# Project-Based Learning (PBL) Practical Report

## 2D Traffic Density Mapping Using NodeMCU, Sound Sensor, and ThingSpeak

\*\*Group Members:\*\*

1. Sumit Singh

2. Vatsal Mishra

3. Kumar Ankit Singh

4. Mudit Srivastava

Submitted To: Dr. Hemlata Goyal

Submitted By:  
Name: Sumit Singh  
Registration No.: 229311207

## 1. Objective

To design and implement a real-time traffic density monitoring and signal adjustment system using NodeMCU, a sound sensor, and the ThingSpeak IoT platform. The system measures environmental noise levels, correlates them with traffic density, and adjusts signal durations to optimize traffic flow dynamically.

## 2. Problem Statement

Urban traffic management is a significant challenge, particularly during peak hours. Existing solutions often fail to adapt to real-time traffic conditions. This project introduces a scalable, low-cost system that uses ambient noise levels as a proxy for traffic density to adjust traffic signal durations dynamically.

## 3. Microcontroller Structure

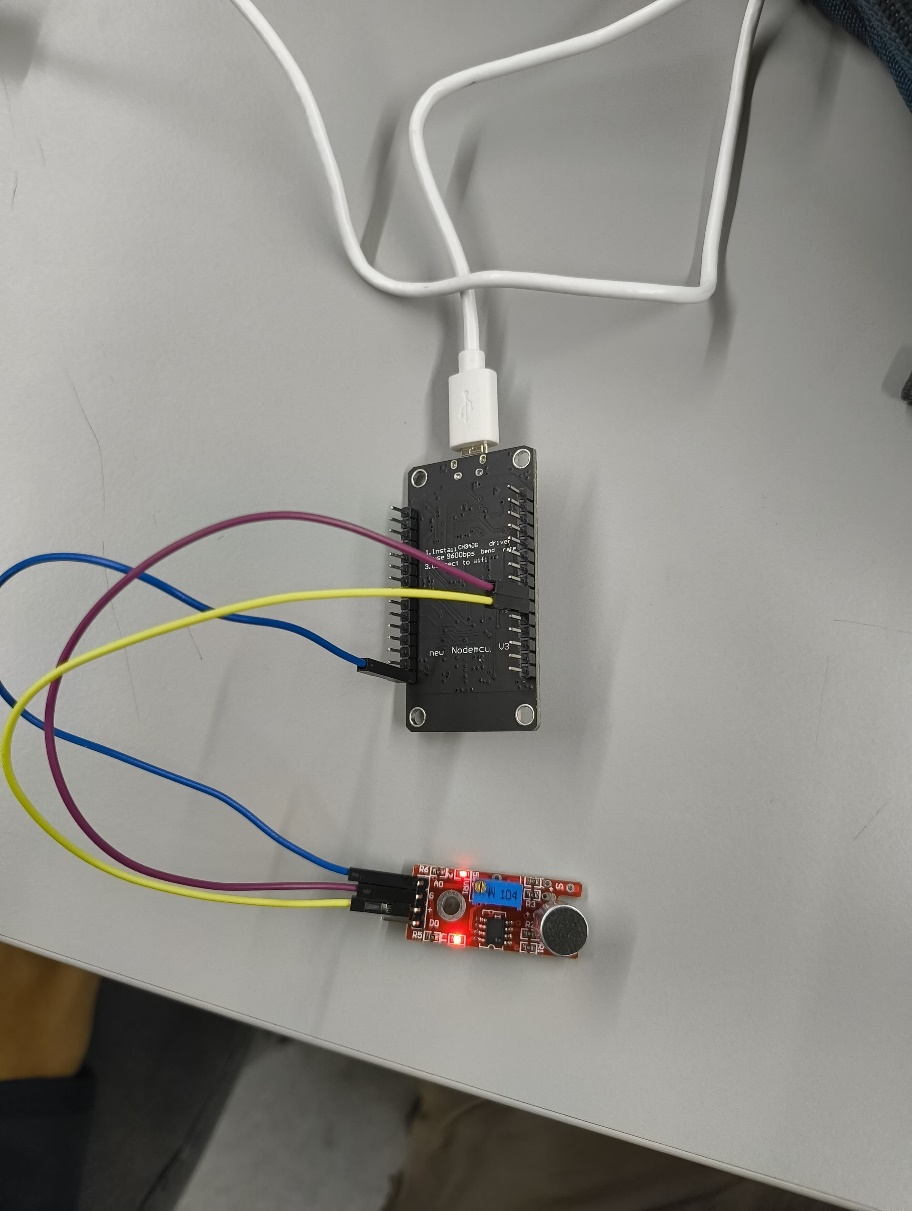
NodeMCU (ESP8266):  
- Wi-Fi Connectivity: Enables real-time data transmission to ThingSpeak.  
- Analog Input Support: Reads noise level data from the sound sensor.  
- Power Efficiency: Operates on a low-power 3.3V supply, making it suitable for IoT applications.  
  
The microcontroller collects noise level data, converts it into traffic density estimates, and sends the processed data to the cloud for further analysis and visualization.

## 4. Sensors Used

Sound Sensor (e.g., KY-038):  
- Detects ambient noise and converts it into analog signals.  
- Noise levels correlate with traffic density in this system.  
- Compact and efficient for real-time monitoring.

## 5. Circuit Diagram

Connections:

  
- Sound Sensor to NodeMCU:  
 - A0 (Sound Sensor) → A0 (NodeMCU)  
 - G (Sound Sensor) → G (NodeMCU)  
 - + (Sound Sensor) → 3V3 (NodeMCU)  
  
This setup allows accurate detection of environmental noise and efficient data transmission.

## 6. Code Explanation

Functionality:  
1. Connects the NodeMCU to a Wi-Fi network.  
2. Reads analog sound data from the sound sensor.  
3. Calibrates noise levels to estimate traffic density.  
4. Sends real-time traffic density data to ThingSpeak.  
5. Uses Python for adjusting traffic signal durations based on density.

Arduino Code:

#include <ESP8266WiFi.h>  
#include <ThingSpeak.h>  
  
const char\* ssid = "YourWiFi";  
const char\* password = "YourPassword";  
unsigned long channelID = 123456;   
const char\* apiKey = "YourAPIKey";  
  
WiFiClient client;  
const int soundPin = A0;  
  
void setup() {  
 Serial.begin(115200);  
 WiFi.begin(ssid, password);  
 ThingSpeak.begin(client);  
  
 while (WiFi.status() != WL\_CONNECTED) {  
 delay(500);  
 Serial.print(".");  
 }  
 Serial.println("\nConnected to Wi-Fi!");  
}  
  
void loop() {  
 int soundLevel = analogRead(soundPin);  
 float noiseDb = 20 \* log10(soundLevel + 1);  
 int trafficDensity = min(500, max(50, (int)(noiseDb \* 10)));  
  
 Serial.print("Noise (dB): ");  
 Serial.print(noiseDb);  
 Serial.print(", Traffic Density: ");  
 Serial.println(trafficDensity);  
  
 ThingSpeak.setField(1, noiseDb);  
 ThingSpeak.setField(2, trafficDensity);  
  
 if (ThingSpeak.writeFields(channelID, apiKey) == 200) {  
 Serial.println("Data successfully sent to ThingSpeak.");  
 } else {  
 Serial.println("Failed to send data to ThingSpeak.");  
 }  
  
 delay(5000);  
}

Traffic Light Management System Code –

import numpy as np

import pandas as pd

import time

import random

import requests

# ThingSpeak API details

THINGSPEAK\_CHANNEL\_ID = "2756320"

THINGSPEAK\_API\_KEY = "HJ98F8G2001K4NQJ"

THINGSPEAK\_READ\_URL = f"https://api.thingspeak.com/channels/{THINGSPEAK\_CHANNEL\_ID}/fields/1.json?api\_key={THINGSPEAK\_API\_KEY}&results=1"

# Load real-time traffic data from the CSV file

def load\_real\_time\_data(filename="real\_time\_traffic1.csv"):

df = pd.read\_csv(filename)

return df

# Fetch real-time sound level data from ThingSpeak

def fetch\_sound\_level():

try:

response = requests.get(THINGSPEAK\_READ\_URL)

data = response.json()

if 'feeds' in data and data['feeds']:

latest\_feed = data['feeds'][-1]

sound\_level\_db = float(latest\_feed['field1'])

return sound\_level\_db

else:

print("No data available from ThingSpeak.")

return None

except Exception as e:

print(f"Error fetching data from ThingSpeak: {e}")

return None

# Adjust traffic signal durations based on traffic density

def adjust\_traffic\_signals(df):

for i, row in df.iterrows():

scaling\_factor = row['traffic\_density'] / 500

df.at[i, 'green\_light\_duration'] = min(90, max(30, row['green\_light\_duration'] \* scaling\_factor))

df.at[i, 'red\_light\_duration'] = 120 - df.at[i, 'green\_light\_duration']

return df

# Simulate traffic diversion if a particular intersection's traffic is too high

def traffic\_diversion(df, threshold=400):

high\_traffic\_intersections = df[df['traffic\_density'] > threshold]

for i, row in high\_traffic\_intersections.iterrows():

least\_traffic\_index = df['traffic\_density'].idxmin()

if least\_traffic\_index != i:

diverted\_traffic = random.randint(20, 50)

df.at[i, 'traffic\_density'] -= diverted\_traffic

df.at[least\_traffic\_index, 'traffic\_density'] += diverted\_traffic

print(f"Diverting {diverted\_traffic} vehicles from {row['intersection\_id']} to {df.loc[least\_traffic\_index, 'intersection\_id']}")

return df

# Simulate traffic signal cycles

def simulate\_traffic\_cycle(df):

while True:

for i, row in df.iterrows():

print(f"\n{row['intersection\_id']} Traffic Signal Cycle:")

print(f" Green Light Duration: {row['green\_light\_duration']} seconds")

print(f" Yellow Light Duration: {row['yellow\_light\_duration']} seconds")

print(f" Red Light Duration: {row['red\_light\_duration']} seconds")

time.sleep(row['green\_light\_duration'])

print(f" Yellow Light ON")

time.sleep(row['yellow\_light\_duration'])

print(f" Red Light ON")

time.sleep(row['red\_light\_duration'])

def save\_traffic\_data\_to\_csv(df, filename="final\_traffic\_density.csv"):

df.to\_csv(filename, index=False)

print(f"Traffic data saved to {filename}")

# Traffic Management System main function

def main():

df = load\_real\_time\_data()

print("Initial Traffic Data:")

print(df)

# Fetch real-time sound level and update traffic density

sound\_level\_db = fetch\_sound\_level()

if sound\_level\_db:

df['traffic\_density'] += sound\_level\_db / 10 # Example: Map sound level to density increment

df = adjust\_traffic\_signals(df)

print("\nAdjusted Traffic Signals based on Traffic Density:")

print(df)

df = traffic\_diversion(df, threshold=400)

print("\nTraffic Data After Diversion:")

print(df)

save\_traffic\_data\_to\_csv(df)

print("\nSimulating Traffic Signal Cycles...")

simulate\_traffic\_cycle(df)

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 7. Results

1. Real-time noise levels (in dB) and corresponding traffic density are displayed on the Arduino IDE serial monitor.  
2. Traffic signal durations are dynamically adjusted based on traffic density.  
3. Data is uploaded to ThingSpeak for visualization.

