Noise Pollution Monitoring using IoT

Phase-4 Document submission

TEAM MEMBER

NAME: K RAKESH SARAN

Reg No: 411421104038

Project: Monitoring environmental noise pollution.

Phase-4: Development Part-2

INTRODUCTION:

» In an increasingly urbanized world, noise pollution has become a significant concern, impacting the well-being of individuals and the environment. To address this issue, we have embarked on an innovative project harnessing the power of the Internet of Things (IoT) to monitor and manage noise pollution effectively. This project is a holistic approach that incorporates various activities, from feature engineering to model training and evaluation, aimed at creating a sustainable solution to this modern-day challenge.The project's foundation lies in the utilization of IoT devices, such as sensors and data communication technologies, to gather real-time acoustic data from diverse urban and industrial environments. This data serves as the bedrock for our efforts to analyze, understand, and ultimately mitigate noise pollution.

PROJECT OBJECTIVE:

» Real-Time Noise Data Collection: The primary objective of this project is to develop an IoT-based noise pollution monitoring system that can continuously collect real-time acoustic data from various urban and industrial areas. This data will serve as the foundation for analysis and decision-making.

» Data Analysis and Feature Engineering: To understand the patterns and sources of noise pollution, the project aims to implement advanced data analysis techniques, including feature engineering, to extract meaningful information from raw acoustic data.

» Model Training and Predictive Analytics: The project seeks to build machine learning models that can predict noise pollution levels accurately and identify sources of excessive noise. This includes developing algorithms that can anticipate noise spikes, correlate with specific events, and provide insights for mitigation strategies.

» Long-Term Sustainability: Ensuring the sustainability of the monitoring system is an overarching goal. This involves developing a maintenance plan, data storage and management strategies, and the establishment of partnerships with relevant stakeholders for continued support.

HARDWARE COMPONENTS USED:

ESP8266 NodeMCU

KY-038

SSD-1306 OLED

CONNECTION INSTRUCTIONS:

1.Connect the OLED Display:

» Connect the OLED display's VCC pin to the ESP8266's 3.3V pin.
» Connect the OLED display's GND pin to the ESP8266's GND pin.
» Connect the OLED display's SDA pin to the ESP8266's D2 (NodeMCU)
» Connect the OLED display's SCL pin to the ESP8266's D1 (NodeMCU)

2.Connect the KY-038 Sound Sensor:

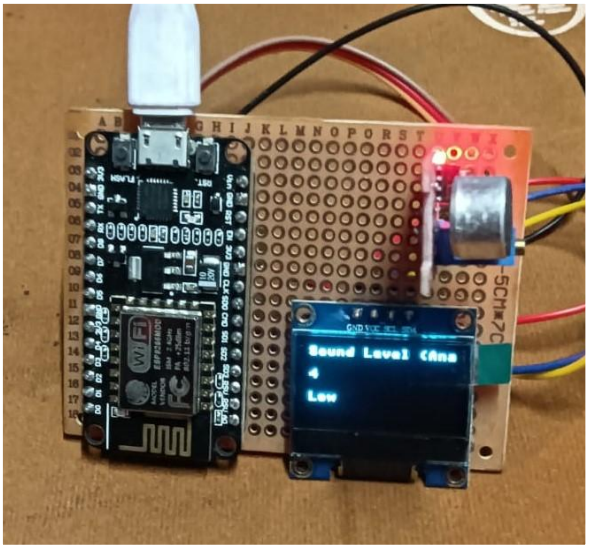
» Connect the KY-038 sensor's GND pin to the ESP8266's GND pin.
» Connect the KY-038 sensor's + pin to the ESP8266's 3.3V or 5V pin.
» Connect the KY-038 sensor's OUT pin to a digital pin on the ESP8266.

3.Power Supply:

» The ESP8266 board is properly powered USB or with battery module. Connect the 3.3V or 5V and GND pins to a reliable power source, and don't forget to connect the common GND between the ESP8266, OLED display, and KY-038 sound sensor.

NOISE POLLUTION MONITORING MODULE:

» The prepared noise pollution module for monitoring the environmental noise pollution is connected as per the above given instruction's.



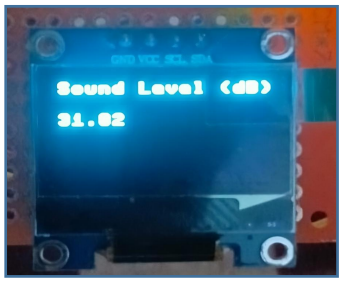
Program:

```
import machine
import ssd1306
from machine import ADC, I2C, Pin
import time

# I2C pins (D1 for SDA, D2 for SCL on ESP8266 NodeMCU)
i2c = I2C(sda=Pin(4), scl=Pin(5))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

# KY-038 sound sensor connected to analog pin A0
sound_sensor = ADC(0)

# Calibration values for your specific setup (analog value -> Hertz)
calibration_value = 1000    # Replace with your calibration value
frequency_unit = "Hz"    # Unit for the frequency value

def read_sound_level():
    # Read the analog value from the sound sensor
    sound_value = sound_sensor.read()
    return sound_value
```

```
    def convert_to_hertz(sound_value, calibration_value):
        # Convert analog value to Hertz using calibration value
        if sound_value > 0:
            frequency = (sound_value / calibration_value)    # Customize the formula based on calibration
            return frequency
        else:
            return 0    # Minimum value (or customize as needed)

    def display_sound_value(sound_frequency, unit):
        oled.fill(0)    # Clear the display
        oled.text("Sound Frequency:", 0, 0)

        oled.text("{:.2f} {}".format(sound_frequency, unit), 0, 20)

        oled.show()

    while True:
        sound_value = read_sound_level()
        sound_frequency = convert_to_hertz(sound_value, calibration_value)
        display_sound_value(sound_frequency, frequency_unit)
        time.sleep(1)    # Update the display every second
```

OBTAINED OUTPUT:

» The code is executed on physical hardware and provide a real time output for the MicroPython Script and the noise value is displayed in the provided SSD1306 OLED display and update's the noise level every 1 second.



Corrections to be fixed:

» The noise sensor used in this project has to be callibirated to produce the accurate noise readings for the environmental usage and noise evaluation.

» The output which is displayed on the SSD1306 OLED has to modified with graph animations and noise alerts according to the noise observed by the sound sensor.

» The noise pollution monitoring module has to be powered with an external power source and so it can be taken anywhere and the battery can be recharged to increase the module scalability.

Conclusion:

The project involves creating a MicroPython-based system to measure and display sound levels on an ESP8266 NodeMCU using a KY-038 sound sensor and an OLED screen. Here are the key points to summarize the project,

Hardware Setup: The project uses an ESP8266 NodeMCU microcontroller, an I2C OLED screen, and a KY-038 sound sensor. The sound sensor is connected to an analog pin of the NodeMCU.
Calibration: Accurate calibration is crucial for converting analog sensor values to meaningful units such as decibels (dB) or Hertz (Hz). You need known reference values and units for calibration.
Code Development: The MicroPython code reads the analog values from the sound sensor, performs calculations (e.g., for dB), and displays the results on the OLED screen. Various functions are defined for reading, calculating, and displaying sound levels.
Display: The sound levels are displayed on the OLED screen, updating at regular intervals (e.g., every second). The OLED screen shows dB depending on the calibration and code setup.
Accuracy and Limitations: Accurate sound level measurements require proper calibration and possibly specialized equipment. The code provides a basic representation and may not be suitable for precise noise level monitoring without proper calibration.
Further Development: For more accurate measurements or real-world applications, you might consider using calibrated microphones or sound level meters and more advanced hardware.