GPS and all it’s mysterious data

Asmita Hari, Luke Batchelder, and Pawan Shetty

Abstract

We created code to extract the latitude, longitude, and speed of a given vehicle at several points from the raw GPS data that was given to us. This data was not only saved into kml files but was processed to eliminate certain points while also marking points as turns or stops. To do this we made use of proven data cleaning techniques while also careful use of meaning imparted by the data given over time. All of these elements were run through n-fold cross-validation testing to determine a single best result. The point of this system was that many routes worth of GPS data could be overlaid in one large kml file giving us a comprehensive overview of what turns and stops exist in the area (as well as the generally acceptable speeds along certain routes). We overlaid this data onto google earth to evaluate the effectiveness of our extraction and altered our hyperparameters accordingly (such results will be visible in the appendix). The result was a system that could effectively determine routes found from gps data and could use this data to determine what stops and turns existed in a given environment. While the stop and turn mapping was imperfect it was found that a reasonable balance could be achieved with the right hyperparameters giving a system that could eliminate much of the noise while providing high accuracy results.

Overview:

Much of this work was the process of creating an ensemble system using the basics of big data analysis. This work in turn will provide a demonstration of how to convert raw data to readable input, clean it as needed, and be able to derive valuable information from the results. In addition, this work will be using the raw data to answer 2 questions that would have notable academic and commercial merit. Specifically can we meaningfully derive traffic signage from a number of routes taken by a few users? If successful it would give a system that could provide a dynamic means of mapping roads quickly.

The project was performed with a general research phase where the best tools for the job were found and the core pipeline was theorized. Once developed this pipeline was tested on small datasets for basic functionality. After we had reasonable results we ran a training regiment on all of our data and examined our results to determine what fixes need to be made. These fixes were then applied over a training regime in which different values that could produce different results were made and used.

On this project, all members had the following roles,

Luke Batchelder: primary writer and alteration of hyperparameters dealing with turning, stopping, and whether the car was at a stopped point.

Asmita: research and development for handling many types of GPS data,testing hyperparameters and kml to google earth translation.

Pawan: code refactors for a major chunk of the program, fine-tuning the program, hyperparameter testing, and kml to google earth translation.
All members worked on the development of the core pipeline from GPS to kml data and we hope that this report will prove instructive as to the work performed and the capabilities of the code on the assigned datasets.

This report will have the following sections:

1.) Challenges and requirements
2.) Systems to provide requirements
3.) Code outline
4.) Testing and results
5.) General conclusions
6.) Appendix of images and outputs
7.) References

Challenges and requirements
There are a number of errors that you can always find in data such as outliers, missing values, inconsistent values, and duplicate records. The nature of the Arduino gathering process meant that we had no shortage of any of these. However, the nature of the data meant that there were hard limits to what could be classified as a reasonable next point in the sequence as data that was too far away from previous points or if speed or angle was too differentiated. As a result tracking a number of previous points gave us important information on if our current point was part of a false record. Overall we had the following fixes to the proposed problems.

Issue	Fix
If the vehicle is parked, you do not need multiple data points at that same location	If is a vehicle is parked do not record the points
If the vehicle is traveling in a straight line at a smooth speed Minimize points	Compare the last few points, if all are the same speed we should limit the number of future points at the same speed we are taking in.
The Arduino sometimes burps and writes two GPS sentences to the same line of the data file.	The lat and long are very precise with 6 points after the decimal as shown in [4]. This means we are not likely two points that were gathered in the same way
The Arduino sometimes loses its mind and starts recording GPS values that jump all over the place. This especially happens if it loses connection to the antenna.	Needed to use the previous points to determine if we just moved farther than reasonably possible [3] showed that the worst-case scenario for a difference between points was 30 minutes and that the highest

You may need to ignore meaningless junk.	speed the user would be going would be 100 mph so any difference in the distance should be ignored.
When the trip first starts up, the GPS device is not moving. Do not worry about the data points when the vehicle has not started moving yet.	Needed specialized stop detectors that could ensure that unneeded points were not recorded when the vehicle was no longer moving.
when the vehicle parks, the GPS device stops moving. We must make sure these points are not included as well.	Needed function ensuring that if the car was still stopped after being stopped the point wasn't recorded.
Finding speed and heading	Thankfully these were included on each data line
You need to convert the GPS sentences	Need to use precise calls of the csv reader to interpret what lines should go where and why
Generate a KML file which can be examined in Google Earth.	Need to generate a file that will record the base coordinates in a path file and the stop's and turns as separate colored points
Storage, processing all gps data so they could be added to one larger map	Needed to be able to add to one larger kml file for each run to retain one larger total kml file
Stop detection	Mix of time window and speed to capture when a car had encountered a traffic regulated stop[1] We would also use agglomerative clustering to ensure that many stop points in the same small space only reported back as the one to ensure clean data.
Stop detection at non-stop locations	Because users would need to generally always stop for traffic mandated stops vs stopping on their own volition the stops would occur at those sites were dramatically less than normal. This mean that we could eliminate clusters that had a sufficiently small number of stops in them.
Detect turns	Similar system to stops would be used with clustering used to cut down on unneeded points that were adjacent with each other followed by a system that could eliminate clusters lacking points. For the turns themselves, a similar system to detecting stops was used with examining a change in degrees over a sufficient number of points.

This was in addition to a number of data cleaning steps we needed to perform to make sure the data we were getting was even helpful enough to tell us that we were performing these fixes correctly.

Design Elements:

Our fixes could, in totality, be summed up to four major need functionality pieces, Data Checking, Temporal Data evaluation, Clustering and data storage.
Because of the available memory factor in this program we would need to do many of these processes in assembly line fashion with the idea of taking raw input and generating refined labeled output.

Data Checking:

1) Irrelevant data: dealt with through data filtering, Points that were not of the type GPGGA and GPRMC.
2) Duplicates: chances of encountering two points that had equal latitude and longitude at up to 6 decimal places was highly unlikely so we could eliminate any data that had the same latitude and longitude of the previous point
3) Syntax errors: Missing data was not uncommon, Sometimes this did not impide our efforts as things like the missing angle in GPRMC could be accounted for via assuming no meaningful change in angle had taken place, same went with speed. Other times missing latitude and longitude would be a sign that a particular data point was too corrupt to use. Given the abundance of singular points and that more important elements like turns and stops would be accounted for via a pattern of data found over many points. It was decided that scraping these bad points would be the most efficient way to get the data we needed.
4) Anomalous Data: These are the points which have a higher jump which looks like the car jumps from one side to the other side, or across an ocean. To overcome this, while reading the data difference between the current and previous latitude and longitude were found. If the jump was greater than a certain threshold these values were ignored.

Temporal Data Evaluation:

This ended up being our most complex element by far and required the most testing to the point where it’s best to break the discussion of it into 3 sub functions with the class storing data needed for it’s execution. These were stop detection, turn detection and same speed detection which were all performed in that order. To make each function work we needed to incorporate a data storage object for the last 10 points with points themselves being an object of speed, time, and degree where possible (as some points did not have all of this data). Each one of these use hyperparameters tuned to the larger example dataset of the 117 files (see training section for details)

Stop Detection:

As found from [1] determining stop points required a mix of time and speed evaluation. For this functionality a special container would be created when a point falling below a given speed threshold was found. If points below said threshold continued to be found for a time span greater than minimum stop time and a point above to the given speed threshold was found below the maximum stop time was found (signaling that user was in fact at some traffic stop had was not having the vehicle wait while errands were being run) the first point in the special container would be added to a stop point container.

Turn Detection:

As seen in [2] turns are determined by a specific change in angle over time. For us this could be determined by evaluating all points in the dataset and finding if the total change in angle had been more than a set value for a given series of points. If the change in angle was too small (aka too negative) it would be clear that the vehicle had taken a left turn and the point at the beginning of the 10 point list would be added to a container holding our left points, same went for our right points but with the net change being positive. Because not all such turns had a requirement to stop before the turn could be made there were some cases where this method could find particularity sharp turns in a road to be the equivalent of a left or right turn however given that this data would seem important enough to the average driver anyway that it should be retained provided by our code.

Same speed Detection:

If it was determined that a given point shouldn't be a vitally important one like a turn or stop point we need to consider if it is saying anything important about the dataset in general aka if it is indicative of some demonstrable change in angle or momentum. If it is not it needs to be removed. This would be done in a way similar to turn detection as points over the extent of the container were observed to find if the car had been generally going the same angle and same pace for long enough. If it had it would begin to eliminate every other point of the path to ensure minimal unnecessary use of data.

Clustering:

Due to stop and turn data being the product of many routes put together there were a lot of points on the map. Many scattered points on a map can overwhelm a viewer looking for a simple narrative. How can we reduce the size of a data set down to a smaller set of spatially representative points? For this we used agglomeration from the sklearn package. We used a distance threshold of 0.0009. Agglomeration returns a list of labels. We then defined a custom function which finds the medoid and returns a list.

Results for All the stops and turns without agglomeration are visible in the appendix under fig [3] and fig [4]. From the image we observed that after combining all the stops and turns we got a lot of repeated pins on the map. To avoid this we decided to agglomerate the stops and turns. While the hyperparameter for the agglomerative clustering distance had to be tuned this gave us allowed us to have a program with broad enough definitions of stops and turns to find them while not creating additional ones.

Data Storage:
The process of creating one path from a kml file was not a sufficiently memory intensive process to require any special consideration outside of, determine what points are effective and record them to a KML File. It is possible that a sufficiently large number of paths could require intermediate csv files to store the container elements every 200 gps files or so but given that the current data we have access is comfortably readable by our machines we prioritized minimizing the total amount of storage that would be used every run while also decreasing the total runtime required. We succeeded in doing this though the use of dictionary objects to ensure that duplicate data was not being stored limiting the total number of points we needed to record while also making use of the data pruning tactics discussed under data cleaning and same speed detection to ensure only important points were kept.

Overall our system used a mostly sentinel object pattern to mold the data into usable points with a global variables system used to test our system with a number of different possible variables until we generated data that was effective

CODE OVERVIEW:
Main:
Determines if the code will be creating a path or if it will be reading a series of paths to determine turns and stops. It calls getKMLPath() or getStopsandTurns based on user input (note in both cases the file/files used need to be in a predetermined folder containing only the files needed.

If user chooses choice a then getKMLPath() is called:

Uses the filename provided by the users and reads the gps data pre-process the data by removing duplicate latitude and longitude values, eliminating latitude and longitude values with a bigger jump, throwing out the lines which give error and finally returns a list of latitude and longitude which is then written to a kml file called the FinalPath.kml.

If user choose choice b then getStopsandTurns () is called:

This method does the following:

1) Asks the user to input the directory path.
2) Reads in all the text files from the given directory process each line gives the latitude, longitude,time, turn data(which says if the given point is a right, turn or left turn) if the

line is of type GPGGA. If the type of the line is GPRMC then we find the longitude, latitude, time,speed and angle.

3) After finding the latitude and longitude, following checks are performed:

    a. CheckZero: Given two latitude and longitude points, this method finds the absolute difference between the latitudes and longitudes. Using these differences we check if the sum of squares of difference is less than max change which is 0.5 and if the sum of the difference is greater than min change which is 0.000005 then we return true.

    b. Stop Point:
    The assumptions to find stop points are:
        - If the current speed is less than stop velocity of 0.005 then the given point is a stop point.
        - Otherwise, we use a list of stop points and check if the time difference between current stop time and previous stop is less than 10s and greater than 120s, then the given point is a stop point.

    c. Turn Points: Evaluating all points in the dataset and finding if the total change in angle had been more than a set value for a given series of points. If the change in angle was too small (aka too negative) it would be clear that the vehicle had taken a left turn and the point at the beginning of the 10 point list would be added to a container holding our left points, same went for our right points but with the net change being positive

    d. After finding the speed, right turn and left turn then add any points that do not satisfy velocity to the coordinate list.

4) This process is continued for all the files in the directory.
5) Finally all the stops, left and right turns and all other coordinates are combined into different lists.
6) These combined lists are processed by agglomerating the stop points, left turns, right turns which gives a merged new list.
7) We then start creating the kml file using the new list by
    - loop through stop points and add a pin on the kml with the name stop and a color of red.
    - loop through left turns and add a pin with the name left and color of yellow.
    - Loop through right turns and add a pin with the name right and color of green.
    - Lastly adding in all other coordinates to the kml file
8) Finally the kml file is created with the name FinalStopsAndTurnsFile.kml.

Results:

Fig [1] Results of a Path File for 2019_09_12__141120_gps_file_PNFLD_to_RIT.txt.

Fig [2] Screenshot of a kml file generated for a Path file.

Fig [3] as depicts an example of a full extraction of stops and turns based on a sufficiently large set of gps data.

Fig [4] Screenshot of the Stops and Turns after agglomeration.

Fig [5] Kml file for the Stops and Turns after agglomeration .

Testing For Ideal Hyperparameters

While our system could effectively determine routes taken by the user the process of creating impactful long term data still required specific thresholds to classify turns and stops. Because our data was inherently unlabeled we had to tune it to create the data that would be the most helpful to the onlooker. Overall we had the following hyper parameters that saw adjustment Stop time, Max Distance from algomerated Clusters, Db min stop neighbors, Minimum turn angle, and stop Velocity. While we had other hyperparameters like max and min change that were used to judge if we had points that were anomalous these did not require changes as they served a more general purpose. Instead these hyper parameters could significantly change our data with small changes. Before tuning these results were often unreliable Fig 7 but through tuning we could deal with edge cases to create better results as seen in Fig 11 compared to Fig 10.

Conclusion:

Overall our answer to our main question of, Can we meaningfully derive traffic signage from a number of routes taken by a few users was: we can meaningfully derive traffic signage followed by that one user but only so long as said user has habits that could be considered different from the standard traffic that were followed to a similar or greater degree or in other words. You can determine the traffic signage from one or more users that they follow but you will also get their personal stops as false positives. This work proved inconclusive on what the ratio of false positives to true positives were but it would be interesting future work

It is clear to see why google has had the greatest success with pulling data from many users at once as it has a large number of paths from users that have to obey global landmarks would produce the best results. With a more limited number of users personal preferences can cause clusters of stops or turns that can often only be removed if you are willing to reduce the true positive rate of the classifier to do so by cutting rarely used stops or turns. However even with these hurdles this code will serve as an effective foundation for more comprehensive work on unlabeled learning for traffic data.

As said at the beginning this was all a team effort and this was an interesting opportunity to work on a big data project with code and data spread between a number of different users. Interestingly while much work was done though creating code by different users and sharing over github the nature of complex tuning practices and a number of code revisions lead to one of our better tenquines being the use of remote access to other teammates computers to program as a team. This is not to say that this work should be considered the final word on the system. While the scope of this project did not include the use of pre-existing databases of stops we still like to aim to perform more comprehensive work in future with labeled data. This would allow us to explore the use of a ROC curve of correct vs incorrect turns and stops which would prove just as if not more insightful than our visalations. However our current configurations still give a good general picture of what drivers that generally use the routes provided would have to look out for.

Appendix:

Result Images:

Fig [1] example path

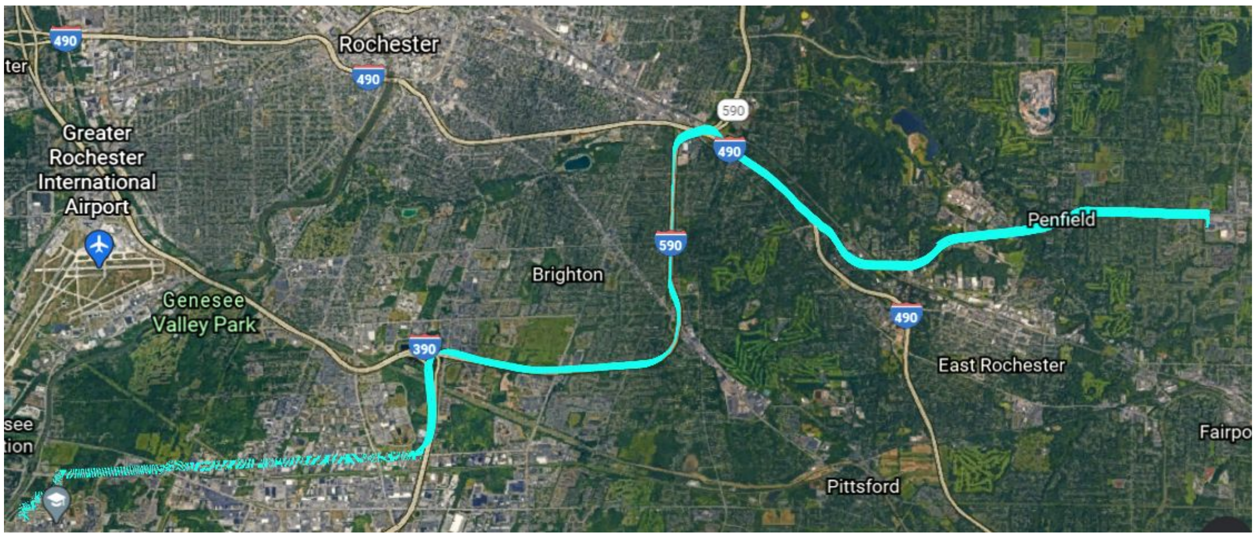
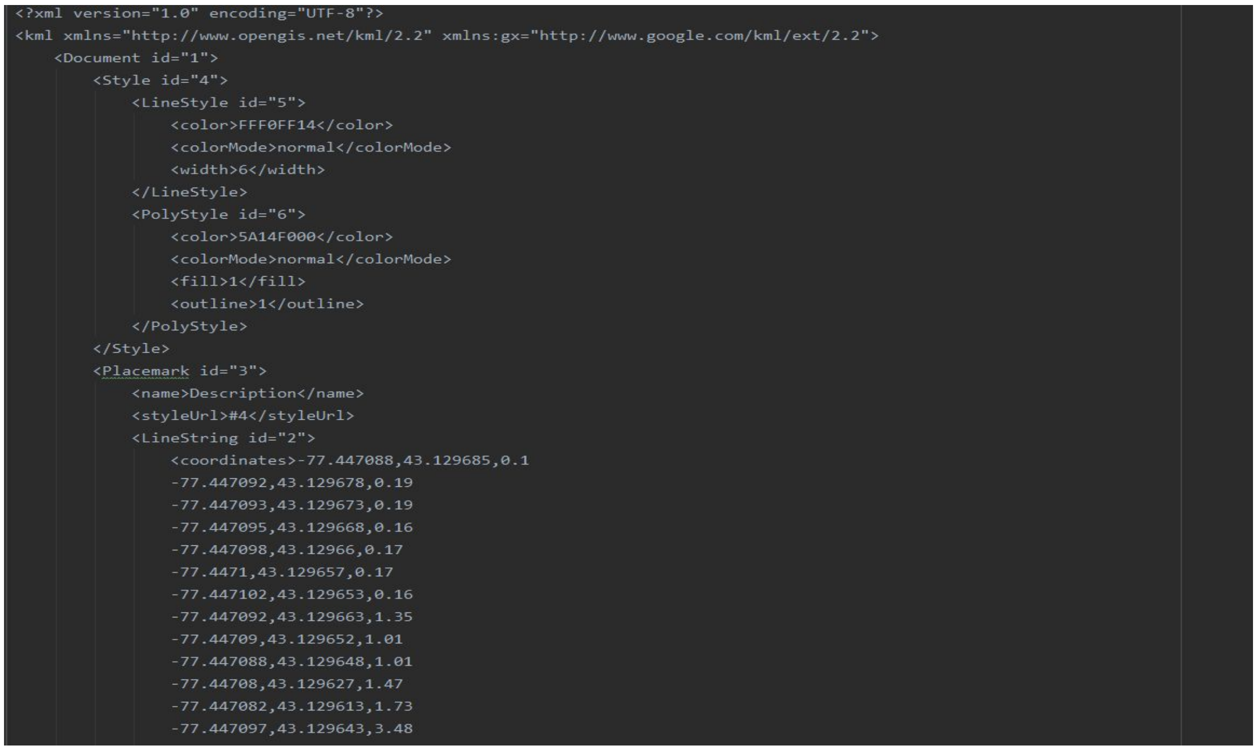


Fig [2] KML File for a path



```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2">
    <Document id="1">
        <Style id="4">
            <LineStyle id="5">
                <color>FFF0FF14</color>
                <colorMode>normal</colorMode>
                <width>6</width>
            </LineStyle>
            <PolyStyle id="6">
                <color>5A14F000</color>
                <colorMode>normal</colorMode>
                <fill>1</fill>
                <outline>1</outline>
            </PolyStyle>
        </Style>
        <Placemark id="3">
            <name>Description</name>
            <styleUrl>#4</styleUrl>
            <LineString id="2">
                <coordinates>-77.447088,43.129685,0.1
                -77.447092,43.129678,0.19
                -77.447093,43.129673,0.19
                -77.447095,43.129668,0.16
                -77.447098,43.12966,0.17
                -77.4471,43.129657,0.17
                -77.447102,43.129653,0.16
                -77.447092,43.129663,1.35
                -77.44709,43.129652,1.01
                -77.447088,43.129648,1.01
                -77.44708,43.129627,1.47
                -77.447082,43.129613,1.73
                -77.447097,43.129643,3.48
```

Fig [3] depicts an example of a full extraction of stops and turns based on a sufficiently large set of gps data which is before agglomeration

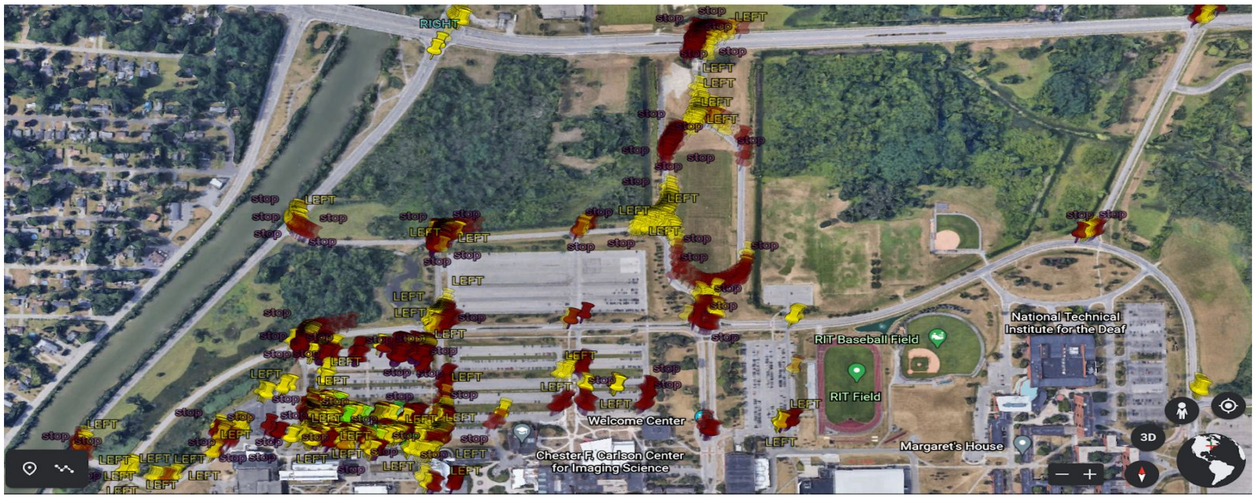

RIGHT
stop
LEFT
National Technical Institute for the Deaf
RIT Baseball Field
RIT Field
Welcome Center
Chester F. Carlson Center for Imaging Science
Margaret's House
3D

Fig [4] Screenshot of Stops and Turns after agglomeration

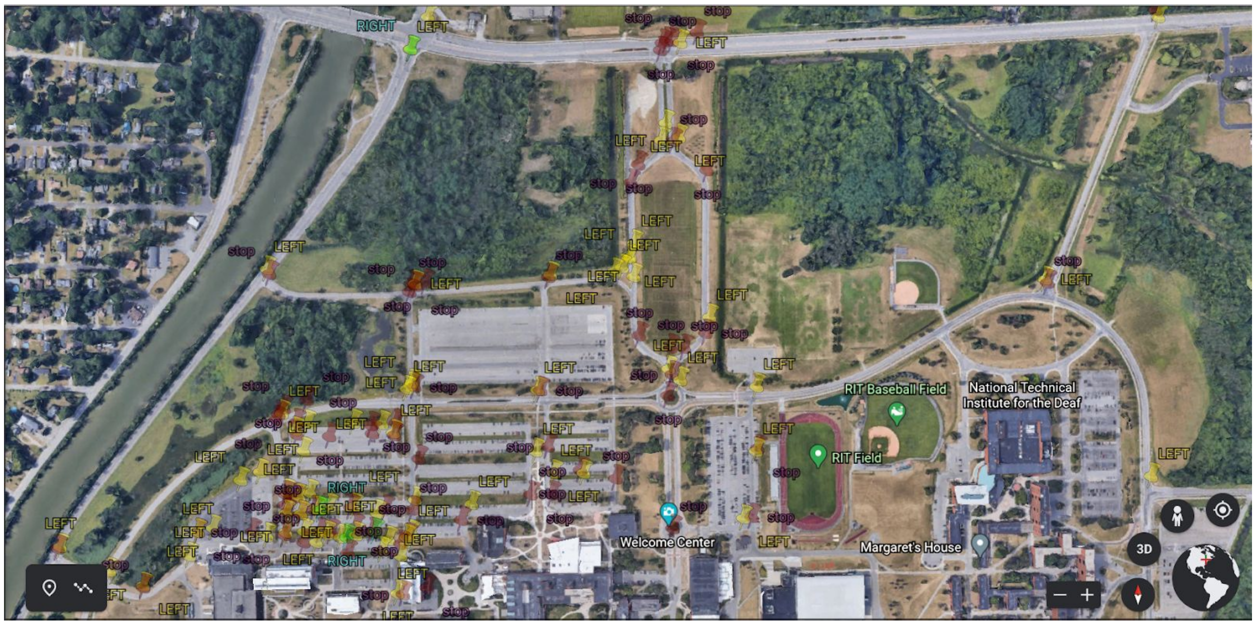
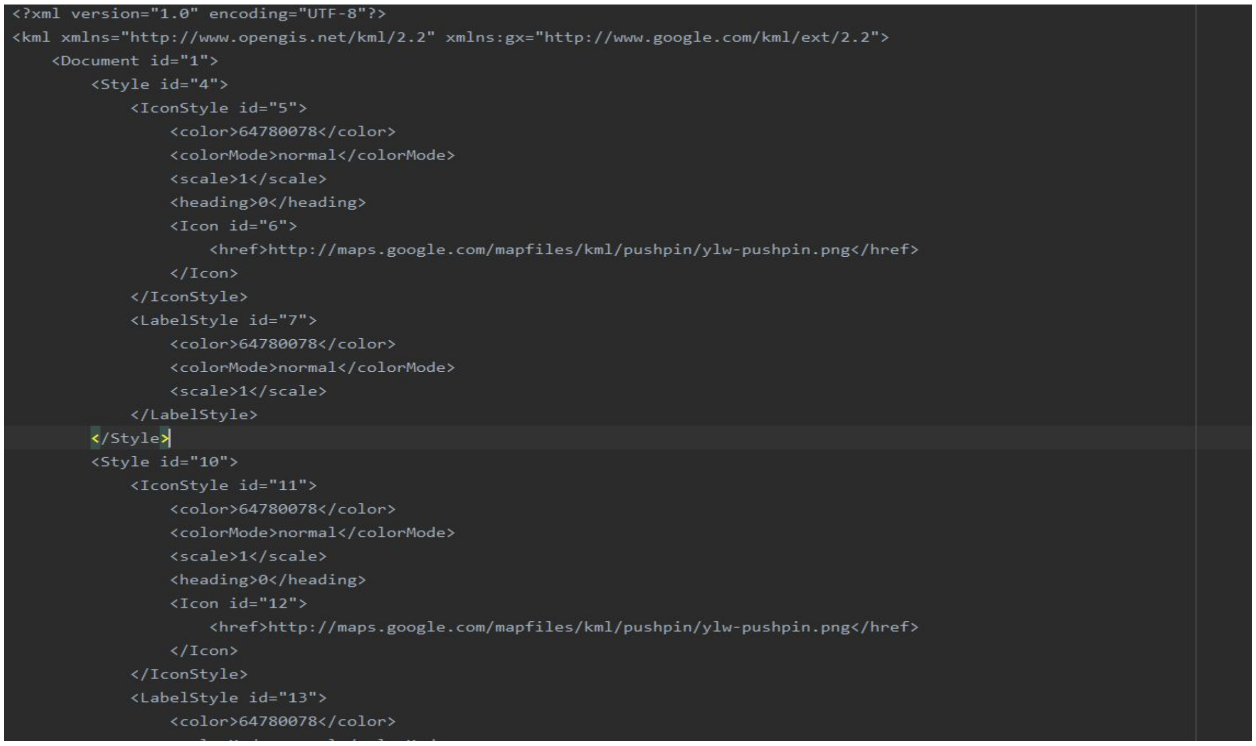


Fig [5] KML file for Stops and Turns



```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2">
    <Document id="1">
        <Style id="4">
            <IconStyle id="5">
                <color>64780078</color>
                <colorMode>normal</colorMode>
                <scale>1</scale>
                <heading>0</heading>
                <Icon id="6">
                    <href>http://maps.google.com/mapfiles/kml/pushpin/ylw-pushpin.png</href>
                </Icon>
            </IconStyle>
            <LabelStyle id="7">
                <color>64780078</color>
                <colorMode>normal</colorMode>
                <scale>1</scale>
            </LabelStyle>
        </Style>
        <Style id="10">
            <IconStyle id="11">
                <color>64780078</color>
                <colorMode>normal</colorMode>
                <scale>1</scale>
                <heading>0</heading>
                <Icon id="12">
                    <href>http://maps.google.com/mapfiles/kml/pushpin/ylw-pushpin.png</href>
                </Icon>
            </IconStyle>
            <LabelStyle id="13">
                <color>64780078</color>
```

Testing examples

Fig [6] Pre agglomerative clustering

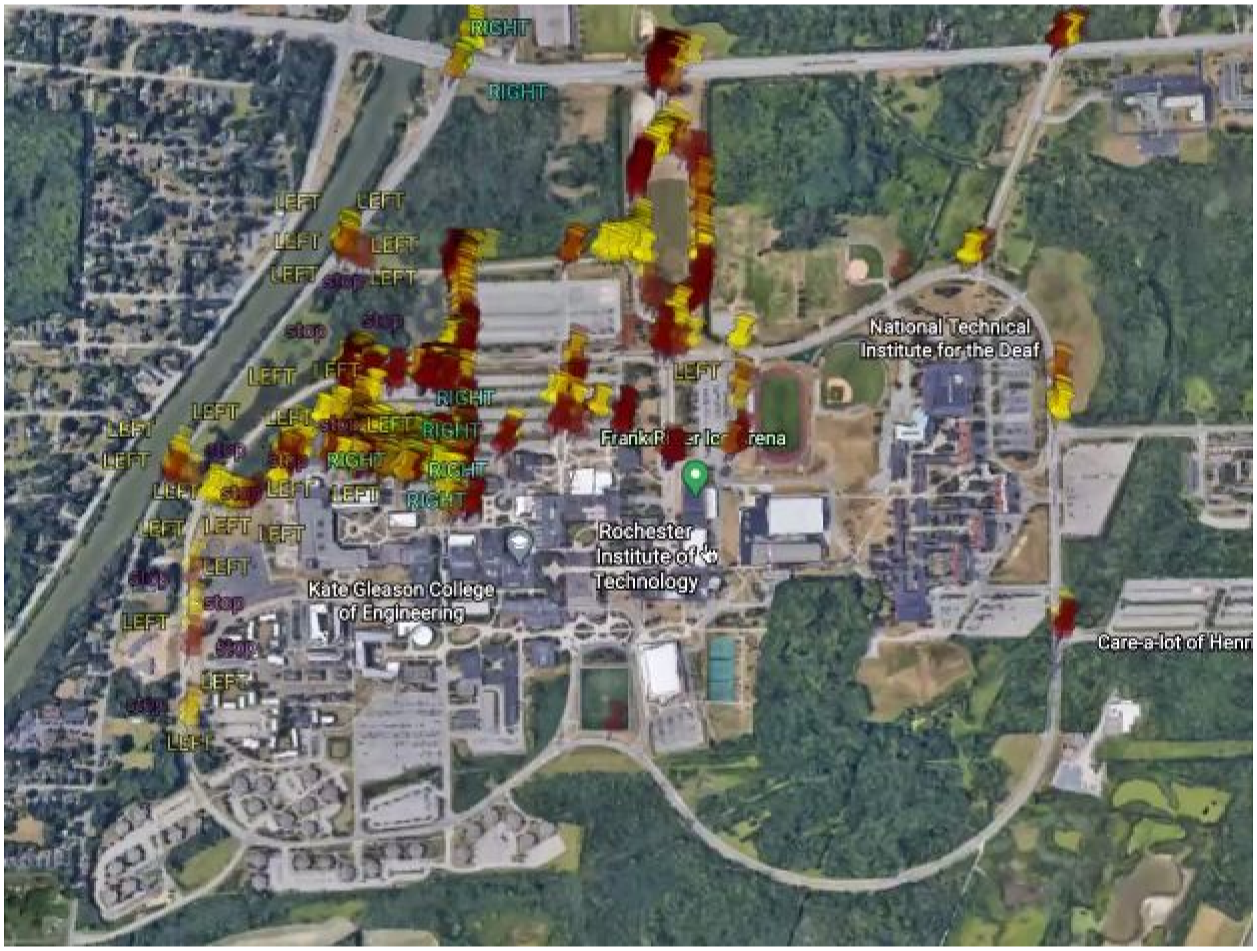

RIGHT
RIGHT
LEFT
stop
National Technical Institute for the Deaf
Frank Ritter Ice Arena
Rochester Institute of Technology
Kate Gleason College of Engineering
Care-a-lot of Henri

Fig [7] example pre-turned variable result

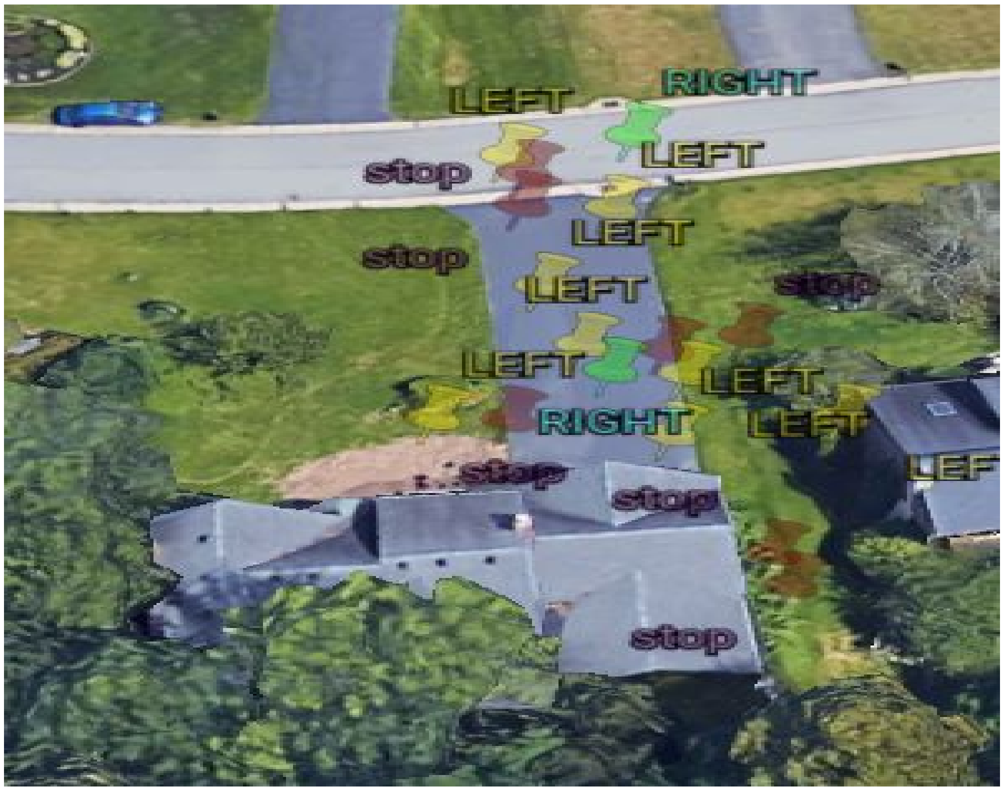


Fig [8] Testing for distance threshold of 0.001

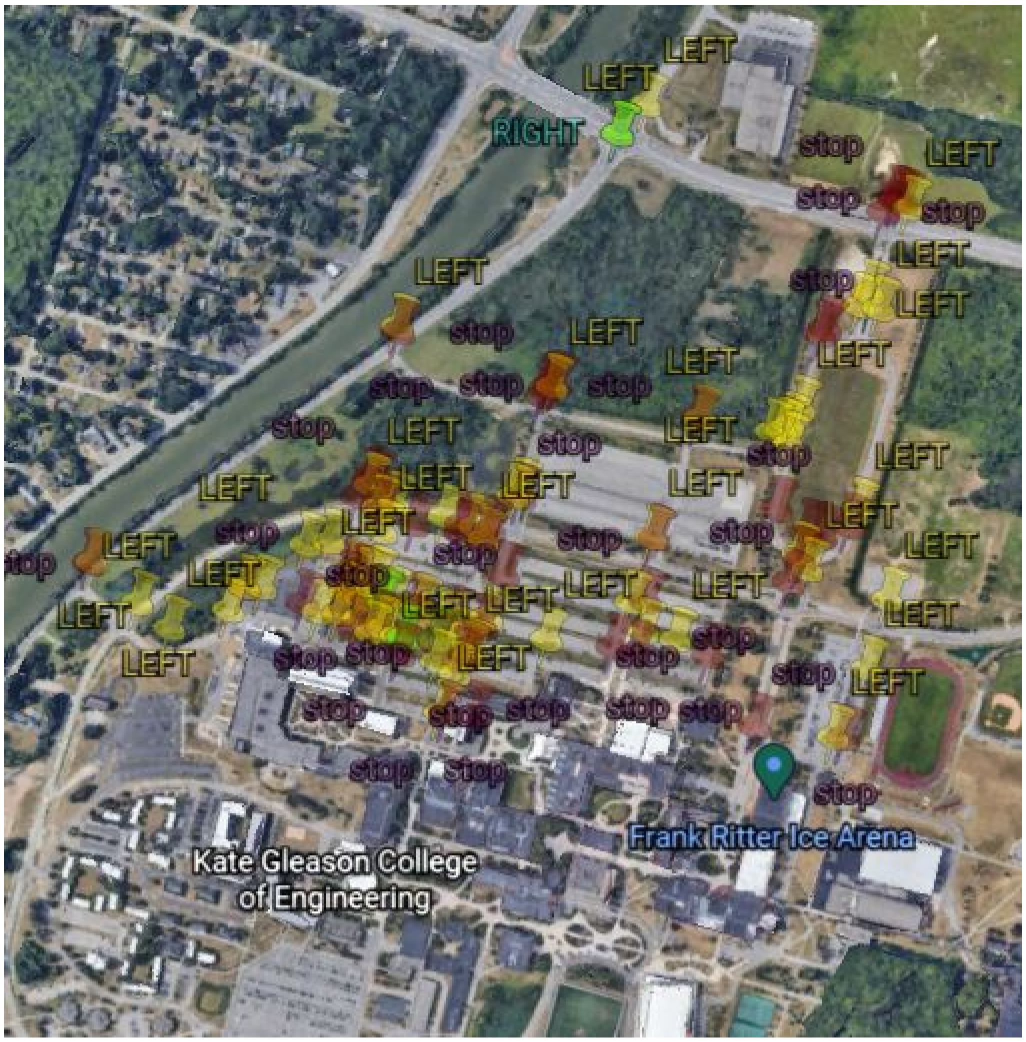
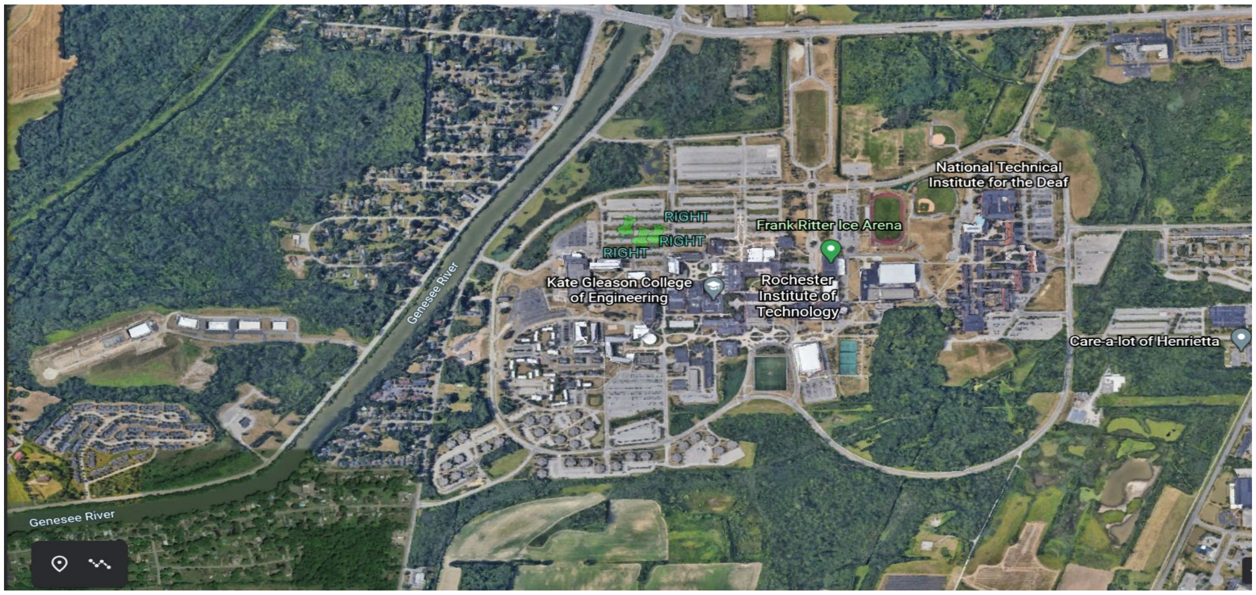
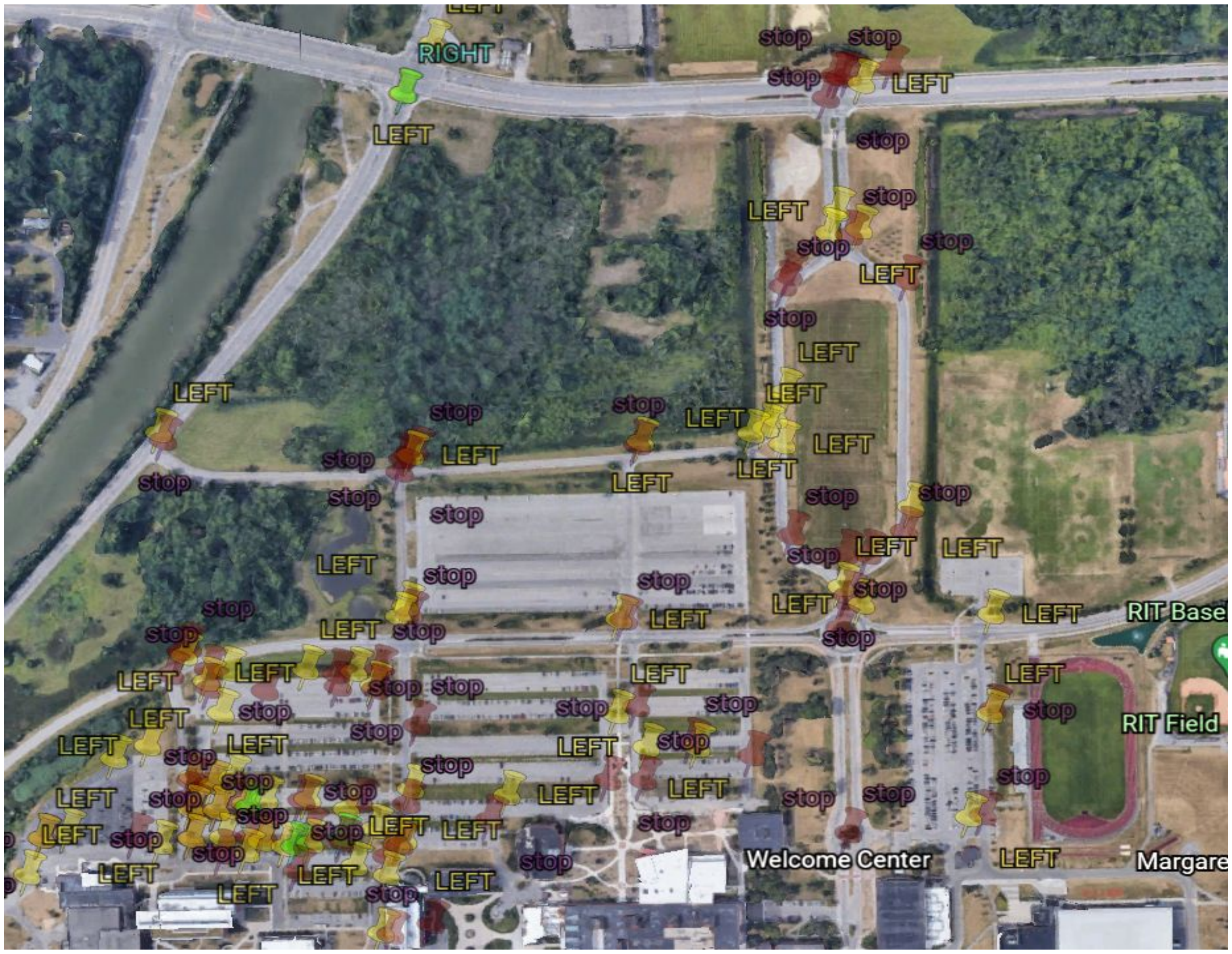


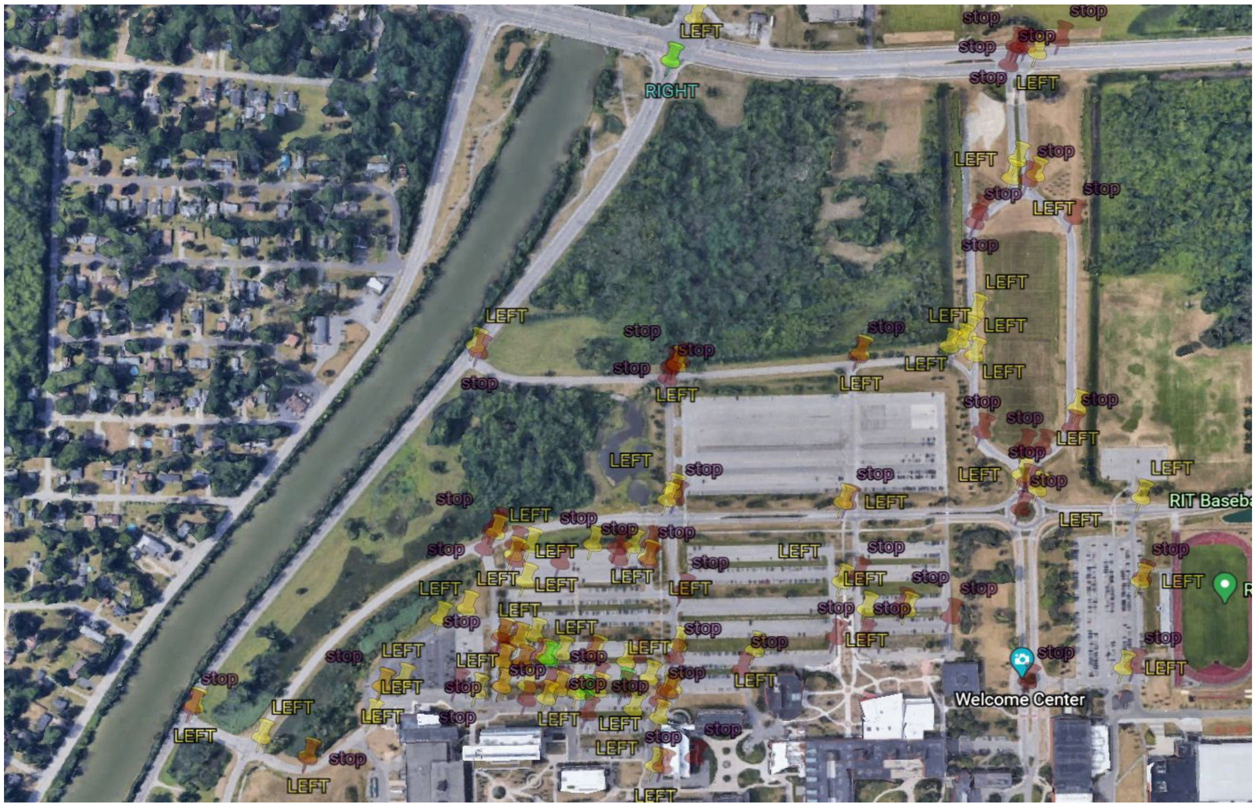
Fig [9] Stopping velocity of 0.0:


National Technical Institute for the Deaf
RIGHT
RIGHT
RIGHT
Frank Ritter Ice Arena
Kate Gleason College of Engineering
Rochester Institute of Technology
Genesee River
Care-a-lot of Henrietta
Genesee River

Fig[10] Stopping velocity of 0.001



Fig[11] Stopping velocity of 0.0005



References

[1] Yang, X., Tang, L., Zhang, X., & Li, Q. (2018). A data cleaning method for big trace data using movement consistency. *Sensors*, *18*(3), 824.

[2] Phondeenana, Peerapat & Noomwongs, Nuksit & Chantranuwathana, Sunhapos & Thitipatanapong, Raksit. (2013). Driving Maneuver Detection System based on GPS Data. 10.13140/2.1.1734.3685.

[3] Langley, R. B. (1995). NMEA 0183: A GPS receiver interface standard. *GPS world*, *6*(7), 54-57.

[4] Yee, W. (2019, June 20). A Gentle Introduction to IoT/GPS Trajectory Clustering and Geospatial Clustering. Retrieved November 30, 2020, from https://towardsdatascience.com/a-gentle-introduction-to-iot-gps-trajectory-clustering-and-geospatial-clustering-daba8da8c41e