THE MITSUBISHI-A TEAM

Final Report

Authors:
JEREMY J. LIN[1]
TOMORO MOCHIDA[2]
RILEY C. W. O'NEILL[3]
ATSURO YOSHIDA[4]

Mentors:
DR. SHUNSUKE KANO[+]
DR. MASASHI YAMAZAKI[*]
MR. AKINOBU SASADA[*]

Institute
[1] University of California Irvine
[2] Tohoku University
[3] University of Minnesota, Twin Cities
[4] Nagoya University
[+] Academic Mentor, Tohoku University, MathCCS
[*] Industrial Mentor, Mitsubishi Electric

August 9, 2023

Contents

1   Introduction

Map matching is the process of determining the correct traveling route taken by a person or vehicle using a road network (map) and trajectory data obtained from GPS or other positioning sensors. While it may be easy for humans to estimate, it is difficult to implement as an algorithm. Handling GPS errors and route selection at junctions and parallel roads well is the key to accurate map matching. Various map matching algorithms have been provided so far and they are typically classified into four types depending on the method used: geometric, topological, probabilistic, and advanced map matching. Each of these methods has its advantages and disadvantages in terms of accuracy, implementation, generalizability, assumptions, etc. In last year's g-RIPS Sendai 2022 Mitsubishi-A project [Aka+22], three methods were proposed for registration: Wasserstein method, electrical force method, and harmonic oscillator method.

Our aim of this project is to develop new map matching algorithms to improve upon previous work. We propose four key approaches: signature curve geometric registration, deep learning based registration, AHP map matching, and reinforcement learning-based Fuzzy logic. We implemented and tested these four approaches (as well as combinations of them) and compared their performance with existing algorithms.

2   Notation, Problem Summary, and Statement

2.1   Notation

We formulate a graph-theoretic formulation of roads as follows:

Definition 2.1. Graph - a set of points $V = (v_1, v_2, \ldots, v_n) \subseteq \mathbb{R}^d$ equipped with a list of edges $E = \{(e_{i,1}, e_{i,2})\}_{i=1}^m \subseteq \mathbb{N}_n \times \mathbb{N}_n$ ($\mathbb{N}_n = (1, 2, \ldots n)$) which denotes connected pairs of points, i.e. $v_{e_{i,1}}$ connects to $v_{e_{i,2}}$ for all $1 \leq i \leq m$. We assume $e_{i,1} \neq e_{i,2} \ \forall \ i$ to avoid storing needless information.

Definition 2.2. Vertex / Node - a point in a graph. We use these terms interchangeably.

Definition 2.3. Road Network - a directed graph G representing roads under the geometric realization. We leave this deliberately broad, possibly including both junction points and intermediary points:

Definition 2.4. Junction point - a vertex p_i in a road network with a degree of connectivity d_i that satisfies $d_i = 1$ or $d_i \geq 3 \ \forall \ i = 1, N$. Intuitively, this is a point where one's trajectory can or must change (without taking an illegal u-turn); while the degree 1 points are not "junctions," these are still important points that must be considered when examining the connectivity of the map graph.

Definition 2.5. Intermediary point - a vertex in a graphical representation of a road with a degree connectivity 2. Intuitively, this is a point where one's trajectory cannot change (without taking an illegal u-turn or exiting the road network.

Define the road segment and map adjacency graph as follows:

Definition 2.6. Road segment - a directed subgraph of the road network where the two end nodes n_1 and n_N are junction points (i.e. a degree of connectivity that satisfies $d_i = 1$ or $d_i \geq 3 \ \forall \ i = 1, N$), and all intermediary nodes between them are strictly degree 2 ($d_i = 2 \ \forall \ 1 < i < N$).

This is simply to say that if one is on a road segment, one cannot alter one's trajectory unless one travels to an end of the segment (unless taking a U-turn). We leave U-turns to another matter of pre/postprocessing.

We also define

Definition 2.7. Map adjacency graph - an abstraction of the road network that encodes the connectivity of road segments - i.e., what path decisions one has when driving or walking. Each edge is a road segment.

2.2   Problem Summary and Statement

Let us fix $d \geq 2$ (but almost everywhere we consider the case $d \in \{2, 3\}$).

Definition 2.8 (Trajectory). A **trajectory** T is a timeseries of points $P = (p_1, p_2, \ldots, p_n) \subseteq \mathbb{R}^d$ equipped with several features $F = (f_1, f_2, \ldots, f_n) \subseteq \mathbb{R}^k$, where $k \geq 1$. We mandate that F at least includes timestamps, but could possibly include additional attributes, as explained thus:

1. Timestamp - a sequence $T_t = (t_1, t_2, \ldots, t_n) \subseteq \mathbb{R}_+ = [0, \infty)$ that is strictly increasing ($t_1 < t_2 < \cdots < t_n$).

2. Speed (optional) - a sequence $T_s = (v_1, v_2, \ldots, v_n) \subseteq \mathbb{R}_+$.

3. Direction of travel/head (optional) - a sequence of unit vectors $T_u = (u_1, \ldots, u_n) \subseteq \mathbb{S}^d$ which are the unit velocity vectors ("direction") of each point in the trajectory.

4. Acceleration (optional) - a sequence $T_a = (a_1, a_2, \ldots, a_n) \subseteq \mathbb{R}$.

5. Gyroscopic measurements (optional) - a sequence $T_g = (g_1, g_2, \ldots, g_n) \subseteq \mathbb{R}^c$.

6. Elevation data (optional) - $T_z = (z_1, z_2, \ldots, z_n) \subseteq \mathbb{R}$.

7. other features (optional) - $T_o = (o_1, o_2, \ldots o_n) \subseteq \mathbb{R}^q$. These may include signature curves (in $\mathbb{R}^2$), other geometric features, and features learned from machine learning, all of which shall be discussed later.

Remark 2.9. It is worth noting that the speed, acceleration, and gyroscopic sensors may not sample at the same rate as the GPS sensor - usually at a higher sampling frequency. However, g-RIPS Sendai 2022 [Aka+22] implemented an interpolation algorithm to align/fuse these data to the GPS points. As a consequence, we do not need to worry about misaligned sampling frequencies, and we are immeasurably grateful for this foundation to work off of.

We now define the map-matching problem on which we shall work for the rest of the summer:

Problem 2.10. Map-Matching Problem. Given a road network (V, E) and a GPS trajectory T with features F and coordinates P, match T to the ground truth (or closest) route taken in (V, E). There are two main subsets of map matching:

1. Online map matching, or "live" matching - this is intended to happen in real-time as an entity traverses a route. This has many applications for autonomous navigation systems.

2. Offline map matching - this happens after an entity's entire course of movement has been completed, used to register the complete trajectory from point A to point B. This has many applications to map synchronization and map fusion problems.

The main goal of this project is to develop innovative new methodologies that can be leveraged in online and/or offline registration of GPS trajectories using several different approaches, ideally with greater generalizability, reduced error, and applicability to modern industrial applications at scale. In doing so, we hope to achieve a new paradigm for unsupervised GPS feature extraction, geometric/topological/learned feature registration, and fuzzy registration vis-a-vis reinforcement learning and other state of the art technologies.

3　Background

In this section, we briefly review several existing map matching techniques: Topological Map Matching Algorithms, Fuzzy Logic Map Matching Algorithms, Dijkstra's Algorithm for Offline Matching, and Deep learning for Trajectory Registration.

3.1　Topological Map Matching Algorithm

The classical geometric map matching algorithm, such as point-to-point, point-to-curve, and curve-to-curve map matching algorithms, uses only geometric information, i.e., coordinates of trajectory points and distance. It focuses only on the arc shape and does not reflect other information. Therefore, although it has the advantage of being simple and fast, it lacks accuracy. In order to improve the geometric map matching algorithm, the topological map matching algorithm uses other factors such as connectivity, proximity, and contiguity of edges in addition to the geometric information. Here, we present Greenfeld's algorithm [Gre02].

His algorithm consists of two methods, which he called InitialMapping() and Map(). The first algorithm InitialMapping() finds an initial match of a point to a vertex. It is applied in the following situations:

1. When the first GPS point p_0 is received.

2. When the distance between the new GPS point p_t and the point p_{t-1} exceeds a pre-selected distance tolerance.

3. When the main map matching algorithm is unable to successfully map a particular GPS point.

When these situations are met, InitialMapping() algorithm is executed according to the following steps:

1. Find the closest vertex v_0 to a point p_0. This is essentially a geometric only matching process.

2. Determine all edges in the road network that are connected to node v_0.

3. Map the next point p_1 onto one of the selected edge e_1.

The second algorithm Map() is applied only after an initial match found by InitalMapping(). It is the main algorithm of his approach that uses topological reasoning and a weighting scheme to match a point p_t with an edge e_t. It takes the following steps:

1. Obtain the next GPS point p_t.

2. Form a GPS line segment between points p_{t-1} and p_t.

3. Evaluate the proximity (distance) and orientation (direction or azimuth) of the GPS line to the currently matched edge e_{t-1} and determine whether p_t maps onto the edge e_{t-1}.

4. If p_t maps onto e_{t-1}, set $e_t := e_{t-1}$. Otherwise, find another edge, e_t, which is either connected to e_{t-1} or is nearby downstream from e_{t-1}. The edge e_t is also selected based on the same proximity and orientation evaluation scheme.

5. Go back to Step 1.

As a method of the evaluation in Step 3, he assigned a weight to the edge e_{t-1} according to the direction of the GPS line segment $p_{t-1}p_t$, the distance between p_t and e_{t-1}, and the intersection of $p_{t-1}p_t$ and e_{t-1}. This is the topological part of his algorithm.

Remark 3.1. It is worth noting that he was focusing on online map matching, so when considering a point at a certain time his algorithm (basically) used no more information about future points than that point.

His program is simple and straightforward but has some problems including:

1. InitialMapping() is less accurate, since it simply takes the closest vertex.

2. He considered that proximity is the most important factor to determine weights, but this is not always true. For example, Quddus [Qud+03] reported the following case (see Figure 1): His algorithm can detect that both points p_1 and p_2 match the edge e_1. However, for point p_3, his algorithm gives the mismatching to the edge e_2 instead of e_4.

3 Since his algorithm only uses information available from coordinates, it is heavily influenced by outliers.

In Subsection 6.1 we introduce a revised topological map matching algorithm that incorporates the analytic hierarchy process and uses direction and speed data of trajectory points in addition to distance data.

3.2   Fuzzy Inference System

Fuzzy logic is an approach to computing based on degree of truth, usually represented as probability ranges from 0 to 1 rather than the usual boolean logic. Fuzzy logic aims at modeling the imprecise modes of reasoning that play an essential role of human rational decision making and can be viewed as the extension of multi valued logic ([Zad88]). A Fuzzy Inference System is the framework for applying fuzzy logic to obtain a numerical output from a given input.
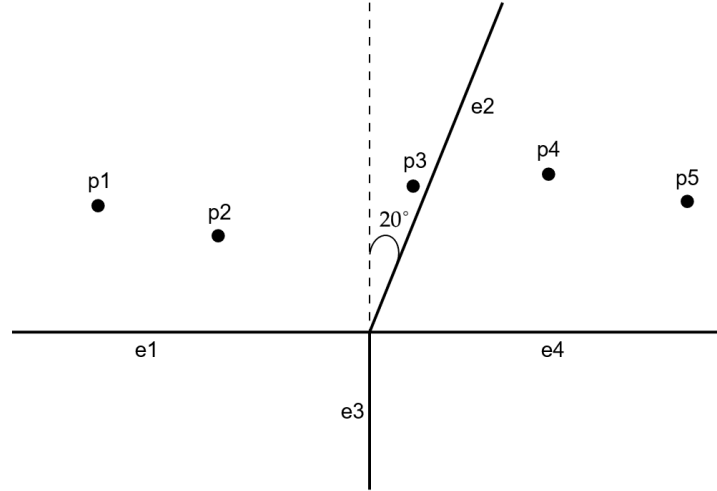
Figure 1: Mismatching in Greenfeld's algorithm

Definition 3.2 (Fuzzy Inference System)**.** Fuzzy Inference System (illustrated in figure 2) is a process of formulating mapping from an input to an output using some if-then rules. The FIS rule base is made of N rules of the following form :

$$R_i : \textbf{If } S_1 \text{ is } L_1^i \textbf{ and } \cdots S_p \text{ is } L_p^i$$
$$\textbf{Then } Y_i$$

where :

- **Rule** $R = (R_1, R_2, \ldots, R_N)$, R_i is the ith rule base.

- **Crisp input** is a vector $S = (S_1, S_2, \ldots, S_p) \in \mathbb{R}^p$

- **Fuzzy input** is a vector $L = (L_1, L_2, \ldots, L_N), L_j = (L_j^1, L_j^2, \ldots, L_j^p) \in [0,1]^p$ such that $L_j^i \in [0,1]$, in where decision making process occurs. where L_j^i represent the transformed value of input S_j in rule R_i.

- **Membership function** $\mu_{L_j^i} : \mathbb{R} \to [0,1]$ (resp. $\mu_{O_j^i}$) is a function that transformed input (S_j) (resp. fuzzy output Y_j) into fuzzy input L_j^i. (resp. output O_j^i).

- **Fuzzy output** $Y = (Y_1, Y_2, \cdots, Y_q)$ such that Y_i is the ith solution calculated from fuzzy input.

- **Crisp Output** or **conclusion** $O = (O_1^1, O_1^2, \cdots, O_q^N)$ is the linguistic output variable Y_j in rule R_i

Fuzzy inference system algorithm consists of three main steps illustrated by figure 2 :

1. Fuzzification: fuzzifying input values (S) with membership functions$(\mu_{L_j^i})$ to a fuzzy input(L).

2. Fuzzy inference step: operating all applicable rules from fuzzy input (L) to fuzzy output (Y).

3. Defuzzication: De-fuzzifying fuzzy output(Y) set back to a crisp output value (O).

In the following example we will discuss a simplified version Fuzzy Inference System given by Quddus [QNO06].
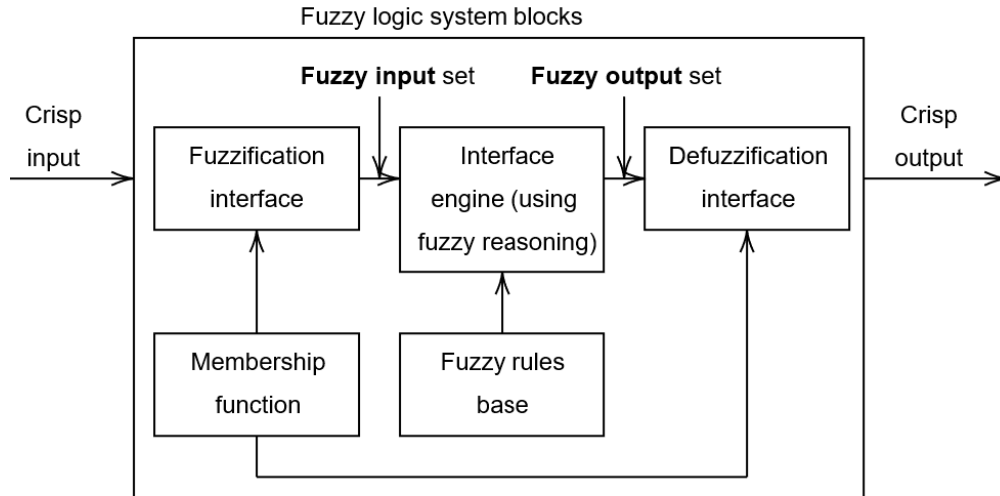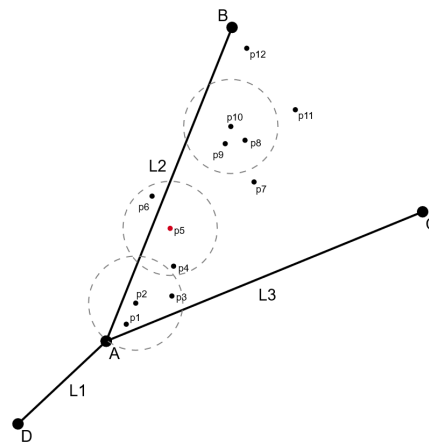
Figure 2: Fuzzification-Defuzzification pipeline.

Figure 3: three legged junction example

Example 3.3. Suppose we would like to decide which link (DA, AB, or AC) does vehicle at position p2 travels in a three legged junction illustrated in figure 3. Denote PQ as the perpendicular distance from P to link AB, θ as the direction of the vehicle at P obtained from the navigation sensor, and $\Delta\varphi = |90° - \theta|$ as the angular difference of the vehicle. Suppose PQ and $\Delta\varphi$ are the two primary determining factors, also referred to as crisp input, as to whether P is matched to AB. A knowledge based rules can be formed based on these two inputs, for example :

1. R_1 : If PQ is short and $\Delta\varphi$ is small then the possibility of matching P on link AB is high.

2. R_2 : If PQ is long and $\Delta\varphi$ is large then the possibility of matching P on link AB is low.

Where terms such as short (or long) and small (or large) are linguistic values that can be defined on the range of PQ and $\Delta\varphi$. We can start step 1 by transforming our crisp input PQ (S_1) and $\Delta\varphi(S_2)$ into a fuzzy input denoted by $L_1 = (L_1^1, L_1^2)$ and $L_2 = (L_2^1, L_2^2)$ using some membership function $(\mu_{L_j^i})$ where $i, j \in 1, 2$ that is known from domain knowledge.

Once the input is fuzzified, we can proceed to the next step by evaluating a fuzzy output using some fuzzy operator. The "if" part of the rule (e.g. PQ is short) is called an antecedent or premises. When the

rule consists of more than one antecedent we can use fuzzy operators to evaluate the results of the rule, also commonly called rule strength. Two fuzzy operators that are normally used are the AND and OR operator. Min (minimum) and prod (product) are popular functions for AND operator, while max (maximum) and probabilistic OR are widely used for OR operator [QON07].

In this example, suppose PQ is equal to 15m and $\varphi = 15°$. Figure 7 and figure 11 illustrate how we can calculate the fuzzy inputs (L_1, L_2) and strength of each rules. The strength of each rules are calculated by applying the min function for the **AND** operator. Since the fuzzy input from the first rule are $L_1^1 = 0.5$ and $L_2^1 = 0.8$, the strength of the first rule is equal to $min(L_1^1, L_2^1) = 0.5$. Similarly, since the fuzzy input for rule two is equal to $L_1^2 = 0.4$ and $L_2^2 = 0$) so the strength of the first rule is equal to $min(L_1^2, L_2^2) = 0$.

Figure 4   Figure 5   Figure 6

Figure 7: Rule 1

Figure 8   Figure 9   Figure 10
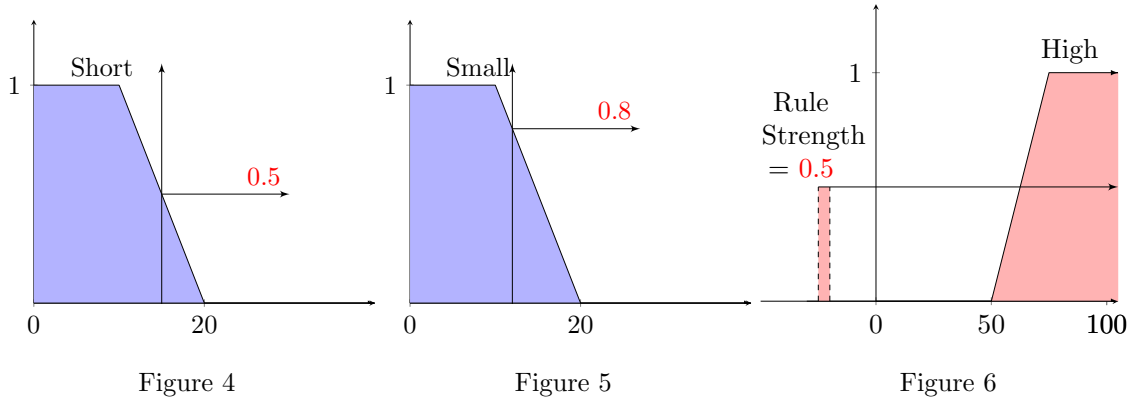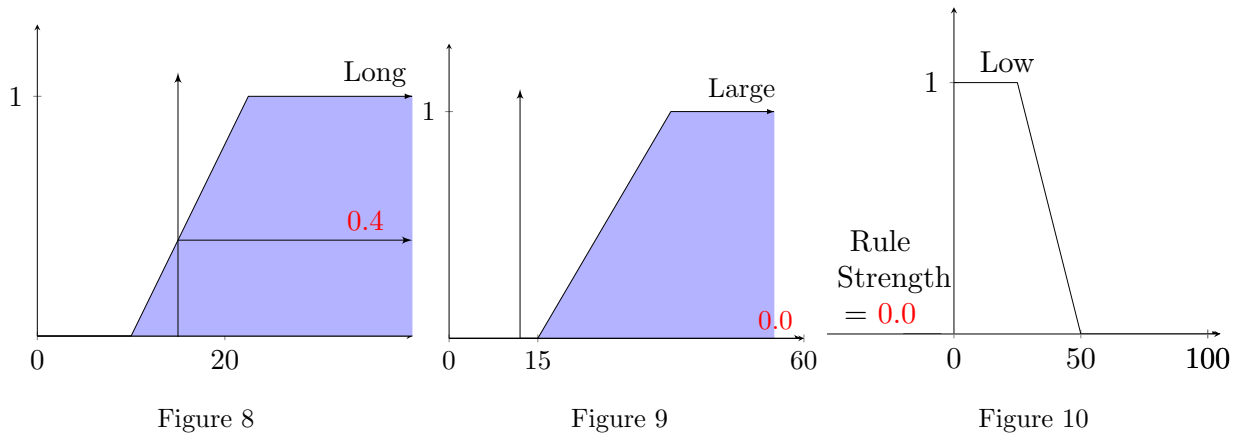
Figure 11: Rule 2

We can then convert this rule strength into a fuzzy output($Y = Y_1, Y_2$) using two membership function shown in the figure 6 and 10. Lastly we can proceed to defuzzifying step where we aggregate these fuzzy output into a crisp output(O_1). Area of the centroid (Mandani FIS method) or linear combination of each output(Sugeno FIS method) are one of the few common de-fuzzifying methods that can be used for our final decision making process.

3.3   Dijkstra's Algorithm for Offline Matching

Rather than do an exhaustive brute-force map matching algorithm, which requires $\mathcal{O}(n^2)$ computations, many state of the art algorithms (e.g. Valhalla [SH22]) employ Dijkstra's algorithm [Dij59], and [Aka+22] utilized Dijkstra's [Dij59] as well. For a given graph, this employs a greedy registration framework with edge weights to identify the path from a starting point to end point that best minimizes the total edge weights. Interestingly, Dijkstra formulated the algorithm in only twenty minutes and published it as a 3 page "note on two problems in connexion with graphs", and it remains one of the most infleuntial algorithms in graph theory and shortest path methods.

The algorithm is shown in Algorithm 1. Select p_0 as the starting node and, let d_i denote the "distance" (i.e. lowest sum of weights on connecting edges) from p_0 to node p_i.

Algorithm 1: Dijkstra's Algorithm

Input: Graph G, source node p_0
Output: Shortest distances from p_0 to all other nodes
1 Initialize distance to p_0 as 0 and all other nodes to ∞ Create a priority queue Q; Insert p_0 into Q.
2 while Q *is not empty* **do**
3   Extract node u with minimum distance from Q; **foreach** *neighbor v of u* **do**
4   **if** *distance to v through u is shorter than current distance* **then**
5   Update distance of v to the new shorter distance;
6   **end if**
7   **if** *v is not visited* **then**
8   Add v to Q;
9   **end if**
10   **end foreach**
11 end while

Note: this algorithm was typeset by ChatGPT [Ope23a], with minor modifications, corrections, and changes of notation. We have verified with [Dij59] that it is indeed correct. This is the only portion of the report written or typeset using ChatGPT.

3.4   Deep learning for Trajectory Registration

Relatively little has been done (publicly) in the way of deep learning for GPS trajectory registration due to the lack of large, publicly available labelled datasets with many features. [Liu+20] utilized a rather exhaustive geometric, topological, and speed matching (i.e. max feasible speed of the roadway) analysis to propose candidate routes. Finding the rigid speed matching unideal, they deploy machine learning to predict each road's speed using a bidirectional Conv-LSTM RNN. The training of this Conv-LSTM requires road-wise traffic usage data, which we do not have access to. Other approaches such as [WT16] also employ historical data to train a Hidden Markov Model (HMM), but their data is not public either. [HK16] employed a simple 2 layer neural network to reposition the GPS points slightly prior to inputting them into the registration algorithm - interestingly, they implement it for real time applications, which were trained using the horizontal displacement between the GPS point and the ground truth trajectory for selections from the OpenStreetMaps [Ope23b] traces data.

4   Our Approach

4.1   Stay Point Mitigation and Outlier Detection by DBSCAN

It is known that stay points and outliers which may be caused by GPS errors may prevent map matching algorithms from finding the correct route. It is researched in [Jaf22] that Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [Est+96] may be utilized to detect and mitigate stay points in a GPS trajectory. In this research project, we developed a method to utilize DBSCAN to detect outliers. Furthermore, in addition to these applications, we propose a method to automatically determine a parameter

passed into the algorithm according to each input of points. Although we could not actually integrate the methods that we developed into map matching algorithms due to the time constraints of this research project, we managed to implement the methods in Python and expect that it will improve the performances of our algorithms. The details of these methods are talked about in Subsection 9.1.

4.2 Datasets

The GPS data that is publicly available is severely lacking, especially for the purposes of deep learning. State of the art open-source frameworks like Valhalla [SH22] utilize over 18 million verified trajectories to train their registration framework (a Hidden Markov Model), but this trajectory data is not open-source. Companies like Google and Apple, the creators of two of the most popular navigational smartphone applications, certainly have access to far more trajectory data than that to train their models.

If we are to come anywhere close to the success of existing industrial models with a data-driven approach, we are going to need an extensive amount of data. The largest datasets we can access (or plan to assemble) are thus:

1. KCMMN - "Dataset for testing and training of map-matching algorithms" [Kub+15a] - 100 different trajectories spanning over 5,000 km of roads across the globe. Features are restricted to GPS coordinates and timestamps only; has verified ground truth trajectories.

2. BDD100K [Yu+20] - 100,000 trajectories of 40 second travel sequences. Contains GPS coordinates, timestamps, IMU data (highly accurate speed, direction, and gyroscopic data), and videos; has no ground truths.

3. OpenStreetMaps Traces [Ope23b] - very large, publically available dataset consisting of GPS coordinates and elevation data for many trajectories collected around the globe. The dataset is constantly growing. Notably there is no IMU data or ground truth.

4. 2022 G-RIPS Mitsubishi-A data - collected from former participants walking around Sendai. Unclear if it is just GPS coordinates and timestamps or if it contains IMU data as well. Should have ground truths, but we have not yet seen the data at this time.

We can summarize the publicly available data as follows and see the clear gaps:

Dataset	Raw GPS Timeseries	IMU data	Velocity	Elevation	Ground Truth
KCMMN [Kub+15a]	V	X	X	X	V
BDD100K [Yu+20]	V	V	V	X	X
OpenStreetMaps [Ope23b]	V	X	V	V	X
EnviroCar [Env23]	V	X	V	X	X

This incentivizes some means to infer IMU/Velocity/Elevation data for the KCMMN dataset, the only one with ground truth: i.e. *data fusion*.

4.3 Data Fusion: Estimating IMU Data for KCMMN

In the absence of a large trajectory dataset equipped with both IMU data and ground truths, we propose to first leverage the BDD100K data (with IMU data) to estimate IMU data for the KCMMN trajectories. This can be readily achieved using a physics-informed neural network with filtering, or a relatively lightweight GNN, RNN, or transformer. Such data may also be estimated for the OpenStreetMaps trajectory data, provided it works well for KCMNN.

Speed and direction of travel are of greater intuitive interest than the gyroscopic measurements, but we should try to estimate all possibly useful values that we can. These quantities may also be approximated directly from the GPS trajectories and timestamps with some form of local filtering and/or regression.

4.3.1　Preliminary Results and Changes to Approach

We first implemented a 3-layer NNConv GNN using PyTorch Geometric (PyG) to model the IMU data using graphs built off the GPS trajectories using the BDD100K dataset; each layer of which used a 3-layer network to weight the features based on the differences in the input trajectories. Overall, this failed, despite modifications to the loss to use percent error instead of absolute error. Despite PyG having a DataParallel module (notably one with exceptionally poor documentation and no examples), training with multiple GPU's was exceptionally slow and tedious (it was only realized later that CSV's read in 16x slower than NPZ files, but even after converting the data, it was still very slow). It remains a very curious matter. Finally, after PyG's distributed processes crashed two A-100 GPU's at the Minnesota Super-computing Institute, we resorted to abandon PyG and try a 1-dimensional CNN instead.

It appears that the 1D CNN experienced vanishing gradients, ultimately returning all 0's. It was then thought that this was due to the padding (most trajectories in the BDD100K had about 40 points, but a few had 80), but after implementing a masking framework for the loss, this still converged to 0's. After various attempts to remedy this, we ultimately decided to focus on training the edge affinity function instead.

Given the difficulties in training an accurate model with the BDD100K dataset [Yu+20] (and paired with unanswered requests for clarification on the units on the BDD100K data), we elected to approximate the velocities and directions of travel for the KCMMN dataset [Kub+15a] using simple 1D differentiation filters. While most of the data follows regular sampling, a few points have irregular timesteps, which makes higher order differentiation filters a little more challenging to implement. Ultimately we used the neighboring two point approximation to approximate the velocity:

$$v(x[i]) \approx \frac{\vec{p}[i+1] - \vec{p}[i-1]}{t[i+1] - t[i-1]}$$

From this, we approximate the speed $\|v(\vec{p}[i])\|$ and direction of travel as the angle from the x-axis. This allowed our Fuzzy-Inference and AHP based methods to run on KCMMN. In the future, far more robust approximations should be used, i.e. local parametric curve fitting from the timestamps, but time did not allow this to be implemented.

5　Dijkstra's Algorithm with Learnable Edge Affinity Function (DA-LEAF)

We propose here a novel framework for offline matching using machine learning.

5.1　Motivation

In the ideal setting, we would have many features for the raw GPS trajectories (x-y coordinate, timestamp, speed, direction, gyroscope readings, learned features from contrastive learning, curvatures, etc). However, the ground truth routes in the KCMMN dataset [Kub+15a] only have an ordered set of map edges with no timestamp - i.e. we only have x-y coordinates, and possibly approximations for curvatures after curve fitting. While registration is conceivable with these quantities alone, it is conceivably adventitious to utilize more information. To this end, we propose a learnable K-Nearest Neighbor edge weighting scheme for Dijkstra's algorithm: DA-LEAF, or Dijkstra's Algorithm with Learnable Edge Affinity Function. The regime is illustated in Figure 12 and described momentarily.
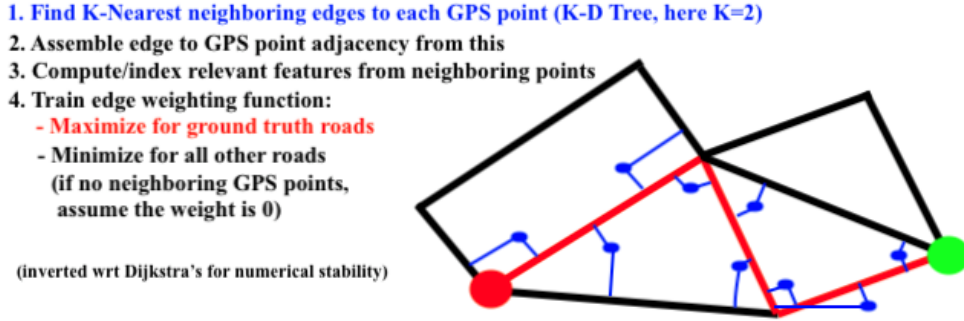
Figure 12: Learnable weights for Dijkstra's algorithm.

Recall the definitions of road segment and map adjacency graph:

Definition 5.1. Road segment - a directed subgraph of the road network where the two end nodes n_1 and n_N are junction points (i.e. a degree of connectivity that satisfies $d_i = 1$ or $d_i \geq 3 \ \forall \ i = 1, N$), and all intermediary nodes between them are strictly degree 2 ($d_i = 2 \ \forall \ 1 < i < N$).

This is simply to say that if one is on a road segment, one cannot alter one's trajectory unless one travels to the end of the segment (unless taking a U-turn). We leave U-turns to another matter of pre/postprocessing.
We also define

Definition 5.2. Map adjacency graph - an abstraction of the road network that encodes the connectivity of road segments - i.e., what path decisions one has when driving or walking. Each edge is a road segment.

Given the trajectory graph $\text{Tr} = (V, E, F)$, where $V = \{p_1, p_2, ...p_K\}$, and the features F contain (optional) auxiliary information about each vertex (speed, direction of travel, gyroscopic readings, elevation, signature curvature, learned features from contrastive learning, etc.), we formulate a learnable Dijkstra's weighting as follows:

DA-LEAF Reverse K-NN Search and Edge Affinity Training Algorithm

1. For each point in the GPS trajectory, retrieve the K-Nearest Neighboring road segments.

2. Compute the displacement vector to the road segment from each trajectory point, i.e. $D(p, e) := p - \underset{q \in e}{\text{Argmin}} \, |p - q|$. Denote $|D|(p, e) := \min_{q \in e} |p - q|$.

3. For each road segment (edge) e in the sub-map adjacency graph (edges within buffer of the GPS trajectory), extract all GPS point indices that bear the edge as a K-nearest neighbor (i.e. $S(e) := \{i : p_i \in V, e \in KNN(p_i)\}$). Note: these indices can be precomputed for training, but strong augmentations on the map and point data (i.e. adding GPS coordinate noise that could affect KNN, noise to map coordinates) would require recomputing KNN. This could be done in a parallel fashion for training.

4. Train an edge weight function $f(e) = f(V[S(e)], F[S(e)], D(V[S(e)], e), |D|(V[S(e)], e))$ as a neural network. The function should be high when e is in the ground truth trajectory, and low when e is not - this is how we formulate the loss.

5. Use this edge weight function with Dijkstra's algorithm.

5.1.1   Implementation: Data Processing & Model Input

Given unforeseen difficulties in accurately modeling IMU/acceleration/speed data from the GPS trajectories, we resolved to implement the edge affinity model using just the GPS trajectory points in the reverse K-NN edge search with $K = 5$ for each trajectory point. Assuming that the edge affinity for an edge is 0 if no trajectory points map to it, this helps mitigate computational overhead and ease of training.

We considered implementing graph neural networks for this task, but the graph construction requires some non-trivial consideration as the adjacent GPS points can be very sparse and vary greatly in the temporal

domain. This, paired with our previous difficulties with GNN's and Pytorch Geometric, lead us to implement a point-cloud based network instead using 3 inputs: signed distance to the edge, projected signed distance to the first node, and projected signed distance to the second node. Figure 13 shows these 3 quantities. We also considered using the GPS timestamps as well, but empirical results have not shown a strong motivation for doing so.

The K-NN point to edge search was done via exhaustive distance computations over the entire map subgraph, which is computationally exhaustive. Certainly there are ways of expediating this computation by means of say, a K-D tree, depth-first search, or spatial buffer query, but these methods were not explored for the purposes of time.
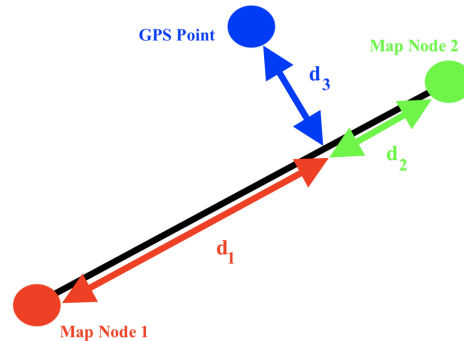

Figure 13

5.2   Model Architecture & New Pooling Operator

The network structure consists of 3 key components:

1. **Point-wise feature encoder**: 3 layer neural net.

2. **Feature Aggregation (pooling)**: to be discussed.

3. **Decoder (classifier)**: 2 layer neural net.

As the point-sets vary in size for each edge, we need some form of feature aggregation (pooling operation) to produce something of consistent shape for classification. As first attempts, we tried:

1. Summation: $\sum_{i=1}^{n} f(x_i)$ - the more nearby trajectory points an edge has, the more likely it's in the G.T.

2. Averaging: $\frac{1}{n} \sum_{i=1}^{n} f(x_i)$ - erases information on number of points, but more regularity.

Overall, these pooling operations did not work well: the model tended towards voting all one class or the other, or some 50:50 mixture of predictions. This alone does not suggest these pooling operations are unideal: class imbalance may be to blame as the GPS trajectory edges comprise less than 10% of all the edges that are a GPS K-NN neighbor (so even fewer of the total edges in the extracted subgraph). To test whether class imbalance was to blame or these pooling operations are genuinely bad, a variety of losses were considered, including Binary Cross Entropy (BCE), a reweighted BCE (RBCE) to penalize mispredicting ground truths more, and the Focal Loss [Lin+17]. The latter two aim to account for class imbalance more; the Focal Loss is a further improvement upon the RBCE as it penalizes mispredictions more and maintains a reweighting factor. Define the y-switch operator for ease of notation:

$$(x)_y = x_y = \begin{cases} x & : y = 1 \\ 1 - x & : y = 0 \end{cases}$$

The loss terms for each model output x and corresponding ground truth label y are as follows:

$$\text{Binary Cross Entropy:} \quad \text{BCE}(x, y) = -y \log(x_y) - (1 - y) \log(1 - x_y) \tag{5.1}$$

$$\text{Reweighted BCE:} \quad \text{RBCE}(x, y) = \alpha_y \text{BCE}(x, y) \quad : \quad \alpha \in (0, 1) \tag{5.2}$$

$$\text{Focal loss:} \quad \text{FL}(x, y) = -\alpha_y (1 - x_y)^\gamma \log(x_y) \quad \text{[Lin+17]} \tag{5.3}$$

The summation and averaging pooling layers were tried with all of these losses, but none of these yielded any improvements in classification accuracy: hence, some other form of feature aggregation is necessary.

To this end, we implemented a (seemingly) new and novel form of pooling for point cloud networks. Instead of relying on a single extracted feature (mean, average), we instead extract several features that maximize or minimize a selected measure of similarity. We utilized the l^1 norm, but others are feasible (i.e. squared l^2). The procedure is thus: Extract k features with maximal l^1 norm and k features with minimal l^1 norm:

$$I_{top} = k\text{-argmax}_i \|X[i]\|_1; \quad I_{bot} = k\text{-argmin}_i \|X[i]\|_1$$

Where k-argmax/min returns the indices of the k highest and lowest values, respectively. We utilized $k = 10$ for our implementation. Then, concatenate these values into a single array:

$$V = (X[I_{top}], X[I_{bot}])$$

Set $m = 2k$. Extract:

1. Dot products: $A_{ij} = V_i \cdot V_j : i < j$ ($\frac{m^2 - m}{2}$ values).

2. l^1 Distances: $D_{ij} = \|V_i - V_j\|_1$ ($\frac{m^2 - m}{2}$ values).

3. l^1 magnitudes: $M_i = \|V_i\|_1$ (m values).

4. Vectors: V ($m * n$ (feat dim) values).

Flattening all these arrays and concatenating for a single vector yields a total of $4k^2 + 2kn$ output values. But as mentioned before, some edges have very small point-sets: what is to be done when there are not k values?

To remedy this matter, we implement a padding operation for the pooling layer: if there are fewer than k points at an edge, we pad the output indices by repeating the index corresponding to the max/min value for I_{top} / I_{bot} (respectively). This introduces 0's into the extracted distances, i.e. we tell the model that there are not many points coinciding at the edge, which can be incorporated into the decision making process. This is analogous to the positional encodings used for transformers. If there are fewer than $2k$ points: redundancies are still had, and 0's appear in different places of the distances, so the model can still be "aware" that there are still realtively few points nearby.

For the final implementation, we used a 3-layer encoder: $\mathbb{R}^4 \to \mathbb{R}^{60} \to \mathbb{R}^{120} \to \mathbb{R}^{60}$, which corresponds to an output with 1600 values after pooling. From here, another 3-layer neural network serves as the classifier: $\mathbb{R}^{1600} \to \mathbb{R}^{60} \to \mathbb{R}^{30} \to \mathbb{R}^1$ for the output probability. All layers used Rectifed Linear Units (ReLU) for activation except the last layer, which used the sigmoid function for a 0-1 probability value.

We also briefly examined normalizing vs. not normalizing the projected sets by dividing by the length of the edge, and using the absolute distance instead of oriented distance. This necessitates further experimentation, as does the use of gradient clipping, dropout, and batch normalization to mitigate overfitting and aid training.

5.3 Dijkstra's Algorithm From Edge Affinity Weights

The edge affinity model produces 0-1 probabilities, which we must convert to a weighting for Dijkstra's algorithm. Dijkstra's seeks to follow the path with the lowest weight, so we must invert the spectrum and add

some nonlinearity to heighten the difference between 0's and 1's (0's should become almost non-traversable, i.e. very high weight, 1's should have 0 weight). Thereby, we construct the weights as thus:

$$W(p) = -\log(p + \epsilon) - \log(1 + \epsilon)$$

adding the ϵ to avoid infinities and the second term to ensure everything is positive; $\epsilon = 10^{-6}$ worked well for our experiments.

For implementation, we assume the starting and ending nodes are the map vertices closest to the start and end GPS points - there's possible room for improvement here, but time was a limiting factor. Dijkstra's algorithm is then used to extract the shortest path in between the points.

5.4 Computational Implementation

All code was implemented in Python. All edge affinity models were implemented in PyTorch; Scipy's sparse implementation of Dijkstra's algorithm was used, and Numpy was used for intermediary processing. The 100 trajectories in the KCMMN dataset [Kub+15a] were split on a 80:20 training to testing split, which corresponds to 251,095 edges with non-empty point sets for training. The latitude-longitude coordinates were projected from WGS-84 (EPSG:4326) to OpenStreetMaps meter coordinates (EPSG:3857) for computation of distances. EPSG:3857 was used as it is a global frame and the KCMMN trajectories are in both Europe and North America - notably this could lead to distortion, and better local coordinate systems should be considered for possible improvements.

Our final model was trained for 10 epochs on the BDD100K dataset at the Minnsota Super-computing Institute on a single A-100 GPU - this took about 3 hours. Further training was halted due to system maintenance, but this seems to have been sufficient.

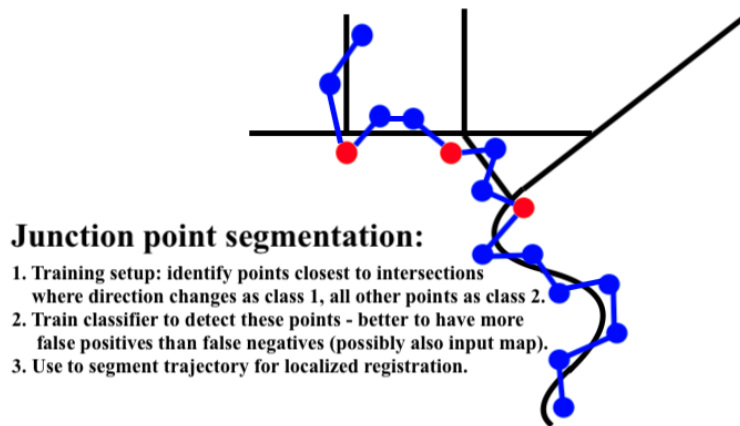### 5.5 Trajectory Segmentation Via Junction Classification

Figure 14: Junction point identification for GPS trajectory segmentation.

As suggested by Gabriel Gress, the ability to cluster GPS trajectories into segments prior to registration would be a highly computationally adventitious innovation as it would drastically reduce the number of candidate paths and branches that need to be examined (with Dijkstra's or otherwise) and also give a smaller area of candidate paths. This would also make the computation parallelizable (with some after-the-fact path connection verification and possible refinements, if necessary).

One could examine modern deep clustering techniques [Ren+22], but these tend to be more interested in clustering each input into a corresponding cluster - not clustering each input into several clusters. Moreover, this is more akin to a segmentation problem. However, if one deliberates the matter a little further, the 1-D segmentation problem boils down to identifying key junction points - i.e. where one turns from one road segment to another - which is easily regarded as a binary classification problem.

To train such a model requires verified trajectories - i.e. the [Kub+15a] dataset. A training set is easily assembled by identifying the GPS points closest to the road junctions where turns occur as the juncture points, and all other points as non-juncture points. Naturally, the identification depends on the GPS sampling frequency, and if dense around the junction points, one could enlarge the juncture point class (if within d distance and K-nearest neighbors of the point). The model could simply be a GNN, RNN, or transformer. Once trained, it can immediately be used for identifying possible junction points. In practice, it is better to have a model that identifies all true-positives with several false-positives than miss several true-positives, and this should be considered when formulating the loss.

5.6   Stay-Point and U-Turn Detection

The matter of determining when an entity has ceased moving and when it has taken a U-turn (particularly within a road segment) is of critical import for preprocessing our GPS trajectories, and time permitting, we will explore deep-learning approaches to identify/cluster such regions of the GPS timeseries. It is highly conceivable to hand-craft such detection using heuristics, but [JEN22] employed Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to detect them, which achieved significant data reduction and processing time with the same accuracy. It is unclear at this time how to improve upon DBSCAN - an algorithm that has earned the "Test of Time" Award, but we could attempt to do so by training a network to leverage the unique timeseries-graph structure of GPS trajectories. It is unclear how such would be trained in practice, but the matter merits further investigation.

6   Map Matching Algorithms

In this section we propose two map matching algorithms: AHP map matching algorithm and Fuzzy logic map matching. Our algorithms are classified as online map matching (Problem 2.10). They find the correct edge for each trajectory point. The flowchart of algorithms is shown in Figure 15. Algorithms are divided into three phases: initial map matching process (IMP), subsequent map matching process along a link (SMP1), and subsequent map matching process at a junction (SMP2). Once input is obtained, we run IMP to find the correct edge for the first (few) trajectory point(s). After IMP, we run SMP1 to verify whether the next point still matches the same edge as the previous point. If yes, we repeat SMP1 for the next point. If not, we execute SMP2 for that point to find the correct edge. Then we go back to SMP1 and continue this process until we reach the last trajectory point. The details of each algorithm are explained in the following Subsections 6.1, 6.2.

6.1   AHP Map Matching Algorithm

The AHP stands for the analytic hierarchy process. This is a decision-making method that combines mathematical analysis with human judgment. It utilizes hierarchical classification to deal with complex and abstract information. AHP has not been used much for map matching. To the best of our knowledge, the only paper that mentions AHP is the one by [Mah+22], but even then different methods are used. So this algorithm is relatively new and also simple and easy to understand.

The AHP is incorporated into IMM and SMP2 parts (Figure 16). As input, we use a road network, trajectory points, and the speed and direction data of each trajectory point.

6.1.1   Initial Map Matching Process (IMP)

The purpose of the initial map matching process (IMP) is to specify the first matching. The IMP takes the following steps:

1. Identify a set of candidate edges.

2. Assign a weight to each candidate edge using AHP based on distance and direction data.

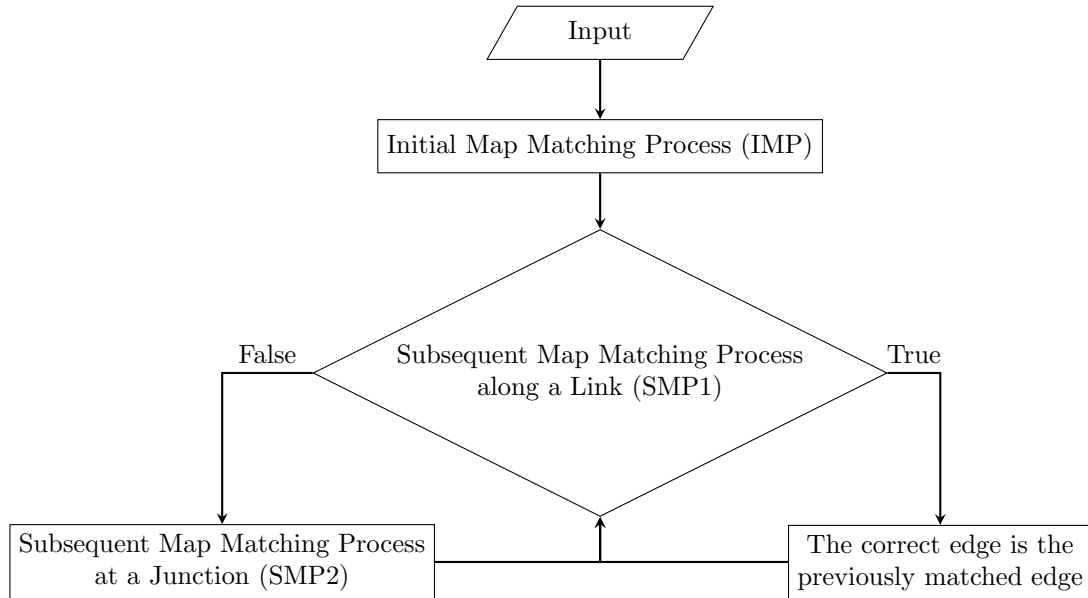3. Take the highest weight edge as the correct edge for that point.

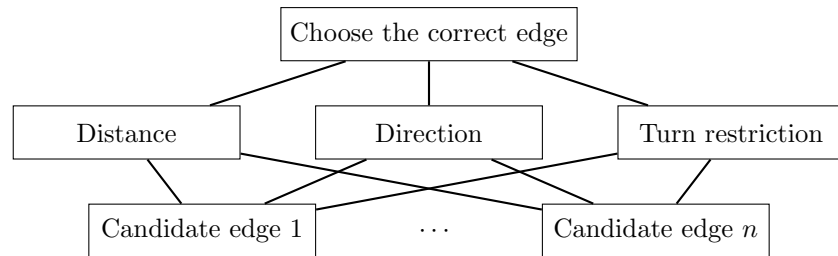Figure 15: The flowchart of AHP map matching algorithm

Figure 16: AHP layer

Table 1: The pairwise comparison matrix for distance. $d_i \coloneqq \mathrm{dist}(p_t, d_i)$

α_{ij}	range
1	$0 \le d_j - d_i \le 1$
2	$1 < d_j - d_i \le 3$
3	$3 < d_j - d_i \le 5$
4	$5 < d_j - d_i \le 7$
5	$7 < d_j - d_i \le 9$
6	$9 < d_j - d_i \le 11$
7	$11 < d_j - d_i \le 13$
8	$13 < d_j - d_i \le 15$
9	$15 < d_j - d_i$
$1/\alpha_{ji}$	$d_j - d_i < 0$

Table 2: The pairwise comparison matrix for direction. θ_i is the angle difference between the direction of p_t and the direction of e_i

β_{ij}	range
1	$0 \le \theta_j - \theta_i \le 10$
2	$10 < \theta_j - \theta_i \le 30$
3	$30 < \theta_j - \theta_i \le 50$
4	$50 < \theta_j - \theta_i \le 70$
5	$70 < \theta_j - \theta_i \le 90$
6	$90 < \theta_j - \theta_i \le 110$
7	$110 < \theta_j - \theta_i \le 130$
8	$130 < \theta_j - \theta_i \le 150$
9	$150 < \theta_j - \theta_i$
$1/\beta_{ji}$	$\theta_j - \theta_i < 0$

In the first stage, we check whether the speed of the vehicle at the first point p_0 is less than 3m/s or not. If yes, we skip the analysis of p_0 and run IMP for the next point p_1. This is because if the vehicle speed is less than 3m/s, the GPS data is less reliable [Tay+01; OQN03]. We continue this speed check until the first point where the speed is more than 3m/s is obtained. Once such a point, say p_t, is obtained, next we draw the error polygon and take edges that intersect or are contained in this polygon as candidate edges. So candidate edges are the edges that have the possibility of matching p_t. Let $e_1, \ldots, e_n$ be the candidate edges. After identifying the candidate edge, we define a weight for each candidate edge using AHP. A weight is given based on distance data and direction data of each candidate edge. To do this, we first consider weights $w_1^{\mathrm{dist}}, \ldots, w_n^{\mathrm{dist}}$ for distance. To begin we construct the pairwise comparison matrix $M_{\mathrm{dist}} = [\alpha_{ij}]$ for distance. Each (i, j)-component of M_{dist} is defined by Table 1. Here, the distance $\mathrm{dist}(p, e)$ between a point p and an edge e is defined by

$$\mathrm{dist}(p, e) \coloneqq \sup_{x \in e} \mathrm{d}(p, x),$$

where $\mathrm{d}(p, x)$ is the usual euclidean distance between two points (Figure 17). Then we take the geometric mean g_i of each row and let $S = g_1 + \cdots + g_n$. Finally the weight w_i^{dist} for distance of the candidate edge e_i is determined by $e_i = g_i/S$. We repeat the same procedure to obtain weights $w_1^{\mathrm{dir}}, \ldots, w_n^{\mathrm{dir}}$ for distance using the hierarchical classification of Table 2. Now each candidate edge has two weights: one for distance and another for direction. Finally the total weight $\mathrm{TW}(e_i)$ for e_i is defined by

$$\mathrm{TW}(e_i) \coloneqq c^{\mathrm{dist}} w_i^{\mathrm{dist}} + c^{\mathrm{dir}} w_i^{\mathrm{dir}},$$

where c^{dist} and c^{dir} are the coefficients that reflect the relative importance of the factors. These values are provided by [VQB12] and given by Table 3, which vary depending on the map environment (see also subsection 6.1.4). After these processes we select the highest weight edge as the correct edge for p_t.
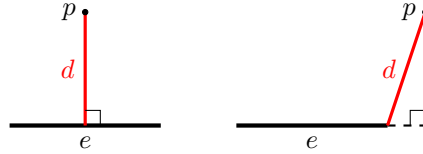
Figure 17: Distance $\text{dist}(p, e)$ between a point and an edge

Table 3: Relative importance

	urban	suburban	rural
c^{dist}	0.0806	0.4376	0.5563
c^{dir}	0.3715	0.4642	0.4237
c^{turn}	0.5479	0.0982	0.020

6.1.2 Subsequent Map Matching Process along a Link (SMP1)

After each matching, we always run the subsequent map matching process along a link (SMP1) for the next trajectory point. In SMP1 we verify whether the next point matches the previously selected edge. Let p_t be the current point, e be the previously selected edge for the previous point p_{t-1}, and q_{i-1} be the projection point of p_{t-1} onto e. At first, if the speed of p_t is zero, then we automatically conclude that p_t matches e. In other cases, we use two factors to determine this:

1. How far p_t is from the next junction point.

2. How small the angle difference between the direction of p_t and that of p_{t-1} is.

For the first part, we set d_1 as the distance between q_{i-1} and the next junction point and d_2 as the product of the speed of p_{t-1} and time interval between p_{t-1} and p_t. We then define $\Delta d := d_1 - d_2$. For the second part, we calculate Δh as the angle difference between the direction of p_t and that of p_{t-1}. If both conditions $\Delta d > 30$ and $\Delta h < 5$ are satisfied, then we conclude that p_t still matches the previously selected edge. Otherwise, we proceed to the subsequent map matching process at a junction.

6.1.3 Subsequent Map Matching Process at a Junction (SMP2)

The subsequent map matching process at a junction (SMP2) is executed only for the point that does not meet the SMP1. SMP2 takes a similar process as IMP, which is:

1. Identify a set of candidate edges.

2. Assign a weight to each candidate edge using AHP based on distance, direction, and turn restriction data.

3. Take the highest weight edge as the correct edge for that point.

Let p_t be the current point. The first part is exactly the same as the one in IMP and we assume that $e_1, \ldots, e_n$ are candidate edges. In the second part, similar processes used in IMP are also employed except that we use a new factor, which is turn restriction data, in addition to distance and direction data. So the method for obtaining the weights for distance and direction is the same as for the IMP. The turn restriction data reflects if the vehicle on the previously selected edge can legally turn onto each candidate edge, that is if each candidate edge is connected to the previously selected edge and is not the wrong way down a one-way street. The pairwise comparison matrix $M_{\text{turn}} = [\gamma_{ij}]$ for turn restriction is defined by Table 4, and weights $w_1^{\text{turn}}, \ldots, w_n^{\text{turn}}$ for turn restriction are obtained the same way as the other two weights. At this stage, each candidate edge has three weights: one for distance, another for direction, and the third one for turn restriction. Then the total weight $\text{TW}(e_i)$ for e_i is defined by

$$\text{TW}(e_i) := c^{\text{dist}} w_i^{\text{dist}} + c^{\text{dir}} w_i^{\text{dir}} + c^{\text{turn}} w_i^{\text{turn}},$$

where the coefficients $c^{\text{dist}}, c^{\text{dir}}$, and c^{turn} are given by Table 3. Finally, the highest weight edge is selected as the correct edge for p_t.

Table 4: The pairwise comparison matrix for turn restriction

γ_{ij}	range
1	If the vehicle can (or cannot) legally turn onto both e_i and e_j
9	If the vehicle can legally turn onto e_i and cannot turn onto e_j
1/9	If the vehicle can legally turn onto e_j and cannot turn onto e_i

6.1.4   Map Environment

In IMP and SMP2, we have to specify the map environment for each trajectory point to determine which coefficient value should be used. As [VQB12] reported, the importance of each factor depends on the map environment. We determine the map environment for each trajectory point individually rather than for the entire road network because some road networks may have multiple features, such as a combination of urban and suburban areas. The calculation for finding the map environment is based on [VQB12]. First we draw the circle of radius 200 centered at the current point p_t. Next we count the number N of junction points within this area and calculate the total length L (km) of roads within this area. Then the ratio N/L is used for determining the map environment. If the ratio is greater than 6.81 we conclude that the map environment around p_t is urban, if the ratio is smaller than 2.88, then the map environment is assumed to be rural. Otherwise, the map environment is considered suburban.

6.2   Fuzzy Logic Map Matching Algorithm

A Fuzzy-logic Map Matching algorithm utilizes Fuzzy Inference System to make a decision. Three different Fuzzy Inference System (FIS) is implemented for IMP, SMP 1, and SMP 2. Quddus [QON07] proposes using Takagi-Sugeno-Kang that averages out the rule outputs

$$Z = \frac{\Sigma_{i=1}^{N}\omega_i Z_i}{\Sigma_{i=1}^{N}\omega_i}$$

Where ω_i is the rule strength calculated from the fuzzy rules and Z_i is a defuzzifcation function. Since the output of this FIS is the likelihood of matching the position fix to the candidate link. The constants used to calculate the output are 10 when output is low, 50 when output is average and 100 when output is high.

We also introduces a set of weight a_i, that represents on how confident we are about the output result produced by the ith rule, where $\sum_{i=1}^{n} a_i = 1$. Our new proposed rule aggregation is :

$$Z^* = \frac{\Sigma_{i=1}^{N}\omega_i a_i Z_i}{\Sigma_{i=1}^{N} a_i * \omega_i}$$

We will discuss more about the reasoning to include this new weight in section 8.4

6.2.1   Initial Map Matching Process (IMP)

The Initial Map Matching starts by identifying all possible candidate links inside an elliptical error confidence region around position fix (GPS location) based on some error model. FIS is then calculated for all the candidate link and link with the highest FIS score is then selected. IMP processed is then repeated until the matched link is picked for three consecutive position fix. Quddus [QNO06] proposed using four input variables in the IMP step, which includes : 1) Speed of the vehicle, 2) Heading error, 3) perpendicular distance and 4) contribution of satellite geometry to the positioning error, which represented by the horizontal dilution of precision (HDOP). Since KCMMN data set doesn't have a HDOP value, therefore we decided not utilize HDOP as one of our input. Sigmoidal Membership function is chosen in the fuzzification. We then modified the fuzzy rules in order to accommodate our selection of input. We proposes these following rules to calculate fuzzy output in this FIS :
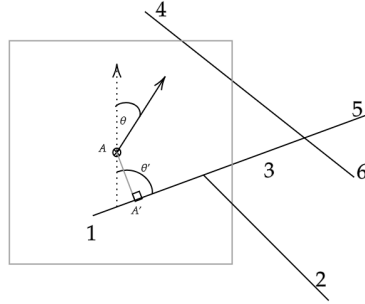
Figure 18: Four candidate link (1, 2, 3, 4) is detected in the error region. Heading error is defined as the difference between θ and θ'

.

1. if speed is high and heading error is small then output is average.

2. if speed is high and heading error is small then output is low.

3. if perpendicular distance is short and speed is high then output is high.

4. if perpendicular distance is long and speed is low then output is low.

5. if perpendicular distance is short and heading error is small then output is high.

6. if perpendicular distance is long and heading error is large then output is low.

The FIS is applied to all links within the confidence region and the link which gives the highest likelihood is taken as the correct link among the candidate link. Since the link for the first position fix may not be the actual link, IMP step can be performed to a few first position fix. If the FIS identifies the same link for those position fixes then the link is chosen as a first correct link.

6.2.2   Subsequent Map Matching Process along a Link (SMP1)

Once the Initial link is chosen, SMP 1 are deployed to track whether the next position fix is still travelling through the current selected link. The input for the FIS are the speed of the vehicle, Heading Increment ($|\theta - \theta'|$), α and β and $\Delta d = d - d_2$, where d_2 speed of the previous position fixed multiplied by the time difference. In this Implementation we followed the rules proposed by Gorte [GPS14] :
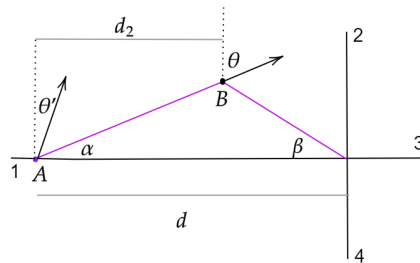

Figure 19: diagram of input in SMP 1 system

1. If α and β is below 90° then output is high.

2. If Δd is positive and α is above 90° then output is low.

3. If Δd is positive and β is above 90° then output is low.

4. If heading increment is small and α and β is below 90° then output is high.

5. If heading increment is small and Δd is positive and α above 90° then output is low.

6. If heading increment is small and Δd is positive and β above 90° then output is low.

7. If heading increment is large and α and β is below 90° then output is low.

8. If speed is zero then output is high.

9. If Δd is negative then output is average.

10. If Δd is positive then output is low.

11. If speed is high and heading increase is small then output is average.

12. If speed is high and heading increase is 180 then output is high.

FIS score above 60 indicates that the current position fix is still travelling in the previous selected link [QNO06].

6.2.3   Subsequent Map Matching Process at a Junction (SMP2)

When the FIS score obtained from SMP 1 step is less than 60, this indicates that the vehicle is entering a junction and SMP 2 is used to determine which link should be selected for the current position fix. SMP 2 algorithm is similar to IMP algorithm , but with two additional input and four additional rules [GPS14]. The two additional input are : 1) the link connectivity (1 when previous link are connected to candidate link, 0 otherwise) 2) distance error, which is defined as the difference between distance travelled by the vehicle and the shortest path travelled through road network(see 20).
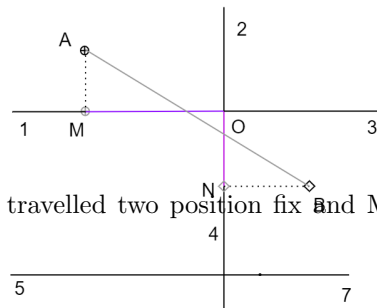

Figure 20: AB is the distance travelled two position fix and MO + ON is the shortest path travelled for candidate link 4

The 4 additional rules that incorporated the connectivity and distance error are:

- If connectivity is low then output is low.

- If connectivity is high then output is high.

- If distance error is low then output is low.

- If distance error is high then output is high.

Similar to the IMP process candidate link with the highest FIS score is selected as the new matched link.

7   Implementation

We implemented AHP and Fuzzy Map matching algorithm using geopanda and Osmnx libraries in python.to evaluate their performance numerically. The code written for the project can be found at : https://github.com/walkre-niboshi/G-RIPS-2023-Mitsubishi-A. For both the AHP and Fuzzy, an error polygon, rather than an ellipsoid, is constructed in order to simplify our calculations. KCMMN dataset is used to test the performance of our algorithm. We use the following method proposed by Newson and Krumm [NK09] :

$$\text{Err} = \frac{d_- + d_+}{d_0},$$

where d_0 is the length of the correct route, d_- is the length of the prediction erroneously subtracted from the correct route, and d_+ is the length of the prediction erroneously added outside the correct route. See figure 21.
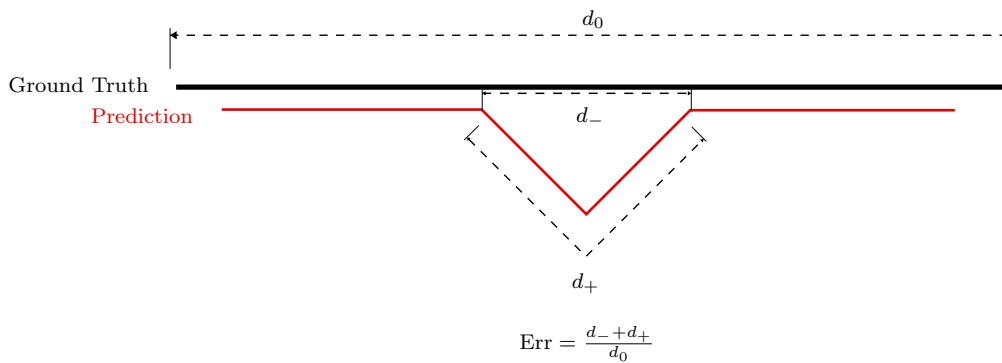
$d_0 = $ length of ground truth

$d_- = $ length of prediction route erroneously subtracted

$d_+ = $ length of prediction route erroneously added

Figure 21: Error Formula by Newson and Krumm

A lower value indicates that our algorithm perform really well in matching trajectory points to a map, while a higher value indicates that our algorithms perform poorly. Both AHP and Fuzzy logic map matching algorithm is tested in 20 trajectories. We then compare our methods to Fast Map Matching (FMM) framework [YG18], which is based on hidden Markov model.

8   Results

8.1   DA-LEAF: Dijkstra's Algorithm with Learnable Edge Affinity Function

8.1.1   Training & Testing

Our model has a very high overall training accuracy of about 93%, wherein 85% of ground truth edges were correctly identified (when thresholding the output probability at .5 to obtain these scores). The unnormalized edge projection models have a lower testing accuracy of 86% at the best, but only correctly identify about 10% of ground truth edges. This could be suggestive of three things: 1. that our model is drastically overfitting, 2. the testing data's distribution does not match that of the training data, or 3. the output testing probabilities require a lower threshold than .5.

As the trajectories are around the globe, the accuracy/distortion is subject to some variation under projection to the OpenStreetMaps meter grid, as well as possible variations in GPS sampling frequency.

These considerations could suggest that the distribution of the testing data is non-identical to the training data, and that more trajectories are needed for global training (KCMMN only has 100 trajectories total). This coincides somewhat with the "No Free Lunch Theorem," i.e. there is no model that can perform well on all possible distributions.

The other (non-exclusive) possibility is overfitting. We have done numerous trials in cutting the number of parameters in the model, but each such cut has yielded lower training *and* testing accuracy. It is not yet clear what part of the model can/should be cut. Further experiments with dropout, batch normalization, and gradient clipping should be done in future work.

8.1.2   Map Predictions

Here we include visualizations of the output model probabilities. In this section, the blue line (or points, when zoomed in) denotes the GPS points, which can be regarded as the ground truth route for most of this section. The colored lines are the map grid color coded by the edge affinity model: increasing from red (low), green, cyan, blue, to purple (high). While the probabilities take values in $[0, 1]$, we have yet to find an efficient plotting implementation in Python that shows the probabilities as a continuous heatmap and allows trajectory point overlays - time is, again, a limiting factor. For evaluation, green and red should be regarded as negative predictions (green a little less confident); cyan, blue, and purple should be regarded as positive predictions (i.e. in the trajectory, with purple the most certain).

Figures 22 to 25 show the results on a trajectory in the training dataset. A close inspection of Figures 22 shows a purple line coinciding with the central GPS points - the model has correctly identified the ground truth here, but there's also relatively little around. Figure 24 shows a close up of the region where the model does quite well, although an offshooting road also has a quite high output probability. Figures 23 and 25 reveal the model fares far worse in dense urban areas, with only a small bit of purple towards the end of the trajectory in 23 and the rather clean cut 25 completely missed.

As for testing, Figures 26 and 27 reveal a far lower spectrum of output values, with very little high predictions, but more green toward the ground truth trajectory than in most other places.
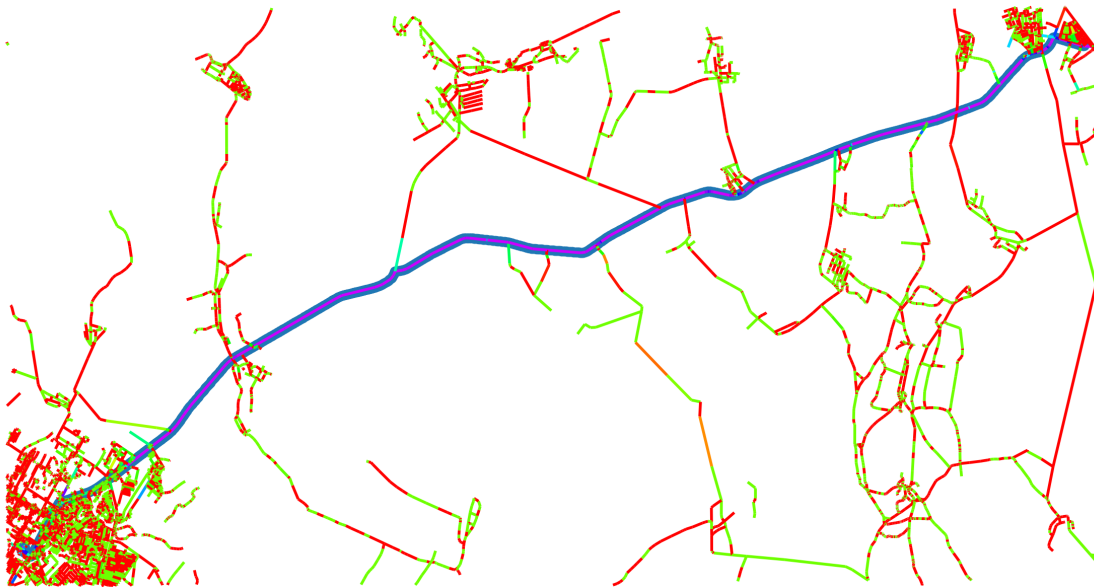

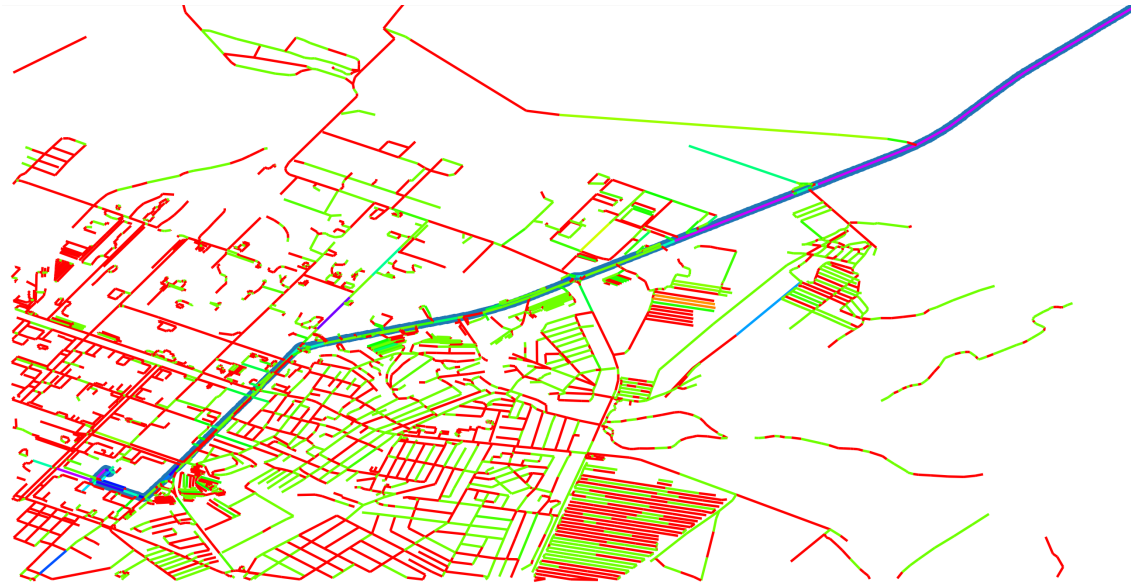Figure 22: Training results: overall, good.

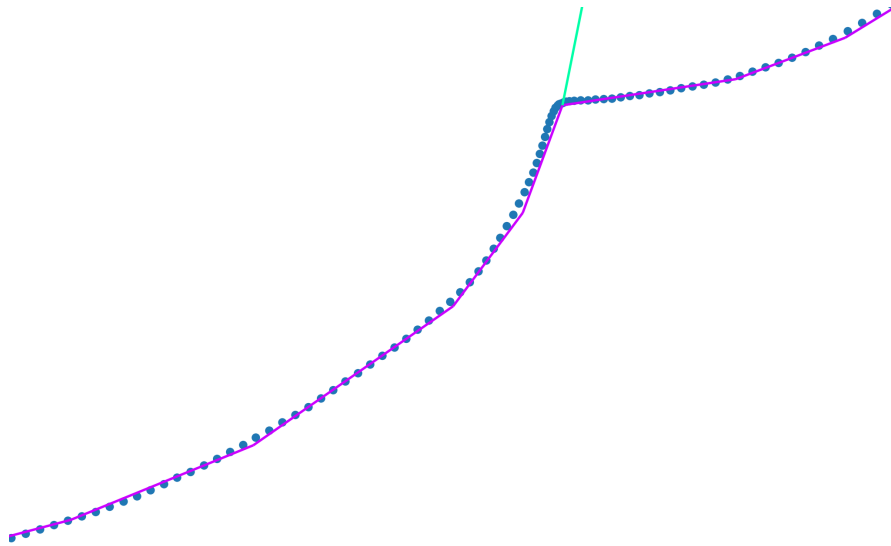Figure 23: Training results in dense urban area: not so good.

Figure 24: Training results: good.

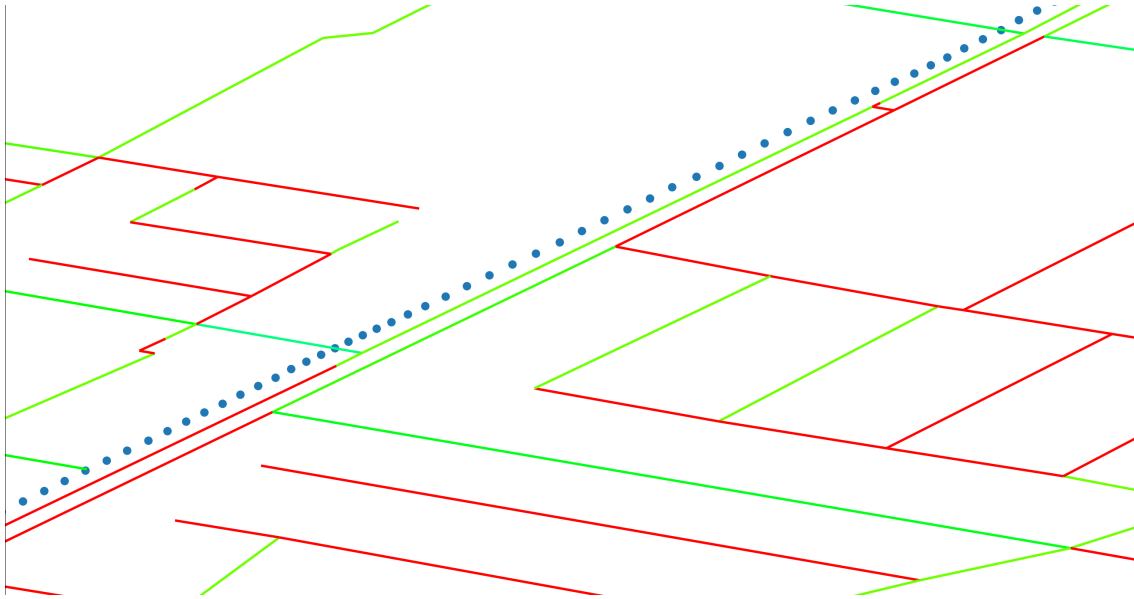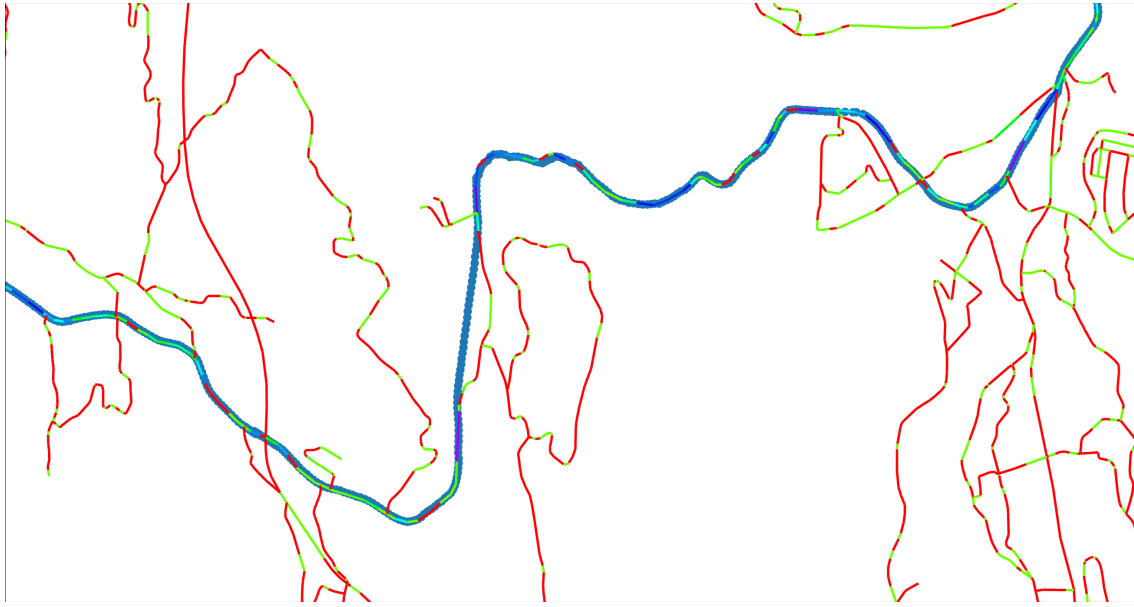Figure 25: Training results: bad.

Figure 26: Testing results.

Figure 27: Testing results.

8.1.3   Use with Dijkstra's Algorithm: DA-LEAF

Despite the poor testing accuracy with thresholding, we elected to try the model with Dijkstra's algorithm as discussed in section 5.3 as it was only a few lines of code more. To our delight, the model performs really quite well, even on testing data. This suggests a lower output probability threshold can and should be used. As time is presently a limiting factor, we cannot produce hard metrics of the algorithm's accuracy at this moment, but we include some of the resulting plots. The model evaluation and Dijkstra's algorithm between the two nodes is quite fast, completing in well under a minute. The computational bottleneck is, at present, constructing the reverse K-NN edge to point adjacency, which still only takes a few minutes, but there is lots of room for improvement here using more sophisticated search methods, such as K-D trees, depth first searches, spatial queries, and other means.

We now present the output trajectory predictions from Dijkstra's algorithm when using our learned edge affinity function (i.e. DA-LEAF). Here, the orange points denote the predicted path colored at the map nodes; the blue is the raw GPS trajectory, and everything else is as in the previous section. Apologies that the plots are a bit hard to see, but again, time is a limiting factor.

Figures 28 to 30 show results on a trajectory in the training set. Overall the algorithm identifies the correct route, but notably Figure 29 shows a departure from the ground truth in the dense urban area. This suggests that there is certainly room for improvement in the model's performance, especially in urban areas. It seems empirically that KCMMN consists largely of longer country drives, with less data representation in urban areas. Mitigating this issue and procuring more urban data should be of quintessential interest for further work, as the trajectory identification is somewhat more trivial in rural areas.

Figures 31 to 35 show the testing results of the algorithm, which appear to be better than the training results. Despite the poor testing accuracy of the thresholded model, DA-LEAF identifies most of the correct route here. Even when the GPS trajectory appears to traverse a non-existent road, the model correctly identifies the "ground truth" path (Figure 35). This would seem to have strong applications in map completion and/or detection of an incomplete route or identification of new routes, which is of tremendous interest to industry research at large. This certainly merits further investigation and development.
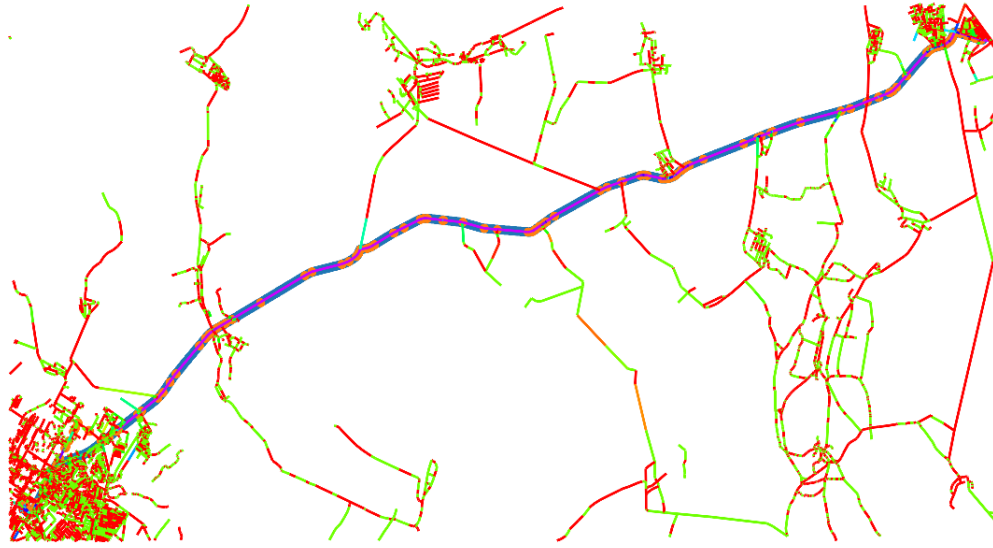
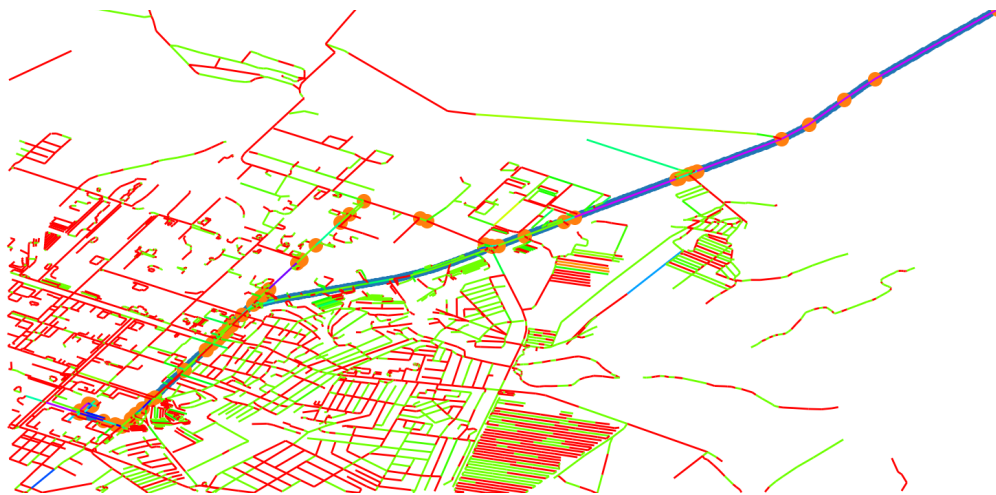Figure 28: DA-LEAF training results.

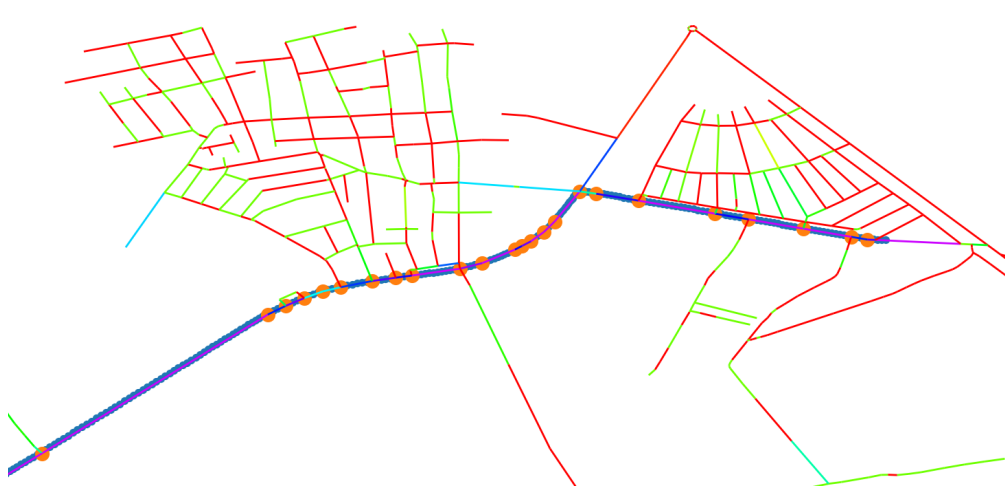Figure 29: DA-LEAF training results.

Figure 30: DA-LEAF training results.
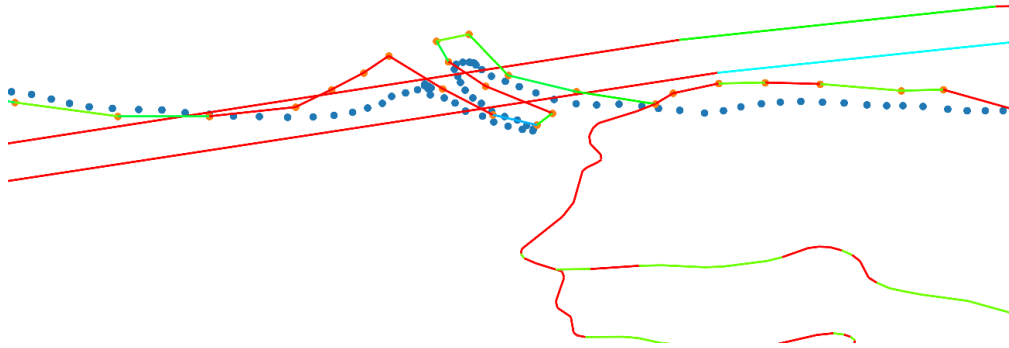

Figure 31: DA-LEAF testing results.

Figure 32: DA-LEAF testing results: the model does quite well with the zig-zag here and various bizzare intersections. Note the road in the lower corner resembles the silhouette of a face.

Figure 33: DA-LEAF testing results: the model is completely correct here, even upon entry into the urban area.
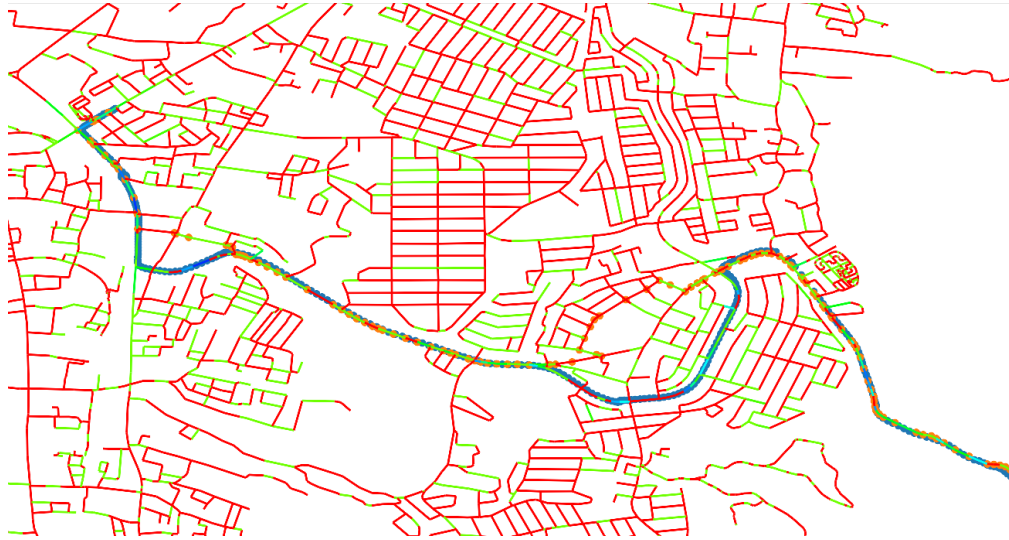
Figure 34: DA-LEAF testing results: the model is almost completely correct here, apart from one snafu within the relatively dense urban area.
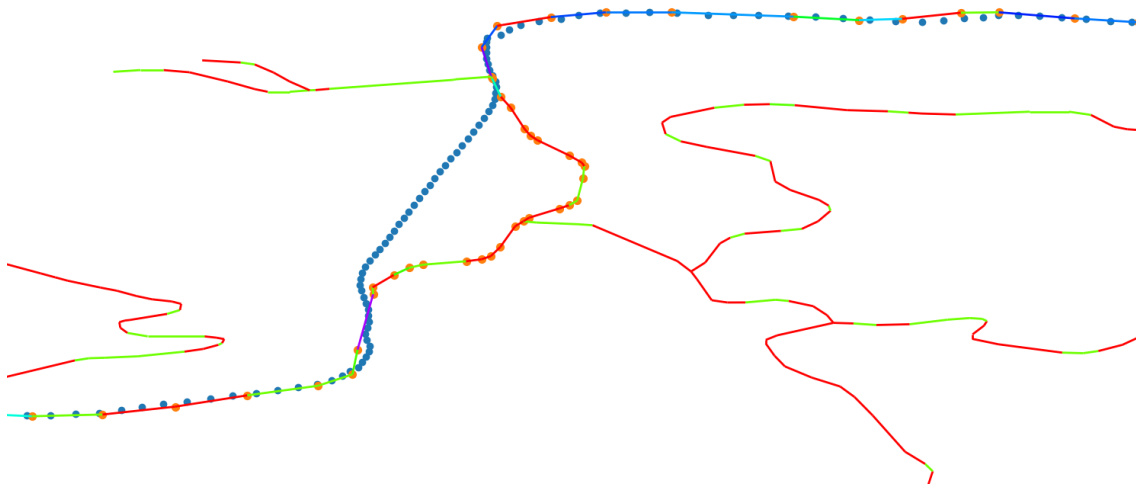

Figure 35: DA-LEAF testing results: the model makes a small mistake here and skips over convex loop in the middle right of this figure. Otherwise, the trajectory is mostly correct.
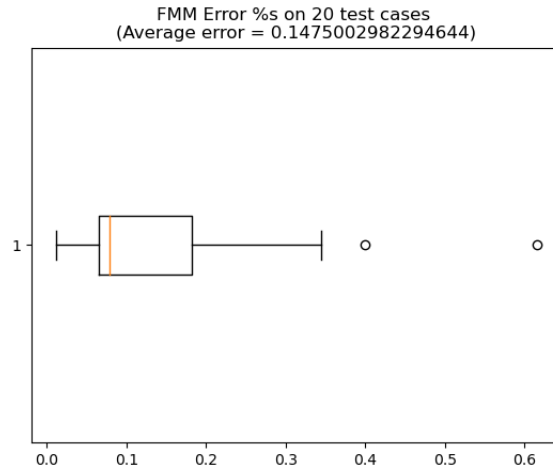
8.2   Performance evaluation
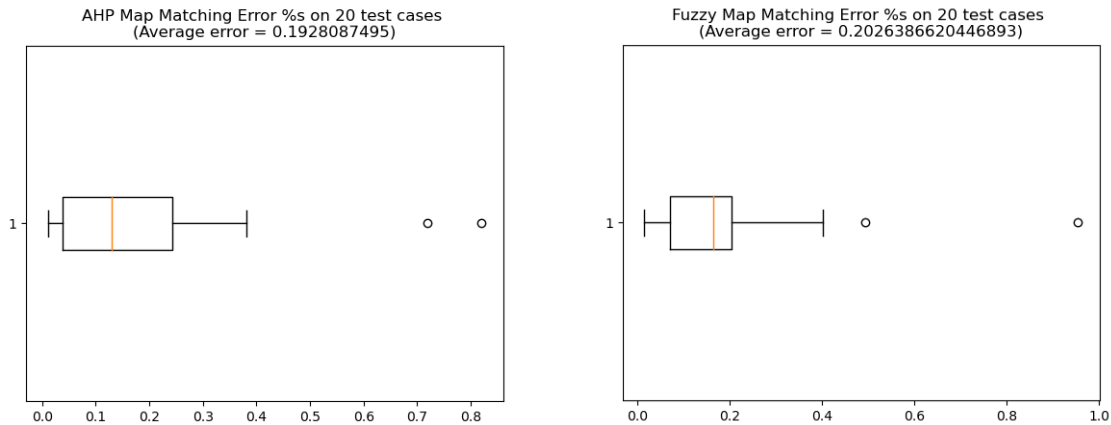

Figure 36: Performance of FMM

Figure 37: Performance of AHP map matching   Figure 38: Performance of fuzzy logic map matching

From the result shown in Figure 36, 37, and 38, we observe that our methods perform relatively well with 19 and 20 percent error for AHP and Fuzzy logic, respectively. However both of our methods still were not able to beat FMM, which we believe due to two following reasons: data availability and parameter tuning. We used KCMMN data set to test the performance of our method because this data set included ground truth, however KCMMN data set only has the GPS position and time, while both our methodologies require other inputs such as speed and direction data. To circumvent this problem, we estimated both speed and direction using the location data. We believe that this estimation causes our method to perform poorly on some trajectories that have more noisy measurements. Due to time limitations, we were not able to optimize the parameter that are needed for our map matching algorithm. We believe that our algorithm will be able to perform significantly better if we were able to address these problems.

8.3   AHP Map Matching Results

One of the results of AHP map matching algorithm is shown in Figure 39. The yellow line represents the true traveling route, the black line represents the matching result, and the red points behind the two lines

represent trajectory points. Except for the big mismatch in the middle left of the figure, the entire outline looks satisfactory. However if we zoom in on local areas we can find some mismatching such as branches (Figure 40) and jumps (Figure 41). These mismatching are more likely to occur near junctions and around points where there are many curves.

The algorithm has both advantages and disadvantages. First, It is simple and easy to understand, which helps us with the implementation. In terms of speed it is comparable to other existing algorithms and we believe that the execution speed could be much higher through the use of other programming languages and parallelization techniques. Furthermore, the accuracy is not so bad, at least it provides results that allow us to grasp the overall outline.

As for the disadvantages, it is very sensitive to measurement errors since we use only two or three factors in each step. If any of these data is inaccurate, it could directly affect the outcomes and result in bad mismatching. Also there is a large variation in accuracy depending on the trajectory data. When the trajectory points are placed linearly, the algorithm tends to work well but when the trajectory points are curved, mismatching are more likely to occur.

To improve the algorithm, we need to find a way to deal with those mismatching. But we believe it would be challenging to avoid mismatching solely with our algorithm. Therefore, implementing post-processing will be necessary, which is explained in Appendix 9.2.
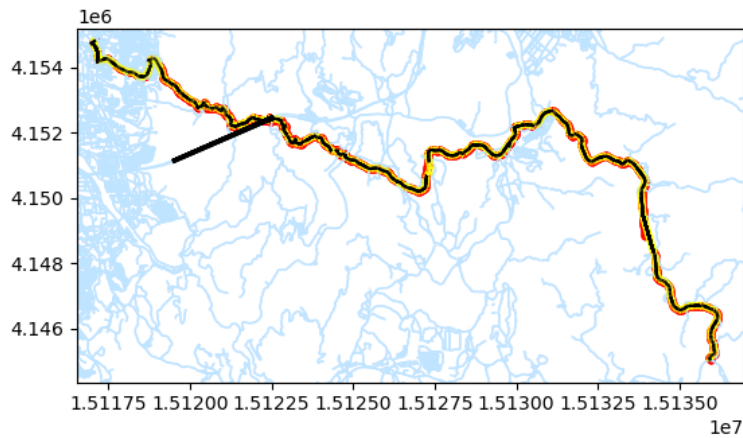

Figure 39: The result of AHP map matching
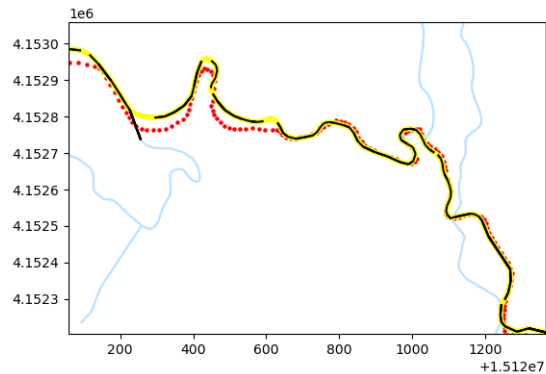

Figure 40: Some branches

Figure 41: Some jumps
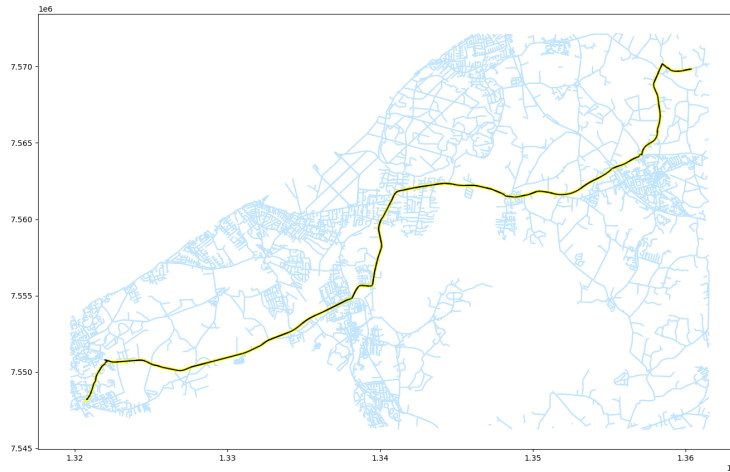
8.4   Fuzzy Logic Map Matching Results

Figure 42: Fuzzy logic map matching process in KCMMN data set

Figure 42 shows an example of using fuzzy logic map matching. The yellow line represents the ground truth and the black line represents the road selected by our algorithm. Similar to AHP, Fuzzy logic Map Matching also suffers to branches and jump that occurs due to selecting incorrect link.

We introduces a rule weight a_i to indicate how confident we are with a certain rules. For example in our KCMMN implementation, since we know that speed and velocity are estimated rather than measured, we can put higher weight on connectivity and perpendicular distance and less on speed and velocity. Figure 43 shows the results of Fuzzy logic map matching before we implemented the rule weight. Blue line represents the GPS trajectory data, black line represents which link is selected and green point represent where the trajectory point is matched to the selected link. We observe that on some of the trajectory points are matched to the wrong link that caused the branches to occurs. Figure 44 shows the results after we implemented the rule weight in our FIS system, her we observe that the trajectory is matched correctly even in the complicated junctions.

During our implementation of fuzzy logic map matching, we notice that although this method performs as well as other methods, it is less sensitive to error when some subsets of the input is miss-specified. Fuzzy logic map matching error rate increase by 10 percentage point when the direction was incorrectly specified, which is relatively lower than our other method that differ by 60 percentage point.

One disadvantage of Fuzzy Logic map matching is that the computational time is relatively slower than both FMM and AHP method. One possible solution is to implement this algorithm in C manually in order to accelerate the computing time.
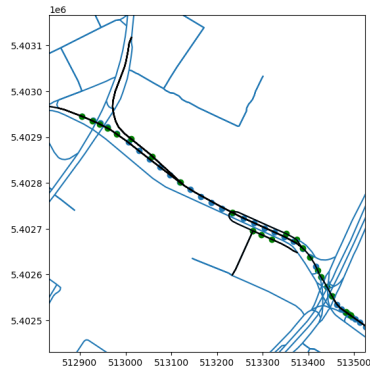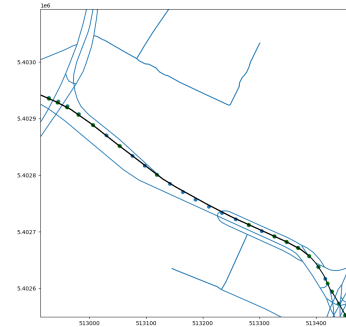
Figure 43: Before

Figure 44: After

9 Appendix

9.1 Preprocessing by DBSCAN

In this subsection, we talk about stay point mitigation and outlier detection by DBSCAN, and a method to automatically determine a parameter of the algorithm.

DBSCAN is a clustering algorithm based on metric information. The algorithm has two parameters: **minPts** and **eps**. Then the algorithm takes a set of points in space as its input and classifies them into three categories: core points, reachable points, and unreachable points. Among these classes, core points and reachable points belong to a cluster, meanwhile, unreachable points do not belong to any cluster. The **minPts** parameter is the density threshold for a point to become a core point, and the **eps** parameter is the search radius for each point. In [Jaf22], they used DBSCAN to detect and mitigate stay points by replacing each cluster with a single point. Here, inspired by their method, we propose to utilize DBSCAN to detect outliers simply by labeling the unreachable points of DBSCAN as outliers. Since many map matching algorithms depend on the number of trajectory points for their computation time, conducting DBSCAN as preprocessing would speed up those algorithms.

Although these methods of stay point mitigation and outlier detection are simple, the choice of the parameters of the algorithm is critical when actually conducting the algorithm as preprocessing of the map matching algorithm. If the **eps** parameter is too small, too many points may be classified as unreachable points. On the other hand, if the **eps** parameter is too large, an undesirably large portion of points may be grouped into a single cluster. As for the **minPts** parameter, in [San+98] they suggested setting it to $2 \times d$, where d is the dimension of the space. As for the **eps** parameter, though such a simple way to determine the value is not known unlike **minPts**, the following heuristic based on "elbows" is said to be a good way [Est+96]:

1. Prepare an empty list **l**;

2. For each point in the input set, compute the distance between the point and the **minPts**-th closest point from it, and put it into **l**;

3. Sort **l** in ascending order;

4. Plot the set $\{(i, \mathbf{l}[\mathbf{i}]) : 0 \leq i < \mathbf{len}(\mathbf{l})\}$ in a plane;

5. Consider a curve that interpolates the plotted points;

6. The elbows, where the curve rapidly goes up, are considered to be good candidates for the **eps** parameter.

Algorithm 2: Elbow Detection Algorithm

Input: List $\mathbf{l}$ of the **minPts**-distances, degree θ, integer d

Output: Set of real numbers

1 Set **points** $= \{\{(i, \mathbf{l}[\mathbf{i}]) : 0 \le i < \mathbf{len}(\mathbf{l})\}\}$; Rotate **points** $-\theta$ degree around the origin;

2 Compute the least square polynomial fit of degree d of the rotated points;

3 Find the local minimals m of the fitting polynomial;

4 Rotate m θ degree around the origin;

5 Extract points whose x-coordinate is in $[0, \mathbf{len}(\mathbf{l}))$ from m, and return their x-coordinates as the output;

When one tries this heuristic, it poses a problem how to determine the interpolation curve and elbows.

To deal with this problem, we propose an elbow detection algorithm in Algorithm 2.

Although this algorithm works well for GPS trajectory points as far as we manually checked with our eyes, it is desired that it is verified in various different tasks. Also, because we tested only the least square polynomial fit as the smooth curve which interpolates the points in this research project, it may be possible that other methods are superior to it. Therefore, it remains as tasks that should be examined in the future to develop a method to find proper values for θ and d, and to test other methods of the interpolation curve.

9.2 Postprocessing

As we have seen in the sections of the AHP and fuzzy-logic map matching algorithms, to achieve greater accuracy, it is necessary to deal with many "jumps" and "branches" that the route predicted by these algorithms includes. We suggest after the map matching algorithm applying postprocessing to the predicted route so as to interpolate the jumps and remove the branches. Specifically, we propose the following procedure:

1. Find the shortest path from the start point to the end point of the predicted route;

2. Subtract the edges in the shortest path from the set of the edges in the predicted route by the map matching algorithm;

3. Remove all connected components of the shape of a linear graph.

The reason why the removed connected components have to be linear graphs in step 3 is that the correct route may have some loops in it, and thus, we do not want to mistakenly remove them.

References

[Aka+22] Tomoya Akamatsu, Gabriel Gress, Katelynn Huneycutt, and Seiya Omura. g-RIPS Sendai 2022 Mitsubishi-A group final report. https://www.mccs.tohoku.ac.jp/g-rips/report/pdf/MITSUBISHI-A_FinalReport.pdf. 2022.

[Bou00] Mirielle Boutin. "Numerically Invariant Signature Curves". In: *International Journal of Computer Vision*. Springer. 2000.

[Bro+23] Jason Brown, Riley O'Neill, Jeff Calder, and Andrea L Bertozzi. "Utilizing contrastive learning for graph-based active learning of SAR data". In: *Algorithms for Synthetic Aperture Radar Imagery XXX*. Vol. 12520. SPIE. 2023, pp. 181–195.

[Cha+20] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. "A survey on map-matching algorithms". In: *Databases Theory and Applications: 31st Australasian Database Conference, ADC 2020, Melbourne, VIC, Australia, February 3–7, 2020, Proceedings 31*. Springer. 2020, pp. 121–133.

[Che+20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[Dij59]   Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[Env23]   EnviroCar. *EnviroCar Data.* https://envirocar.org/analysis.html?lng=en. 2023.

[Est+96]   Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[GPS14]   Nikolai Gorte, Edzer Pebesma, and Christoph Stasch. "Implementation of a Fuzzy Logic Based Map Matching Algorithm in R". In: (2014).

[Gre02]   Joshua Greenfeld. "Matching GPS observations to locations on a digital map". In: 2002.

[GS20]   Ajay Kr Gupta and Udai Shanker. "Study of fuzzy logic and particle swarm methods in map matching algorithm". In: *SN Applied Sciences* 2.4 (2020), p. 608.

[Hig]   High-assurance Mobility Control Lab. *Autonomous Driving.* https://hmc.unist.ac.kr/research/autonomous-driving/.

[HK16]   Mahdi Hashemi and Hassan A. Karimi. "A Machine Learning Approach to Improve the Accuracy of GPS-Based Map-Matching Algorithms (Invited Paper)". In: *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. 2016, pp. 77–86. DOI: 10.1109/IRI.2016.18.

[HO14]   Daniel J Hoff and Peter J Olver. "Automatic solution of jigsaw puzzles". In: *Journal of mathematical imaging and vision* 49 (2014), pp. 234–250.

[Jaf22]   Minoo Jafarlou. "Improving Fuzzy-Logic based Map-Matching Method with Trajectory Stay-Point Detection". In: *arXiv preprint* (2022). arXiv: 2208.02881 [cs.LG].

[JEN22]   Minoo Jafarlou, Omid Mahdi Ebadati E., and Hassan Naderi. "Improving Fuzzy-Logic based Map-Matching Method with Trajectory Stay-Point Detection". In: *arXiv preprint* (2022). arXiv: 2208.02881 [cs.LG].

[Jou98]   Lionel Jouffe. "Fuzzy inference system learning by reinforcement methods". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 28.3 (1998), pp. 338–355.

[Kub+15a]   Matěj Kubička, Arben Cela, Philippe Moulin, Hugues Mounier, and Silviu-Iulian Niculescu. "Dataset for testing and training of map-matching algorithms". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 1088–1093.

[Kub+15b]   Matěj Kubička, Arben Cela, Philippe Moulin, Hugues Mounier, and Silviu-Iulian Niculescu. *Dataset for testing and training of map-matching algorithms: website.* Public GPS Trajectories. https://zenodo.org/record/57731. 2015.

[Lin+17]   Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[Liu+20]   Zhijia Liu, Jie Fang, Yingfang Tong, and Mengyun Xu. "Deep learning enabled vehicle trajectory map-matching method with advanced spatial–temporal analysis". In: *IET Intelligent Transport Systems* 14.14 (2020), pp. 2052–2063.

[Mah+22]   Alireza Mahpour, Hossein Forsi, Alireza Vafaeenejad, and Arman Saffarzadeh. "An improvement on the topological map matching algorithm at junctions: a heuristic approach". In: *International journal of transportation engineering* 9.4 (2022), pp. 749–761.

[MST22]   Marcella Manivel, Milena Silva, and Robert Thompson. "Iterative Respacing of Polygonal Curves". In: *SN Computer Science* 3.5 (2022), p. 419.

[NK09]   Paul Newson and John Krumm. "Hidden Markov Map Matching through Noise and Sparseness". In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '09*. The 17th ACM SIGSPATIAL International Conference. Seattle, Washington: ACM Press, 2009, p. 336. ISBN: 978-1-60558-649-6. DOI: 10.1145/1653771.1653818. URL: http://portal.acm.org/citation.cfm?doid=1653771.1653818 (visited on 06/30/2022).

[Ope23a]   OpenAI. *ChatGPT*. Version 3.5. chat.openai.com. 2023.

[Ope23b]   OpenStreetMap. Public GPS Trajectories. https://www.openstreetmap.org/traces. 2023.

[OQN03]   Washington Y Ochieng, Mohammed A Quddus, and Robert B Noland. "Map-matching in complex urban road networks". In: *Brazilian Journal of Cartography (Revista Brasileira de Cartografia)* 55.2 (2003), pp. 1–18.

[OR13]   Takayuki Osogami and Rudy Raymond. "Map matching with inverse reinforcement learning". In: *Twenty-Third International Joint Conference on Artificial Intelligence*. Citeseer. 2013.

[QNO06]   Mohammed A Quddus, Robert B Noland, and Washington Y Ochieng. "A high accuracy fuzzy logic based map matching algorithm for road transport". In: *Journal of Intelligent Transportation Systems* 10.3 (2006), pp. 103–115.

[QON07]   Mohammed A Quddus, Washington Y Ochieng, and Robert B Noland. "Current map-matching algorithms for transport applications: State-of-the art and future research directions". In: *Transportation research part c: Emerging technologies* 15.5 (2007), pp. 312–328.

[Qud+03]   Mohammed A Quddus, Washington Yotto Ochieng, Lin Zhao, and Robert B Noland. "A general map matching algorithm for transport telematics applications". In: *GPS solutions* 7 (2003), pp. 157–167.

[Ren+22]   Yazhou Ren, Jingyu Pu, Zhimeng Yang, Jie Xu, Guofeng Li, Xiaorong Pu, Philip S Yu, and Lifang He. "Deep clustering: A comprehensive survey". In: *arXiv preprint* (2022). arXiv: 2210.04142 [cs.LG].

[San+98]   Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. "Density-based clustering in spatial databases: The algorithm gdbscan and its applications". In: *Data mining and knowledge discovery* 2 (1998), pp. 169–194.

[SH22]   Siavash Saki and Tobias Hagen. "A practical guide to an open-source map-matching approach for big GPS data". In: *SN Computer Science* 3.5 (2022), p. 415.

[Tay+01]   George Taylor, Geoffrey Blewitt, Doerte Steup, Simon Corbett, and Adrijana Car. "Road reduction filtering for GPS-GIS navigation". In: *Transactions in GIS* 5.3 (2001), pp. 193–207.

[VQB09]   Nagendra R Velaga, Mohammed A Quddus, and Abigail L Bristow. "Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems". In: *Transportation Research Part C: Emerging Technologies* 17.6 (2009), pp. 672–683.

[VQB12]   Nagendra R Velaga, Mohammed A Quddus, and Abigail L Bristow. "Improving the performance of a topological map-matching algorithm through error detection and correction". In: *Journal of Intelligent Transportation Systems* 16.3 (2012), pp. 147–158.

[Wöl21]   Adrian Wöltche. "Evolving map matching with markov decision processes". In: (2021).

[WT16]   SUN Wenbin and XIONG Ting. "A low-sampling-rate trajectory matching algorithm in combination of history trajectory and reinforcement learning". In: *Acta Geodaetica et Cartographica Sinica* 45.11 (2016), p. 1328.

[YG18]   Can Yang and Gyozo Gidofalvi. "Fast map matching, an algorithm integrating hidden Markov model with precomputation". In: *International Journal of Geographical Information Science* 32.3 (2018), pp. 547–570. DOI: 10.1080/13658816.2017.1400548. eprint: https://doi.org/10.1080/13658816.2017.1400548. URL: https://doi.org/10.1080/13658816.2017.1400548.

[Yu+20]   Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. "Bdd100k: A diverse driving dataset for heterogeneous multitask learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2636–2645.

[Zad88]   Lotfi A Zadeh. "Fuzzy logic". In: *Computer* 21.4 (1988), pp. 83–93.

[ZG08]   Yongqiang Zhang and Yanyan Gao. "A fuzzy logic map matching algorithm". In: *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 3. IEEE. 2008, pp. 132–136.