PROBLEM STATEMENT 06:

GPS Toll based System Simulation using Python

Team: Logic loom

Mentor: Dr. Anju Pratap

Member 1: Hima Rose George

Member 2: Anjitha Anil

Member 3: Ashwini Anil

Abstract

This project involves simulating a toll booth system to analyze vehicle movements and toll payments using discrete-event simulation. The simulation is implemented with SimPy, GeoPandas, and Geopy libraries to provide a comprehensive analysis and visualization of vehicle behaviors in relation to toll booth interactions.

The toll booth is geographically located at a specific coordinate, and a predefined toll amount is set for vehicles within a 50 km radius. The project simulates three types of vehicles—Truck, Car, and Bus—each starting from different locations with initial account balances. The simulation tracks their movements, checks their proximity to the toll booth, and deducts the toll amount from their balances if applicable.

Key steps in the simulation include:

1. Defining the toll booth location and toll amount.
2. Initializing vehicle data with their types, locations, and account balances.
3. Creating a simulation environment and defining vehicle processes to simulate toll booth interactions.
4. Calculating distances from the toll booth and determining toll applicability.
5. Updating vehicle balances based on toll payments and simulating passage through the toll booth.
6. Visualizing the locations of the toll booth and vehicles using GeoPandas.

The results of the simulation demonstrate how vehicles with different starting locations and balances interact with the toll booth, providing insights into toll payment behaviors and financial implications. The visualization further aids in understanding the geographic aspects of the simulation, showcasing the spatial distribution of vehicles and the toll booth. This project highlights the practical application of simulation and geographic data analysis in managing and studying toll booth systems.

Introduction

The GPS-based Toll Collection System Simulation project provides a framework for simulating toll collection operations, integrating various Python libraries to create a dynamic environment. It aims to give students a practical understanding of toll collection systems through key concepts like vehicle movement, toll zone definitions, and payment processing.

The core components of the simulation include predefined routes marked by GPS coordinates, the identification of toll zones, distance calculations, toll charge computations, and payment processing. By utilizing SimPy for event-driven simulation, GeoPandas and shapely for geospatial analysis, GeoPy for distance calculations, pandas for data handling, and Matplotlib and Folium for visualization, the project combines data analytics and visualization techniques.

The project simulates vehicles such as cars, buses, and trucks moving through toll zones, calculating distances and determining charges based on predefined rates. These toll charges are deducted from simulated user accounts, mimicking real-world transactions. SimPy manages the event-driven simulation, while GeoPandas and shapely define toll zones and check intersections. GeoPy ensures precise distance calculations between GPS points. Data management and analysis are handled using pandas, with visualizations provided by Matplotlib and Folium.

The simulation workflow includes setting up the environment with a road network and toll zones using geospatial data, and initializing vehicles with starting points and destinations. As vehicles move, SimPy updates their GPS coordinates, and shapely and GeoPandas check for toll zone intersections. Charges are calculated based on travel distance or fixed fees for zone crossings and deducted from user accounts.

The project also generates reports on vehicle movements and toll collections, offering insights into the performance and dynamics of toll collection systems. The main challenges involve accurately simulating vehicle movements, optimizing performance for numerous transactions, and integrating diverse components for a cohesive operation.

The project also generates detailed reports on vehicle movements and toll collections, offering valuable insights into system performance and operational dynamics. Despite challenges in accuracy, optimization, and integration, it serves as a comprehensive educational tool for understanding toll collection systems.

Methodology

1. Setup Environment

- This section sets up the geospatial data for the road network, toll zones, and vehicle locations:
- Define Road Network and Toll Zones Using Geospatial Coordinates
- You defined the toll booth location and vehicle starting points using geospatial coordinates.

2. Simulate Vehicle Movement

- Simulation Environment:
- Created a simulation environment using SimPy.
- Defined a vehicle function to simulate vehicle movements and toll payments.
- Vehicles are simulated to arrive at different times.

3. Detect Toll Zone Crossings

- The distance to the toll booth is calculated using the geodesic distance from the geopy library.
- Vehicles within a certain range (50 km) are subject to toll charges.

4. Calculate Toll Charges

- Toll charges are determined based on the distance to the toll booth.
- Vehicles pay the toll if their balance is sufficient.

5. Simulate Payments

- Toll charges are deducted from the vehicle's balance.
- The remaining balance or lack of funds is printed for each vehicle.

6. Analytics and Reporting

- Geospatial data for vehicle routes and toll booth location are plotted using GeoPandas and Matplotlib.
- Different colours are used for each vehicle's route for clarity

Literature Review

Developing countries like India needs a significant improvement in infrastructure such as Roads or Highways. Construction of these highways is a costly affair, which can't be invested by the government alone. Normally Public private partnerships are made to construct such a huge project. The money spent on these projects can be regained by collecting toll from the passengers who use the roads. The toll collection system, especially in India faces some problems such as long queue lines, escaping from toll plazas etc. These systems can service only 300 vehicles per hour, and if more than that number of vehicles arrive at that plaza, server traffic jams may occur. To solve this, we are proposing to create geo-fences using GPS by giving latitude and longitude of the corner of the toll plaza. By comparing the position of the vehicle and toll plaza, the owner of the vehicle can be charged from the account.

The necessity for vehicles to stop or slow down for toll fee payment results in traffic congestion and reduces fuel efficiency. Hence, a system that enables road users to pay the toll fees without stopping or slowing down was proposed and developed. Hardware and software designs were carried out to develop a Global Positioning System (GPS)-based highway toll collection system. This system was developed using a Raspberry Pi 2 microcontroller. Different modules such as GPS module, Liquid Crystal Display (LCD) module, speaker, wireless Wi-Fi router modem and wireless Wi-Fi adapter were incorporated and integrated with the microcontroller to perform a few specific functions. In general, the system utilized GPS coordinates to detect whether a vehicle passed through predefined locations in the database and the travel details were recorded. The Raspberry Pi 2 microcontroller was configured as a personal cloud server to allow online access of travel logs. This developed system presents a different approach for highway toll collection which eliminates travel delays and construction of expensive gantries or toll booths.

The service time and time headway of the vehicles are used to define the equivalency factor of different vehicle classes at the toll plaza. Both the service time and time headway are point measures and do not account for the system delay (time difference when a vehicle enters the queue and leaves the tollbooth) caused to the vehicle. The present study aims to quantify the system delay incurred to the vehicles at electronic toll collection (ETC) lanes under mixed traffic conditions using a microsimulation approach. Field data collected from one toll plaza located on a National Highway are used for simulation model generation. A new terminology called system delay-based

toll equivalency factor (DTEF) is introduced to convert the different vehicle classes into equivalent passenger cars. The DTEF variation for different approach volumes and heavy commercial vehicle (HCV) compositions were checked at different ETC penetration levels. A total of 288 scenarios have been worked out, and simulation runs have been made for all such scenarios to obtain the DTEF values. The average DTEF value of HCV was obtained as 2.20. Further, it is found that with an increase in approach volume and HCV share in the traffic stream, the DTEF value increases. The outcome of the present study will be useful to field practitioners and engineers to determine the capacity in equivalent DTEF/hr and level of service of a toll plaza in terms of system delay.

This paper describes about the modelling concept of traffic flow at expressway toll plaza. Nowadays, Electric Toll Collection (ETC) system has been rapidly popularized in Japan, as shown in Figure 1. Many road authorities expect that more than 80 percent of running vehicles will equip on-board ETC units within a couple of years. Under such high proportion of ETC vehicles, the road authorities have to revise the design of toll plaza in order to prevent undesirable near-miss by increasing the number of ETC exclusive lanes, modifying the geometry or enlarging the area of the plaza. From the sake of road authorities, the traffic simulation model which can consider the interaction between traffic conditions and the design of toll plaza is to be developed. Traffic conditions should be evaluated in terms of not only LOS but also in safety. As there was less knowledge of the driving behavior of ETC vehicles in toll plaza, we had to start with the precise survey of vehicle trajectories in the plaza in advance of the modelling.

Congestion is a complicated problem that often occurs in big cities. Congestion occurs not only on arterial roads but also on toll roads. The impact of this traffic jam is extraordinary. According to data from the Ministry of National Development Planning of the Republic of Indonesia, in the State capital of Indonesia alone, the value of losses due to congestion every year reaches 65 trillion. In addition, a study found the fact that congestion also increases the risk of heart failure, high blood pressure, and coronary artery disease. Since 2017 the Indonesian Government through the regulation of the Ministry of Public Works has required electronic payment transactions on toll roads. The media used is an electronic toll card. By using this media, the vehicle must really stop at the Automatic Toll Gate to be able to tap the electronic toll card. But the relatively long reading time often actually adds to the traffic jam. This study aims to create a system with a GPSbased mobile application to reduce the toll queue payment at the Automatic Toll Gate through simulations using Vissim software. With Vissim software, an

analysis of six simulation scenarios was carried out. The simulation results show that a GPS-based mobile application has succeeded in reducing the toll payment queue.

Reference

While there may not be a direct reference specifically detailing a GPS toll-based simulation using Python, you can find relevant information from various sources that cover the components and technologies involved in such a system.

1. **Python Programming and Simulation**:
    - "Python for Data Analysis" by Wes McKinney can be useful for understanding data manipulation and analysis, which is crucial for handling GPS data.
    - "Automate the Boring Stuff with Python" by Al Sweigart provides practical examples of automating tasks using Python, which can be adapted for toll system simulations.
2. **Research Papers and Case Studies**:
    - Research papers on GPS-based toll collection systems and simulations. Databases like IEEE Xplore or Google Scholar can provide access to relevant papers.
    - Case studies on smart toll systems or GPS-based tolling implementations published by transportation authorities or research institutions.
3. **Open-Source Projects and Code Repositories**:
    - GitHub repositories related to GPS tracking and toll systems. Searching for terms like "GPS toll system Python" on GitHub can yield useful results. For example, repositories that show GPS tracking implementations using Python.
    - Open-source projects related to intelligent transportation systems or automated toll collection.
4. **Government and Industry Reports**:
    - Reports by transportation departments or industry analyses on automated tolling and GPS-based systems can provide real-world context and data.
    - Documents and publications from agencies like the Federal Highway Administration (FHWA) or the International Bridge, Tunnel and Turnpike Association (IBTTA).

5. Literature review

Nagothu, S.K., 2016, April. Automated toll collection system using GPS and GPRS. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (pp. 0651-0653). IEEE.

Tan, J.Y., Ker, P.J., Mani, D. and Arumugam, P., 2016, November. Development of a GPS-based highway toll collection system. In *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)* (pp. 125-128). IEEE.

Bari, C.S., Chandra, S. and Dhamaniya, A., 2023. Estimation of system delay-based toll equivalency factors at toll plazas using simulation. *International Journal of Transportation Science and Technology*, *12*(3), pp.822-835.

Shitama, T., Horiguchi, R., Akahane, H. and Xing, J., 2006, July. Traffic simulation for expressway toll plaza based on successive vehicle tracking data. In *Proceeding of international symposium of transport simulation* (pp. 1-21).

Dismantoro, D., Pratomo, I. and Sumpeno, S., 2020, November. Minimizing Toll Payment Queue using GPS-Based Mobile Applications. In *2020 Fifth International Conference on Informatics and Computing (ICIC)* (pp. 1-7). IEEE.

Results and Discussion

Results

The simulation was conducted using SimPy, GeoPandas, and Geopy to model the interactions between various vehicles and a toll booth system. The key results are summarized below:

1. **Distance Calculation and Toll Applicability**:
    - Distances from each vehicle's starting location to the toll booth were calculated using the geodesic method from Geopy.
    - Vehicles within a 50 km radius of the toll booth were required to pay the toll amount.
2. **Toll Payment and Balance Update**:
    - Vehicles with sufficient balance at the time of passing the toll booth successfully paid the toll, and their remaining balances were updated accordingly.
    - Vehicles without sufficient balance could not pay the toll, which was indicated in the simulation logs.
3. **Simulation Logs**:
    - Detailed logs provided information on each vehicle's initial balance, distance to the toll booth, toll amount incurred, and remaining balance after the toll payment.
    - Logs also recorded the passage of vehicles through the toll booth, marking whether the toll was successfully paid.
4. **Geospatial Visualization**:
    - A Geodata Frame was created to visualize the geographic locations of the toll booth and the vehicles.
    - The visualization displayed the spatial distribution of the vehicles in relation to the toll booth, highlighting the vehicles within the toll-applicable range.

Discussion

The results of the simulation provide several insights into the toll booth system and its interaction with different vehicles:

1. **Toll Collection Efficiency**:
    - The simulation demonstrated the system's efficiency in identifying vehicles within the toll-applicable range and processing toll payments based on their account balances.
    - This efficiency can be leveraged to optimize real-world toll collection processes by understanding vehicle movement patterns and financial readiness for toll payments.
2. **Financial Management**:
    - The simulation highlighted the importance of maintaining sufficient balances in vehicle accounts to ensure uninterrupted transit through toll booths.
    - Transport companies can use such simulations to better manage their financial planning, ensuring that their fleet is adequately funded to avoid delays and penalties.
3. **Geospatial Analysis**:
    - The geospatial visualization provided a clear representation of vehicle positions and their interactions with the toll booth.
    - This analysis is crucial for planning toll booth placements, understanding traffic flow, and identifying high-traffic areas where toll booths could be strategically placed to maximize revenue.
4. **Simulation Utility**:
    - The use of SimPy for discrete-event simulation proved effective in modeling real-world scenarios involving multiple entities and events.
    - The integration of GeoPandas and Geopy for geographic data handling and distance calculation added depth to the simulation, making it a robust tool for transport and logistics analysis.

Conclusion

The GPS toll-based system simulation using Python demonstrates several significant advantages:

1. **Efficiency and Automation:** The system automates toll collection, eliminating the need for physical toll booths. Vehicles are charged based on their GPS-tracked routes, which significantly reduces traffic congestion typically caused by manual toll collection processes.

2. **Accuracy:** GPS technology allows for precise location tracking. This accuracy ensures that drivers are charged appropriately based on the actual distance they travel, promoting fairness and reducing discrepancies in toll collection.

3. **User Convenience:** Drivers benefit from a seamless experience, as they do not need to stop at toll booths. This not only saves time but also enhances the overall travel experience.

4. **Cost Reduction:** The reduction in physical infrastructure and personnel requirements leads to lower operational costs. Maintenance costs for toll booths are also eliminated.

5. **Scalability**: The simulation, built using Python, demonstrates that the system can be scaled to accommodate various road networks and tolling schemes. This adaptability makes it suitable for different geographical regions and traffic conditions.

6. Environmental Benefits: By reducing the need for vehicles to stop and start at toll booths, the system can lead to smoother traffic flow, decreased fuel consumption, and lower vehicle emissions, contributing to a reduction in the overall environmental footprint of road transport.

In summary, the project showcases the potential of integrating GPS technology with toll collection systems to create a more efficient, user-friendly, and cost-effective solution

Future Scope

Several areas can be explored to further enhance the GPS toll-based system:

1. **Real-Time Data Integration:** Integrating real-time traffic data can optimize toll rates based on current traffic conditions. For example, dynamic pricing could be implemented, where toll rates vary depending on traffic congestion levels, encouraging off-peak travel and reducing congestion.

2. **Security Enhancements:** To ensure the integrity of the system, robust security measures must be implemented. This includes protections against GPS spoofing, encryption of data transmissions, and secure storage of user information.

3. **Integration with Payment Systems:** Seamless integration with various digital payment platforms (e.g., mobile wallets, credit/debit cards, bank transfers) can enhance user convenience. Automated billing and notifications can be implemented to keep users informed of their charges.

4. **Vehicle Classification:** The system can be improved to automatically classify different types of vehicles (e.g., cars, trucks, motorcycles) using machine learning or sensor data. Different toll rates can then be applied based on the vehicle type, reflecting the wear and tear caused by different vehicles.

5. **Multi-Lane and Multi-Road Support:** Extending the simulation to support complex road networks, including highways, urban roads, and rural roads, can make the system more versatile. This involves handling multiple lanes, varying speed limits, and different types of toll roads.

6. **User Feedback and Optimization:** Implementing a feedback mechanism allows users to report issues and suggest improvements. Continuous monitoring and analysis of system performance can help in making iterative enhancements, ensuring the system remains efficient and user-friendly.

7. **Policy and Regulatory Compliance:** The system must comply with local and international regulations regarding toll collection, data privacy, and transportation. This includes adhering to standards set by transportation authorities and ensuring user data is handled responsibly.

8. **Environmental Impact Analysis:** Assessing the environmental benefits of the system can provide valuable insights. This includes analyzing fuel savings and reduced emissions due to decreased idling times at toll booths and smoother traffic flow. These insights can be used to promote the system as an environmentally friendly alternative.

By addressing these future scope areas, the GPS toll-based system can evolve into a comprehensive, robust, and widely adopted solution for modern toll collection. This will not only improve efficiency and user experience but also contribute to better traffic management and environmental sustainability.

Acknowledgment

I would like to express my sincere gratitude to ChatGPT for providing invaluable assistance in the development of the GPS toll-based system using Python. The guidance and support received were instrumental in overcoming various challenges and enhancing the overall quality of the project. Your expertise and responsiveness were greatly appreciated, and this project would not have been as successful without your contributions.

Thank you, ChatGPT, for your support and dedication

Materials and resources

https://github.com/himarg22/gpstollsystem.git

Appendix

```
import simpy

import geopandas as gpd

from geopy.distance import geodesic

from shapely.geometry import Point, LineString

import matplotlib.pyplot as plt


# Define the new toll booth location

toll_booth_location = (9.4981, 76.3388) # Alappuzha, Kerala


# Define vehicle locations with (latitude, longitude), initial account
balances, base toll amounts, and names

vehicle_data = [

    {'type': 'Truck', 'name': 'Kottayam', 'location': (9.5916, 76.5221),
'balance': 10000.0, 'destination': (9.9312, 76.2673), 'base_toll': 20.0,
'start_time': 6 * 60},

    {'type': 'Car', 'name': 'Changanassery', 'location': (9.4420, 76.5363),
'balance': 5000.0, 'destination': (9.9833, 76.6150), 'base_toll': 5.0,
'start_time': 11 * 60},

    {'type': 'Bus', 'name': 'Puthupally', 'location': (9.5674, 76.6310),
'balance': 7500.0, 'destination': (9.9815, 76.5735), 'base_toll': 10.0,
'start_time': 8 * 60}

]


# Define a GeoDataFrame for vehicle starting points, toll booth, and
destinations

gdf_points = gpd.GeoDataFrame(

    {

        'id': ['Toll Booth', 'Truck Start', 'Car Start', 'Bus Start',
'Truck End', 'Car End', 'Bus End'],

        'geometry': gpd.points_from_xy(

            [toll_booth_location[1], vehicle_data[0]['location'][1],
vehicle_data[1]['location'][1], vehicle_data[2]['location'][1],

             vehicle_data[0] ['destination'][1], vehicle_data[1]
['destination'][1], vehicle_data[2] ['destination'][1]],

            [toll_booth_location[0], vehicle_data[0]['location'][0],
vehicle_data[1]['location'][0], vehicle_data[2]['location'][0],
```

```
            vehicle_data[0] ['destination'][0], vehicle_data[1]
['destination'][0], vehicle_data[2] ['destination'][0]]
        )
    }
)
# Define a GeoDataFrame for routes from vehicles to the toll booth and to
their destinations
gdf_routes = gpd.GeoDataFrame(
    {
        'id': ['Truck Route', 'Car Route', 'Bus Route'],
        'geometry': [
            LineString([Point(vehicle_data[0]['location'][1],
vehicle_data[0]['location'][0]), Point(toll_booth_location[1],
toll_booth_location[0]),
                        Point(vehicle_data[0] ['destination'][1],
vehicle_data[0] ['destination'][0])]),
            LineString([Point(vehicle_data[1] ['location'][1],
vehicle_data[1] ['location'][0]), Point(toll_booth_location[1],
toll_booth_location[0]),
                        Point(vehicle_data[1] ['destination'][1],
vehicle_data[1] ['destination'][0])]),
            LineString([Point(vehicle_data[2] ['location'][1],
vehicle_data[2] ['location'][0]), Point(toll_booth_location[1],
toll_booth_location[0]),
                        Point(vehicle_data[2] ['destination'][1],
vehicle_data[2] ['destination'][0])])
        ]
    }
)
# Define congestion level that affects toll prices
congestion_level = 1.5 # 1.0 = normal, > 1.0 = high congestion, < 1.0 = low
congestion
# Define dynamic toll pricing based on time of day and congestion
def get_dynamic_toll(base_toll, vehicle_type, time_of_day,
congestion_level):
    # Peak hours are 8-10 AM and 5-7 PM
    if 8 <= time_of_day < 10 or 17 <= time_of_day < 19:
        peak_factor = 1.5
    else:
        peak_factor = 1.0
```

```
 # Different toll rates for different vehicle types
    vehicle_factor = 1.0
    if vehicle_type == 'Truck':
        vehicle_factor = 2.0
    elif vehicle_type == 'Bus':
        vehicle_factor = 1.5

    toll = base_toll * peak_factor * vehicle_factor * congestion_level

    # Increase toll by additional percentages during specific time
intervals
    if 6.5 <= time_of_day < 7.5:
        toll *= 1.05
        print(f"Additional 5% toll applied for {vehicle_type} at
{time_of_day:.2f} hours.")
    elif 8.5 <= time_of_day < 9.5:
        toll *= 1.08
        print(f"Additional 8% toll applied for {vehicle_type} at
{time_of_day:.2f} hours.")
    elif 11.5 <= time_of_day < 12.5:
        toll *= 1.10
        print(f"Additional 10% toll applied for {vehicle_type} at
{time_of_day:.2f} hours.")

    return toll
# Define a function to convert time in hours to HH:MM format
def convert_to_hhmm(hours):
    hh = int(hours)
    mm = int ((hours - hh) * 60)
    return f"{hh:02}: {mm:02}"

# Define a function to simulate vehicles passing through a toll
def vehicle (env, vehicle_id, data):
    vehicle_type = data['type']
    name = data['name']
    location = data['location']
```

```
    balance = data['balance']
    base_toll = data['base_toll']
    destination = data['destination']
    start_time = data['start_time']

    yield env.timeout(start_time)  # Start the journey at the specified time
    start_time_hr = env.now / 60  # Convert start time to hours
    print(f"Time: {convert_to_hhmm(start_time_hr)} - {vehicle_type} from {name} ({location}) starts the journey with a balance of {balance:.2f}.")
    # Simulate arrival time based on distance and an average speed of 50 km/h
    distance_to_toll = geodesic(location, toll_booth_location).km
    travel_time = distance_to_toll / 50 * 60  # convert hours to minutes
    yield env.timeout(travel_time)

    time_of_day = (env.now / 60) % 24  # Get the current time in hours
    total_toll = get_dynamic_toll(base_toll, vehicle_type, time_of_day, congestion_level)
    if balance >= total_toll:
        balance -= total_toll
        print(f"Time: {convert_to_hhmm(env.now / 60)} - {vehicle_type} from {name} ({location}) reached the toll booth, incurred a toll of {total_toll:.2f}, and has a remaining balance of {balance:.2f}.")
    else:
        print(f"Time: {convert_to_hhmm(env.now / 60)} - {vehicle_type} from {name} ({location}) reached the toll booth but does not have enough balance to pay the toll.")

    # Simulate remaining travel to destination
    distance_to_destination = geodesic(toll_booth_location, destination).km
    travel_time_to_destination = distance_to_destination / 50 * 60
    yield env.timeout(travel_time_to_destination)
    end_time_hr = env.now / 60  # Convert end time to hours
    print(f"Time: {convert_to_hhmm(end_time_hr)} - {vehicle_type} has reached its destination at {destination}.")

# Define a simulation environment
env = simpy.Environment()
```

```
# Create processes for each vehicle

for i, data in enumerate(vehicle_data, start=1):

    env.process(vehicle(env, i, data))

# Run the simulation

env.run()


# Plot the locations and routes on a map with different colors for each
vehicle's route

fig, ax = plt.subplots(1, 1, figsize=(10, 10))

gdf_points.plot(ax=ax, color='blue', marker='o', markersize=50,
label='Locations')

gdf_routes[gdf_routes['id'] == 'Truck Route'].plot(ax=ax, color='orange',
linewidth=2, label='Truck Route')

gdf_routes[gdf_routes['id'] == 'Car Route'].plot(ax=ax, color='green',
linewidth=2, label='Car Route')

gdf_routes[gdf_routes['id'] == 'Bus Route'].plot(ax=ax, color='purple',
linewidth=2, label='Bus Route')


# Annotate the points with vehicle type and names

for x, y, label in zip(gdf_points.geometry.x, gdf_points.geometry.y,
gdf_points.id):

    ax.text(x, y, label, fontsize=12, ha='right')


plt.title("Vehicle Routes and Toll Booth in Kerala")

plt.legend()

plt.show()
```

Output

Time: 06:00 - Truck from Kottayam ((9.5916, 76.5221)) starts the journey with a balance of 10000.00.

Time: 06:27 - Truck from Kottayam ((9.5916, 76.5221)) reached the toll booth, incurred a toll of 60.00, and has a remaining balance of 9940.00.

Time: 07:25 - Truck has reached its destination at (9.9312, 76.2673).

Time: 08:00 - Bus from Puthupally ((9.5674, 76.631)) starts the journey with a balance of 7500.00.

Additional 8% toll applied for Bus at 8.66 hours.

Time: 08:39 - Bus from Puthupally ((9.5674, 76.631)) reached the toll booth, incurred a toll of 36.45, and has a remaining balance of 7463.55.

Time: 09:50 - Bus has reached its destination at (9.9815, 76.5735).

Time: 11:00 - Car from Changanassery ((9.442, 76.5363)) starts the journey with a balance of 5000.00.

Time: 11:27 - Car from Changanassery ((9.442, 76.5363)) reached the toll booth, incurred a toll of 7.50, and has a remaining balance of 4992.50.

Time: 12:41 - Car has reached its destination at (9.9833, 76.615).