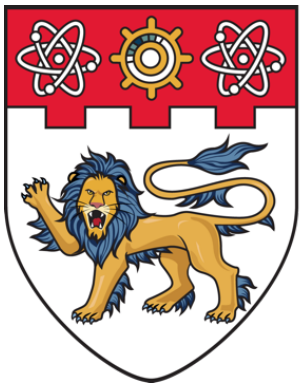


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AI6128 Urban Computing

Project 2

Trajectory and Road Network Data Analysis

authored by

Ong Jia Hui
jong119@e.ntu.edu.sg
G1903467L

Zhu ZhiCheng
zh0098ng@e.ntu.edu.sg
G1903395J

Tan Mengxuan
mtan099@e.ntu.edu.sg
G1903482L

December 3, 2020

Abstract

This project report on the topic of trajectory and road network data analysis will begin by outlining the project background and the given tasks to fulfill. In Section 2, we will describe our data preparation process for both the taxis' trajectory and road network data from Porto City, Portugal. We also documented our preliminary exploratory data analysis (EDA) on the trajectory dataset as well as the reason for our choice of visualization tool, Folium.

The raw GPS points of these processed trajectory data will then be visualized using Folium. It is expected that the raw data will not be aligned to the road network in these plots. Next, we will describe the map matching process using Fast Map Matching (FMM) algorithm proposed by Can Yang and Gyozo Gidofalvi [7] and utilize its open-source tool to map our trajectory dataset to the road segments.

The output of FMM will contain aligned GPS points, which will be plotted onto Porto City's road network again to visualize the differences in Section 5. With these mapped trajectory points, we then performed two route analysis such as extracting the top five most traversed and the longest average travelling time road segments. The algorithm we designed and the extracted road segments were stated and visualized in the Section 6.

Lastly, our team highlighted and analyzed the less satisfactory results in Section 5 and reasoned that it is caused by erroneous outlier points in Section 7. We implemented outlier detection and removal onto our trajectory data and re-performed map matching using FMM. These improved trajectory routing will also be plotted and shown using Folium.

Contents

1 Introduction

1.1 Project Background

Many vehicles such as taxis that are moving in an urban space have their locations sensed in real time nowadays. The sensed data corresponds to sequences of time-stamped points and is called trajectory data. Raw trajectory data usually involves noises (including sensor ones and measurement ones). As a result, the locations of a vehicle as indicated by the raw coordinates are often not on the road networks. A common practice is to map the trajectory data (of GPS points) to road networks before it is visualized and analyzed. This project is to conduct several tasks of preprocessing, visualizing and query processing trajectory data, where one important process is to map the trajectory data to a road network.

1.2 Goals

There are a total of six tasks for this project, with the last being an optional and bonus task. The main goal for this project is to perform trajectory and road network data analysis. The first step is to download the two sets of data, namely the trajectory and road network data. Once the datasets are prepared, visualizations for the raw GPS points from the trajectory dataset should be plotted onto the road network. The first ten trips will serve as a basis to represent the trajectory dataset. For the third task, we are asked to perform map matching to map the raw GPS points to the road segments. Fast Map Matching (FMM) [7] is recommended for this task. The results will then be visualized and analyzed in the fourth task. The fifth compulsory task requires us to derive and implement the extraction of the top five most traversed and longest average travelling time road segments and visualize them on graphs. The final bonus task wants us to identify less satisfactory map matching cases and implement ideas to improve mapping routes.

2 Task 1: Data Preparation & Exploratory Data Analysis

2.1 Trajectory Data Preparation

The full dataset originated from a Kaggle competition titled ,"ECML/PKDD 15: Taxi Trajectory Prediction (I)". The "train.csv" file provided by the Kaggle competition contains a total of 1,710,670 trajectories for all the 442 taxis running in Porto City, Portugal. Each data sample corresponds to one completed trip. It contains a total of nine features, described as follows:

1. TRIP_ID: (String) It contains an unique identifier for each trip;
2. CALL_TYPE: (char) It identifies the way used to demand this service.
3. ORIGIN_CALL: (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service.
4. ORIGIN_STAND: (integer): It contains an unique identifier for the taxi stand.
5. TAXI_ID: (integer): It contains an unique identifier for the taxi driver that performed each trip;
6. TIMESTAMP: (integer) Unix Timestamp (in seconds). It identifies the trip's start;
7. DAYTYPE: (char) It identifies the daytype of the trip's start.
8. MISSING_DATA: (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing
9. POLYLINE: (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start;

In task 1, we were tasked to extract the first 1,000 trajectories to be used for the remaining trajectory and road network data analysis tasks. This subset of trajectory data will be denoted as "train-1000". Out of the nine features in the dataset, only the "POLYLINE" feature column is relevant for this project.

2.1.1 Trajectory Data Processing

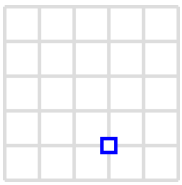
Table 1: Examples of original data type (POLYLINE) versus converted WKT formats

FORMAT	TYPE	EXAMPLE
WGS84	POLYLINE	[[-8.618643,41.141412],[-8.618499,41.141376]]
WKT	POINT	POINT (-8.618643 41.141412)
WKT	LINESTRING	LINESTRING(-8.618643 41.141412, -8.618499 41.141376)

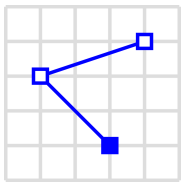
Trajectories can be represented using a markup language called Well-known text (WKT), which is often used to represent vector geometry objects. Each row of the "POLYLINE" column, which contains a nested array of list of lists, can be represented as a "LineString" (Figure 1b). Each list in the nested list array can be seen as a Point. Each LineString needs to have at least two "Point" (Figure 1a).

To better manage the "POLYLINE" feature in train-1000, we made use of a Python package called Shapely[1] to convert the string-based GPS coordinates from the array form into a LineString object.

[1] https://github.com/Toblerity/Shapely



(a) Point



(b) LineString

Figure 1: Examples of WKT formats

Each LineString object represents a trajectory in the dataset. Subsequently, we loaded all of the LineString objects into a GeoDataFrame using the GeoPandas[2] open-source library that extends the datatypes used by pandas to allow spatial operations on geometric types.

2.1.2 Trajectory EDA

We performed preliminary exploratory data analysis (EDA) on the train-1000 dataset and found that the "MISSING_DATA" column does not reflect empty GPS coordinates in the "POLYLINE" row. This is because all 1,000 "MISSING_DATA" values were denoted as False, but we found one missing GPS coordinate in the 763th row out of the 1,000 data instances.

Table 2: Statistics of train-1000's POLYLINE column

Description	Statistics
Empty Polyline	763th row (0 point)
Single point Polylines	9 rows [55, 115, 143, 191, 241, 308, 341, 430, 958]
Polyline with most points	505th row (493 points)
Total number of Points	46,774
Total number of valid LineStrings	990

Table 2 shows some statistics we obtained while generating the train-1000's GeoDataFrame. There are total of 46,774 GPS points in the train-1000 dataset. As at least two points are required to form a LineString, we found 9 polylines that only have a single GPS point. The total 990 LineString objects can be generated from the given subset.

[2]https://github.com/geopandas/geopandas

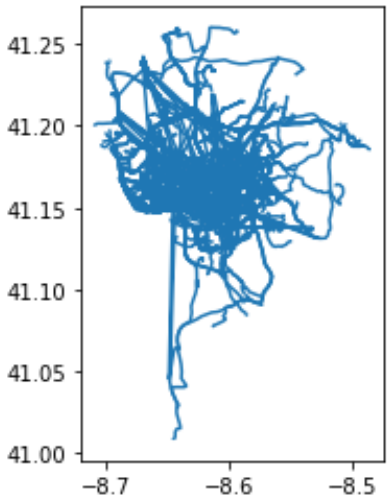


Figure 2: GeoDataFrame plot of all train-1000 LineString geometry objects.

The GeoDataFrame provides a plotting function like a normal pandas dataframe. Using this plot function, we can visualize the 990 LineString in a simple graph shown in Figure 2.

2.2 Road Network Data Preparation

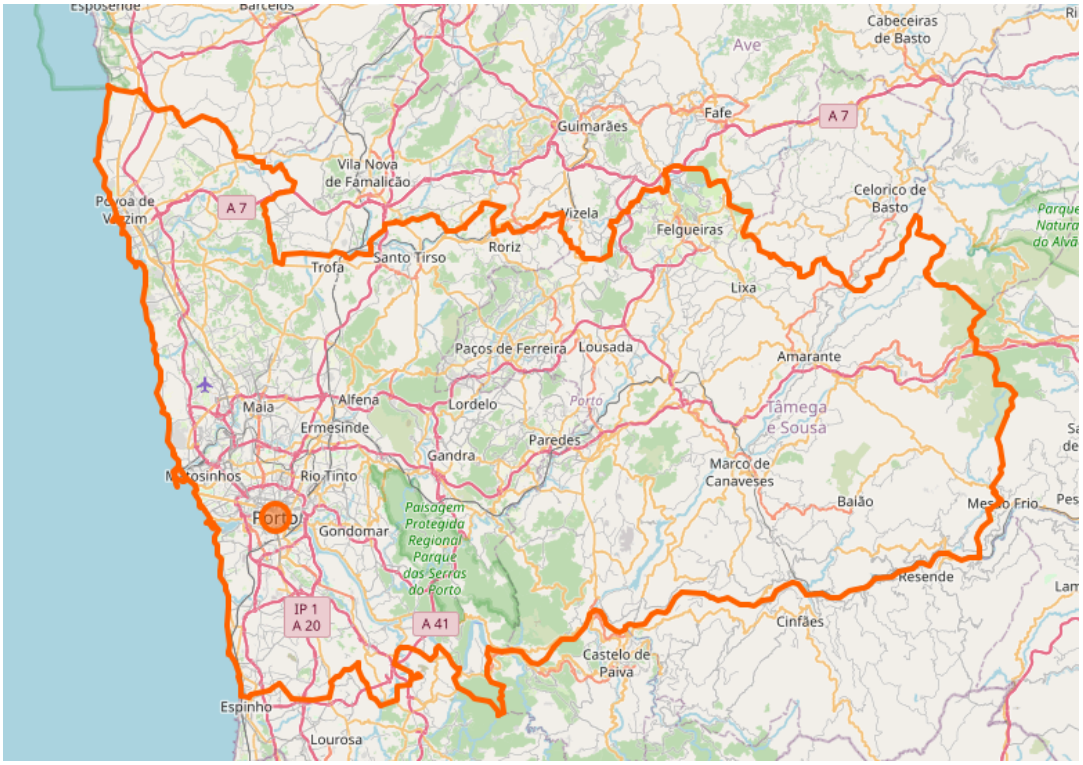


Figure 3: Extracted road network data within Porto's administratrive boundary (Relation:3459013)

The road network data of Porto City, Portugal is obtained from OpenStreetMap [5]. We downloaded the nodes and edges information of the road networks within Porto's administrative boundary[3] in Portugal as shown in Figure 3. In this project, we explored two types of network visualization libraries, namely OSMnx [1] and Folium[4]. Both of them are Python packages that can be used to retrieve, model, analyze, and visualize street networks from OpenStreetMap [5].

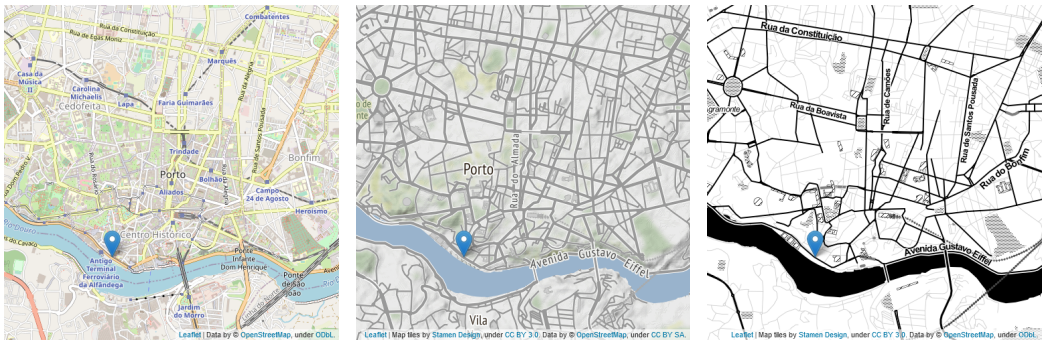
(a) OSMnx plot of Porto

(b) Folium plot of Porto with trajectories

Figure 4: Plots of Porto city by two different visualization methods, OSMnx (left) and Folium (right)

[3] https://www.openstreetmap.org/relation/3459013#map=10/41.2319/-8.1354
[4] https://github.com/python-visualization/folium

The main differences between the two visualization methods are shown in Figure 4. OSMnx downloads and loads high quality nodes and edges from OpenStreetMap, which makes it much slower to run than Folium. Folium allows plotting of points and lines onto an interactive Leaflet map, which overlays the geometry shapes onto a colored map from OpenStreetMap.



(a) OpenStreetMap Tile (b) Stamen Terrain Tile (c) Stamen Toner Tile

Figure 5: Different Leaflet base map tiles provided by Folium

Folium also provides a variety map tiles to choose from, as shown in Figure 5. These tiles increase the flexibility of visualizations for our trajectory plots. The OpenStreetMap tile (Figure 5a) provides detailed small roads to highway road coloring, but may not be suitable for multi-colored trajectories plotting as they will be hard to see. The Stamen Toner tile (Figure 5c) provides a white canvas background for plotting, but has less obvious minor road outlines. It is suitable for plotting short trajectories, which will be less visible in colored maps. Stamen Terrain tile (Figure 5b) can be seen as a middle ground between the two and will be used most commonly in our visualizations for this report.

2.2.1 Road Network Data Processing for OSMnx

Plotting on Folium does not require loading of graph data from files, but this is required by the OSMnx library [1]. There are two main ways of saving and loading of graph nodes and edges in OSMnx, namely saving and loading to a GraphML file format or separately saving the nodes and edges to graph shape (".shp") files. We mainly used the second method as the FMM (Fast Map Matching) library [7] used in Task 3 (Section 4) uses the edges shape file to load its Network object.

3 Task 2: GPS Point Visualization

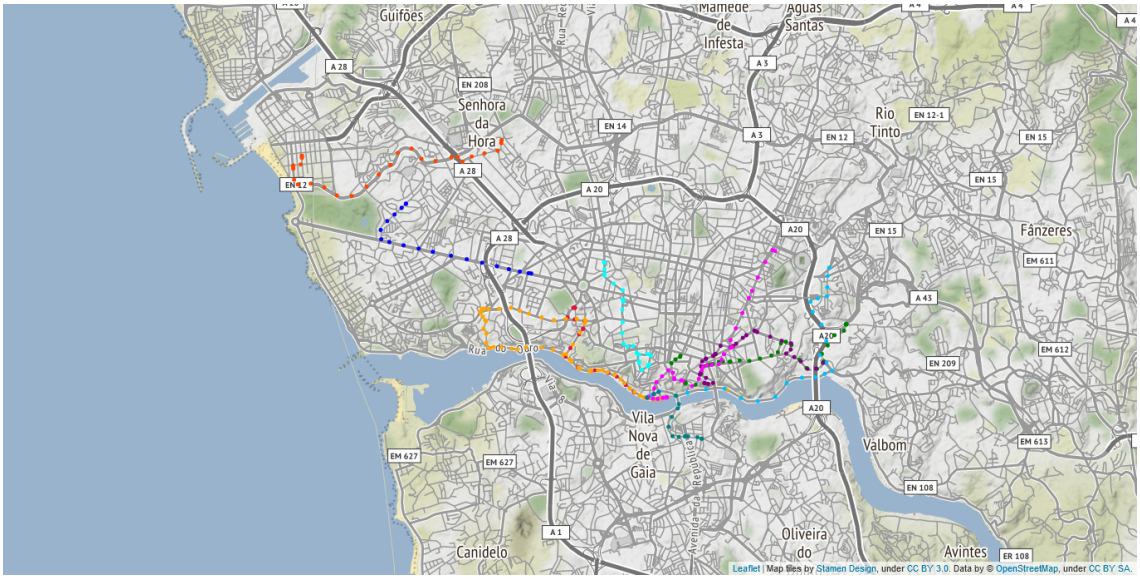
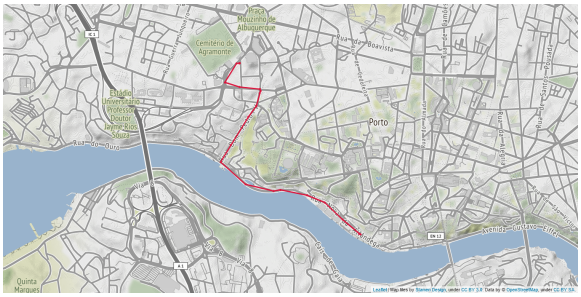
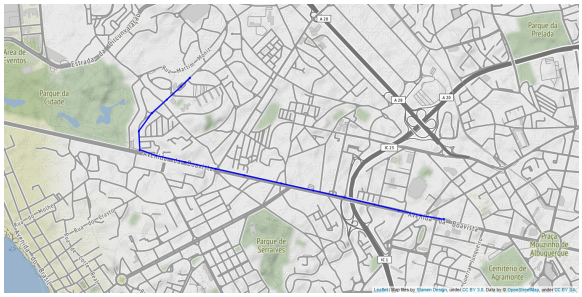


Figure 6: Raw GPS points of the first 10 trips in train-1000 (best viewed with digital zoom).

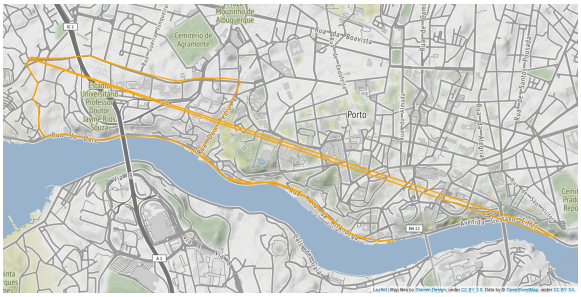
As described in the trajectory data preparation of Section 2, we converted the "POLYLINE" GPS coordinates array into a GeoDataFrame of LineString geometry objects. We are also able to reverse this process to get back the GPS points for plotting. By using Python's list slicing, we are able to get the trajectories of the first 10 trips and plot their raw points onto the Folium's Leaflet map. Figure 6 shows the GPS points plotted in 10 different colors, each representing one trajectory row instance.



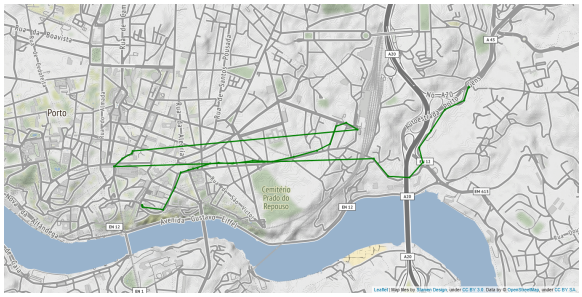
(a) Raw Trajectory 1



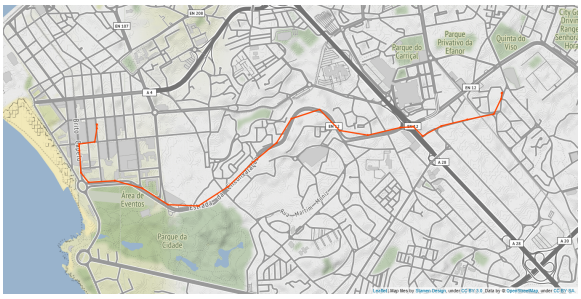
(b) Raw Trajectory 2



(c) Raw Trajectory 3

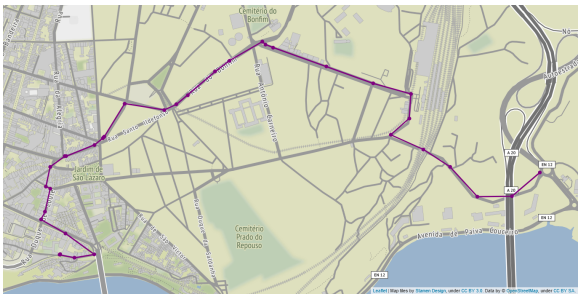


(d) Raw Trajectory 4

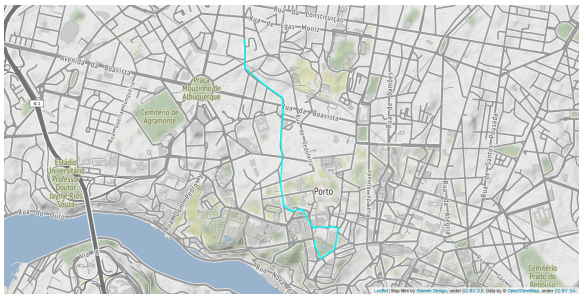


(e) Raw Trajectory 5

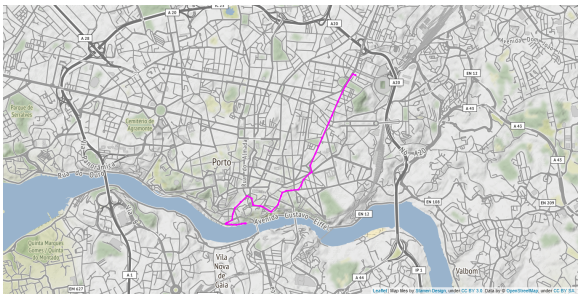
(f) Raw Trajectory 6



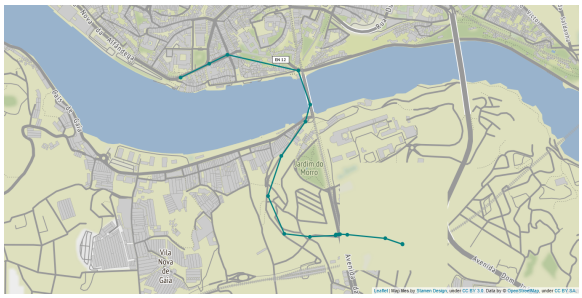
(g) Raw Trajectory 7



(h) Raw Trajectory 8



(i) Raw Trajectory 9



(j) Raw Trajectory 10

Figure 7: Individual plots for each of the 10 GPS trajectories with rough lines connecting each point.

Figure 7 shows the individual GPS points of the first 10 trajectories in train-1000. The raw LineString of these 10 trajectories are also drawn onto the same plots for better visualization of the GPS locations. As map matching has not been performed on the raw LineString data, the lines are not aligned to the road segments and are simply shown as a visual aid. Figures 7c and 7d are clear examples of unaligned "trajectory" lines.

In the zoom-in view of the GPS coordinates of Trajectory 4 shown in Figure 8, we can see that these GPS points are not aligned to the actual road segments. This once again shows the importance of map matching for trajectory data to align these GPS points to the most appropriate road segments. Figure 9 shows the zoom-in view of the stacked GPS points (top-left) of Trajectory 3. These stacked GPS points may likely be stay points, which need to be carefully handled in Task 5 (Section 6.

Figure 8: Zoom-in view of unaligned GPS points of Trajectory 4.

Figure 9: Zoom-in view of stacked GPS points of Trajectory 3.

4 Task 3: Map Matching

Given that the trajectories produced by the GPS points are not properly aligned to the roads segments as shown in Figure 7 of Section 3, it is essential to perform map matching to predict the actual locations given these inaccurate GPS points. In this section, we will first give an introduction to the problem of map matching followed by an overview of the main features of the Fast Map Matching (FMM) [7] algorithm. Finally, we will describe how the FMM algorithm is being applied to the train-1000 dataset.

4.1 Problem Definition

Before studying the map matching algorithm, lets first look at the definition of the map matching problem.

The map or the road network where we aim to project the raw GPS points onto is represented by $G = (V, E)$ where V is the set of nodes representing intersections and dead ends of road segments and E represents the road edges connecting the nodes. The goal of a map matching algorithm is to take in G and a sequence of temporally ordered GPS points denoted by $tr = \langle p_1, p_2, ..., p_N \rangle$ as inputs, and output a path denoted by $Out\_path = Path(\hat{p}_1, \hat{p}_2, ..., \hat{p}_N)$ where $\hat{p}_n$ represents the projected GPS point p_n onto the road network G and $Path()$ is a function that connects the projected points into an actual trajectory route on the road network.

4.2 FMM

The main goal of FMM is to deal with large amount of GPS data with fast computation speed. The FMM aims to achieve this objective in three stages:

- In the first stage, an upper bounded origin-destination hash table is first created from the road network to store information about all pairs of shortest paths whose lengths are under a certain threshold.
- Map matching is performed in the second stage where computationally expensive routing queries are replaced with searches in the hash table create in the previous stage.
- In the last stage, the problem of reverse movements or redundant routes are being identified and is being dealt with by applying a penalty.

In the following subsections, a brief description of the main features of each of the three stages will be given. The details of these can be found in the original paper [7].

4.2.1 Stage One: Precomputing

In the first stage, all pairs of shortest paths (SP) between nodes in G whose lengths are under a threshold Δ are calculated and stored in an Upper Bounded Origin Destination Table (UBODT). This hash table will be used in the subsequent stages to mainly reduce computation time when retrieval of SP between any two nodes in the road network is required.

4.2.2 Stage Two: Map Matching

In this stage, the Hidden Markov Model (HMM) is integrated with the precomputation done in the previous stage to infer the road segments traversed by a vehicle given its GPS points. This stage is executed in four steps:

- **Step 1: Candidate Search**

  This step mainly searches for candidate road edges given GPS points. An example of a candidate edge e given a GPS point p is shown in Figure 10.

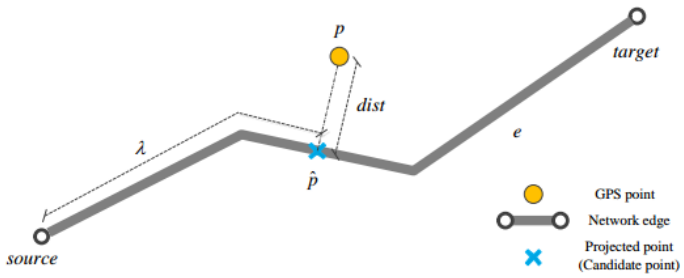


Figure 10: Example of a candidate road edge given a GPS point. From [7].

The location of the taxi is estimated as the projected point of p on $e.geom$, denoted by $\hat{p}$. The Euclidean distance from p to $\hat{p}$ is denoted as $dist$ and λ is the offset distance along $e.geom$ from the source node $e.s$ to $\hat{p}$. The projected point $\hat{p}$ on a road is found using a linear referencing algorithm [6] and by iterating through **each road segment** in the candidate road edge e and returning the one with the minimum distance. Formally, a candidate C for a GPS point p is defined as:

$$C = (\hat{p}, e, dist, \lambda) \tag{1}$$

However given the dense road network in urban cities, not just one but a **set of candidate road edges** are being constructed from a single point p. This set of k candidate edges is found by searching for the $k-$Nearest Neighbors (KNN) of road edges around p within a search radius of r, hence the set is denoted by:

$$CS(p) = KNN(p)_{k,r} \tag{2}$$

Therefore, given a trajectory of GPS points $tr = \langle p_1, p_2, ..., p_N \rangle$, a trajectory candidate set (TRCS) is constructed as:

$$TRCS(tr) = \langle CS(p_1), CS(p_2), ..., CS(p_N) \rangle \tag{3}$$

In $TRCS(tr)$, each candidate is denoted by C_n^i representing the i-th candidate matched to p_n where $1 \leq i \leq k$ and $1 \leq n \leq N$.

- **Step 2: Optimal path inference (OPI) using HMM**

  Before finding the optimal path from the candidate set $TRCS(tr)$ using HMM, the Shortest Path (SP) length between each candidate point C_n^i at the n^{th} step and all candidate points C_{n+1}^i at the $n+1^{th}$ step need to be found. These SP pairs represent the distance between the projected points on road edges and there are in total k^2N of them. Given the large amount of SP pairs, they can be retrieved directly from UBODT which significantly reduces computation during this inference stage. The SP distance from C_n to C_{n+1} is denoted as $l_{n,n+1}^C$ and $d_{n,n+1}$ denotes the Euclidean distance between the raw GPS points before they are projected on the road network. With $l_{n,n+1}^C$ and $d_{n,n+1}$, we can now proceed to compute the transition probabilities between road segments and emission probabilities from GPS points to points on road edges. In the FMM paper [7], the transition probability is defined as:

$$tp(C_n, C_{n+1}) = \frac{min(d_{n,n+1}, l_{n,n+1}^C)}{max(d_{n,n+1}, l_{n,n+1}^C)} \tag{4}$$

The formulation of Equation 4 essentially solves several problems that were not dealt with in previous works [4, 8]. These problems include having the transition probability greater than 1 and division by 0 error. For the emission probability, it follows that in [4] and assumes the GPS point follows a zero mean gaussian distribution around the taxi's true location. Hence, the emission probability of a candidate C_n is formulated as:

$$C_n.ep = \frac{1}{\sqrt{2\pi}\sigma} e^{-(C_n.dist)^2/2\sigma^2} \tag{5}$$

where σ is the standard deviation of the GPS error. Given the formulations of transition and emission probabilities in Equation 4 and 5, the score of a candidate path in $TRCS(tr)$ can be computed by:

$$score = \sum_{n=1}^{N-1} tp(C_n, C_{n+1}) \times C_{n+1}.ep \tag{6}$$

Finally, an optimal path O_path can be inferred efficiently using the Viterbi algorithm [3] as the path with the maximum score and O_path is denoted by:

$$O\_path = \langle \hat{C}_1, \hat{C}_2, ..., \hat{C}_N, \rangle \tag{7}$$

where $\hat{C}_n$ contains the road edge point in the optimal path at the n^{th} step.

- **Step 3: Complete Path Construction**

  After the O_path is obtained, a complete path C_path that concatenates the SPs connecting consecutive candidates in O_path is constructed in this step. C_path is denoted by:

$$C\_path = \langle \hat{C}_1.e, ..., \hat{C}_n.e, Path(\hat{C}_n.e.t, \hat{C}_{n+1}.e.s), \hat{C}_{n+1}.e, ..., \hat{C}_N.e, \rangle \tag{8}$$

  The query of $Path(\hat{C}_n.e.t, \hat{C}_{n+1}.e.s)$ is also performed in UBODT. Unlike in previous works [2, 9] where both the SP distance and path are queried and stored for each transition of candidates, FMM only queries the SP path after the O_path is obtained, thus saving up on computational resources.

- **Step 4: Geometry Construction**

  In the last step, the corresponding geometry is constructed by concatenating the polyline for each road edge in C_path.

With more effective use of UBODT created in Stage One, several methods in dealing with unnecessary long distance routing queries are also being proposed in FMM. The details can be found in [7].

Figure 11 (obtained from FMM documentation page[5]) illustrates the process of map matching done in this stage briefly. The raw GPS points (the orange points) are clearly not aligned with the road network. FMM then projects these GPS points onto the road network illustrated by the green crosses. Finally, the green lines are used to connect these green crosses to form the complete trajectory of the taxi.

4.2.3 Stage Three: Removing Reverse Movements

This last stage acts as a refinement to the OPI step in Stage Two.

Reverse movements of taxis can be observed either when the vehicles move back and forth on the two directed edges corresponding to a single bidirectional road segment. Two such examples are shown in Figure 12.

[5] https://fmm-wiki.github.io/docs/documentation/output/

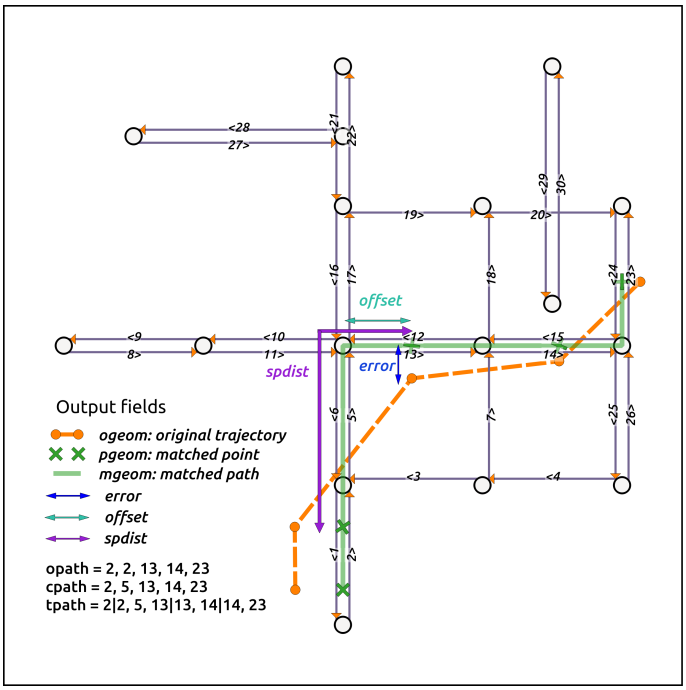


Figure 11: A simple illustration of the process of map matching done in Stage Two.

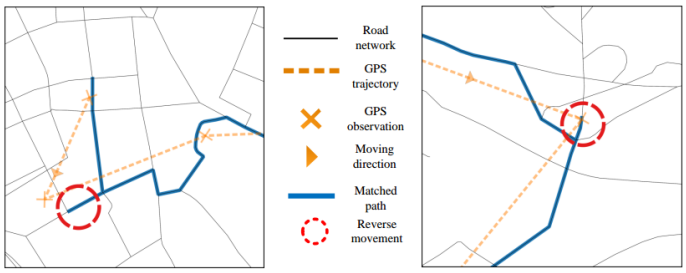


Figure 12: Two examples of reverse movement observed in matching real-world GPS data. From [7].

Although it could happen that the vehicle is actually moving back and forth on a road segment, a more common case is that there exists a bidirectional road segment and the error in GPS observation is large [7]. As a result, the path containing reverse movement can be returned in the OPI step of Stage Two because it has a higher score than the correct path. Therefore, a penalty is introduced when computing $l^C_{n,n+1}$. This will lead to a lower transition probability for reverse movements and they eventually gets eliminated in the OPI step. The detailed formulation of $l^C_{n,n+1}$ can be found in [7].

4.3 Applying FMM

Now, we move on to talk about how we apply FMM on our train-1000 dataset. Table 3 displays the parameters used when we apply the FMM on the dataset. A threshold Δ of 3 km is used for the construction of UBODT. In the candidate search step, when building the candidate set using KNN, we set k to 16 and the search radius r to 0.5 km. To calculate the emission probability in the OPI step, we set the GPS error σ to 0.05 km. This set of parameters is similar to that used by the author

Table 3: Parameters used in FMM.

Δ	3 km
k	16
r	0.5 km
σ	0.05 km

in an example[6] to apply FMM on the road network of Stockholm. However, for the dataset that we are using, we tuned the value of k and found the value of 16 gives more reasonable results as compared to that of 8.

After applying the FMM algorithm on the train-1000 dataset, 997 trajectories of taxis are able to be mapped successfully to the road network. FMM is not able to map 3 trajectories to the road network from the raw GPS points. One of them is for the trip in row 763 of train-1000 where no raw GPS point is present as stated in Table 2. The other two trips are those in row 87 and 612. We will treat the raw GPS points from these two trips as noise for now since only two out of one thousand of them cannot be handled by FMM. However, further analysis has to be done in the future to find out the exact reason for FMM's behaviour on these two trips.

[6] https://github.com/cyang-kth/fmm/blob/master/example/osmnx_example/map_matching.ipynb

5 Task 4: Route Visualization

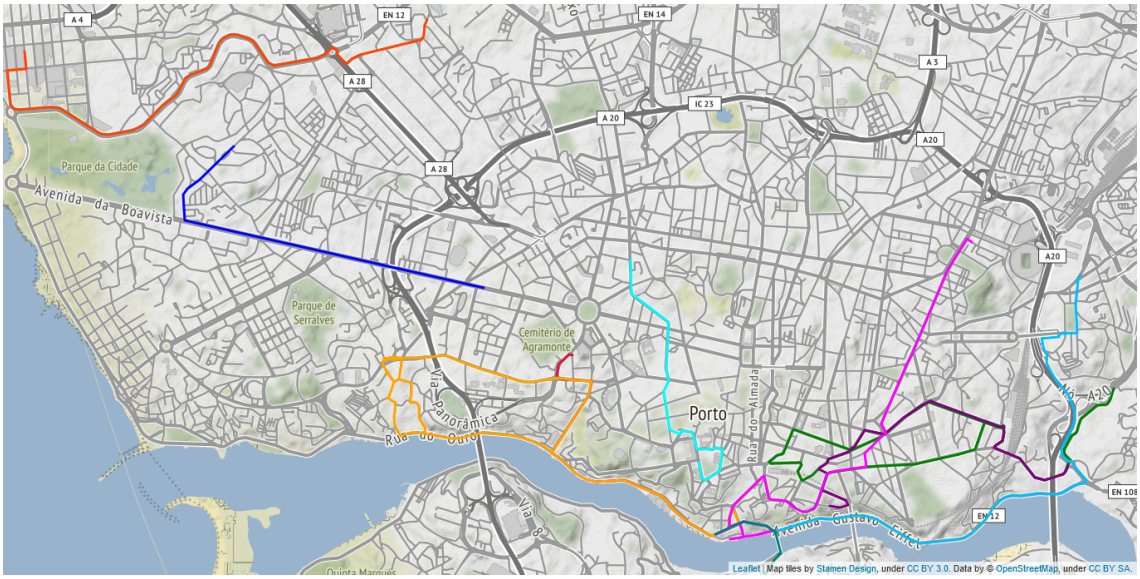
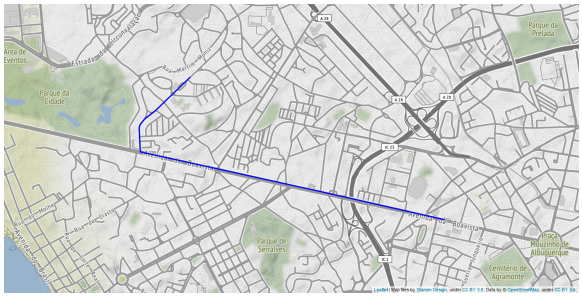


Figure 13: Mapped trajectories of the first 10 trips in train-1000.

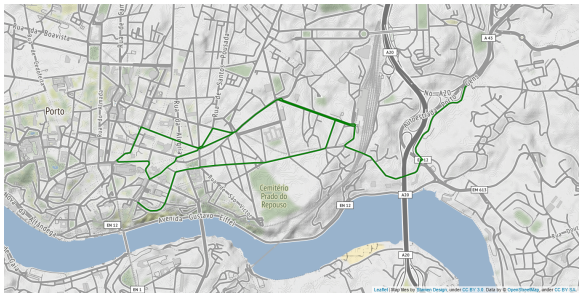
After performing map matching, we use the output, "match_result.csv" generated by FMM [7] to plot the routes from train-1000 on the Porto City road network map. Figure 13 shows the first 10 trajectories in train-1000 plotted in 10 different colors. Note that the red trajectory has overlapped with the light orange trajectory in Figure 13, which means that the two trajectories took the same route towards the south coastal area.

(a) Trajectory Route 1

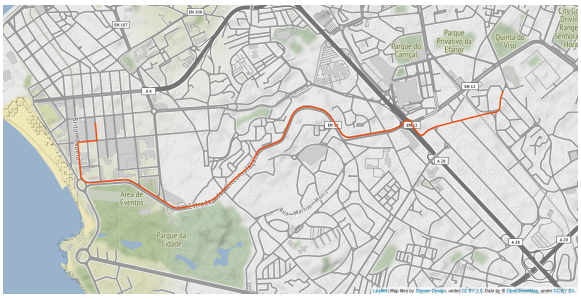


(b) Trajectory Route 2

(c) Trajectory Route 3



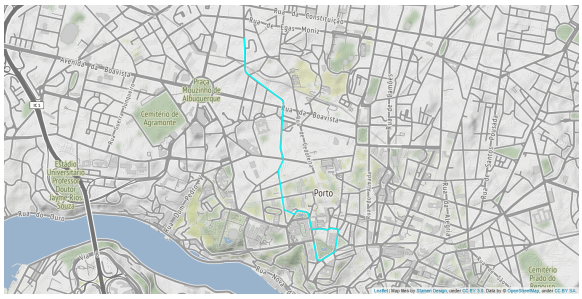
(d) Trajectory Route 4



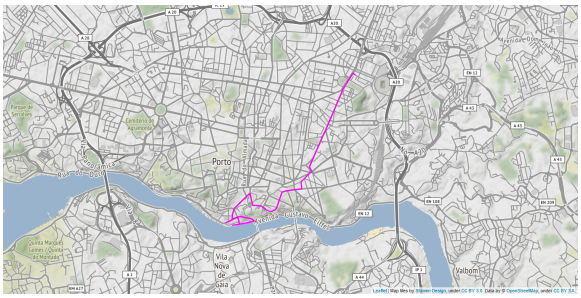
(e) Trajectory Route 5

(f) Trajectory Route 6

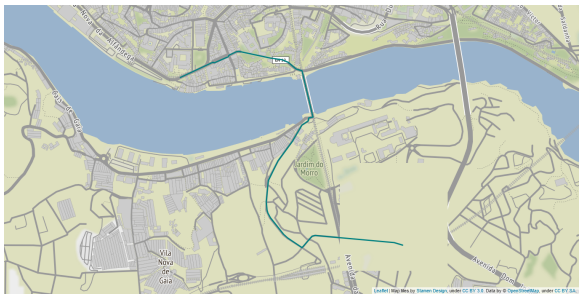
(g) Trajectory Route 7



(h) Trajectory Route 8



(i) Trajectory Route 9



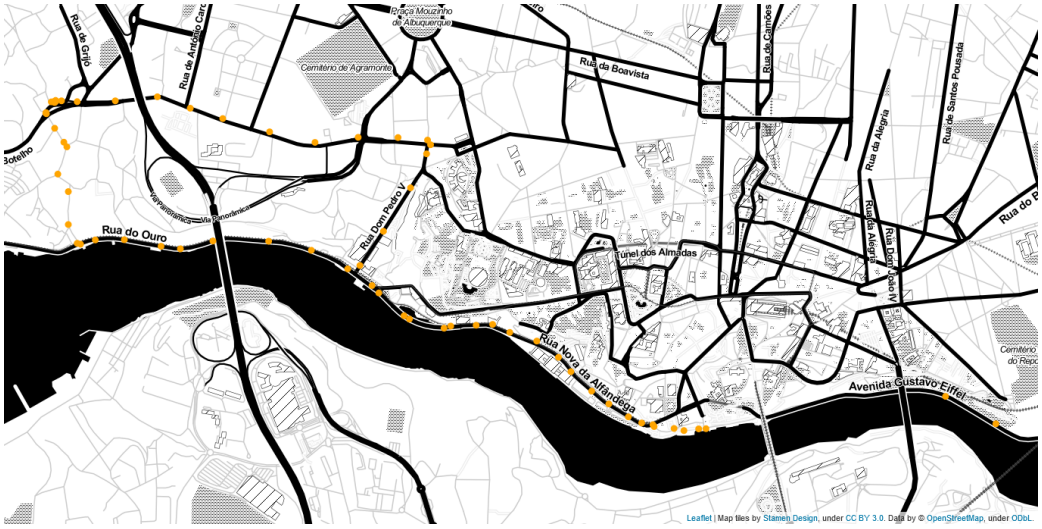
(j) Trajectory Route 10

Figure 14: Individual plots for each of the 10 routes after map matching.

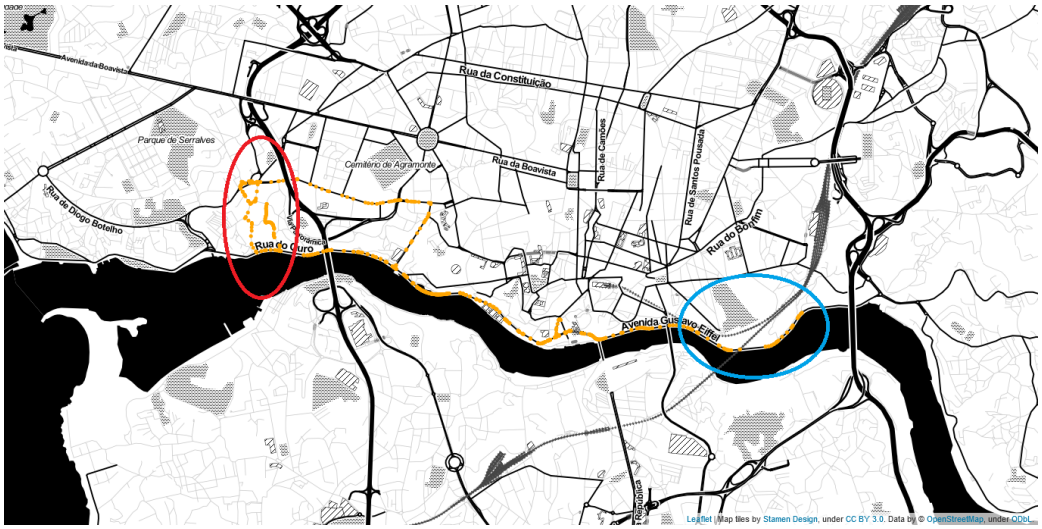
Figure 14 shows the individual mapped routes of the first 10 trajectories in train-1000. FMM [7] has mapped the trajectories for route **1** and **2** (Figure 14a and 14b) onto the road segments very well especially along the road turns. Some other positive examples include route **5**, **6**, **7**, **8**, **9** and **10**, shown in Figure 14e, 14f, 14g, 14h, 14i and 14j respectively.

However, we found less ideal results from FMM [7] for routes **3** and **4** shown in Figure 14c and 14d respectively. To analyze the reasons for these poor performance, we plotted the comparison plots between the raw GPS points and FMM mapped points in Figure 15 and 16 for Trajectory 3 and 4. We found out that for trajectory 3, there are a number of additional GPS points which appeared in the area highlighted in the red circle of Figure 15b. The appearance of these additional points resulted in the hourglass trajectory seen in Figure 14c. In addition, FMM also added new points to the tail of the trajectory as shown in the blue circle of Figure 15b. Route 4 shows a scenario whereby FMM removed incorrect GPS points and resulted in change in trajectory routing. This is exemplified in the area highlighted by the purple circle in Figure 16b, whereby the removal of the original two raw GPS points, resulted in the additional of multiple points (area in red circle) to link up the gap. This causes unnatural trajectory routing. We highly suspect these unsatisfactory results are caused by outliers in

the trajectory data. We will further explore ways to filter these outliers in Section 7.

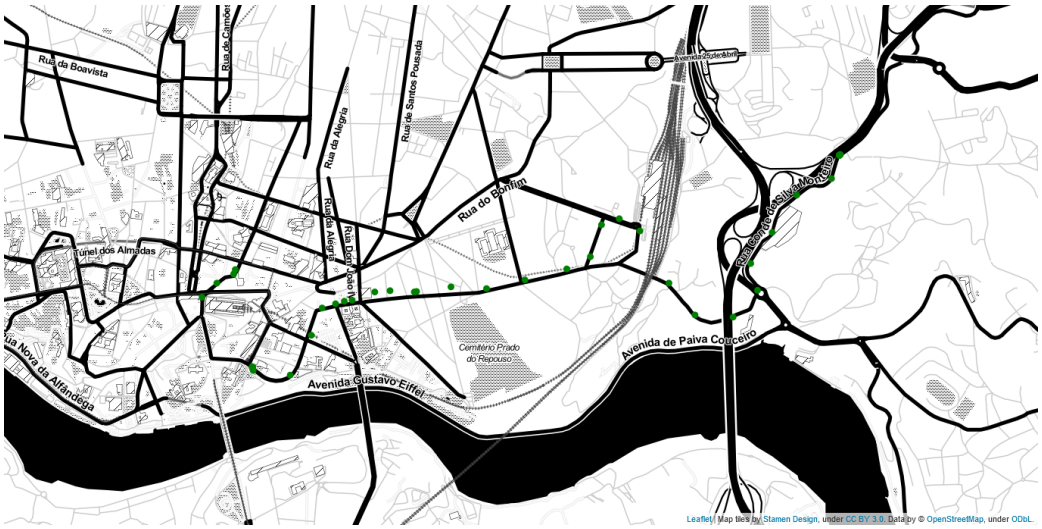


(a) Raw GPS points (increased marker sizes)

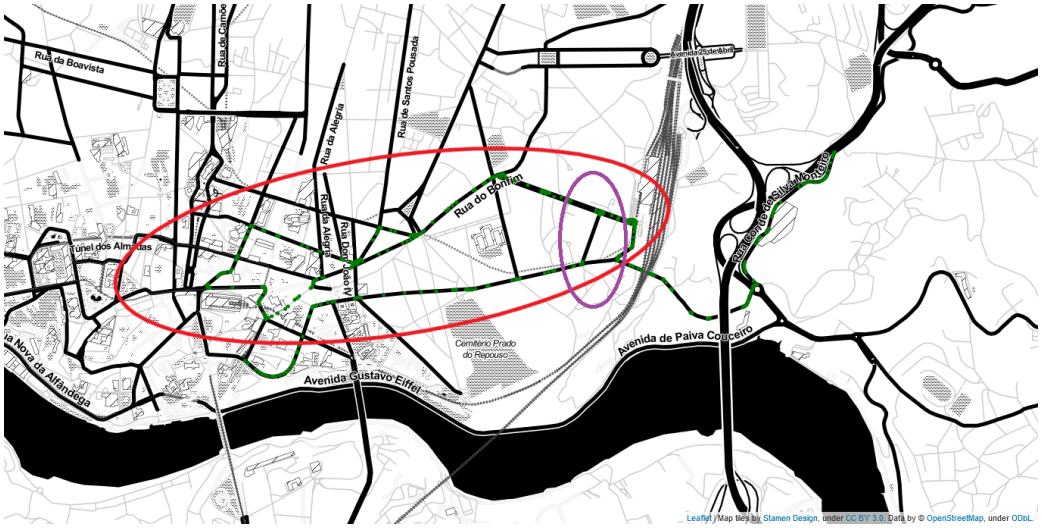


(b) Mapped GPS points

Figure 15: Comparison between original GPS points and FMM mapped points of Trajectory 3.



(a) Raw GPS points (increased marker sizes)



(b) Mapped GPS points

Figure 16: Comparison between original GPS points and FMM mapped points of Trajectory 4.

6 Task 5 Route Analysis

6.1 Top Five Most Traversed Road Segments

6.1.1 Algorithm and Implementation

Task 5 is based on matched routes for the first 1,000 trajectories in data which are saved in "match_result.csv". Each matched trajectory contains following important features:

1. cpath: (list) It contains the roads traversed by the matched route.
2. opath: (list) It contains roads matched to each original GPS points in trajectory.
3. offset: (list) It contains the distance from the matched point to the start of the matched road.
4. length: (list): It contains length of the matched road for each original GPS point.
5. spdist: (list): It contains shortest traversed distance between consecutive matched points in matched route for original GPS points;
6. mgeom: (list) It contains points which form the geometry of the matched route;

In this task, we use cpath in each matched result to count the passed frequency for each road segment. Then we select the five roads which have highest frequency in 1,000 cpath lists as the most often five roads.

6.1.2 Results Visualization

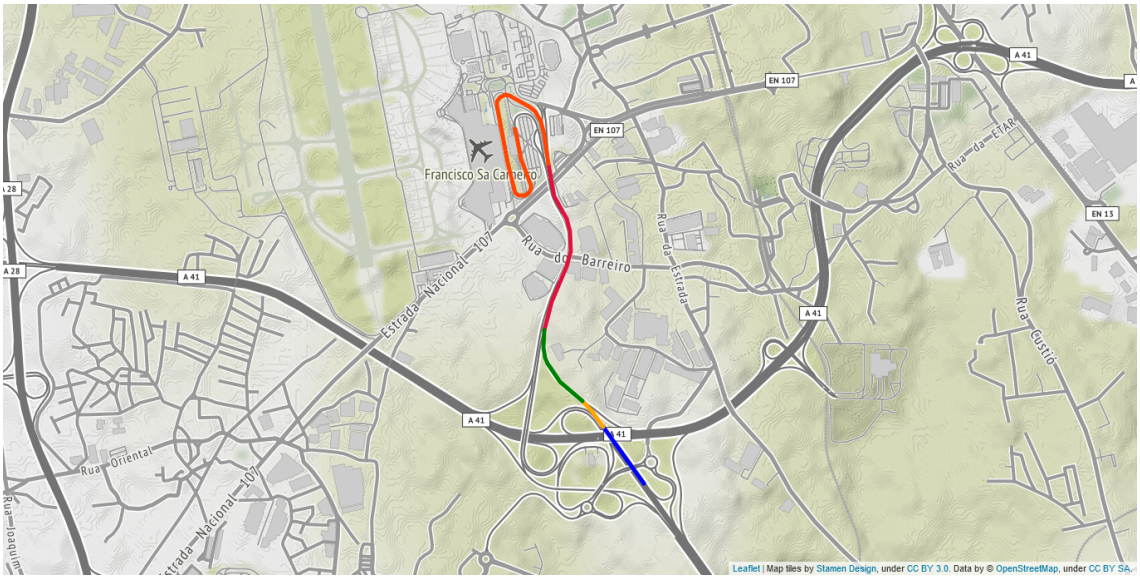


Figure 17: Five most traversed road segments plotted in five different colors.

Figure 17 shows the visualization of our results from our most traversed road segment algorithm. It shows that the top five most traversed road segments are all driving towards or returning from the Francisco Sá Carneiro Airport. It also shows that the five road segments are connected, which is unsurprising as the taxis have to take the same highway to and fro the airport most of the time. The benefits of using Folium to visualize trajectories can also be exemplified in Figure 17, whereby it reveals the point of interest ("airport") and helps to reason why the top five road segments are connected ("highway").

6.2 Top Five Longest Travelling Time Road Segments

6.3 Algorithm and Implementation

In this task, we need to estimate each road average travelling time of the 1,000 trajectories. Unlike five most often traversed road segments, this task is more uncertain due to the following reasons:

1. Most of the trajectories start from the middle of a road and end in the middle of other road.
2. Many consecutive GPS points are very close, which might be caused by stay point or traffic jam.
3. Some roads have very few travelling logs for 1,000 trajectories, and some roads have enough travelling logs but total travelling distance for these logs is very small, which makes its average travelling time less confident.

To solve these problem, the algorithm is described as below. The time between each consecutive GPS points is 15 seconds (15s) as indicated in Kaggle. We use an example to explain our time estimation algorithm for this task Figure 18.

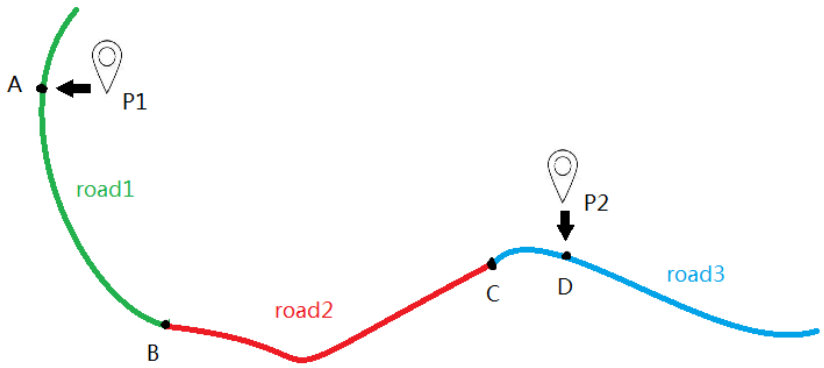


Figure 18: Algorithm of time estimation

Suppose two consecutive GPS points from a trajectory is P_1 and P_2, and their corresponding matching point on matched road is A and D respectively. The travelling route for this car in 15s is $A \longrightarrow B \longrightarrow C \longrightarrow D$, where B is the joint point of $road_1$ and $road_2$ and C is the joint point of $road_2$ and $road_3$. Assume length of AB, BC, CD are L_{AB}, L_{BC}, L_{CD} respectively. Hence, the total travel distance between the two consecutive GPS point in 15s is:

$$L_{P_1P_2} = L_{AB} + L_{BC} + L_{CD}$$

We also assumed the car is moving at a constant speed between two consecutive GPS and grouped each GPS pairs and their adjoining road as a three-element tuple, which we called a "log":

$$(road_1, L_{AB}, \frac{L_{AB}}{L_{P_1P_2}} \times 15), \quad (road_2, L_{BC}, \frac{L_{BC}}{L_{P_1P_2}} \times 15), \quad (road_3, L_{CD}, \frac{L_{CD}}{L_{P_1P_2}} \times 15)$$

The first element of each tuple is road ID, the second element is passing distance on this road and the third element is estimated passing time of corresponding part of road segment.

The above example is just two consecutive GPS points in one trajectory, we need to count each consecutive GPS pairs on 1,000 trajectories. Denote all logs as $LOGs$, then we estimate the passing

time for one specific $road_k$ as:

$$meanT_k = L_{road_k} \times (\sum_{log \in LOGs} log[2] \times I_{\{log[0]=road_k\}}) \ / \ (\sum_{log \in LOGs} log[1] \times I_{\{log[0]=road_k\}})$$

where I is the indicator function. It means we use the length of this road divide the total passing distance and times total passing time as its single pass time estimation.

For those GPS points which are very close (or at exact the same position) which makes the passing time longer than normal, we considered them as stay points and filtered these out by adding a condition: $log[1] >= \epsilon$ to remove those with short passing distance .

For road which does not have more than 10 logs we are not confident of its passing time estimation, so this road will not be counted in passing time ranking. Also, if a road have more than 10 logs but the total passing distance of its logs is still a small value comparing to its total length, we will not rank this road segment (e.g. a road have 10 logs but each of logs only pass $\frac{1}{100}$ part of this road).

6.3.1 Results Visualization

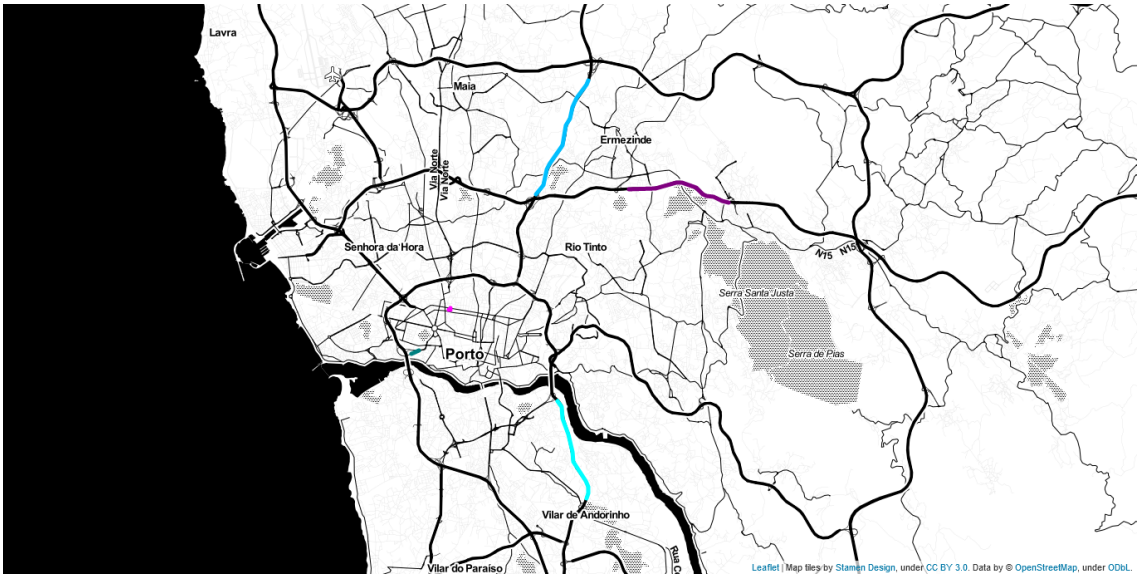
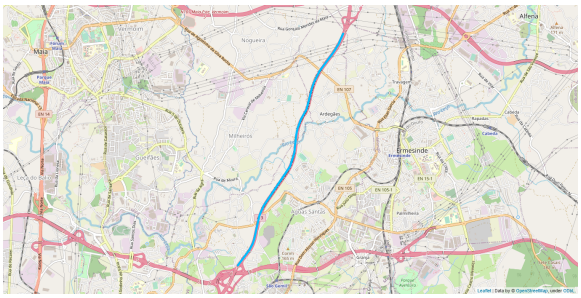
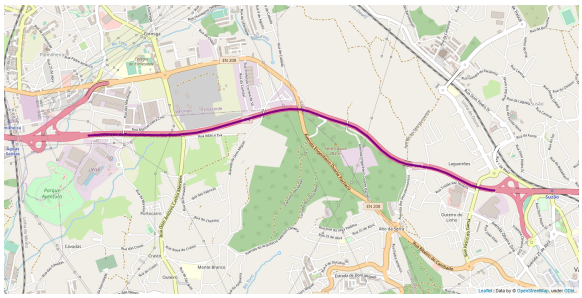


Figure 19: Five longest average travelling time road segments plotted in five different colors.

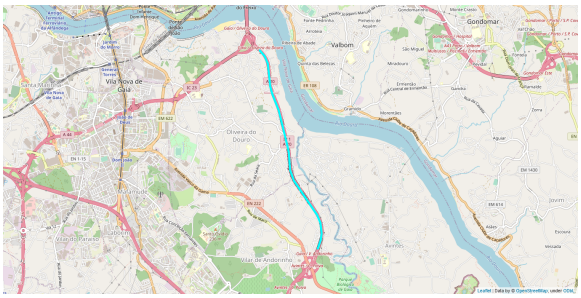
Figure 19 shows the visualization of our results from our longest travelling time road segment algorithm. It highlights the top five longest average travelling time road segments in five different colors. The results contains a mix of long and short road segments. We will plot the individual road segments to visualize the possible reasons behind the differences in length.



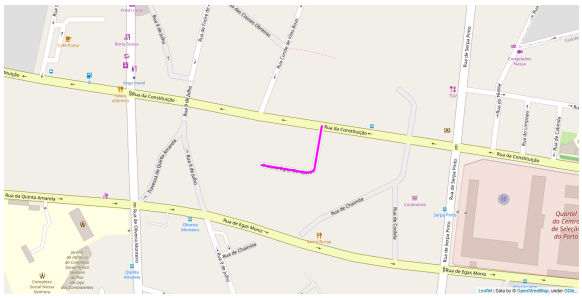
(a) Road Segment 1



(b) Road Segment 2



(c) Road Segment 3



(d) Road Segment 4

(e) Road Segment 5

Figure 20: Individual plots for 5 road segments of longest travelling time.

For long road segments like road segment 1, 2, 3 and 5 as shown in Figure 20a, 20b, 20c and 20e respectively, their roads looked to be on highways, which are prone to traffic congestion. This could be the reason for their long average travelling time. Furthermore, there is one anomaly, which is the short road segment 4 shown in Figure 20d. The reason why it was flagged could be that it is a minor road that requires slow navigation.

7 Task 6: Advanced Map Matching Ideas

As shown in Section 5, the Trajectory Route **3** (Figure 15) and Trajectory Route **4** (Figure 16) produced by the FMM [7] are less satisfactory as compared to the map matching results of other trajectories. This can be seen from the unnecessary paths displayed in Figure 15b and 16b. Hence, in this section, we will aim to improve the results of Trajectory Route **3** and Trajectory Route **4** by first introducing the intuition behind our approach followed by the proposed algorithm that helps us achieve the improved results. Finally, we will end this section by displaying the visualizations of the improved results.

7.1 Outlier Detection

Figure 24a and 25a display the raw GPS points of Trajectory Route 3 and 4 respectively when being connected by lines in the temporal order that they are being produced. In both of these figures, we can observe long lines connecting pairs of raw GPS points that are produced one after another. For instance in Figure 24a, we can see a line connecting a GPS point near the location of *Avenida Gustavo Eiffel* and another GPS point near *Estádio Universitário Prof. Dr. Jayme Rios Souza*. A connection between this two locations suggests that the taxi has traversed between these two places which are approximately 6km apart in 15 seconds since GPS points are collected every 15 seconds according to the data description given in Section 2.1. However, we know that it is not possible for a taxi to travel at such speed, thus we can infer that there exist noises or outliers in the Raw GPS points in the train-1000 dataset. To identify these outliers, we first plot the distribution of Euclidean distances between consecutive raw GPS points for all trips shown in Figure 21.

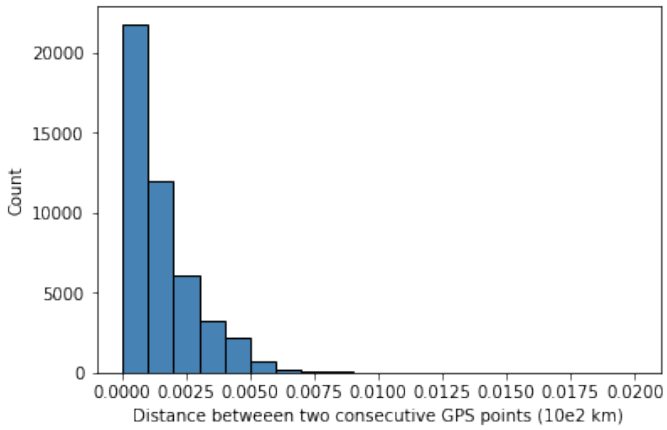


Figure 21: Distribution of Euclidean distance between consecutive raw GPS points in all trips.

From Figure 21, we can see that majority of the distance between pairs of GPS points are less than 500m which indicates a reasonable speed the taxi is travelling in a 15 seconds timeframe. However, Figure 21 also suggests that some taxis has travelled more than 1km in 15 seconds (240km/h), which is highly unlikely given the speed limit of vehicles on the road.

The boxplot in Figure 22 is able to give us a clearer view on where the outliers may lie. The black colored points which lie at least 1.5 times the interquartile range away from the third quartile (75^{th} percentile) indicates that a Euclidean distance of approximately 1km between two consecutive GPS points may indicate a presence of outliers. These outliers may be produced during the trips as a result of GPS error or error in the data collection process.

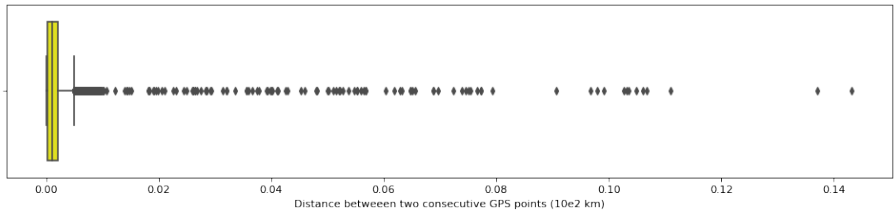


Figure 22: Boxplot of Euclidean distance between consecutive raw GPS points in all trips. Outliers are illustrated by the black colored points which lie at least 1.5 times the interquartile range away from the third quartile.

7.2 Proposed Algorithm

Given the presence of outliers in the raw GPS points of trips in train-1000, we would like to propose a simple algorithm to identify and remove these outliers before applying the FMM algorithm on them for map matching.

Input: The sequence of raw GPS points for a taxi trip arranged in time order `tr` = $[p_1, p_2, ..., p_N]$.
Minimum threshold `threshold_min`.
Maximum threshold `threshold_max`.
Threshold step `threshold_step`
Output: New sequence of raw GPS points for the same taxi trip `tr_new` where `len(tr_new)` $\leq N$.

```
tr_new = [ ];
tr_new.append(tr[0]);
cur_point = tr_new[0];
threshold = threshold_min;
for idx in range(len(tr) - 1) do
    next_point = tr[idx + 1];
    if distance(cur_point, next_point) < threshold then
        tr_new.append(next_point);
        threshold = threshold_min;
        cur_point = next_point;
    else
        threshold = threshold + threshold_step;
        threshold = min(threshold, threshold_max);
    end
end
return tr_new
```

Algorithm 1: Pseudocode for detecting and removing outlier GPS points in a taxi trip

---

Algorithm 1 displays the pseudocode for detecting and removing outlier GPS points in a taxi trip. We use a `distance()` function to compute the Euclidean distance between two consecutive GPS points. We first compare this distance with `threshold_min` and if it is smaller than `threshold_min`, we will append the more recent GPS point into `tr_new`. If not, we will ignore the more recent GPS point and increase `threshold_min` by `threshold_step` and use that as the 'new' threshold for comparing distance of subsequent GPS points. The reason for increasing the current threshold by a tiny amount is that skipping a GPS point would mean that `cur_point` and `next_point` would be more than 15 seconds apart in the next iteration. Hence, we allow some additional distance to be added to the current threshold to detect outliers. However, we would also ensure that the current

threshold is below the maximum threshold `threshold_max`. We set `threshold_min`, `threshold_max` and `threshold_step` to 1km, 2km and 0.2km respectively. We iterate all GPS points and return `tr_new` which contains the raw GPS points after the removal of outlier GPS points.

7.3 Results Visualization

Now, we apply FMM again, but this time on the new set of GPS points `tr_new` for each taxi trip. The results for the improved route trajectories of the first 10 trips are visualized in Figure 23. We could observe that those trajectories (i.e., route **1**, **2**, **5**, **6**, **7**, **8**, **9**, **10**) which had good results (shown in Section 5) before the removal of outlier points, still retain their performances after we preprocess their raw GPS points using Algorithm 1. In addition, the results of FMM on route **3** and **4** have improved after the raw outlier GPS points are being removed.

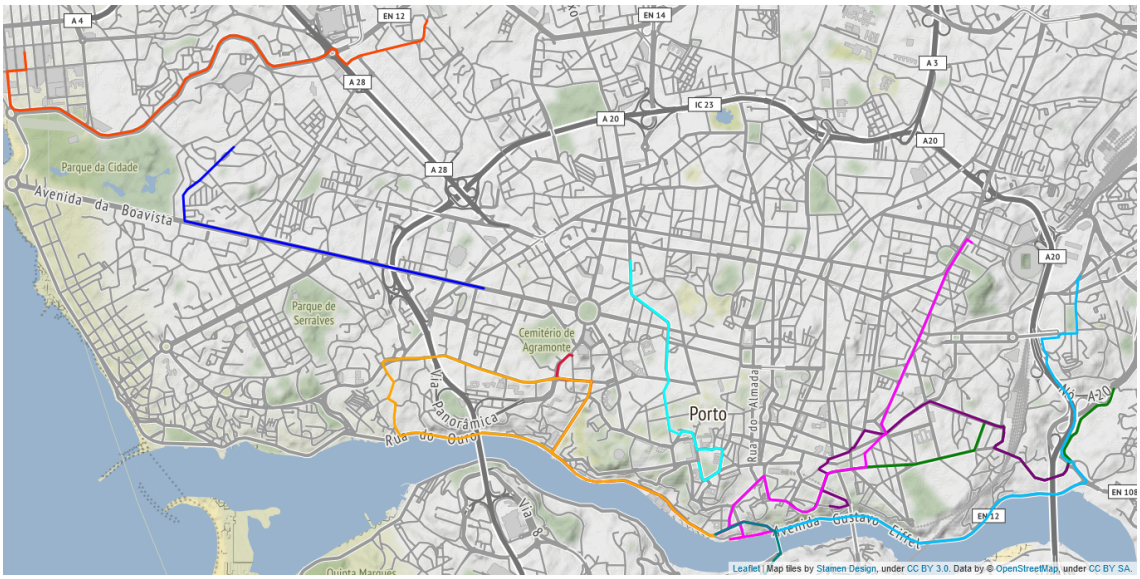
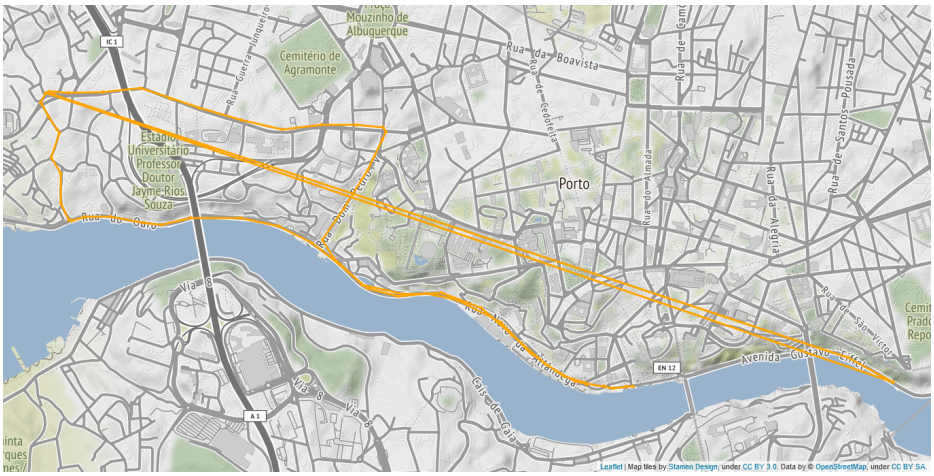


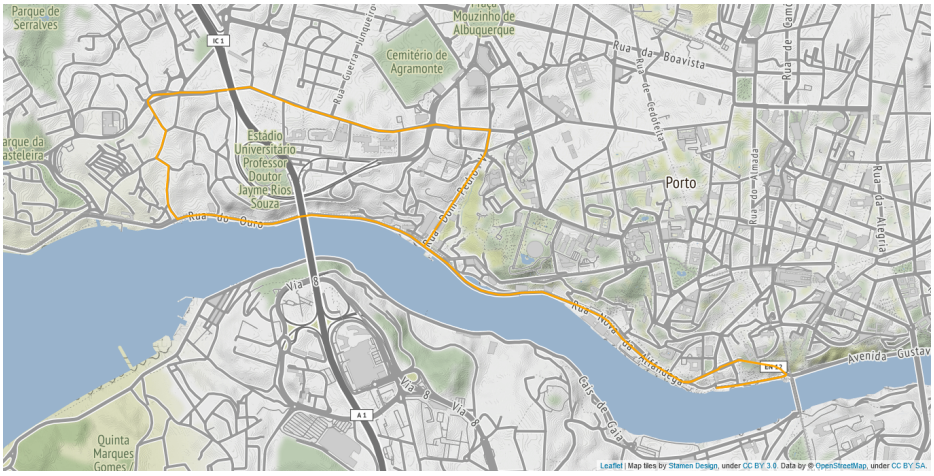
Figure 23: Improved trajectory routing after outlier removal.

Looking closer at the improved results of route **3** and **4** in Figure 24c and Figure 25c respectively, the hourglass shaped trajectory of route **3** originally seen in Figure 24b and 15b is now being removed as shown in Figure 24c. Also, several unnatural paths of route **4** originally seen in Figure 25b and 16b are also removed and shown in the improved result in Figure 25c. In general, the improved trajectories of route **3** and **4** have become more natural and reasonable after the removal of outlier GPS points from those trips.



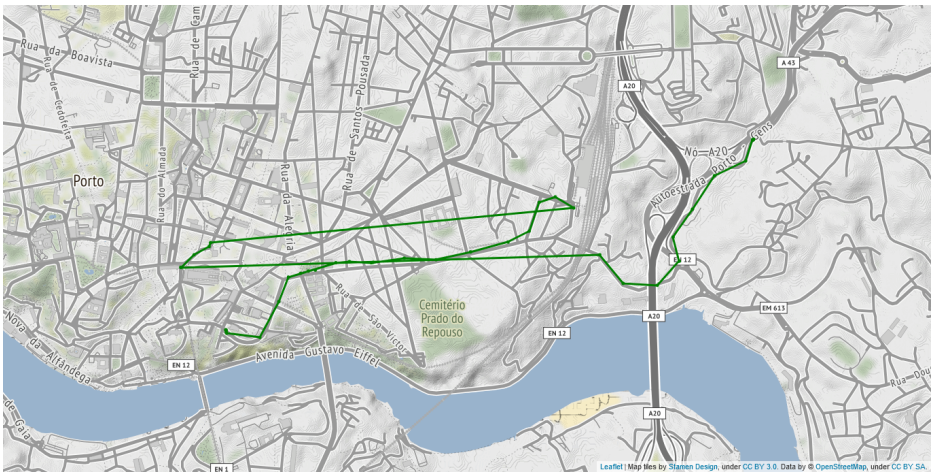
(a) Raw Trajectory 3 with outliers

(b) Original Routing with hourglass pathing

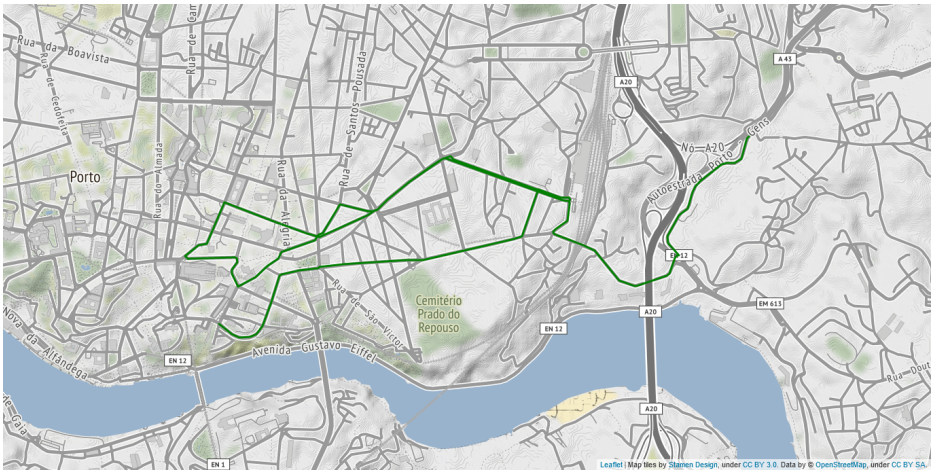


(c) Improved Routing after outlier removal

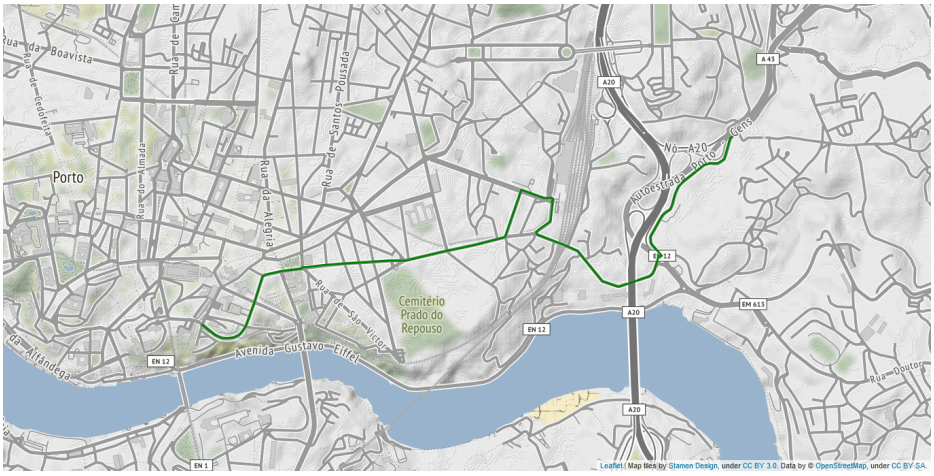
Figure 24: Comparison between raw trajectory, original routing and aided routing after outlier removal of Trajectory 3.



(a) Raw Trajectory 4 with outliers



(b) Original Routing with unnatural roundabout driving



(c) Improved Routing after Outlier removal

Figure 25: Comparison between raw trajectory, original routing and aided routing after outlier removal of Trajectory 4.

8 Conclusion

We started this course project by first understanding the motivations and goals behind analysing trajectory and road network data followed by an exploratory data analysis of GPS points collected from taxi trips completed in Porto City, Portugal. We also realised that Folium is a better visualization tool as compared to OSMnx for trajectory data in terms of both computation speed and aesthetic quality.

After having a better understanding of the goals and data for the project tasks, we then move on to visualise the raw GPS points for the first 10 taxi trips in Section 3 and showed that most GPS points are not aligned to the actual road segments due to the presence of GPS error. Therefore, in Section 4, we introduce a map matching algorithm: FMM [7] which can be used to correct these GPS errors and project the raw GPS points on the road network. A description on how the FMM is applied to the train-1000 dataset is also provided. Next, in Section 5, the results of FMM on the raw GPS trajectories are illustrated. Again the trajectory routes of the first 10 trips are being visualised and we give a discussion on both the positive and negative results produced by FMM.

Moving on to Task 5 in Section 6, we describe the algorithms for finding both the top five most traversed road segments and longest travelling time road segments. We visualized the results using Folium and realised that the top five most traversed road segments are all for driving towards or returning from the Porto airport and most of the long travelling time road segments are highways that can be prone to traffic congestion.

Finally, Section 7 presents our approach for the bonus Task 6. We identify that the presence of outliers in the raw GPS points of taxi trips is the cause for the negative results produced by FMM shown in Section 5. Therefore, an algorithm is proposed to detect and remove these outliers and then apply FMM again on the new set of GPS points. The outputs from FMM on the new set of GPS points display more natural and reasonable taxi trajectories on the road network.

References

[1] Geoff Boeing. "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In: *Computers, Environment and Urban Systems* 65 (2017), pp. 126–139.

[2] Bi Yu Chen et al. "Map-matching algorithm for large-scale low-frequency floating car data". In: *International Journal of Geographical Information Science* 28.1 (2014), pp. 22–38.

[3] G David Forney. "The viterbi algorithm". In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278.

[4] Yin Lou et al. "Map-matching for low-sampling-rate GPS trajectories". In: *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. 2009, pp. 352–361.

[5] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017.

[6] Mahmood Rahmani and Haris N Koutsopoulos. "Path inference from sparse floating car data for urban networks". In: *Transportation Research Part C: Emerging Technologies* 30 (2013), pp. 41–54.

[7] 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.

[8] Jing Yuan et al. "An interactive-voting based map matching algorithm". In: *2010 Eleventh international conference on mobile data management*. IEEE. 2010, pp. 43–52.

[9] Zhe Zeng et al. "Acceleration of map matching for floating car data by exploiting travelling velocity". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE. 2015, pp. 2895–2899.