**TECHNICAL REPORT**

**GPS TOLL BASED SYSTEM SIMULATION**

**TEAM NAME: INTELlSPIRIT**

**DATE OF SUBMISSION: 15/07/2023**

**Abstract**

In a nation where transportation reigns supreme and Transportation has emerged as a dominant part of the country. Toll plazas stand as linchpins in road networks. Yet, manual toll collection persists, burdening both time and resources. Toll plazas play a crucial role in maintaining road transportation. At present, manual toll collection is the most widely used collection method in India. It significantly requires a toll collector or attendant. Due to manual intervention, the processing time at toll plazas is highest. Enter Geo Toll-Link—a pioneering solution poised to reshape this landscape. Fuelled by GPS technology, Cloud Technology, IoT it transcends the conventional toll plaza. This project aims in designing a system, which automatically identifies the vehicle that advance towards the Starting point of the highway and initiates the system to process the distance travelled through the highway. Geo Toll-Link's real-time accuracy and automation pave the way for smoother journeys, propelling India's transportation into the future.

**Introduction**

Tolling can be defined as the practice of fee collection from motorists who use roadway facilities (U.S. Department of Transportation Federal Highway Administration, n.d.). The collected toll fees are often used for expansion, operational and maintenance purposes. Traditionally, toll fee was manually collected via cash or card transactions at dedicated toll booths. With this, time consumption is more. To overcome this, automation process of GPS toll plaza system simulation has a been introduced. A GPS toll-based system simulation is a computer-based representation of a toll collection system that utilizes GPS technology to track and charge vehicles for using certain roads, highways, or bridges. The simulation aims to allowing for testing, evaluation, and optimization of various scenarios and parameters without the need for physical infrastructure.

In this simulation, GPS data is used to track the movement of vehicles and determine when they enter and exit designated toll zones. The system can then calculate the applicable tolls based on factors such as distance travelled, vehicle type, and time of day.

By using a simulation, stakeholders can gain valuable insights and optimize the design and operation of GPS toll-based systems before actual implementation, reducing costs and improving overall efficiency.

**Motivation Behind the Project**

Now a days as the count of vehicle is increasing, it becomes difficult to collect the fees manually and control the traffic and other necessary needs. The project has been designed for the automation of toll tax payments using GPS Technology. Main motives of the Project are:

* Increased accuracy
* Improved user experience
* Reduced labour costs
* Efficient toll collection
* Environmental benefits

**Data Source**

For a GPS toll-based system, the data sources you might need include:

1. GPS Data: Real-time location data from GPS devices installed in vehicles.

2. Map Data: Detailed mapping information, including roads, highways, and toll locations, from providers like Google Maps, OpenStreetMap, or proprietary sources.

3.Traffic Data: Current traffic conditions to calculate dynamic toll rates, available from services like INRIX, TomTom, or local traffic authorities.

4.Vehicle Information: Data on vehicle type, size, and weight, which can affect toll rates, sourced from vehicle registration databases.

5.Toll Rates: Predefined toll rates for different road segments, available from toll authorities or operators.

6.Payment Data: Information on payment methods and transactions, often handled by payment gateways and financial institutions.

7.User Data: Information about the drivers or vehicle owners, including account details and payment preferences, stored in secure databases.

Combining these data sources will enable a comprehensive GPS toll-based system that can dynamically calculate and charge tolls based on real-time conditions and vehicle specifics.

**Working of the Project**

The provided code simulates a toll collection system using the simpy library for discrete event simulation. It defines classes for Toll Zone, User, and Vehicle, where Toll Zone represents areas where tolls are collected, User represents vehicle owners with specific balances and vehicle types, and Vehicle represents the movement of vehicles through toll zones. The Vehicle class calculates tolls based on the distance travelled within each toll zone and adjusts the user's balance accordingly. The Simulation Manager class initializes the simulation environment, creates instances of vehicles and users, and runs the simulation for a specified duration. After the simulation, it visualizes the results using folium for mapping vehicle routes and toll zones, and matplotlib for plotting graphs of the simulation data. Alerts for toll payments and a summary of the simulation results are displayed using tkinter message boxes. Finally we can summarize the working of this project as Vehicle Equipped with GPS, Toll Charge, Data Analysis, Payment Processing, GPS Data Transmission.

**Advantages of the project**

The advantages of GPS toll-based system simulation include:

1. Cost savings: Simulation reduces the need for physical infrastructure and field testing, saving costs and resources.

2. Increased accuracy: Simulation allows for precise modeling of toll zones, vehicle movements, and payment processing, reducing errors and inaccuracies.

3. Improved efficiency: Simulation enables testing and optimization of various scenarios, leading to enhanced system performance and efficiency.

4. Flexibility and scalability: Simulation allows for easy modifications and expansions to accommodate changing transportation needs and infrastructure.

5. Reduced congestion: Simulation helps optimize toll pricing and traffic management, reducing congestion and improving travel times.

6. Environmental benefits: Simulation can evaluate the impact of tolling on emissions and encourage more sustainable transportation options.

7. Improved user experience: Simulation enables testing of user-friendly payment interfaces and notification systems, enhancing driver satisfaction.

8. Risk reduction: Simulation identifies potential issues and allows for mitigation strategies, reducing risks and improving overall system reliability.

By leveraging these advantages, GPS toll-based system simulation enables stakeholders to design, optimize, and operate efficient, effective, and sustainable tolling systems.

**Conclusion**

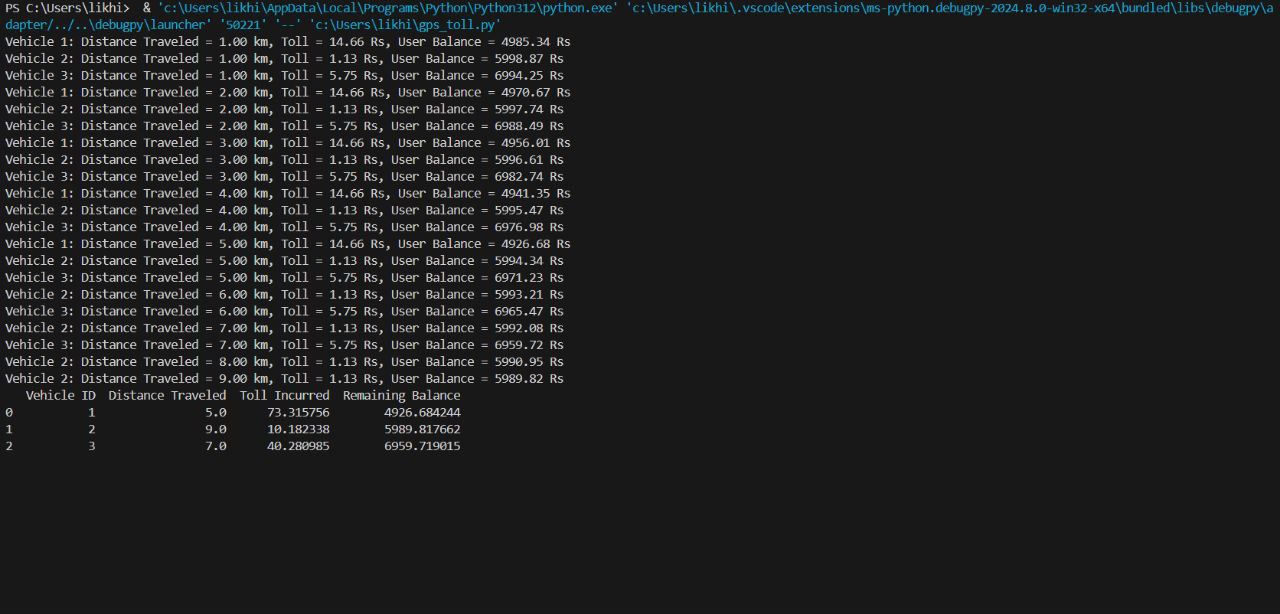
From this project we can conclude that by implementing GPS based toll system, the need for Physical toll plaza would be eliminated. thus, resulting reduction of traffic congestion near toll plaza. Since the entire toll tax collection will happen automatically, hassle free experience can be delivered to the vehicle drivers on the road. GPS toll-based system simulation is a powerful tool for designing, testing, and optimizing tolling systems. The simulation provides a virtual environment to analyse and optimize the system's performance, allowing for data-driven decision-making and informed investments. By adopting GPS toll-based system simulation, cities and transportation authorities can create more efficient, sustainable, and user-friendly transportation systems, improving the overall travel experience for millions of people.

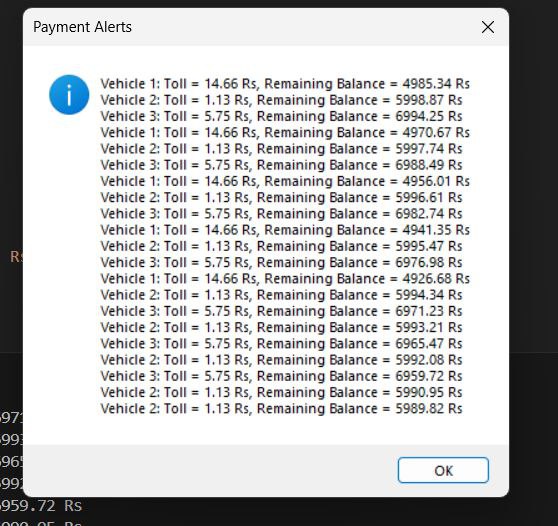
**Working code**

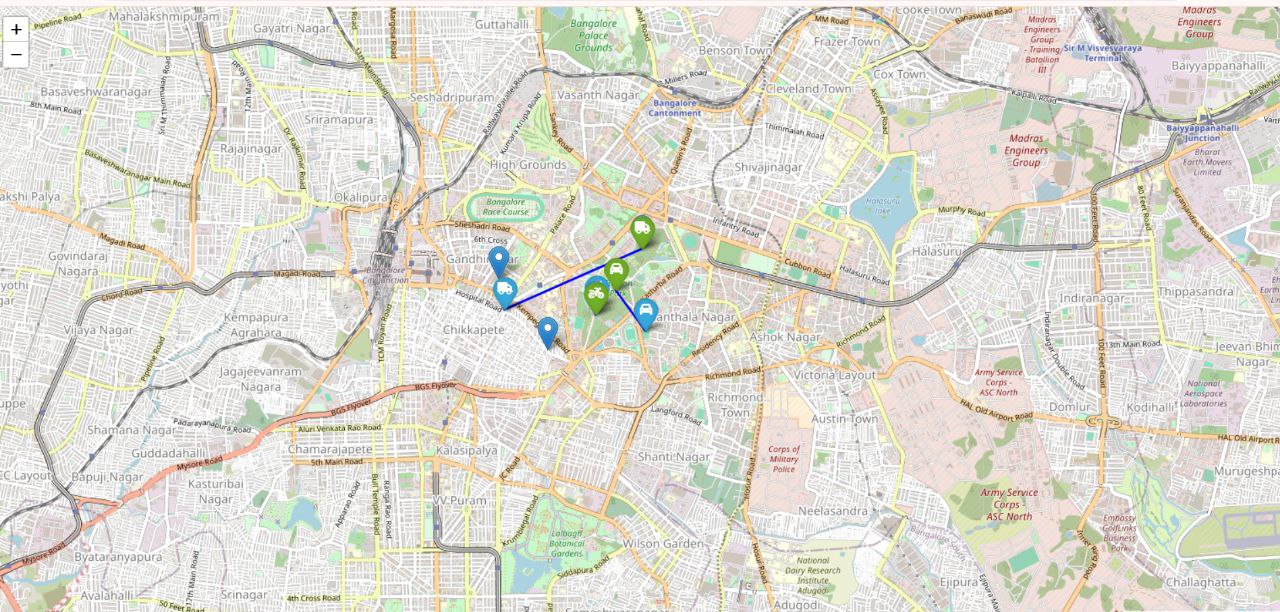
import simpy  
import geopandas as gpd  
from shapely.geometry import Point, LineString  
import pandas as pd  
import matplotlib.pyplot as plt  
import folium  
import webbrowser  
import tkinter as tk  
from tkinter import messagebox  
  
# Constants  
SIMULATION\_TIME = 100  # Simulation time in minutes  
SIM\_SPEED = 60  # Speed of simulation (km/h)  
  
# List to collect alerts  
alerts = []  
  
# Function to collect payment alerts  
def collect\_payment\_alert(vehicle\_id, toll, balance):  
    alert\_message = f"Vehicle {vehicle\_id}: Toll = {toll:.2f} Rs, Remaining Balance = {balance:.2f} Rs"  
    alerts.append(alert\_message)  
  
# Function to show all collected alerts at once  
def show\_all\_payment\_alerts():  
    root = tk.Tk()  
    root.withdraw()  # Hide the root window  
    all\_alerts\_message = "\n".join(alerts)  
    messagebox.showinfo("Payment Alerts", all\_alerts\_message)  
    root.destroy()  
  
# Function to show summary message  
def show\_summary\_message(df):  
    root = tk.Tk()  
    root.withdraw()  # Hide the root window  
    summary\_message = "Simulation Summary:\n\n" + df.to\_string(index=False)  
    messagebox.showinfo("Summary", summary\_message)  
    root.destroy()  
  
class TollZone:  
    def *init*(self, zone\_id, location, rate\_per\_km):  
        self.id = zone\_id  
        self.location = location.buffer(0.01)  # Creating a buffer around the location  
        self.rate\_per\_km = rate\_per\_km  
  
class User:  
    def *init*(self, user\_id, balance, vehicle\_type, distance):  
        self.id = user\_id  
        self.balance = balance  
        self.vehicle\_type = vehicle\_type  
        self.distance = distance  
  
class Vehicle:  
    def *init*(self, env, vehicle\_id, start\_location, end\_location, user):  
        self.env = env  
        self.vehicle\_id = vehicle\_id  
        self.start\_location = start\_location  
        self.end\_location = end\_location  
        self.route = LineString([self.start\_location, self.end\_location])  
        self.distance\_traveled = 0  
        self.current\_location = start\_location  
        self.user = user  
        self.toll\_incurred = 0  # Total toll incurred  
        self.action = env.process(self.move())  
  
    def move(self):  
        while self.distance\_traveled < self.user.distance:  
            yield self.env.timeout(1)  # Move every 1 minute in simulation time  
            self.distance\_traveled += SIM\_SPEED / 60  # Update distance traveled (km per minute)  
            self.current\_location = self.route.interpolate(self.distance\_traveled / self.route.length, normalized=True)  
            current\_toll = self.calculate\_toll()  
            self.user.balance -= current\_toll  
            self.toll\_incurred += current\_toll  # Accumulate the toll incurred  
            print(f"Vehicle {self.vehicle\_id}: Distance Traveled = {self.distance\_traveled:.2f} km, Toll = {current\_toll:.2f} Rs, User Balance = {self.user.balance:.2f} Rs")  
            collect\_payment\_alert(self.vehicle\_id, current\_toll, self.user.balance)  
  
    def calculate\_toll(self):  
        total\_toll = 0  
        for zone in SimulationManager.toll\_zones:  
            if self.route.intersects(zone.location):  
                intersected = self.route.intersection(zone.location)  
                if isinstance(intersected, LineString):  
                    distance\_in\_zone = intersected.length  
                    toll = distance\_in\_zone \* zone.rate\_per\_km  
                    total\_toll += toll  
  
        # Adjust toll rates for different users  
        if self.user.vehicle\_type == 'truck':  
            total\_toll \*= 50  # Example discount rate  
        elif self.user.vehicle\_type == 'motorcycle':  
            total\_toll \*= 80  # Example surcharge rate  
        elif self.user.vehicle\_type =='car':  
             total\_toll \*= 60  
        return total\_toll  
        
  
class SimulationManager:  
    toll\_zones = [  
        TollZone(1, Point(77.5899, 12.9716), 5),  
        TollZone(2, Point(77.5946, 12.9781), 7),  
        TollZone(3, Point(77.5800, 12.9750), 6),  
        TollZone(4, Point(77.5850, 12.9680), 8)  
    ]  
  
    users = [  
        User(1, 5000, 'truck', 5.0),  
        User(2, 6000, 'motorcycle', 8.5),  
        User(3, 7000, 'car', 6.2)  
    ]

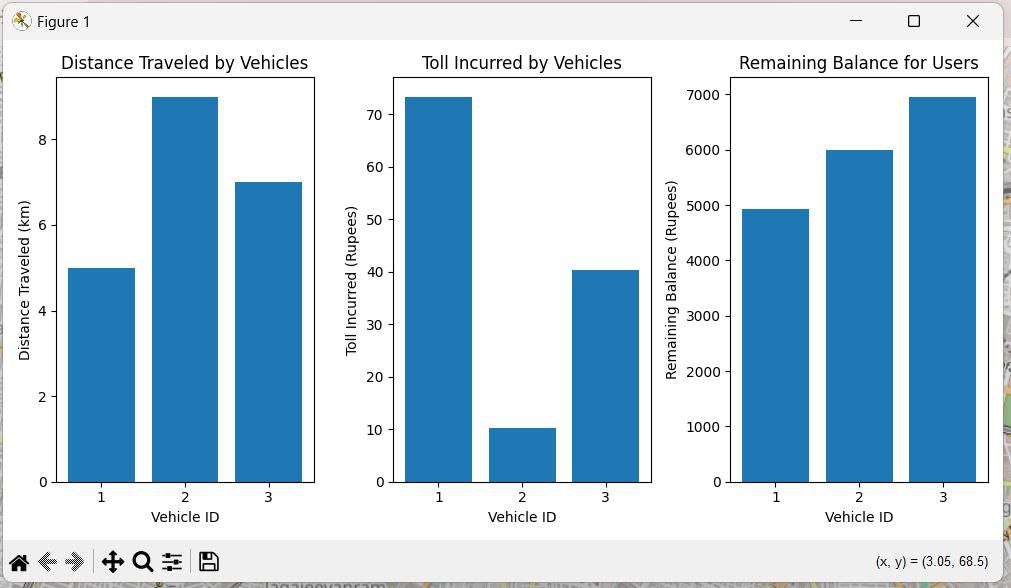
@staticmethod  
    def run\_simulation():  
        env = simpy.Environment()  
        vehicles = [  
            Vehicle(env, 1, Point(77.5806, 12.9721), Point(77.5946, 12.9781), SimulationManager.users[0]),  
            Vehicle(env, 2, Point(77.5900, 12.9723), Point(77.5899, 12.9716), SimulationManager.users[1]),  
            Vehicle(env, 3, Point(77.5950, 12.9700), Point(77.5920, 12.9740), SimulationManager.users[2])  
        ]  
  
        env.run(until=SIMULATION\_TIME)  
  
        # Collect data for all vehicles  
        data = {  
            'Vehicle ID': [v.vehicle\_id for v in vehicles],  
            'Distance Traveled': [v.distance\_traveled for v in vehicles],  
            'Toll Incurred': [v.toll\_incurred for v in vehicles],  
            'Remaining Balance': [v.user.balance for v in vehicles]  
        }  
  
        df = pd.DataFrame(data)  
        print(df)  
  
        # Show all payment alerts at once  
        show\_all\_payment\_alerts()  
  
        # Visualization (Folium map)  
        m = folium.Map(location=[12.9716, 77.5946], zoom\_start=14)  
  
        # Mark toll zones on the map  
        for zone in SimulationManager.toll\_zones:  
            folium.Marker([zone.location.centroid.y, zone.location.centroid.x],  
                          tooltip=f"Toll Zone {zone.id}").add\_to(m)  
  
        # Plot vehicle routes and their types  
        vehicle\_icons = {  
            'truck': 'truck',  
            'motorcycle': 'motorcycle',  
            'car': 'car'  
        }  
  
        for vehicle in vehicles:  
            icon = vehicle\_icons.get(vehicle.user.vehicle\_type, 'car')  # Default to car icon if type not found  
            folium.Marker([vehicle.start\_location.y, vehicle.start\_location.x],  
                          icon=folium.Icon(color='blue', icon=icon, prefix='fa'),  
                          tooltip=f"Vehicle {vehicle.vehicle\_id} (Start)").add\_to(m)  
            folium.Marker([vehicle.end\_location.y, vehicle.end\_location.x],  
                          icon=folium.Icon(color='green', icon=icon, prefix='fa'),  
                          tooltip=f"Vehicle {vehicle.vehicle\_id} (End)").add\_to(m)  
            folium.PolyLine(locations=[(point[1], point[0]) for point in list(vehicle.route.coords)],  
                            color='blue', weight=2.5, opacity=1).add\_to(m)  
  
        # Save map as HTML file and open it in default web browser  
        m.save('simulation\_map.html')  
        webbrowser.open('simulation\_map.html')  
  
        # Visualization of data in graphs  
        plt.figure(figsize=(10, 5))  
        plt.subplot(1, 3, 1)  
        plt.bar(df['Vehicle ID'], df['Distance Traveled'])  
        plt.xlabel('Vehicle ID')  
        plt.ylabel('Distance Traveled (km)')  
        plt.title('Distance Traveled by Vehicles')  
  
        plt.subplot(1, 3, 2)  
        plt.bar(df['Vehicle ID'], df['Toll Incurred'])  
        plt.xlabel('Vehicle ID')  
        plt.ylabel('Toll Incurred (Rupees)')  
        plt.title('Toll Incurred by Vehicles')  
  
        plt.subplot(1, 3, 3)  
        plt.bar(df['Vehicle ID'], df['Remaining Balance'])  
        plt.xlabel('Vehicle ID')  
        plt.ylabel('Remaining Balance (Rupees)')  
        plt.title('Remaining Balance for Users')  
  
        plt.tight\_layout()  
        plt.show()  
  
        # Show summary message  
        show\_summary\_message(df)  
  
# Run the simulation  
SimulationManager.run\_simulation()

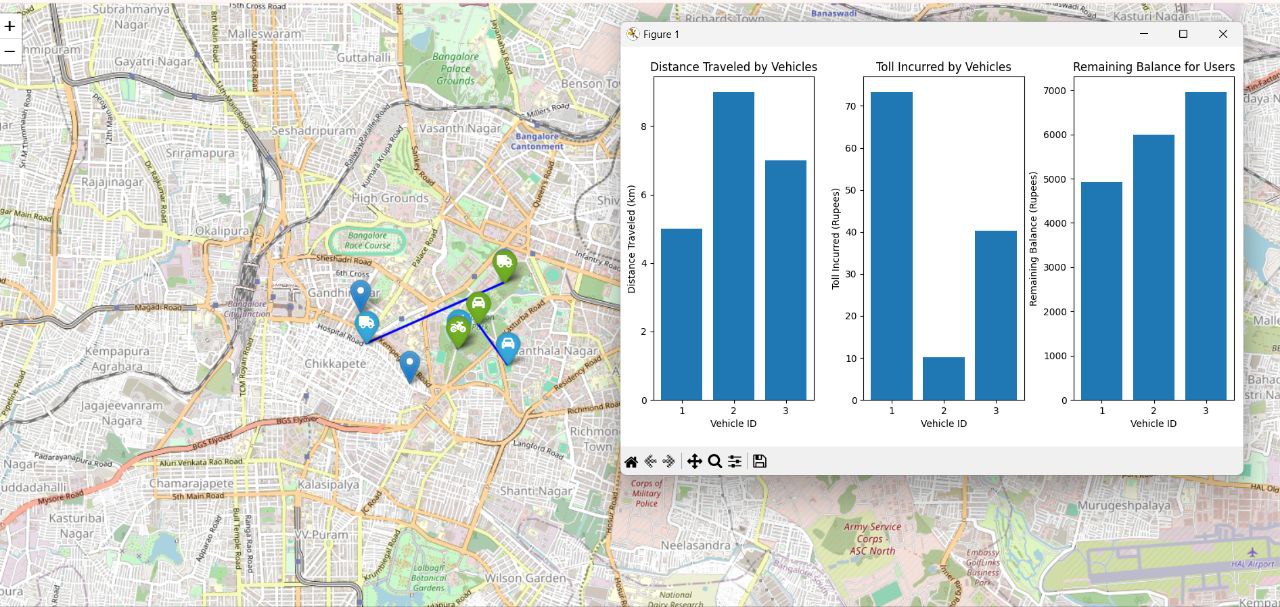
**Output**

* After running the code .
* It will give popup message alert of toll charges.

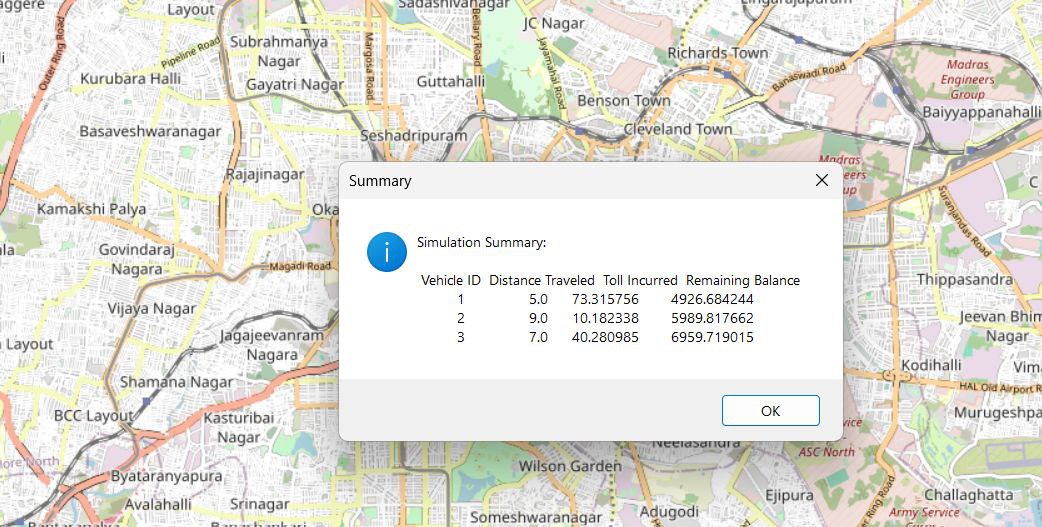


* It shows all the tolls and vehicles in the map
* It represents in as graph of distance, toll incured and balance.

****

****

* It gives message alert of summary

****

**Results**:

Result model link : <https://akshitha173.github.io/gps_toll_system/>

Github link : <https://github.com/akshitha173/gps_toll_system.git>