The Mitsubishi A-Team: Map-Matching

Jeremy J. Lin, Tomoro Mochida, Riley C. W. O'Neill, Atsuro Yoshida
Academic Mentor: Dr. Shunsuke Kano
Industry Mentors: Dr. Masashi Yamazaki, Akinobu Sasada

July 14, 2023
g-RIPS

Table of Contents

Map Matching

Map matching is the process of determining the correct traveling route taken by a person or vehicle using road network data and trajectory data obtained from GPS or other positioning sensors.

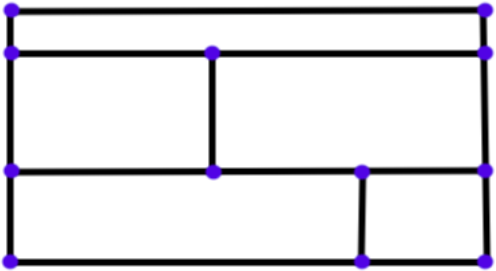
(a) Google Map

(b) Self-driving Car [1]

Road Network

Definition (Road Network)

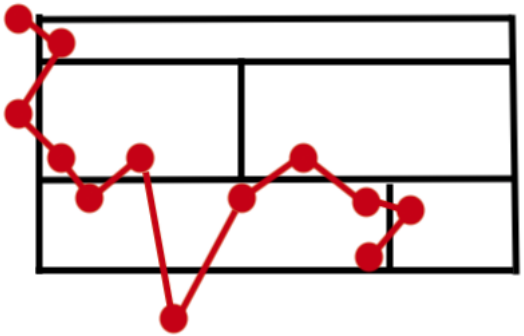
A road network (V, E) is a graph drawn on a plane.



GPS Trajectory

Definition (GPS Trajectory)

A GPS trajectory Tr is a timeseries of points in a plane.

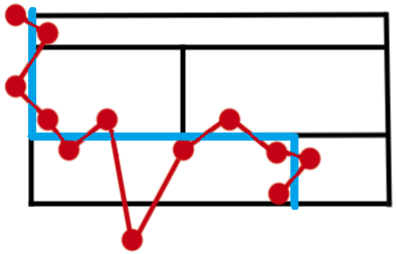


Map Matching Problem

Here is the main problem our research project addresses:

Problem (Map Matching Problem)

Given a road network (V, E) and a GPS trajectory Tr, match Tr to the actual route in the map.



Previous Research

There is a lot of previous research in map matching algorithms:

1. Topological Map Matching Algorithm
2. Map Matching Based on Hidden Markov Model
3. Dijkstra's Algorithm for Offline Map Matching

Datasets

State of the art trajectory registration algorithms deployed at the industrial level utilize millions of trajectories for training and inference, leveraging the device's sensors (IMU (accelerometer/gyroscope), speedometer, compass, etc) for improved results.

In the absence of such an expansive dataset, we are left to scour the internet for data:

Dataset	Raw GPS Timeseries	IMU data	Velocity	Elevation	Ground Truth
KCMMN [2]	V	X	X	X	V
BDD100K [3]	V	V	V	X	X
OpenStreetMaps [4]	V	X	V	V	X
EnviroCar [5]	V	X	V	X	X

However, none of these have IMU, velocity, *AND* ground truth trajectories.

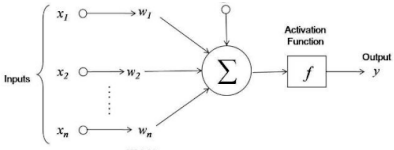
Data Fusion and Processing

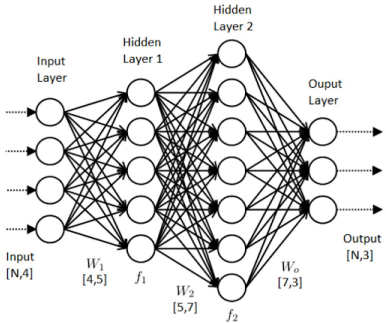
- In the absence of a dataset with speed, direction, IMU data, *AND* ground truth trajectories, we propose to train a *Graph Neural Network (GNN)* on the BDD100K dataset to approximate IMU/velocity data from the raw trajectories for use on the KCMMN dataset.
- BDD100K: 5GB compressed, 47GB as uncompressed JSON's, 3GB after removing trajectories with missing GPS or gyroscopic data (2,000 of 520,000 trajectories) and resaving as CSV's.
- Data processed at Minnesota Super-computing Institute due to storage overhead.

What is a graph neural network?

Table of Contents

What is a Neural Network?





"Universal approximator" function with layers of the form:

$$x^k = f(W^k x^{k-1} + b^k)$$

$f : \mathbb{R}^{d_k} \to \mathbb{R}^{d_k}$ nonlinear function; $W^k \in \mathbb{R}^{d_k \times d_{k-1}}$, $b^k \in \mathbb{R}^{d_k}$ learnable
Trained to minimize some loss function over some dataset via backpropagation.

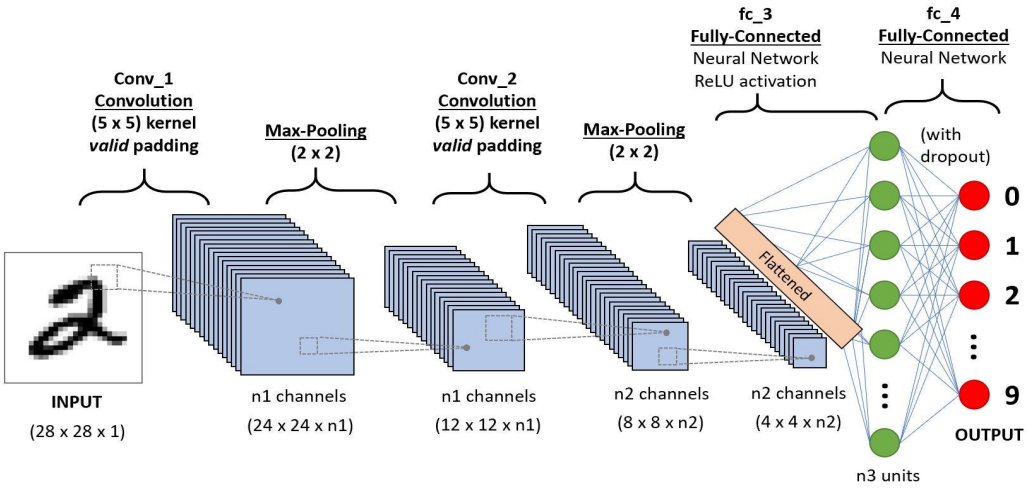
Convolutional Neural Networks (CNNs): learn a local filter of the form:

$$x^{k+1} = b^{k+1} + \sum_{y^k \in N(x^k)} F^{k+1}(x^k - y^k) y^k$$

where F is compactly supported. For images and grid like data, F is just a matrix; with pooling operations, this is incredibly powerful. Note this is a numerical approximation to the integral

$$(F * I)(x) = \int F(x - y) I(y) dy.$$

Convolutional Neural Networks



Graph (Convolutional) Neural Networks

- Graph - points + pairwise connectivity, i.e. trajectories and roads. Want a network structure that can leverage the connectivity and extract spatial-temporal features.
- PyTorch Geometric (PyG) has over 100 different forms of graph convolutional neural networks implemented, each of which leverage different properties of the graph. Of those in PyG, we identified 3 network layer structures of interest:
  1. NNConv: $x_i^{k+1} = W^k x_i^k + \sum_{j \in N(i)} F_k(x_j^0 - x_i^0) x_j^k$, where $F_k : \mathbb{R}^{d_0} \to \mathbb{R}^{d_k \times d_{k+1}}$ is a neural network.
  2. FeaStConv: $x_i^{k+1} = \frac{1}{|N(i)|} \sum_{j \in N(i)} \sum_{h=1}^{H} q_h(x_i^k - x_j^k) W_h^k x_j^k$, where $q_h(z) = \text{Softmax}_h(u_h^T z + c_h)$ is an attention head.
  3. GMMConv: $x_i^{k+1} = \frac{1}{|N(i)|} \sum_{j \in N(i)} \sum_{h=1}^{H} g_h(x_i^k - x_j^k) W_h^k x_j^k$, where g_k is a learnable Gaussian function.
- We postulate that a naturally translation invariant network would be adventitious, at least for the first layer, e.g.:

$$x_i^{k+1} = b^k + \frac{1}{|N(i)|} \sum_{j \in N(i)} F^k(x_j^0 - x_i^0) W^k [x_j^0 - x_i^0]$$

but nothing of this form appears to be implemented yet in PyG - implementing such a layer is of tremendous interest.

GNN for Feature Extraction / Modeling

1. Modeling IMU/velocity data:
   - 3 filter layers of NNConv GNN network in PyG. Each filter consists of a 3 layer neural network.
   - Graph construction: 11-KNN (time) neighbors graph (this is identical to a 1D convolution, just for proof of concept).
   - If successful, this will give robust approximations of the acceleration 3-vector, gyroscopic 3-vector, speed, and direction of travel.
2. Unsupervised Graph contrastive learning: (if (1.) is successful)
   - leverage the large size of BDD100K, KCMMN, and OpenStreetMaps data to learn more robust features for registration.
   - Implement SimCLR contrastive learning framework [6] on graphs - next slide.

Graph Learning Frameworks: Results

- Implemented 3 layer NNConv GNN for modeling IMU data, each with 3-layer filter on input graph.
- Result: not good. Possible vanishing gradient problem.
- Additionally, PyG DataParallel is not optimized for distributed models - actually slower to use more GPU's.
- Hence, graph neural networks were not examined further for this study.

What to do instead?

1D CNN's for Modeling IMU Data

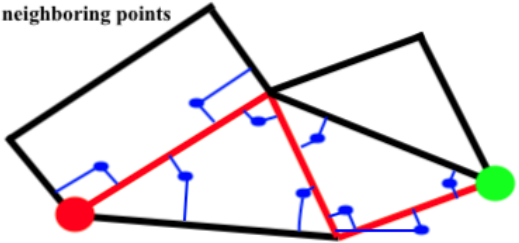
- Learn local filter to estimate IMU data.
- Result: not good either... possible vanishing gradients?
- BDD100K data is "questionable" - does not specify CRS for GPS points or units of acceleration/speed. Requests for clarification have gone unanswered.
- Conclusion: abandoned modeling IMU data to focus on training edge affinity function.

GNNs for Learnable Edge Affinity Function

Suitable for Dijkstra's algorithm, Fuzzy inference, etc.
Define $S(e) = \{i : e \in KNN(p_i)\}$, $D(p, e) = \text{argmin}_{q \in e} |q - p|$, $|D|(p, e) = \min_{q \in e} |q - p|$.
Train $f(e) = f(V[S(e)], [F[S(e)]; |D|(V[S(e)]; D(V[S(e)], e)], e)$ as a neural network.

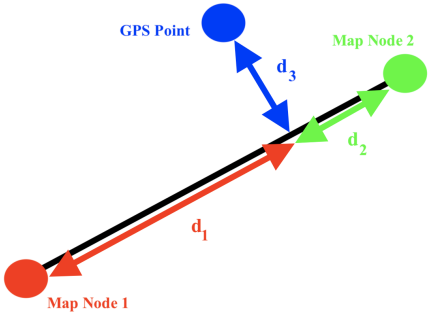
1. **Find K-Nearest neighboring edges to each GPS point (K-D Tree, here K=2)**
2. **Assemble edge to GPS point adjacency from this**
3. **Compute/index relevant features from neighboring points**
4. **Train edge weighting function:**
   - **Maximize for ground truth roads**
   - **Minimize for all other roads (if no neighboring GPS points, assume the weight is 0)**

(inverted wrt Dijkstra's for numerical stability)



Edge Affinity Model Architecture

1. Point-wise feature encoder: 3 layer neural net. Input: signed distance to edge, projection's signed distance to each node.
2. Feature Aggregation (pooling): to be discussed
3. Decoder (classifier): 2 layer neural net



Feature Aggregation for Machine Learning (aka Pooling)

As the set of points corresponding to each edge under the inverse KNN adjacency vary in cardinality, we need some means of aggregating the point-wise features into a single vector of fixed size for machine learning.
We try the following as an initial implementation:

Summation: $\sum_{i=1}^{n} f(x_i)$ - intuitively, if an edge has more nearby trajectory points, it's more likely in the ground truth trajectory.

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

Results of Summation/Average pooling

Not good. Model tends to vote all trajectory, all non-trajectory, or 50-50 split.

New Pooling Operations for Point-Based Networks

We Introduce a novel new pooling operation for data of varying cardinality (e.g. pointclouds): Extract k features with maximal l^1 norm and k features with minimal l^1 norm:

$$I_{top} = \text{Kargmax}_i \|x[i]\|_1; \quad I_{bot} = \text{Kargmin}_i \|x[i]\|_1$$

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

Set $m = 2k$. Extract:
Dot products: $A_i j = V_i \cdot V_j : i < j$ ($\frac{m^2-m}{2}$ values)
l^1 Distances: $D_{ij} = \|V_i - V_j\|_1$ ($\frac{m^2-m}{2}$ values)
l^1 magnitudes: $M_i = \|V_i\|_1$ (m values) Vectors: V ($m * n$ (feat dim) values)
Total: $4k^2 + 2kn$ values output.
Q: What to do if there's not k points in the edge set?

Padding

If there are fewer than k points at an edge... Let's repeat the maximum/minimum value for the respective max/min index set.
This introduces 0's into the extracted distances, i.e. we tell the model not many points are there, 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, but in different places. Is this a bad thing? Unclear.

Loss Formulation

Define the y-switch operator for ease of notation: $(x)_y = \begin{cases} x & : y = 1 \\ 1 - x & : y = 0 \end{cases}$

Binary Cross Entropy: $BCE(x, y) = -y \log(x_y) - (1 - y) \log(1 - x_y)$

Reweighted BCE: $\alpha_y BCE(f(x), y)$ for $\alpha \in (0, 1)$

Focal loss: $FL(x, y) = -\alpha_y (1 - x_y)^\gamma \log(x_y)$

Results & Discussion

High training accuracy (95% overall, 84% of trajectory edges correctly identified)

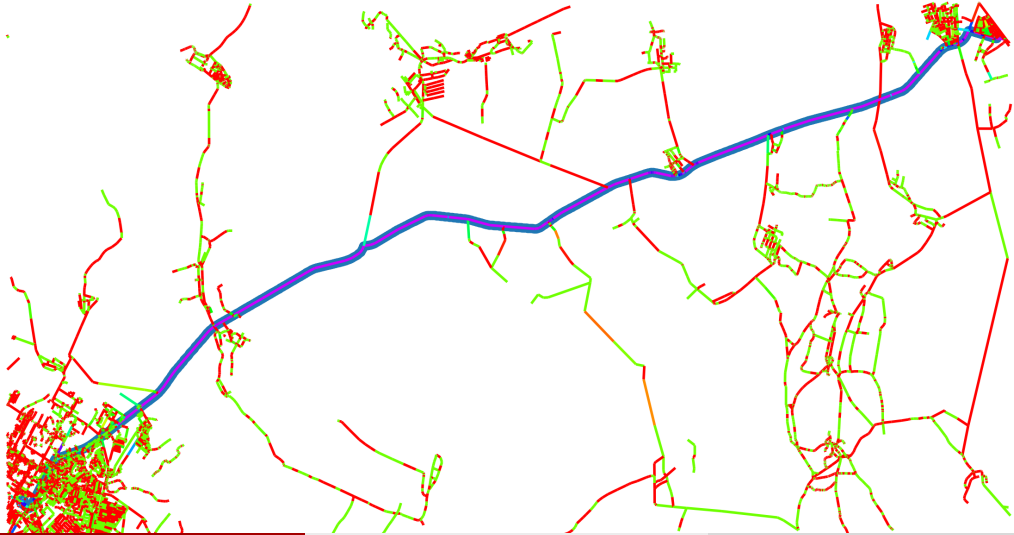
Low positive testing accuracy (82% overall, 10% of trajectory edges identified.)

Q: Is this overfitting? Or is it differences in training/testing distributions? Or both?

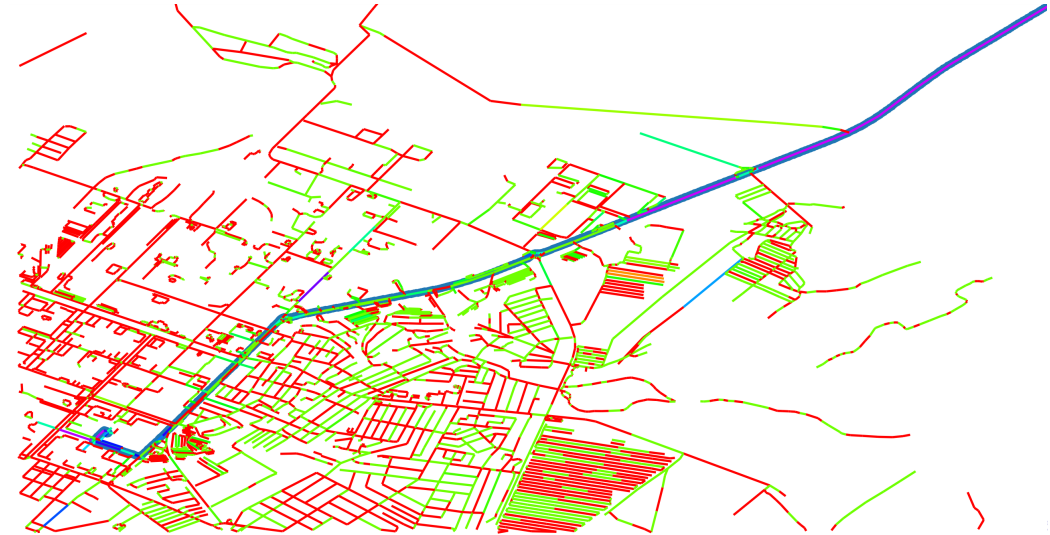
$\sim$ The train/test split was done by trajectory... quite possible that these vary in distribution.
$\sim$ Overfitting - more work needs to be done to determine where excess in model is.
$\sim$ The pooling operation results in a very large output dimension - how to best mitigate this?

Attempts to mitigate overfitting so far have resulted in lower training AND testing accuracy... interesting.

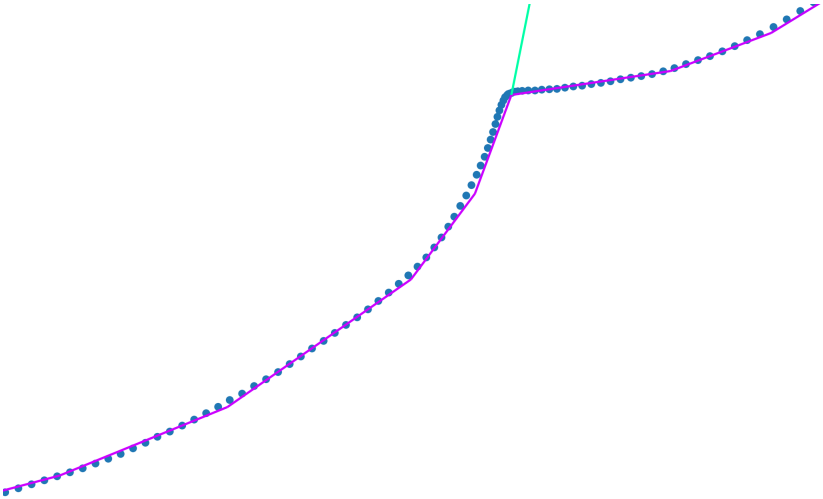
Training Results



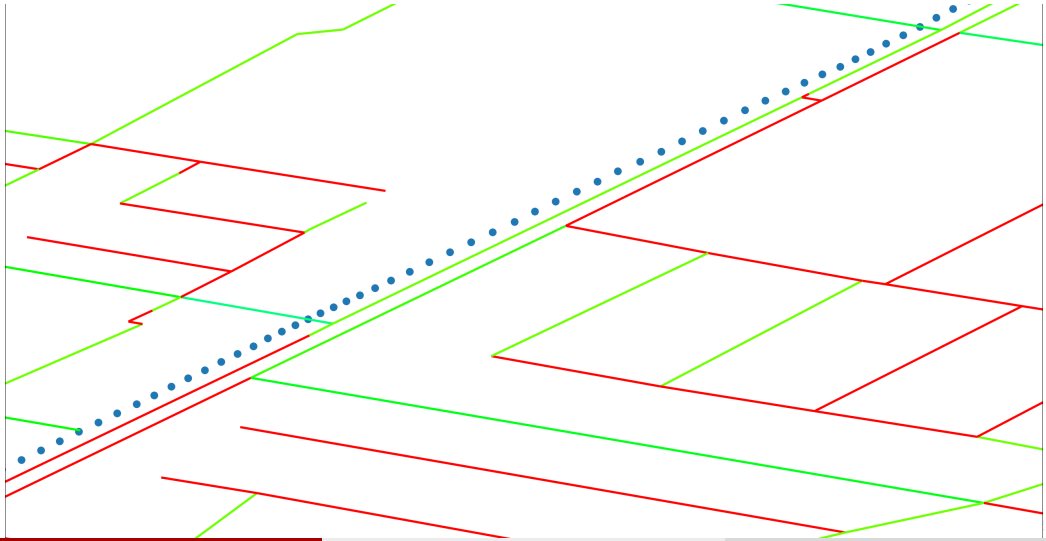
Training Results: Enlarged to Show Texture



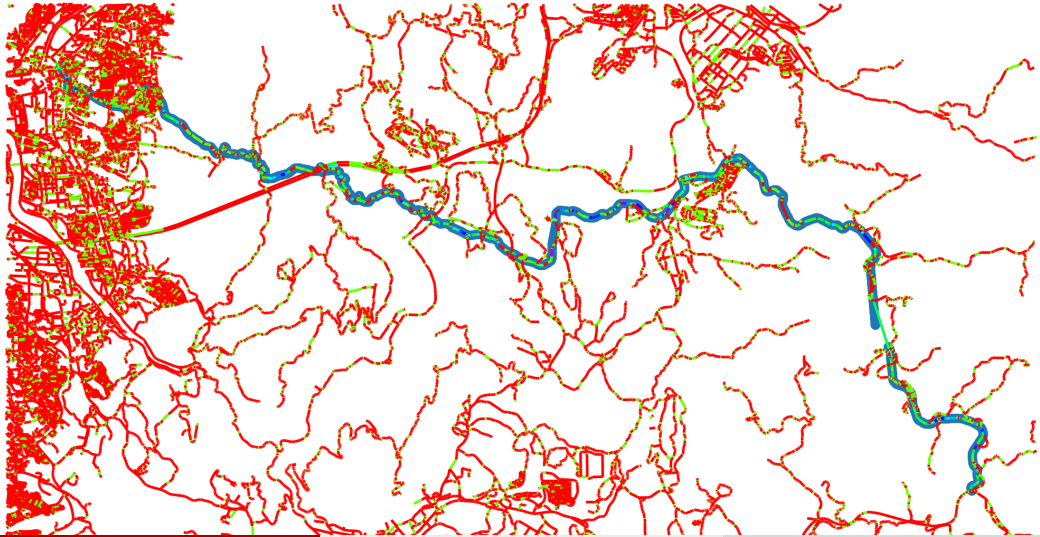
Training Results: The Good



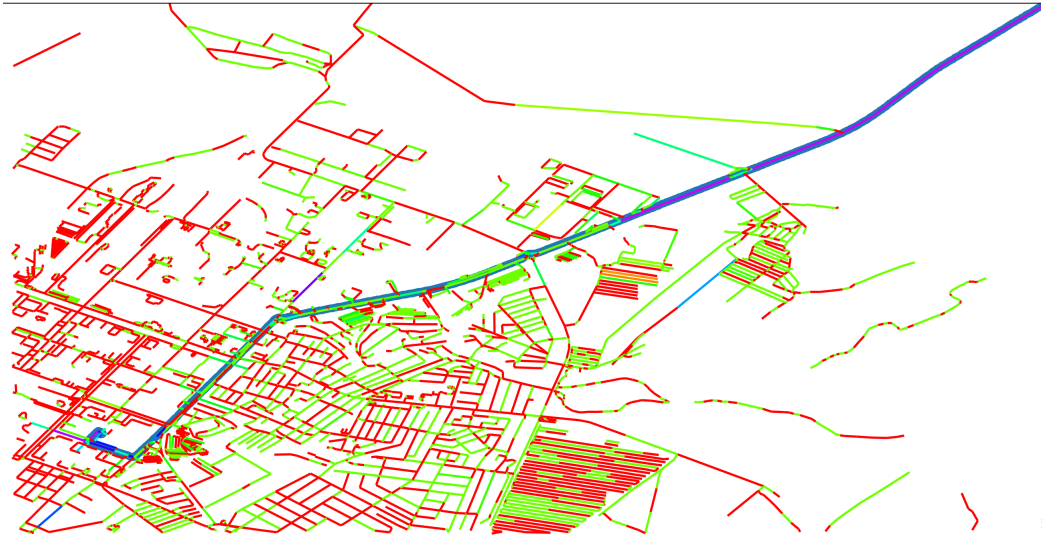
Training Results: The Bad



Testing Results



Testing Results



Takeaways & Directions for Future Work

The model has potential... but can it outperform hand-crafted feature extraction?

Possible Model / Training improvements:

1. Modifications to training: learning rate decay, adaptive weighting, longer training time.
2. Data is very clean - add obvious measures to mitigate common-sense mistakes?
3. Input feature engineering: absolute values instead of signed distances? When/what to normalize?
4. Pool sooner? Too many layers for an easy problem?

Data Improvements:

1. GET MORE DATA - reduce susceptibility to differences in projection
2. GET DATA WITH IMU/VELOCITY DATA

Also: apply Dijkstra's algorithm to results for evaluation: quite possible it works even for testing da

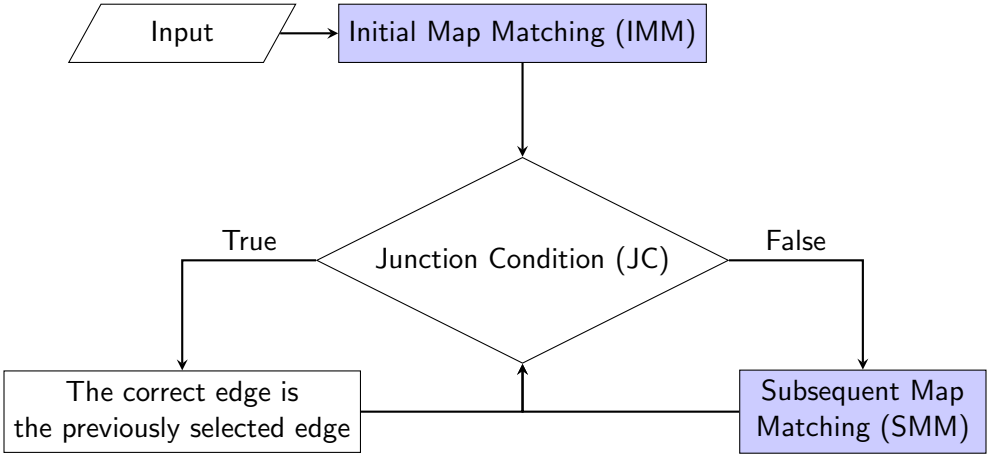
Table of Contents

What is AHP?

The Analytic Hierarchy Process (AHP) is a method of decision making that combines mathematical analysis with human judgment. It uses hierarchical classification to handle complex or abstract information.

AHP has not been widely applied to map matching, so our algorithm is a relatively new method.

Flowchart of AHP Map Matching Algorithm



Initial Map Matching (IMM)

The purpose of initial map matching is to specify the correct edge that matches the first trajectory point p_0.
The initial map matching uses:

- Distance data, which is the data of the distance between the point p_0 and each candidate edge.
- Direction data, which is the data of the angle difference between the heading of the first point p_0 and the direction of each candidate edge.

Using AHP, we assign a weight to each candidate edge and finally select the highest weight edge as the correct edge for p_0.

Junction Condition (JC)

After IMM, SMM, and also JC itself, we always check the junction condition (JC) for the next trajectory point, say p_t. The junction condition verifies whether the point still matches the previously selected edge. It is based on

- How far the point is from the next junction point.
- How small the angle difference between the direction of p_t and that of p_{t-1} is.

If the point is considered to have not yet crossed the junction, the previously selected edge is taken as the correct edge for that point. Otherwise, we proceed to the subsequent map matching.

Subsequent Map Matching (SMM)

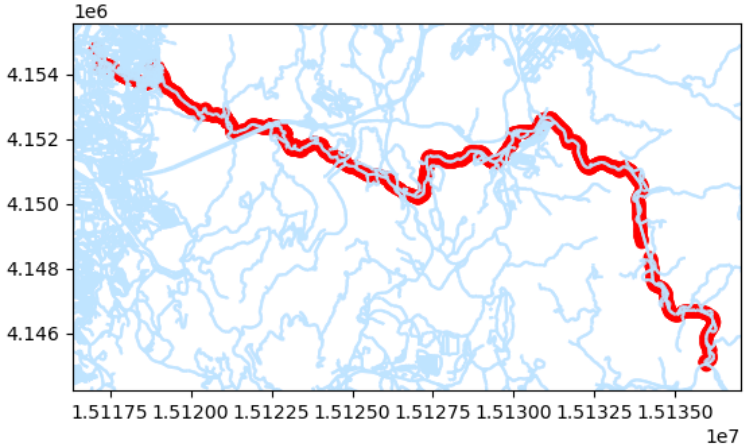
The subsequent map matching (SMM) is performed on a point that fails to meet the junction condition. It detects the matching of that point based on

- Distance data (same as IMM).
- Direction data (same as IMM).
- Turn restriction data, which reflects if the vehicle on the previously selected edge can legally turn onto each candidate edge.

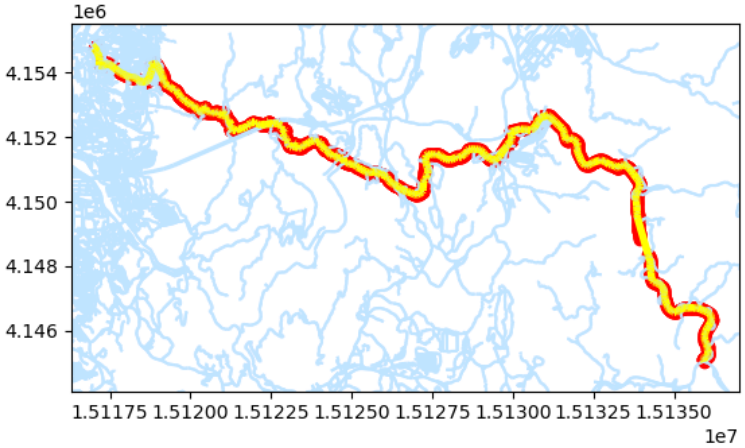
Using these data we choose the correct edge for the point.

We go back to the junction condition and repeat this process until we reach the last trajectory point.

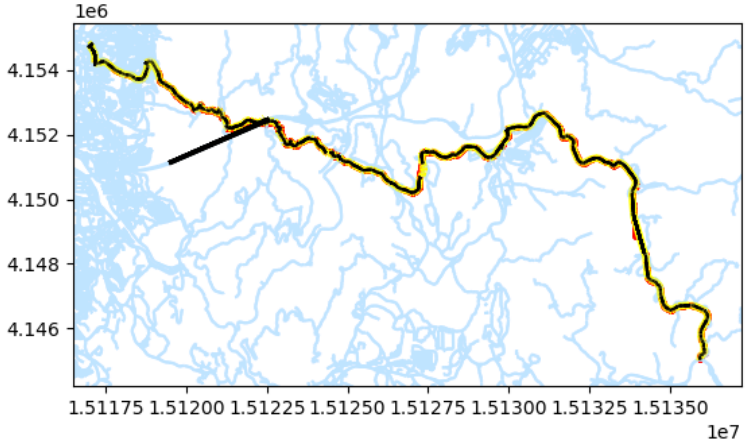
Implementation Result



Implementation Result

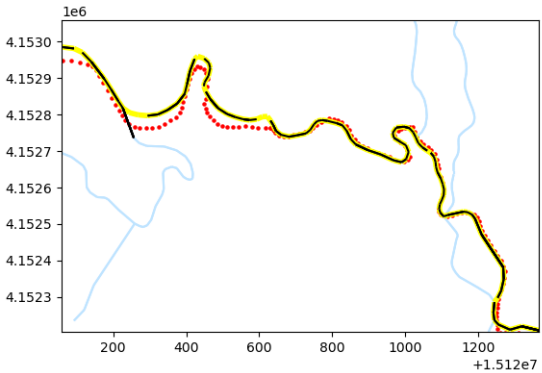


Implementation Result



Implementation Result: Example of Mismatching

Some branches



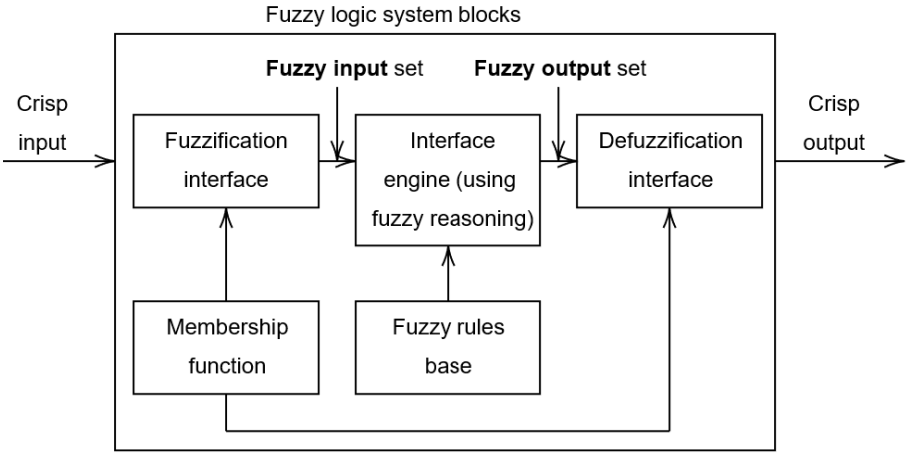
Some jumps

Summary

- Advantages
  1. Simple and easy to understand.
  2. Relatively fast.
  3. Considerably good accuracy.
- Disadvantages
  1. Sensitive to measurement errors.
  2. Large variation in accuracy depending on trajectory data.
- Future work
  1. Postprocessing for detecting and fixing mismatching.

Table of Contents

Fuzzy Inference System



Fuzzy Logic Map Matching algorithm

- Fuzzy logic Map Matching (FLMM) algorithm consists of two main step :
  - Finding the correct link the vehicle is traveling on which can be divided. This step can further be divided into two sub-step:
    - Finding initial correct link (IMP step)
    - Finding a correct link when the vehicle crosses an intersection (SMP 2)
  - tracking vehicle around the link and detecting whether the object crossed the intersection (SMP 1). [7]

	IMP	SMP 1	SMP 2
Number of Input	4	6	6
Number of Rule	6	12	10

Initial Map Matching Process

- Input :
  1. Speed of the vehicle, v (m/s)
  2. Horizontal dilution of Precission (HDOP)
  3. Perpendicular Distance, PD (m)
  4. Heading error, $HE = |\theta - \theta'|$
- Algorithm:
  1. Find all candidate links inside the error bound
  2. select edges with the highest FIS 1 output
  3. repeat until an edges is selected consecutively

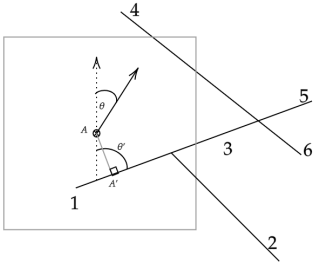


Figure: IMP implementation

SMP 1

- Additional input :
  1. Speed of the vehicle, v (m/s)
  2. Perpendicular Distance, PD (m)
  3. $\Delta d = d - d_2$
  4. Heading Increment, $HI = |\theta' - \theta|$
  5. α and β.
- SMP 1 step :
  1. Calculate FIS 2 score from current trajectory point and the previous matched point and edges.
  2. If FIS 2 is greater than a certain cutoff then conclude that the current edges still in the trajectory, and calculate the projection point.
  3. Else, indicate that current trajectory is entering a junction, and begin SMP 2 step.

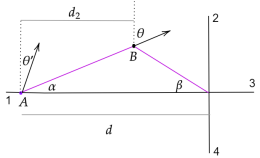
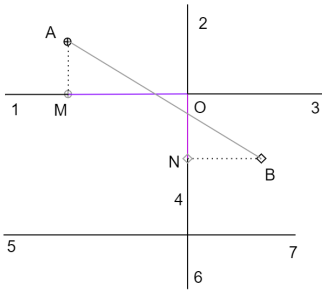


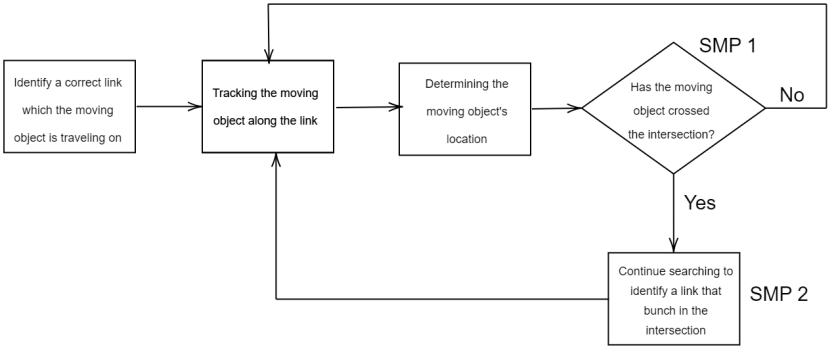
Figure: FIS2 diagram

SMP 2

- Input
  1. Speed of the vehicle, v (m/s)
  2. Horizontal dilution of Precission (HDOP)
  3. Perpendicular Distance, PD (m)
  4. Heading error, $HE = |\theta - \theta'|$
  5. Link Connectivity
  6. Distance error, difference between distance travelled from last position(A) and shortest path from the last matched position(M) to the current position (B) around the links (e.g $\epsilon = AB - (MO + ON)$)
- SMP 2 step :
  1. Find all candidate link inside the error bound
  2. Select the edge with the highest FIS 3 output



Map Matching Flow Chart



FLMM Result

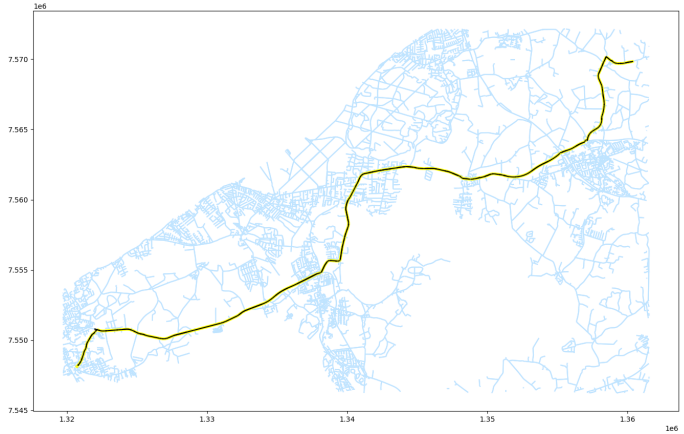


Figure: map matching process in KCMMN data set

FLMM Results

- Advantage
  1. Intuitive and easy to add or remove input variable
  2. Less sensitive to input errors
  3. Relatively good result and less sensitive to measurement errors
- Disadvantage
  1. Algorithm is slower than other method we tried.
  2. Rules and parameter value in our implementations are too simple
- Future Works
  1. Incorporate data driven algorithm to tune FLMM parameter (see Figure)
  2. Adding new rules to the system (e.g :deciding whether map is urban versus suburban)

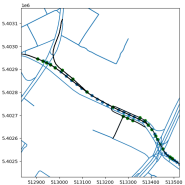


Figure: Before

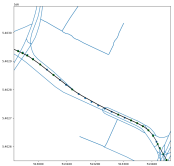
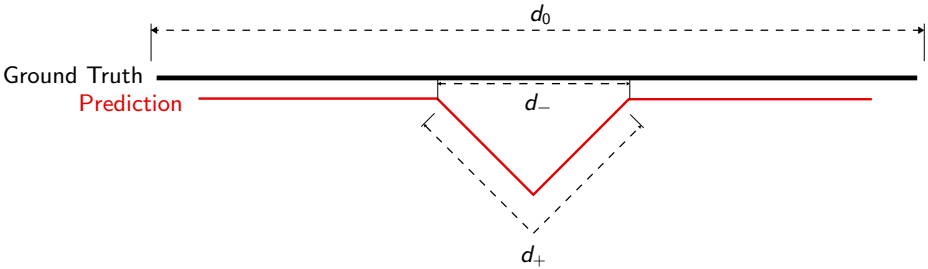


Figure: After

Table of Contents

Evaluation



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

d_0 = length of ground truth

d_- = length of prediction route erroneously subtracted

d_+ = length of prediction route erroneously added

Figure: Error Formula by Newson and Krumm [8]

Results

- AHP and Fuzzy incorporated other variable such as speed and bearing direction, which are not available in the test dataset.
- Parameter tuning needs to be done to achieve better performance.

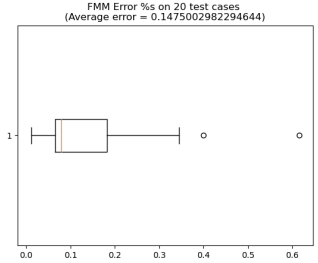


Figure: FMM Average Error 14.7%

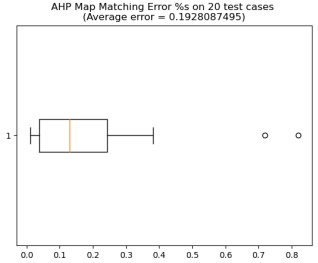


Figure: AHP Average Error 19.2%

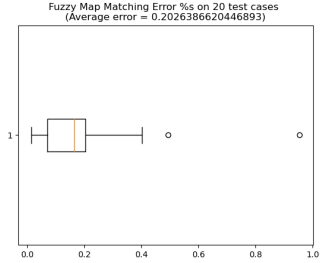


Figure: Fuzzy Average Error 20.2%

Table of Contents

Preprocessing

Due to time constraints, we could not conduct experiment in preprocessing of mitigating stay points and eliminating outliers, though we have already implemented it.
We expect conducting preprocessing reduces the error rate and computation time.

The DBSCAN Clustering Algorithm

"Density-Based Spatial Clustering of Applications with Noise" (DBSCAN) is a clustering algorithm based on metric information. This algorithm classifies the points into the 3 categories:

1. Core points;
2. Reachable points;
3. Unreachable points.

Mitigating Stay Points by DBSCAN

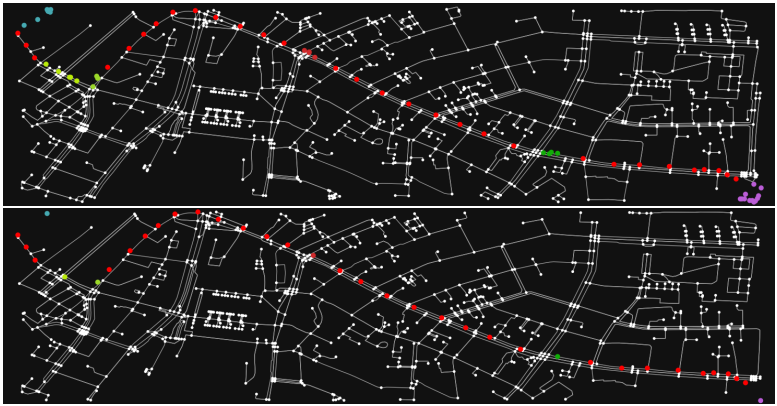


Figure: Before averaging each cluster. The red points are the unreachable points detected by DBSCAN, and each non-red color represents one cluster.

Figure: After averaging each cluster.

Eliminating Outliers by DBSCAN

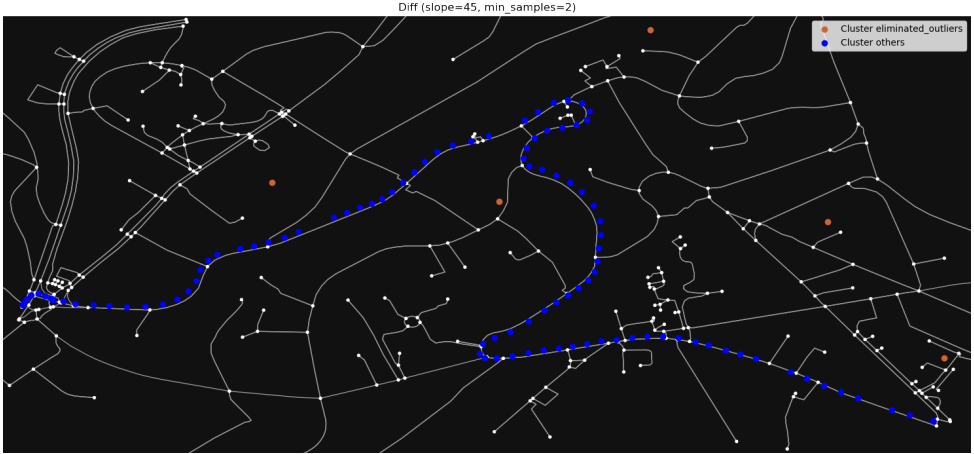


Figure: The orange points were classified as unreachable points by DBSCAN

Postprocessing

As we have seen, the routes the fuzzy logic MM and AHP find may have a lot of "jumps" and undesired "branches".
In order to deal with these problems, it may be promising to conduct postprocessing such as:

1. Interpolating the jumps;
2. Cutting off the branches.

Interpolating Jumps

We propose filling the jumps simply by the shortest paths:

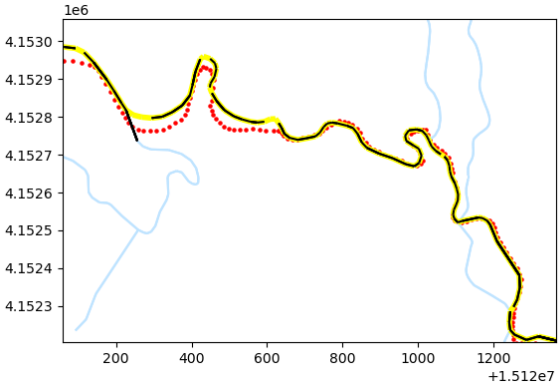


Figure: Jumps

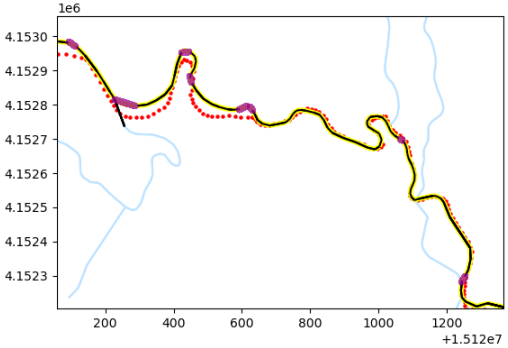


Figure: Filling Jump by Shorted Paths

Cutting off Branches

We propose a procedure to cut off the branches as follows:

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

Cutting off Branches

Cutting off Branches

1. Find the shortest path from the start point to the end point.

Cutting off Branches

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.

Table of Contents

Acknowledgements

We would like to extend our sincere gratitude and thanks to Mitsubishi, Tohoku University, the AIMR, the UCLA IPAM, Suito Sensei & G-RIPS staff, and the G-RIPS 2022 Mitsubishi A team for making this work possible.

We would also like to thank our academic and industrial mentors: Dr. Shunsuke Kano, Dr. Masashi Yamazaki, and Akinobu Sasada.

References I

[1] [Online]. Available: https://hmc.unist.ac.kr/research/autonomous-driving/.

[2] M. Kubička, A. Cela, P. Moulin, H. Mounier, and S.-I. Niculescu, "Dataset for testing and training of map-matching algorithms," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2015, pp. 1088–1093.

[3] F. Yu, H. Chen, X. Wang, *et al.*, "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.

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

[5] EnviroCar, *Envirocar data*, https://envirocar.org/analysis.html?lng=en, 2023.

References II

[6] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.

[7] M. A. Quddus, R. B. Noland, and W. Y. Ochieng, "A high accuracy fuzzy logic based map matching algorithm for road transport," *Journal of Intelligent Transportation Systems*, vol. 10, no. 3, pp. 103–115, 2006.

[8] P. Newson and J. 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*, Seattle, Washington: ACM Press, 2009, p. 336, ISBN: 978-1-60558-649-6. DOI: 10.1145/1653771.1653818. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1653771.1653818 (visited on 06/30/2022).

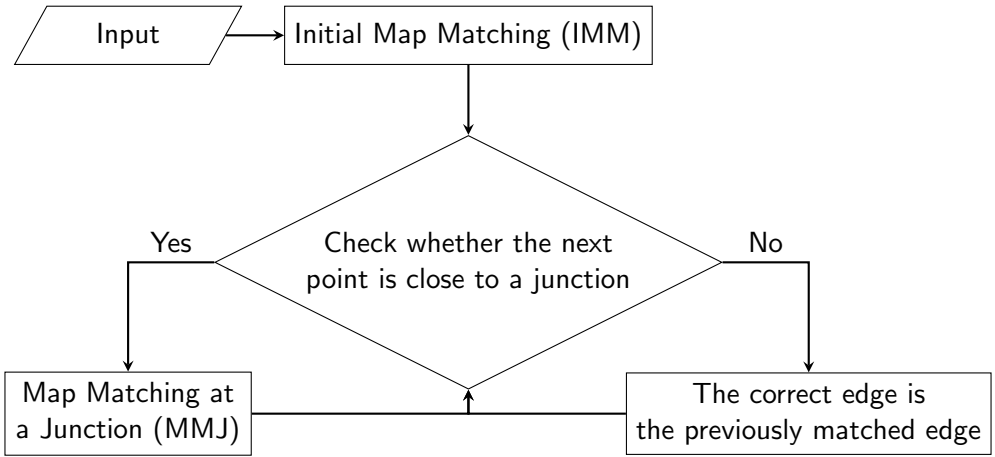
Thank You!

Questions?

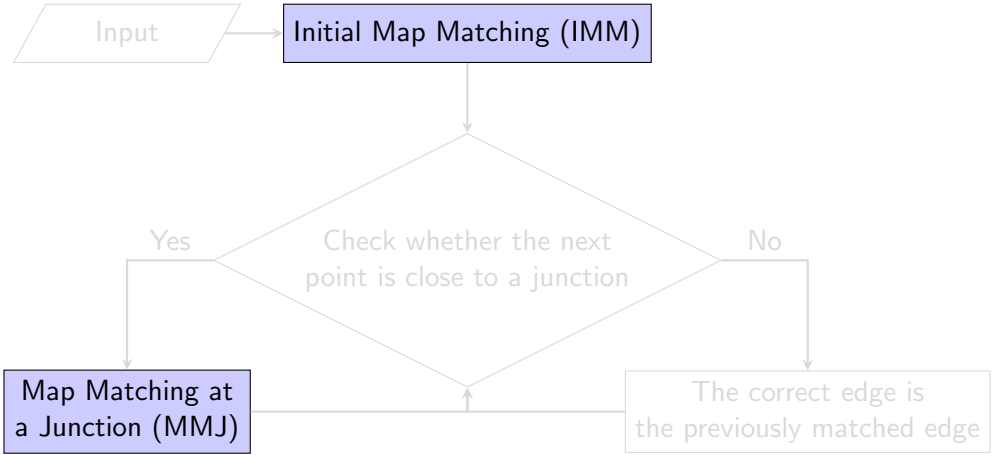
Table of Contents

Miscellaneous Page

Flow of our algorithm



Flow of our algorithm



Initial Map Matching

The purpose of Initial Map Matching is to specify the correct edge that matches the first trajectory point p_0.

Initial Map Matching takes the following steps:

1. Identify candidate edges. They are the edges that have the possibility of matching p_0.
2. Assign a weight to each candidate edge. Weights are computed by using two factors: distance and direction.
3. Take the highest weight edge as the correct edge for p_0.

Initial Map Matching

The purpose of Initial Map Matching is to specify the correct edge that matches the first trajectory point p_0.

Initial Map Matching takes the following steps:

1. Identify candidate edges. They are the edges that have the possibility of matching p_0.
2. Assign a weight to each candidate edge. Weights are computed by using two factors: distance and direction.
3. Take the highest weight edge as the correct edge for p_0.

Initial Map Matching

The purpose of Initial Map Matching is to specify the correct edge that matches the first trajectory point p_0.

Initial Map Matching takes the following steps:

1. Identify candidate edges. They are the edges that have the possibility of matching p_0.
2. Assign a weight to each candidate edge. Weights are computed by using two factors: distance and direction.
3. Take the highest weight edge as the correct edge for p_0.

Initial Map Matching

The purpose of Initial Map Matching is to specify the correct edge that matches the first trajectory point p_0.

Initial Map Matching takes the following steps:

1. Identify candidate edges. They are the edges that have the possibility of matching p_0.
2. Assign a weight to each candidate edge. Weights are computed by using two factors: distance and direction.
3. Take the highest weight edge as the correct edge for p_0.

Initial Map Matching: Step 2

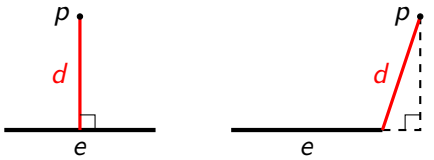
Let $e_1, \ldots, e_n$ be candidate edges obtained in Step 1.
First, we define the pairwise comparison matrix $A = [\alpha_{ij}]$ for distance using hierarchical classification as follows:

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

Here,

$$d_i := \mathrm{dist}(p_0, e_i) = \inf_{x \in e_i} \mathrm{d}(p_0, x),$$

$$d_j := \mathrm{dist}(p_0, e_j).$$



Initial Map Matching: Step 2

Based on the matrix A, we compute weights $w_1^{\mathrm{dist}}, \ldots, w_n^{\mathrm{dist}}$ for distance.

	A			geometric mean	weight
e_1	α_{11}	$\cdots$	α_{1n}	$\sqrt[n]{\alpha_{11} \cdots \alpha_{1n}} =: g_1$	$w_1^{\mathrm{dist}} := g_1/S$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
e_n	α_{n1}	$\cdots$	α_{nn}	$\sqrt[n]{\alpha_{n1} \cdots \alpha_{nn}} =: g_n$	$w_n^{\mathrm{dist}} := g_n/S$

$$S := \sum_{i=1}^{n} g_i$$

The same procedure is performed for the direction data and then we get weights $w_1^{\mathrm{dir}}, \ldots, w_n^{\mathrm{dir}}$ for direction.

Initial Map Matching: Step 2

Based on the matrix A, we compute weights $w_1^{\mathrm{dist}}, \ldots, w_n^{\mathrm{dist}}$ for distance.

	A			geometric mean	weight
e_1	α_{11}	$\cdots$	α_{1n}	$\sqrt[n]{\alpha_{11} \cdots \alpha_{1n}} =: g_1$	$w_1^{\mathrm{dist}} := g_1/S$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
e_n	α_{n1}	$\cdots$	α_{nn}	$\sqrt[n]{\alpha_{n1} \cdots \alpha_{nn}} =: g_n$	$w_n^{\mathrm{dist}} := g_n/S$

$$S := \sum_{i=1}^{n} g_i$$

The same procedure is performed for the direction data and then we get weights $w_1^{\mathrm{dir}}, \ldots, w_n^{\mathrm{dir}}$ for direction.

Initial Map Matching: Step 2

Based on the matrix A, we compute weights $w_1^{\mathrm{dist}}, \ldots, w_n^{\mathrm{dist}}$ for distance.

	A			geometric mean	weight
e_1	α_{11}	$\cdots$	α_{1n}	$\sqrt[n]{\alpha_{11} \cdots \alpha_{1n}} =: g_1$	$w_1^{\mathrm{dist}} := g_1/S$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
e_n	α_{n1}	$\cdots$	α_{nn}	$\sqrt[n]{\alpha_{n1} \cdots \alpha_{nn}} =: g_n$	$w_n^{\mathrm{dist}} := g_n/S$

$$S := \sum_{i=1}^{n} g_i$$

The same procedure is performed for the direction data and then we get weights $w_1^{\mathrm{dir}}, \ldots, w_n^{\mathrm{dir}}$ for direction.

Initial Map Matching: Step 2

Based on the matrix A, we compute weights $w_1^{\mathrm{dist}}, \ldots, w_n^{\mathrm{dist}}$ for distance.

	A			geometric mean	weight
e_1	α_{11}	$\cdots$	α_{1n}	$\sqrt[n]{\alpha_{11} \cdots \alpha_{1n}} =: g_1$	$w_1^{\mathrm{dist}} := g_1/S$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
e_n	α_{n1}	$\cdots$	α_{nn}	$\sqrt[n]{\alpha_{n1} \cdots \alpha_{nn}} =: g_n$	$w_n^{\mathrm{dist}} := g_n/S$

$$S := \sum_{i=1}^{n} g_i$$

The same procedure is performed for the direction data and then we get weights $w_1^{\mathrm{dir}}, \ldots, w_n^{\mathrm{dir}}$ for direction.

Initial Map Matching: Step 2

Finally we define the total weight $\mathrm{TW}(e_i)$ for each candidate edge e_i by

$$\mathrm{TW}(e^i) := c^{\mathrm{dist}} w_i^{\mathrm{dist}} + c^{\mathrm{dir}} w_i^{\mathrm{dir}},$$

where c^{dist} and c^{dir} are coefficients reflecting the relative importance of distance and direction, given by

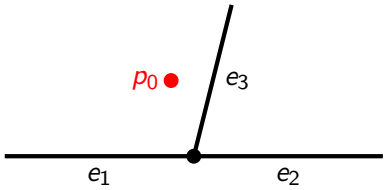
	urban	suburban	rural
c^{dist}	0.0806	0.438	0.556
c^{dir}	0.372	0.464	0.434

The highest edge is taken as the correct edge that matches p_0.

Exmple of AHP-based map matching

Example

$$A = \begin{bmatrix} 1 & 2 & 0.25 \\ 0.5 & 1 & 0.2 \\ 4 & 5 & 1 \end{bmatrix}.$$



Say,

$$d_1 = \text{dist}(p_0, e_1) = 10,$$
$$d_2 = \text{dist}(p_0, e_2) = 12,$$
$$d_3 = \text{dist}(p_0, e_3) = 3.$$

IMP Algorithm

Algorithm IMP Algorithm

```
Count = 0
if Count ≤ 3 then
  repeat
    Find candidate edges
    Perform FIS-1 for all candidate edge
    Pick candidate edge with highest FIS-1 output
    if previous selected edge = current selected edge then
      Count = Count + 1
    end if
  until Count = 3
end if
Proceed with SMP 1
```

SMP-1 Algorithm

Algorithm SMP-1 Algorithm

```
Stay = TRUE
if Stay = TRUE then
  repeat
    output = FIS-2(current trajectory, previous edge)
    if output ≥ 60 then
      current edge = previous edge
    else
      Stay = FALSE
    end if
  until Stay = FALSE
end if
Proceed with SMP 2
```

SMP-2 Algorithm

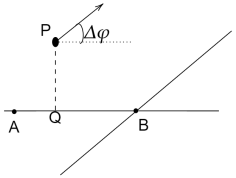
Algorithm SMP-2 Algorithm

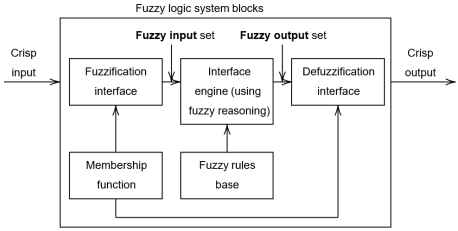
Find candidate edges
Perform FIS-3 for all candidate edge
Pick candidate edge with highest FIS-3 output
Proceed with SMP 1

FIS Example

- Denote PQ and $\Delta\varphi$ are the two primary input and Z as an output that measure the likelihood of P matched to Q
- PQ $= 18$ $\Delta\varphi = 12$
- FIS have these following rules:

1. R_1: If PQ is short and $\Delta\varphi$ is small then the possibility of matching P on link AB (Z) is high.
2. R_2: If PQ is long and $\Delta\varphi$ is large then the possibility of matching P on link AB (Z) is low.





Rule 1

Calculating the strength of Rule 1 (ω_1)

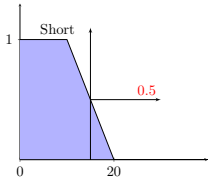


Figure: Membership Function of PQ

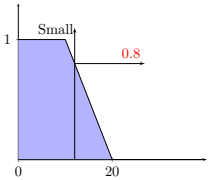
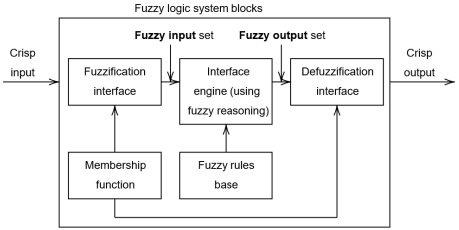


Figure: Membership function of $\Delta\varphi$

$$\omega_1 = \min(0.8, 0.5) = 0.5$$



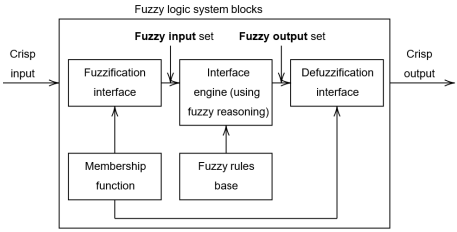
Defuzzification Stage

Takagi-Sugeno-Kang Fuzzy Inference System :

$$Output = \frac{\omega_1}{\omega_1 + \omega_2} Z_1 + \frac{\omega_2}{\omega_1 + \omega_2} Z_2$$

Where : Z_1 and Z_2 is an output of function based on the rule. e.g:

$$Z = \begin{cases} 50, & \text{if possibility of matching is high} \\ 10, & \text{if possibility of matching is low} \end{cases}$$



FIS 1

- Input :
  1. Speed of the vehicle, v (m/s)
  2. Horizontal dilution of Precission (HDOP)
  3. Perpendicular Distance, PD (m)
  4. Heading error, $HE = |\theta - \theta'|$
- Rule :
  1. If v is high and HE is small then L1 is average
  2. If v is high and HE is Large then L1 is low
  3. If HDOP is good and PD is short then L1 is average
  4. If HDOP is good and PD is long then L1 is low
  5. If HE is small and PD is short then L1 is average
  6. If HE is large and PD is long then L1 is low

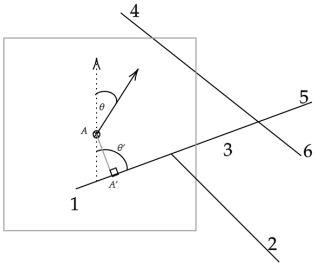


Figure: FIS1 diagram

FIS 2

- Additional input :
  1. $\Delta d = d - d_2$
  2. $HI = |\theta' - \theta|$
  3. α and β.
- Rule :
  1. If $\alpha < 90$ and $\beta < 90$ then L2 is high
  2. If $\Delta d > 0$ and ($\alpha \geq 90$ or $\beta \geq 90$) then L2 is low*
  3. If HI is small and $\alpha < 90$ then L2 is low
  4. If HI is small and $\Delta d \geq 0$ and (α or $\beta \geq 90$) then L2 is low*
  5. If HI is large and $\alpha < 90$ and $\beta < 90$ then L2 is low
  6. If HDOP is good and v is zero then L2 is high
  7. If HDOP is good and $\Delta d < 0$ then L2 is average
  8. If HDOP is good and $\Delta d \geq 0$ then L2 is low
  9. If v is high and HI is small then L2 is average
  10. If HDOP is good and v is high and HI is 180^o then L2 is high

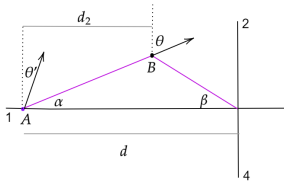
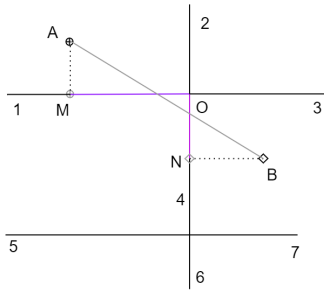


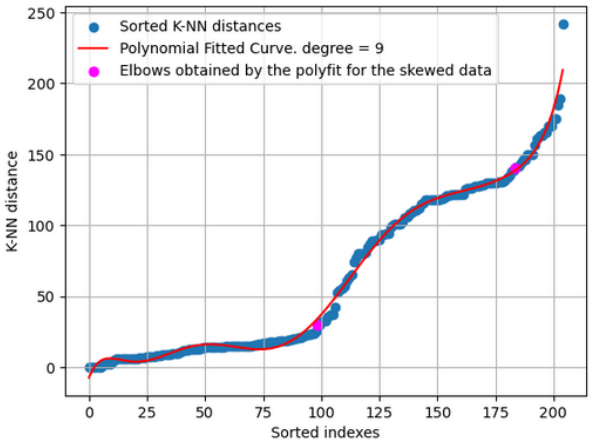
Figure: FIS2 diagram

FIS 3

- Additional input :
  1. Link Connectivity
  2. Distance error, difference between distance travelled from last position(A) and shortest path from the last matched position(M) to the current position (B) around the links (e.g $\epsilon = AB - (MO + ON)$)
- IMP rule plus some additional rules:
  1. If connectivity is low then L3 is low
  2. If connectivity is high then L3 is high
  3. If distance error is low then L3 is high
  4. If distance error is high then L3 is low



Improvement to DBSCAN proprecessing



The DBSCAN algorithm has the 2 parameters:

1. The eps parameter decides the search radius;
2. The min_pts parameter decides the threshold for a point to be classified into a core point.

We developed a method to determine the appropriate eps corresponding each data instance. We expect this method is more robust than one in the previous research, where the eps is fixed based on specific dataset.

Improvement to DBSCAN proprecessing

Futhermore, we propose "2-steps DBSCAN preprocessing":

1. The first DBSCAN with a relatively small eps mitigates stay points;
2. The second DBSCAN with s relatively big eps mitigates outliers.