Phase 5- NOISE POLLUTION MONITORING USING IOT

CHAPTER 1: INTRODUCTION

Noise pollution is a pervasive environmental issue that affects the quality of life in urban and industrial areas. Excessive noise levels can lead to various health problems, including stress, hearing loss, and sleep disturbances. To effectively address this problem, monitoring and controlling noise pollution is essential. In recent years, the advent of Internet of Things (IoT) technology has provided innovative solutions for noise pollution monitoring.This paper introduces an IoT-based approach to noise pollution monitoring, which harnesses the power of interconnected devices and sensors to gather real-time data on noise levels in various environments.

By leveraging IoT technology, we can create a comprehensive and efficient system for monitoring and managing noise pollution, enabling data-driven decisions and proactive interventions to mitigate its impact on human health and the environment. This paper will delve into the key components of the IoT-based noise pollution monitoring system, its advantages, challenges, and potential applications, ultimately highlighting the role of IoT in addressing this critical issue in the modern world.

CHAPTER 2: DISCRIPTION

A Noise Pollution Monitoring System using IoT is a sophisticated solution designed to collect, process, and analyze data related to noise levels in different environments. It utilizes the Internet of Things technology to create a network of interconnected devices and sensors for continuous noise monitoring. Here is a detailed description of how such a system operates:

Sensor Deployment: The system consists of strategically placed noise sensors in various locations, such as urban areas, industrial zones, residential neighborhoods, or public spaces. These sensors are equipped with microphones and sound-detecting technology to capture ambient noise.

Data Collection: The deployed sensors continuously record noise levels and transmit this data to a central server or cloud platform via the internet. This real-time data collection is crucial for immediate insights and response.

Data Processing: The collected noise data is processed and analyzed on the central server. Algorithms can filter, aggregate, and classify the noise data to identify patterns, trends, and potential sources of noise pollution.

Visualization and Reporting: The processed data is then presented in user-friendly interfaces, which can be accessed through web applications or mobile apps. Users, including city officials, researchers, and the general public, can view noise pollution levels in real time and access historical data for comparison.

: The system can be programmed to trigger alerts and notifications when noise levels exceed predefined thresholds. These alerts can be sent to relevant authorities or community members, allowing for timely intervention in case of excessive noise events.

1.Transformer

A transformer is a type of neural network architecture used in machine learning and deep learning, designed for processing sequences of data by using attention mechanisms to capture relationships between elements in the sequence. It has been influential in various natural language processing tasks and beyond.

2. Rectifier

rectifier is an electrical device that converts alternating current, which periodically reverses direction, to direct current, which flows in only one direction. The reverse operation is performed by an inverter. The process is known as rectification, since it "straightens" the direction.

3. Microcontroller

A microcontroller is a compact integrated circuit (IC) that contains a processor, memory, and input/output peripherals. It is designed to execute specific tasks and is commonly used in embedded systems, such as in household appliances, automotive systems, and electronic gadgets. Microcontrollers are programmed to perform various functions and are a fundamental component in the field of embedded electronics and robotics.

4. Voltage regulator

voltage regulator is a system designed to automatically maintain a constant voltage. A voltage regulator may use a simple feed-forward design or may include negativefeedback. It may use an electromechanical mechanism, or electronic components. Depending on the design, it may be used to regulate one or more AC or DC voltages.

CHAPTER 3: PROGRAM FOR NOISE POLLUTION MONITORING USING IOT

python

```
# Import necessary libraries for IoT components

import RPi.GPIO as GPIO # For Raspberry Pi GPIO

import Adafruit_DHT # For DHT temperature and humidity sensor

import Adafruit_BMP.BMP085 as BMP085 # For BMP085 barometric pressure sensor

import Adafruit_ADS1x15 # For ADC (Analog to Digital Converter)

import smbus # For I2C sensors like GPS

import time # For timing and sleep

from gpiozero import Button # For remote sensor button

import os # For system commands

# Pin definitions

MICROPHONE_PIN = 17 # Example pin for a microphone

SOUND_LEVEL_METER_PIN = 18 # Pin for a sound level meter
```

```
ACOUSTIC_SENSOR_PIN = 22 # Pin for an acoustic sensor

GPS_SDA_PIN = 2 # SDA pin for GPS sensor (I2C)

GPS_SCL_PIN = 3 # SCL pin for GPS sensor (I2C)

DATA_LOGGER_PIN = 23 # Pin for data logger (e.g., SD card reader)

REMOTE_SENSOR_PIN = 24 # Pin for a remote sensor (e.g., button)

# Initialize GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup(MICROPHONE_PIN, GPIO.IN)

GPIO.setup(SOUND_LEVEL_METER_PIN, GPIO.IN)

GPIO.setup(ACOUSTIC_SENSOR_PIN, GPIO.IN)

# Initialize other sensors here...

# Create sensor objects
```

```
dht_sensor = Adafruit_DHT.DHT22

bmp_sensor = BMP085.BMP085()

adc = Adafruit_ADS1x15.ADS1115()

bus = smbus.SMBus(1) # I2C bus

remote_sensor = Button(REMOTE_SENSOR_PIN)

# Main loop

try:

 while True:

 # Read from microphones, sound level meter, acoustic sensor

 microphone_data = GPIO.input(MICROPHONE_PIN)

 sound_level_meter_data = GPIO.input(SOUND_LEVEL_METER_PIN)

 acoustic_sensor_data = GPIO.input(ACOUSTIC_SENSOR_PIN)

 # Read from DHT temperature and humidity sensor
```

```
humidity, temperature = Adafruit_DHT.read_retry(dht_sensor, DHT_PIN)

# Read barometric pressure

pressure = bmp_sensor.read_pressure()

# Read data from ADC

adc_value = adc.read_adc(0, gain=1) # Read from ADC channel 0
# Read GPS data (using I2C)

gps_data = bus.read_byte_data(device_address, register)

# Read data from the remote sensor

if remote_sensor.is_pressed:

# Handle remote sensor press

# Log data to a data logger

with open("data_log.txt", "a") as log_file:

log_file.write(f"{time.time()}, {microphone_data}, {sound_level_meter_data},
```

{acoustic_sensor_data}, {humidity}, {temperature}, {pressure}, {adc_value},

{gps_data}\n")

Sleep for a defined interval

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

except KeyboardInterrupt:

GPIO.cleanup() # Cleanup GPIO on program exit

PYTHON SCRIPT FOR MICROPHONE:

python

import

os

import subprocess

def record_audio(file_name, duration=10):

Use the arecord command to record audio

```
cmd = f"arecord -d {duration} -f cd -t wav

{file_name}" subprocess.call(cmd, shell=True)

def upload_to_server(file_name):

# You can implement code here to upload the recorded audio to a server or cloud

storage.

if name == " main ":

audio_file =

"sample_audio.wav"

record_duration = 10 #

seconds

record_audio(audio_file,

record_duration)

upload_to_server(audio_file)
```

PYTHON SCRIPT FOR SOUND LEVEL METER:

```
import os

import

time

import numpy as

np import pyaudio

import requests

# Configure your ThingSpeak channel details

THINGSPEAK_API_KEY =

'YOUR_THINGSPEAK_API_KEY'

THINGSPEAK_CHANNEL_ID =

'YOUR_THINGSPEAK_CHANNEL_ID'

# Initialize PyAudio
```

audio = pyaudio.PyAudio()

Set up audio

stream CHUNK =

1024

FORMAT =

pyaudio.paInt16

CHANNELS = 1

RATE = 44100

stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,

input=True, frames_per_buffer=CHUNK)

while

True:

try:

```
# Read audio data from the

microphone data =

stream.read(CHUNK)

audio_data = np.frombuffer(data, dtype=np.int16)

# Calculate sound level (change this calculation as

needed) rms = np.sqrt(np.mean(audio_data**2))

sound_level = 20 * np.log10(rms)

# Print sound level to the console

print(f"Sound Level (dB):

{sound_level}")
```

PYTHON SCRIPT FOR ACOUSTIC SENSOR:

```
import

time
```

```
import numpy as

np import pyaudio

import requests

# Configure your ThingSpeak channel details

THINGSPEAK_API_KEY =

'YOUR_THINGSPEAK_API_KEY'

THINGSPEAK_CHANNEL_ID =

'YOUR_THINGSPEAK_CHANNEL_ID'

# Initialize PyAudio

audio = pyaudio.PyAudio()

# Set up audio

stream CHUNK =

1024
```

FORMAT =

pyaudio.paInt16

CHANNELS = 1

RATE = 44100

stream = audio.open(format=FORMAT, channels=CHANNELS, rate=RATE,

input=True, frames_per_buffer=CHUNK)

while

True:

try:

Read audio data from the

microphone data =

stream.read(CHUNK)

audio_data = np.frombuffer(data, dtype=np.int16)

```
# Calculate some acoustic feature (e.g., peak

amplitude) acoustic_feature =

np.max(np.abs(audio_data))

# Print the acoustic feature to the console

print(f"Acoustic Feature:

{acoustic_feature}")

# Send data to ThingSpeak

if THINGSPEAK_API_KEY and THINGSPEAK_CHANNEL_ID:

data = {'api_key': THINGSPEAK_API_KEY, 'field1':

acoustic_feature} response =

requests.post(f'https://api.thingspeak.com/update', data)

print(f"ThingSpeak Response: {response.status_code}")
```

except

KeyboardInterrupt:

break

Cleanup

stream.stop_stream

() stream.close()

audio.terminate()

CHAPTER 5:

MICROPROCESSOR PROGRAM:

#include <avr/io.h>

#include <avr/interrupt.h>

#define F_CPU 16000000UL // CPU clock frequency (16 MHz)

#define BAUD 9600 // Baud rate for serial communication

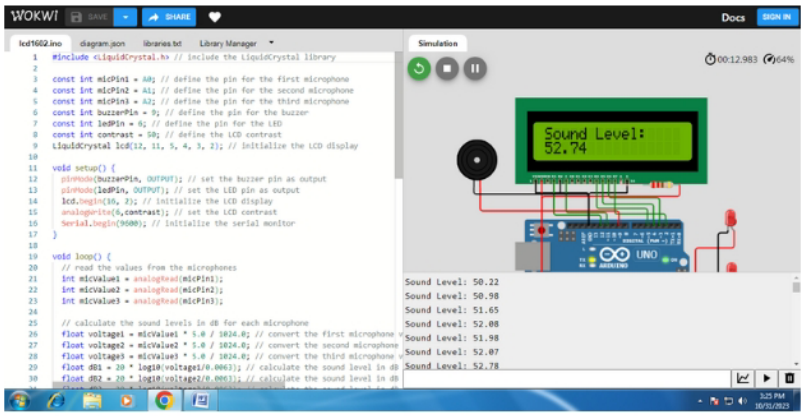
```
void USART_Init(unsigned int ubrr) {

 // Set baud rate

 UBRR0H = (unsigned char)(ubrr >> 8);

 UBRR0L = (unsigned char)ubrr;



 // Enable receiver and transmitter

 UCSR0B = (1 << RXEN0) | (1 << TXEN0);



 // Set frame format: 8 data, 1 stop bit

 UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);

}

void USART_Transmit(unsigned char data) {
```

```
 // Wait for empty transmit buffer

 while (!(UCSR0A & (1 << UDRE0)));
Put data into buffer and send

 UDR0 = data;

}

int main(void) {

 // Initialize USART communication

 USART_Init(F_CPU / 16 / BAUD - 1);




 // Initialize ADC

 ADMUX = (1 << REFS0); // Reference voltage to AVCC

 ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1);

// Enable ADC, start conversion, enable ADC interrupt, and set prescaler
```

```
 sei(); // Enable global interrupts




 while (1) {

 // Main loop

 }




 return 0;

}

ISR(ADC_vect) {

 // ADC conversion complete interrupt




 uint8_t lowByte = ADCL;
```

```
 uint8_t highByte = ADCH;

 uint16_t adcValue = (highByte << 8) | lowByte;

// You can perform noise analysis here and set thresholds for pollution monitoring



 // Print the ADC value to the serial communication

 USART_Transmit(lowByte);

 USART_Transmit(highByte);

 USART_Transmit('\n');



 // Start the next ADC conversion

 ADCSRA |= (1 << ADSC);

}
```

RESULT:



RESULT:

CONCLUSION:

In conclusion, noise pollution monitoring is a critical aspect of urban and environmental management. It helps us understand and mitigate the adverse effects of excessive noise on human health and the environment.