEM DEFINITION AND DESIGN THINKING

## AIR POLLUTION MONITORING IOT PROJECT UNDERSTANDING AND APPROACH

### INTRODUCTION

Air is getting polluted because of the release of toxic gases by industries, vehicle emissions and increased concentration of harmful gases and particulate matter in the atmosphere. The level of pollution is increasing rapidly due to factors like industries, urbanization, increase in population, vehicle use which can affect human health. Particulate matter is one of the most important parameters having a significant contribution to the increase in air pollution. This creates a need for measurement and analysis of real-time air quality monitoring so that appropriate decisions can be taken in a timely period.

### UNDERSTANDING THE PROBLEM

#### 1. AIR POLLUTION

Air pollution is one of the biggest threats to the present-day environment. Everyone is being affected by air pollution day by day including humans, animals, crops, cities, forests and aquatic ecosystems. Besides that, it should be controlled at a certain level to prevent the increasing rate of global warming

#### 2. SOLUTION OVERVIEW

The project involves setting up IoT devices to measure air quality parameters and make the data publicly available for raising awareness about air quality and its impact on public health. The objective is to create a platform that provides real-time air quality information to the public. This project includes defining objectives, designing the IoT monitoring system, developing the data-sharing platform, and integrating them using IoT technology and Python.

### APPROACH TO SOLVING THE PROBLEM

#### 1. DEFINE PROJECT SCOPE

The first step is to clearly define the scope of the project. This includes determining the size of the area to be covered, the number of spaces to be monitored, and the level of detail required in reporting.

#### 2. SENSOR DEPLOYMENT

**a. Sensor Selection**

Choose sensors based on the pollutants of interest and their measurement accuracy, sensitivity, and precision .Consider the type of sensor technology (e.g., optical, electrochemical, gravimetric) suitable for your monitoring objectives..

**b. Sensor Placement:**

Determine the optimal locations for sensor deployment to capture representative air quality data .Place sensors in areas with high pollution levels, near pollution sources, and in areas with high human exposure (e.g., near schools, hospitals.)

**HARDWARE COMPONENTS:**

**Air Quality Sensors:**

- Air quality sensors are the core components for measuring different air pollutants. Common sensors include:
  - MQ series sensors (e.g., MQ-7 for CO, MQ-135 for CO2)
  - Particulate matter (PM) sensors (e.g., SDS011, PMS5003)
  - Ozone (O3) sensors
  - Nitrogen dioxide (NO2) sensors

**Microcontroller:**

- NodeMCU (ESP8266) or Arduino boards (e.g., Arduino Uno, Arduino Nano) to interface with sensors, process data, and send it to a central server or cloud.

**Communication Module:**

- WiFi module (e.g., ESP8266, ESP32) to connect the microcontroller to the internet, enabling data transmission to the server or cloud.

**Power Supply:**

- Power source (e.g., battery, power adapter) to power the sensors, microcontroller, and communication modules.

**Breadboard and Jumper Wires:**

- Breadboards for prototyping and jumper wires to connect components on the breadboard.

**LCD Display:**

- LCD module (e.g., 16x2 character LCD) to display real-time air quality data.

**Enclosure:**

- A protective housing to house the components and protect them from environmental conditions (optional, especially for outdoor installations).

**Other Components:**

- Resistors, capacitors, and other electronic components for interfacing sensors with the microcontroller.

**Optional Components:**

- GPS module: To record location information along with air quality data.
- Temperature and humidity sensors: To measure environmental conditions.
- Solar panels and related components: For powering the system using solar energy (useful for remote or outdoor installations).

**SOFTWARE COMPONENTS:**

**Arduino IDE:**

- The Arduino Integrated Development Environment (IDE) is commonly used for programming the microcontroller (e.g., NodeMCU, Arduino boards). It allows writing, compiling, and uploading firmware code to the microcontroller.

**CODE:**

```
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

char auth[] = "YOUR_BLYNK_AUTH_TOKEN"; // Replace with your Blynk auth token
char ssid [] = "YOUR_WIFI_SSID";
char pass [] = "YOUR_WIFI_PASSWORD";

const int coSensorPin = A0;   // Analog pin for CO sensor (MQ-7)
const int co2SensorPin = A1;  // Analog pin for CO2 sensor (MQ-135)

LiquidCrystal_I2C lcd(0x27, 16, 2);  // Address for 16x2 LCD
```

```
void setup() {
 Serial.begin(115200);
 Blynk.begin(auth, ssid, pass);
 lcd.begin();
 lcd.backlight();
}

void loop() {
 Blynk.run();

 // Read sensor values
 int coSensorValue = analogRead(coSensorPin);
 int co2SensorValue = analogRead(co2SensorPin);

 // Convert sensor values to air quality levels
 float coLevel = map(coSensorValue, 0, 1023, 0, 100);  // Assuming a linear mapping
 float co2Level = map(co2SensorValue, 0, 1023, 0, 100);  // Assuming a linear mapping

 // Display pollution levels on the Blynk app
 Blynk.virtualWrite(V0, coLevel);   // Virtual pin V0 for CO level
 Blynk.virtualWrite(V1, co2Level);  // Virtual pin V1 for CO2 level

 // Display pollution levels on the LCD
 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("CO: ");
 lcd.print(coLevel);
 lcd.print("%");

 lcd.setCursor(0, 1);
 lcd.print("CO2: ");
 lcd.print(co2Level);
 lcd.print("%");
```

```
  // Check if pollution level is below 60% and send a notification
  if (coLevel < 60 || co2Level < 60) {
    Blynk.notify("Air pollution below 60%! Action needed.");
  }

  delay(5000);  // Adjust delay based on your sampling frequency
}
```

## HARDWARD COMPONENTS

### ➢ AIR QUALITY SENSORS:

Air quality sensors are the core components for measuring different air pollutants common sensors include:

- **MQ series sensor (e.g., MQ-7 for CO, MQ-135 for CO2)**
- **Particulate matter (PM) sensors (e.g., SDS011, PMS5003)**
- **Ozone(O3) Sensors**
- **Nitrogen dioxide (NO2) sensors**

### ➢ MICROCONTROLLER:

NodeMCU (ESP8266) or Arduino boards (e.g., Arduino Uno, Arduino Nano) to interface with sensors, process data and send it to a central server cloud.

### ➢ COMMUNICATION MODULE:

Wi-Fi module (e.g., ESP8266, ESP32) to connect the microcontroller to the internet, enabling data transmission to the server or cloud.

### ➢ POWER SUPPLY:

Power source (e.g., battery, power adapter) to power the sensor, microcontroller and communication modules.

### ➢ BREADBOARD AND JUMPER WIRES:

Breadboards for prototyping and jumper wires to connect components on the breadboard.

- **LCD DISPLAY:**

LCD module (e.g.,16x2 character LCD) to display real time air quality data.

- **ENCLOSURE:**

A protective housing to house and components and protect them from environmental condition (optional, especially for outdoor installations).

## OTHER COMPONENTS

- **Resistor:**

Air Quality Monitoring Networks allow the measurement, operation and predictive analysis of the evolution of air pollution in different areas (urban areas, industrial areas, special nature conservation areas, etc.) Some stations are equipped with meteorological sensors and/or noise level meters to measure noise levels.

- **Capacitors:**

Variable air capacitors are used in circumstances where the capacitance needs to be varied. They are sometimes used in resonant circuits, such as radio tuners, frequency mixers or antenna impedance matching applications. Another use for variable capacitors is while prototyping an electronic circuit design.

- **Software code:**

```
Import time
From datatime import datatime
From sds011 import SDS011
 Import matplotlip .pyplot as plt
#initialize thePMsensor
 sds=SDS011("\dev\ttyUSB0")
```

```
use_query_mode=true)

def read _pm_sensor():

“””””Read PM sensor data.”””””

try:

pm_data =sds.query():

pm25,pm10=pm_data

 except Exception as e:

      print(f"Error reading PM sensor: {e}")

      return None


# Data storage (in-memory for this example)

pm25_data = []

pm10_data = [] import time

from datetime import datetime

from sds011 import SDS011

import matplotlib.pyplot as plt


# Initialize the PM sensor

sds = SDS011("/dev/ttyUSB0", use_query_mode=True)


def read_pm_sensor():

   """Read PM sensor data."""

   try:

      pm_data = sds.query()

      pm25, pm10 = pm_data
```

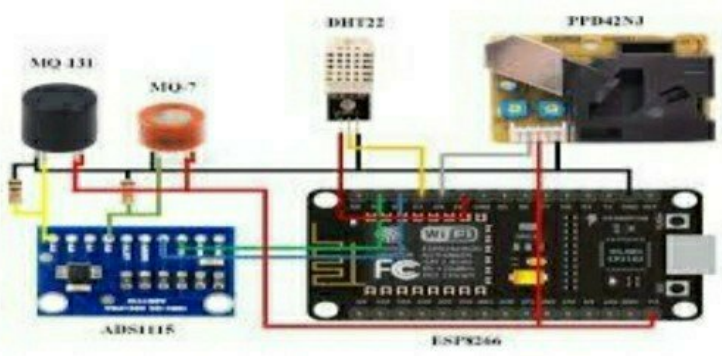
```
def record_pm_data():
    """Record PM sensor data."""
    global pm25_data, pm10_data
    pm_values = read_pm_sensor()
    if pm_values:
        pm25, pm10 = pm_values
        timestamp = datetime.now()
        pm25_data. append((timestamp, pm25))
        pm10_data. append((timestamp, pm10))

# Data visualization
def plot_pm_data():
    """Plot PM sensor data."""
    timestamps, pm25_values = zip(*pm25_data)
    _, pm10_values = zip(*pm10_data)

    plt.figure(figsize=(10, 6))
    plt.plot(timestamps, pm25_values, label="PM2.5 (µg/m³)")
    plt.plot(timestamps, pm10_values, label="PM10 (µg/m³)")
    plt.xlabel("Timestamp")
    plt.ylabel("Concentration")
    plt.title("Air Pollution Monitoring")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
# Main loop for data collection
try:
    while True:
        record_pm_data()
        time.sleep(300)  # Record every 5 minutes (adjust as needed)
except KeyboardInterrupt:
    # Plot data on KeyboardInterrupt (Ctrl+C)
    plot_pm_data()
```

- **HARDWARE DESING**



❖ **Software code:**

```
import
random import
time


class
AirQualityMonitor: def
init__(self):
self.temperature = 0
self.humidity = 0
self.air_quality_index = 0

def read_sensor_data(self):
self.temperature = round(random.uniform(18.0, 30.0), 2)
self.humidity = round(random.uniform(30.0, 60.0), 2)
self.air_quality_index = random.randint(0, 500)

def display_sensor_data(self):
```

```
print(f"Timestamp: {time.time()}")
print(f"Temperature: {self.temperature} °C")
print(f"Humidity: {self.humidity} %")
print(f"Air Quality Index: {self.air_quality_index}")


def run(self):
while True:

self.read_sensor_data()
self.display_sensor_data() time.sleep(1)


if __name__== "__main__":
air_quality_monitor = AirQualityMonitor()
air_quality_monitor.run()
```

❖ **Output:**

Timestamp: 1662760902.849983

Temperature: 24.77 °C

Humidity: 42.33 %

Air Quality Index: 375

Timestamp: 1662760903.850208

Temperature: 21.46 °C

Humidity: 51.56 %

Air Quality Index: 89

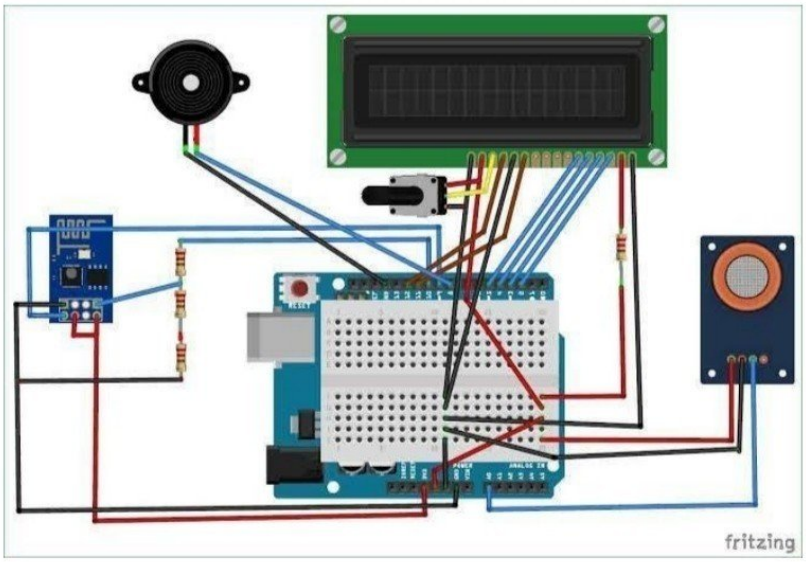
Timestamp: 1662760904.850433

Temperature: 28.82 °C

Humidity: 44.61 %

Air Quality Index: 187

➢ **Hardware Designing:**



## 3. DATA COLLECTION AND PROCESSING

### a. Sensor Deployment:

Deploy air quality sensors strategically in the target area. Ensure sensors are properly calibrated and maintained. Set the sampling rate for sensors based on the pollutants being measured and monitoring objectives. Continuous or periodic sampling may be used.

### b. Sensor Networks:

Establish a network of sensors if monitoring a large area. Ensure sensors are interconnected for data transmission.

### c. Remote Sensing:

In addition to ground-based sensors, consider using remote sensing technologies like satellite imagery for broader-scale monitoring.

### d. Data Storage and Management:

Establish a data storage system to securely store and organize collected data. Consider using cloud-based solutions for scalability and accessibility.

### e. Alert Systems:

Develop real-time or threshold-based alert systems to notify authorities and the public when air quality exceeds safety limits .Use color-coded scales or air quality indices (e.g., AQI - Air Quality Index) to communicate the level of pollution to the public.

## 4. USER INTERFACE DEVELOPMENT

### a. Real-time Data Display:

Display real-time air quality data, including pollutant concentrations, air quality indices, and weather conditions. Provide current readings and historical trends for various pollutants, both at the city or region level and at specific monitoring sites.

### b. Alerts and Notifications:

Implement an alert system to notify users when air quality exceeds predefined thresholds or when there are health advisories.

## 5. DATA VISUALIZATION AND ANALYTICS

Data visualization and analytics play a vital role in making sense of the vast amount of air pollution data collected from monitoring sensors. These tools help stakeholders, including scientists, policymakers, and the public, understand trends, identify pollution sources, and make informed decisions

## 6. USER ADOPTION AND EDUCATION

User adoption and education are critical aspects of successful air pollution monitoring initiatives. To ensure that the public, stakeholders, and decision-makers can effectively use and benefit from air quality data.

## 7. MAINTENANCE AND MONITORING

Maintenance and ongoing monitoring are crucial components of any air pollution monitoring system to ensure its reliability, accuracy, and longevity.

**8. SCALABILITY**

Design the system with scalability in mind, allowing for easy expansion to cover additional areas as needed.

**9. DATA SECURITY AND PRIVACY**

Implement robust security measures to protect sensor data and user information. Comply with relevant data privacy regulations.

**10. CONTINUOUS IMPROVEMENT**

Regularly evaluate the system's performance and gather feedback from users to identify areas for improvement and optimization.

**CONCULSION**

The results obtained from the experiments are verified through Google data. Moreover, the led indicators help us to detect the air quality level around the setup. However, the project experiences a drawback that is it cannot measure the ppm values of the pollutant components separately. Therefore, it is possible to conclude that the designed prototype can be utilized for air quality, humidity and temperature of the surrounding atmosphere successfully.