Fast and Safe Route Planning Project

Group 10

Authors: Abhinandan Desai, Mihir Naresh Shah

1. Abstract

There are several routes that can be taken between a source and a destination. Each of these routes have different attributes that define it. In this project, the aim is to find the best route between a source and a destination by using GPS data from several files, each having different routes. These GPS files have unwanted data, missing data and noise in it which is pre-processed and data is cleansed. Routes are analysed using different attributes like number of turns taken, number of stops in the route, time taken etc. The best route between the given source and destination is computed.

2. Overview

GPS data is taken as input and each line is read, processed and knowledge is extracted from the data. The GPS files contains two types of data – GPRMC data and GPGGA data. The GPRMC data is used to compute different attributes for the route and GPGGA data is ignored because all the required information can be extracted from the GPRMC data. The validity of data is checked and anomalies and noise in data is removed. The co-ordinates from the data are read, converted and plotted on Google Earth using KML files.

The main objective is to find the best route from a given source to the destination in the sense that time taken using that particular route is minimum. The objective function constitutes the minimum time taken from source to destination and other parameters are considered to be regularized which will help to minimize time. Each parameter is given a weightage to normalize the cost function which further helps make sure that the parameters being regularized won't drastically change or affect the main objective function.

There are certain parameters that are regularized to help with minimizing the total time taken in the route. The number of stops in the route are computed. Stops in the route are generally due to stop signs, traffic signals and errands. A threshold of a low speed is set, below which the vehicle can be considered to be slowing down and the time for which it stays slow or stops is also recorded. Reducing these stops or slowing down also will help in getting a better route.

The number of turns that are taken in the route also affect the time taken for the route. Turns generally cause vehicles to slow down, increase time and also hinder fuel efficiency. Traffic signals are mostly present during turns and probability of free turns are less. A path with less turns will help maintain better speeds which in turn help with fuel efficiency and also reduce total time taken.

In Section 3, we will discuss the distribution of workload between the members of the project. In Secton 4, we will discuss the process we followed during the conversion of GPS to KML files and also the issues and anomalies we encountered during the conversion. In Section 5, we provide our cost function for the project. We define our objective function and regularization attributes and provide weightage to them. In Section 6, we explain our process to detect stops due to different situations. In Section 7, we explain our process to detect turns and also the process to determine whether it was a left turn or a right turn. Section 8 talks about the assimilation of the complete project. It will discuss the design process and the functions used in the project. In Section 9, we display some possible routes for the project and our best route. In Section 10, we display the hazard files for the best route. In

Section 11, we talk about the results of the project and the challenges we faced while designing the project. In Section 12, we provide a conclusion of the complete project and what we learnt from it.

3. Team Composition

Our team had two members and each of them worked on important aspects of the project. The following summarizes the contribution of each member in the project:

Mihir Shah – Worked on identifying the stops and recognizing and handling the different situations like traffic, traffic signals, etc due to which the stops were occurring. Understood and created pins to display the stops accurately in the visualization of the routes.

Abhinandan Desai – Worked on understanding how to detect the turns using longitude and latitude values. Understood how to read bearing angles. Worked on detecting the number of turns present in a particular path and distinguishing between right turns and left turns. Detected anomalies in the given GPS files and discarded them to further reduce unwanted data.

Teamwork – Worked together to understand what attributes were provided in the GPS data and how to read them and convert them to KML files. Visualised and manually checked all 173 KML files on Google Earth. Understood the required conversion, and the latitudes and longitudes were converted to fractional degrees from degrees and minutes format. Worked on data cleaning and pre-processing like identifying if there were some missing values or inappropriate values and used only required data, so that performance of the code is faster and storage is also efficient.

4. Converting GPS to KML files

There are 173 GPS files which get converted to KML files and these KML files are visualised using Google Earth and different routes and paths can be seen.

The GPS files needed to be stored properly and correctly referred through using a proper path when reading it into the code. Once, a GPS file is read, it is processed to read the attributes present in it. There are two types of data in it - $GPRMC data and $GGPGA data. We figured out that $GPRMC data is enough to provide us with important attributes which are:

- Time
- Speed
- Validity of data
- Latitudes and Longitudes
- Tracking angle
- Date
- Numeric Checksum

We checked the validity of each $GPRMC data line. If it is invalid, we discard it. We also check the number of attributes for each line of attributes, if they are not enough then that data is ignored. Any other data then $GPRMC is neither read nor saved, this helps in efficiency of performance and storage.

The latitude and longitude values in the GPS files are present in degrees and minutes format. This format cannot be used to plot them on Google Earth. The values are converted into fractional degrees format and these are stored as GPS co-ordinates. These GPS co-ordinates computed from each row are saved in a list in a latitude, longitude tuples.

Using ***simplekml*** module in python, a KML variable is created. This variable is used to set the altitude such that the variation in vehicle speed can be observed. The saved co-ordinates are put into this variable, the color scheme for the GPS co-ordinates plot is set along with the trail width. This variable is then used to save the data into a KML file. The generated KML file is then uploaded to Google Earth and the route can be visualised easily.

There are anomalies present in the GPS files. All the GPS files can be processed and are converted to KML files without any problem but there are GPS files which have noise present in it. When these GPS files are converted to KML files and plotted on Google Earth, we observe that there are files in which the vehicle does not move at all or moves in a very small space like a parking lot. The only required source and destinations are Professor Kinsman's home and Rochester Institute of Technology but there are files where the routes either start from other places or end up at other places. There are files which seem to be going through a permissible path but the path ends in between roads which is likely due to loss of data or missing data. Such data can be considered anomalies and don't need to be considered during calculation of cost function. The following screenshots show anomalies:

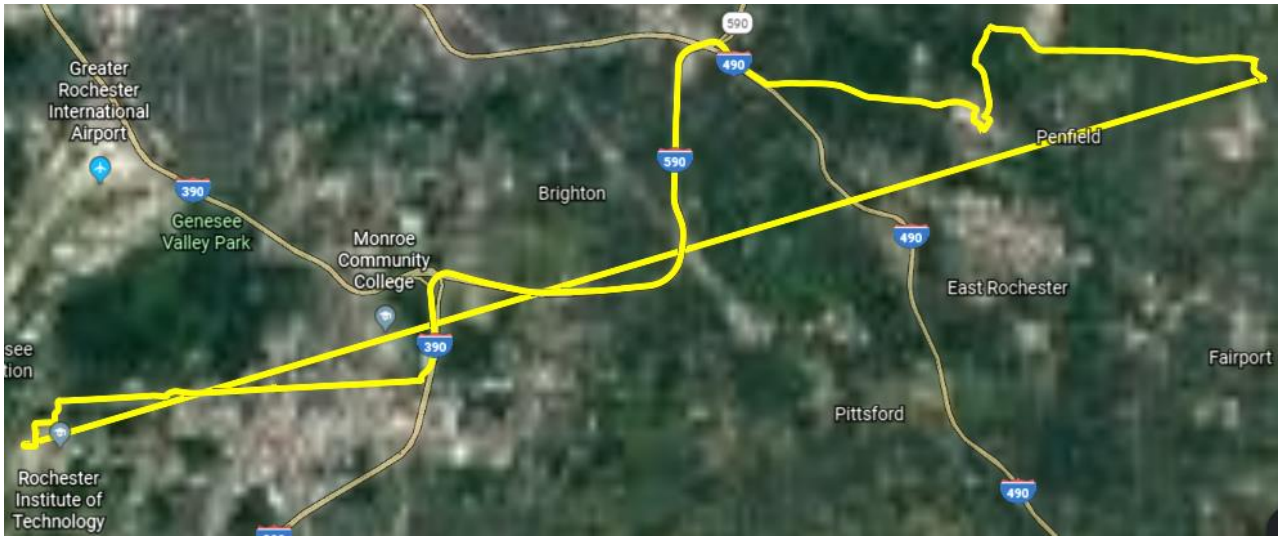


Figure 1: There is direct connection between source and destination although there is no clear route present there.

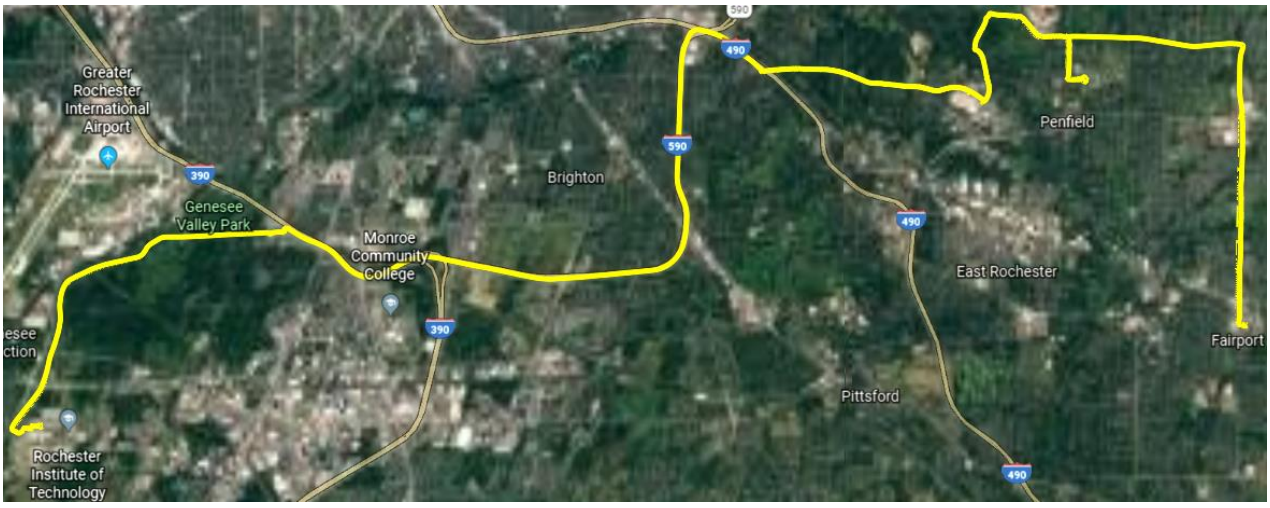


Figure 2: One endpoint is Rochester Institute of Technology but the other endpoint is in Fairport which is not the location of Professor Kinsman's home.

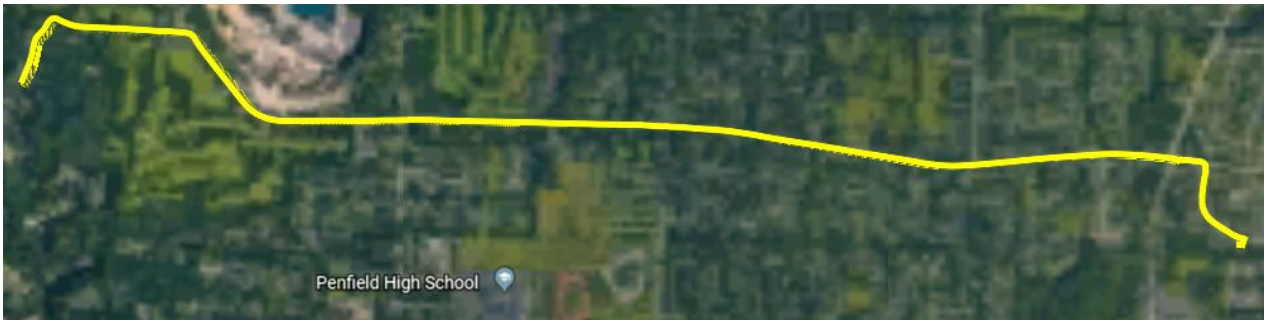


Figure 3: The route does start from Professor Kinsman's home but it ends in the middle of the route possibly due to loss of data.

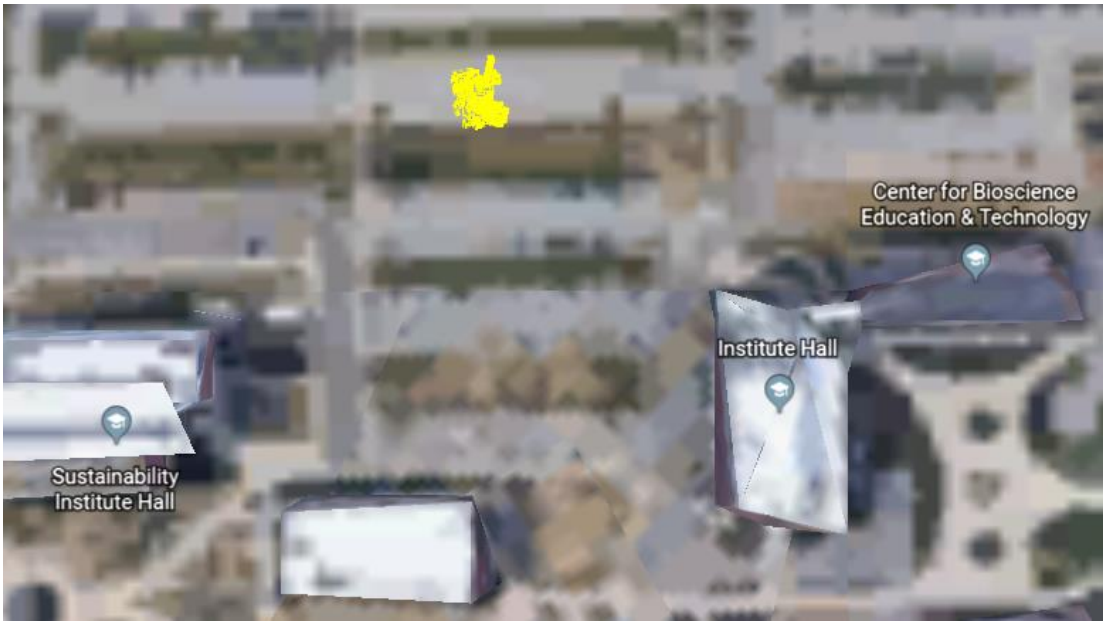


Figure 4: This shows the plot of a vehicle which either moved around in the parking lot or which was idle but due to error was generating GPS data.

5. Cost Function

After creating the KML files and plotting it on Google Earth we observed the different routes we found. We looked at number of turns in each route, we looked at the number of times the vehicle stopped on the route, we tried to estimate which route is longer by comparison and we looked at variation of speeds in the routes. There are certain parameters that we would like to minimize for the cost function. We have decided to give each of them a weightage while calculating the cost function. The objective function will be minimizing total time taken from source to destination

We have observed that Dr. Kinsman was very keen on minimizing the time spent in traffic, at stop signs, or at stop lights. So, while designing the cost function we will try minimize this stopping time more than other parameters. Hence this parameter will have the maximum weight. Similarly, we will give weightage to other parameters like speed of car, number of turns etc. The weightage assigned to each of the parameters are as follows in decreasing order:

Sr.No.	Parameters	Weightage given
A	The trip time to work from home and vice versa.	50%
B	The time spent in traffic, at stop signs, or at stop lights	15%
C	Number of left turns and right turns	15%
D	Total number of stops	10%
E	Maximum velocity	10%

Using the above table our cost function will be as follows:

COST FUNCTION = (0.5*A) + (0.15*B) + (0.15*C) + (0.1*D) + (0.1*E)

The weights that we have designated are based on our assumptions of how much a particular parameter will affect the cost function. The total weights will always add up to 1.

6. Stop Detection

To detect all the stops, present in a particular route, the most important attribute that is required is the speed of the vehicle. The distance travelled and the duration of the stops are the next important attributes for stop detection.

Firstly, we mark the point where speed drops **below 10 mph** i.e. our **threshold** consideration. This point marks that, there is a probability that the vehicle is going to come to a stop soon. We then mark the point where the speed of the vehicle goes **above 10 mph threshold** mark. Now that we have two points marked, we **check the distance** that the vehicle travelled between these two points. If the distance travelled is **below 0.09 miles**, then we are sure that the car had to stop between the marked points. If the distance is greater than 0.09 miles, then we consider that the car was moving slowly, maybe due to traffic.

After the speed and distance travelled conditions are satisfied, we **check the time** for which the vehicle was below 10 mph. We classify the stop reasons according to the time duration the vehicle had to stop and the classifications are as follows:

- **Stop Sign –** If the duration of the stop is **below 7 seconds**, then the vehicle had to slow down or come to halt because there was a stop sign present on the route. **Red pins** are used to mark and visualise these points in the route.
- **Traffic Signals –** If the duration of the stop is **between 7 seconds to 50 seconds**, then the most probable reason is that the vehicle was waiting at a stop signal in the route and started moving when the signal turned green. **Green pins** have been used to mark and visualise these points in the route.
- **Errands –** if the duration of the stop is **greater than 50 seconds**, then the most probable reason to stop is that there was an errand that had to be completed like eat breakfast, shopping, return a book to the library etc. **Orange pins** are used to mark and visualise these points in the route.

After plotting these pins on the route, we observed that there are few cases in which the vehicle stops for **more than 7 seconds** at a **stop sign** and these points are marked under **traffic signals**. As it is seen rarely, we do not increase the threshold from 7 seconds to a higher value.

7. Turn Detection

To detect all the turns in a particular route the main attribute to consider is the tracking angle of a point. To analyze if a turn has been taken by the vehicle, we check the **difference** between the **tracking angle** between two points. The second point is selected by **skipping 30** GPS co-ordinates which we have considered as our window to detect if a turn is taken. **Purple pins** have been used to mark turns on the KML file plots.

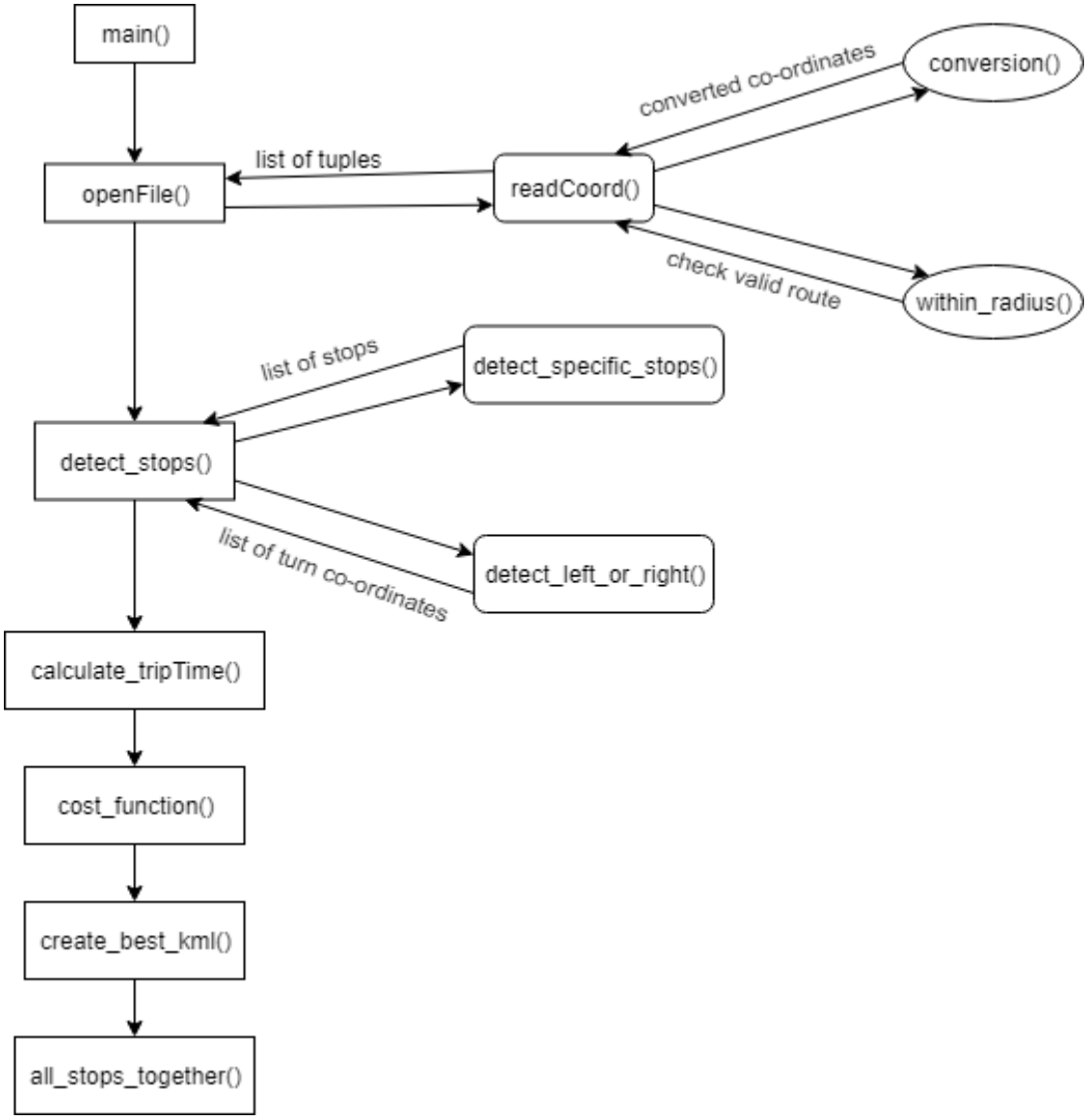
The calculated difference between the tracking angles are checked to classify if the vehicle took a left or right turn. Different conditions are used to determine this:

- **Left Turn –** If the current tracking angle is **less than** the end point tracking angle then the difference will be **negative** and if the difference is between **60 to 120 degrees** then we determine that turn as a **left turn**.
- **Right Turn -** If the current tracking angle is **greater than** the end point tracking angle then the difference will be **positive** and if the difference is between **60 to 120 degrees** then we determine that turn as a **right turn**.
- **Special Case (Angle Reset) –** The measurements of angles **range** from **0 to 360 degrees** and so a special case arises due to this.
  For example, let's say that the current tracking angle is **350 degrees** and the end point tracking angle is **55 degrees**. This actually happened because the angle got **reset** after the vehicle reached **360 degrees** at a co-ordinate and then it **turned 55 degrees** further which classifies as a turn because the actual angle difference is **65 degrees** but the **calculated difference** will be **295 degrees** and this will not be considered as a turn rather it will be skipped as maybe it might mean that a **U-turn** happened. So, **to correct** such an error we **add 360 degrees** to the end point tracking angle and then calculated difference falls under a range of a turn and it is classified as a turn.

In general, we also observed that while turning, the vehicle might come to a stop and so such stops are also taken into consideration and it is useful as reducing turns will also help reduce stops in the route.

8. Assimilation of the Project

We used the Python programming language to implement the whole project. We structured the implementation into functions that deal with different aspects of the project. Each function handles a subproblem and returns a generated result to help with the final plotting of the routes.
Here is our workflow of the complete project:



Workflow Diagram

Each function in the implemented code deals with a subsection of the project. The following list explains each of the function:

- **openFile() –** This function is handles the first aspect of the project i.e. taking in the GPS files from a proper folder at a given path. A loop is applied to process each file and each row or line of data in a particular file is read, split and then the complete file is sent to the **readCoord()** function which deals with the attributes of the file it received.

- **readCoord() -** This function deals with processing the GPS file that was read and passed to the function. The data that is received by the function has different attributes or features that will help create the KML file. We figured out that the $GPRMC line of data has all the required attributes that are needed for further implementation. We understood that the latitudes and longitudes are in degrees and minutes format which need to be converted to fractional

degrees to plot on Google Earth. The following algorithm will explain how we deal with processing the data:

ALGORITHM:

if it is $GPRMC data
else continue
    if it is valid then 'A' is present
    else 'V' is present which is invalid so continue
        if all the attributes are present (length of the line)
        else continue
            process the line
            if longitude value is 'W' then '-1' is multiplied
                convert value to fractional degrees
                store coords_value →long_value*(-1)
            else longitude value is 'E' then value will be positive
                convert value to fractional degrees
                store coords_value →long_value
            if latitude value is 'S' then '-1' is multiplied
                convert value to fractional degrees
                store coords_value →lat_value*(-1)
            else latitude value is 'N' then value will be positive
                convert value to fractional degrees
                store coords_value →lat_value

create KML variable → kml
kml.coords → coords_value (latitude and longitude values for GPS plot)
kml.color → yellow (color for GPS line in plot)
kml.width → 4 (thickness of GPS line in plot)
save KML file.

- **conversion() –** This function carries out a **format conversion** for the latitude and longitude values. The GPS file has latitude and longitude values in the form of **degrees and minutes** format. This format cannot be used to plot the values on Google Earth. This function helps convert these values to a **fractional degrees** format which can then be accurately plotted. Following conversion steps are carried out for the final value:

  Step 1:
  Original format → original_value (degrees*100 + minutes)
  Divide original_value by 100 and split it on the decimal
  Store the two values → value_beforeDecimal and value_afterDecimal

  Step 2:
  Subtract value_beforeDecimal*100 from original_value
  This gives the minute fragment of the original_value → minute_fragment

  Step 3:
  Add value_beforeDecimal and (minute_fragment/60) → final_value
  Then return the **final_value** which is now in **fractional degrees format**

- **within_radius() –** Two base co-ordinate pairs are set in this function. One of the co-ordinate pair is the latitude-longitude pair for Rochester Institute of Technology (RIT) and the other latitude-longitude pair is for home:
    - **RIT latitude-longitude pair:** -77.68016333333334, 43.085848333333324
    - **Home latitude-longitude pair:** -77.43771166666667, 43.13834333333324

  After all the co-ordinate tuples are saved for a GPS file in **readCoords()**, the first and the last latitude-longitude pair is sent to this function. It is checked if the input pairs lie within a **radius of 175 metres of haversine distance** from the base pairs, as we assume that radius as the **geofence** for the source and destination. If the input pairs both satisfy the condition only then the file is further checked for stop signs and number of turns else it is considered an **anomaly.**

- **cost_function() –** The aim of the function is to calculate the **final cost function** of the project in order to find out the **best route** from the given source and destination. The cost function defined in **Section 5** is calculated in the function by using all the objective function and the regularization data that the function received. The **minimum cost function value** provides the best route in this project.

- **detect_specific_stops() –** This function helps categorize of classify the situations for which the vehicle has stopped. Firstly, the speed in the GPS data is present in **knots**, we convert the speed to **miles per hour (mph)** by multiplying the values by **1.1508** because it is one of the most common speed format used for vehicle speeds. The classification of the stops are same as explained in **Section 6** of the report. A separate count is maintained for different situations and the total time is also maintained for the stop. These values are returned to be used further for the cost function calculation.

- **detect_left_or_right() –** The main aim of this function is to detect **turns** in the route. Different conditions are used to distinguish between left turn and right turns. A condition is also laid down for a special case in turns. The conditions are same as explained in **Section 7** of the report. The co-ordinates of the turns are returned to be used further for the calculation of the cost function.

- **calculated_tripTime() –** This function's aim is to calculate the **objective function** of the project. It calculates the **trip time taken** for a particular route. It is calculated by the **difference** between the **start time** at the source and **end time** at the destination. It also takes in all the regularization data that will be required to calculate the final cost function. At the end, it calls the **cost_function()** sending all the acquired regularization data and the objective function data.

- **detect_stops() -** The main aim of this function is to collect the total number of stops due to stops signs, traffic lights, turns, etc. and the time spent at these stops. The function also calls two other functions – **detect_left_or_right()** and **detect_specific_stops()** which contribute in the requirements to calculate the total stops. All the collected is passed on to next function in the order of the workflow – **calculated_tripTime()**.
- **create_best_kml() –** This function creates the **KML files** for a GPS file given to it with the data of stops and turns. It firstly, converts the GPS file to a KML file. Then, using the stops and turn co-ordinate points data, it creates a KML file for each type of the stop situation and turns

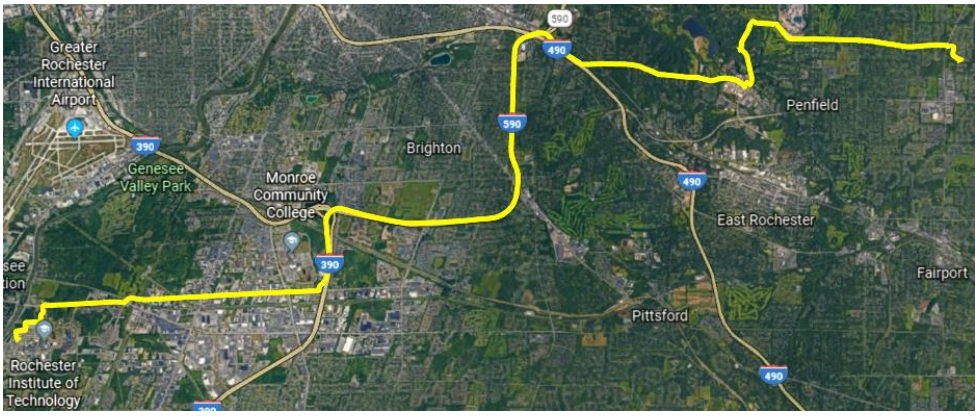
which **visualise** these points on Google Earth using different **colored pins**. This function is called in the **main()** function to mark the pins onto the KML file of the best route having the minimum cost function.

- **all_stops_together() –** This function aims to combine all the stops and turns pins together onto one KML file. This then gives us only two KML files – one with the best route and one with all the pins on stops and turns in that particular route. This function is called in the **main()** function.

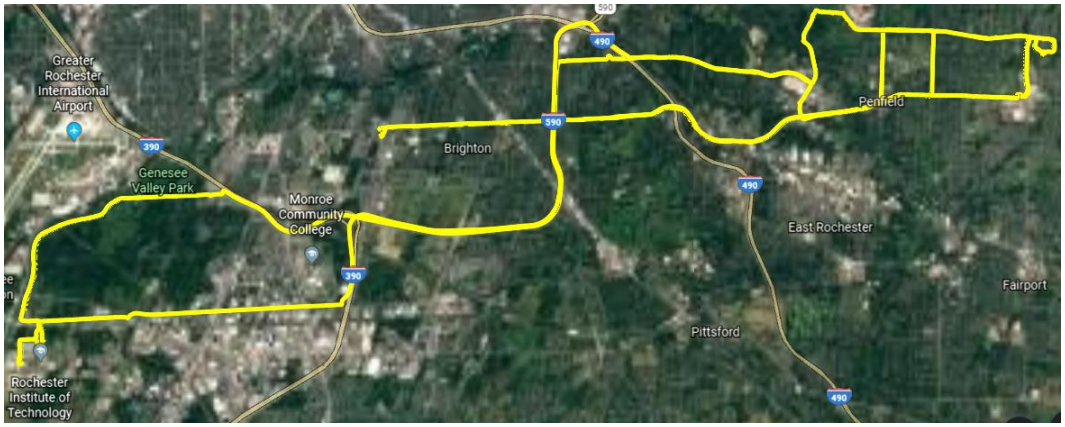
To store the values that are computed by processing the GPS files, we are using Python **lists** to store the values in-order. We are using **tuples** to save latitude, longitude and speed values. These three values help plot the KML file and create a GPS track line on Google Earth with speed variations. In addition to this, **tuples** ensure that the values become immutable and can be extracted easily to plot the KML files. A **Python dictionary** is maintained that stores different values that will be required to calculate the cost function. This dictionary also helps in plotting the important pins for the best route in the project. The collection of different **lists** also acts as an intermediate database. Different lists have been defined to store various attribute values and these lists can be accessed to find that particular data values.

9. Detected Routes – KML Files

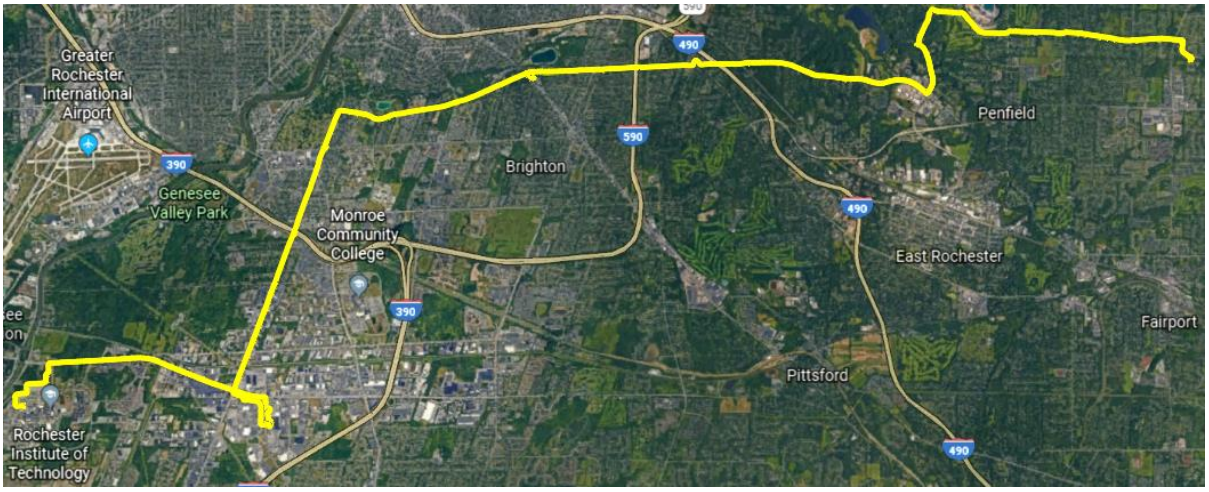
The following figures show examples of some routes that were detected after generating KML Files and plotted on Google Earth:



Route 1

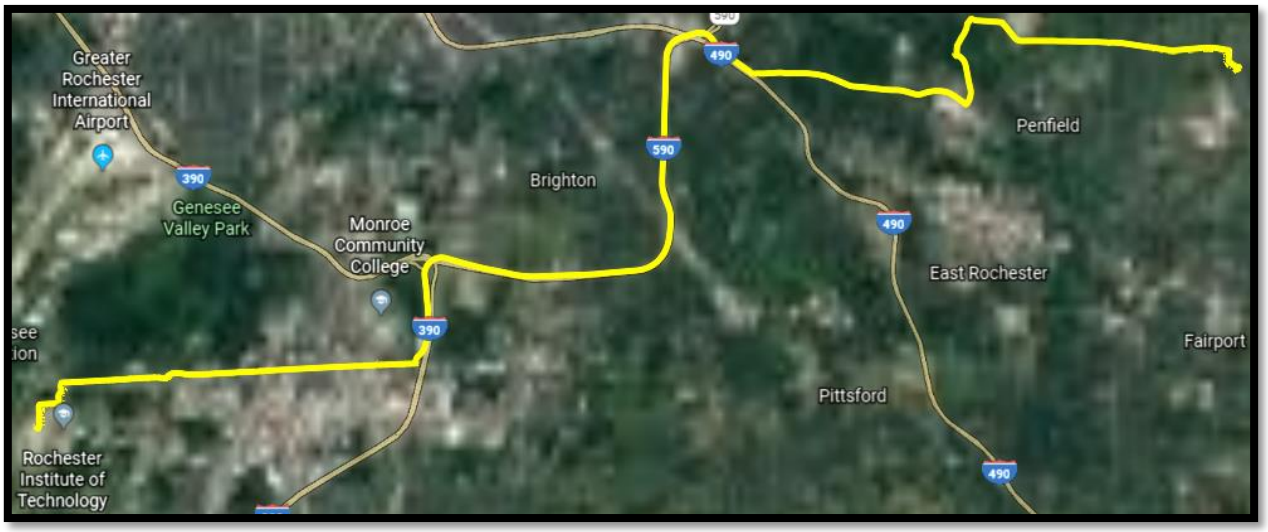


Route 2



Route 3

According to our cost function the **best** calculated route is:



BEST ROUTE

File with minimum Cost Function:

2019_05_04__1230_23.txt

Cost Function for Best Route:

Cost Function = 42.6066

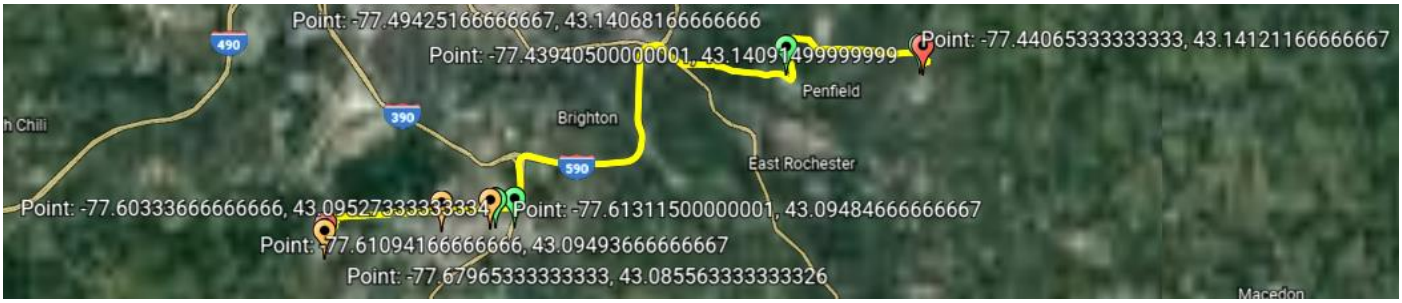
10. Hazard File Results

The color scheme for stops and turns is as follows:

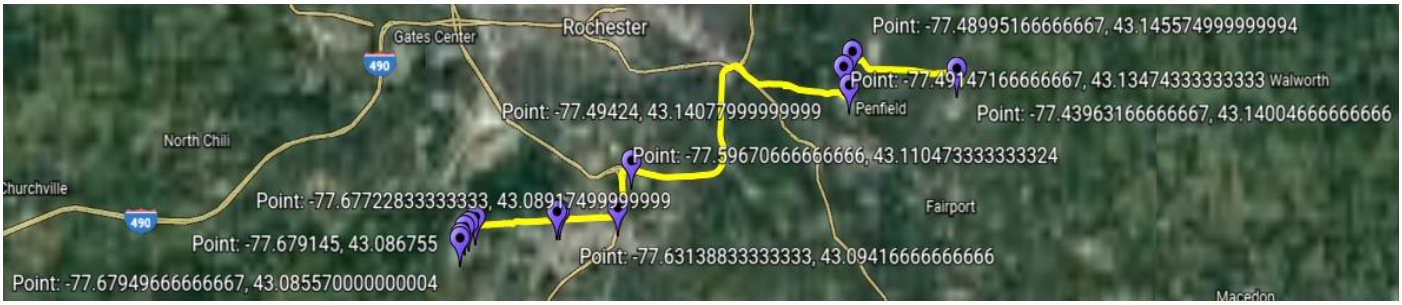
Traffic Signal stops Stop Signs Errand Stops Turns

Here we show two KML files –

- Best route KML file with **all stops**:



- Best route KML file with **all turns** –



Descriptions of the Hazards file generated by our code:

- **Group10_GPSProject_errands.kml –** This KML file contains all the pins that represent stops due to errands.
- **Group10_GPSProject_stop_signs.kml –** This KML file contains all the pins that represent stops due to stop signs in the route.
- **Group10_GPSProject_traffic_signals.kml –** This KML file contains all the pins that represent stops due to traffic signals on the route.
- **Group10_GPSProject_left_right.kml –** This KML file contains all the pins that represent turns in the route.
- **Group10_GPS_Hazards.kml –** This KML file contains all the pins that represent all stops and all the turns present in the route together.

11. Discussion

The project is able to calculate the best route from a given source to a given destination using different attributes by developing a cost function and ranking the attributes and providing them weights accordingly. The objective function of the project is the total trip time of the route. By reducing the time spent at stops, reducing the number of turns in the route, reducing the number of stops and reducing the maximum velocity helped in shaping the best cost function.

Data cleaning and preparation took a lot of time and effort. Firstly, the data in the GPS files were not in the required format. Speed had to be converted from knots to miles per hour (mph). The GPS co-ordinates had to be converted into fractional degrees. After this, missing data and noise had to be recognized and removed. There were anomalies also present in the GPS files where the source and the destination were not the required ones. Such files had to be discarded or ignored.

Developing the cost function also took a lot of time. It had to be updated many times as we were able to compute or extract more information from the files. The cost function components had to be given weightage properly to ensure that the regularization components don't affect the objective function such that the best route is changes completely.

Understanding how to read bearing angle between two points also took a lot of time and effort. Even after understanding how bearing angle works, we had to make sure that we don't miss out on any turns in the route. There was a special case that needed to be handled where the angle resets after the tracking angle goes higher than 359 degrees. Understanding and handling this case was challenging.

12. Conclusion

This is a challenging but an informative project. We were able to apply the knowledge and understanding of developing a cost function to return accurate and effective results. We also learnt and gained new information while developing this project. We learnt to process GPS file data and then convert it to create KML files. We were able to plot and visualise these KML files. We were able to observe and understand the plots, that helped us to improve our cost function. We learnt about haversine distance and how it can be used to implement the concept of Geofencing and detect anomalies and remove them and keep only relevant data. We learnt about bearing angles and how to read them. We learnt how to use the tracking angle present in the $GPRM data in order to determine turns in the routes. We were able to see the implementation of our cost function and find out the best route.

The project can be used for commercial application to find out the best route between a source and destination. It will be able to calculate a route where the vehicle does not need to take many turns, does not need to stop much and also will be able to maintain an efficient speed throughout the route and reach the destination in less time, safely and smoothly.