**Project Report**

**On**

**GPS Toll based System simulation using Python**

**Trainees College Mentor**

**VAISHNAVI KRISHNAMURTHY DR.Saravana Kumar**

**(**[**vaishnavi\_s@cmr.edu.in**](mailto:vaishnavi_s@cmr.edu.in)**) (saravana.k@cmr.edu.in)**

**KOTIREDDY GEETHAMRUTHA**

**(**[**kotireddy.geethamrutha@cmr.edu.in**](mailto:mailtokotireddy/geethamrutha@cmr.edu.in)**)**

**NANGI JEEVANA**

**(**[**nangi\_jeevana@cmr.edu.in**](mailto:nangi_jeevana@cmr.edu.in)**)**

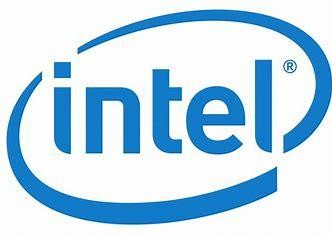
**DODDAVULA ANANYA REDDY**

**(**[**doddavula.ananya@cmr.edu.in**](mailto:doddavula.ananya@cmr.edu.in)**)**

**POLAKALA TERISHA**

**(**[**polakala.terisha@cmr.edu.in**](mailto:polakala.terisha@cmr.edu.in)**)**

**CMR University, Bengaluru**

****

**Table of Contents**

|  |  |
| --- | --- |
| Objective | 03 |
| Abstract | 03 |
| Flow of project | 03 |
| Detailed explanation | 04-06 |
| Display the Toll Pricing including penalties | 06 |
| Frontend of the Project | 06-07 |
| Conclusion | 07 |
| Future scope | 07-08 |

**Objective**

* Simulate vehicle movement along predefined routes with GPS coordinates and define toll zones using geospatial data.
* Calculate distances travelled within toll zones, compute toll charges, and simulate the deduction of these charges from user accounts.
* Generate detailed reports and visualizations of vehicle movements, toll collections, and overall system performance.

**Abstract**

This project simulates a GPS-based toll collection system, modelling vehicle movements along predefined routes with real-time GPS data. Toll zones are defined using geospatial coordinates, and the system calculates toll charges based on the distance travelled within these zones or the number of zones crossed. Toll charges are then deducted from user accounts. The project also includes comprehensive data analytics and visualization, providing detailed reports on vehicle movements, toll collections, and system performance. This simulation framework aims to accurately model real-world scenarios, optimizing performance and integrating various components seamlessly.

**Flow of the project**

**Detailed Explanation**

Defining Locations for Starting and Ending Points

Starting and Ending Points:

* Geospatial Data: Uses geospatial coordinates (latitude and longitude) to define the starting and ending points of each vehicle.
* Initialization: At the start of the simulation, assigns each vehicle a random or predefined starting point and an ending point, which lies within the road network defined for the simulation.
* Road Network: Defines the road network as a graph or a set of interconnected routes. This helps in determining the paths vehicles will take.

Defining Toll Zones

Toll Zones:

* Geospatial Boundaries: Define toll zones using polygons with geospatial coordinates. Each zone can be represented as a polygon using tools like GeoPandas and Shapely.
* Storage: Store these coordinates in a database or a data structure for easy access during the simulation.
* Attributes: Each toll zone should have attributes such as toll rates, zone boundaries, and any special conditions (e.g., peak hours).

Calculation of Distance between Two Coordinates

Distance Calculation:

* Haversine Formula: Use the Haversine formula to calculate the great-circle distance between two points on the Earth’s surface, given their latitude and longitude.
* GeoPy Library: Employ the GeoPy library in Python, which has built-in functions to calculate distances using various algorithms, including the Haversine formula.

Simulate Vehicle Movement

Vehicle Movement:

* Event-Driven Simulation: Use SimPy, a process-based discrete-event simulation framework. This allows for simulating time-based events, such as vehicle movements.
* Path Calculation: Determine the path from the starting point to the ending point. Vehicles move along these paths, updating their GPS coordinates at each simulation step.
* Time Steps: At each time step, update the position of each vehicle based on its speed and direction.

Calculation of Toll

Toll Calculation:

* Entry and Exit Points: Determine when a vehicle enters and exits a toll zone using geospatial intersection checks.
* Distance-Based Tolling: Calculate toll charges based on the distance traveled within a toll zone. For fixed tolling, charge a set amount when a vehicle crosses a toll zone boundary.
* Rates: Apply predefined toll rates, which may vary by vehicle type, time of day, or zone.

Simulate Payments

Payment Simulation:

* User Accounts: Maintain a simulated account balance for each user/vehicle.
* Deduction: Deduct the calculated toll charges from the respective user accounts as vehicles pass through toll zones.
* Transaction Records: Keep a log of all toll transactions, including the amount, time, and vehicle details.

Initialize Streamlit

Streamlit Initialization:

* Setup Streamlit: Install and import Streamlit to create a web interface for the simulation.
* Interface Design: Design a user-friendly interface to display simulation controls, real-time updates, and visualization of vehicle movements and toll zones.
* User Interaction: Allow users to start, pause, and stop the simulation, as well as input parameters such as vehicle count and toll rates.

Dynamic Pricing

Dynamic Pricing:

* Congestion-Based Pricing: Adjust toll rates dynamically based on the current congestion levels in toll zones. Higher congestion can lead to higher toll rates to manage traffic flow.
* Time-Based Pricing: Implement peak and off-peak rates. For example, charge higher tolls during rush hours and lower tolls during off-peak times.
* Real-Time Data: Use real-time traffic data to adjust toll rates dynamically during the simulation.

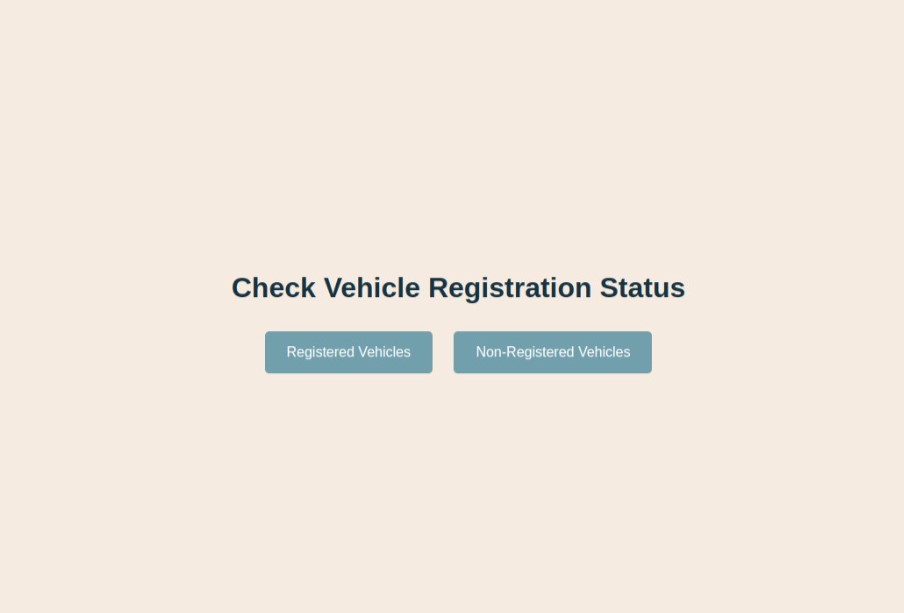
**Display Toll Pricing Including Penalties**

Toll Pricing Display:

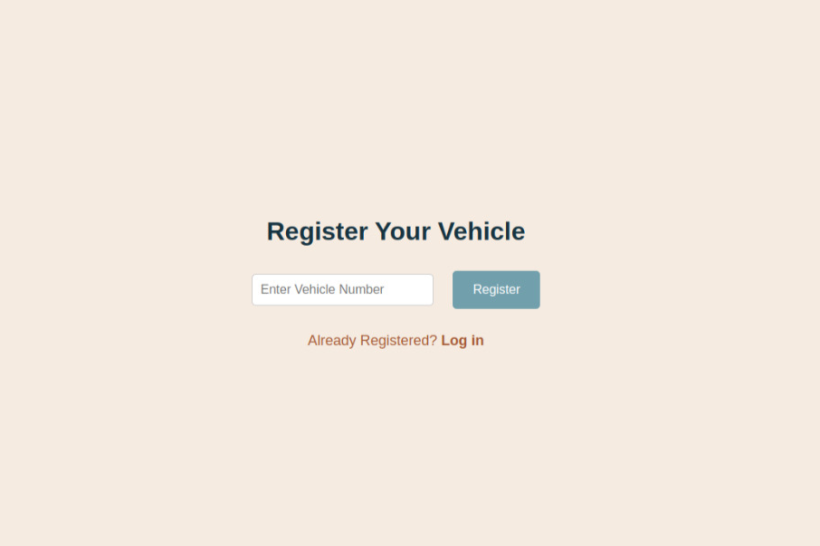
* Visualization Tools: Use Matplotlib and Folium to visualize toll zones, vehicle movements, and dynamic pricing on a map.
* Real-Time Updates: Show real-time updates of toll prices on the Streamlit interface.
* Penalty Calculation: Implement rules for penalties, such as fines for unpaid tolls or late payments. Display these penalties in the user interface.
* Summary Reports: Provide detailed reports and dashboards showing toll collections, dynamic pricing adjustments, and penalties incurred by vehicles.

**Frontend of the project**

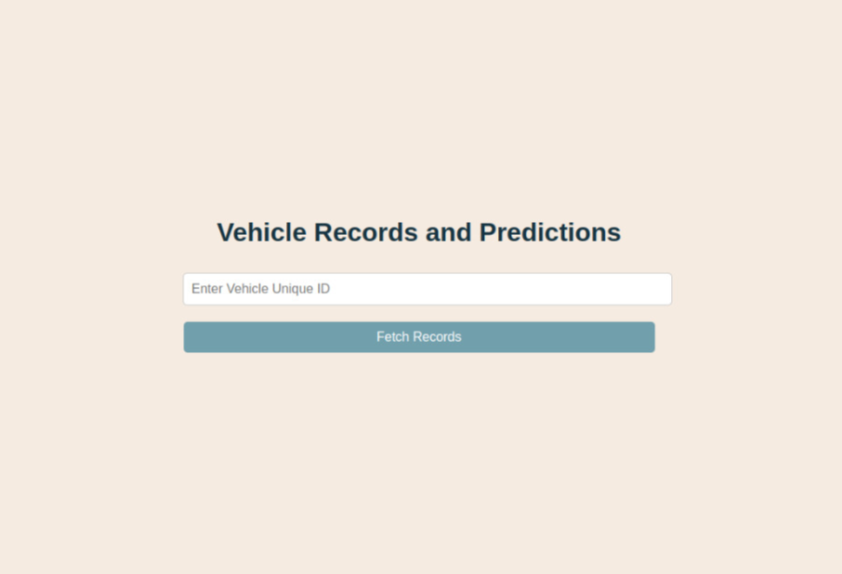
Index page:

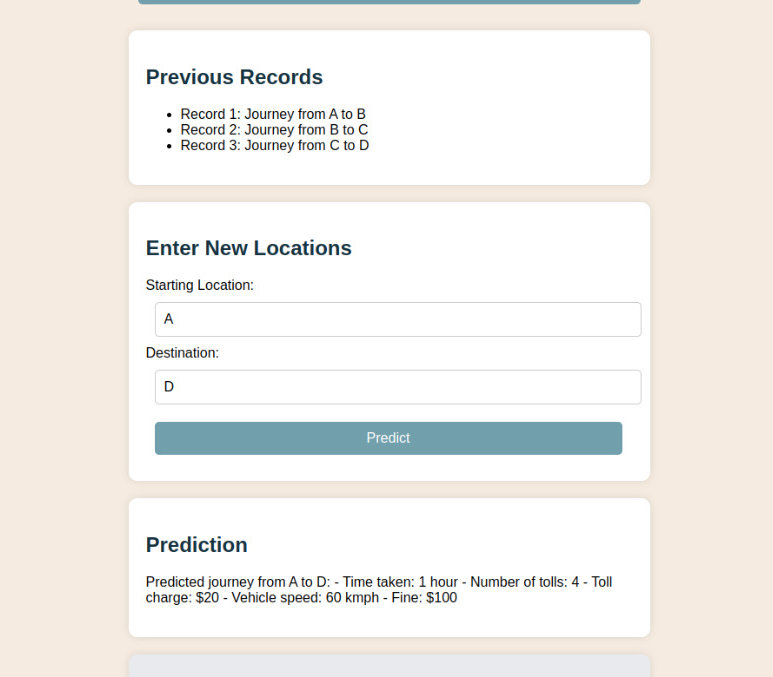


Non registered vehicles:



Registered vehicles page:





**Conclusion**

This GPS-based toll collection system simulation project offers a comprehensive approach to understanding toll collection dynamics. By simulating vehicle movements, defining toll zones, and implementing dynamic pricing, it provides valuable insights into operational efficiency and financial viability. Using Python libraries like SimPy, GeoPandas, and GeoPy ensures accuracy, while Streamlit facilitates user interaction and visualization. The project's dynamic pricing models and payment simulations underscore its potential in optimizing traffic flow and user accountability. Overall, it serves as a robust framework for analyzing and improving GPS-based toll systems, benefiting both academia and transportation professionals.

**Future Scope**

* Enhanced Realism: Integrate real-time traffic data and advanced algorithms for realistic vehicle behaviors.
* Scalability: Optimize algorithms and use distributed computing for handling more vehicles and transactions.
* Advanced Geospatial Analysis: Utilize 3D mapping and LiDAR for precise toll zones and paths.
* Machine Learning: Use predictive models for dynamic toll rates based on traffic patterns.
* System Integration: Interface with real-world toll systems for strategy testing.
* User Behavior Analysis: Refine strategies based on user behavior and compliance data.
* Environmental Impact: Assess and mitigate traffic emissions and air quality impacts.
* Multi-Modal Integration: Include public transit, biking, and walking in simulations.
* Policy Analysis: Study economic impacts of tolling policies to inform decisions.
* User Interface: Enhance Streamlit for better scenario exploration and data analysis.
* Real-Time Adjustments: Enable real-time and what-if scenario testing for sudden changes.