International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

GPS-Enabled Toll Collection System (Tolltrack) Using Raspberry PI

Vadlapally Yashwanth Reddy [1], K.Shirisha [2]

[1] Student, Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Gandipet, India.
[2]Assistant Professor, Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Gandipet, India.

Abstract
The rapid increase in vehicular traffic has created significant challenges in traditional toll collection systems, including congestion, fuel wastage, and inefficient fixed pricing models. Existing methods such as manual toll booths and RFID-based FASTag systems still require vehicles to slow down or stop, leading to delays and reduced efficiency. This project proposes **TollTrack**, a GPS-enabled toll collection system using Raspberry Pi that eliminates the need for physical toll plazas and introduces a fully automated, distance-based tolling mechanism. The system is built using a Raspberry Pi as the On-Board Unit (OBU), integrated with a GPS module to continuously capture real-time vehicle location. The collected coordinates are transmitted to a cloud-based backend server using Wi-Fi or GSM communication. The backend applies geofencing techniques to detect entry and exit from toll zones and calculates the distance travelled using the Haversine formula. Based on this distance, toll charges are computed dynamically and updated in real time. The system provides accurate tracking with minimal delay and ensures fair pricing by charging users only for the distance travelled. It reduces congestion, eliminates manual intervention, and enhances transparency through real-time monitoring and logging. The proposed solution is scalable, cost-effective, and aligns with future GNSS-based tolling systems, making it suitable for large-scale deployment in smart transportation infrastructure.

Keywords: Raspberry Pi, GPS, Geofencing, Toll Collection, IoT, Haversine Formula, Smart Transportation, GNSS

1. Introduction

The TollTrack system is an automated toll collection solution designed to eliminate traditional toll booths by using GPS-based tracking and distance-based billing. Unlike conventional toll systems that rely on manual payment or RFID-based scanning, this system enables seamless toll collection without requiring vehicles to stop or slow down.

With the increasing number of vehicles on highways, traditional toll systems have become inefficient due to congestion, delays, and fixed toll charges that do not reflect actual road usage. FASTag has improved efficiency but still depends on physical infrastructure and is prone to scanning delays and errors.

The proposed system leverages **Raspberry Pi, GPS technology, and cloud computing** to create a fully automated tolling mechanism. The system continuously tracks vehicle location, detects toll zones using geofencing, and calculates toll charges based on the exact distance travelled. This ensures fair pricing, reduces fuel consumption, and enhances overall traffic flow.

By integrating IoT and real-time data processing, TollTrack provides a scalable and future-ready solution aligned with modern GNSS-based tolling systems.

1.1 Problem Definition

Traditional toll collection systems face several operational and efficiency-related challenges that impact both users and highway authorities. Manual toll booths require vehicles to stop for payment, resulting in long queues, increased travel time, and fuel wastage. Even with the introduction of RFID-based FASTag systems, vehicles are still required to slow down, and issues such as scanning errors, tag misreads, and lane congestion continue to persist.

Another major limitation of existing systems is the use of **fixed toll pricing**, where users are charged the same amount regardless of the distance travelled. This leads to unfair billing, especially for short-distance travelers who end up paying more than their actual road usage.

Additionally, traditional systems require significant physical infrastructure, including toll plazas, manpower, and maintenance, leading to high operational costs. There is also a lack of transparency, as users are not provided with detailed insights into their travel distance or toll charges.

These challenges highlight the need for a **fully automated, infrastructure-less toll collection system** that can provide real-time tracking, fair pricing, and seamless vehicle movement without interruptions.

1.2 Existing Applications

Existing toll collection methods include manual systems, RFID-based systems, and camera-based recognition systems, each with its own limitations.

Manual toll collection systems rely on cash or card payments at toll booths, which are time-consuming and prone to human error. These systems often lead to heavy congestion, especially during peak hours.
RFID-based systems such as FASTag improve efficiency by enabling automatic toll deduction. However, they still depend on physical toll plazas and require vehicles to pass through designated lanes. Issues such as tag misreads, network delays, and blacklisted tags can disrupt the process.

Automatic Number Plate Recognition (ANPR) systems use cameras and image processing techniques to identify vehicles. While they reduce the need for RFID tags, they require expensive infrastructure and are sensitive to environmental conditions such as poor lighting, weather, and dirty number plates.

Mobile-based GPS tolling solutions have also been explored, but they depend heavily on smartphone availability, battery life, and user compliance, making them unreliable for large-scale deployment.

1.3 Proposed Application

The proposed system, **TollTrack**, is a GPS-based automated toll collection solution that eliminates the need for physical toll plazas and enables fair, distance-based billing. The system utilizes a Raspberry Pi as an On-Board Unit (OBU) installed in the vehicle, integrated with a GPS module to continuously capture real-time location data. These coordinates are transmitted to a backend server through Wi-Fi or GSM communication, where geofencing techniques are applied to detect the vehicle's entry and exit from predefined toll zones. Once the vehicle exits the toll zone, the system calculates the total distance travelled using the Haversine formula and computes the toll amount dynamically based on a predefined rate per kilometer. The calculated toll can be automatically deducted through integrated digital payment systems such as UPI or prepaid wallets, ensuring a seamless and contactless experience. Additionally, the system provides real-time tracking, trip history, and transparent billing information to users. By removing the dependency on physical infrastructure, reducing congestion, and ensuring accurate billing, TollTrack offers a scalable and efficient solution aligned with modern GNSS-based tolling systems.

1.4 Requirements Specification

Requirement Specifications describe the artifact of Software Requirements and Hardware Requirements used in this project.

Software Requirements

1. **Operating System:** Raspberry Pi OS / Linux
2. **Programming Language:** Python 3.8 or later
3. **Backend Framework:** Flask / FastAPI
4. **Database:** MySQL / PostgreSQL

5. **Libraries/Frameworks:**
    - Requests (for HTTP communication)
    - GPS libraries (GPSD / PyGPS)
    - Folium (for map visualization)
    - JSON (for data formatting)
6. **Communication Protocols:** HTTP / HTTPS
7. **Cloud Services:** AWS / Local Server
8. **Security:** HTTPS communication, authentication mechanisms
9. **Additional Tools:** REST APIs for data exchange

Hardware Requirements

1. **Processor:** Raspberry Pi (3 / 4 / Zero W)
2. **GPS Module:** NEO-6M / SIM868 (for location tracking)
3. **Communication Module:** GSM (SIM800L / SIM868) or Wi-Fi
4. **Memory:** 2GB–4GB RAM (depending on Pi model)
5. **Storage:** 16GB–32GB SD Card
6. **Power Supply:** Power bank or vehicle battery
7. **Antenna:** GPS and GSM antennas
8. **Optional Components:** Display module (LCD/OLED)

2. Literature Survey

"GPS Based Toll System Simulation" by Praneet More, Chirag Ranpise, et al. presents a simulation framework for toll collection using GPS and geofencing. The system models vehicle movement across virtual toll zones and calculates tolls dynamically based on entry and exit points. It also explores machine learning-based pricing strategies. However, the work is limited to simulation and lacks real-world hardware validation. This paper provides a strong foundation for understanding GPS-based toll computation.

"A GPS-Driven Toll System Using Geofencing" by Manasi Gaikwad, Jagruti Gawade, et al. proposes a fully GPS-based tolling system that eliminates physical toll booths by using virtual geofences. The system detects when vehicles enter and exit toll zones and calculates distance-based charges. While effective, it highlights challenges such as GPS inaccuracies and signal loss. This paper is highly relevant to the proposed TollTrack system's core concept.

"Toll Plaza Automation Using GPS" by Gaurav Tyagi, Kanika Paliwal, et al. introduces a GPS-based toll collection system implemented using smartphones. The system calculates toll charges based on vehicle movement and integrates with payment gateways. Although cost-effective, it depends on mobile devices, which limits reliability. This study emphasizes the need for dedicated hardware like Raspberry Pi for consistent performance.

"GPS Based E-Toll Gate Collection System" by Goutham K, Gowtham M, et al. focuses on distance-based toll calculation using entry and exit coordinates. The system ensures fair billing by charging users according to actual distance travelled. However, it lacks detailed backend implementation and real-time processing. This paper strongly supports the concept of distance-based toll pricing used in TollTrack.

"Smart Toll Collection: A GPS Based Automated System" by R. Subha and Neelabru Pal presents a GPS-triggered tolling mechanism that detects vehicles at predefined locations and automatically deducts toll charges. While simple and effective, the system relies on fixed toll points rather than continuous tracking, limiting accuracy. It serves as a basic reference for GPS-based toll detection.

"GPS Tracking System Using Raspberry Pi Zero W" by Pooja Nayak, Prakash, et al. demonstrates real-time vehicle tracking using Raspberry Pi and GPS modules. The system highlights challenges such as GPS drift and network instability but proves the feasibility of using Raspberry Pi as an On-Board Unit (OBU). This paper is directly relevant to the hardware implementation of TollTrack.

"Development of a GPS-Based Highway Toll Collection System" by Jin Yeong Tan, Pin Jern Ker, et al. presents an early implementation of a GPS-based toll system using Raspberry Pi, GPS modules, and cloud communication. The system detects toll points and performs automatic deductions. Although based on older hardware, it validates the practical feasibility of GNSS-based tolling systems.

"GPS Navigation and Toll Charging System" by IoT Research Community explains the mathematical foundation of toll calculation using the Haversine formula. It discusses techniques to compute accurate distances between GPS coordinates and reduce errors caused by signal noise. This paper is crucial for understanding the distance calculation methodology used in the proposed system.

3. Methodology

This section describes the approach used to design, develop, and implement the TollTrack system, ensuring a reliable, scalable, and real-time toll collection solution based on GPS technology.

3.1 Technologies Used

This section outlines the key technologies used in developing the TollTrack system.

a) **Programming Language:** Python 3.8 or later, Python is used for both Raspberry Pi programming and backend development due to its simplicity, extensive library support, and ease of integration with hardware components such as GPS modules.
b) **Hardware Platform:** The Raspberry Pi acts as the On-Board Unit (OBU), responsible for collecting GPS data, processing it, and transmitting it to the backend server in real time.
c) **Backend Framework:** Flask is used to build a lightweight backend server that receives GPS data, processes it, performs geofencing checks, and calculates toll charges.
d) **Database: MySQL / PostgreSQL:** A relational database is used to store vehicle data, trip records, toll transactions, and system logs for efficient retrieval and analysis.
e) **Communication:** The system uses HTTP-based communication to transmit GPS data from the Raspberry Pi to the backend server in JSON format.
f) **Geospatial Processing:** The Haversine formula is used to calculate the distance between consecutive GPS coordinates for accurate toll computation.
g) **Visualization:** The Folium library is used to generate real-time maps showing vehicle movement and travel paths.

3.2 Development Process:

The development of the TollTrack system follows an iterative and modular approach to ensure flexibility and continuous improvement.

a) **Requirement Analysis:** The system requirements were identified by analysing existing toll collection methods such as manual booths and RFID-based systems. Key issues such as congestion, fixed pricing, and infrastructure dependency were identified, leading to the need for a GPS-based automated solution.
b) **System Design:** The architecture was designed as a distributed client-server model, where the Raspberry Pi acts as the client (OBU) and the Flask server acts as the backend processing unit. The design ensures modularity, allowing each component to function independently while maintaining seamless integration.
c) **Module Implementation:** The system is divided into multiple modules:

- **GPS Data Acquisition Module:** Collects real-time location data from the GPS module.
- **Data Transmission Module:** Sends GPS coordinates to the server using HTTP requests.
- **Geofencing Module:** Detects vehicle entry and exit from toll zones.
- **Distance Calculation Module:** Computes distance using GPS coordinates.
- **Toll Calculation Module:** Calculates toll based on distance travelled.

d) **Backend Integration:** The Flask server processes incoming GPS data, validates coordinates, performs geofencing, calculates distance, and updates toll values in real time.
e) **Testing:** Each module was tested individually (unit testing) and then integrated to ensure proper communication between components. Real-time testing was performed using a moving vehicle. During testing, I observed small variations in GPS readings, which were handled by filtering out minor coordinate changes.
f) **Deployment and Evaluation:** The system was deployed in a real-world environment, and performance metrics such as GPS accuracy, communication delay, and toll calculation accuracy were evaluated.

4. Design of TollTrack System

The TollTrack system is designed as a modular, scalable, and real-time GPS-based toll collection platform. It follows a distributed architecture where the vehicle unit and backend server interact continuously to process location data and compute toll charges. The design focuses on accuracy, automation, and elimination of physical toll infrastructure.

4.1. System Architecture

The system consists of multiple components interacting through GPS data transmission and backend processing, as shown in Figure 4.1.

Key Components:

- **Raspberry Pi (On-Board Unit):** Captures GPS data and sends it to the server
- **GPS Module:** Provides real-time latitude and longitude coordinates
- **Backend Server (Flask):** Processes data, performs geofencing, and calculates toll
- **Database:** Stores trip data, distance, and toll records
- **Map Visualization:** Displays vehicle movement and route tracking

The system operates in a continuous loop where GPS data is collected, transmitted, processed, and updated in real time.

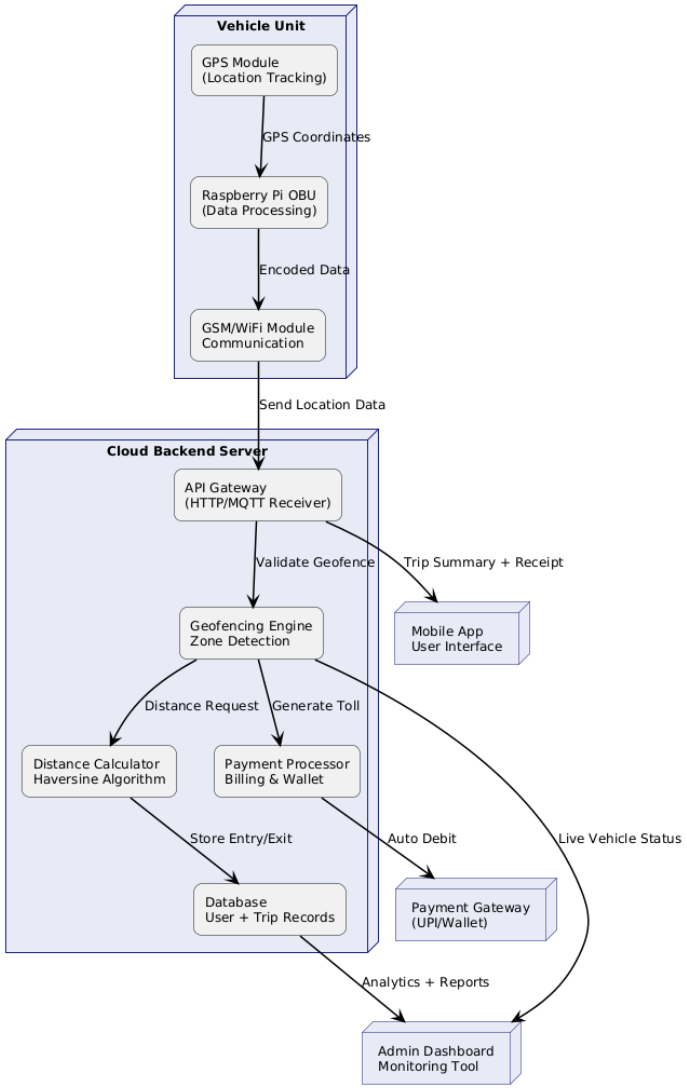


Figure 4.1: System architecture diagram

4.2. Database Design

The system uses a structured database to store vehicle movement and toll-related information.

Main Data Elements:

- **Vehicle Data:** Vehicle ID and identification details
- **Trip Records:** Timestamp, GPS coordinates, and travel path
- **Distance Data:** Incremental and total distance travelled
- **Toll Data:** Calculated cost based on distance

Core Functions:

- storeTripData() → Stores GPS coordinates and timestamps
- calculateDistance() → Updates total distance travelled

- computeToll() → Calculates toll amount based on distance

This design ensures efficient storage, retrieval, and processing of real-time data.

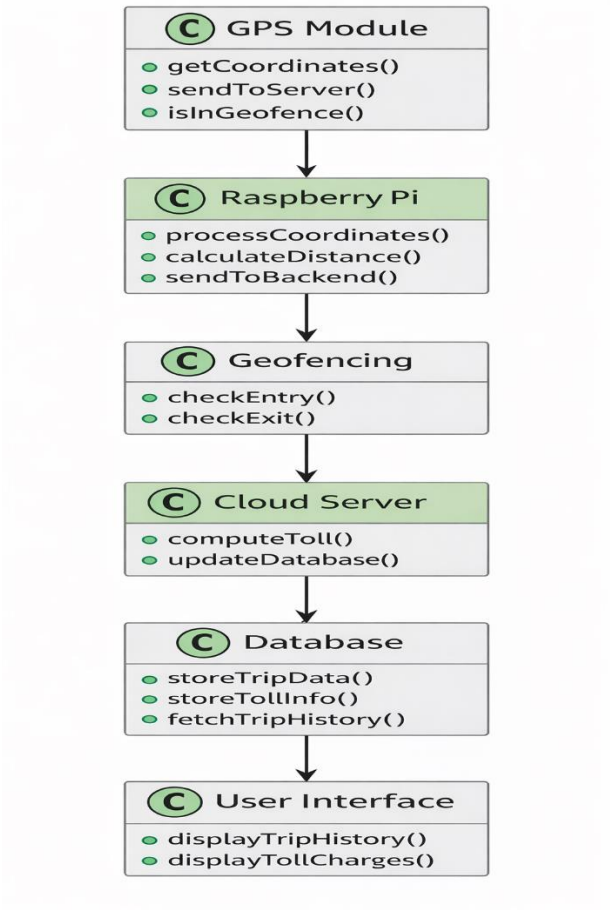


Figure 4.2: ER diagram

4.3. Control Flow showing System Activity

The control flow diagram illustrates how the system processes GPS data and calculates toll charges step by step.

Workflow:

1. Raspberry Pi collects GPS coordinates
2. Data is sent to the backend server
3. Server validates incoming data
4. Geofencing module checks toll zone entry/exit
5. Distance is calculated incrementally
6. Toll is computed dynamically
7. Results are stored and visualized

This flow ensures continuous tracking and real-time toll updates.

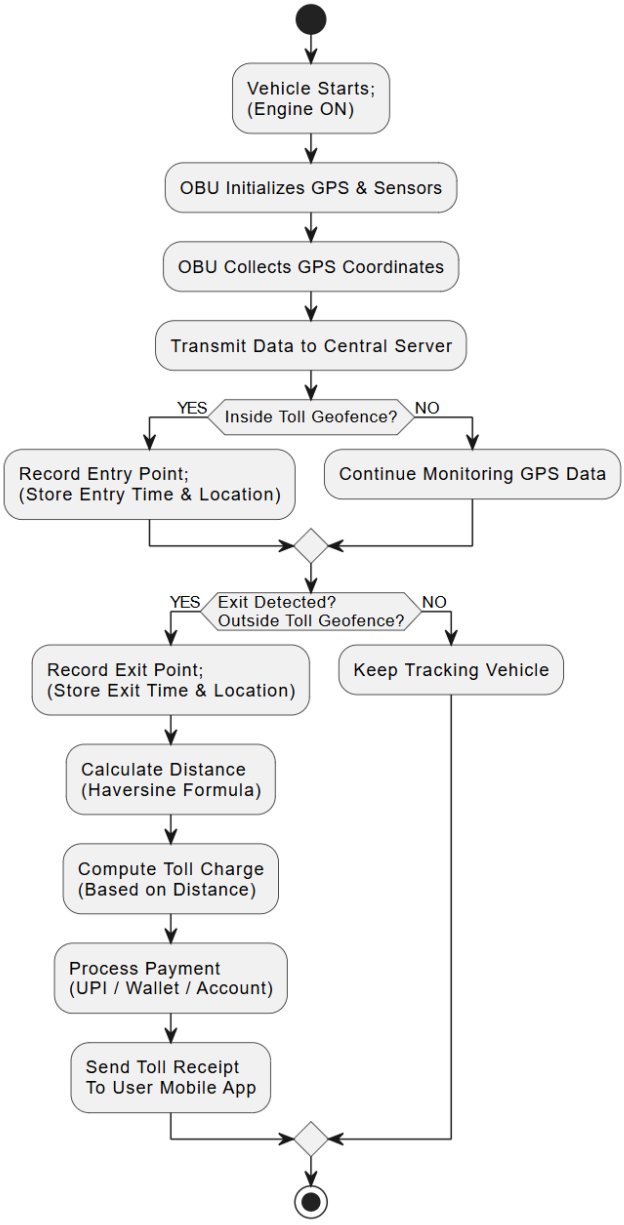


Figure 4.3: Control Flow Diagram

4.4. Modular Architecture & Object-Oriented Structure

The system is designed using modular components, each responsible for a specific function.

Key Modules & Responsibilities:

- **GPS Module:**
  - Captures real-time coordinates
  - Provides latitude and longitude data
- **Raspberry Pi Module:**
  - Processes GPS data

  - Sends data to backend server
- **Geofencing Module:**
  - Detects whether vehicle is inside toll zone
  - Identifies entry and exit points
- **Distance Calculation Module:**
  - Computes distance between consecutive coordinates
  - Accumulates total travel distance
- **Toll Calculation Module:**
  - Calculates toll based on distance and rate
  - Updates cost dynamically
- **Database Module:**
  - Stores trip logs and toll data
  - Retrieves historical data
- **Visualization Module:**
  - Displays real-time vehicle tracking on maps

This modular approach improves scalability, maintainability, and system reliability.

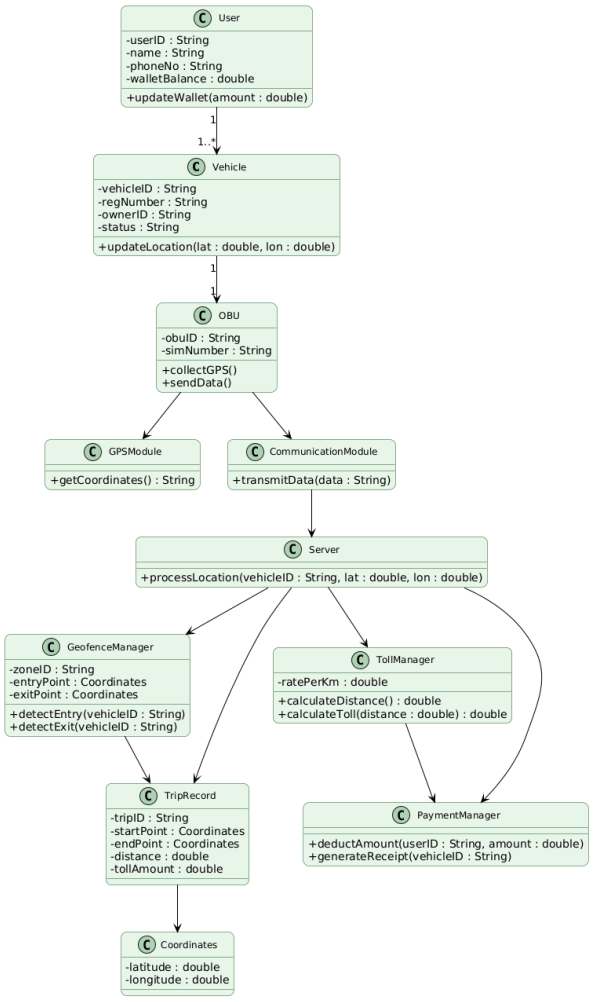


Figure 4.4: Class diagram

4.5 Role-Based System Interaction

The system supports interaction between different components and users.

Actors & Interactions:

- **User (Vehicle Owner):**
  - Travels through toll zones
  - Views toll charges and trip history
- **Admin:**
  - Monitors system activity
  - Manages toll zones and pricing
- **System (TollTrack Engine):**
  - Automatically processes GPS data
  - Calculates toll without manual intervention

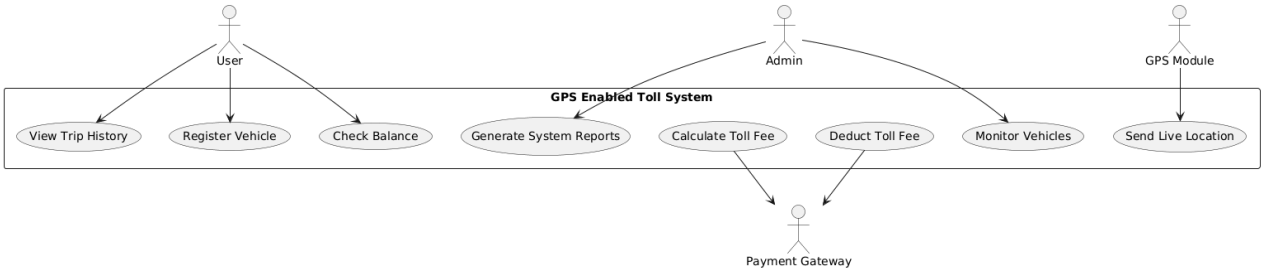


Figure 4.5: Use case diagram

4.6. Vehicle-Based Process Workflow

The sequence diagram represents the step-by-step interaction between system components during toll processing.

Process Flow:

1. Vehicle starts and Raspberry Pi initializes
2. GPS module captures location data
3. Data is sent to backend server
4. Server checks geofence entry
5. Distance is calculated continuously
6. On exit, total distance is computed
7. Toll amount is calculated and stored
8. Results are displayed to the user

This workflow ensures seamless toll calculation without any physical interruption.

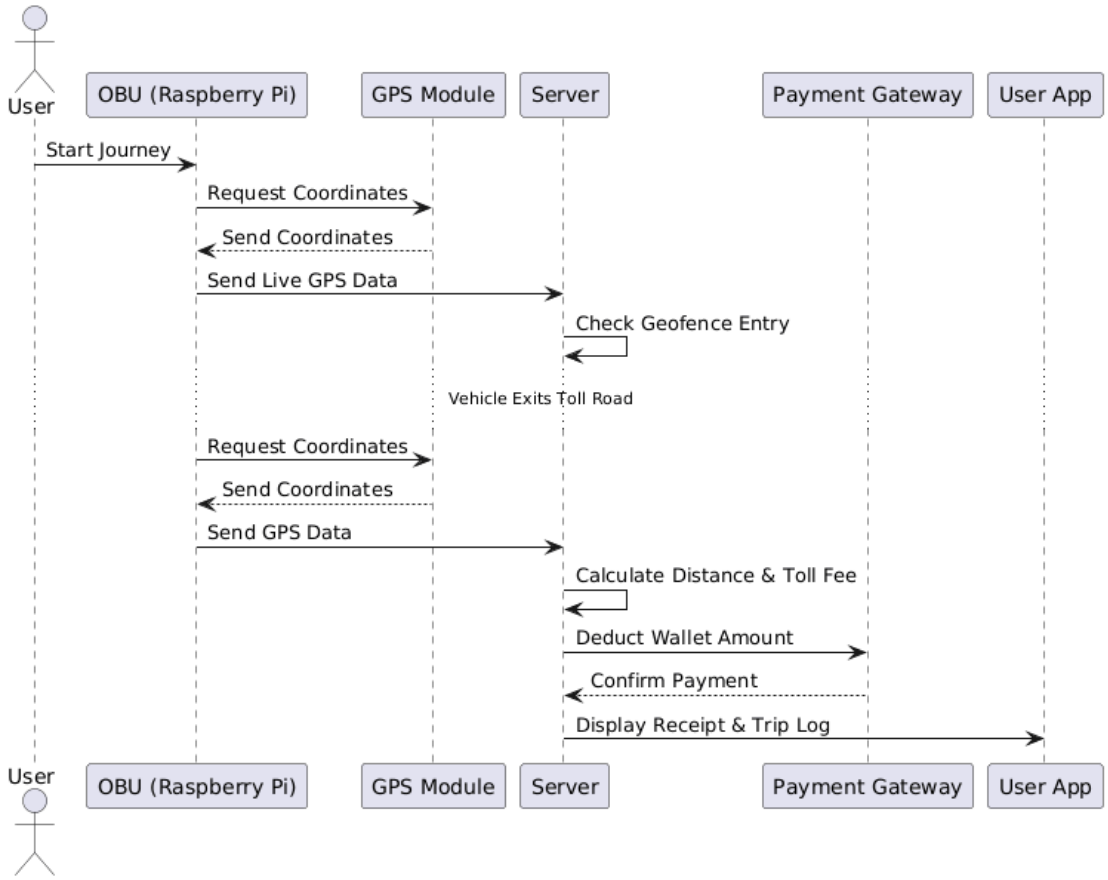


Figure 4.6: Sequence Diagram for Vehicle

5. Implementation

5.1 System Architecture

The TollTrack system is implemented as a real-time, distributed application consisting of a Raspberry Pi–based On-Board Unit (OBU) and a Flask-based backend server. The Raspberry Pi continuously collects GPS data from the GPS module and transmits it to the backend server using HTTP requests. The backend server processes this incoming data, performs geofencing checks, calculates the travelled distance, and computes toll charges dynamically.

The system follows a modular architecture where each component operates independently while maintaining seamless communication. The main modules include the GPS data acquisition module, communication module, geofencing module, distance calculation module, toll computation module, and database module. This design ensures scalability, flexibility, and efficient real-time processing of vehicle data.

5.2 Database Integration

The system uses a structured database to store GPS data, trip details, and toll information. The database maintains records such as vehicle ID, timestamps, latitude, longitude, total distance travelled, and calculated toll charges.

Data is received from the Raspberry Pi in JSON format and stored sequentially in the database. Each new GPS coordinate is appended to the trip record, allowing continuous tracking of the vehicle's path. The system updates the total distance and toll amount dynamically as new data is received.

Database operations are handled through backend functions that ensure efficient storage, retrieval, and updating of data. This enables real-time tracking, historical analysis, and transparency in toll computation.

5.3 GPS and Communication Integration

The GPS module is interfaced with the Raspberry Pi using serial communication. It continuously sends location data in the form of NMEA sentences or AT command responses. The Raspberry Pi extracts relevant information such as latitude and longitude from this data.

Once valid coordinates are obtained, the system sends them to the backend server using HTTP POST requests. The data is structured in JSON format, ensuring easy parsing and compatibility with the server.

To ensure reliability, the system includes validation checks for GPS signals, ignores invalid or zero coordinates, and implements retry mechanisms in case of network failure. This ensures consistent and accurate data transmission.

5.4 Geofencing and Distance Calculation

Geofencing is implemented using predefined coordinate points that represent toll road boundaries. For each incoming GPS coordinate, the system checks whether the vehicle is within the toll zone by calculating its proximity to these predefined points.

Once the vehicle enters the toll zone, the system begins tracking distance. The distance is calculated incrementally between consecutive GPS coordinates using the Haversine formula. This method provides accurate measurement of the actual path travelled rather than relying only on entry and exit points.

To improve accuracy, small fluctuations in GPS data are filtered using threshold values, ensuring that noise does not affect distance calculation.

5.5 Toll Calculation Engine

The toll calculation module computes the total toll based on the distance travelled within the toll zone. The system uses a linear pricing model where the toll amount is calculated as:

Toll = Distance × Rate_per_km

The rate is predefined and can be configured by the administrator. The system updates the toll amount dynamically as the vehicle moves within the geofence.

Additional checks are implemented to ensure that toll calculation starts only after entering the toll zone and stops immediately after exiting. This prevents incorrect billing and ensures fairness in pricing.

5.6 User Interface and Visualization

The system provides a basic visualization interface using the Folium library, which generates interactive maps displaying the vehicle's path. Each GPS coordinate is plotted on the map, creating a visual representation of the journey.

The interface also displays key information such as total distance travelled and toll charges. This improves transparency and helps users understand how the toll is calculated.

The visualization is accessible through a web browser, allowing easy monitoring of the system without requiring additional applications.

5.7 Error Handling and Reliability

The system includes several mechanisms to ensure reliability and robustness during operation.

- **GPS Signal Handling:** Invalid or weak GPS signals are ignored to prevent incorrect calculations.
- **Noise Filtering:** Small coordinate changes are filtered out to reduce GPS drift errors.
- **Network Failure Handling:** In case of communication failure, the system retries data transmission to avoid data loss.
- **Data Validation:** All incoming data is validated on the server before processing.

The modular design allows each component to function independently, making the system easier to debug and maintain. These measures ensure stable and continuous operation under real-world conditions.

6. Testing and Results

The TollTrack system was tested to evaluate its performance, accuracy, and reliability under real-time conditions. The testing phase focused on validating GPS tracking, geofencing detection, distance calculation, and toll computation.

Figure 6.1 shows the overall setup of the system, including the Raspberry Pi, GPS module, and communication interface used for real-time data transmission.

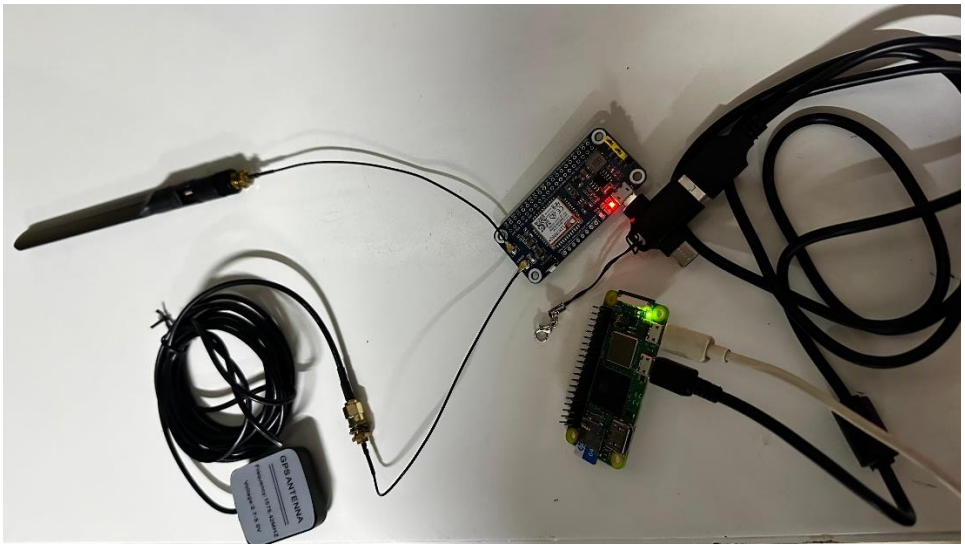
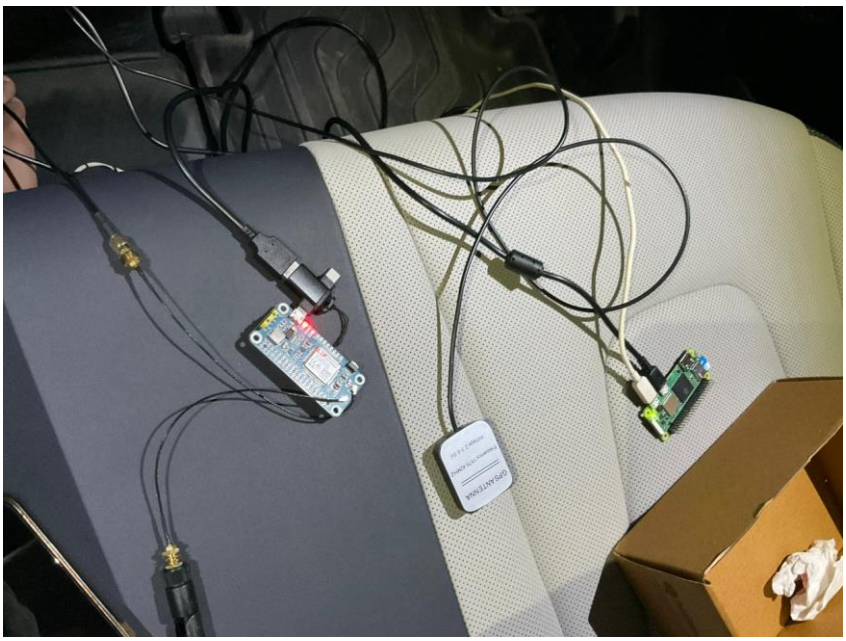


Figure 6.1: Project Setup Overview

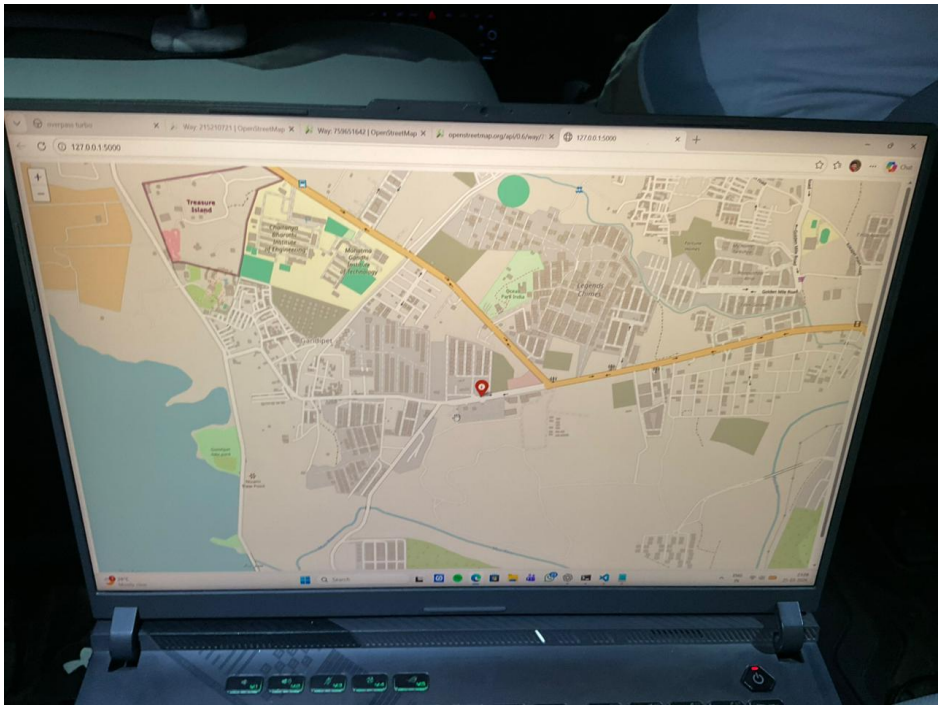
Figure 6.2 demonstrates the initialization of the GPS module and the acquisition of the first valid location coordinates. The system begins tracking once a stable GPS signal is obtained.



```
(tolltrack-env) yash@Yash:~ $ python gps_reader.py
GPS + Highway Tracking Started...

⏳ Waiting for GPS signal...
🎉 First GPS Fix: 17.385109, 78.329937

📍 Location: 17.385109, 78.329937

📍 Location: 17.38512, 78.329994

📍 Location: 17.385134, 78.330045

📍 Location: 17.385138, 78.330091

📍 Location: 17.385161, 78.330157

📍 Location: 17.385179, 78.330217

📍 Location: 17.385192, 78.330277

📍 Location: 17.385219, 78.330337
🚗 Entered Highway

📍 Location: 17.385268, 78.330389
📏 Distance Added: 0.0078 km

📍 Location: 17.385292, 78.330449
📏 Distance Added: 0.0069 km

📍 Location: 17.385326, 78.330505
📏 Distance Added: 0.0070 km

📍 Location: 17.385342, 78.330557
📏 Distance Added: 0.0058 km

📍 Location: 17.38536, 78.330596
```

Figure 6.2: Project Setup in Car and GPS Tracking Initialization

Figure 6.3 shows the real-time plotting of GPS coordinates on the map, where the vehicle's current position is displayed along with its movement path.



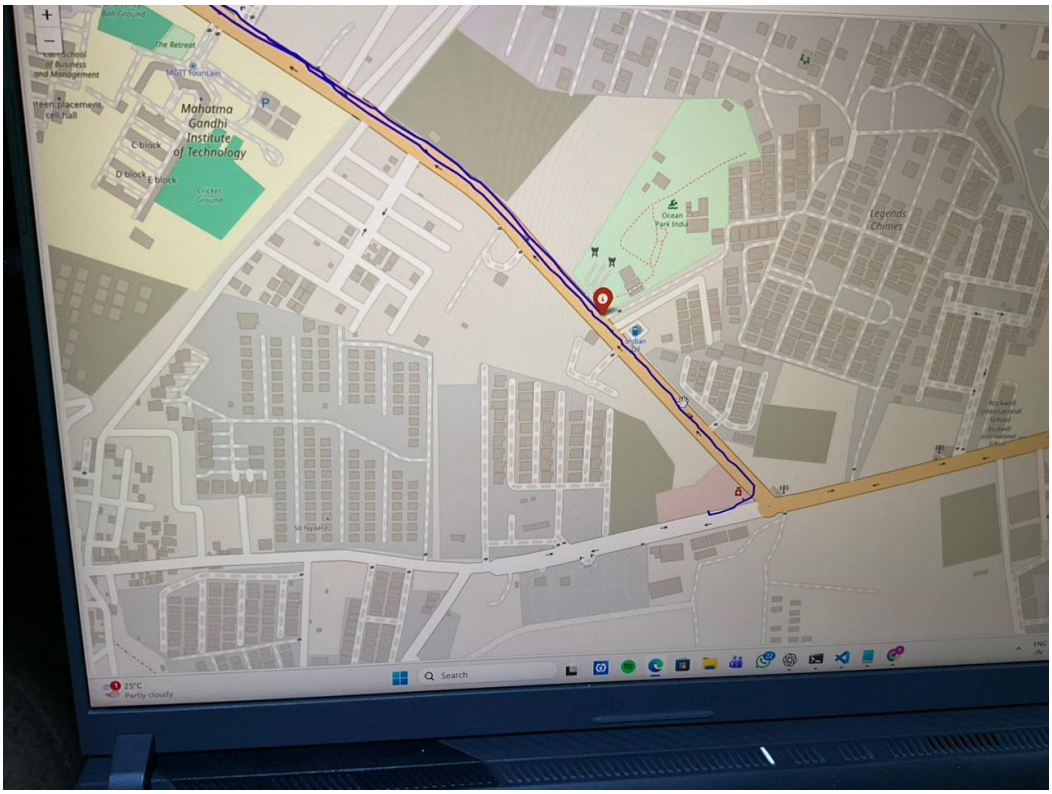


Figure 6.3: Real-Time Location Tracking

Figure 6.4 illustrates the server-side processing where incoming GPS data is received and logged. The backend begins calculating distance and toll once the vehicle enters the toll zone.

```
(tolltrack-env) PS G:\TollTrack> python server.py
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.20.10.2:5000
Press CTRL+C to quit
Stored: {'time': '23:56:03', 'lat': 17.385164, 'lon': 78.329914, 'distance': 0, 'cost': 0}
172.20.10.4 - - [25/Mar/2026 23:56:03] "POST /location HTTP/1.1" 200 -
127.0.0.1 - - [25/Mar/2026 23:56:10] "GET / HTTP/1.1" 200 -
Stored: {'time': '23:56:12', 'lat': 17.385164, 'lon': 78.329913, 'distance': 0, 'cost': 0}
172.20.10.4 - - [25/Mar/2026 23:56:12] "POST /location HTTP/1.1" 200 -
127.0.0.1 - - [25/Mar/2026 23:56:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Mar/2026 23:57:30] "GET / HTTP/1.1" 200 -
Stored: {'time': '23:57:46', 'lat': 17.385165, 'lon': 78.329913, 'distance': 0, 'cost': 0}
172.20.10.4 - - [25/Mar/2026 23:57:46] "POST /location HTTP/1.1" 200 -
Stored: {'time': '23:59:08', 'lat': 17.385164, 'lon': 78.329913, 'distance': 0, 'cost': 0}
172.20.10.4 - - [25/Mar/2026 23:59:08] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:01:55', 'lat': 17.385109, 'lon': 78.329937, 'distance': 0.006624806926293446, 'cost': 0.013249613852586892}
172.20.10.4 - - [26/Mar/2026 00:01:55] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:01:57', 'lat': 17.38512, 'lon': 78.329994, 'distance': 0.012795813245580143, 'cost': 0.025591626491160285}
172.20.10.4 - - [26/Mar/2026 00:01:57] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:01:59', 'lat': 17.385134, 'lon': 78.330045, 'distance': 0.0184271416630149, 'cost': 0.0368542833260298}
172.20.10.4 - - [26/Mar/2026 00:01:59] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:01', 'lat': 17.385138, 'lon': 78.330091, 'distance': 0.0184271416630149, 'cost': 0.0368542833260298}
172.20.10.4 - - [26/Mar/2026 00:02:01] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:03', 'lat': 17.385161, 'lon': 78.330157, 'distance': 0.025883096971908767, 'cost': 0.05176619394381753}
172.20.10.4 - - [26/Mar/2026 00:02:03] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:05', 'lat': 17.385179, 'lon': 78.330217, 'distance': 0.032557201514550105, 'cost': 0.06511440302910021}
172.20.10.4 - - [26/Mar/2026 00:02:05] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:06', 'lat': 17.385192, 'lon': 78.330277, 'distance': 0.03908615213803286, 'cost': 0.07817230427606572}
172.20.10.4 - - [26/Mar/2026 00:02:06] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:09', 'lat': 17.385219, 'lon': 78.330337, 'distance': 0.046125413602170 86, 'cost': 0.0922508272043417 2}
172.20.10.4 - - [26/Mar/2026 00:02:09] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:10', 'lat': 17.385268, 'lon': 78.330389, 'distance': 0.05388008674852481, 'cost': 0.10776017349704962}
172.20.10.4 - - [26/Mar/2026 00:02:10] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:13', 'lat': 17.385292, 'lon': 78.330449, 'distance': 0.06078366764993094, 'cost': 0.12156733529986188}
172.20.10.4 - - [26/Mar/2026 00:02:13] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:14', 'lat': 17.385326, 'lon': 78.330505, 'distance': 0.06782681144203463, 'cost': 0.13565362288406926}
172.20.10.4 - - [26/Mar/2026 00:02:14] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:16', 'lat': 17.385342, 'lon': 78.330557, 'distance': 0.07362452451441003, 'cost': 0.14724904902882005}
172.20.10.4 - - [26/Mar/2026 00:02:16] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:19', 'lat': 17.38538, 'lon': 78.330616, 'distance': 0.08117777710612673, 'cost': 0.16235555421225345}
172.20.10.4 - - [26/Mar/2026 00:02:19] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:02:24', 'lat': 17.385428, 'lon': 78.330656, 'distance': 0.08799716524031659, 'cost': 0.17599433048063318}
```

Figure 6.4: Server-Side Data Processing

Figure 6.5 displays the map visualization generated by the system, showing the complete path travelled by the vehicle within the toll zone.

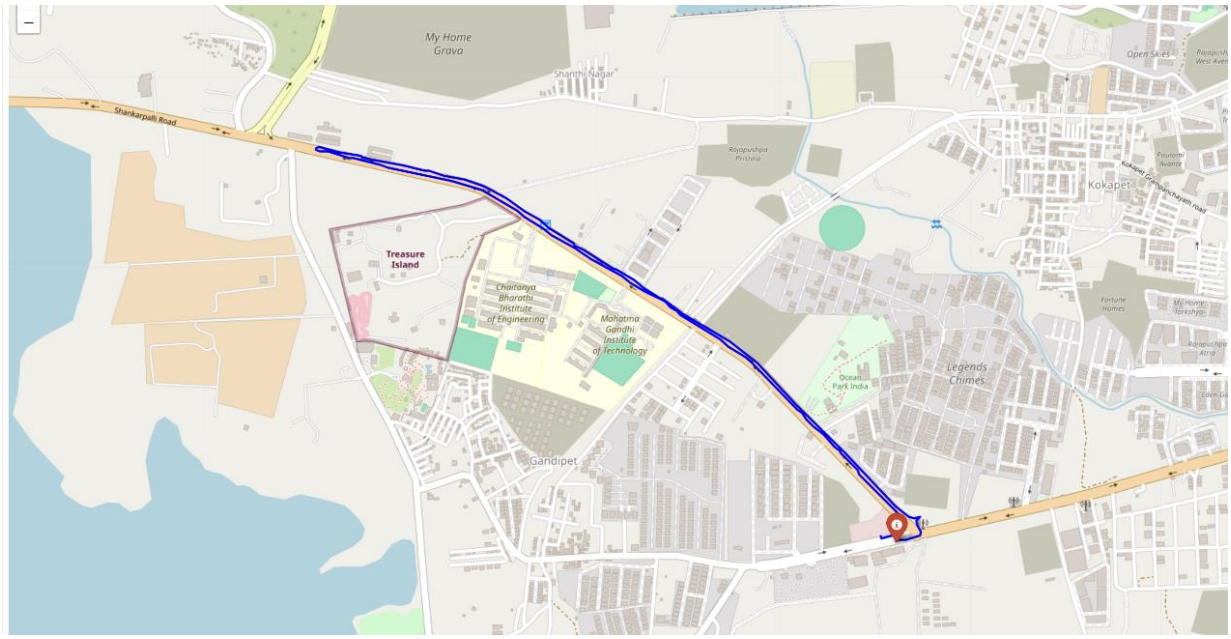


Figure 6.5: Map Visualization of Vehicle Path

Figure 6.6 shows the final toll calculation, where the system computes the total distance travelled and the corresponding toll amount dynamically.

```
172.20.10.4 - - [26/Mar/2026 00:40:22] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:23', 'lat': 17.385087, 'lon': 78.330348, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:23] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:25', 'lat': 17.385093, 'lon': 78.33034, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:25] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:27', 'lat': 17.385093, 'lon': 78.330335, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:27] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:29', 'lat': 17.385091, 'lon': 78.330331, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:29] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:31', 'lat': 17.385086, 'lon': 78.330324, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:31] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:32', 'lat': 17.385084, 'lon': 78.330315, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:32] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:34', 'lat': 17.385081, 'lon': 78.330306, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:34] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:36', 'lat': 17.385078, 'lon': 78.330299, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:36] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:38', 'lat': 17.385077, 'lon': 78.330295, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:38] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:41', 'lat': 17.385077, 'lon': 78.330291, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:41] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:42', 'lat': 17.385076, 'lon': 78.330288, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:42] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:44', 'lat': 17.385074, 'lon': 78.330285, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:44] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:45', 'lat': 17.385073, 'lon': 78.330283, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:45] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:48', 'lat': 17.385071, 'lon': 78.330281, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:48] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:50', 'lat': 17.385069, 'lon': 78.330278, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:50] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:51', 'lat': 17.385067, 'lon': 78.330276, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:51] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:53', 'lat': 17.385065, 'lon': 78.330274, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:53] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:55', 'lat': 17.385063, 'lon': 78.330272, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:55] "POST /location HTTP/1.1" 200 -
Stored: {'time': '00:40:57', 'lat': 17.385062, 'lon': 78.330271, 'distance': 4.485637814010065, 'cost': 8.97127562802013}
172.20.10.4 - - [26/Mar/2026 00:40:57] "POST /location HTTP/1.1" 200 -
```

Figure 6.6: Distance and Toll Calculation

The system was tested in real-world conditions by placing the Raspberry Pi setup inside a vehicle and driving through a predefined toll route. The results demonstrate that the system performs efficiently and accurately.

7. Conclusion and Future Scope

7.1 Conclusion

The **TollTrack** system presents an efficient and modern solution to the limitations of traditional toll collection methods. By leveraging **GPS technology, Raspberry Pi, and cloud-based processing**, the system successfully eliminates the need for physical toll plazas and enables fully automated toll collection. The implementation demonstrates real-time vehicle tracking, accurate distance calculation, and dynamic toll computation based on actual road usage.

One of the key achievements of this system is the introduction of distance-based tolling, which ensures fair pricing compared to fixed-rate toll systems. The integration of geofencing allows seamless detection of toll zone entry and exit without requiring any manual intervention. The system also reduces traffic congestion, fuel consumption, and operational costs associated with maintaining toll infrastructure.

The testing results confirm that the system operates reliably under real-world conditions, with acceptable GPS accuracy, minimal communication delay, and stable performance. The use of map visualization further enhances transparency by providing a clear representation of vehicle movement and toll calculation.

Overall, the **TollTrack** system proves to be a scalable and practical solution for next-generation toll collection and aligns with future **GNSS-based tolling initiatives**.

7.2 Future Scope

Although the current system achieves its core objectives, several improvements can be implemented to enhance its functionality and scalability.

- **Integration of Real-Time Communication Protocols:** Implementing MQTT instead of HTTP can improve communication efficiency and reduce latency.
- **Digital Payment Integration:** The system can be extended to support automatic toll deduction through UPI, wallets, or banking APIs for a complete end-to-end solution.
- **Advanced Geofencing and Map Matching:** Incorporating map-matching algorithms can improve accuracy and reduce errors caused by GPS drift or parallel road detection.
- **Multi-Vehicle Tracking System:** Future versions can support tracking multiple vehicles simultaneously with centralized monitoring dashboards.
- **Mobile and Web Application Development:** Developing user-friendly applications can allow users to track trips, view toll history, and receive notifications.
- **Enhanced GPS Accuracy Using Sensor Fusion:** Integration of additional sensors such as IMU can improve tracking accuracy in areas with weak GPS signals.
- **Cloud-Based Scalable Architecture:** Migrating the backend to cloud platforms like AWS or Azure can improve scalability, reliability, and data analytics capabilities.
- **AI-Based Dynamic Pricing:** Future systems can use AI to adjust toll rates based on traffic conditions, peak hours, and road usage patterns.

With these enhancements, the TollTrack system can evolve into a fully intelligent and large-scale toll management solution suitable for smart transportation systems.

References

1. Praneet More, Chirag Ranpise, et al., "GPS Based Toll System Simulation," International Journal of Advanced Research in Computer and Communication Engineering, 2025. https://ijarcce.com/wp-content/uploads/2025/04/IJARCCE.2025.14422.pdf
2. Manasi Gaikwad, Jagruti Gawade, et al., "A GPS-Driven Toll System Using Geofencing," IRJMETS, 2025. https://www.irjmets.com/uploadedfiles/paper/issue_3_march_2025/68621/final/fin_irjmets1741356975.pdf
3. Gaurav Tyagi, Kanika Paliwal, et al., "Toll Plaza Automation Using GPS," International Journal of Research Publication and Reviews, 2024. https://ijrpr.com/uploads/V5ISSUE4/IJRPR24771.pdf
4. Goutham K., Gowtham M., et al., "GPS Based E-Toll Gate Collection System," International Journal of Engineering Research & Technology, 2023. https://www.ijert.org/research/gps-based-e-toll-gate-collection-system-IJERTCONV10IS11039.pdf
5. R. Subha, Neelabru Pal, et al., "Smart Toll Collection: A GPS-Based Automated System," International Journal for Research in Applied Science & Engineering Technology, 2023. https://www.ijraset.com/fileserve.php?FID=36287
6. Pooja Nayak, Prakash, et al., "GPS Tracking System Using Raspberry Pi Zero Wireless," International Journal for Research in Applied Science & Engineering Technology, 2023. https://www.ijraset.com/research-paper/gps-tracking-system-using-raspberry-pi-zero-wireless
7. Jin Yeong Tan, Pin Jern Ker, et al., "Development of a GPS-Based Highway Toll Collection System," International Journal of Engineering Trends and Technology, 2016. https://www.researchgate.net/publication/315929517_Development_of_a_GPS-based_highway_toll_collection_system
8. IoT Research Community, "GPS Navigation and Toll Charging System," 2023. https://scholar.google.com/scholar?q=GPS+Navigation+and+Toll+Charging+System