# NOISE POLLUTION MONITIRING

**Project Definition:** The project involves deploying IoT sensors to measure noise pollution in public areas and providing real-time noise level data accessible to the public through a platform or mobile app. The primary objective is to raise awareness about noise pollution and enable informed decision-making. This project includes defining objectives, designing the IoT sensor system, developing the noise pollution information platform, and integrating them using IoT technology and Python.

## Design Thinking:

### 1.Project Objectives: Define objectives such as real-time noise pollution monitoring, public awareness, noise regulation compliance, and improved quality of life.

Certainly! Here are definitions for each of the objectives you mentioned:

**Real-Time Noise Pollution Monitoring**:

- **Definition**: Real-time noise pollution monitoring refers to the continuous and instantaneous measurement and analysis of noise levels in a given area or environment. It involves the use of sensors and technology to provide up-to-the-minute data on noise pollution, including information on noise sources, intensity, frequency, and duration.

**Public Awareness**:

- **Definition**: Public awareness, in the context of noise pollution, is the degree to which the general population is informed and educated about the sources, effects, and consequences of noise pollution. It involves efforts to disseminate information, raise consciousness, and promote understanding among the public about the importance of noise control and its impact on health and well-being.

**Noise Regulation Compliance**:

- **Definition**: Noise regulation compliance refers to the adherence to established laws, regulations, and standards that govern acceptable noise levels and practices in a specific region or jurisdiction. It involves individuals, businesses, and organizations following noise-related rules and taking measures to minimize or mitigate their noise emissions to avoid legal violations.

**Improved Quality of Life**:

- **Definition**: Improved quality of life, in the context of noise pollution, signifies an enhanced overall well-being and living conditions for individuals and communities due to reduced exposure to disruptive and harmful noise. It encompasses various factors such as better physical and mental health, increased comfort, improved sleep, and a more pleasant living environment, all resulting from effective noise control measures.

These objectives are often interrelated and serve as essential components of noise pollution management and mitigation strategies. By monitoring noise in real-time, raising public awareness, ensuring regulatory compliance, and striving to enhance the quality of life, efforts can be made to reduce the negative impact of noise pollution on individuals and communities.

**2.IoT Sensor Design: Plan the deployment of IoT noise sensors in various public areas to measure noise levels.**

Deploying IoT noise sensors in various public areas to measure noise levels requires careful planning and consideration of several factors. Here is a step-by-step plan for the deployment of these sensors:

**Define Objectives and Scope:**

- Clearly define the objectives of the noise monitoring project, including what specific areas you want to cover and the purpose of collecting noise data (e.g., regulatory compliance, public awareness, research).

- Determine the scope of the deployment, such as the number of sensors needed, the geographical locations, and the frequency of data collection.

**Sensor Selection:**

- Choose appropriate IoT noise sensors based on your project requirements. Consider factors such as sensor accuracy, range, connectivity options (e.g., Wi-Fi, cellular), power source (e.g., battery or wired), and environmental suitability.

- Ensure that selected sensors are capable of real-time data collection and transmission.

**Site Survey:**

- Conduct a thorough site survey to identify the best locations for sensor placement. Key factors to consider include noise sources, population density, urban planning, and regulatory zones.

- Ensure that the selected locations are representative of the areas you want to monitor.

** Network Infrastructure:**

- Ensure that there is a reliable network infrastructure (e.g., Wi-Fi or cellular coverage) in place to transmit data from the sensors to a central data repository or server.

- Evaluate network security measures to protect sensor data.

** Data Storage and Management:**

- Set up a centralized data repository or server to collect, store, and manage data from the IoT noise sensors.

- Implement data retention policies and backup procedures to ensure data integrity.

**Power Supply:**

- Determine the power source for the sensors. Battery-powered sensors may require periodic maintenance, while wired sensors will need a stable power supply.

  - Consider alternative energy sources, such as solar panels, for remote or outdoor sensor locations.

**Data Analysis and Visualization:**

  - Develop or choose appropriate software for data analysis and visualization. This software should allow you to process and interpret noise data effectively.

  - Create user-friendly dashboards or reports for stakeholders and the public to access and understand the noise data.

** Regulatory Compliance:**

  - Ensure that the deployment and operation of the sensors comply with local regulations, privacy laws, and data protection requirements.

  - Obtain any necessary permits or approvals for the installation of sensors in public areas.

** Testing and Calibration:**

  - Test the sensors to ensure accurate noise measurement and data transmission.

  - Periodically calibrate the sensors to maintain their accuracy.

**Maintenance and Support:**

  - Establish a maintenance plan to regularly inspect and service the sensors.

  - Provide a mechanism for reporting sensor malfunctions or issues and offer technical support.

** Public Engagement:**

  - Promote public awareness of the noise monitoring project through outreach and education campaigns.

  - Make noise data and findings available to the public through websites or mobile apps.

** Data Privacy and Security:**

  - Implement robust security measures to protect sensor data and user privacy.

  - Anonymize and aggregate data when sharing it with the public to protect individual privacy.

** Data Validation and Quality Control:**

  - Implement quality control processes to ensure the accuracy and reliability of collected data.

  - Regularly validate and cross-reference sensor data with other sources if possible.

** Evaluation and Adaptation:**

  - Periodically evaluate the effectiveness of the sensor deployment in achieving project objectives.

- Be prepared to adapt the deployment strategy based on the results of evaluations and changing needs.

**Reporting and Accountability:**

- Provide regular reports to stakeholders, regulatory authorities, and the public about the noise levels and the impact of noise mitigation measures.

By following this comprehensive plan, you can effectively deploy IoT noise sensors in public areas to measure noise levels and achieve your project objectives.

**3.Noise Pollution Information Platform: Design a web-based platform and mobile app to display real-time noise level data to the public.**

Designing a web-based platform and mobile app to display real-time noise level data to the public requires careful planning and consideration of user experience, data visualization, and functionality. Here is a high-level outline of the steps involved in creating such a platform:

**Define Project Objectives and Scope:**

- Clearly outline the goals of the platform, such as providing real-time noise data for public awareness, and establish the scope of the project.

**User Research and Personas:**

- Conduct user research to understand the needs and preferences of your target audience.

- Create user personas to represent different user types and their specific requirements.

** Platform Architecture:**

- Determine the technical architecture of the platform, including the choice of web and mobile development technologies.

- Plan for scalability to accommodate growing user numbers and data.

**Data Sources and Integration:**

- Identify the sources of real-time noise data, such as IoT sensors or government monitoring stations.

- Develop APIs or data connectors to retrieve and integrate this data into the platform.

**User Interface (UI) Design:**

- Create wireframes and mockups for the web platform and mobile app, considering the user personas and user journey.

- Design an intuitive and user-friendly interface for accessing noise level data.

**Development:**

- Develop the web-based platform and mobile app following best practices for web and mobile application development.

  - Implement features for real-time data display, user registration, and user preferences.

** Real-Time Data Visualization:**

  - Implement dynamic charts and maps to display real-time noise level data in an easy-to-understand format.

  - Allow users to filter and customize data views based on location, time, and other parameters.

**Location-Based Services:**

  - Utilize GPS or location services to provide users with noise data specific to their current location.

  - Offer location-based notifications or alerts if noise levels exceed certain thresholds.

**User Accounts and Notifications:**

  - Enable users to create accounts and set preferences for noise alerts and notifications.

  - Implement push notifications to alert users to significant noise events or updates.

**Accessibility and Inclusivity:**

  - Ensure that the platform and app are accessible to users with disabilities, following web accessibility guidelines (e.g., WCAG).

  - Provide content in multiple languages to cater to a diverse user base.

**Data Privacy and Security:**

  - Implement robust security measures to protect user data and ensure data privacy.

  - Comply with relevant data protection regulations (e.g., GDPR, CCPA).

**Testing and Quality Assurance:**

  - Conduct thorough testing to identify and address any bugs or usability issues.

  - Perform load testing to ensure the platform can handle a high volume of users and data.

** Deployment and Hosting:**

  - Deploy the web platform and mobile app to secure and reliable hosting environments.

  - Set up monitoring and logging to track platform performance and user interactions.

**Public Awareness and Outreach:**

  - Promote the platform and mobile app to the public through marketing and outreach efforts.

  - Educate users about the benefits of noise level data and how to use the platform effectively.

**Maintenance and Updates:**

- Establish a maintenance plan for regular updates, bug fixes, and feature enhancements.

- Continuously monitor and improve the platform based on user feedback and evolving needs.

**Evaluation and Analytics:**

- Implement analytics tools to track user engagement and platform usage.

- Use data-driven insights to make improvements and measure the platform's impact on public awareness.

**Support and User Assistance:**

- Provide user support channels, such as helpdesk or FAQs, to assist users with platform-related issues.

- Offer user assistance for interpreting noise data and taking action based on the information provided.

By following these steps, you can design and develop a web-based noise pollution information platform and a complementary mobile app that effectively deliver real-time noise level data to the public while prioritizing usability, security, and accessibility.

**4.Integration Approach: Determine how IoT sensors will send data to the noise pollution information platform.**

Integrating IoT sensors with the noise pollution information platform involves establishing a data pipeline for the sensors to send their data to the platform's backend infrastructure. Below, I outline a common integration approach for connecting IoT sensors to the platform:

**Sensor Selection and Configuration:**

- Choose IoT noise sensors capable of data transmission using standard communication protocols (e.g., MQTT, HTTP, CoAP).

- Configure the sensors with the necessary network settings, such as Wi-Fi credentials or cellular connectivity parameters.

**Data Collection at the Sensor:**

- Configure the sensors to collect noise level data at regular intervals or in real-time.

- Ensure that the sensors have built-in data processing capabilities, such as data aggregation or noise level averaging, if required.

**Data Transmission Protocols:**

- Select an appropriate data transmission protocol that suits the sensors and the platform. Common choices include:

  - MQTT (Message Queuing Telemetry Transport): A lightweight and efficient protocol suitable for IoT devices.

  - HTTP/HTTPS: Standard web protocols for transmitting data over the internet.

- CoAP (Constrained Application Protocol): Designed for resource-constrained IoT devices.

**Connectivity Options:**

- Depending on the sensors' capabilities and location, choose the appropriate connectivity options:

- Wi-Fi: Suitable for sensors in areas with Wi-Fi coverage.

- Cellular: Ideal for remote or mobile sensors that require wide-area coverage.

- Ethernet: Wired connectivity for stationary sensors.

**Data Encryption and Security:**

- Implement encryption mechanisms (e.g., TLS/SSL) to secure data transmission between the sensors and the platform.

- Authenticate sensors using credentials or certificates to ensure data integrity and prevent unauthorized access.

**Sensor Data Format:**

- Define a standard data format or protocol for the sensors to transmit data to the platform. This format should include noise level readings, timestamps, and sensor identifiers.

- Common data formats include JSON or XML.

**Edge Processing (Optional):**

- For resource-constrained sensors, consider implementing edge processing to preprocess data locally before transmission.

- Edge processing can reduce data transmission bandwidth and latency.

** Data Gateway (Optional):**

- If multiple sensors are deployed in proximity, consider using a data gateway or edge device to collect data from nearby sensors and relay it to the platform.

- The gateway can provide connectivity options, data aggregation, and local processing capabilities.

**Cloud or Server-Side Backend:**

- Set up a cloud-based or server-side backend infrastructure to receive, process, and store data from the sensors.

- Implement a message broker or server to handle incoming sensor data.

**API Endpoints:**

- Create RESTful or MQTT API endpoints that sensors can use to send data to the platform.

- Ensure the API endpoints are secured with authentication and authorization mechanisms.

** Data Ingestion and Processing:**

- Develop scripts or services on the backend to ingest data from the sensors, validate it, and store it in a database.

- Implement data validation and error handling to ensure data integrity.

** Real-Time Processing:**

- Set up real-time data processing components to analyze incoming data, generate alerts, or update real-time visualizations on the platform.

**Data Storage:**

- Store sensor data in a secure and scalable database, such as a time-series database or relational database.

- Implement data retention policies to manage storage costs.

**Monitoring and Logging:**

- Implement monitoring and logging solutions to track the health and performance of the sensor data ingestion process and the platform.

**Data Access APIs:**

- Develop APIs that allow the web-based platform and mobile app to retrieve and display real-time and historical sensor data.

**Security and Access Control:**

- Implement access control and authentication mechanisms to ensure that only authorized users and devices can access sensor data.

**Data Visualization:**

- Use data visualization libraries or tools to create real-time noise level charts, maps, and dashboards for users to access on the web platform and mobile app.

**Error Handling and Alerts:**

- Implement error handling and alerting mechanisms to notify administrators or support staff of any issues with data ingestion or sensor connectivity.

**Testing and Quality Assurance:**

- Conduct rigorous testing of the entire data pipeline, including sensor connectivity, data transmission, data processing, and data visualization.

**Maintenance and Scalability:**

- Establish a maintenance plan for ongoing updates and scalability to accommodate additional sensors or increased data volume.

**ESP8266**

Sound level meters are commonly utilized in sound pollution studies for the quantification of various sorts of noise, especially for industrial, environmental, mining, and aircraft noise. The reading from a sound level meter doesn't correlate well to human-perceived loudness, which is best measured by a loudness meter. Specific loudness may be a compressive nonlinearity and varies at certain levels and certain frequencies. These metrics also can be calculated in several other ways.

Here we are going to make an IoT based decibel meter that will measure the sound in decibels(dB) using a sound sensor and display it to the LCD display along with that, it will also be pushing the readings to the wokwi IoT platform making it accessible from across the world.

**Components Required :**

- Node MCU Board
- Microphone sensor
- 16*2 LCD Module
- Breadboard
- Connecting wires

**How does Microphone Module Work?**

The microphone based sound sensor is used to detect sound. It gives a measurement of how loud a sound is. The sound sensor module is a small board that mixes a microphone (50Hz-10kHz) and a few processing circuitry to convert sound waves into electrical signals. This electrical signal is fed to on-board LM393 High Precision Comparator to digitize it and is made available at the OUT pin.

The module features a built-in potentiometer for sensitivity adjustment of the OUT signal. We will set a threshold by employing a potentiometer. So that when the amplitude of the sound exceeds the edge value, the module will output LOW, otherwise, HIGH. Apart from this, the module has two LEDs. The facility LED will illuminate when the module is powered. The Status LED will illuminate when the digital output goes LOW.

The sound sensor only has three pins: VCC, GND & OUT. VCC pin supplies power for the sensor & works on 3.3V to 5V. OUT pin outputs HIGH when conditions are quiet and goes LOW when sound is detected.

**Connectivity :**

**Bluetooth Low Energy (BLE) :**

Bluetooth Low Energy is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries.

**Wi-Fi :**

Wi-Fi is a wireless networking technology that allows devices such as computers (laptops and desktops), mobile devices (smart phones and wearables), and other equipment (printers and video cameras) to interface with the Internet. It allows these devices—and many more—to exchange information with one another, creating a network.

Internet connectivity occurs through a wireless router. When you access Wi-Fi, you are connecting to a wireless router that allows your Wi-Fi-compatible devices to interface with the Internet.

**ZIGBEE:**

Zigbee is a standards-based wireless technology developed to enable low-cost, low-power wireless machine-to-machine (M2M) and internet of things (IoT) networks. Zigbee is for low-data rate, low-power applications and is an open standard.

**Cloud:**

**Beeceptor:**

Beeceptor is an easiest and straightforward HTTP request & response service designed to provide you with static & predefined HTTP responses. Build a no-code Rest/SOAP API for storing and retrieving data over HTTP.

**Protocol:**

**MQTT:**

MQTT is a standards-based messaging protocol, or set of rules, used for machine-to-machine communication. Smart sensors, wearables, and other Internet of Things (IoT) devices typically have to transmit and receive data over a resource-constrained network with limited bandwidth.

**HTTP:**

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.

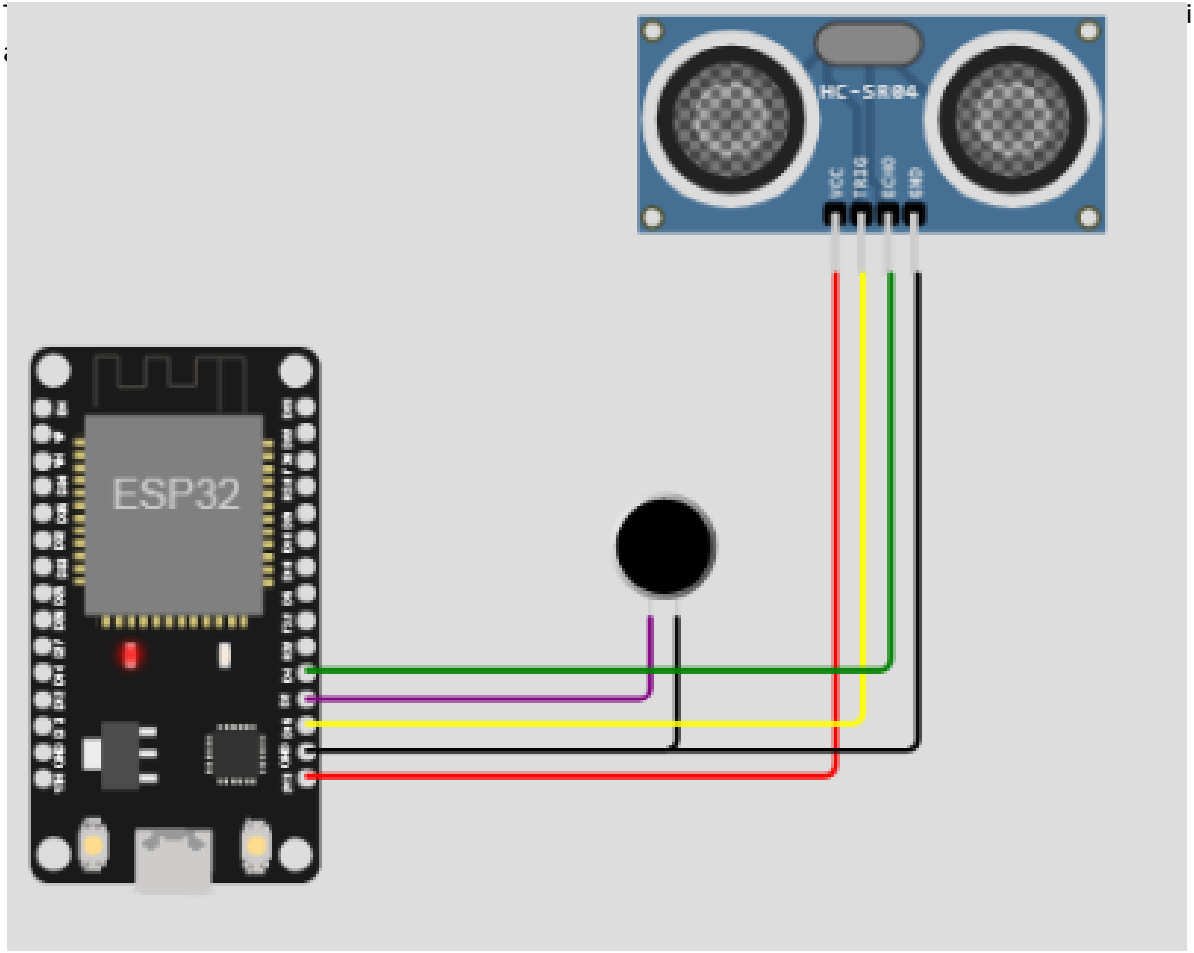
**AMQP:**

The Advanced Message Queuing Protocol is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing, reliability and security.

**Working of the Project:**

Now that you have understood the code, you can simply upload it to your NodeMCU board and the project should start working. To make sure the values are correct, I compared them to an android application on my phone that could measure sound. As you can see from the pictures, the results were quite close.

**Circuit Diagram for IoT Sound Meter:**




In the above diagram, we have connected the power pins of the sound sensor and LCD display to 3v3 and GND pin of NodeMCU. Along with that, we have also connected the SCL and SDA pins of the module to D1 and D2 respectively, and the OUT pin of the sound sensor to A0 pin.

**Program for IoT Decibel Meter:**

Here, we have to develop a code that takes input from the sound sensor and maps it value to decibels and after comparing the loudness, it should not only print it to the 16*2 LCD display but should also send it to the wokwi server.

The complete code for this project can be found at the bottom of this page. You can directly copy-paste it in your IDE and change only three parameters i.e. SSID, pass, and auth token. The explanation of the code is as follows.

In the very first part of the code, we have included all the necessary libraries and definitions. Also, we have defined the necessary variables and objects for further programming.

Further ahead, we have created a wokwi function to handle the virtual pin that our gauge is connected to. We are simply sending the values stored in the dB variable to the V0 pin.

In the setup part of the code, we are defining the pin mode as input and beginning the LCD display as well as the Blynk function. In the setup part of the code, we are defining the pin mode as input and beginning the LCD display as well as the Blynk function

**Python Code :**

```
import machine
import time
import urequests
import ujson
import network
import math

# Define your Wi-Fi credentials
wifi_ssid = 'OPPO_A78(5G)'
wifi_password = 'Bhasith786@noisE'  # actual Wi-Fi password

# Connect to Wi-Fi
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(wifi_ssid, wifi_password)

# Wait for Wi-Fi connection
while not wifi.isconnected():
    pass

# Define ultrasonic sensor pins (Trig and Echo pins)
ultrasonic_trig = machine.Pin(15, machine.Pin.OUT)
ultrasonic_echo = machine.Pin(4, machine.Pin.IN)

# Define microphone pin
microphone = machine.ADC(2)
calibration_constant = 2.0
noise_threshold = 60  # Set your desired noise threshold in dB

# Firebase Realtime Database URL and secret
firebase_url = 'https://noise-pollution-bd0ab-default-rtdb.asia-
southeast1.firebasedatabase.app/'  # Replace with your Firebase URL
firebase_secret = 'nBsgyQFTqHUe4qExlaZX6VL3mpf5gn6BlpnMiuR0'  # Replace with your
Firebase secret

def measure_distance():
```

```
    # Trigger the ultrasonic sensor
    ultrasonic_trig.value(1)
    time.sleep_us(10)
    ultrasonic_trig.value(0)

    # Measure the pulse width of the echo signal
    pulse_time = machine.time_pulse_us(ultrasonic_echo, 1, 30000)

    # Calculate distance in centimeters
    distance_cm = (pulse_time / 2) / 29.1
    return distance_cm

def measure_noise_level():
    # Read analog value from the microphone
    noise_level = microphone.read()

    noise_level_db = 20 * math.log10(noise_level / calibration_constant)
    return noise_level, noise_level_db

# Function to send data to Firebase
def send_data_to_firebase(distance, noise_level_db):
    data = {
        "Distance": distance,
        "NoiseLevelDB": noise_level_db
    }
    url = f'{firebase_url}/sensor_data.json?auth={firebase_secret}'

    try:
        response = urequests.patch(url, json=data)  # Use 'patch' instead of
'put'
        if response.status_code == 200:
            print("Data sent to Firebase")
        else:
            print(f"Failed to send data to Firebase. Status code:
{response.status_code}")
    except Exception as e:
        print(f"Error sending data to Firebase: {str(e)}")

try:
    while True:
        distance = measure_distance()
        noise_level, noise_level_db = measure_noise_level()

        print("Distance: {} cm, Noise Level: {:.2f} dB".format(distance,
noise_level_db))
```

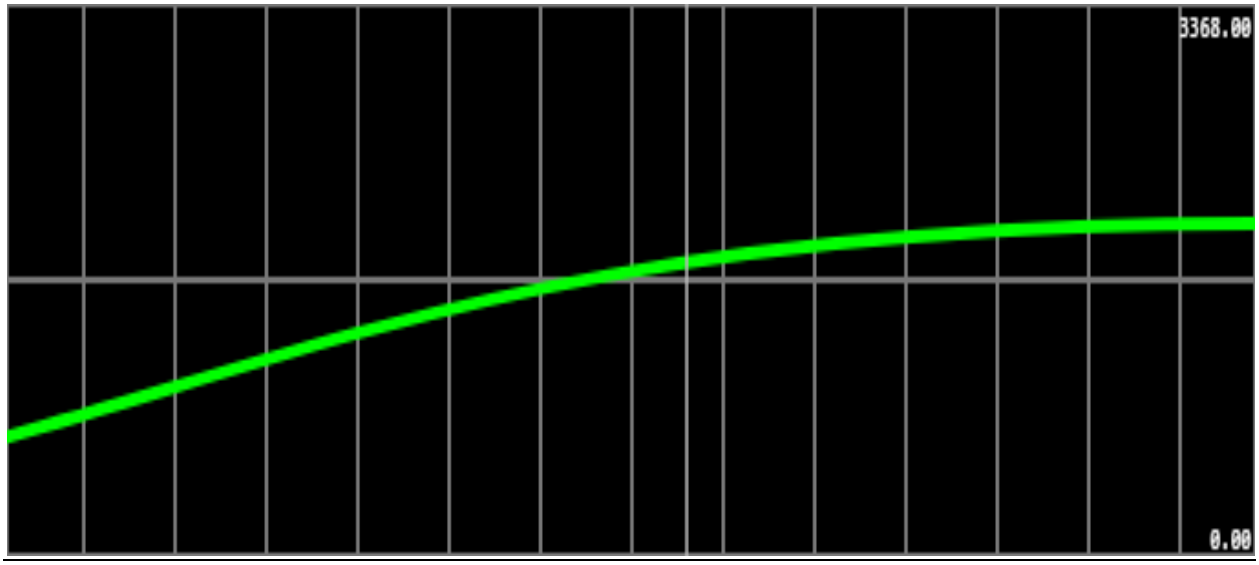
```
        if noise_level_db > noise_threshold:
            print("Warning: Noise pollution exceeds threshold!")

        # Send data to Firebase
        send_data_to_firebase(distance, noise_level_db)

        time.sleep(1)  # Adjust the sleep duration as needed

except KeyboardInterrupt:
    print("Monitoring stopped")
```

**Code for Diagram:**

```
{
  "version": 1,
  "author": "Md Kaja Bhasith",
  "editor": "wokwi",
  "parts": [
    {
      "type": "wokwi-esp32-devkit-v1",
      "id": "esp",
      "top": -72.1,
      "left": 52.6,
      "attrs": { "env": "micropython-20231005-v1.21.0" }
    },
    { "type": "wokwi-microphone", "id": "mic", "top": -16.98, "left": 263.79,
"attrs": {} },
    {
      "type": "wokwi-hc-sr04",
      "id": "ultrasonic1",
      "top": -190.5,
      "left": 274.3,
      "attrs": { "distance": "88" }
    }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "mic:1", "esp:D2", "purple", [ "v0" ] ],
    [ "mic:2", "esp:GND.1", "black", [ "v0" ] ],
    [ "ultrasonic1:VCC", "esp:3V3", "red", [ "v0" ] ],
    [ "ultrasonic1:TRIG", "esp:D15", "yellow", [ "v0" ] ],
    [ "ultrasonic1:ECHO", "esp:D4", "green", [ "v0" ] ],
    [ "ultrasonic1:GND", "esp:GND.1", "black", [ "v0" ] ]
  ],
  "serialMonitor": { "display": "plotter" },
  "dependencies": {}
}
```

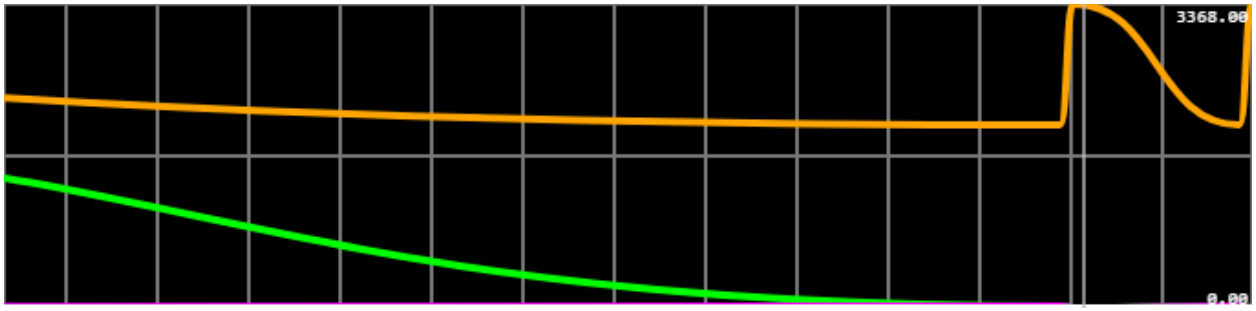
**Output:**

Lower Noise in Environment:



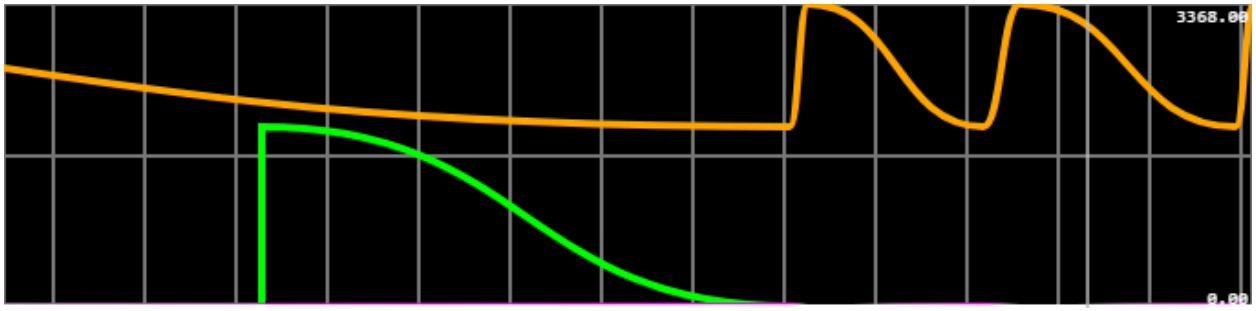

In_dB: 50

Normal Noise in Environment:




In_dB: 55

High Noise in Environment:




In_dB: 89