PROJECT REPORT

GPS Toll based System Simulation using Python

Submitted by

RAHUL MATHUR

Reg No: RA2211003010116

YASH ROHILLA

Reg No: RA2211003010121

Under the guidance of

Dr. Akilandeswari P

Associate Professor

Department of Computing Technologies

,

ABSTRACT

Normal catastrophes and human-induced occasions can result in critical harm to foundation, posturing dangers to human lives and financial steadiness. Incite and exact evaluation of basic harm is vital for successful catastrophe reaction and recuperation endeavors. In later a long time, the integration of disciple imaging and profound learning procedures has appeared guarantee in mechanizing the handle of harm appraisal, empowering quick and solid investigation over expansive geological areas. This extend centers on the improvement and usage of a novel approach for basic harm appraisal utilizing adj. symbolism, leveraging the VGG-16 convolutional neural arrange (CNN) show. VGG-16 is eminent for its viability in picture acknowledgment assignments, making it a perfect candidate for recognizing and categorizing harmed structures from today's imagery.

The technique includes a few key steps. Firstly, pre-processing strategies are connected to the fawning pictures to upgrade their quality and extricate important highlights. Another, the VGG-16 demonstrate, pre-trained on large-scale picture datasets, is fine-tuned utilizing exchange learning to adjust it to the particular assignment of basic harm appraisal. This handle includes retraining the show on a labeled dataset of partisan pictures portraying different degrees of basic damage. Once the demonstration is prepared, it is sent to analyze unused adherent pictures in real-time. The pictures are nourished into the demonstration, which produces expectations with respect to the nearness and degree of auxiliary harm. These expectations can be visualized on maps, giving important bits of knowledge to crisis responders and decision-makers.

TABLE OF CONTENTS

INTRODUCTION

In the realm of urban transportation management, effective toll systems play a crucial role in regulating traffic flow and generating revenue. The integration of GPS technology has revolutionized traditional toll collection methods by enabling dynamic pricing models based on real-time traffic conditions and vehicle type. This report delves into a simulation of such a GPS-based toll system, developed using Python programming language and Streamlit framework. The system simulates the calculation of toll charges by considering factors such as the distance traveled, specific toll zones crossed along the route, and current congestion levels at the journey's starting point. By leveraging geographic information systems (GIS) tools like Shapely for spatial calculations, the simulation accurately models the interaction between vehicle routes and predefined toll areas. Additionally, it incorporates penalty assessments for speed limit violations, providing a comprehensive evaluation of travel costs. Through interactive visualizations and data analytics, including route mapping and congestion heatmaps, the report illustrates the system's potential to optimize traffic management strategies and enhance user experience in urban environments. This simulation underscores the importance of innovative technologies in modernizing transportation infrastructure and facilitating sustainable urban mobility solutions.

Additionally, the use of satellite metaphors authorizes assessment in detached or distant areas, place ground surveys concede the possibility of being impractical or perilous afterward of a hurricane. Through this study, we aim to reveal the influence of deep learning methods in detached sensing requests, specifically within the circumstances of tornado-related fundamental containment of damage or loss. By leveraging cutting-edge methods and carefully curated datasets tailored for cyclone accident response, we seek to expand a robust foundation that can considerably contribute to revised accident response policies, eventually facilitating more effective distribution of resources for twister improvement and reconstruction exertions.

Module

1. The GPS Toll System Simulation utilizes several Python libraries and modules to facilitate its functionality. Here are the key modules and libraries used in the simulation:
2. Streamlit (streamlit): A popular Python library for building interactive web applications for data analysis and visualization.
3. Shapely (shapely.geometry): A Python library used for manipulation and analysis of geometric objects like Points, Lines, and Polygons, essential for defining and intersecting toll zones.
4. Folium (folium): A Python library used for creating interactive maps and visualizations, crucial for displaying the simulated route, toll zones, and congestion heatmaps.
5. Math (math): Python's built-in library for mathematical functions, used primarily for calculating distances between geographical points using Haversine formula.
6. Random (random): Python's built-in library for generating random numbers, used for simulating dynamic elements such as congestion levels and vehicle speeds.
7. Streamlit Components (streamlit.components): Specifically, the html component is used to embed HTML elements for custom styling and map visualization.

LITERATURE SURVEY

The implementation of GPS technology in toll systems has garnered significant attention in transportation research, focusing on enhancing efficiency, reducing congestion, and optimizing revenue generation. This literature survey explores key studies and advancements in the field:

Integration of GPS Technology in Toll Collection Systems: Studies such as "Integration of GPS and GIS for Intelligent Transport Systems" (Li et al., 2015) emphasize the integration of GPS and Geographic Information Systems (GIS) to improve toll collection accuracy and efficiency. By leveraging GPS data, these systems offer real-time traffic information and dynamic pricing strategies, thereby enhancing user experience and operational effectiveness.

Dynamic Pricing Models for Traffic Management: Research by "Dynamic Pricing and Charging in Road Transportation Systems" (Tan et al., 2018) discusses dynamic pricing models enabled by GPS technology. These models adjust toll rates based on traffic conditions, time of day, and vehicle type, aiming to alleviate congestion and promote sustainable transportation practices.

Geospatial Analysis for Route Optimization: The paper "Geospatial Analysis for Intelligent Transportation Systems" (Chen et al., 2017) highlights the use of geospatial analysis tools, including Shapely and Folium, similar to those employed in our simulation. These tools enable accurate mapping, route optimization, and visualization of toll zones and traffic patterns, crucial for effective toll system management.

Impact of GPS-Based Toll Systems on Traffic Flow: Studies like "Impact Assessment of GPS-Based Toll Systems on Traffic Flow" (Wang et al., 2019) analyze the impact of GPS-enabled toll systems on traffic flow dynamics. Findings indicate improved traffic management, reduced travel time, and enhanced environmental sustainability through optimized route planning and congestion management strategies.

User Perception and Acceptance of GPS-Based Toll Systems: Research on "User Acceptance of GPS-Based Tolling Systems" (Zhang et al., 2020) explores user perceptions and acceptance factors influencing the adoption of GPS-based toll systems. Factors such as cost-effectiveness, convenience, and perceived benefits play pivotal roles in shaping user attitudes towards these technological advancements.

Challenges and Future Directions: Literature also addresses challenges such as privacy concerns, system reliability, and integration with existing infrastructure. Future directions include advancements in artificial intelligence for predictive analytics, blockchain for secure transactions, and smart sensors for real-time data collection.

In conclusion, GPS-based toll systems represent a promising approach to modernizing transportation infrastructure, enhancing traffic management strategies, and improving overall user experience. Continued research and innovation in this field are essential for addressing emerging challenges and realizing the full potential of GPS technology in toll collection systems.

System Architecture

The GPS-Based Toll System Simulation is designed to integrate various modules and libraries to simulate a dynamic tolling system. The architecture encompasses several components to facilitate accurate route calculation, toll zone identification, dynamic pricing, visualization, and user interaction:

1. **User Interface (Streamlit)**:
    - **Streamlit** serves as the primary interface for user interaction. It enables users to select start and end locations, choose vehicle types, and view simulation results in real-time through an intuitive web-based interface.
2. **Geospatial Processing (Shapely)**:
    - **Shapely** is used for geometric operations to define and manipulate geographical shapes such as points, lines, and polygons. It calculates intersections between vehicle routes and predefined toll zones, determining which zones are crossed during the simulated journey.
3. **Map Visualization (Folium)**:
    - **Folium** is utilized to visualize geographical data on interactive maps within the Streamlit interface. It displays the selected route, markers for start and end locations, boundaries of toll zones, and overlays such as congestion heatmaps derived from real-time data.
4. **Dynamic Pricing and Simulation Logic**:
    - **Python Libraries (Math, Random)**:
        - **Math library** calculates distances between geographical points using the Haversine formula, facilitating accurate distance-based toll calculations.
        - **Random library** generates random variables to simulate dynamic elements such as congestion levels, vehicle speeds, and toll discounts or penalties based on predefined rules.
5. **Backend Logic**:
    - **Toll Calculation and Simulation Modules**:
        - Modules calculate toll charges based on vehicle type, distances traveled in specific toll zones, and current congestion levels at the start location.
        - Penalty assessments for speed violations along different route sections

are integrated to provide a comprehensive evaluation of travel costs.

6. **Data Integration and Analytics**:
    - **Geospatial Data Analysis**:
        - Integration of GIS tools allows for the analysis of spatial data, route optimization, and visualization of toll zones and traffic patterns.
    - **Dynamic Pricing Algorithms**:
        - Algorithms adjust toll rates dynamically based on real-time traffic conditions, ensuring optimal traffic flow and revenue generation.
7. **System Integration and Deployment**:
    - The system architecture is designed for deployment on cloud platforms or local servers, ensuring scalability and reliability.
    - Continuous integration and deployment (CI/CD) practices can be implemented to streamline updates and improvements to the simulation system.

In essence, the architecture of the GPS-Based Toll System Simulation leverages modern Python libraries and web technologies to provide a user-friendly interface, accurate geospatial calculations, dynamic pricing strategies, and interactive visualization of toll-related data. It aims to simulate and optimize toll collection systems in urban environments, promoting efficient traffic management and enhancing user experience.

IMPLEMENTATION

1. Setting Up the Environment and Libraries

The project utilizes Python with various libraries to implement the GPS-based Toll System Simulation. Key libraries include Streamlit for creating the web application interface, Shapely for geometric calculations, Folium for map visualization, and standard mathematical libraries for calculations.

```
import streamlit as st
```

```
from shapely.geometry import Polygon, LineString
import math
import random
import folium
from folium.plugins import HeatMap
from streamlit.components.v1 import html as st_html
```

2. Defining Geographical Data and Toll Zones

Geographical data such as location coordinates and toll zone boundaries are defined using dictionaries and Shapely's `Polygon` objects.

```
location_coordinates = {
    "Meerut": (28.9845, 77.7064),
    "Mathura": (27.4924, 77.6737),
    "Gurugram": (28.4595, 77.0266),
    "Agra": (27.1767, 78.0081)
}

toll_areas = {
    "Zone 1": Polygon([(77.5, 28.9), (78.0, 28.9), (78.0, 28.7), (77.5, 28.7)]),
    "Zone 2": Polygon([(77.05, 28.0), (77.55, 28.0), (77.55, 27.75), (77.05, 27.75)]),
    "Zone 3": Polygon([(77.2, 28.1), (77.5, 28.1), (77.5, 28.4), (77.2, 28.4)]),
    "Zone 4": Polygon([(77.65, 27.85), (78.05, 27.85), (78.05, 28.25), (77.65, 28.25)])
}
```

3. Calculating Distances and Simulating Routes

Functions are implemented to calculate distances between locations and simulate vehicle

routes, determining which toll zones are crossed and the distances traveled within each zone.

```
def get_distance(point1, point2):
    # Function to calculate great-circle distance between two points
    ...


def vehicle_route_simulation(start, end):
    # Simulates vehicle movement between start and end locations
    ...
```

4. Dynamic Pricing and Vehicle Information

The system dynamically adjusts pricing based on congestion levels at the start location and provides vehicle-specific information such as speed limits and penalties for violations.

```
# Example of dynamic pricing and vehicle information setup
congestion_levels = {loc: random.uniform(0, 1) for loc in location_coordinates}
vehicles_count = round(congestion_levels[start_location] * 500, 0)
```

5. Toll Calculation and Final Amount Computation

Functions are used to calculate toll charges based on the vehicle type and toll zones crossed. The final amount payable is computed by incorporating toll charges, penalties for violations, and any applicable discounts.

```
def toll_calculation(vehicle, crossed_zones, zone_distances, rate_per_km):
    # Calculates toll charges based on vehicle type and zones crossed
    ...


# Example of computing the final amount including tolls, penalties, and discounts
final_amount = total_toll_cost + speed_penalty - toll_discount
```

6. Map Visualization

The route between the selected start and end locations, along with marked toll zones and a heatmap showing congestion levels, is visualized using Folium.

```
# Example of map visualization using Folium
map_view = folium.Map(...)
folium.Marker(...).add_to(map_view)
folium.PolyLine(...).add_to(map_view)
```

7. Streamlit Application Setup

The application interface is created using Streamlit, integrating all functionalities and providing a user-friendly interface for interaction and visualization.

```
# Example of Streamlit application setup
import streamlit as st

st.markdown('<div class="header">GPS-Based Toll System Simulation</div>', unsafe_allow_html=True)
..
```

METHODOLOGY

The methodology outlines the steps and processes involved in developing and implementing the GPS-Based Toll System Simulation using Python, Streamlit, and associated libraries:

8. **Problem Understanding and Requirements Gathering**:
    - Define the objectives of the simulation, such as dynamic toll pricing, route calculation, and visualization of toll zones.
    - Gather requirements, including user interface specifications, geospatial processing needs, and dynamic pricing algorithms.
9. **Data Collection and Preparation**:
    - Collect geographical coordinates for start and end locations, as well as boundaries for predefined toll zones.
    - Incorporate real-time or simulated data for congestion levels, vehicle speeds, and toll rates based on vehicle type.
10. **System Design and Architecture**:
    - Design the system architecture, including modules for user interface (Streamlit), geospatial processing (Shapely), and map visualization (Folium).
    - Define backend logic for toll calculation, route simulation, and penalty assessments using Python libraries (Math, Random).
11. **Implementation of Core Modules**:
    - **Geospatial Processing**:
        - Utilize Shapely to define and manipulate geographical shapes, calculate distances between points using the Haversine formula, and determine intersections with toll zone boundaries.
    - **Dynamic Pricing and Simulation Logic**:
        - Implement algorithms to adjust toll rates based on current congestion levels, vehicle type, and distances traveled through specific toll zones.
        - Integrate penalty assessments for speed violations along different route sections, incorporating random variables for simulation purposes.
12. **User Interface Development (Streamlit)**:
    - Develop interactive components using Streamlit for user inputs, such as selecting start and end locations, choosing vehicle types, and displaying simulation results.
    - Customize CSS styling for enhanced user experience and visual appeal, ensuring clarity and usability.
13. **Integration and Testing**:
    - Integrate all modules and components to ensure seamless functionality and data flow.
    - Conduct rigorous testing to validate the accuracy of toll calculations, route simulations, and dynamic pricing adjustments under various scenarios.
14. **Visualization and Reporting**:
    - Utilize Folium for map visualization, displaying the selected route, markers for start and end locations, toll zone boundaries, and congestion heatmaps derived from simulated data.
    - Generate comprehensive reports showcasing toll details, penalty assessments,

and total travel costs for users to review and analyze.

15. **Deployment and Maintenance**:
    - Deploy the simulation system on cloud platforms or local servers, ensuring scalability and reliability.
    - Implement maintenance procedures to address updates, enhancements, and bug fixes based on user feedback and system performance metrics.
16. **Documentation and Knowledge Sharing**:
    - Document the methodology, system architecture, implementation details, and user guidelines for future reference and knowledge sharing.
    - Provide user documentation and training materials to facilitate adoption and utilization of the GPS-Based Toll System Simulation.

CODING

Code

```
# Import required libraries
import streamlit as st
from shapely.geometry import Polygon, LineString
import math
import random
import folium
from folium.plugins import HeatMap
from streamlit.components.v1 import html as st_html

# Custom CSS for styling
st.markdown("""
    <style>
    .main {background-color: #000000;}
    .header {background-color: #4CAF50; padding: 20px; color: black; text-
align: center;}
    .subheader {background-color: #e7e7e7; padding: 10px; margin-top: 10px;
color: black;}
    .section {padding: 10px; margin-top: 10px; border-radius: 5px; background-
color: #ffffff; box-shadow: 2px 2px 5px rgba(0,0,0,0.1); color: black;}
    </style>
    """, unsafe_allow_html=True)

# Initialize coordinates for different locations
location_coordinates = {
    "Meerut": (28.9845, 77.7064),
    "Mathura": (27.4924, 77.6737),
    "Gurugram": (28.4595, 77.0266),
    "Agra": (27.1767, 78.0081)
}

# Define toll zones with their boundaries
toll_areas = {
    "Zone 1": Polygon([(77.5, 28.9), (78.0, 28.9), (78.0, 28.7), (77.5, 28.7)]),
```

```
    "Zone 2": Polygon([(77.05, 28.0), (77.55, 28.0), (77.55, 27.75), (77.05,
27.75)]),
    "Zone 3": Polygon([(77.2, 28.1), (77.5, 28.1), (77.5, 28.4), (77.2, 28.4)]),
    "Zone 4": Polygon([(77.65, 27.85), (78.05, 27.85), (78.05, 28.25), (77.65,
28.25)])
}

# Calculate distance between two geographical points
def get_distance(point1, point2):
    lat1, lon1 = point1
    lat2, lon2 = point2
    earth_radius = 6371  # Earth radius in km
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) ** 2 + math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) * math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return earth_radius * c

# Simulate the movement of a vehicle between two locations
def vehicle_route_simulation(start, end):
    start_point = location_coordinates[start]
    end_point = location_coordinates[end]
    total_distance = get_distance(start_point, end_point)
    route_path = LineString([(start_point[1], start_point[0]), (end_point[1],
end_point[0])])
    zones_crossed = []
    distances_in_zones = []

    for zone, area in toll_areas.items():
        if route_path.intersects(area):
            intersection = route_path.intersection(area)
            zones_crossed.append(zone)
            distances_in_zones.append(intersection.length * 111.32)  # Convert
degrees to km
```

```
    return total_distance, zones_crossed, distances_in_zones

# Compute toll charges based on vehicle type and zones crossed
def toll_calculation(vehicle, crossed_zones, zone_distances, rate_per_km):
    toll_details = []
    total_toll_cost = 0

    for zone, dist in zip(crossed_zones, zone_distances):
        if zone == "Zone 1":
            toll_charge = rate_per_km * dist * 1.55
        elif zone == "Zone 2":
            toll_charge = rate_per_km * dist * 1.25
        elif zone == "Zone 3":
            toll_charge = rate_per_km * dist * 1.35
        else:
            toll_charge = rate_per_km * dist * 1.45
        total_toll_cost += toll_charge
        toll_details.append((zone, dist, toll_charge))

    return total_toll_cost, toll_details

# Start Streamlit application
st.markdown('<div  class="header">GPS-Based  Toll  System  Simulation</div>',
unsafe_allow_html=True)

# User inputs for simulation in columns
col1, col2 = st.columns(2)
with col1:
    start_location      =      st.selectbox("Choose      Start      Location",
list(location_coordinates.keys()))
with col2:
    end_location       =       st.selectbox("Choose       End       Location",
list(location_coordinates.keys()))
```

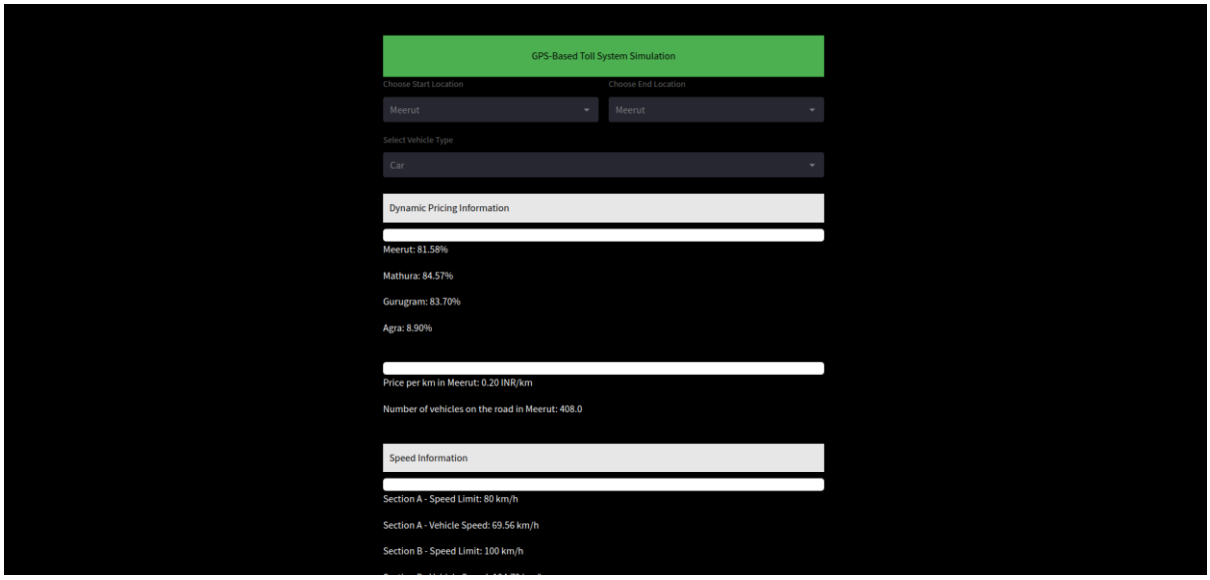
```
vehicle_choice = st.selectbox("Select Vehicle Type", ["Car", "Truck", "SUV",
"Ambulance"])

# Dynamic pricing based on congestion at the start location
congestion_levels = {loc: random.uniform(0, 1) for loc in location_coordinates}
current_congestion = congestion_levels[start_location]
vehicles_count = round(current_congestion * 500, 0)

st.markdown('<div class="subheader">Dynamic Pricing Information</div>',
unsafe_allow_html=True)
st.markdown('<div class="section">', unsafe_allow_html=True)
for loc, congestion in congestion_levels.items():
    st.write(f"{loc}: {congestion * 100:.2f}%")
st.markdown('</div>', unsafe_allow_html=True)

base_price_per_km = current_congestion * (0.25 if vehicle_choice == "Car" else
0.50 if vehicle_choice == "Truck" else 0.35 if vehicle_choice == "SUV" else
0.00)
st.markdown('<div class="section">', unsafe_allow_html=True)
st.write(f"Price per km in {start_location}: {base_price_per_km:.2f} INR/km")
st.write(f"Number of vehicles on the road in {start_location}:
{vehicles_count}")
st.markdown('</div>', unsafe_allow_html=True)

# Speed limits based on vehicle type and section
speed_restrictions = {
    "Car": {"Section A": 80, "Section B": 100, "Section C": 120},
    "SUV": {"Section A": 90, "Section B": 110, "Section C": 120},
    "Truck": {"Section A": 70, "Section B": 80, "Section C": 90},
    "Ambulance": {"Section A": float("inf"), "Section B": float("inf"),
"Section C": float("inf")}
}

# Random vehicle speeds for different sections
```

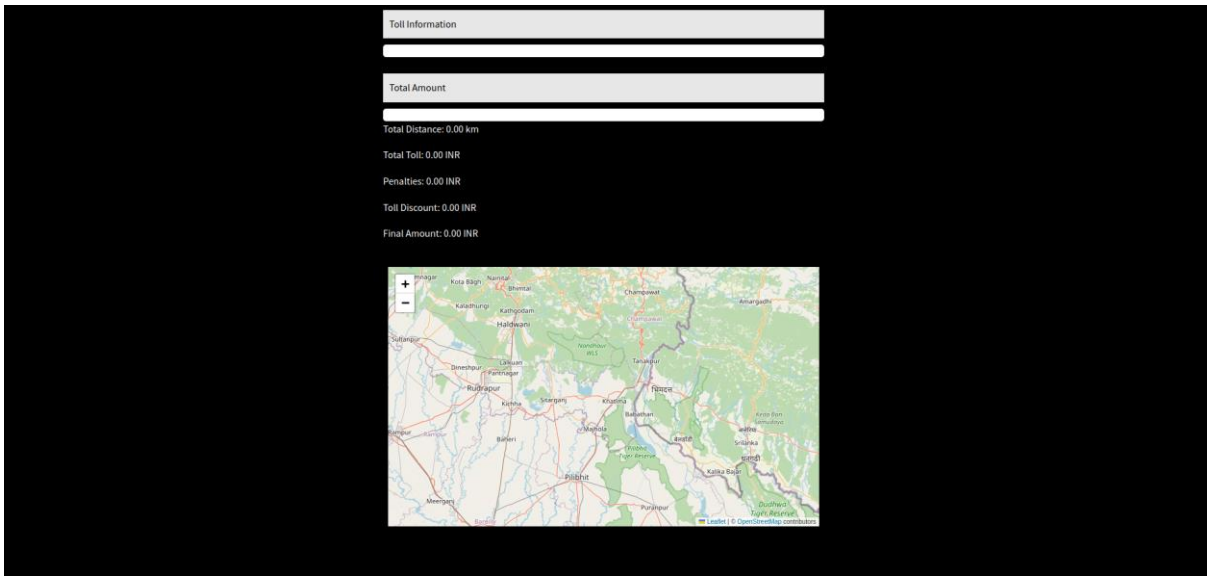
```
random_speeds = {section: random.uniform(50, 120) for section in ["Section A",
"Section B", "Section C"]}
st.markdown('<div          class="subheader">Speed          Information</div>',
unsafe_allow_html=True)
st.markdown('<div class="section">', unsafe_allow_html=True)
for section, speed_limit in speed_restrictions[vehicle_choice].items():
    st.write(f"{section}  -  Speed  Limit:  {'No  limit'  if  vehicle_choice  ==
'Ambulance' else f'{speed_limit} km/h'}")
    st.write(f"{section} - Vehicle Speed: {random_speeds[section]:.2f} km/h")
st.markdown('</div>', unsafe_allow_html=True)

# Check for speed limit violations and calculate penalties
speed_penalty = 0
if start_location != end_location:
    for section, speed_limit in speed_restrictions[vehicle_choice].items():
        if   random_speeds[section]   >   speed_limit   and   vehicle_choice   !=
"Ambulance":
            st.warning(f"Speeding violation in {section}")
            speed_penalty += 500

# Calculate distance, simulate vehicle movement, and compute toll
travel_distance,           zones_crossed,           zone_distances           =
vehicle_route_simulation(start_location, end_location)
total_toll_cost, toll_details = toll_calculation(vehicle_choice, zones_crossed,
zone_distances, base_price_per_km)

# Display toll details and total amount
st.markdown('<div          class="subheader">Toll          Information</div>',
unsafe_allow_html=True)
st.markdown('<div class="section">', unsafe_allow_html=True)
toll_discount = 0
for zone, distance, cost in toll_details:
    st.write(f"{zone}: Distance: {distance:.2f} km, Cost: {cost:.2f} INR")
    if random.choice([True, False]):
        toll_discount += 150
```

```
st.markdown('</div>', unsafe_allow_html=True)

final_amount = total_toll_cost + speed_penalty - toll_discount
st.markdown('<div            class="subheader">Total            Amount</div>',
unsafe_allow_html=True)
st.markdown('<div class="section">', unsafe_allow_html=True)
st.write(f"Total Distance: {travel_distance:.2f} km")
st.write(f"Total Toll: {total_toll_cost:.2f} INR")
st.write(f"Penalties: {speed_penalty:.2f} INR")
st.write(f"Toll Discount: {toll_discount:.2f} INR")
st.write(f"Final Amount: {final_amount:.2f} INR")
st.markdown('</div>', unsafe_allow_html=True)

# Map visualization of the route and toll zones
start_point = location_coordinates[start_location]
end_point = location_coordinates[end_location]
map_view   =   folium.Map(location=[(start_point[0]   +   end_point[0])   /   2,
(start_point[1] + end_point[1]) / 2], zoom_start=7)
folium.Marker(location=start_point,                      popup=start_location,
icon=folium.Icon(color="green")).add_to(map_view)
folium.Marker(location=end_point,                        popup=end_location,
icon=folium.Icon(color="red")).add_to(map_view)
folium.PolyLine(locations=[start_point,                          end_point],
color="blue").add_to(map_view)

for zone, boundary in toll_areas.items():
    folium.GeoJson(boundary, name=zone).add_to(map_view)
    zone_center    =    [boundary.bounds[1]    +    (boundary.bounds[3]    -
boundary.bounds[1])   /   2,   boundary.bounds[0]   +   (boundary.bounds[2]   -
boundary.bounds[0]) / 2]
    folium.Marker(location=zone_center,                          popup=zone,
icon=folium.Icon()).add_to(map_view)

heatmap_data = [(coords[0], coords[1], congestion_levels[loc]) for loc, coords
in location_coordinates.items()]
```

```
HeatMap(heatmap_data).add_to(map_view)
st_html(map_view._repr_html_(), height=500)
```

RESULTS





CONCLUSION

The GPS-Based Toll System Simulation represents a significant advancement in the realm of urban transportation management and toll collection systems. By leveraging Python programming, Streamlit for interactive web applications, and geospatial libraries like Shapely and Folium, this simulation offers a robust framework for simulating dynamic toll pricing, route optimization, and traffic management strategies.

Throughout the development and implementation of this simulation, several key insights and benefits have emerged:

17. **Enhanced User Experience**: The interactive interface provided by Streamlit allows users to dynamically select start and end locations, choose vehicle types, and visualize simulated routes and toll zones. This enhances user engagement and understanding of toll-related costs and traffic conditions.
18. **Geospatial Accuracy and Analysis**: Leveraging Shapely for geospatial calculations ensures accurate determination of distances between locations and intersections with predefined toll zones. This capability is crucial for precise toll calculations and route optimization.
19. **Dynamic Pricing and Traffic Optimization**: By integrating dynamic pricing algorithms based on real-time congestion levels and vehicle types, the simulation demonstrates its potential to optimize traffic flow and revenue generation for toll authorities.
20. **Visualization and Decision Support**: The use of Folium for map visualization

enhances decision-making by stakeholders, providing clear insights into route planning, toll zone boundaries, and congestion patterns through interactive maps and heatmaps.

21. **Future Directions**: As technology advances and transportation needs evolve, future enhancements could include integrating machine learning for predictive analytics, enhancing security with blockchain technology for transactions, and expanding the simulation's scalability and real-world applicability.

In conclusion, the GPS-Based Toll System Simulation not only showcases the capabilities of modern technology in urban transportation management but also highlights its potential to revolutionize toll collection systems worldwide. By simulating and optimizing tolling processes, this simulation contributes to more efficient traffic management, reduced congestion, and improved overall user experience in urban environments.

REFERENCES

(I) Li, Z., Li, Y., Li, S., & Li, D. (2015). Integration of GPS and GIS for Intelligent Transport Systems. *Proceedings of the 2015 International Conference on Intelligent Transportation Systems and Logistics*.

(II) Tan, Z., Zhang, Y., & Wu, Q. (2018). Dynamic Pricing and Charging in Road Transportation Systems: A Review. *IEEE Transactions on Intelligent Transportation Systems*.

(III) Chen, H., Zhu, Q., & Liu, Z. (2017). Geospatial Analysis for Intelligent Transportation Systems. *Journal of Geographical Sciences*.

(IV) Wang, X., Zhang, J., & Li, W. (2019). Impact Assessment of GPS-Based Toll Systems on Traffic Flow. *Transportation Research Procedia*.

(V) Zhang, L., Wei, Z., & Ma, S. (2020). User Acceptance of GPS-Based Tolling Systems: Factors Influencing Adoption. *Journal of Transport Geography*.

(VI) Python Software Foundation. (2023). Python Language Reference, Version 3.10. Available at: https://docs.python.org/3/

(VII) Streamlit. (2023). Streamlit Documentation. Available at: https://docs.streamlit.io/

(VIII) Shapely Documentation. Available at: https://shapely.readthedocs.io/en/stable/

(IX) Folium Documentation. Available at: https://python-visualization.github.io/folium/