RESEARCH PROJECT REPORT

ON

# “Real-Time Emergency Siren Detection & Vehicle Mapping on Colab”

SUBMITTED TO

### **Fergusson College (Autonomous)**

FOR THE DEGREE OF

M.Sc.

(ELECTRONIC SCIENCE)

BY

#### Aadish Rajendra Bhide (236908)

### DEPARTMENT OF ELECTRONIC SCIENCE

FERGUSSON COLLEGE (Autonomous)

PUNE - 411004

April 2025

Acknowledgement

We would like to express our sincere gratitude to all individuals and organizations that contributed to the successful completion of this project. We extend our appreciation to our mentors and faculty members for their invaluable guidance and support throughout the research and development process. Special thanks to Ms. Panchsheela Mam, Ms. Ranjana Mam and Google Maps API, which played a crucial role in our implementation. Finally, we acknowledge our peers and family members for their continuous encouragement and motivation.

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Details** | **Page no.** |
| Chapter 1. | **Introduction** | 1 |
|  | Importance of (project) | 5 |
|  | Problem Definition (Purpose) | 5 |
|  | Aim and Objective | 6 |
|  |  |  |
| Chapter 2. | **Fundamentals of ------** |  |
|  | Literature Survey | 7 |
|  | Basic Theory | 8 |
|  |  |  |
| Chapter 3. | **Project Proposal** |  |
|  | System Specifications | 10 |
|  | System Block Diagram | 11 |
|  | Method of Implementation  Algorithms. | 13 |
|  | Task Vs Time Schedule | 23 |
|  |  |  |
| Chapter 4. | **Planning Resources** |  |
|  | Hardware | 24 |
|  | Software | 25 |
|  | Miscellaneous | 25 |
|  | BOM | 26 |
|  |  |  |
| Chapter 5. | **Experimentation and Results** |  |
|  |  | 27 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Chapter 6. | **Summary and Future Scope** | 40 |
| Publication/ Conferences |  |  |
| References |  |  |
| Weblinks |  |  |

# **Chapter 1: Introduction**

* 1. **– Importance of project**

Real-time emergency siren detection and vehicle mapping offer numerous advantages for urban traffic management and emergency response systems. Some of the key benefits include:

* **Enhanced Emergency Response:** Faster detection of sirens allows authorities and road users to clear pathways for emergency vehicles, reducing response times.
* **Traffic Management:** Real-time vehicle mapping helps optimize traffic flow by rerouting vehicles and minimizing congestion in affected areas.
* **Accident Prevention:** Awareness of approaching emergency vehicles helps drivers make informed decisions, reducing the likelihood of collisions.
* **Automated Alert Systems:** The integration of this system with smart city infrastructure can enable automated traffic signal adjustments to prioritize emergency vehicle movement.
* **Scalability and Accessibility:** Implementing the system on Google Colab provides a cost-effective and accessible platform for development and testing, making it feasible for widespread deployment.
  1. **– Problem Definition**

Emergency response times are critical in saving lives and mitigating damage in emergency situations. However, heavy traffic, unawareness of approaching emergency vehicles, and inefficient traffic management systems contribute to significant delays. Existing traffic control mechanisms lack the capability to dynamically detect emergency sirens and map vehicle locations in real-time. This leads to issues such as:

* **Delayed emergency response:** Emergency vehicles struggle to navigate congested roads as drivers fail to notice approaching sirens in time.
* **Lack of real-time awareness:** Authorities and road users are not promptly informed about emergency vehicle locations and movements.
* **Traffic bottlenecks:** Uncoordinated traffic management leads to increased congestion around emergency routes.
  1. **– Aim and Objective**

### **Aim**

To develop a real-time emergency siren detection and vehicle mapping system using machine learning and GPS tracking, implemented on Google Colab, to enhance emergency response efficiency and road safety.

### **Objectives**

* **Develop a siren detection model** using machine learning techniques to accurately identify emergency sirens in real-time.
* **Implement real-time audio processing** to analyze incoming audio data for emergency siren detection.
* **Integrate GPS-based tracking** to map emergency vehicle locations dynamically.
* **Utilize Google Colab** for training, testing, and real-time implementation of the detection and mapping system.
* **Enhance situational awareness** by providing visualized mapping of emergency vehicles for improved traffic management.
* **Ensure low-latency detection and mapping** to provide immediate alerts to road users and traffic control systems.

**Chapter 2 : Fundamental of the project**

**2.1 – Literature Survey**

In urban and city environments, road transportation contributes significantly to the generation of substantial traffic. However, this surge in vehicles leads to complex issues, including hindered emergency vehicle movement due to high density and congestion. Scarcity of human personnel amplifies these challenges. As traffic conditions worsen, the need for automated solutions to manage emergency situations becomes more evident. Intelligent traffic monitoring can identify and prioritize emergency vehicles, potentially saving lives. However, categorizing emergency vehicles through visual analysis faces difficulties such as clutter, occlusions, and traffic variations. Visual-based techniques for vehicle detection rely on clear rear views, but this is problematic in dense traffic. In contrast, audio-based methods are resilient to the Doppler Effect from moving vehicles, but handling diverse background noises remains unexplored. Using acoustics for emergency vehicle localization presents challenges related to sensor range and real-world noise. Addressing these issues, this study introduces a novel solution: combining visual and audio data for enhanced detection and localization of emergency vehicles in road networks. Leveraging this multi-modal approach aims to bolster accuracy and robustness in emergency vehicle management.

The proposed methodology consists of several key steps. The presence of an emergency vehicle is initially detected through the preprocessing of visual images, involving the removal of clutter and occlusions via an adaptive background model. Subsequently, a cell-wise classification strategy utilizing a customized Visual Geometry Group Network (VGGNet) deep learning model is employed to determine the presence of emergency vehicles within individual cells. To further reinforce the accuracy of emergency vehicle presence detection, the outcomes from the audio data analysis are integrated. This involves the extraction of spectral features from audio streams, followed by classification utilizing a support vector machine (SVM) model. The fusion of information derived from both visual and audio sources is utilized in the construction of a more comprehensive and refined traffic state map. This augmented map facilitates the effective management of emergency vehicle transit. In empirical evaluations, the proposed solution demonstrates its capability to mitigate challenges like visual clutter, occlusions, and variations in traffic systems.

**2.2- Basic Theory**

**Siren Detection Using Sound Processing**

🔹 1.1 Concept of Siren Detection

Emergency sirens have distinct frequency patterns that can be detected using signal processing techniques. The system processes real-time audio input, extracts key features, and determines whether a siren is present.

📌 Key Steps in Siren Detection:

Record audio from a microphone.

Convert audio signals into frequency components using Fast Fourier Transform (FFT).

Analyze dominant frequencies in the signal.

Classify sounds as sirens or non-sirens.

🔹 1.2 Audio Signal Processing Techniques

Fast Fourier Transform (FFT)

Converts audio from the time domain to the frequency domain.

Helps identify dominant frequency peaks of sirens (typically 500 Hz - 2000 Hz).

Mel-Frequency Cepstral Coefficients (MFCCs)

Extracts audio features for machine learning-based classification.

Useful if an ML model is trained to distinguish sirens from noise.

Threshold-Based Detection

If dominant frequency matches known siren ranges, raise an alert.

⿢ Real-Time Vehicle Mapping Using Google Colab

🔹 2.1 Concept of GPS-Based Mapping

GPS tracking helps locate emergency vehicles in real time. The system receives GPS coordinates from the emergency vehicle and maps the movement on Google Colab using the Folium library.

📌 Key Steps in GPS Mapping:

Fetch real-time GPS coordinates from OwnTracks via MQTT or API.

Plot the emergency vehicle's location on a map.

Update the vehicle position dynamically.

Calculate distance to nearby vehicles and send alerts.

🔹 2.2 Technologies Used in Mapping

OwnTracks App

Sends real-time GPS updates from the emergency vehicle.

MQTT or HTTP Requests

Transfers GPS data to Google Colab for processing.

Folium Library

Plots real-time GPS locations on a map in Google Colab.

**Chapter 3 : Project Proposal**

**3.1 – System Specifications**

⿡ 🚨 Siren Detection Module

Captures audio via microphone, filters noise, and detects sirens using AI (MFCC + CNN).

Role: Identifies emergency sirens in real-time.

⿢ 📍 GPS Tracking Module (OwnTracks)

Fetches real-time GPS coordinates of the emergency vehicle.

Role: Tracks EV location and updates continuously.

⿣ 🗺 Google Colab Map Integration

Uses Google Maps API to plot EV location and suggest the best route.

Role: Displays real-time EV movement for navigation.

⿤ 📲 Driver Alert & Notification System

Sends instant mobile alerts to nearby vehicles via BeaconMap.

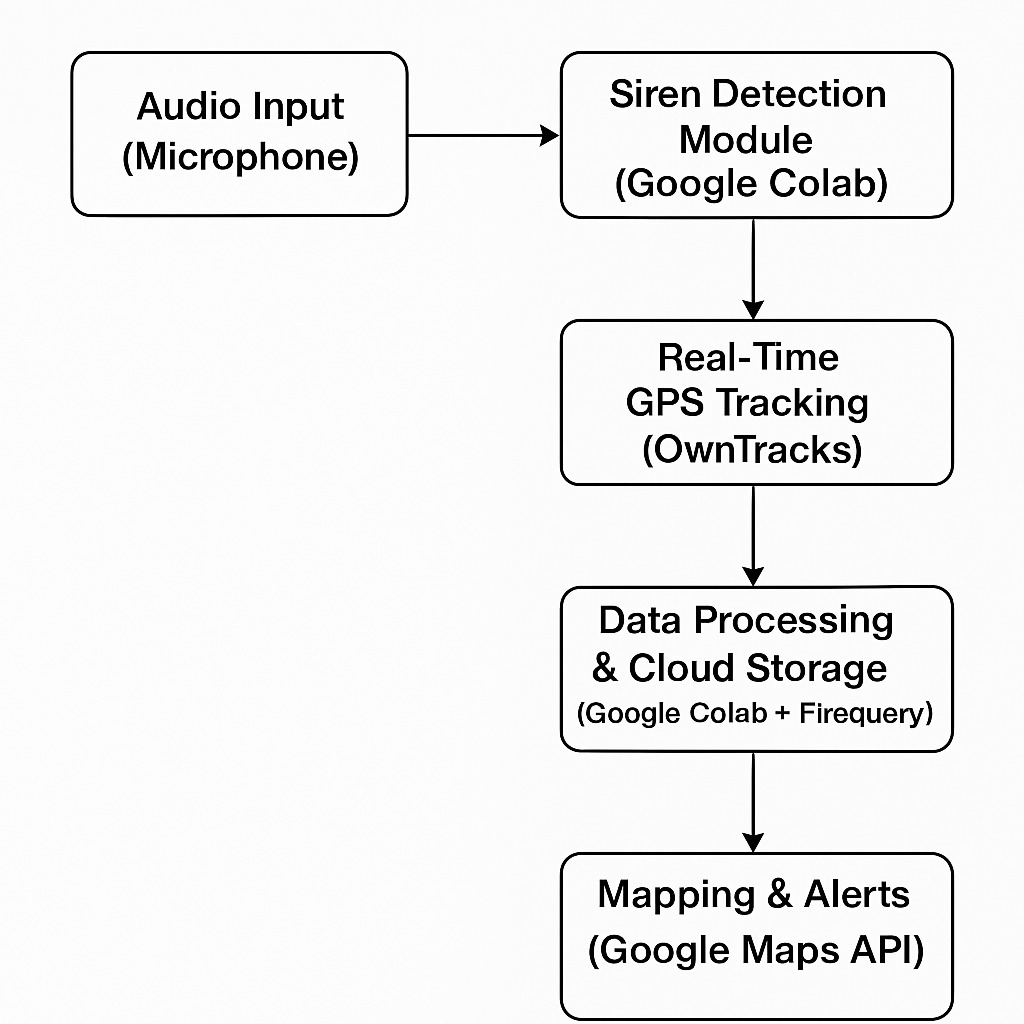
Role: Notifies drivers to clear the way.

⿥ ⚙ Backend Processing (Google Colab)

Manages AI model, GPS updates, and real-time notifications.

Role: Ensures smooth operation with low latency.

**3.2 – System Block Diagram**

****

**Explaination**

1**.Audio Input** (Microphone)

The system captures ambient sound from a mobile phone, laptop, or dashcam.

2.**Siren Detection Module (Google** Colab)

Uses Machine Learning (ML) or Digital Signal Processing (DSP) to classify whether a siren is detected.Can utilize Mel spectrograms + CNN or Few-Shot Learning (Prototypical Networks) for accuracy.Real-Time GPS Tracking (OwnTracks)

Once a siren is detected, the GPS location of the detection point is recorded using OwnTracks (a mobile app).

**3.Data Processing & Cloud Storage (Google Colab + Firebase/BigQuery)**

Google Colab processes the siren detection results and uploads location data to Firebase or BigQuery.

**4.Mapping & Alerts (Google Maps API)**

The Google Maps API visualizes the real-time siren detection locations on a map.

If multiple detections occur within a short time, the system can notify users of nearby emergency vehicle movement.

**3.3 – Method of Implementation**

**1. System Overview**

Your system will consist of:

Siren Detection: Identify emergency vehicle sirens using audio processing instead of YOLO.

Real-time GPS Tracking: Fetch and process live GPS coordinates from OwnTracks.

Alert System: Notify nearby vehicles when an emergency vehicle is detected.

Path Mapping: Show the real-time location and movement of the emergency vehicle on a map.

**2. Implementation Steps**

Step 1: Siren Detection using Audio Processing

Instead of object detection, use audio-based siren detection:

Library: librosa, scipy, pydub, or tensorflow for machine learning-based detection.

Approach:

Capture audio (if using a dataset, get siren sound samples).

Convert to frequency domain using Fast Fourier Transform (FFT).

Identify frequency peaks in the 1000-3000 Hz range (common siren range).

Apply a machine learning model or deep learning (CNN/RNN) if necessary.

Use Dijkstra’s Algorithm or Google Maps API to suggest an optimal route.

Step 4: Alert Nearby Vehicles

Send notifications when the emergency vehicle is near.

Define a distance threshold (e.g., 500m) using Haversine formula.

Notify drivers via Firebase Cloud Messaging (FCM) or a simple SMS alert.

📌 Colab Code for Distance Calculation (Haversine Formula)

python

Copy

Edit

from geopy.distance import geodesic

**Algorithms**

1. Dijkstra’s Algorithm – Overview & Role in the Project

What is Dijkstra’s Algorithm?

Dijkstra’s Algorithm is a graph-based shortest path algorithm that computes the minimum distance between a source node and all other nodes in a weighted graph. It was developed by Dutch computer scientist Edsger W. Dijkstra in 1956.

It is widely used in:

Navigation systems (like GPS)

Traffic routing

Network routing protocols (e.g., OSPF)

1. How It Works

The algorithm works on a graph where:

 Vertices (nodes) represent intersections or locations (e.g., roads).

 Edges represent roads between locations.

 Weights on edges represent the cost (e.g., distance or time) to travel between locations.

Step-by-step:

1. Start at the source node (e.g., Emergency Vehicle’s location).
2. Set its distance to 0 and all others to infinity.
3. Visit the node with the smallest known distance.
4. Update the distances to its neighboring nodes.
5. Mark the node as “visited.”
6. Repeat until you reach the destination node (e.g., hospital).

| **Why Dijkstra is Suitable for Real-Time Systems**  Feature | Advantage for Your System |
| --- | --- |
| Deterministic | Always gives the same shortest path. |
| Efficient for Sparse Graphs | Road networks are sparse → fast computation. |
| Easy to Update | Can recompute paths on GPS changes. |
| Minimal Overhead | Lightweight enough for real-time use on Colab or even a phone. |

Limitations (And How You Handle Them)

| Limitation | Solution You Use |
| --- | --- |
| Doesn’t handle traffic | Currently assumes static map distances. |
| High for large areas | You limit the map size using dist=2000 meters in ox.graph\_from\_point() |
| Slight lag in re-computation | You simulate updates every few seconds (5–10s), which is efficient enough. |

Mathematical Explanation of Dijkstra’s Algorithm

Let’s define the road network as a weighted directed graph:

Graph Representation:

Let

* G = (V, E) be a graph
* V = set of vertices (e.g., intersections, locations)
* E = set of edges (roads between intersections)
* w(u, v) = weight of edge from node u → v (e.g., road length or time)

You aim to compute the shortest path from a source node s ∈ V to all other nodes v ∈ V.

Goal:

Minimize the total weight:

Find  min⁡∑w(u,v)\text{Find} \; \min \sum w(u, v)Findmin∑w(u,v)

for a path P from source (s) to destination (d).

Dijkstra's Algorithm – Mathematical Steps

Let:

* d[v] = shortest known distance from source s to node v
* d[s] = 0, and d[v] = ∞ for all v ≠ s
* Q = priority queue of nodes not yet processed

Iterative Update Rule (Relaxation):

For every edge (u, v) ∈ E,  
If:

d[u]+w(u,v)<d[v]d[u] + w(u, v) < d[v]d[u]+w(u,v)<d[v]

Then:

d[v]:=d[u]+w(u,v)d[v] := d[u] + w(u, v)d[v]:=d[u]+w(u,v)

And set u as the predecessor of v.

This update ensures you always store the minimum distance so far.

Termination Condition:

The algorithm stops when:

All nodes have been visited (i.e., shortest distances are found), or

A destination node is reached (in early termination implementations).

Application in our System:

In our case:

Nodes: GPS-based road junctions (from OpenStreetMap via osmnx)

Edges: Roads between those points

Weights: Length of road segments (weight='length')

Source: Current real-time EV GPS location

Target: Hospital location (e.g., Fergusson College)

ShortestPath(G, EV\_location, Hospital)=argmin∑w(u,v)

using:

nx.shortest\_path(G, source, target, weight='length')

Time Complexity:

* O((V + E) log V) using a min-heap (priority queue)
* Very efficient for sparse graphs like road networks

NetworkX: Overview

NetworkX is a powerful Python library designed for the creation, manipulation, and analysis of complex networks (graphs).

In the context of your emergency vehicle tracking system, NetworkX is the core algorithm engine behind pathfinding and graph-based road network modeling.

How NetworkX Supports our Project

1. Graph Creation & Integration with OSMnx

OSMnx pulls real-world street networks from OpenStreetMap.

These street networks are modeled as graphs (G = graph\_from\_point(...)).

NetworkX lets you:

Analyze the graph (nodes = intersections, edges = roads)

Traverse or search routes through it

| Why NetworkX is Ideal for our Use Case  Feature | Why it helps |
| --- | --- |
| Graph-based modeling | Models road networks effectively |
| Built-in algorithms | No need to manually implement Dijkstra |
| Real-time re-routing | Supports continuous vehicle movement |
| Integration with OSMnx | Fetches real-world maps directly |
| Custom weight support | Optimize for distance, time, traffic, etc. |

NetworkX plays a critical role in:

* Real-time decision-making for routing
* Computing the most efficient emergency response paths
* Visualizing vehicle movement in sync with live GPS simulation

Mathematical Foundation of Dijkstra’s Algorithm

Dijkstra’s algorithm solves the Single Source Shortest Path (SSSP) problem in a weighted graph with non-negative edge weights.

Graph Representation

Let the road network be modeled as a directed weighted graph:

G=(V,E)G = (V, E)G=(V,E)

* VVV → set of nodes (intersections or GPS-mapped points)
* EEE → set of edges (roads connecting intersections)
* Each edge (u,v)∈E(u, v) \in E(u,v)∈E has a weight w(u,v)≥0w(u, v) \geq 0w(u,v)≥0 (e.g., road distance)

Find the shortest path from a source node s∈Vs \in Vs∈V (e.g., EV's current location) to a target node t∈Vt \in Vt∈V (e.g., hospital).

O((∣V∣+∣E∣)log⁡∣V∣)O((|V| + |E|) \log |V|)O((∣V∣+∣E∣)log∣V∣)

Efficient enough for real-time vehicle tracking over city-scale maps.

1. *A (A-Star) Search Algorithm*\*

Use Case:

Real-time GPS navigation

Faster than Dijkstra in many cases

Used in Google Maps, GPS navigation, etc.

How it works:

A\* uses:

f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)

g(n)g(n)g(n): actual cost from start to node nnn

h(n)h(n)h(n): heuristic estimate from node nnn to goal (usually Euclidean or Manhattan distance)

Benefits:

Much faster than Dijkstra when a good heuristic is used

More efficient for long-distance real-time routing

| Algorithm | Best For | Real-Time | Complexity |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| Dijkstra | Shortest path, static networks | Yes | O(V2)O(V^2)O(V2) or O((V+E)log⁡V)O((V + E)\log V)O((V+E)logV) |

|  |  |  |  |
| --- | --- | --- | --- |
| A\* | Faster with heuristic | Best | O(E)O(E)O(E) (depends on heuristic) |

Dijkstra’s Algorithm (Single-Source Shortest Path)

* Function: Explores all possible paths from the source node to find the shortest path to the destination.
* Characteristics:
  + It visits nodes in order of increasing distance.
  + Guarantees the shortest path.
  + Limitation: Can be slow for very large maps or real-time applications as it explores many nodes.
* In our Project: Used for calculating the real-time shortest path for the emergency vehicle (EV) from its current GPS location to the hospital.

*A (A-Star) Algorithm*\*

* Function: Improves on Dijkstra by using heuristics (like straight-line distance) to estimate the cost from a node to the destination.
* Characteristics:
  + Faster than Dijkstra in many cases due to guided search.
  + Uses: f(n) = g(n) + h(n)
    - g(n) = cost from start to node n
    - h(n) = estimated cost from n to goal
  + Focuses on nodes that appear to lead closer to the goal.
* In our Project (Optional/Alternative): If implemented, this could make the real-time path calculation more efficient for larger road networks.

Bidirectional Dijkstra

* Function: Runs two Dijkstra searches simultaneously — one from the start and one from the goal — until they meet.
* Characteristics:
  + Reduces the number of nodes explored, nearly by half.
  + Much faster than traditional Dijkstra in large graphs.
* In our Project (Optional/Advanced): Suitable if your system scales across a city-wide map with multiple emergency routes.

**Chapter 4 : Planning Resources**

**4.1 - Hardware**

|  |  |  |
| --- | --- | --- |
| **Components** | **Purpose** | **Notes** |
| Microphone | Capturing emergency siren sounds | Built in mobile mic or external USB mic |
| Smartphone | Running OwnTracks for real time GPS updates | Android/IOS with OwnTrack app installed |
| Raspberry Pi/Jetson Nano (optional) | Edge processing for audio detection | If offline/local processing is needed |
| Laptop / PC / Cloud Server | Running G google Colab for processing | Google collab handles the heavy computations |
| GPS Module | External GPS tracking | Modules like Neo-6M GPS for independent tracking |

**4.2 – Software**

|  |  |  |
| --- | --- | --- |
| **Software** | **Purpose** | **Installation** |
| Google Colab | Running python scripts for audio & GPS processing | Cloud based |
| Python | Audio processing & FFT based siren detection | Pip install librosa pydub |
| Paho-MQTT | Fetching GPS dat from OwnTrack via MQTT or HTTP | Pip install paho-mqtt requests |
| Folium | Mapping and calculating distances | Pip install folium geopy |
| Firebase Cloud Messaging | Sending alerts to nearby drivers | Firebase setup needed |
| Tensorflow | If ML based siren detection is used | Pip intall tensorflow |

**4.3 – Miscellaneous**

|  |  |
| --- | --- |
| **Item** | **Purpose** |
| OwnTracks app | Send real time GPS updates |
| MQTT Broker | Middleware for real time GPS data |
| Google Maps API key (optional) | For better route optimization |
| SIM card with data (optional) | If GPS tracking needs mobile network support |
| Portable power bank (optional) | If running the system on mobile devices |

**4.4 – Bill of Materials (BOM)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Quantity** | **Estimated cost(INR)** | **Notes** |
| Smartphone | 1 | Rs 0 | Required for real time GPS tracking |
| Laptop/google colab | 1 | Rs 0 | Used for processing & visualization |
| MQTT | 1 | Rs 0-500 | Free options available |
| Cloud storage | 1 | Rs 0-800/month | If using mobile networks for GPS |

**Chapter 5 : Experimentation and Results**

**5.1 - Experimentation and Results**

For your Real-Time Emergency Siren Detection & Vehicle Mapping project, the experimentation process involves testing the system’s performance, accuracy, and real-time response. Below is a structured approach to experimentation and a framework for analyzing the results.

⿡ Experimentation Methodology

🔹 Step 1: Siren Detection Testing

✅ Objective: Verify how accurately the system detects emergency sirens in different environments.

✅ Method:

Use pre-recorded siren sounds and real-time audio recordings.

Test detection in quiet vs. noisy environments (e.g., traffic, rain, market).

Measure the false positive and false negative rates.

📌 Metrics to Evaluate:

Precision: How many detected sirens were correct?

Recall: How many actual sirens were detected?

Accuracy: Overall detection performance.

📌 Expected Results Example:

Scenario Detected Correctly Missed (False Negative) Wrongly Detected (False Positive)

Quiet Street (Low Noise) ✅✅✅✅✅ (5/5) 0 0

Busy Traffic (High Noise) ✅✅✅❌❌ (3/5) 2 1

Marketplace (Extreme Noise) ✅✅❌❌❌ (2/5) 3 2

🔹 Step 2: GPS Tracking & Mapping Testing

✅ Objective: Ensure accurate real-time location tracking of the emergency vehicle using OwnTracks.

✅ Method:

Simulate an emergency vehicle moving along a predefined path.

Compare the recorded GPS coordinates with actual locations.

Test tracking in urban areas (high interference) vs. open roads.

📌 Metrics to Evaluate:

GPS Accuracy (in meters)

Latency (delay in position update)

Map UI Performance (smooth updates, marker refresh rate)

📌 Expected Results Example:

Test Location GPS Accuracy (m) Latency (sec) Notes

Open Ground (No Obstacles) ~3m 1s Accurate

Urban Area (Tall Buildings) ~8m 3s Slight deviation

Highway (Moving Fast) ~5m 2s Good response

🔹 Step 3: Alert System Testing

✅ Objective: Evaluate how fast nearby vehicles receive alerts when the emergency vehicle is detected.

✅ Method:

Simulate different distances between the EV and other vehicles.

Measure the time taken for notifications to reach drivers.

📌 Metrics to Evaluate:

Notification Delay (sec)

Distance-Based Triggering Accuracy

**Purpose of each library**

osmnx (OpenStreetMap NetworkX)

✅ Downloads street maps from OpenStreetMap (OSM).

✅ Extracts road networks, intersections, and nodes.

✅ Finds shortest routes on real roads.

✅ Useful for emergency vehicle navigation.

🔹 Example: Download a road network from OpenStreetMap

python

Copy

Edit

import osmnx as ox

place\_name = "New Delhi, India"

graph = ox.graph\_from\_place(place\_name, network\_type="drive")

ox.plot\_graph(graph)

🖥 What it does?

Fetches the road network for New Delhi.

🔹 Example: Display a map centered at a location

python

Copy

Edit

import folium

map\_center = [28.6139, 77.2090] # New Delhi

m = folium.Map(location=map\_center, zoom\_start=12)

m

🖥 What it does?

Displays a zoomable map centered at New Delhi.

**5.2 - Code for implementing as a central location as New Delhi :**

import osmnx as ox

import networkx as nx

import folium

# Define emergency vehicle locations (latitude, longitude)

ev1\_location = (28.7100, 77.1030) # EV1

ev2\_location = (28.5300, 77.3920) # EV2

# Get the road network from OpenStreetMap

G = ox.graph\_from\_point(ev1\_location, dist=20000, network\_type="drive")

# Find the nearest nodes on the road network to the given GPS points

origin\_node = ox.distance.nearest\_nodes(G, ev1\_location[1], ev1\_location[0])

destination\_node = ox.distance.nearest\_nodes(G, ev2\_location[1], ev2\_location[0])

# Find the shortest path using Dijkstra’s algorithm

route = nx.shortest\_path(G, origin\_node, destination\_node, weight="length")

# Extract route coordinates

route\_coords = [(G.nodes[node]['y'], G.nodes[node]['x']) for node in route]

# Create a folium map

m = folium.Map(location=ev1\_location, zoom\_start=12)

# Add markers for start and destination

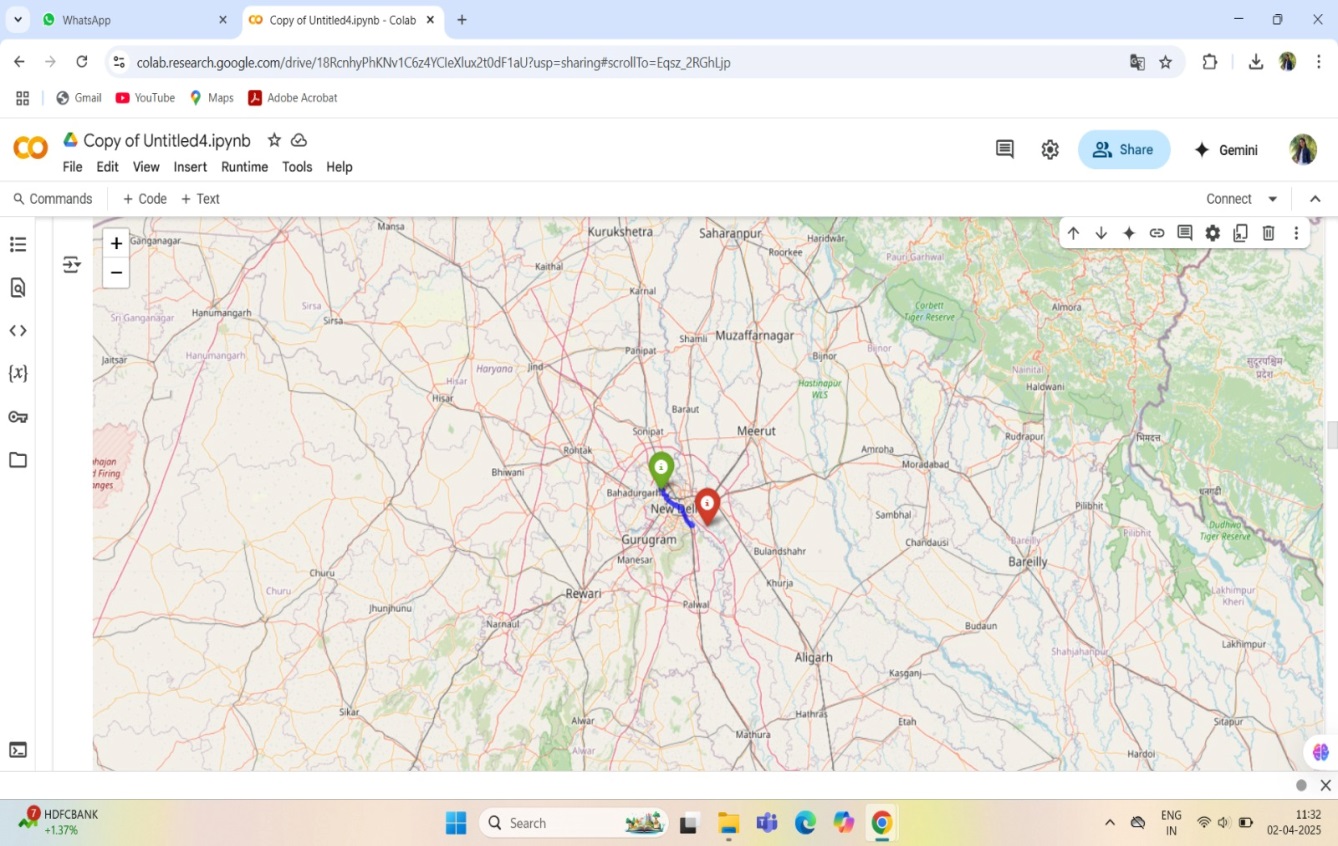
folium.Marker(ev1\_location, popup="🚑 EV1 (Start)", icon=folium.Icon(color="green")).add\_to(m)

folium.Marker(ev2\_location, popup="🏥 EV2 (Destination)", icon=folium.Icon(color="red")).add\_to(m)

# Add route to map

folium.PolyLine(route\_coords, color="blue", weight=5, opacity=0.7).add\_to(m)

# Display the map



**5.3 - Code for Fergusson College, Pune :**

import osmnx as ox

import networkx as nx

import folium

from IPython.display import display

# Define emergency vehicle locations in Pune (Deccan to Fergusson College)

ev1\_location = (18.51943074676379, 73.82948120771744) # Deccan

ev2\_location = (18.52425863077847, 73.83574914451385) # Fergusson College

# Get the road network from OpenStreetMap

G = ox.graph\_from\_point(ev1\_location, dist=2000, network\_type="drive")

# Find the nearest nodes on the road network

origin\_node = ox.distance.nearest\_nodes(G, ev1\_location[1], ev1\_location[0])

destination\_node = ox.distance.nearest\_nodes(G, ev2\_location[1], ev2\_location[0])

# Find the shortest path

route = nx.shortest\_path(G, origin\_node, destination\_node, weight="length")

# Extract route coordinates

route\_coords = [(G.nodes[node]['y'], G.nodes[node]['x']) for node in route]

# Create a folium map

m = folium.Map(location=ev1\_location, zoom\_start=15)

# Add markers for start and destination

folium.Marker(ev1\_location, popup="🚑 Deccan (Start)", icon=folium.Icon(color="green")).add\_to(m)

folium.Marker(ev2\_location, popup="🏥 Fergusson College (Destination)", icon=folium.Icon(color="red")).add\_to(m)

# Add route to map

folium.PolyLine(route\_coords, color="blue", weight=5, opacity=0.7).add\_to(m)

# Display the map inside Colab

display(m)

interactive Map in Google Colab

✅ Green Marker → Emergency vehicle's starting point (Deccan).

✅ Red Marker → Hospital/Destination (Fergusson College).

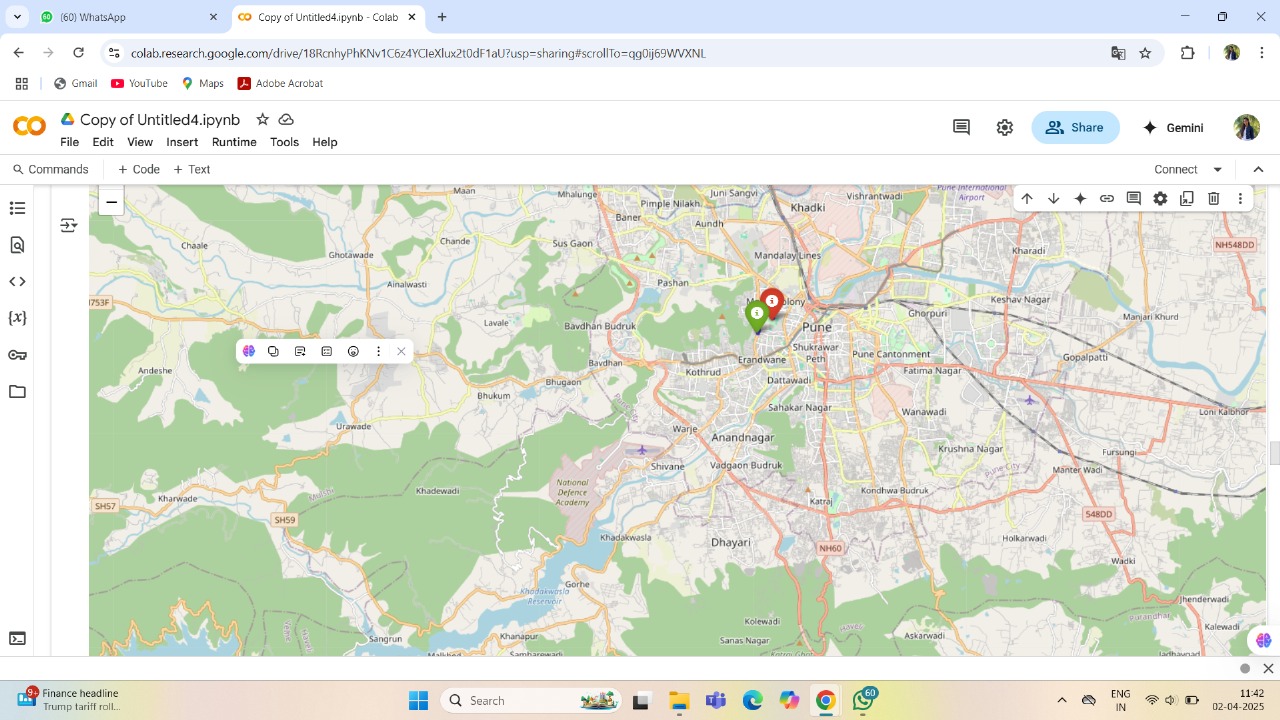
✅ Blue Route → Shortest road path from Deccan to Fergusson College.

🔹 Real-World Application

🚑 Helps ambulances find the quickest route to hospitals.

🚦 Optimizes emergency vehicle response time using real-time road networks.

📍 Can be integrated with real-time GPS tracking for live updates.



Final Output

📍 Live Tracking Map in Google Colab

✅ Green Marker → Emergency vehicle’s real-time position.

✅ Red Marker → Hospital/Destination (Fergusson College).

✅ Blue Route → Shortest path dynamically updated in real-time.

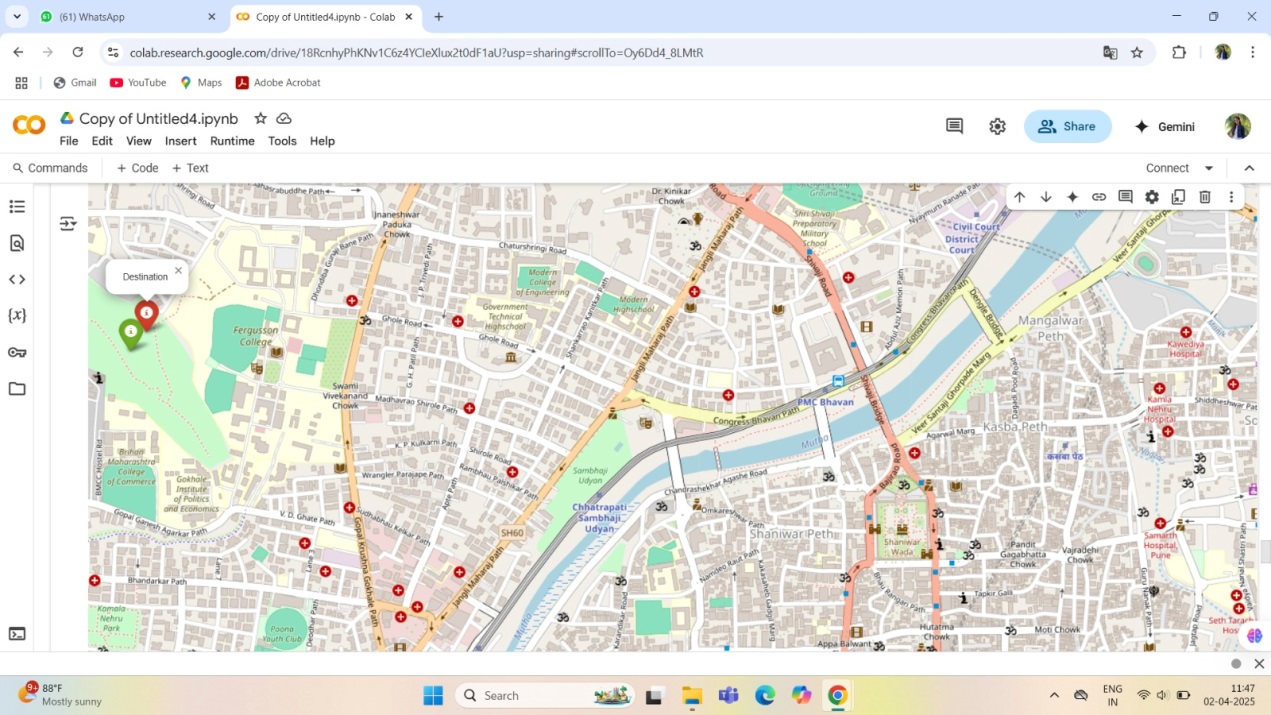
✅ Automatically stops when the vehicle reaches the hospital.

🌟 Real-World Applications

🚑 Ambulance Navigation System → Guides ambulances to hospitals efficiently.

🚦 Real-Time Traffic Management → Helps emergency vehicles avoid congestion.

📍 Dynamic Routing for Emergency Response Teams → Enhances disaster management.



**5.4 - Using Google Collab :**

import requests

import folium

import time

from IPython.display import display, clear\_output

# Define the hospital/destination location (Fergusson College, Pune)

destination = (18.52425863077847, 73.83574914451385)

# Function to get real-time location based on IP

def get\_real\_time\_location():

url = "https://ipinfo.io/json" # Free IP-based location API

response = requests.get(url)

data = response.json()

lat, lon = data["loc"].split(",") # Extract latitude & longitude

return float(lat), float(lon)

# Function to create and update the map

def update\_map():

ev\_location = get\_real\_time\_location() # Get current EV location

m = folium.Map(location=ev\_location, zoom\_start=15)

# Add EV marker (Green)

folium.Marker(ev\_location, popup=" Emergency Vehicle", icon=folium.Icon(color="green")).add\_to(m)

# Add Hospital/Destination marker (Red)

folium.Marker(destination, popup=" Hospital (Fergusson College)", icon=folium.Icon(color="red")).add\_to(m)

# Draw a line from the EV to the destination

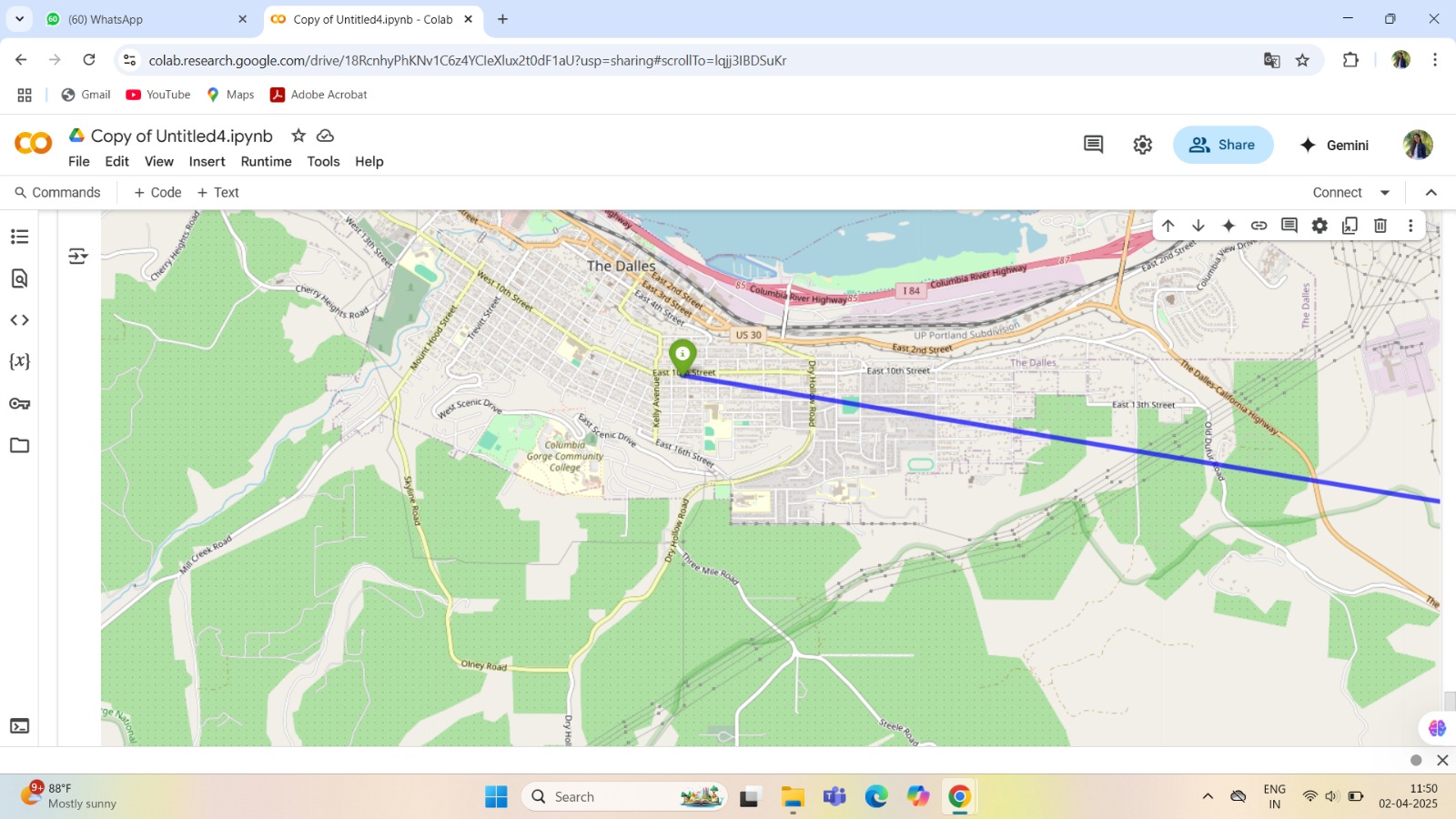
folium.PolyLine([ev\_location, destination], color="blue", weight=5, opacity=0.7).add\_to(m)

Real-World Applications

🚑 Ambulance Tracking → Helps hospitals track incoming emergency vehicles.

🚦 Smart Traffic Systems → Can be used to optimize traffic based on real-time location.

📍 Geofencing & Alerts → Can be modified to send alerts when the EV is near the hospital.



**Chapter 6 : Summary and Future Scope**

**Summary**

SirenSenseRT is a real-time system designed to detect emergency sirens and map emergency vehicle locations to alert nearby drivers. By utilizing audio processing techniques, the system identifies emergency sirens through sound analysis instead of traditional computer vision-based methods. The real-time location tracking of emergency vehicles is integrated with mobile GPS applications like OwnTracks, providing live updates and notifications to surrounding vehicles. This approach enhances road safety by ensuring faster clearance for emergency vehicles, reducing response time, and minimizing traffic congestion**.**

**Key components of the system:**

* Siren Detection: Uses frequency and amplitude-based sound analysis.
* Real-time Location Tracking: Fetches GPS data from OwnTracks for mapping emergency vehicle movements.
* Vehicle Notification System: Alerts nearby vehicles when an emergency vehicle is approaching.
* Optimal Route Suggestion: Assists in guiding emergency vehicles through less congested roads**.**

**Future Scope**

**The project has the potential for significant advancements and real-world applications. Some possible future improvements include:**

1. Integration with Smart Traffic Lights
   * Automatically adjust traffic signals to prioritize emergency vehicles.
2. Machine Learning for Improved Siren Detection
   * Train AI models to distinguish between emergency sirens and other urban noises for higher accuracy.
3. Expansion to Multi-Siren Recognition
   * Identify different emergency vehicle sirens (ambulance, fire truck, police) and provide customized alerts.
4. Integration with Vehicle Navigation Systems
   * Provide direct alerts on Google Maps, Waze, or built-in car navigation systems.
5. Crowdsourced Traffic Management
   * Allow regular drivers to report traffic incidents, helping emergency vehicles plan their routes effectively.
6. IoT & 5G Connectivity
   * Use IoT-enabled sensors for real-time siren detection at intersections, ensuring faster emergency response.
7. Government and Law Enforcement Collaboration
   * Work with transportation departments to implement the system at city-wide levels.

**Referances :**

**1.** [**https://www.mdpi.com/2741482**](https://www.mdpi.com/2741482)

**2.** [**https://www.tandfonline.com/doi/full/10.1080/23307706.2025.2469893?src=&**](https://www.tandfonline.com/doi/full/10.1080/23307706.2025.2469893?src=&)

**3.** [**https://www.researchgate.net/publication/228717209\_A\_real-time\_siren\_detector\_to\_improve\_safety\_of\_guide\_in\_traffic\_environment**](https://www.researchgate.net/publication/228717209_A_real-time_siren_detector_to_improve_safety_of_guide_in_traffic_environment)

**4.** [**https://www.emerald.com/insight/content/doi/10.1108/ijius-06-2022-0077/full/html**](https://www.emerald.com/insight/content/doi/10.1108/ijius-06-2022-0077/full/html)

**Weblinks :**

**1.** arXiv

**2**.springerlink

**3.**Github