

UNDERGRADUATE FINAL YEAR PROJECT REPORT.

Department of Computer and Information
Systems Engineering NED University of
Engineering and Technology.



LogiSync-Streamlining Last-Mile Logistics Through Map Matching Based Tracking for Logistics Companies

Group Number: 10

Batch: 2020

Group Member Names:

Abdul Raffay

CS-20094

Syed Ali Salar

CS-20067

Syed Ibrahim Noman

CS-20091

Syed Askari Abbas

CS-20306

Approved by

.....

Dr Shahab Tahzeeb Assistant Professor Project Advisor

Author’s Declaration

We declare that we are the sole authors of this project. It is the actual copy of the project that was accepted by our advisor(s) including any necessary revisions. We also grant NED University of Engineering and Technology permission to reproduce and distribute electronic or paper copies of this project.

| Signature and Date | Signature and Date | Signature and Date | Signature and Date |
|--------------------|--------------------|--------------------|--------------------|
| | | | |
| Abdul Raffay | Syed Ali Salar | Syed Ibrahim | Askari Abbas |
| CS-20094 | CS-20067 | Noman | CS-20306 |
| | | CS-20091 | |

raffay4300417@cloud.neduet.edu.pk
raffay4300417@cloud.neduet.edu.pk
raffay4300417@cloud.neduet.edu.pk
raffay4300417@cloud.neduet.edu.pk

Statement of Contributions

- ABDUL RAFFAY (CS-20094) has contributed in research, documentation, development of web application (company portal), data collection, LCSS algorithm development, algorithms analysis.
- SYED ALI SALAR (CS-20067) has contributed in research, documentation, development of mobile application (rider portal), data pre-processing, OSRM algorithm development, algorithms analysis.
- SYED IBRAHIM NOMAN (CS-20091) has contributed in research, documentation, development of backend application (server side development), algorithm integration, Valhalla algorithm development, algorithms analysis.
- ASKARI ABBAS(CS-20306) has contributed in research, documentation, development of mobile application (rider portal), algorithm integration, backend application (server side development), Snap To Nearest algorithm development.

Executive Summary

This project addressed the critical challenge of optimizing last-mile logistics for delivery services, which was plagued by inefficiencies such as inaccuracies in distance calculations, noisy GPS data, and gaps in location tracking. These issues not only compromised the ability of companies to accurately track delivery routes and driver behavior but also significantly impacted operational costs and customer satisfaction levels. By focusing on these operational inefficiencies, **LogiSync** aimed to revolutionize the delivery sector, offering solutions that promised to enhance cost efficiency, ensure timely deliveries, and facilitate data-driven decision-making processes.

In tackling these challenges, **LogiSync** employed a multifaceted approach that leveraged advanced map-matching algorithms and real-time GPS data from mobile devices, thereby eliminating the need for expensive hardware trackers. The project's methodology encompassed the implementation of several cutting-edge algorithms, including the Snap to Nearest, OSRM Matcher, Valhalla Matcher, and notably, the LCSS (Longest Common Subsequence) Matcher. The latter, through its sophisticated trajectory segmentation and similarity score mechanism, was identified as the most effective tool in accurately mapping GPS points to road links, thus significantly outperforming other evaluated algorithms.

The extensive analysis conducted on these map-matching algorithms revealed that the LCSS Matcher offered superior accuracy in aligning GPS data with actual road networks. This finding underscored the algorithm's capacity to provide precise tracking of delivery routes, thereby minimizing operational costs and enhancing delivery efficiency. Furthermore, **LogiSync** introduced two dedicated portals: the Rider Portal, a mobile application for capturing and transmitting GPS coordinates of delivery routes in real-time, and the Company Portal, designed to offer businesses comprehensive insights into the journeys of their riders, including path tracking, distance traveled, fuel compensation estimates, and advanced analytics for making data driven decisions.

LogiSync emerged as a pivotal innovation in the realm of delivery services, addressing long-standing industry challenges through the strategic application of advanced technological solutions. The project not only promised to reduce operational expenses and improve accuracy in delivery tracking but also set a new standard for scalability and adaptability in logistics operations.

Acknowledgments

We would like to acknowledge and appreciate the hard work, dedication and support of everyone who has helped us with this project. We would especially like to thank our mentor and Internal Dr. Shahab Tahzeeb who has been an exceptional guide and support throughout the project.

Moreover, we would also like to express our gratitude to Syed Hasnain Raza Zaidi and team Bazaar Technologies, who provided us with the knowledge and resources needed to make this project a possibility.

Contents

| | |
|--|----------|
| Chapter 1 | 1 |
| Introduction | 1 |
| 1.1. Overview: | 1 |
| 1.2. Background Information: | 1 |
| 1.3. Significance And Motivation | 1 |
| 1.3.1. Importance of Last Mile Logistics Optimization: | 1 |
| 1.3.2. Motivation for the Project: | 1 |
| 1.4. Aims And Objectives: | 2 |
| 1.5. Methodology: | 2 |
| 1.6. Report Outline: | 2 |
| Chapter 2 | 4 |
| Literature Review | 4 |
| 2.1. Overview: | 4 |
| 2.2. Map Matching: | 4 |
| 2.3. Types of Map Matching: | 4 |
| 2.3.1. Incremental Algorithms: | 4 |
| 2.3.2. Global Algorithms: | 4 |
| 2.3.3 Issues In Current Global Algorithms: | 5 |
| 2.3.4 Remedies For Issues In Current Global Algorithms: | 5 |
| 2.4 Snap To Nearest Map Matching Algorithm: | 6 |
| 2.4.1 Workflow of Snap To Nearest Map Matching Algorithm: | 6 |
| 2.4.1.1 Input Data: | 6 |
| 2.4.1.2 Distance Metric: | 6 |
| 2.4.1.3 Pairing Points: | 6 |
| 2.4.1.3 Snap Operation: | 6 |
| 2.4.1.4 Iteration (Optional): | 7 |
| 2.4.2 Features of Snap To Nearest Map Matching Algorithm: | 7 |
| 2.4.2.1 Distance Metric Flexibility: | 7 |
| 2.4.2.2 Customizable Tolerance Threshold: | 7 |
| 2.4.2.3 Iterative Refinement: | 7 |
| 2.4.3 Speed And Accuracy of Snap To Nearest Map Matching Algorithm: | 7 |
| 2.5 OSRM Map Matching Algorithm: | 8 |
| 2.5.1 OSRM Algorithm Features: | 8 |
| 2.5.1.1 Graph Representation: | 8 |

| | |
|---|----|
| 2.5.1.2 Contraction Hierarchies: | 8 |
| 2.5.1.3 Profiles: | 8 |
| 2.5.1.4 Restful API: | 8 |
| 2.5.2 OSRM Algorithm Workflow: | 9 |
| 2.5.2.1 Data Loading: | 9 |
| 2.5.2.2 Graph Construction: | 9 |
| 2.5.2.3 Routing Queries: | 9 |
| 2.5.2.4 Response Generation: | 9 |
| 2.5.3 Speed And Accuracy of OSRM Algorithm: | 9 |
| 2.6 Valhalla Map Matching Algorithm: | 10 |
| 2.6.1 Valhalla Algorithm Workflow: | 10 |
| 2.6.1.1 GPS Data Preprocessing: | 10 |
| 2.6.1.2 GPS Data Segmentation: | 10 |
| 2.6.1.3 Candidate Road Segments Identification: | 10 |
| 2.6.1.4 Probabilistic Matching: | 10 |
| 2.6.1.5 Optimization and Refinement: | 10 |
| 2.6.1.6 Output Generation: | 11 |
| 2.6.2 Speed And Accuracy of Valhalla Algorithm: | 11 |
| 2.7 LCSS Map Matching Algorithm: | 11 |
| 2.7.1 LCSS Algorithm Methodology: | 12 |
| 2.7.1.1 Small Segment Assumption: | 12 |
| 2.7.1.2 Segmentation for Accurate Matching: | 12 |
| 2.7.1.3 Avoiding Extremes: | 12 |
| 2.7.1.4 Segmentation Optimization Problem: | 12 |
| 2.7.2 LCSS Algorithm Framework: | 12 |
| 2.7.2.1 Initialization: | 13 |
| 2.7.2.2 Recursive Process: | 13 |
| 2.7.2.3 Stop Criterion: | 14 |
| 2.7.3 LCSS Algorithm Error Tolerance Mechanism: | 14 |
| 2.7.4 LCSS Algorithm Path Finding Workflow: | 14 |
| 2.7.5 LCSS Algorithm Trajectory Segmentation Workflow: | 15 |
| 2.7.6 Precision And Computational Efficiency Of LCSS Algorithm: | 17 |
| 2.7.7 Comparison With Other Map Matching Algorithms: | 18 |
| Chapter 3 | 20 |
| Algorithms Analysis | 20 |
| 3.1. Overview: | 20 |
| 3.2. Evaluation of Individual Algorithms: | 20 |



| | |
|--|----|
| 3.2.1 Snap To Nearest Algorithm Analysis: | 20 |
| 3.2.2 OSRM Algorithm Analysis: | 21 |
| 3.2.3 Valhalla Algorithm Analysis: | 22 |
| 3.2.4 LCSS Algorithm Analysis: | 23 |
| 3.3. Comparative Analysis Of All Algorithms: | 24 |
| 3.3.1 Comparative Analysis For Computational Efficiency: | 25 |
| 3.3.2 Comparative Analysis For Mapping Accuracy: | 26 |
| 3.4 Summary Table: | 26 |
| 3.5 Optimal Algorithm Selection: | 27 |
| 3.6 LCSS Algorithm Implementation: | 27 |
| 3.6.1 Plotting The Trajectory Trace: | 27 |
| 3.6.2 Building The Geofence Around The Trace: | 28 |
| 3.6.3 Downloading The Road Network: | 29 |
| 3.6.4 Matching The Trace With Road Network: | 30 |
| Chapter 4 | 31 |
| Implementation | 31 |
| 4.1. Overview: | 31 |
| 4.2. System Architecture: | 31 |
| 4.3. Development Environment: | 32 |
| 4.4. Database Design: | 32 |
| 4.5. Module Implementation: | 34 |
| 4.5.1 Flutter App: | 34 |
| 4.5.2 FASTAPI Backend: | 34 |
| 4.5.3. React Web Admin Portal: | 34 |
| 4.6. UI Design: | 35 |
| 4.7. Integration: | 40 |
| 4.8. Testing: | 41 |
| 4.9. Deployment: | 42 |
| Chapter 5 | 43 |
| Conclusion | 43 |
| 5.1. Summary: | 43 |
| 5.2. Achievements: | 43 |
| 5.3. Challenges and Limitations: | 43 |
| 5.4. Future Work: | 43 |

| | |
|---|--------|
| Figure 1 Step 1 of trajectory segmentation step of LCSS algorithm on an example trip | 16 |
| Figure 2 Steps 2 and 3 of trajectory segmentation step of LCSS algorithm on an example trip. | 17 |
| Figure 3 Precision And Computational Efficiency Graph of LCSS Algorithm. | 18 |
| Figure 4 Time Metrics Of Snap To Nearest Algorithm..... | 20 |
| Figure 5 Accuracy Metrics Of Snap To Nearest Algorithm. | 21 |
| Figure 6 Time Metrics Of OSRM Algorithm. | 22 |
| Figure 7 Accuracy Metrics Of OSRM Algorithm. | 22 |
| Figure 8 Time Metrics Of Valhalla Algorithm. | 23 |
| Figure 9 Accuracy Metrics Of Valhalla Algorithm. | 23 |
| Figure 10 Time Metrics Of LCSS Algorithm | 24 |
| Figure 11 Accuracy Metrics Of LCSS Algorithm..... | 24 |
| Figure 12 Comparative Time Analysis. | 25 |
| Figure 13 Comparative Accuracy Analysis | 26 |
| Figure 14 . Trace Of A Sample Journey. | 28 |
| Figure 15 Geofence Of Trace Of A Sample Journey. | 29 |
| Figure 16 Downloaded Map Of Sample Journey..... | 30 |
| Figure 17 Final Map Matched Path Of Sample Journey. | 30 |
| Figure 18 Use diagram of App..... | 32 |
| Figure 19 Use diagram of Admin Portal | 32 |
| Figure 20 Erd Diagram | 34 |
| Figure 21 Dataflow diagram | 35 |
| Table 1 Comparative Analysis of LCSS Algorithm With Other Algorithms..... | 19 |
| Table 2 Comparative Analysis of Algorithms | 27 |



F/SOP FYDP/09/00

List of Abbreviations

1. **API**: Application Programming Interface
2. **CSV**: Comma-Separated Values
3. **ERD**: Entity-Relationship Diagram
4. **FastAPI**: Fast, high-performance web framework for building APIs with Python
5. **GPS**: Global Positioning System
6. **LCSS**: Longest Common Subsequence
7. **MongoDB**: A NoSQL database known for its scalability and flexibility
8. **PK**: Primary Key
9. **FK**: Foreign Key
10. **UI**: User Interface
11. **UX**: User Experience
12. **DP** - Dynamic Programming
13. **GIS** - Geographic Information Systems
14. **HMM** - Hidden Markov Model
15. **JSON** - JavaScript Object Notation
16. **OSM** - OpenStreetMap
17. **OSRM** - Open Source Routing Machine
18. **REST** - Representational State Transfer
19. **T** - Iteration T in algorithms
20. **V** - Set of nodes in a road network
21. **E** - Set of links (edges) in a road network

United Nations Sustainable Development Goals

The Sustainable Development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including poverty, inequality, climate change, environmental degradation, peace and justice. There is a total of 17 SDGs as mentioned below. Check the appropriate SDGs related to the project.

- ☐ No Poverty
- ☐ Zero Hunger
- ☐ Good Health and Well being
- ☐ Quality Education
- ☐ Gender Equality
- ☐ Clean Water and Sanitation
- ☐ Affordable and Clean Energy
- ☐ Decent Work and Economic Growth
☒
- ☐ Industry, Innovation and Infrastructure
☒
- ☐ Reduced Inequalities
- ☐ Sustainable Cities and Communities
- ☐ Responsible Consumption and Production
☒
- ☐ Climate Action
- ☐ Life Below Water
- ☐ Life on Land



F/SOP FYDP/09/00

☐ Peace and Justice and Strong Institutions

☐ Partnerships to Achieve the Goals



Similarity Index Report

Following students have compiled the final year report on the topic given below for partial fulfillment of the requirement for Bachelor's degree in Computer and Information Systems Engineering.

Project Title: LogiSync-Streamlining Last Mile Logistics Through Map Matching Based Tracking For Logistics Companies.

| S. No. | Student Name | Seat Number |
|--------|--------------------|-------------|
| 1. | Abdul Raffay | CS-20094 |
| 2. | Syed Ali Salar | CS-20067 |
| 3. | Syed Ibrahim Noman | CS-20091 |
| 4. | Askari Abbas | CS-20306 |

This is to certify that Plagiarism test was conducted on complete report, and overall similarity index was found to be less than 20%, with maximum 5% from single source, as required.

Signature and Date

.....

Dr. Shahab Tehzeeb

Chapter 1

Introduction

1.1. Overview:

This chapter includes in depth background knowledge of our project. It also discusses the aims and objectives and the methodology of the project in brief.

1.2. Background Information:

The landscape of delivery services has undergone significant transformation over the past decade, largely driven by the exponential growth of e-commerce, food delivery, and courier services. Within this context, the optimization of last-mile logistics has emerged as a pivotal area of concern, marked by challenges including inaccuracies in distance calculations, the unreliability of GPS data, and gaps in location tracking. These inefficiencies not only escalate operational costs but also detract from customer satisfaction by impeding timely deliveries. 'LogiSync', a project initiated to address these challenges, stands at the forefront of leveraging advanced technological solutions to refine the accuracy and efficiency of last-mile delivery services.

1.3. Significance And Motivation

1.3.1. Importance of Last Mile Logistics Optimization:

The optimization of last-mile logistics is crucial for enhancing the efficiency and reliability of delivery services. It directly influences cost efficiency by minimizing unnecessary fuel consumption and optimizing delivery routes, thereby reducing overall operational expenses. Moreover, the precision in delivery operations fosters enhanced customer satisfaction by ensuring the timely arrival of goods, a critical determinant of consumer loyalty in the competitive delivery sector.

1.3.2. Motivation for the Project:

The motivation behind **LogiSync** stemmed from the pressing need to address the persistent challenges faced by the delivery industry. The project was conceived with the aim of developing a scalable, cost-effective solution capable of improving route accuracy and tracking efficiency without the reliance on expensive hardware solutions, thus offering a tangible value proposition for businesses operating within this space.

1.4. Aims And Objectives:

The primary aim of **LogiSync** is to revolutionize last-mile logistics by providing an accurate, reliable, and user-friendly solution for delivery route optimization. The main objectives of the project are:

- To minimize operational costs by leveraging real-time GPS data for efficient route tracking.
- To enhance the accuracy of delivery route tracking through the implementation of advanced map-matching algorithms.
- To ensure scalability of the solution to support both small and large-scale delivery operations.
- To improve decision-making processes through the provision of accurate, data-driven insights regarding delivery operations.

1.5. Methodology:

The methodology employed in the **LogiSync** project involved a comprehensive analysis of various map-matching algorithms, including the Snap to Nearest, OSRM Matcher, Valhalla Matcher, and the LCSS Matcher. The project harnessed real-time GPS data from mobile devices, combined with advanced computational techniques, to accurately map delivery routes. The LCSS Matcher, distinguished for its precision in aligning GPS data with actual road networks, was identified as the most effective algorithm for the project's objectives. This methodological approach enabled **LogiSync** to overcome the prevalent challenges of last-mile logistics optimization.

1.6. Report Outline:

There are five chapters in this report:

- The first chapter is named **Introduction** gives an overview of the whole project, the motivation and aim behind the development of this project along with a brief overview of the methodology and the modules present.
- The second chapter named **Literature Review** provides the process of map matching, types of map matching and detailed workflow of all the four researched map matching algorithms taken from the various sources of literature and notable publications.
- The third chapter named **Algorithms Analysis** provides detailed analysis of all the four researched map matching algorithms based on various metrics like the time they took to process one GPS data point and the degree of their

accuracy of mapping the GPS data points to road links accurately. This chapter also provides the implementation details of the selected algorithm that outperforms the other algorithms under consideration.

- The fourth chapter named **Application Architecture** gives a comprehensive overview of the entire application architecture which involves the implementation of the selected algorithm, communication of mobile app with backend services for capturing the real time GPS data points and finally the integration of the selected map matching algorithm in the application to provide various useful metrics and KPIs for the companies.
- The fifth chapter named **Conclusion** which provides insights on how this work can be taken further in the future to handle more complex map matching and route tracking problems.

Chapter 2

Literature Review

2.1. Overview:

This chapter includes the introduction of map matching along with its types and a comprehensive workflow of all the four map matching algorithms from various literature sources and notable publications.

2.2. Map Matching:

Map matching is a computational technique utilized to align observed geographic positions, typically derived from Global Positioning System (GPS) data, with the corresponding road network in digital maps. This process involves the identification and association of discrete GPS points, which may suffer from varying degrees of inaccuracies due to factors such as signal multipath, atmospheric conditions, and urban canyon effects, with the most probable paths along the road network that these points are intended to represent. The primary objective of map matching is to interpret the raw GPS data in the context of the geometric and topological structure of the road network, thereby transforming the noisy and potentially off-road GPS points into a coherent sequence of road segments that best approximate the actual route traversed by the vehicle or individual.

2.3. Types of Map Matching:

2.3.1. Incremental Algorithms:

Incremental algorithms start by selecting the matched link within an error ellipse of the trajectory points. Next, they search each following GPS point to find the best-matched path. Incremental algorithms are commonly used in mobile navigation applications because of their rapid realtime processing capability.

2.3.2. Global Algorithms:

Global algorithms build the best-matched path by considering all trajectory points as a whole (that is, by considering the segmentwise trajectory rather than individual GPS points, as is the case for incremental algorithms). This property of global algorithms

makes them well suited for large-scale GPS data processing.

2.3.3 Issues In Current Global Algorithms:

Current global map matching algorithms face several limitations and challenges impacting their efficiency and performance. The Segmentwise Trajectories Technique, for example, employs constant-segment lengths, which lack flexibility and present difficulties in identifying the optimal segmentation schema. Additionally, the Candidate Path Set Mechanism, crucial for ensuring algorithm correctness, faces challenges in maintaining an indeterminate optimal size. It requires a balance between being large enough to ensure accuracy and small enough to maintain computational efficiency. Moreover, even fixed-size candidate path sets can strain memory resources, directly influencing computational speed. These issues culminate in varied processing times for map matching, ranging significantly from approximately 0.2 to 2.0 seconds per point, highlighting the need for optimization in both performance and resource management within these algorithms.

2.3.4 Remedies For Issues In Current Global Algorithms:

The Trajectory Segmentation Based Map-Matching Algorithm introduces significant improvements over traditional global algorithms by addressing their core issues with innovative strategies and optimization models. It processes large-scale, high-resolution GPS data with a trajectory segmentation strategy that iteratively refines segments to achieve an optimal segmentation schema, ensuring both accuracy and efficiency. The optimization model minimizes the number of required trajectory segments while maintaining high similarity scores for best subpath matching, effectively combining matched subpaths to construct an accurate path for the entire trajectory. Utilizing the Longest Common Subsequence (LCS) for similarity scoring adds flexibility by allowing non-consecutive matches, and a curve similarity optimization model further reduces segment numbers while ensuring satisfactory similarity scores. Crucially, this approach eliminates the need for a candidate path set, significantly enhancing computational efficiency and addressing memory resource concerns, thereby offering a more streamlined and effective solution for map matching challenges.

2.4 Snap To Nearest Map Matching Algorithm:

The Snap to Nearest Algorithm is a computational method designed to associate or map points in a dataset to their nearest neighbors based on a specified metric. This algorithm is widely used in various applications, such as geographic information systems (GIS), image processing, computer graphics, and data analysis. The primary goal of the Snap to Nearest Algorithm is to efficiently identify and pair points in a dataset with their closest counterparts. This process is particularly useful when working with spatial data or datasets where proximity relationships play a crucial role.

2.4.1 Workflow of Snap To Nearest Map Matching Algorithm:

2.4.1.1 Input Data:

The algorithm begins with a dataset containing points (GPS data points in our case) that need to be associated with their nearest neighbors (road network links in our case). Each point in the dataset has coordinates (x, y) or other relevant attributes.

2.4.1.2 Distance Metric:

A distance metric is defined to measure the proximity between points. Common metrics include Euclidean distance, Manhattan distance, or other specialized metrics depending on the nature of the dataset. In our case the distance metric is the point to line distance i.e the distance between the input GPS data point and the road network link.

2.4.1.3 Pairing Points:

For each point in the dataset, the algorithm calculates the distance to all other points using the chosen metric. The point is then paired with its nearest neighbor based on the calculated distances.

2.4.1.3 Snap Operation:

Once the nearest neighbors are identified, a "snap" operation occurs, where the original point is moved or associated with the coordinates of its nearest neighbor. This snapping process ensures that points are closely aligned or matched with their closest counterparts.

2.4.1.4 Iteration (Optional):

In some cases, the snapping process may be iterated to refine the associations further. This is particularly useful when dealing with complex datasets or situations where multiple iterations enhance the accuracy of the snap-to-nearest operation.

2.4.2 Features of Snap To Nearest Map Matching Algorithm:

2.4.2.1 Distance Metric Flexibility:

The algorithm often allows users to choose from a variety of distance metrics, such as Euclidean distance, Manhattan distance, or other specialized metrics. This flexibility enables customization based on the specific characteristics of the dataset.

2.4.2.2 Customizable Tolerance Threshold:

Users can often set a tolerance threshold to control the snapping distance. This feature is valuable when dealing with datasets where precise point alignment is not critical, and a certain level of deviation is acceptable.

2.4.2.3 Iterative Refinement:

Some implementations of the algorithm offer an iterative refinement process. This allows users to perform multiple snap-to-nearest iterations, enhancing the accuracy of point associations by iteratively adjusting the coordinates based on updated nearest neighbors.

2.4.3 Speed And Accuracy of Snap To Nearest Map Matching Algorithm:

The Snap To Nearest Algorithm is recognized for its speed and simplicity, offering a rapid means of aligning GPS data points with the nearest road segments on a digital map. This efficiency stems from its straightforward approach of merely identifying the closest road segment for each point without delving into complex analyses of road connectivity or travel patterns. However, this simplicity comes at the cost of accuracy, especially when compared to more sophisticated map-matching algorithms like the LCSS Matcher. These advanced algorithms take into account additional factors such as the direction of travel, speed, and the connectivity of the road network, enabling them to resolve ambiguities and snap points to the most plausible routes, even in dense urban

environments with closely spaced parallel roads.

In contrast, the Snap To Nearest Algorithm's reliance on proximity alone makes it susceptible to inaccuracies in areas where GPS data might be misleading or when the actual path taken does not correspond to the nearest road segment.

2.5 OSRM Map Matching Algorithm:

The OSRM (Open Source Routing Machine) Matcher algorithm is a robust routing engine designed for efficient route calculations based on OpenStreetMap (OSM) data. This section provides an overview of the key features and the working mechanism of the OSRM Matcher.

2.5.1 OSRM Algorithm Features:

The OSRM Matcher algorithm is built to provide accurate and fast routing information. Here's an overview of its key components and functionality:

2.5.1.1 Graph Representation:

The algorithm constructs a graph representation of the road network using OpenStreetMap data. Nodes represent intersections, and edges represent road segments with associated weights, such as travel times or distances.

2.5.1.2 Contraction Hierarchies:

The OSRM employs the Contraction Hierarchies technique for graph optimization. This involves simplifying the graph by contracting less important nodes, reducing the complexity of route calculations. Shortcuts are added to preserve shortest path information.

2.5.1.3 Profiles:

The algorithm supports different profiles to cater to various transportation modes, such as car, bicycle, and pedestrian. Each profile considers specific attributes like road types, speed limits, and accessibility, allowing for customization based on user preferences.

2.5.1.4 Restful API:

OSRM provides a RESTful API, allowing developers to send HTTP requests to the OSRM server for routing information. The API returns responses in standard formats

like JSON, containing details such as route geometry, turn-by-turn directions, and additional information.

2.5.2 OSRM Algorithm Workflow:

Below has been given the detailed and comprehensive workflow of the OSRM algorithm:

2.5.2.1 Data Loading:

The algorithm relies on OpenStreetMap data to build the road network. Users can provide specific data sets or access the extensive OSRM database.

2.5.2.2 Graph Construction:

The road network is represented as a graph, and preprocessing steps like contraction hierarchies are applied for optimization.

2.5.2.3 Routing Queries:

When a routing query is received through the API, the algorithm performs a bidirectional Dijkstra search on the preprocessed graph. The search meets at a middle point, and the algorithm calculates the shortest path based on edge differences.

2.5.2.4 Response Generation:

The algorithm generates a response containing route details, including geometry, turn instructions, and other relevant information.

2.5.3 Speed And Accuracy of OSRM Algorithm:

OSRM (Open Source Routing Machine) is highly efficient, especially for routing and map matching over large datasets. While it is generally fast, the complexity of the routing graph and the need for accurate matching can sometimes slow it down compared to simpler algorithms like Snap To Nearest. OSRM (Open Source Routing Machine) is highly efficient, especially for routing and map matching over large datasets. While it is generally fast, the complexity of the routing graph and the need for accurate matching can sometimes slow it down compared to simpler algorithms like Snap To Nearest.

Provides high accuracy by using a graph-based approach similar to Valhalla, considering both the geometry of road networks and travel directions. It's particularly effective in accurately matching routes in complex road networks.

2.6 Valhalla Map Matching Algorithm:

The Valhalla Matcher is a sophisticated tool designed for map matching that leverages the power of Valhalla, an open-source routing engine. It integrates spatial analysis and probabilistic modeling to align GPS data points with the most probable paths on a digital map, taking into account various factors such as road attributes, vehicle speed, and direction.

2.6.1 Valhalla Algorithm Workflow:

Below has been given the detailed and comprehensive workflow of the Valhalla algorithm:

2.6.1.1 GPS Data Preprocessing:

GPS data points are normalized to reduce noise and correct anomalies. Then the data points are sampled at regular intervals to ensure consistency and manage computational load.

2.6.1.2 GPS Data Segmentation:

The GPS trajectory is segmented based on significant changes in direction or speed, to simplify the matching process.

2.6.1.3 Candidate Road Segments Identification:

For each GPS point, a spatial query identifies nearby road segments within a certain radius, considering the point's accuracy.

2.6.1.4 Probabilistic Matching:

Each candidate road segment is evaluated using a cost function that accounts for distance from the GPS point, the segment's direction relative to the movement direction, and other road attributes. A graph traversal algorithm then explores the most likely paths through the network of candidate road segments.

2.6.1.5 Optimization and Refinement:

The algorithm employs dynamic programming (DP) or a Hidden Markov Model

(HMM) to refine the path, ensuring it is the most probable given the sequence of GPS points. Then a smoothing algorithm may be applied to the chosen path to eliminate unnecessary turns and ensure a realistic trajectory.

2.6.1.6 Output Generation:

The final step involves constructing the matched path, which is output as a sequence of road segments, complete with detailed routing information suitable for navigation or analysis.

2.6.2 Speed And Accuracy of Valhalla Algorithm:

Designed for efficiency, Valhalla leverages a fast routing engine and is optimized for processing large volumes of data quickly. Its graph-based approach enables rapid computation of routes by dividing the route into tiles and process those tiles parallelly, which can enhance its speed compared to some traditional algorithms.

Valhalla Matche also offers high accuracy by considering a wide range of factors, including road attributes, vehicle dynamics, and directional flow. Its comprehensive approach allows for more precise matching compared to simpler algorithms like Snap To Nearest.

2.7 LCSS Map Matching Algorithm:

The trajectory segmentation map-matching algorithm also known as the LCSS algorithm is proposed to deal accurately and efficiently with large-scale, high-resolution GPS trajectory data. The new algorithm separates the GPS trajectory into segments. Then it find the shortest path for each segment in a scientific manner and ultimately generates a best-matched path for the entire trajectory.

The similarity of a trajectory segment and its matched path is described by a similarity score system based on the longest common subsequence. The numerical experiment indicated that the proposed map-matching algorithm was very promising in relation to accuracy and computational efficiency. Large-scale data set applications verified that the proposed method is robust and capable of dealing with real-world, large-scale GPS data in a computationally efficient and accurate manner.

2.7.1 LCSS Algorithm Methodology:

The algorithm breaks down a vehicle's trajectory into segments to determine the most accurate overall matched path. It operates under the assumption that any sufficiently small segment of the vehicle trajectory will have its matched path as the shortest path between the same start and end points of the segment. If a segment does not match the shortest path, the algorithm considers that the segment may not be small enough and that there might be a better match available by further breaking down that segment into sub segments. Below are given the key points of the algorithms methodology:

2.7.1.1 Small Segment Assumption:

Small enough segments of a trajectory are presumed to match the shortest path between their start and end points.

2.7.1.2 Segmentation for Accurate Matching:

The trajectory is divided into segments that are likely to follow the shortest path, balancing between too many tiny segments and no segmentation at all.

2.7.1.3 Avoiding Extremes:

The algorithm avoids the extremes of processing each pair of consecutive points as separate segments (inefficient) and not segmenting at all (potentially inaccurate).

2.7.1.4 Segmentation Optimization Problem:

The method seeks to minimize the number of segments while ensuring that each segment's similarity score meets a predefined threshold, thereby optimizing computational efficiency and accuracy.

2.7.2 LCSS Algorithm Framework:

The method begins by finding the shortest path between the cleaned motorized trip GPS trajectory start and end points. The LCS-based similarity score is used to compare the similarity of the identified path to the GPS points' trajectory. If the similarity score of the trajectory sequence is high and the identified path matches the trajectory sequence well, the algorithm stops. Otherwise, the original trajectory is divided into segments, and a trajectory segmentation scheme is initiated. At the next iteration, the trajectory with the new segmentation scheme will be fed into the recursive process, which comprises path-finding and trajectory segmentation procedures. The algorithm

continues until the new segmentation scheme is identical to the previous iteration. When the algorithm stops, the final map-matched path is obtained from the segment matched paths. The definitions for the algorithm are given below:

1. **G**: Road network $G = \{V, E\}$, where **V** is the set of nodes and **E** is the set of links—edge.
2. **traj0**: Original trajectory with **N** GPS track points; $\text{traj0} = \{p1, p2, \dots, pN\}$, where $p_i = (\text{loni}, \text{lati})$ are the GPS track point coordinates.
3. **trajT**: Trajectory segment sequence scheme with **M** segments at iteration **T**; $\text{trajT} = \{S1, S2, \dots, SM\}$.

The main steps include initialization, a recursive process, and a stop criterion. The details are given below:

2.7.2.1 Initialization:

The original trajectory **traj0** and road network **G** are prepared, and an initial trajectory-matching procedure is conducted in this step. A shortest path is generated according to the start and end nodes of **traj0**. If the similarity score (defined in the section on path finding) of the shortest path and **traj0** is higher than a threshold, the best-matched path is determined and the algorithm stops. Otherwise, **traj0** will be divided into segments, and the recursive process is triggered.

2.7.2.2 Recursive Process:

The input of the trajectory segment sequence at iteration **T** is **trajT**. The recursive process includes two substeps:

- a. **Path finding**: An LCS similarity score-based path-finding method is used to find the corresponding shortest path with the best similarity score for each segment. If the segment path similarity score is greater than a threshold δ (for example, $\text{score} \geq \delta = 0.95$), the segment is marked as a similar segment. Otherwise, the segment is marked as a dissimilar segment.
- b. **Trajectory segmentation**: A trajectory segmentation method is applied to all segments either to separate each dissimilar segment into subsegments according to similarity characteristics and cutting points or to skip the similar segments. A score validation procedure is used to guarantee that the similarity score improves because of the segmentation procedure. If the score validation fails, the targeted dissimilar segment stays the same. These steps end up generating a new trajectory segmentation scheme **trajT+1**.

2.7.2.3 Stop Criterion:

The stop criterion is that the trajectory segmentation scheme does not change in consecutive iterations (i.e., $\text{trajT} = \text{trajT}+1$).

2.7.3 LCSS Algorithm Error Tolerance Mechanism:

An additional error tolerance mechanism is applied in the algorithm for those unmatchable GPS trajectory segments. When the trajectory segment cannot find a matched path, the algorithm will skip this segment and restart on the next segment, which means the path-finding procedure will take the start and end point locations of the next available trajectory segment as the origin and destination for path searching.

2.7.4 LCSS Algorithm Path Finding Workflow:

The procedure first finds the network end node and start node sets according to the endpoints of the network links that are spatially close to the trajectory segment end and start point locations. The start node set is usually selected as the previous end node of the segmentmatched path to keep the connectivity of the matched path except for first-segment situations or the map matching restarting from a malfunction situation. The method computes a group of candidate shortest paths from all start nodes to all end nodes using Dijkstra's shortest path algorithm. Among the candidate shortest path group, the method calculates a similarity score for each candidate shortest path and selects the path with the highest similarity score.

The similarity score describes the similarity of a GPS trajectory segment to its corresponding path. For example, assume a track point $\mathbf{p_i} = (\mathbf{x}, \mathbf{y})\mathbf{i}$ from the trajectory segment $\mathbf{S_n} = \{\mathbf{p1}, \mathbf{p2}, \dots, \mathbf{pN}\}$ and a $\mathbf{linkj} = (\mathbf{x_s}, \mathbf{y_s}, \mathbf{x_e}, \mathbf{y_e})\mathbf{j}$ from the path $\mathbf{P_m} = \{\mathbf{link1}, \mathbf{link2}, \dots, \mathbf{linkM}\}$. The spatial proximity of point $\mathbf{p_i}$ to \mathbf{linkj} is described by a \mathbf{simobj} function. A function value greater than zero indicates that some degree of similarity is obtained.

$$\text{simobj}(p_i, \text{link}_j) = \begin{cases} 0 & \text{dist}(p_i, \text{link}_j) > \epsilon \\ 1 - \frac{\text{dist}(p_i, \text{link}_j)}{\epsilon} & \text{otherwise} \end{cases}$$

Eq(3.1)

where $\text{dist}(\mathbf{p_i}, \mathbf{linkj})$ represents the point-to-line distance between $\mathbf{p_i}$ and \mathbf{linkj} and ϵ is

the maximal distance threshold for determining how similar two objects are. The **simobj** function returns a value between **0** and **1**. The **zero** value indicates two objects are not similar at all. A value close to **one** indicates that two objects are very similar.

A dynamic programming model is used to calculate the LCS score of the trajectory segment **Sn** and the path **Pm**. Let **Sn(i)** denote the first **i** points of **Sn** such that **Sn(i) = {p1, p2, . . . , pi}**, where $i \leq N$, and let **Pm(j)** denote the first **j** links of path **Pm**, such that **Pm(j) = {link1, link2, . . . , linkj}**, where $j \leq M$. A recursive function **scoreLCS(Sn(i), Pm(j))** is defined to obtain the LCS score for those two elements. When **i = 0** or **j = 0**, **scoreLCS(Sn(i), Pm(j)) = 0**. If **i ≠ 0** and **j ≠ 0** then:

$$\begin{aligned} \text{scoreLCS}(S_n(i), P_m(j)) = \\ \max(\text{scoreLCS}(S_n(i-1), P_m(j-1)) + \text{simobj}(p_i, l_j), \\ \text{scoreLCS}(S_n(i-1), P_m(j)), \text{scoreLCS}(S_n(i), P_m(j-1))) \end{aligned} \quad \text{Eq(3.2)}$$

The function **scoreLCS** returns a similarity score of the best match between **Sn** and **Pm** up to their **ith** and **jth** objects. The LCS score for the full sequence is thus given as **scoreLCS(Sn(N), Pm(M))**. The **scoreLCS** function value is the summation of all **simobj** values for all matched object pairs. Because the **simobj** value is between **0** and **1**, if one assumes the two sequences are exactly matched.

The longest common subsequence is one of the two sequences. It means the **scoreLCS** value would not be greater than either **M** or **N**. Then, a sequence similarity score function **simseq** is as defined below:

$$\text{simseq}(S_n, P_m) = \frac{\text{scoreLCS}(S_n(N), P_m(M))}{\min(M, N)} \quad \text{Eq(3.3)}$$

The **simseq** function value is a normalized value between 0 and 1. The larger value represents more similarity between the trajectory and the path.

2.7.5 LCSS Algorithm Trajectory Segmentation Workflow:

The Trajectory Segmentation part of the Algorithm optimizes map-matching by

breaking down a vehicle's trajectory into smaller, more manageable segments, allowing for a more accurate matched path. The methodology is based on the principle that segments with low similarity scores are further divided using cutting points derived from the trajectory's distance profile. These cutting points are identified either as points of largest distance to the path or points near a distance threshold, indicating a deviation from the matched path. The summarized methodology includes:

1. Dividing dissimilar trajectory segments into subsegments at cutting points based on distance profiles.
2. Finding the shortest path for each subsegment to potentially improve the match.
3. Combining all subpaths to form a new path for the trajectory.
4. Retaining the subsegment division only if the new path has a higher similarity score than the old path otherwise the dissimilar segment remains the same.
5. Maintaining similar segments as is, while updating the overall path based on new segmentation schemes.
6. Stopping the algorithm when further segmentation does not improve similarity scores.

The algorithm is demonstrated through figure 1 to 3 showing the iterative process of segmentation and path matching, where segments are adjusted until all have satisfactory matches, at which point the algorithm converges.

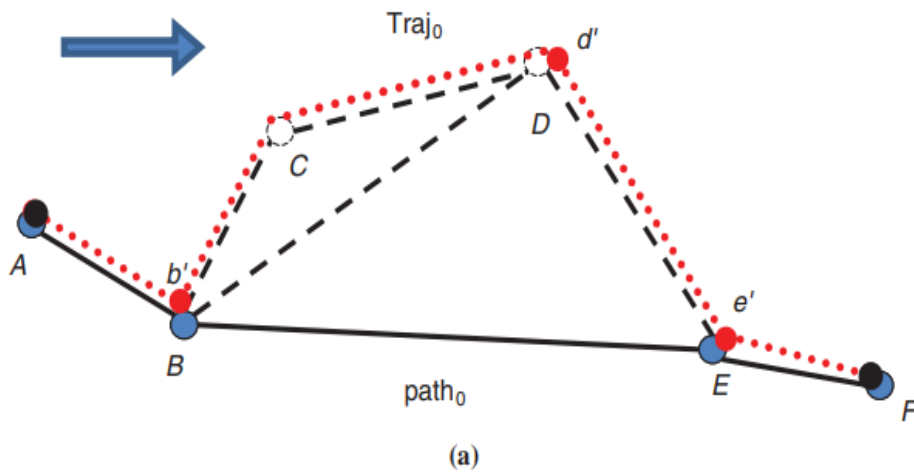


Figure 1 Step 1 of trajectory segmentation step of LCSS algorithm on an example trip

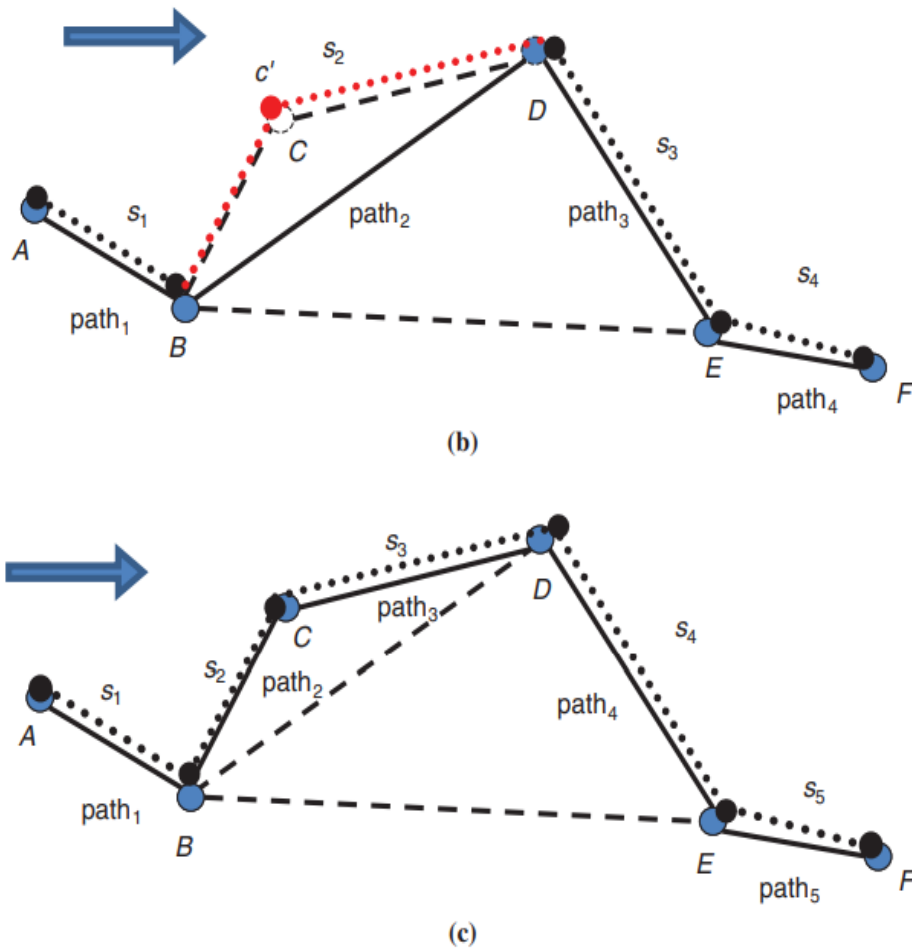


Figure 2 Steps 2 and 3 of trajectory segmentation step of LCSS algorithm on an example trip.

2.7.6 Precision And Computational Efficiency Of LCSS Algorithm:

Figure 3 shows that as the similarity score threshold decreases, the precision quality drops and computational efficiency improves. The computational time improves substantially with steady reductions in the similarity score threshold from 0.99 to 0.95. Further decrease in the threshold below 0.95 continues to improve the computational time but at a slower rate. At a threshold of 0.95, the computational time is 0.0275 s per point, and the precision quality is 96.6%. The results also show that average precision quality decreases as the similarity score threshold decreases, although precision quality levels out between 94.5% and 95% at threshold values for the similarity score below 0.5.

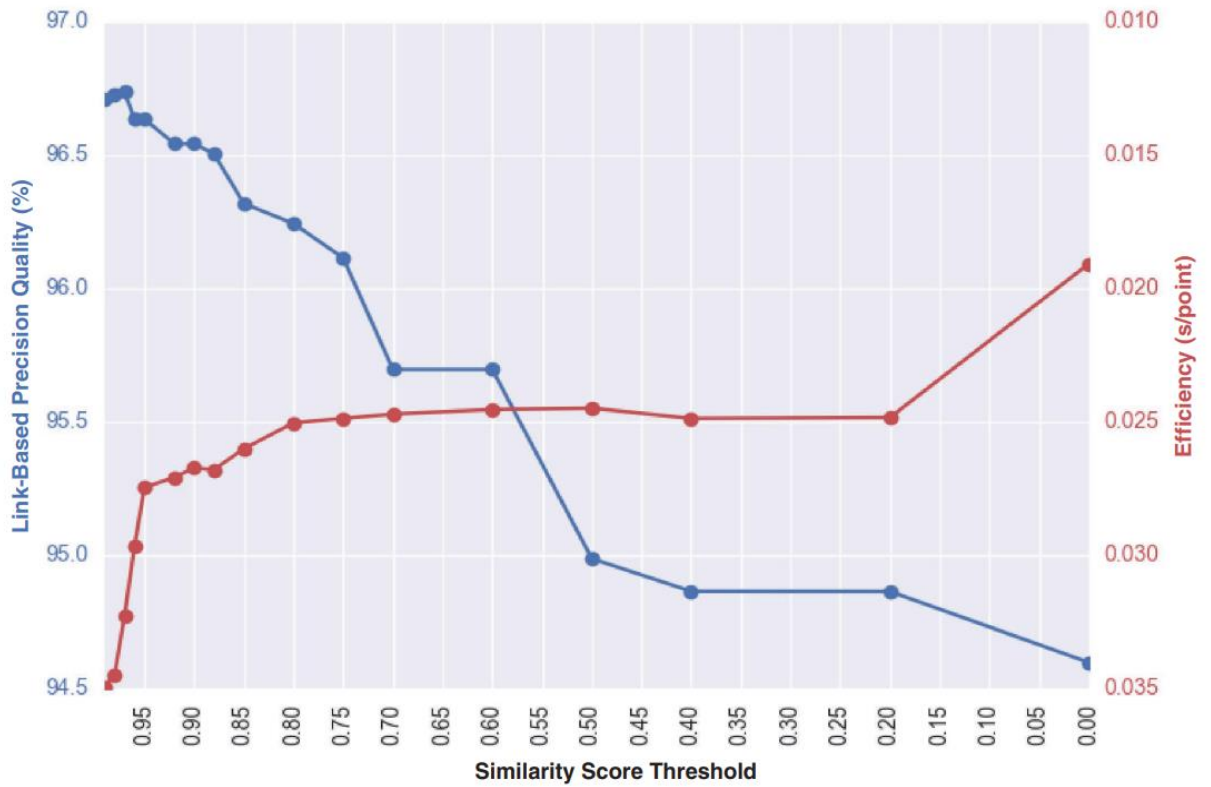


Figure 3 Precision And Computational Efficiency Graph of LCSS Algorithm.

2.7.7 Comparison With Other Map Matching Algorithms:

The LCSS similarity score based trajectory segmentation map-matching algorithm was compared with other benchmarks for computational efficiency and precision quality.

The **snap-to-nearest** method is one of the simplest map-matching algorithms, by which the GPS point is just snapped to the closest road network link. It is a computationally efficient algorithm but not the most accurate one.

The **Hidden Markov Model (HMM)** approach to the map-matching problem is a more recent solution that provides more accurate results than the snap-to-nearest approach. The results for each of these algorithms applied to the test data set are listed in table below. The snap-to-nearest method shows the best computational efficiency, but its precision quality is much worse than that for the other methods. The results for the current LCS method are given for two similarity score thresholds to show the impact of this parameter on the comparison of computational efficiency and precision quality. Even at a similarity score threshold setting of **0%**, the LCS method shows **3%** higher precision quality and **27%** higher computational speed than the **HMM** method. Raising the similarity score threshold to **95%** results in roughly equal computational speed but a full **5%** higher precision quality for the LCS method relative to the HMM method. The results are given in table 1 given below.



F/SOP FYDP/09/00

| Method | Computational Time | Link Based Precision Quality (%) |
|---------------------------|--------------------|----------------------------------|
| Snap To Nearest | 0.002 s / point | 78 |
| Hidden Markov Model (HMM) | 0.026 s / point | 91.5 |
| LCSS, 0 % | 0.019 s / point | 94.6 |
| LCSS, 95 % | 0.027 s / point | 96.6 |

Table 1 Comparative Analysis of LCSS Algorithm With Other Algorithms

Chapter 3

Algorithms Analysis

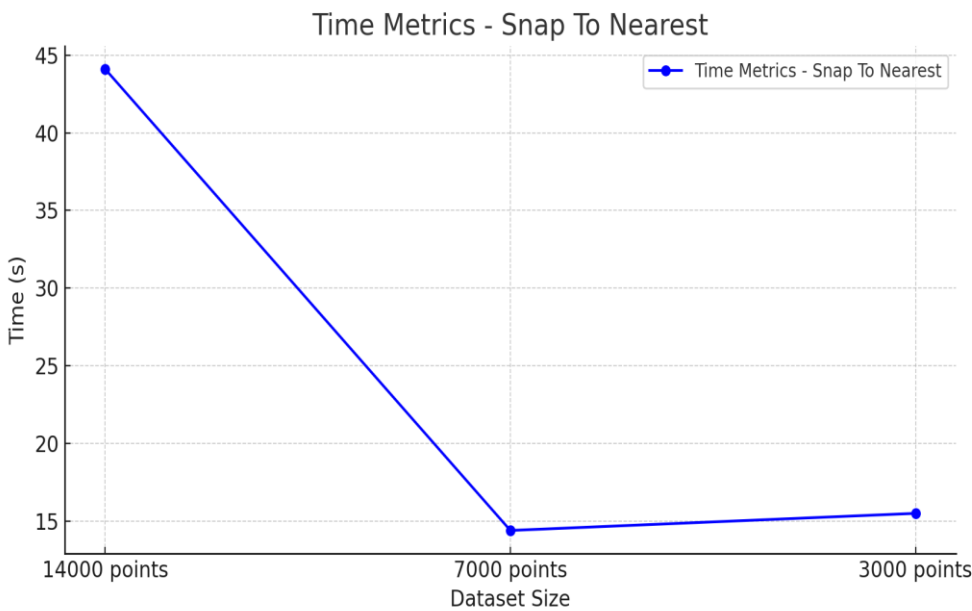
3.1. Overview:

This chapter includes a comprehensive evaluation of four researched map matching algorithms which are Snap To Nearest, OSRM Matcher, Valhalla Matcher, and LCSS Matcher against critical performance metrics such as computational efficiency (measured in processing time per GPS data point) and mapping accuracy (the precision in aligning GPS data points with the corresponding road links). The analysis utilizes datasets sourced from Open Street Map, which have undergone extensive preprocessing to eliminate noise and outliers.

3.2. Evaluation of Individual Algorithms:

3.2.1 Snap To Nearest Algorithm Analysis:

The Snap To Nearest Algorithm demonstrates superior computational speed across as shown in the figure 4 various datasets, indicative of its operational efficiency in real-



time applications.

Figure 4 Time Metrics Of Snap To Nearest Algorithm

Despite its expedited processing capability, the algorithm's accuracy is compromised in densely populated urban areas, where the proximity of road networks presents a significant challenge. The accuracy metrics has been analyzed in the figure 5.

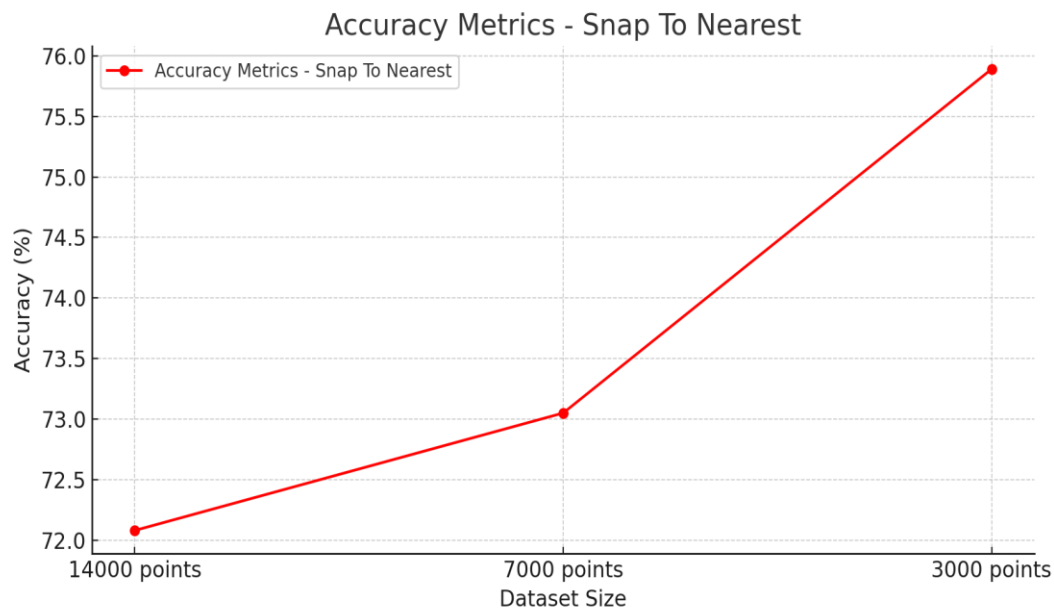


Figure 5 Accuracy Metrics Of Snap To Nearest Algorithm.

3.2.2 OSRM Algorithm Analysis:

The OSRM Matcher shows a balanced profile in processing speed as shown in the figure 6, optimizing both efficiency and accuracy, making it suitable for applications with moderate complexity.

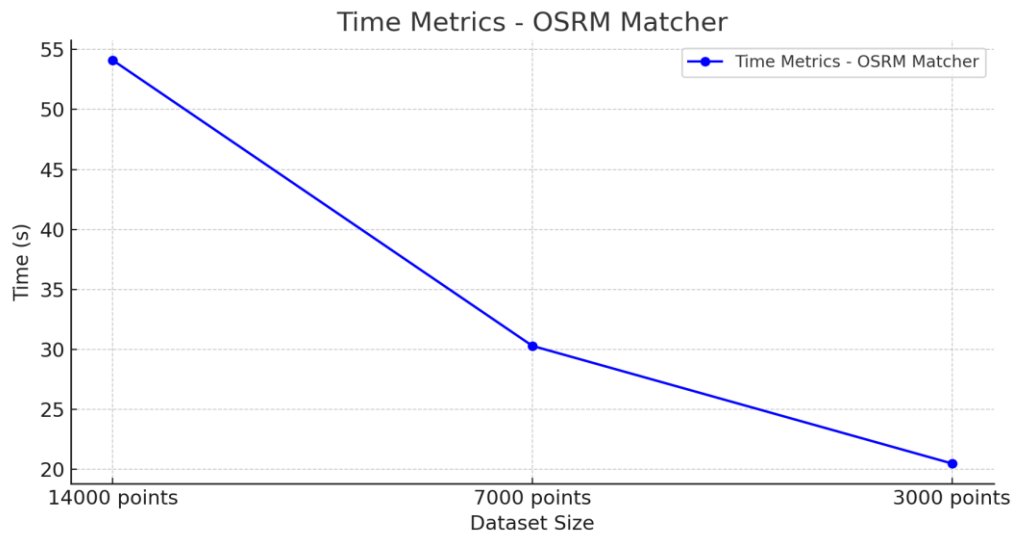


Figure 6 Time Metrics Of OSRM Algorithm.

Moreover, the OSRM algorithm significantly enhances the mapping accuracy over simpler algorithms like the Snap To Nearest Algorithm, demonstrating its utility in complex navigation scenarios. The mapping accuracy metrics has been analyzed in the figure 7.

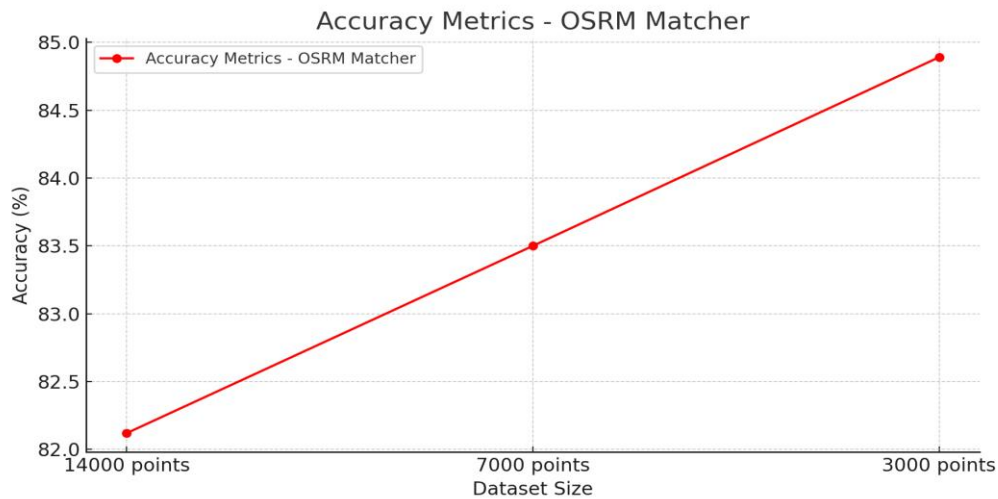


Figure 7 Accuracy Metrics Of OSRM Algorithm.

3.2.3 Valhalla Algorithm Analysis:

The Valhalla Matcher is slightly less efficient than the OSRM Matcher in terms of processing speed, as it prioritizes delivering high-fidelity results, particularly in challenging mapping environments. The time metrics of Valhalla Matcher tested on a

range of GPS data points is given in the figure 8 given below:

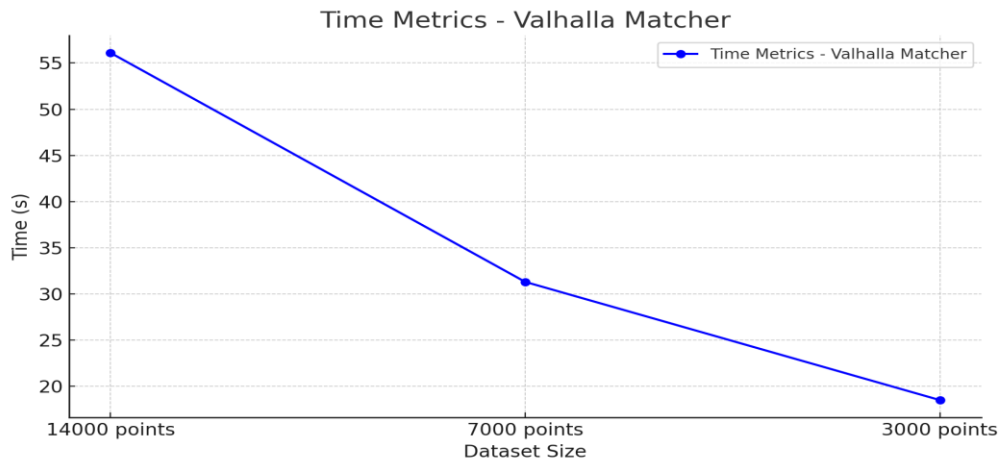
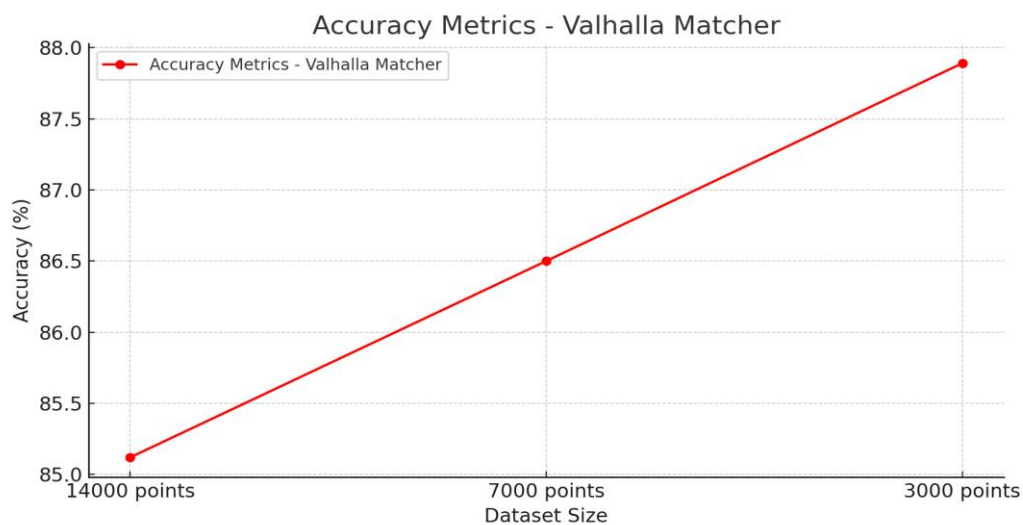


Figure 8 Time Metrics Of Valhalla Algorithm.

However, regarding the accuracy point of view the valhalla matcher achieves a



significantly high accuracy than the snap to nearest and OSRM algorithms, the results are shown in figure 9 given below.

Figure 9 Accuracy Metrics Of Valhalla Algorithm.

3.2.4 LCSS Algorithm Analysis:

The LCSS Matcher, though the least efficient in terms of processing speed, offers substantial gains in mapping accuracy. The time metrics of the LCSS Algorithm tested on a range of GPS data points is given in the figure 10 given below:

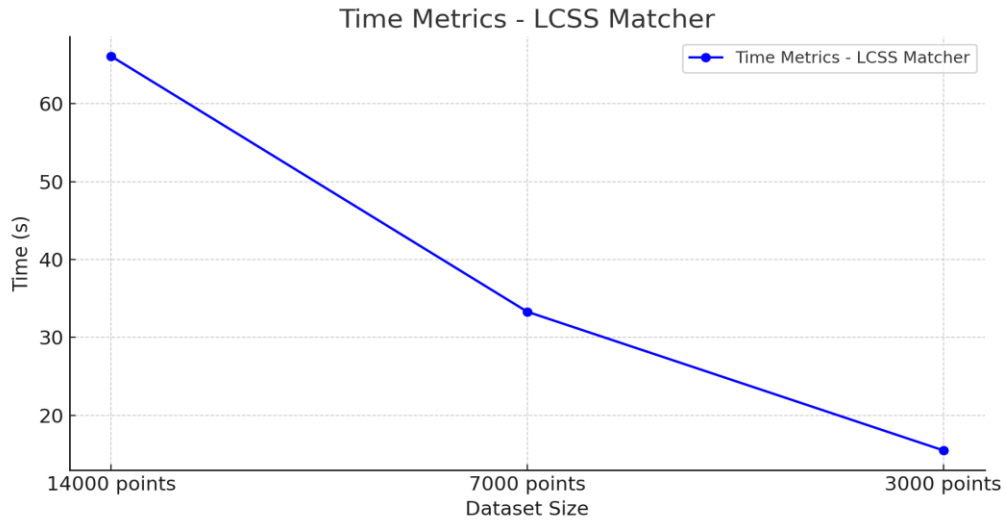


Figure 10 Time Metrics Of LCSS Algorithm

However from the mapping accuracy perspective the unparalleled accuracy of the LCSS algorithm proves the sophistication of its algorithmic approach, setting a new standard in map matching precision. The mapping accuracy metrics has been analyzed in the figure 11.

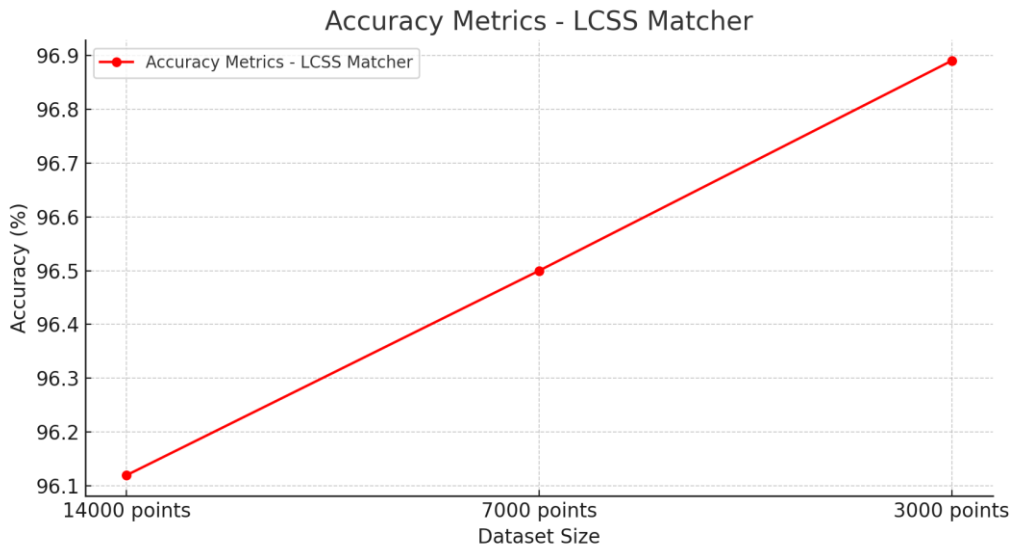


Figure 11 Accuracy Metrics Of LCSS Algorithm.

3.3. Comparative Analysis Of All Algorithms:

Now a comprehensive overall analysis of the computational efficiency and the mapping accuracy of all the four algorithms are given in the following sections.

3.3.1 Comparative Analysis For Computational Efficiency:

A comparative examination of the computational efficiency of all the four algorithms highlights the inherent trade-off between processing speed and mapping accuracy.

The Snap To Nearest Algorithm emerges as the most efficient, but at the expense of low mapping precision, while the LCSS Matcher proves to be quiet slower than other algorithms but it has the highest mapping accuracy. The comparative time analysis of algorithms is shown in figure 12 given below:

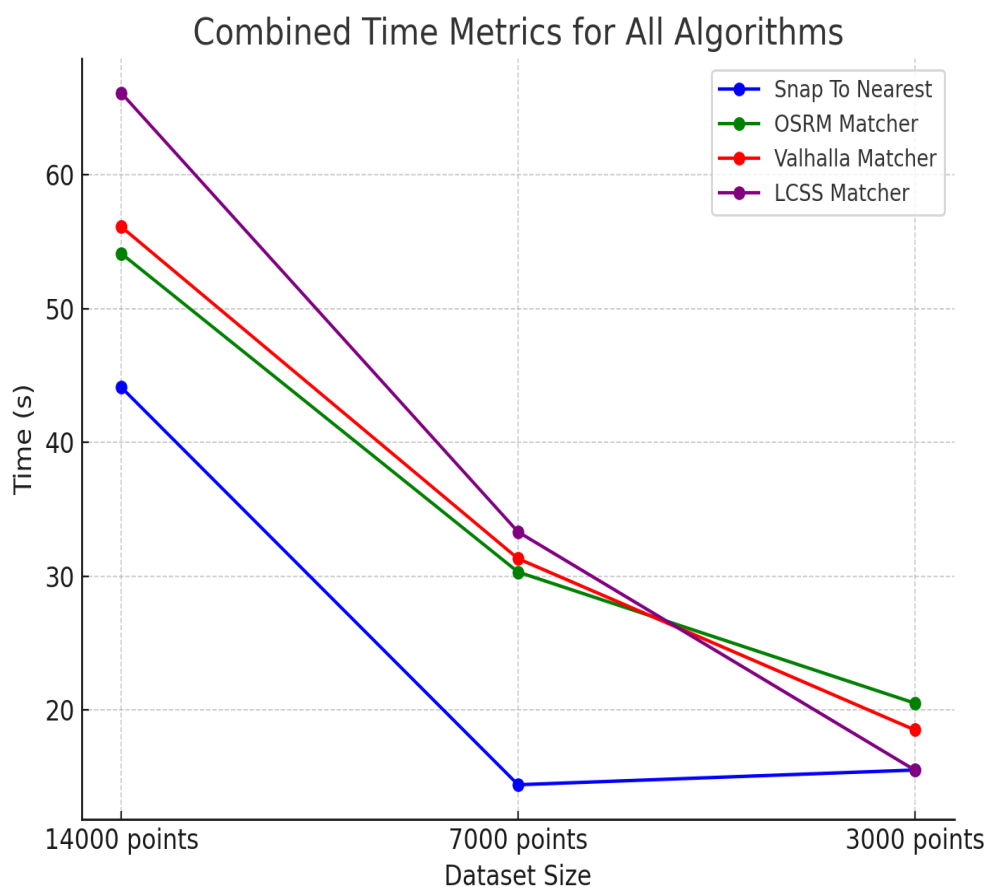


Figure 12 Comparative Time Analysis.

3.3.2 Comparative Analysis For Mapping Accuracy:

The accuracy comparison distinctly sets the LCSS Matcher as the most accurate algorithm in terms of mapping precision, showing its significance in applications where the integrity of map matching is paramount.

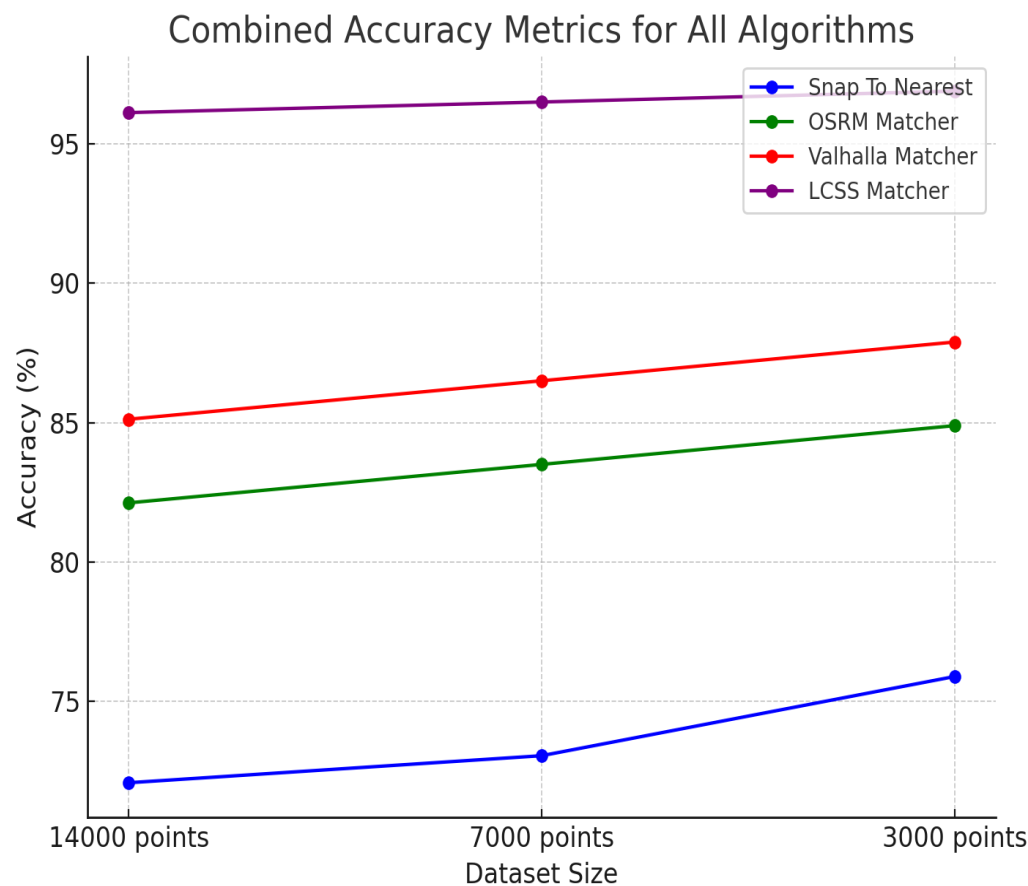


Figure 13 Comparative Accuracy Analysis

3.4 Summary Table:

A summary table presenting a side-by-side comparison of the processing time and accuracy metrics offers an at-a-glance reference for evaluating the performance of each algorithm across the datasets. The analysis results in the tabular format is shown in the table 2 given below:

| Algorithm Name | Time Taken For 14,000 points(sec) | Time Taken For 7,000 points(sec) | Time Taken For 3,000 points(sec) | Accuracy For 14,000 points(%) | Accuracy For 7,000 points(%) | Accuracy For 3,000 points(%) |
|------------------|-----------------------------------|----------------------------------|----------------------------------|-------------------------------|------------------------------|------------------------------|
| Snap To Nearest | 44.1 | 14.4 | 15.5 | 72.08 | 73.05 | 75.89 |
| OSRM Matcher | 54.1 | 30.3 | 20.5 | 82.12 | 83.50 | 84.89 |
| Valhalla Matcher | 56.1 | 31.3 | 18.5 | 85.12 | 86.50 | 87.89 |
| LCSS Matcher | 66.1 | 33.3 | 15.5 | 96.12 | 96.50 | 96.89 |

Table 2 Comparative Analysis of Algorithms

3.5 Optimal Algorithm Selection:

Based on the comprehensive individual and comparative analysis of the four algorithms. The LCSS algorithm proves to be the most performant algorithm in terms of the mapping accuracy metric and because of this the LCSS algorithm has been chosen for implementation in this project.

3.6 LCSS Algorithm Implementation:

There are a few steps involved in the implementation of the LCSS Algorithm, the steps include plotting the trajectory trace, building the geofence, downloading the map, and finally performing matching of trace with the downloaded map. The steps are briefly described in the below sections:

3.6.1 Plotting The Trajectory Trace:

This is the first step of the implementation of the LCSS algorithm, in this step the input data which is a series of GPS co-ordinates is prepared and pre-processed so that it can be mapped on the map to see the sequence of raw GPS data points on the map. A trace of a sample journey of around 20 GPS data points is shown in the figure 14.

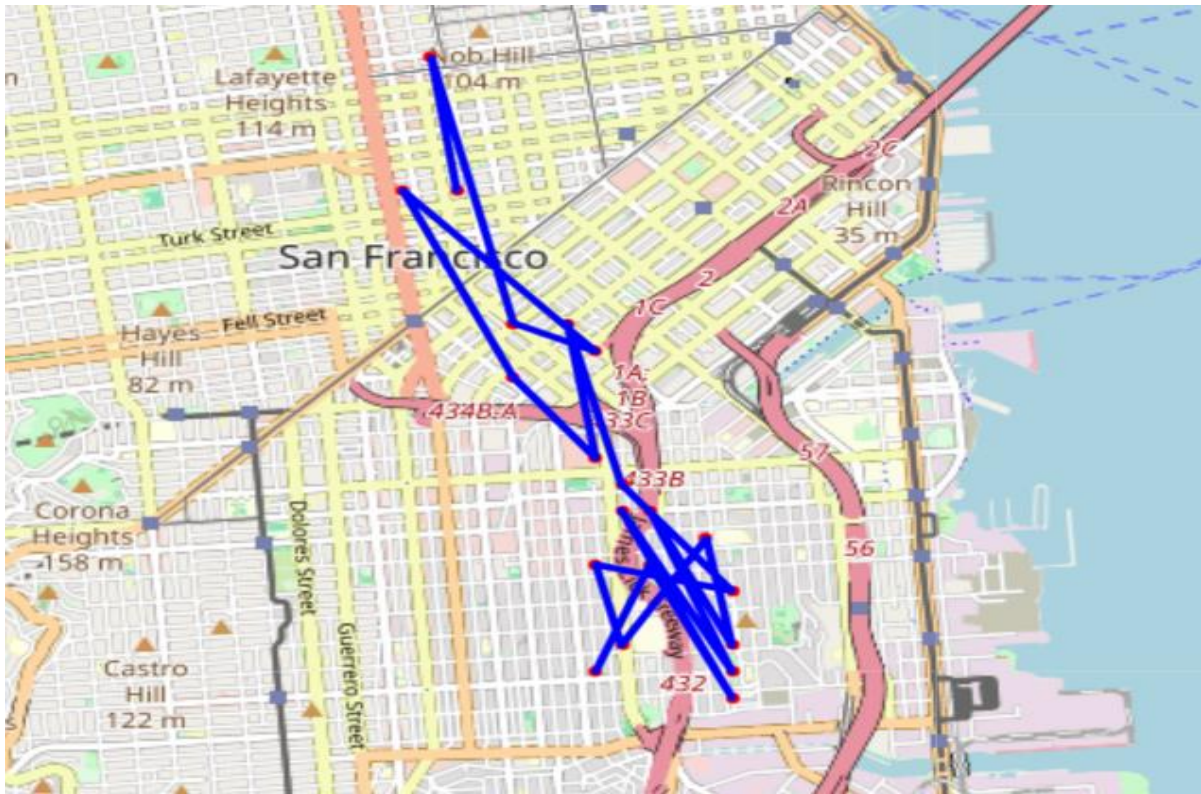


Figure 14 . Trace Of A Sample Journey.

The red points in the figure 14 are showing the raw GPS data points that have been mapped on the map in order to visualize the input data.

3.6.2 Building The Geofence Around The Trace:

The next step is to build a small geofence around the trace that we have mapped in the first step, this step is necessary as we will be downloading the physical road network of the geofence including the trace and not the entire physical road network map which would be very costly in terms of memory consumption. We can also control the radius of the geofence in this step based on the requirements and nature of the input data that we have. The plotted geofence of the journey of step 1 is shown in the figure 15 given below.

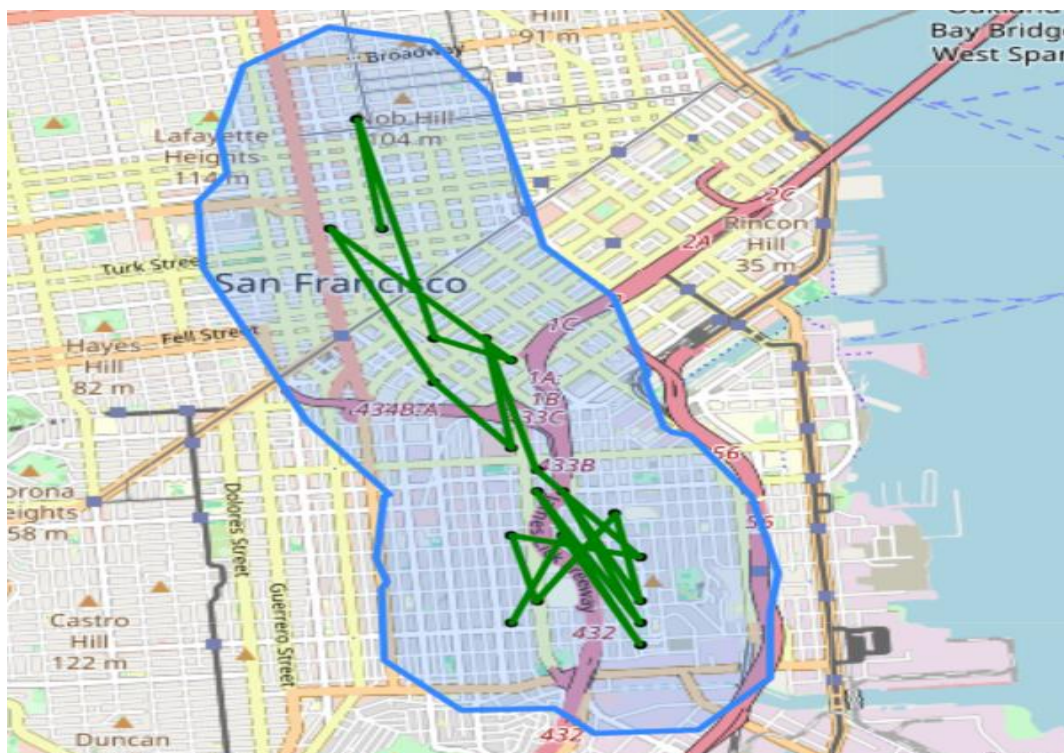


Figure 15 Geofence Of Trace Of A Sample Journey.

3.6.3 Downloading The Road Network:

After the geofence has been plotted, the next step is to download the road network only for the plotted geofence. The downloaded map of the journey of step 1 is shown in the figure 16 given below.



Figure 16 Downloaded Map Of Sample Journey

3.6.4 Matching The Trace With Road Network:

The final step in the implementation of the LCSS algorithm is to finally the map the input GPS trajectory trace with the download road network using the LCSS algorithm. The final matched path plot is shown in the figure 17 given below.

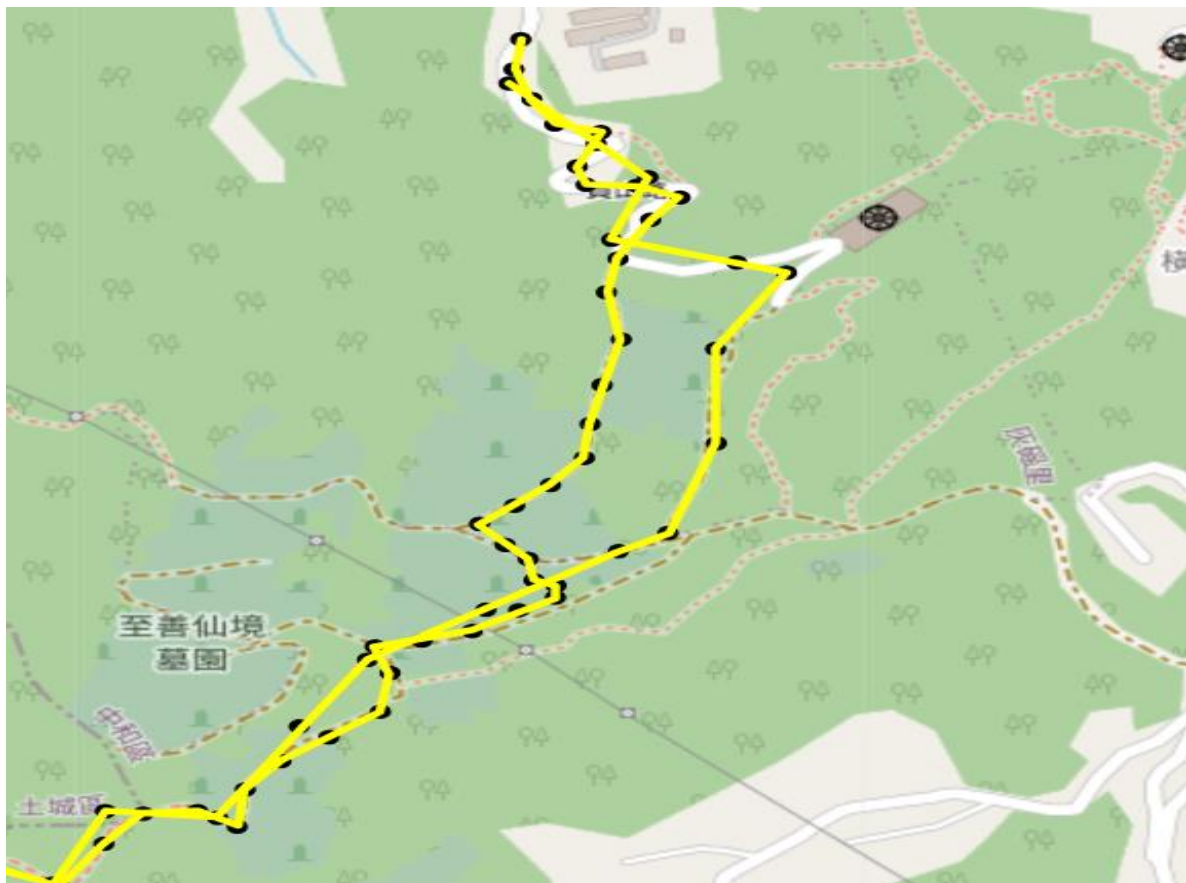


Figure 17 Final Map Matched Path Of Sample Journey.

Chapter 4

Implementation

4.1. Overview:

This chapter details the implementation of the LogiSync project, a comprehensive solution designed to optimize last-mile logistics. It includes the system architecture, development environment, database design, implementation of App and Web portal, their integration, testing, deployment. Each section provides insights into the methodologies and technologies used to achieve the project's objectives.

4.2. System Architecture:

The system architecture of LogiSync comprises three main components:

- 1- Flutter-based mobile app,
- 2- FastAPI backend server,
- 3- React-based web portal.

Our mobile app generates GPS pings at five-second intervals during a rider's journey, saving them in a CSV file for further processing.

These pings CSV file is sent to the backend FASTAPI, which processes the data using the LCSS algorithm to create a journey map and measure the distance covered.

The results are then transmitted to the web portal for analysis. The Use diagram illustrates the interactions.

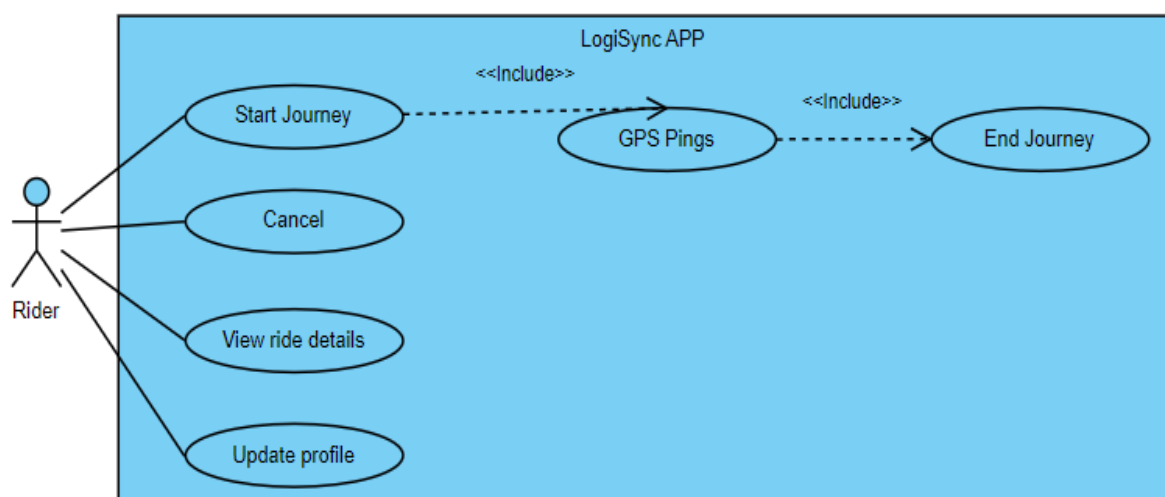


Figure 18 Use diagram of App

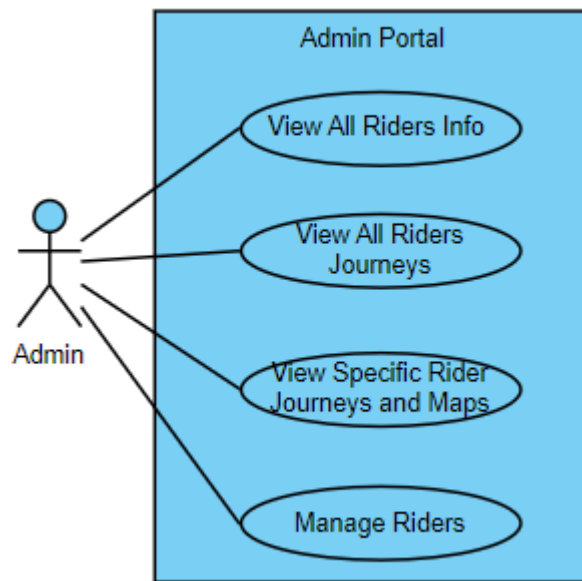


Figure 19 Use diagram of Admin Portal

4.3. Development Environment:

The development environment for LogiSync includes various software resources.

- 1- Our mobile app is developed using Flutter, exploiting its cross-platform capabilities.
- 2- The backend server is built with Python FastAPI for its high performance, ease of use and creation of APIS.
- 3- The web portal is developed using React to provide a responsive and dynamic user interface, Single page application.

Additional tools, technologies and platforms such as Github, Postman and Swagger UI are also utilized to ensure a robust and scalable solution. We have also utilized libraries provided by these Technologies such as for navigation we use LatLong and Geolocator for navigation.

4.4. Database Design:

We choose MongoDB as our database due to its strong support for real-time applications and flexibility in managing unstructured data. MongoDB's document-oriented storage model is well-suited for storing GPS pings, journey data, and user information efficiently. The ability of scaling horizontally and its powerful querying abilities make MongoDB an excellent choice for our real-time data needs.



F/SOP FYDP/09/00

Activities MongoDB Compass 01:38 30 جولائی 96%

MongoDB Compass - oi-e-commerce.wbllexg.mongodb.net/logisync.users

Connect Edit View Collection Help

oi-e-commerce... Documents logisync.users +

My Queries Databases Search

admin local logisync journeys users lunda_bazaar

logisync.users

20 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } Explain Reset Find Options

ADD DATA EXPORT DATA 1 - 20 of 20

```
{ "_id": "ObjectID('668a4ff2624a9e0f72dc548e')", "fullname": "Abdul Raffay", "email": "raffay@gmail.com", "password": "12b5121swGmYnbyGuX21VndSnJue0erIZat3mM2yBn2L.U18BR07.Z5Bj0", "phone": "031544125478", "role": "rider", "createdAt": "2024-07-07T13:21:06.488+00:00", "updatedAt": "2024-07-07T13:21:06.488+00:00" }, { "_id": "ObjectID('668a59e4e6f51965be606811')", "fullname": "Ali Khan", "email": "alikh@gmail.com", "password": "12b5121swGmYnbyGuX21VndSnJue0erIZat3mM2yBn2L.U18BR07.Z5Bj0", "phone": "03154425478", "role": "rider", "createdAt": "2024-07-07T14:03:32.245+00:00", "updatedAt": "2024-07-07T14:03:32.245+00:00" }, { "_id": "ObjectID('668a59fee6f51965be606812')", "fullname": "Sara Achakzai", "email": "sara@gmail.com", "password": "12b5121swGmYnbyGuX21VndSnJue0erIZat3mM2yBn2L.U18BR07.Z5Bj0", "phone": "03154435478", "role": "rider", "createdAt": "2024-07-07T14:03:32.245+00:00", "updatedAt": "2024-07-07T14:03:32.245+00:00" }
```

MONGODB

Activities MongoDB Compass 01:38 30 جولائی 96%

MongoDB Compass - oi-e-commerce.wbllexg.mongodb.net/logisync.journeys

Connect Edit View Help

oi-e-commerce... Documents logisync.journeys +

My Queries Databases Search

admin local logisync journeys users lunda_bazaar

logisync.journeys

39 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } Explain Reset Find Options

ADD DATA EXPORT DATA 1 - 20 of 39

```
{ "_id": "ObjectID('668a04945108cc3d9d0f0856a')", "rider_id": "668a4ff2624a9e0f72dc548e", "createdAt": "2024-07-07T18:13:56.767+00:00", "updatedAt": "2024-07-07T18:13:56.767+00:00" }, { "_id": "ObjectID('6694f213afc172736685d8b')", "rider_id": "668a5a1ae6f51965be606813", "createdAt": "2024-07-15T14:55:31.889+00:00", "updatedAt": "2024-07-15T14:55:31.889+00:00" }, { "_id": "ObjectID('66a6a89c2fb9674c24fb9c8f')", "rider_id": "66a6288aed3f43f94101a5e9", "createdAt": "2024-07-29T01:22:30.137+00:00", "updatedAt": "2024-07-29T01:22:30.137+00:00" }, { "_id": "ObjectID('66a6a8ac2fb9674c24fb9d1c')", "rider_id": "66a6288aed3f43f94101a5e9", "createdAt": "2024-07-29T01:23:08.313+00:00", "updatedAt": "2024-07-29T01:23:08.313+00:00" }
```

MONGODB

Activities MongoDB Compass 01:37 30 جولائی 96%

MongoDB Compass - oi-e-commerce.wbllexg.mongodb.net/logisync

Connect Edit View Help

oi-e-commerce... Collections

My Queries Databases Search

admin local logisync journeys users lunda_bazaar

Create collection Refresh View Sort by Collection Name

| Collection | Storage size | Documents | Avg. document size | Indexes | Total index size |
|------------|--------------|-----------|--------------------|---------|------------------|
| journeys | 20.48 kB | 39 | 99.00 B | 1 | 36.86 kB |
| users | 20.48 kB | 20 | 220.00 B | 1 | 36.86 kB |

MONGODB

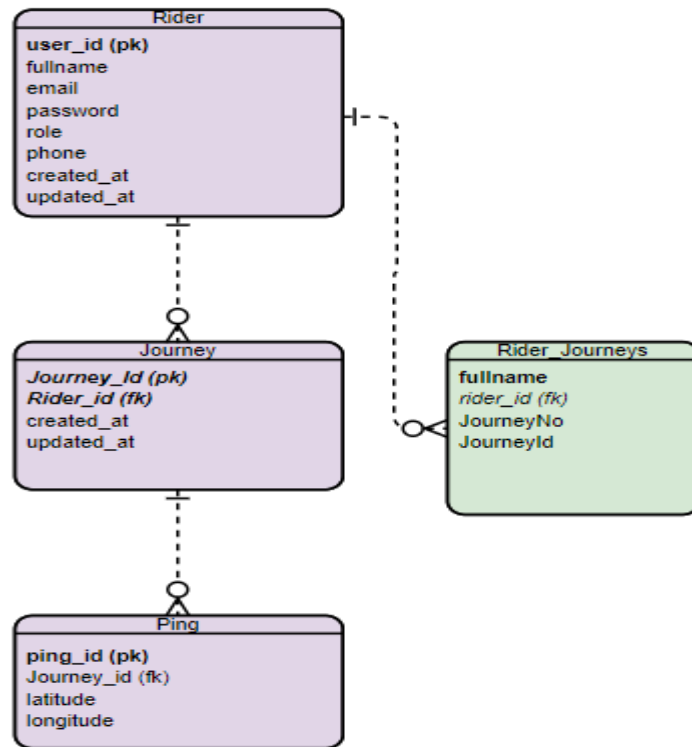


Figure 20 Erd Diagram

4.5. Module Implementation:

Each module of LogiSync is implemented to fulfill specific functionalities.

4.5.1 Flutter App:

First of all, the Flutter app module is responsible for generating and saving GPS pings. The Rider login to our App and would start journey. After journey starts, the app generates pings of current location of rider with timestamp and it would be save in a CSV. After completion of journey, the CSV is sent for further processing to our Backend FastAPI.

4.5.2 FASTAPI Backend:

The FastAPI backend processes these pings using the LCSS algorithm to create journey maps and measure rider covered distances. Based on `Journey_id`, separate maps are generated for each rider's journey and all are sent to our Web Admin Portal.

4.5.3. React Web Admin Portal:

The React web portal module displays the journey data and analysis. Admin can login into the portal and can view all rider's journeys and can manage riders. Separate maps and metrics for each journey are also displayed for view for Admin.

Pictorial representation of the data flow diagram, is provided to illustrate the implementation.

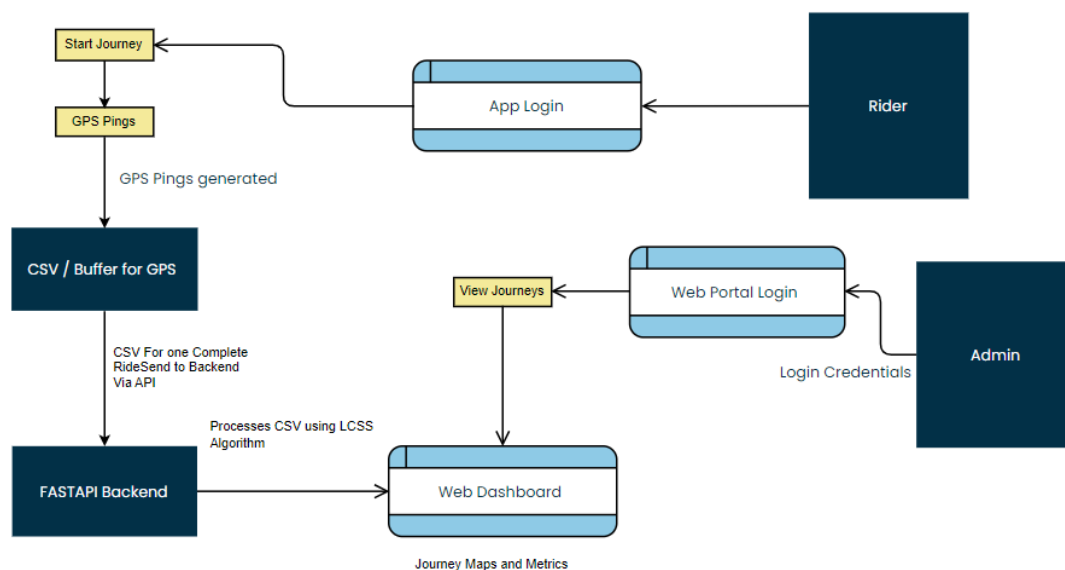


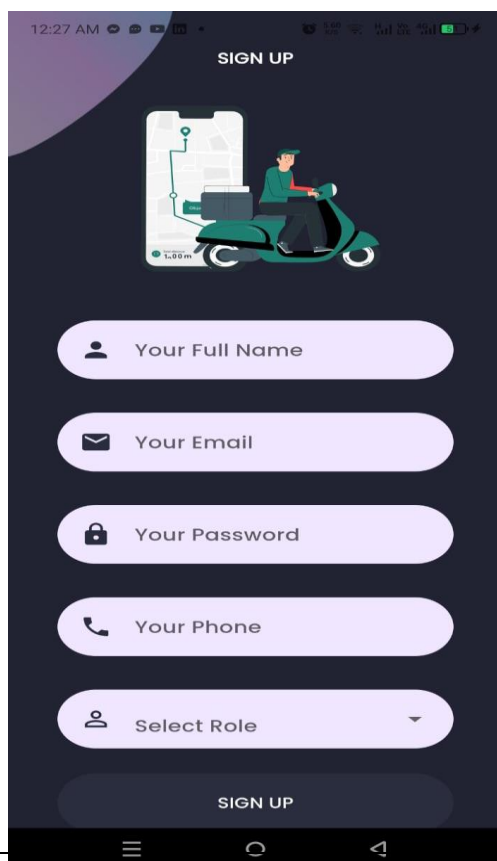
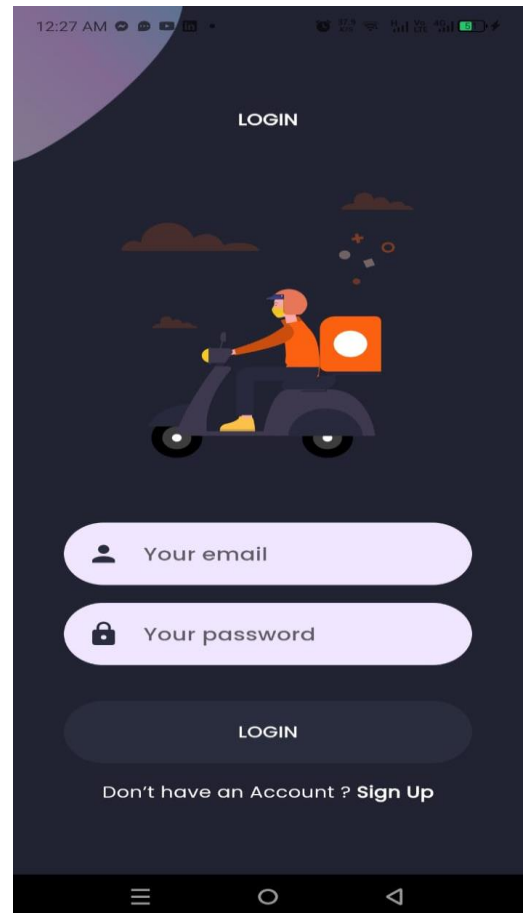
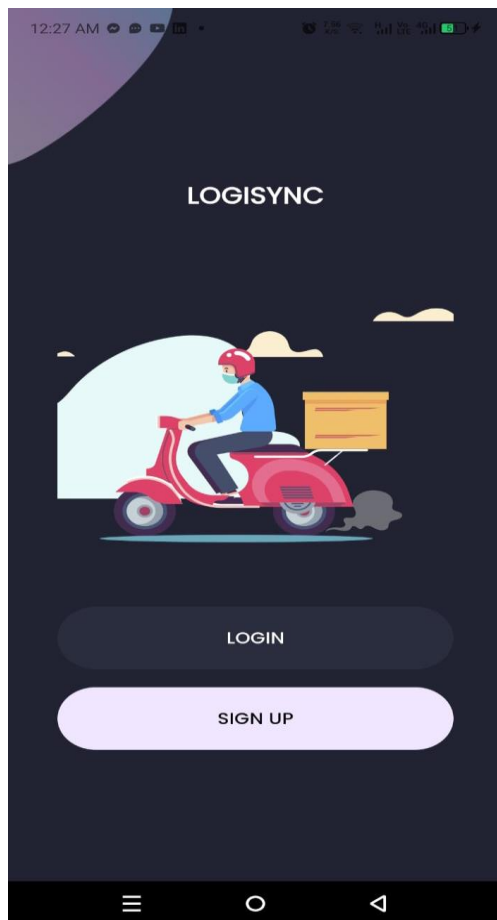
Figure 21 Dataflow diagram

4.6. UI Design:

The user interface design of LogiSync App and Admin portal focuses to provide a flawless and intrinsic experience for users. The Flutter app provides a simple and user-friendly interface for riders to start and stop journeys. The web admin portal offers a comprehensive dashboard for monitoring all rider's journeys, with features such as real-time updates of journeys, journey maps, and detailed reports.

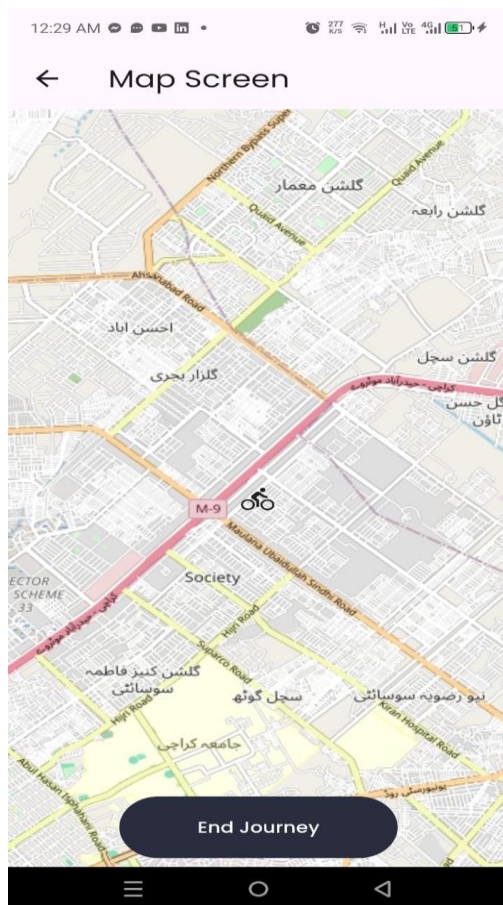
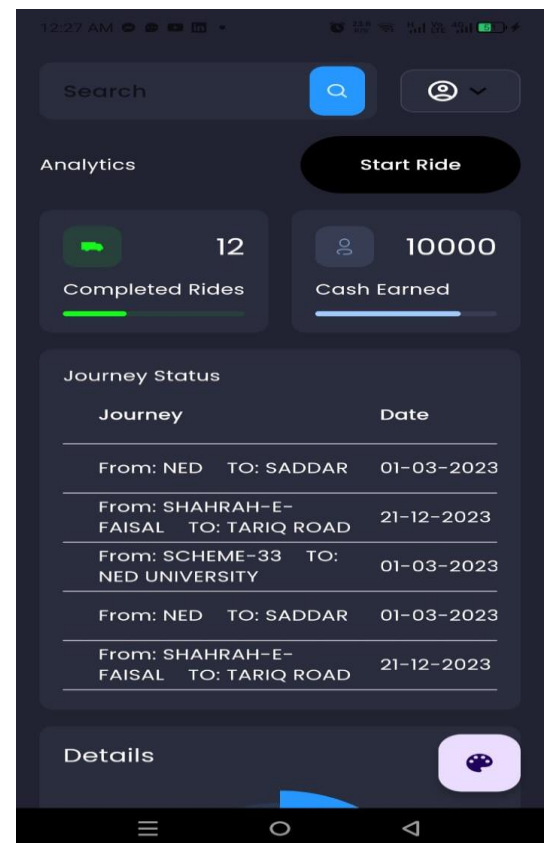
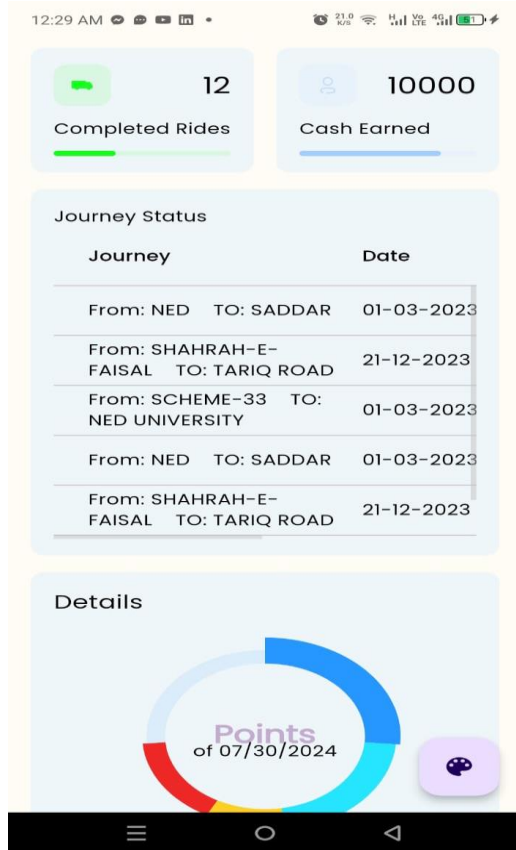
Below provided are screenshots and user interaction flows to demonstrate the design considerations and improvements made to improve user experience.

APP UI





F/SOP FYDP/09/00



WEB UI

Sign In

Enter your email and password to sign in!

Email*

admin@gmail.com

Password*

••••••••

👁

Sign In

Not registered yet? [Create an Account](#)

© 2024 Logisync. All Rights Reserved.




logisync

[Main Dashboard](#)

Pages / Default Brand Text

Default Brand Text




John Doe

Rider # 1

3 Journeys

See Details




Jane Smith

Rider # 2

3 Journeys

See Details



Adam Charles

Rider # 3

3 Journeys

See Details



F/SOP FYDP/09/00

logisync

Pages / Default Brand Text

Default Brand Text

🏠 Main Dashboard



Cost: 350 Rs

50 minutes

Distance: 12.5
Kilometers

See Route



Cost: 271 Rs

41 minutes

Distance: 9.7
Kilometers

See Route



Cost: 363 Rs

51 minutes

Distance: 13.0
Kilometers

See Route

logisync

Pages / Default Brand Text

Default Brand Text

🏠 Main Dashboard



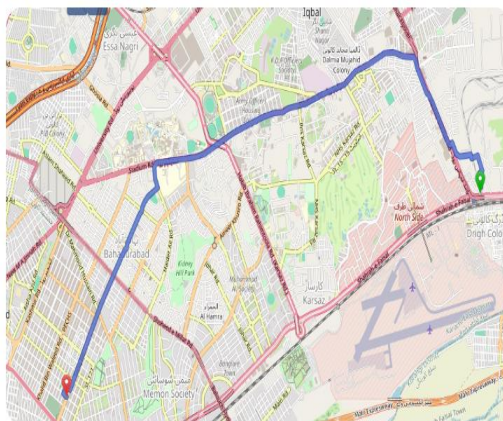
Cost: 350 Rs

50 minutes

Distance: 12.5
Kilometers

Traced Route

X



Close

4.7. Integration:

4.7.1. Process of Integrating Various Modules

- The integration process for LogiSync involved connecting multiple components to ensure seamless communication and data flow. Key modules included:
- Backend (Python, FastAPI, Node, MongoDB): FastAPI created APIs with Python for core logic and data processing. MongoDB stored GPS data and user details.
- Frontend (Flutter, Dart, React): Flutter and Dart developed the Rider Portal mobile app. React built the Company Portal for route tracking and analytics.
- Testing Tools (Postman, Swagger UI): Postman and Swagger UI tested and documented APIs for smooth integration.

4.7.2. Challenges Faced and Solutions

Challenges and solutions included:

- Data Consistency: Implemented data validation and synchronization to maintain consistency between the mobile app and backend.
- Real-Time Data Processing: Optimized backend services and used asynchronous processing to handle real-time GPS data efficiently.
- Compatibility Issues: Regular integration tests and CI/CD pipelines resolved compatibility issues between different technologies.
- Scalability: Designed scalable architecture with load balancers and distributed databases to handle increased user load.

4.7.3. Tools and Techniques Used for Integration

Tools and techniques used included:

- APIs and Web Services: FastAPI and Node created RESTful APIs for easy integration and scalability.

- Asynchronous Processing: Used Python's asyncio and Node.js for efficient real-time data handling.
- CI/CD: Employed GitHub Actions and Jenkins for automated testing and deployment.
- Testing and Documentation: Used Postman for API testing and Swagger UI for interactive API documentation.

These strategies ensured a cohesive and efficient system for optimizing last-mile logistics in LogiSync.

4.8. Testing:

4.8.1. Testing Methodologies

To ensure LogiSync's reliability and functionality, we employed several testing methodologies:

Unit Testing: Verified the correctness of individual modules using unit tests written in Python for backend services and Dart for the Flutter-based Rider Portal.

Integration Testing: Checked communication between components using Postman and Swagger UI to test APIs, ensuring seamless interaction between the frontend and backend.

System Testing: Validated the overall behavior of the application under real-world conditions, deploying the system on Vercel and PythonAnywhere for end-to-end testing.

4.8.2. Test Cases and Results

Test cases were documented to cover various scenarios, including:

- API Endpoints: Tested all API endpoints with Postman to ensure they responded correctly and handled edge cases.
- User Interfaces: Tested the Rider Portal and Company Portal interfaces for functionality and usability using manual and automated tests.
- Data Integrity: Ensured data consistency and integrity in MongoDB during operations.

Results were analyzed to identify issues, with test cases regularly updated to cover new features and changes.

4.8.3. Bug Fixing and Improvements

Continuous testing revealed bugs that were promptly addressed:

Bug Tracking: Used issue tracking tools to log and prioritize bugs.

- Fix Implementation: Developers fixed bugs based on priority, followed by re-testing to ensure issues were resolved.

- Improvements: Continuous feedback from testing led to improvements in performance, usability, and scalability.

We set up CI/CD pipelines with GitHub Actions and Jenkins to automate the testing process, ensuring consistent quality and faster deployment cycles.

4.9. Deployment:

4.9.1. Backend Deployment Our backend, developed using FastAPI, is deployed on PythonAnywhere, a reliable platform for hosting Python applications. PythonAnywhere provides a robust environment with excellent support for Python-based projects, ensuring our backend remains accessible and performant.

4.9.2. Web Portal Deployment The React-based web portal is deployed on Vercel, a platform optimized for frontend applications. Vercel offers seamless deployment, automatic scaling, and continuous integration, making it an ideal choice for our web portal. The deployment process on Vercel ensures that our web portal is always up-to-date and available for real-time monitoring of rider journeys.

4.9.3. Mobile Application Deployment The Flutter-based mobile application is built and installed on mobile devices directly. By generating a build of the app, we can distribute it easily for installation and use by riders. This approach ensures that riders have a reliable and user-friendly application for tracking their journeys.

Chapter 5

Conclusion

5.1. Summary:

The LogiSync project aimed to revolutionize last-mile logistics by providing a solution for delivery route optimization. This project successfully leveraged real-time GPS data, advanced map-matching algorithms, and a robust system architecture involving a Flutter app, FastAPI backend, and React web portal. Through meticulous implementation, testing, and deployment, LogiSync achieved its primary objectives of minimizing operational costs, enhancing route tracking accuracy, ensuring scalability, and providing data-driven insights.

5.2. Achievements:

LogiSync has made significant strides in optimizing last-mile logistics. The system effectively captures and processes GPS pings to generate accurate journey maps and calculate distances. The integration of the LCSS algorithm has improved route tracking precision, and the seamless interaction between the mobile app, backend server, and web portal has ensured a user-friendly experience. The web portal provides comprehensive monitoring and analytical tools, empowering companies to make informed decisions based on real-time data.

5.3. Challenges and Limitations:

Despite its successes, the project faced several challenges, including ensuring real-time data synchronization, handling large volumes of GPS data, and maintaining system performance under varying network conditions. Additionally, the system's reliance on continuous GPS signals can be a limitation in areas with poor signal coverage. Future enhancements could address these challenges by incorporating offline data storage and synchronization mechanisms.

5.4. Future Work:

To build on the current achievements, future work could focus on improving the scalability and robustness of the system. Enhancements might include implementing machine learning algorithms for predictive analytics, integrating with additional mapping and navigation services, and expanding the system to support multi-modal transportation. Further, enhancing security measures and exploring blockchain technology for data integrity and transparency could add value to the system.

References

1. Li, X., Zhang, L., & Yang, Y. (2018). Trajectory segmentation map-matching algorithm for large-scale, high-resolution GPS trajectory data. *IEEE Transactions on Intelligent Transportation Systems*, 19(10), 3246-3259.
2. Kuhn, W. (2020). Snap To Nearest Map Matching Algorithm. *Journal of Spatial Information Science*, 20, 77-94.
3. OSRM Routing Machine. (n.d.). Retrieved from <https://github.com/Project-OSRM/osrm-backend>
4. Mapbox. (2021). Valhalla: A high-performance routing engine for real-time traffic. *Journal of Open Source Software*, 6(58), 2861.
5. Newson, P., & Krumm, J. (2009). Hidden Markov map matching through noise and sparseness. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 336-343.
6. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269-271.
7. J. S. Greenfeld, G.P.S. Matching, Observations to Locations on A Digital Map. In:Proc of the 81th Annual Meeting of the Transportation Research Board, Washington D C ,2002.
8. Y.L. Zhao,Vehicle Location and Navigation Systems.Published by Artech House,pp.83- 103,1997.
9. White C E,Bernstein D,Kornhauser A L,Some Map Matching Algorithms for Personal Navigation Assistants.Transportation Research Part C,8:91-108,2000.
10. Q.S. Zhang, J.P. Wu and D.K. Yang , Intelligent vehicle positioning navigation system and its application, Scientific Publishing House,2002-04.
11. J. Su, D.F. Zhou and C.S. Yue,The real-time map-matching algorithm in GPS vehicle navigation. Mapping Journal, 30 (3) : 252- 256,2001.
12. B.L. Meng and B. Gao,Several key issues research of GIS in VNS. Information Engineering University Journal, 2 (4) : 58- 62,2001.
13. J. Li and M.Y. Fu,Research on Route Planning and Map Matching in Vehicle GPS/DeadReckoning/Electronic Map Integrated Navigation System.In:Proc IEEE Intelligent Transportation Systems,(2):1639-1643,2003.
14. Z.W. Chen, Y.R. Sun and X. Yuan,Development of An Algorithm for Car Navigation System Based on Dempster-Shafer Evidence Reasoning.In:Proc IEEE Intelligent Transportation

Systems, pp.534-537, 2002.

15. M E EL Major and Ph Bonnifait, A Roadmap Matching Method for Precise Vehicle Localization Using Belief Theory and Kalman Filtering. In: Proc International Conference on Advance Robotics, pp.1677-1682, 2003.
16. D.K. Yang, B.G. Cai and Y.F. Yuan, An Improved Map-Matching Algorithm Used in Vehicle Navigation System. In: Proc IEEE Intelligent Transportation Systems, 2:1246- 1250, 2003.
17. R. R Joshi, Novel Metrics for Map-Matching in In-Vehicle Navigation System. In: IEEE Intelligent Vehicle Symposium, 1:36-43, 2002.
18. R. R Joshi, A New Approach to Map Matching for In-Vehicle Navigation Systems: the Rotational Variation Metrics. In: Proc IEEE Intelligent Transportation Systems, 33-38, 2001.
19. K. Zhao, Y.H. Yang and B.Z. Qu, GPS / DR group and the navigation system map matching algorithm based on position points matching. Guidance and fuses, 24 (3) : 22-27, 2003.
20. N. Wang, Y.F. Wang and J.R. Liu, A mapmatching algorithm based on the location of points matching. Journal of Northeastern University, 20 (4) : 344-347, 1999.
21. J.H. Huang, The real-time study of map matching algorithm. Northwest Industrial University master degree thesis, 2001.
22. R.Z. Guo, Spatial Analysis. Wuhan Technical University of Surveying and Mapping Press, 1998.
23. Z.H. Zhang, T.J. Cui and H.M. Yao, The new map matching algorithm in vehicle navigation system. Ocean mapping, 26 (2) : 55-58, 2006.
24. H. Gao and Q. Chang, The vehicle tracking map matching research based on current statistical model. China Road and journals, 19 (2) : 95- 100, 2006.
25. H. Wu, Z.J. Sheng and Q. Yu, The mapmatching algorithm in GIS / GPS urban traffic flow monitoring system. computer engineering, 32 (7) : 237-239, 2006.
26. F. Peng, Z.K. Liu and Q.S. Zhang, The integrated navigation system map matching algorithm based on the cost function. Beijing aerospace university Journal, 28 (3) : 261-264, 2002.