GPS TOLL BASED SYSTEM

A PROJECT REPORT

Submitted by

DHANUSH RANGA G [Reg No: RA2211003010213]

MAANAS SOMANATHAN [Reg No: RA2211003010185]

ALEN GEORGE [Reg No: RA2211003010235]

Under the Guidance of

Dr. KARTHIKEYAN M

Assistant Professor, Department of Computing Technologies

In partial fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR -603203

JULY 2024

ABSTRACT

The aim of this project is to build a practical and effective Python-based simulation of GPS-based toll collection system. This involves simulating vehicles traveling along predetermined paths, detecting when they enter toll zones, and determining the toll amount based on factors such as distance or type of toll zone. The program will handle payments and produce reports for performance.

In order for us to achieve this, we will be using several Python libraries: SimPy, which coordinates vehicle movement and interactions; Geopandas and Shapely that define tolls zones and determine entry into these zones by vehicles; Geopy that helps calculate distances between GPS points; Pandas that manages data; matplotlib, folium for visualizing the results.

This simulation process involves setting up road networks with toll zones, creating vehicles with starting points as well as destinations then simulating their movements. The system will compute the fee when a car enters a payment area and manage payments. As a result, the simulation will generate reports showing how vehicles moved about within the highway network, collected revenue through charging fees and how my system was performing.

The main challenges here are accurately tracking vehicle movements including entries into its toll zone or efficiently managing high numbers of cars in addition to ensuring data confidentiality. If implemented successfully this could demonstrate own merits of GPS technology in terms of dynamic pricing that allows immediate deductions.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 GPS-based Toll Collection System: An Overview

The traditional toll collection system, characterized by manual toll booths, has faced significant challenges such as traffic congestion, toll evasion, and environmental concerns. To address these issues, GPS-based toll collection systems have emerged as a promising alternative. By leveraging the precision and ubiquity of GPS technology, these systems offer the potential for efficient, accurate, and user-friendly toll collection.

1.2 System Architecture and Components

The GPS-based toll collection system is designed to efficiently track vehicle movements, calculate toll charges based on predefined zones, and facilitate secure payment processing. Below is an outline of the core components of the system:

1. Vehicle Tracking System

    Purpose: To monitor and record the real-time location of vehicles.

    Components:

    GPS Devices: Installed on vehicles to continuously transmit their location data.

    Tracking Software: Collects GPS data and maps vehicle movements in real-time.

2. Toll Zone Definition

    Purpose: To establish specific geographic areas where tolls are applicable.

    Components:

    Geospatial Data: Defines the boundaries of toll zones using geographical coordinates.

    GeoJSON/Shape Files: Formats for storing and visualizing the toll zone boundaries.

3. Distance Calculation

    Purpose: To determine the distance traveled by a vehicle within the defined toll zones.

Components:

Road Network Graph: A graph representation of the road network (nodes and edges).

Shortest Path Algorithms: Used to calculate the shortest path between two points on the road network, considering the road distance.

4. Toll Calculation

Purpose: To compute the toll charges based on the distance traveled, vehicle type, and time.

Components:

Toll Rates Database: Contains the rates per unit distance for different vehicle types and time slots.

Calculation Algorithms: Use distance data and rates to compute the total toll charge for each journey.

5. Payment Processing

Purpose: To handle the secure transaction of toll payments.

Components:

Payment Gateway Integration: Connects with payment gateways to process transactions.

User Accounts: Maintain balance and transaction history for each user.

Billing System: Generates bills and receipts for toll charges.

6. Data Management

Purpose: To store and process all relevant data related to vehicles, toll zones, and payments.

Components:

Database Systems: Relational or NoSQL databases to store vehicle data, toll rates, payment transactions, and historical journey data.

Data Processing Tools: ETL (Extract, Transform, Load) tools for data manipulation and analysis.

Data Analytics: Tools and platforms for analyzing travel patterns, toll revenue, and system performance.

Integration of Components:

1. Real-Time Vehicle Tracking:
   - GPS devices on vehicles send location data to the tracking system.
   - The tracking system maps this data onto the road network graph.
2. Toll Zone Entry and Exit Detection:
   - As vehicles move, their positions are continuously checked against the toll zone boundaries.
   - Entry and exit events are logged for each toll zone.
3. Distance and Toll Calculation:
   - When a vehicle exits a toll zone, the distance traveled within the zone is calculated using the road network graph.
   - The toll charge is computed based on the distance, applicable rates, and vehicle type.
4. Payment Processing:
   - The computed toll charge is debited from the vehicle owner’s account.
   - Transactions are processed through a secure payment gateway, and the user is notified of the deduction.
5. Data Storage and Analysis:
   - All vehicle movements, toll calculations, and payment transactions are stored in a central database.
   - Data analytics tools process this data to generate insights and reports for system management and optimization.

By integrating these components, the GPS-based toll collection system ensures accurate tracking, efficient toll calculation, and secure payment processing, thereby improving the overall efficiency and user experience of toll collection operations

1.3 Objectives of the Project

The primary objectives of this project are to develop and evaluate a comprehensive GPS-based toll collection system. The specific goals are as follows:

1. Develop a Simulation Model of a GPS-Based Toll Collection System

   Goal: To create a detailed and functional simulation of a toll collection system that uses GPS technology for vehicle tracking and toll calculation.

Tasks:

Design and implement the system architecture.

Simulate vehicle movements and interactions with toll zones.

Ensure accurate and real-time tracking of vehicles.

2. Evaluate the System's Performance Under Various Traffic Conditions and Toll Rate Scenarios

Goal: To assess how the system performs when subjected to different levels of traffic density and varying toll rates.

Tasks:

Simulate different traffic scenarios, including peak and off-peak hours.

Apply different toll rate structures to evaluate their impact on system performance and user behavior.

Measure system metrics such as transaction throughput, accuracy, and user satisfaction.

3. Analyze the Impact of GPS Accuracy and System Latency on Toll Calculation and Payment Processing

Goal: To understand how the accuracy of GPS data and system processing delays affect the overall performance and reliability of the toll collection system.

Tasks:

Investigate the effects of different levels of GPS accuracy on vehicle tracking and toll calculation.

Evaluate how system latency influences real-time tracking, toll zone detection, and payment processing.

Identify critical thresholds for GPS accuracy and system latency to ensure optimal performance.

4. Explore Potential Optimizations and Improvements for the System

Goal: To identify areas where the system can be enhanced for better efficiency, reliability, and user experience.

Tasks:

Analyze simulation results to pinpoint performance bottlenecks and inefficiencies.

Propose and implement optimizations in the system architecture, algorithms, and processes.

Test the effectiveness of the proposed improvements through further simulations and evaluations.

By achieving these objectives, the project aims to provide a robust and efficient GPS-based toll collection system that can be reliably deployed in real-world scenarios, ensuring accurate toll calculation and an efficient payment processing.

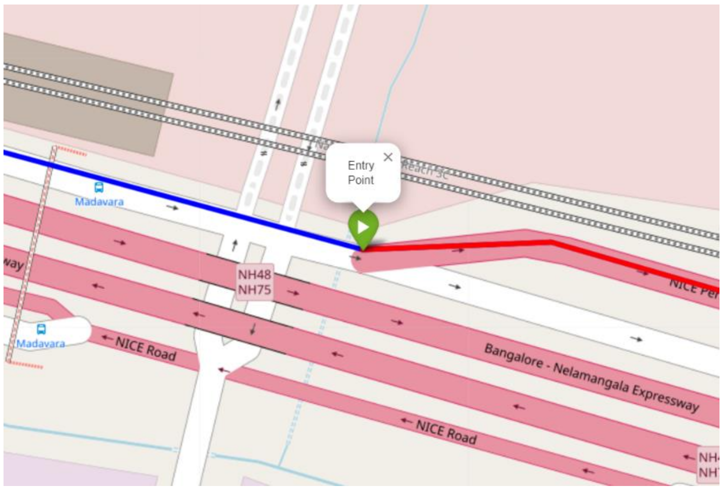
1.4 Scope and Limitations

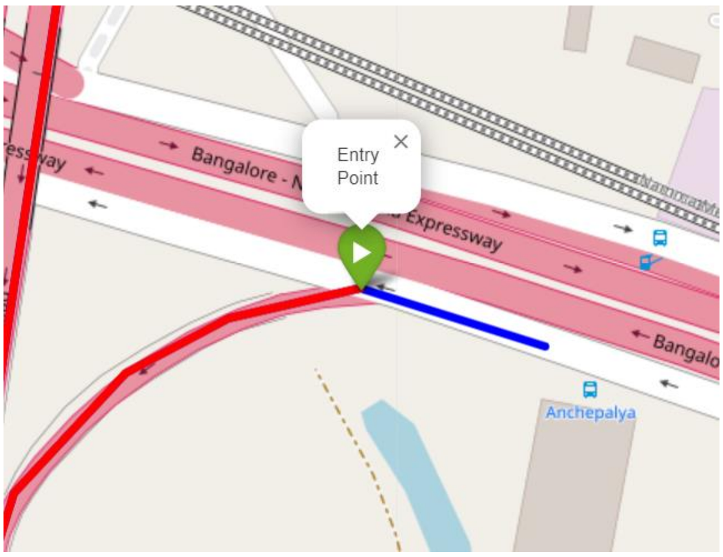
The scope and limitations of the GPS-based toll collection system project outline the boundaries, assumptions, and constraints within which the project operates. This section clarifies what the project will cover and any potential limitations that need to be considered.

Scope:

1. System Components

- Vehicle Tracking System**:** Implementation of a GPS-based tracking mechanism to monitor vehicle movements in real-time.
- Toll Zone Definition**:** Establishment of geographic boundaries for toll zones using geospatial data.





- Distance and Toll Calculation**:** Development of algorithms to calculate the distance traveled within toll zones and compute toll charges based on predefined rates.
- Payment Processing**:** Integration with payment gateways to handle secure toll transactions.
- Data Management**:** Storage and processing of vehicle, toll, and payment data for analysis and reporting.

2. Simulation Model

- Vehicle Movement Simulation: Creating a virtual environment to simulate vehicle movements through defined toll zones.
- Traffic Scenarios: Evaluating system performance under various traffic conditions, including peak and off-peak hours.
- Toll Rate Scenarios: Testing different toll rate structures to analyze their impact on system performance and user behavior.

3. Performance Evaluation

- System Metrics: Measuring key performance indicators such as transaction throughput, accuracy, and latency.
- Impact Analysis**:** Studying the effects of GPS accuracy and system latency on toll calculation and payment processing.

4. Optimization and Improvement

- System Enhancements: Identifying and implementing potential optimizations to improve system efficiency, reliability, and user experience.
- Validation: Testing the effectiveness of proposed improvements through additional simulations and evaluations.

Assumptions:

- GPS Accuracy: It is assumed that the GPS devices used for vehicle tracking provide sufficient accuracy for the purposes of toll zone detection and distance calculation.
- System Latency: The system's processing latency is assumed to be within

acceptable limits to ensure real-time tracking and payment processing.

- User Compliance: It is assumed that all vehicles are equipped with functioning GPS devices and that users comply with the system's requirements for toll payment.
- Stable Network Connectivity: Continuous and stable network connectivity is assumed for real-time data transmission between GPS devices, tracking systems, and payment gateways.

Limitations:

- GPS Signal Loss: Occasional loss of GPS signal or reduced accuracy in certain areas (e.g., tunnels, urban canyons) may affect tracking accuracy and toll calculation.
- System Latency: Delays in data processing and transmission could impact real-time toll zone detection and payment processing.
- Scalability: The simulation model may not fully represent the scalability challenges that could arise in a real-world deployment with a large number of vehicles.
- User Behavior: The simulation does not account for all possible variations in user behavior, such as deliberate evasion of toll zones or non-compliance with payment requirements.
- Infrastructure Variability: Differences in road infrastructure, traffic patterns, and toll zone configurations in different regions may affect the generalizability of the simulation results.

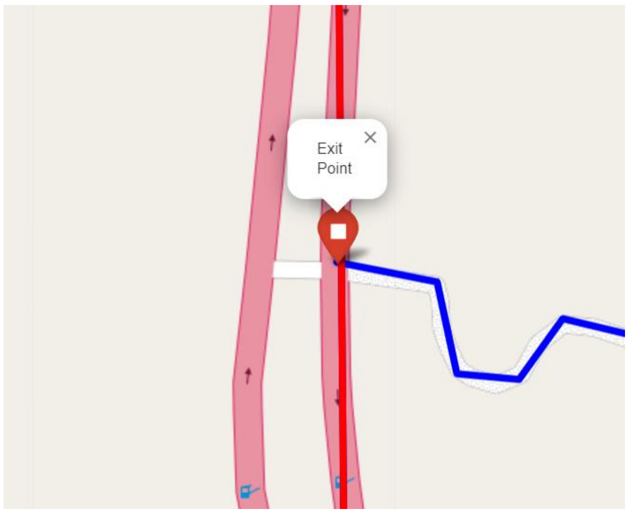
CHAPTER 2

SYSTEM ARCHITECTURE AND DESIGN

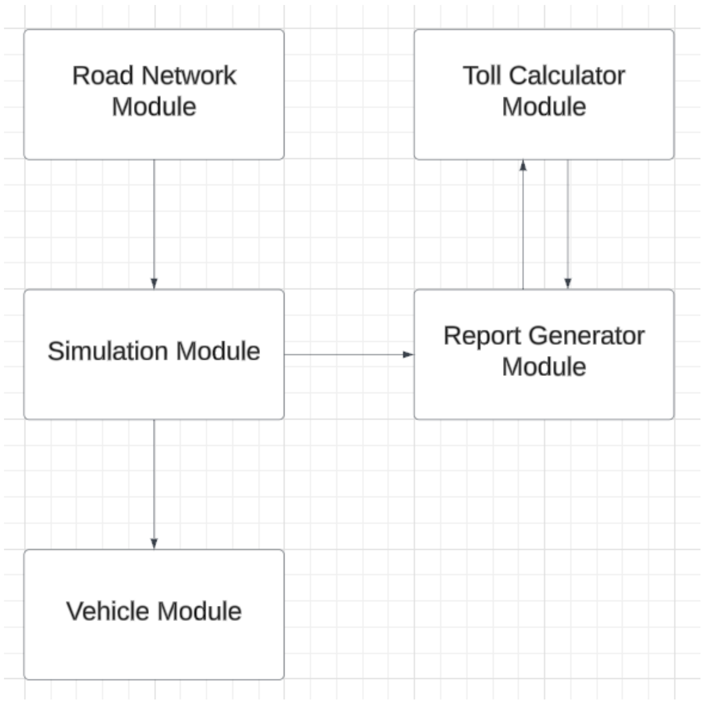
2.1 System Workflow

GPS-based toll collection system works upon a manually well-coordinated series of processes that ensure proper vehicle tracking by the system and calculation, as well as processing of payment against toll. The overall data and process flow of this system is mentioned below:

1. Vehicle Entry
    - OBU Activation: At the entry of the vehicle at the system, all the On-Board Units (OBU)installed in it initializes itself upon receiving co-ordinates from GPS Receiver.
    - Data Transmission: The OBU will transmit its current GPS coordinates to the central server after regular intervals, updating the vehicle's location continuously.
2. Toll Zone Detection
    - Zone boundary checking: Once the central server receives the GPS coordinates, it will be checked whether the location of that vehicle intersects with the pre-entered location for toll zone boundaries in the toll zone database.
    - Event Logging: As soon as the vehicle enters a toll zone, it will log the event and change the vehicle status to be inside the toll zone.



3. Toll Computation
    - Distance Measurement: By utilizing the road network graph, the centre server will compute the distance covered by a vehicle in its toll zone area. It runs the shortest path algorithms to find out the route taken.
    - Computation of Toll Amount: The system computes the toll amount depending upon the travelled distance for predefined rates that can be based on the type of vehicle or timing of the day. This calculation is done through the toll calculation engine.
4. Payment Processing
    - Account Management: The computed toll is deducted from the vehicle owner's available prepaid account balance. The system may alert the user or trigger a charge to a linked payment method if there are not enough funds in the account.
    - Trans- action Confirmation: After the payment is successful, the system confirms the transaction and updates the account balance accordingly.
5. Data Storage
    - Transaction Logging: All details of every transaction, such as the GPS coordinates, toll amount, payment mode, and timestamps, are recorded in this central server's database.
    - Data Analytics: These stored data shall be used for analytical purposes, mainly for report generation related to vehicle movement, toll revenue collection, and system performance.



2.2 Deployment and Integration

Integration of the GPS-Based Toll System

Integration of the GPS-based toll collection system with the other transportation systems and infrastructure could help in increasing functionality and user convenience, in addition to ensuring better efficiency. The following is how the unit can be integrated:

1. Traffic Management System
    - Real-Time Data Exchange: The toll system and the system managing the traffic can be integrated wherein the real-time data requirement would be exchanged by the former and the latter for integrating dynamic toll rates in accordance with the prevailing traffic.
    - Congestion Management: In this outlook, the system changes the price of the tolls or informs drivers about the congestion in some traffic zones in real-time conditions to induce the use of alternative routes.
2. Payment Gateways
    - Multiple Payment Options: It integrates various payment providers, such as credit cards, mobile wallets, and bank transfers, to provide a choice to the user for his/her preferred mode of payment.
    - Safe transactions: Security standard compliance (e.g., PCI DSS) in all payment processes to assure users' data security and the transactions' integrity.
3. Public Transport Systems
    - Fare Integration: Tie-ups with the public transport systems to offer the population integrated fare solutions, simplifying the toll and public transport fare payment process from a single account or card.
    - Discount Programs: The system should have discount programs for customers who transit the toll zones regularly and/or have used the toll roads and public transport systems.

Implementation Strategy

The strategy to implement the system would be planned in terms of hardware, software, and test and validation cycles.

Hardware Infrastructure:

- GPS Devices: Fit the vehicles with good GPS devices that would track the real-time location and relay the negatively to the central server.
- Servers: Numerous servers to implement data processing, storage, and application hosting; this could be cloud or on-premise, depending on the agencies' scalability requirement
- Communication Networks: The need for robust communication networks, either cellular or Wi-Fi, to guarantee a steady stream of data flow from vehicles back to the central server.

Software Deployment:

- Configuration: Deployment of the central server software and OBU software in suitable platforms to allow each component to work with each other frictionlessly.
- Configuration: Customizing the configuration of the system settings to meet the toll rates, vehicle classification, and payment gateway configurations, to meet operational requirements.
- Firmware Update: Define a mechanism for achieving periodic software update and patches considering security or other features of functionality, including remote update on OBU, if applicable.

Testing and Validation

- Quality Assurance: Comprehensive quality assurance checks to detect and correct any issues in the functionality of the system, including unit testing, integration testing, and system testing.
- Performance Testing: This kind of testing of the system under factors like immense volumes of traffic or different transactions in location simultaneously to guaranty the performing testing on the system to ensure its reliability and responsiveness.
- User Acceptance Testing: it allows the stakeholders and users to test and validate a system against the requirements in order to satisfy the users that the system meets their expectations in full deployment.

2.3 Security and Privacy

Implementation of a GPS-based toll collection system inherits several security and privacy issues. Most of those will deal with the protection of user data and ensure that none of this information is infringed upon or the integrity of the system violated.

The main considerations are summarized as follows:

1. Data Protection
    - Encryption: User sensitive information, such as vehicle and payment information, should be encrypted in-transit, using protocols such as HTTPS/TLS, and at rest, using financially sound and secure encryption standards to ward off unauthorized access.
    - Access Controls: Ensure sharing of critical information to only authorized persons; on that note, ensure role-based access and review of user permissions on a periodic basis.
2. System Security
    - Authentication and Authorization: Details strong authentication at both user ends and system administrators to prevent any unauthorized access.
    - Routine Security Audit: Plan for regular security testing and vulnerability scanning. This will test the potential risks or vulnerabilities that may be present within a system and shall realize mitigation actions.
3. Privacy Compliance
    - Data Minimization: Collect only such data that is relevant and necessary to run the toll collection system, thus upholding the Core Principal of Data Minimisation elaborated in multiple regulations like GDPR and CCPA.
    - User Consent: Inform users about the practice of data collection; obtain their consent before the collection or processing of their personal data.
    - Data Subject Rights: Exercise the various rights available under privacy regulations, particularly those rights exercisable in relation to access, rectification, or erasure of data.
4. Fraud Prevention
    - Real-time monitoring of transactions, flagging suspicious activity, including multiple accounts from a single device or location.
    - Secure payment processing: Integrate with well-known payment gateways that correspond to security standards that lower the risks of payment fraud.

CHAPTER 3
METHODOLOGY

This project outlines the basic details of a GPS application simulation for toll collection in Python. It determines tolls by distance covered within predefined zones, charging the user's account for obligation-incurred services.

Methodology

1. Problem Definition: The problem to be solved is the simulation of a GPS-based toll system that not only calculates tolls travelled within a defined zone but also deducts the charges from the account.
2. Requirements Analysis:
    - Simulate vehicle movement, toll zone detection, toll calculation, and payment processing and reporting
    - Accuracy of the system; Scalability; Security
3. System Design:
    - Modular architecture for vehicle simulation, toll zone detection, toll calculation, payment processing, and reporting.
    - Data flow: GPS coordinates, toll zone definitions, toll rates, user accounts → vehicle simulation, toll zone detection, toll calculation, payment processing → reports.
4. Implementation:
    - Set up the environment with geopandas, shapely, folium, simpy, osmnx, networkx, pandas, and geopy.
    - Definition of the road network and toll zones using geospatial data.
    - Simulate vehicle movement using SimPy; update GPS coordinates.
    - Detection of toll zone crossings using shapely and geopandas.
    - Calculate distance-based tolls within zones or fixed zone fees
    - Payment Simulation deducting Charges from User Accounts
    - Visualization using Folium
5. Unit, integration and system testing for accuracy and performance
6. Analysis and Reporting Collect data on vehicle movement, toll charges system performance and produce Reports

7. Documentation Project Documentation, Code, User guide
8. Evaluation and Improvement Evaluate System Performance, get feedback and optimize

Simulation Workflow

- Environment Setting and Road network / Toll zones.
- Vehicle object initialization should be done with starting points, destinations, and a user account.
- Simulation of the movement of vehicles should be done through the update of GPS coordinates.
- Toll zone crossing detection
- Distance/zone-based fees calculation
- Charging through deduction of charges from their accounts
- Data taking and analyzing
- Reporting generation

Problems and Their Solution

- Accuracy: High-precision coordinates, small step sizes, accurate toll zone detection algorithms
- Performance: Improvement in computation, efficient data structures, and handling of large datasets
- Complexity: Modular design, the principles of object-oriented programming.
- Security: Strong security measures concerning every user's data are to be implemented.

By following this methodology, one will end up with a comprehensive, efficient GPS-based toll collection system simulation.

CHAPTER 4
CODING AND TESTING

4.1 CODING

The code is designed to simulate vehicle movement through a road network with toll zones, calculate toll charges, and generate a report summarizing each vehicle's journey.

Step 1: Setup Environment

1. Imports and Libraries:

```
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point, LineString
import folium
import simpy
import osmnx as ox
import networkx as nx
```

   These imports include libraries for geographic data manipulation (geopandas, shapely), mapping (folium), simulation (simpy), and road network analysis (osmnx, networkx).

2. Define Road Network:

```
road_network = gpd.GeoDataFrame({...})
```

   A GeoDataFrame road_network is created, containing LineString geometries representing roads.

3. Define Toll Zones:

```
toll_zones = gpd.GeoDataFrame({...})
```

   A GeoDataFrame toll_zones is created, containing LineString geometries representing toll zones.

4. Visualize Road Network and Toll Zones:

```
m = folium.Map(...)
```

   The road network and toll zones are visualized on an interactive map using folium. Entry and exit points are marked on the map.

5. Download Road Network:

   G = ox.graph_from_point(...)

   The road network graph G is downloaded from OpenStreetMap using osmnx, centered around a specified point with a 6km radius.

Step 2: Simulate Vehicle Movement

1. Vehicle Class Definition:

   class Vehicle:

   def __init__(self, env, vehicle_id, start_location, end_location, toll_zones, rates, accounts, G):

   The Vehicle class is defined to simulate the movement of a vehicle, including attributes such as vehicle_id, start_location, end_location, toll_zones, rates, accounts, and G.
2. Calculate Road Distance:

   def calculate_road_distance(self):

   The calculate_road_distance method calculates the shortest path distance between the start and end locations using the road network graph G.
3. Move Method:

   def move(self):

   The move method simulates the vehicle's movement from the start location to the end location in small steps (step_size). During each step, it checks for intersections with toll zones.
4. Check Toll Zones:

   def check_toll_zones(self, prev_location):

   The check_toll_zones method checks if the vehicle's previous or current location intersects with any toll zone. If it does, it calculates the road distance and records the toll zone.
5. Run Method:

   def run(self):

   The run method orchestrates the vehicle's movement and processes the toll payment once the journey is complete.

6. Process Payment:
   ```
   def process_payment(self):
   ```
   The process_payment method calculates the total toll based on the distance traveled and deducts it from the vehicle's account.

Step 3: Simulation Initialization

1. Define Toll Rates:
   ```
   toll_rates = 0.003025
   ```
   A toll rate is defined.
2. Initialize User Accounts:
   ```
   user_accounts = {i: 100.0 for i in range(6)}
   ```
   User accounts are initialized with a balance of 100.0 for six vehicles.
3. Simulate Vehicle Movement:
   ```
   env = simpy.Environment()
   vehicles = [Vehicle(env, i, ...)]
   env.run(until=100)
   ```
   Vehicles are created with specific start and end locations. The simulation environment is run for 100 units of time.

Step 4: Generate Report

1. Collect Vehicle Data:
   ```
   vehicle_data = []
   for vehicle in vehicles:
   vehicle_data.append({...})
   ```
   Vehicle data is collected into a list of dictionaries.
2. Create DataFrame for Analysis:
   ```
   df = pd.DataFrame(vehicle_data)
   ```
   A DataFrame df is created for analysis.
3. Generate Report:
   ```
   print(df)
   ```
   The DataFrame is printed to display the results, including each vehicle's total toll, account balance, crossed toll zones, and total distance traveled on toll roads.

4.2 TESTING

• Unit Testing: Tests individual components for correctness.

• Integration Testing: Tests interactions between components.

• System Testing: Tests the complete system for compliance with requirements.

• Performance Testing: Evaluates system performance under various conditions.

• Security Testing: Assesses the system's security posture.

• User Acceptance Testing (UAT): Involves end-users in validating the system's functionality and usability.

CHAPTER 5
RESULTS AND ANALYSIS

Overview of the Simulation

1. Road Network and Toll Zones:
   - The road network is defined using Geodata Frame with Line String geometries representing different roads.
   - Toll zones are similarly defined and represented using Line String which vehicles will travel in their journey.
2. Visualization:
   - The road network and toll zones are visualized using Folium to create an interactive map saved as road_network_and_toll_zones.html.
3. Vehicle Class:
   - An instance of the Vehicle class—this handles the attributes of the vehicle, its movement, checking if it is in a toll zone, and making the payment.

Key Component of Vehicle Class

1. Movement Logic:
   - Car will move from their starting point to their ending point, checking if they pass through any toll zones.
   - This movement is simulated in small incremental steps.
2. Toll Zone Check:
   - The vehicle checks whether it has entered any toll zone employing the intersects () method by seeing if the previous location falls within a buffered area of the toll zone.
3. Calculate Distance:
   - Rates road distance using the osmnx library to find the shortest path given by the road network
4. Payment Processing:
   - Whenever a vehicle reaches its destination, the total toll is computed in relation to the exact distance it travels within toll zones and the amount deducted from the account of that vehicle.

Expected Outputs

1. Vehicle Movement and Toll Zone Entry:

- In the output for each vehicle, you are informed about when it has entered into the toll zone along with the identification toll zone ID.

2. cstring Final Report:

Since this is an analytical task, a summary of statistics for all vehicles shall be compiled in one Data Frame. This will contain:

- vehicle: Unique Identification for every vehicle.
- total toll: It is the sum amount of money that is paid for tolls for a journey.
- account balance: It indicates the remaining balance in the vehicle's account after toll payment.
- crossed zones: This is a list of crossed toll zones by the vehicle.
  Total tollway distance: It is the total amount of distance traversed on the tollways. The distance is converted from meters to kilometers.

Analysis of Results

- Tolls Implication: The toll for every vehicle differs since the distance covered by the vehicle in the toll zone affects the account balance.
- Zone Usage: Leveraging the crossed zones column, it demonstrates for each vehicle how those vehicles used which toll zones, showing various levels of engagement with the toll network.
- Account Management: The account balance will help in monitoring vehicle financial health, ensuring that vehicles have enough funds to pay tolls.

Colony
ayakanahalli
epalya
Madhava
barahalli
Chikkab
shmipura
Gangodanahalli
Vidya
ngadeeranal
Maga Road
Go arahatti

CHAPTER 6
CONCLUSION AND FUTURE ENHANCEMENT

The developed simulation of the GPS-based toll-collecting system GMVs, toll calculation, and account management. It calculates toll charges correctly with respect to vehicle paths through pre-defined toll-collecting zones and user account balances.
This base model thus presented is very sound and provides a firm ground for further advanced developments in this aspect.
Some of the key areas concerning expansions comprise:

- Dynamic pricing: Introduction of dynamic toll rates drawing from such variables as time of day, current congestion, and vehicle type.
- Real-time traffic integration: Efficiency in Routes using real-time information available to drivers.
- Improvement of the user interface: Development of the interactive dashboards for visualization of system performance and user data.
- Additional vehicle types: This can include different vehicle categories that would have varying tolls and behaviors.
- User-centered features: Notice and alerts are directly linked to improving the user experience.
- System scalability: The simulation will increase to accommodate more significant road networks and a higher volume of traffic.
- Data-driven optimization: using data analytics to optimize tolls, understanding bottlenecks in the system, and enhancing efficiency all around.

APPENDIX

```
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point, LineString
import folium
import simpy
import osmnx as ox
import networkx as nx

# Step 1: Setup Environment
# Define road network with the specified points
# You can directly download the data from google maps if you have a google maps API
key
road_network = gpd.GeoDataFrame({
    'road_id': [1, 2, 3, 4],
    'geometry': [
        LineString([
            (77.47249734051701, 13.057311536012028),
            (77.4739418482509,  13.056930904868965),
            (77.47442451666745, 13.056948482047458),
            (77.47508762189501, 13.056763921605565),
            (77.47559563361101, 13.056747596644135),
            (77.47688746290746, 13.05710182151551),
            (77.47730212416313, 13.056635736052938),
            (77.47690085826824, 13.054055916031114),
            (77.47665599760786, 13.052118552821883),
            (77.47628292569452, 13.050287567879304),
            (77.47536912206463, 13.044523686811814),
            (77.47539478436559, 13.042154387399776),
            (77.47611790965406, 13.036744993868902),
            (77.47547661989523, 13.029651082626584),
            (77.47554730628895, 13.023372891407181),
            (77.47618197529408, 13.0173627019377),
            (77.47593081655492, 13.013359026660217),
            (77.47461195320281, 13.006684533333498),
            (77.47407761359283, 12.997658400851929),
            (77.47219227011887, 12.98896439645191)
          ]),
          LineString([
            (77.47880993773907, 13.055137922284267),
            (77.47821629937825, 13.055320833309482),
            (77.47779322236094, 13.055225248386666),
            (77.47746054426267, 13.05505455251026),
            (77.47709801930544, 13.054680375648656),
            (77.47690085826824, 13.054055916031114)
          ]),
          LineString([
            (77.47536781949456, 13.043201932046639),
            (77.47574476122185, 13.043129215858327),
            (77.47582313524435, 13.042787449486818),
            (77.47605945283462, 13.042766513508187),
            (77.47622306760007, 13.042993845532644),
            (77.47716930749428, 13.042771880539204),
            (77.47872720112112, 13.042585142500284)
          ]),
          LineString([
            (77.47302238050207, 12.992710856898563),
            (77.47329278144292, 12.992017143148514),
            (77.47366686628436, 12.991652163766275),
            (77.47375088970591, 12.991421165796439),
            (77.47360985039299, 12.991070282395688),
            (77.47305571535811, 12.990704255001114),
            (77.47290971711307, 12.99033416331594),
            (77.4727206644122,  12.98950958292108),
            (77.47274467110745, 12.988997873847191),
            (77.47265164517427, 12.988711316303007)
          ])
```

```
    ]
})

# Define toll zones (using LineStrings for simplicity)
toll_zones = gpd.GeoDataFrame({
    'zone_id': [1, 2, 3],
    'geometry': [
        LineString([
            (77.4739418482509,  13.056930904868965),
            (77.47442451666745, 13.056948482047458),
            (77.47508762189501, 13.056763921605565),
            (77.47559563361101, 13.056747596644135),
            (77.47688746290746, 13.05710182151551),
            (77.47730212416313, 13.056635736052938),
            (77.47690085826824, 13.054055916031114),
            (77.47665599760786, 13.052118552821883),
            (77.47628292569452, 13.050287567879304),
            (77.47536912206463, 13.044523686811814),
            (77.47539478436559, 13.042154387399776),
            (77.47611790965406, 13.036744993868902),
            (77.47547661989523, 13.029651082626584),
            (77.47554730628895, 13.023372891407181),
            (77.47618197529408, 13.0173627019377),
            (77.47593081655492, 13.013359026660217),
            (77.47461195320281, 13.006684533333498),
            (77.47407761359283, 12.997658400851929),
            (77.47219227011887, 12.98896439645191)
        ]),
        LineString([
            (77.47821629937825, 13.055320833309482),
            (77.47779322236094, 13.055225248386666),
            (77.47746054426267, 13.05505455251026),
            (77.47709801930544, 13.054680375648656),
            (77.47690085826824, 13.054055916031114)
        ]),
        LineString([
            (77.47302238050207, 12.992710856898563),
            (77.47329278144292, 12.992017143148514),
            (77.47366686628436, 12.991652163766275),
            (77.47375088970591, 12.991421165796439),
            (77.47360985039299, 12.991070282395688),
            (77.47305571535811, 12.990704255001114),
            (77.47290971711307, 12.99033416331594),
            (77.4727206644122,  12.98950958292108),
            (77.47274467110745, 12.988997873847191),
            (77.47265164517427, 12.988711316303007)
        ])
    ]
})

# Visualize the road network and toll zones
m = folium.Map(location=[13.05, 77.48], zoom_start=13)  # Adjusted center for better
map view
for _, row in road_network.iterrows():
    folium.PolyLine(locations=[(coord[1], coord[0]) for coord in row.geometry.coords],
color='blue', weight=5, popup=f"Road {row.road_id}").add_to(m)
for _, row in toll_zones.iterrows():
    folium.PolyLine(locations=[(coord[1], coord[0]) for coord in row.geometry.coords],
color='red', weight=5, name=f"Zone {row.zone_id}").add_to(m)

# Add markers for the entry points(start of toll zone)
entry_points = [
    (13.056930904868965, 77.4739418482509),
    (13.055320833309482, 77.47821629937825)
]
for entry in entry_points:
    folium.Marker(
        location=entry,
        popup='Entry Point',
        icon=folium.Icon(color='green', icon='play')
    ).add_to(m)

# Add markers for the exit points
exit_points = [
    (13.043201932046639, 77.47536781949456)
]
for exit in exit_points:
    folium.Marker(
        location=exit,
        popup='Exit Point',
        icon=folium.Icon(color='red', icon='stop')
    ).add_to(m)

m.save('road_network_and_toll_zones.html')

# Download the road network for the area of interest
G = ox.graph_from_point((13.025369058991071, 77.47556873116585), dist=6000,
network_type='drive')

# Step 2: Simulate Vehicle Movement
class Vehicle:
    def __init__(self, env, vehicle_id, start_location, end_location, toll_zones, rates,
accounts, G):
        self.env = env
        self.vehicle_id = vehicle_id
        self.start_location = start_location
        self.current_location = start_location
        self.end_location = end_location
        self.toll_zones = toll_zones
        self.rates = rates
        self.accounts = accounts
        self.crossed_zones = set()
        self.total_toll = 0.0
        self.total_distance = 0.0
        self.G = G  # Road network graph
        self.action = env.process(self.run())

    def calculate_road_distance(self):
        start_node = ox.distance.nearest_nodes(self.G, self.start_location.x, 
self.start_location.y)
        end_node = ox.distance.nearest_nodes(self.G, self.end_location.x, 
self.end_location.y)
        self.total_distance = nx.shortest_path_length(self.G, start_node, end_node, 
weight='length')

    def move(self):
        step_size = 0.0001  # Reduced step size for higher precision
        while self.current_location.distance(self.end_location) > step_size:
            prev_location = self.current_location
            new_x = self.current_location.x + (self.end_location.x - self.current_location.x) * 
step_size / self.current_location.distance(self.end_location)
            new_y = self.current_location.y + (self.end_location.y - self.current_location.y) * 
step_size / self.current_location.distance(self.end_location)
            self.current_location = Point(new_x, new_y)
            self.check_toll_zones(prev_location)
            yield self.env.timeout(0.01)
        self.current_location = self.end_location

    def check_toll_zones(self, prev_location):
        for _, zone in self.toll_zones.iterrows():
            # Check if the previous or current location intersects with the buffered zone 
geometry
            if (prev_location.buffer(0.001).intersects(zone.geometry)):
                self.calculate_road_distance()
                if zone.zone_id not in self.crossed_zones:
                    print(f"Vehicle {self.vehicle_id} entered Toll Zone {zone.zone_id} at
{self.current_location}")
                self.crossed_zones.add(zone.zone_id)

    def run(self):
        while not self.current_location.equals_exact(self.end_location, 0.001):
            yield self.env.process(self.move())
        self.process_payment()

    def process_payment(self):
        self.total_toll = self.total_distance * self.rates
        self.accounts[self.vehicle_id] -= self.total_toll
        print(f"Vehicle {self.vehicle_id} finished journey, total toll: {self.total_toll},
remaining balance: {self.accounts[self.vehicle_id]}, total distance: {self.total_distance}
meters")

# Define toll rates
toll_rates = 0.003025  # Set different rates for each toll zone

# Initialize user accounts
user_accounts = {i: 100.0 for i in range(6)}  # Adjusted to 6 vehicles

# Simulate vehicle movement with toll zone detection, toll calculation, and payment
processing
env = simpy.Environment()
vehicles = [Vehicle(env, i, Point(77.47249734051701, 13.057311536012028),
Point(77.47219227011887, 12.98896439645191), toll_zones, toll_rates, user_accounts, G)
for i in range(4)]
# Add a new vehicle with the specified start and end points
vehicles.append(Vehicle(env, 4, Point(77.47880993773907, 13.055137922284267),
Point(77.47605945283462, 13.042766513508187), toll_zones, toll_rates, user_accounts,
G))
vehicles.append(Vehicle(env, 5, Point(77.47276449816579, 13.057260476758666),
Point(77.47265164517427, 12.988711316303007), toll_zones, toll_rates, user_accounts,
G))
```

```
env.run(until=100)

# Collect vehicle data
vehicle_data = []
for vehicle in vehicles:
    vehicle_data.append({
        'vehicle_id': vehicle.vehicle_id,
        'total_toll': vehicle.total_toll,
        'account_balance': vehicle.accounts[vehicle.vehicle_id],
        'crossed_zones': list(vehicle.crossed_zones),
        'total distance on toll roads': vehicle.total_distance/1000
    })

# Create DataFrame for analysis
df = pd.DataFrame(vehicle_data)

# Generate report
print(df)
```

REFERENCES

1. https://shapely.readthedocs.io/en/stable/manual.html
2. https://osmnx.readthedocs.io/en/stable/
3. https://simpy.readthedocs.io/en/latest/
4. https://geopandas.org/en/stable/getting_started/tutorial.html
5. https://nbviewer.jupyter.org/github/python-visualization/folium/tree/main/examples/
6. https://stackoverflow.com/