

NOISE POLLUTION MONITORING

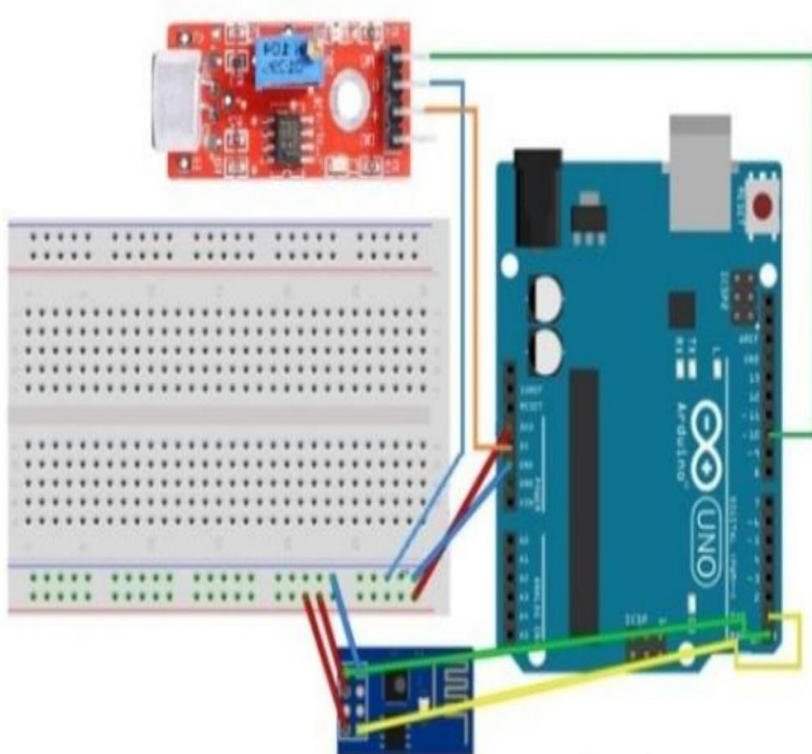
Noise pollution is a growing issue these days. It is necessary to monitor quality and keep it under control for a better future and healthy living for all. Here we propose a noise pollution monitoring system that allows us to monitor and check live as noise pollution in the particular area through IOT. Noise pollution is caused by increased use of machinery and resources as a result of industrialization. It hurts both humans and animals. Noise pollution is becoming a bigger problem, therefore it is important to keep an eye on it for a brighter future and a healthier lifestyle for everyone. In recent years, pollution has had a direct impact on people.

COMPONENTS:

1. Arduino UNO
2. MQ135(Gas sensor)
3. LM393(Noise sensor)
4. ESP8266 WIFI Module
5. 16*2 LCD Display
6. LED
7. Buzzer

TABLE CONNECTION AND CIRCUIT:

Module	Arduino Uno
LM 393 Sound Detection Sensor	
A0	10
Vcc or +	5V
G	Gnd
ESP8266 WiFi Module	
Tx	Rx
Rx	Tx
CH_PD	3.3 V
VCC	3.3 V
Gnd	Gnd



CIRCUIT DIAGRAM OF THE PROPOSED SYSTEM

WORKING PRINCIPLE

ARDUINO UNO: Arduino uno arduino is 8 bit microcontroller board based on the atmega328p. The operating voltage is 5v. It has 14 pins digital input output pins (of which can be used 6 as pwm output) oscillator frequency is 16 mhz it contains everything needed to support the microcontroller simply connect it to a computer with usb cable. It has 6 analog input pins.

MQ135:MQ135 gas sensor the MQ135 is a gas sensor it used for detecting or sensing harmful gases in the atmosphere. It has wide detecting scope. It gives fast response and also it high sensitivity sensor. It is simple and long life device. they are used in air quality control equipment for building offices are suitable for detecting of NH₃, alcohol, benzene, smoke CO₂ etc.

LM393: LM393 sound sensor the sound sensor module provide an easy way to detect sound and it generally used for detecting sound intensity. Module detect the sound has exceeded a threshold value. Sound is detected via microphone and fed into an lm393 opamp. The sound level adjust through pot. The sound increases set value output is low. These module work on DC 3.3-5 voltage.

ESP8266: The esp8266 WIFI module is a self contained soc with integrated TCP/IP protocol stack that can give any microcontroller access to your WIFI network. The ESP8266 is capable of either hosting an application or offloading all WIFI networking functions from another application processor.

16*2LCD DISPLAY: LCD is used for to display the condition there are three conditions in air pollution and three conditions in noise pollution

means air and sound is clear, moderately polluted or highly polluted that is displayed on LED.

PYTHON PROGRAM FOR NOISE POLLUTION MONITORING:

tion monitoring system is a complex project that involves hardware and software components. Below, I'll provide you with a high-level Python program outline for such a project. Please note that this is a simplified example, and you'll need specific sensors and IoT hardware to build a real-world system.

SOURCE CODE:

```
python

import time

import random

from mqtt import MQTTClient # You'll need an MQTT library

# Set up MQTT client and connect to your broker

client = MQTTClient("noise_monitor")

client.connect("your_broker_ip", port=1883)

while True:

    # Simulate noise sensor data (replace with actual sensor readings)

    noise_level = random.randint(40, 100) # Example noise level in Db

# Publish the noise data to a specific MQTT topic

topic = "noise_sensor/data"
```

```
payload = f"Noise Level: {noise_level} dB"

client.publish(topic, payload)

# You can also include logic to trigger alerts based on noise
thresholds

if noise_level > 80:

    alert_topic = "noise_sensor/alert"

    alert_payload = "High noise level detected!"

    client.publish(alert_topic, alert_payload)

# Sleep for a specific interval before taking the next reading

time.sleep(10) # Adjust the interval as needed

# Close the MQTT client connection when done

client.disconnect()
```

In this program:

1. We import necessary libraries, including an MQTT client for communication.
2. We set up an MQTT client and connect it to your MQTT broker.
3. Inside the loop, we simulate noise sensor data using a random value.
4. We publish the noise level data to an MQTT topic.
5. You can include logic to trigger alerts based on predefined noise thresholds.

6. We introduce a delay to control the frequency of data readings.
7. The program runs indefinitely, periodically collecting and publishing noise data.

To build:

Building a complete IoT-based noise pollution monitoring system requires a combination of hardware and software components. Here's a step-by-step guide to build such a system:

Hardware Components:

1. **Noise Sensor:** Choose a suitable noise sensor that can measure sound levels in decibels (dB). Common sensors include microphones with integrated preamplifiers or dedicated noise sensors.
2. **Microcontroller:** Select a microcontroller platform like Raspberry Pi, Arduino, or ESP32 to interface with the noise sensor, collect data, and communicate with the cloud or a local server.
3. **Power Supply:** Provide a stable power source for your microcontroller and sensor, whether through a battery or a dedicated power supply.
4. **Internet Connectivity:** For IoT functionality, you'll need an internet connection, typically through Wi-Fi, Ethernet, or a cellular module.
5. **Enclosure:** Protect your hardware components from environmental factors by placing them in a weatherproof enclosure.

Software Components:

1. **Sensor Interface:** Write or use existing code to interface with the noise sensor and obtain noise level readings.

2. **Data Processing:** Process the sensor data, including calibration, filtering, and data smoothing, to obtain accurate noise level measurements.
3. **IoT Communication:** Implement communication protocols to send the data to a central server or cloud platform. MQTT or HTTP can be used for this purpose.
4. **Cloud Platform (Optional):** If you prefer cloud-based IoT, set up an IoT platform like AWS IoT, Google Cloud IoT, or Microsoft Azure IoT to receive and store the data.
5. **Database:** Store the collected data in a database for long-term storage and analysis. You can use databases like MySQL, PostgreSQL, or cloud-based solutions like AWS DynamoDB.
6. **User Interface:** Develop a web-based dashboard or mobile app to visualize noise data, set thresholds, and receive alerts. You can use technologies like HTML, CSS, JavaScript, and frameworks like React or Vue.js.
7. **Alerting System:** Implement alerting mechanisms that notify users or stakeholders when noise levels exceed predefined thresholds. This can be done using email, SMS, or push notifications.

Steps to Build:

1. **Connect the Hardware:** Assemble the hardware components, including the noise sensor and microcontroller. Ensure proper wiring and connections.

2. **Install the Operating System:** Set up the operating system (e.g., Raspbian for Raspberry Pi) on your microcontroller and configure the network connection.
3. **Sensor Calibration:** Calibrate the noise sensor to provide accurate dB readings. This often involves testing it in a controlled noise environment and adjusting the sensor's output accordingly. process it, and send it to your chosen data storage solution (database or cloud platform).
4. **Software Development:** Write the software to collect noise data,
5. **Cloud Integration:** If using a cloud platform, create a project, set up IoT device management, and configure data ingestion.
6. **Database Configuration:** Set up a database to store the collected noise data, including the time and location of measurements.
7. **User Interface:** Develop a user-friendly interface to display noise data and configure alert settings.
8. **Alerting System:** Implement an alerting system that triggers notifications when noise levels cross specified thresholds.
9. **Testing and Deployment:** Thoroughly test your system in real-world conditions. Make any necessary adjustments and deploy the monitoring system in your target environment.
10. **Maintenance and Monitoring:** Regularly monitor and maintain the system to ensure its proper functioning.