BIG DATA ANALYTICS

CSCI 720

GPS DATA VISUALIZATION PROJECT

Sudheeksha Garg[1]* | **Qiaoran Li**[2]*

[1]Computer Science, Rochester Institute of Technology, Rochester, NY, 14623, USA

Correspondence
Sudheeksha Garg, Computer Sciencce, Rochester Institute of Technology, Rochester, NY, 14623, USA
Email:sg3932@g.rit.edu
Qiaoran Li, Computer Sciencce, Rochester Institute of Technology, Rochester, NY, 14623, USA
Email: qxl8365@g.rit.edu

This project uses unstructured data such as GPS messages generated by an Arduino device and converts it to structured data. There are two types of GPS messages, namely GPRMC and GPGGA. In this project we used GPRMC and it contains information such as Latitude, Longitude, Tracking angle, Speed in knots, Magnetic variation, Date of fix, Navigation receiver, Time of fix, mandatory checksum. We only used Latitude, Longitude, Tracking angle, and Speed in knots, and time. The GPRMC had information of route trips from Penfield to RIT, or from RIT to Penfield, or around Monroe County taken by Professor Kinsman. The GPS messages are used to create a KML path file that helps visualize the path taken. Also, using classifiers turns and stop are detected and pinned on the map. The pins are of three different colors: Yellow, Red, Green. Yellow points to a stop light, Red points to a left turn and Green shows a right turn.

KEYWORDS
Agglomeration, Data Visualization, KML, Geographic Information Systems, Turn Detection, Stop detection

1 | OVERVIEW

The purpose of this project is to use GPS data collected using an Arduino micro-controller and generate a KML file. This KML file can be viewed on Google Earth and the path taken can be visualized. For the first part of the project, anomalies such as multiple points at the same location, incorrect GPS data generated when Arduino loses its mind, data generated when is the car is not moving, data generated when GPS status is invalid and the trip is just or ending is removed. It is explained in depth in section 2. The next part of the project was to detect turns and stop lights using classifiers and mark

* Equally contributing authors.

them. It is described in sections 3 and 4 in more detail. For the next part, batch processing was done on all the files to get a combined path, stops and turns. This data had multiple points for a single turn, so agglomeration was performed on the stops and turns to get just one point that defined the turn or stop the best. If a point was classified as both a stop and a turn, it was rectified to present only turns. It is explained in section 5. Section 6 and 7 shows screenshots of the result of the KML file. Section 9 talks about the contribution of each member. And section 10 and 11, discuss and conclude the project. Section 11 discusses future work.

2 | CONVERTING GPS TO KML

Two prerequisites for the successful conversion between the GPS data and the KML files are, first, understanding how to GPS data is organized and stored in the text file, second, understand how tags are utilized in the KML files, such as path vs. points.

```
# sample data from the GPS text file
$GPRMC,233554.400,A,4305.1642,N,07740.8665,W,0.01,195.70,040319,,,D*75
$GPGGA,233554.600,4305.1642,N,07740.8665,W,2,10,0.93,165.9,M,-34.4,M,0000,0000*59
lng=-77.681114, lat=43.086067, altitude=165.90, speed=0.01, satellites=10, angle=195.7000,
 fixquality=2
```

LISTING 1 GPS Data Example

After a bit of research, we realized that the vehicle locations are recorded in a timely sequence in 2 different formats (3rd lines as ignored as it is generated by professor's debug code) as shown in listing 1. Each line is preceded with a prefix that signifies a specific NMEA sentence type [1]. For example, "GRPMC" stands for GPS specific information, while "GPGGA" represents GPS fix data and undulation. Both type contains basic information such as time, data position, longitude and latitude information. However, each possesses a few unique attributes useful for specific cases. In our scenario, "GRPMC" was preferred because it provides speed and position status.

2.1 | Deduplication

Each GPS text file contain thousands of data, to increase the run-time efficiency of our algorithm, a series of techniques are used to remove duplicated or low-value (do not provide much information) data points.

Remove data point when car is stopped or parked

If the vehicle is stopped or parked, Arduino sensor records multiple data points with the same longitude and latitude measures. Essentially, the additional information served no purpose but provide the time difference. Therefore, we designed algorithm remove all but the first and last occurrences of the stopped or parked data points.

```
        # create a columns that flags all duplicates as false (except for the 1 occurrence)
        coordinate_duplicate = df.duplicated(subset=['lon', 'lat'], keep='first')
        df['flag'] = coordinate_duplicate
        # drop all the duplicated coordinates, keep the first and last occurrences.
        df = df.drop_duplicates(['lon', 'lat', 'flag'], keep='last')

```

LISTING 2 deduplication on stopped or parked car

As shown in listing 2, the resulting data points marks the beginning and ending of each stop or parking, making is easy to calculate the time difference between the two.

Remove data point at the start of the journey

```
        # if the first and second records are the same, remove the 1st
        first_coordinate = df.iloc[0,:2].tolist()
        second_coordinate = df.iloc[1,:2].tolist()
        if first_coordinate == second_coordinate:
            df = df.drop(df.index[0])

```

LISTING 3 start point deduplication

When the trip first starts up, the vehicle is not moving. The previous algorithm will preserve the first and last occurrences of any stop, therefore, when a car is first started up in the parking lot or the drive way, it will be marked as a stop. To address this issue, first and second data points within the data-set will be evaluated, and if they are the same, the first records will be removed, as shown in listing 3.

Remove data point at the end of the journey

```
        # if the last and second last records are the same, remove the last
        last_coordinate = df.iloc[-1, :2].tolist()
        second_last_coordinate = df.iloc[-2, :2].tolist()
        if last_coordinate == second_last_coordinate:
            df = df.drop(df.index[-1])

```

LISTING 4 end point deduplication

Similarity, when the vehicle stops moving when it reaches the destination. The previous algorithm will recognize this as a stopping point, we would like to avoid this by removing the last data point if it is the same the the second to last point, shown as in listing 4.

Sub-sampling

```
        # book-keeping variables
        list_of_straight_index = []
        old_speed = df.iloc[0]['Kn_speed']
        # loop through the dataframe
        for index, row in df.iterrows():
            speed = row['speed_Kn']
            # keep track of the indexes where the speed was constant
            if speed == old_speed:
                list_of_straight_index.append(index)
            else:
                old_speed = speed # update the old speed
        # stepping through the list
        SAMPLE_STEPS = 3 # drop 1 data point for every 3
        for index in range(0, len(list_of_straight_index), 3):
            df.drop(list_of_straight_index[index], inplace=True)

```

LISTING 5 straight line deduplication

If the vehicle is traveling in a straight line, we could ignore some points. Since we are using speed as a criteria to for stop and turn detection, we know for certain that sub-sampling the data will create problems for the algorithms. However, if the speed is constant, there is no way a turn or a stop happened, so we can safely remove some data points in these regions, as shown in listing 5.

2.2 | Anomaly Removal

Sensor Error Based Anomaly

```
    old_longitude, old_latitude = df.iloc[0]['lon'], df.iloc[0]['lat']
for index, row in df.iterrows():
    longitude, latitude, speed = row['lon'], row['lat'], row['speed_Kn']
    temp = get_haversion(pos1=[latitude, longitude], pos2=[old_latitude, old_longitude])
    if temp > 5:
        # print(old_latitude, old_longitude, latitude, longitude, temp)
        df.drop(index, inplace=True)
    # book-keeping for the last
    old_longitude = longitude
    old_latitude = latitude

```

LISTING 6 jumping anomaly

The Arduino sensor sometimes malfunctions, and starts recording GPS values that jump all over the place. This happens when the GPS loose connection while the vehicle is at uncharted territory or a region with poor signal such as tunnels. The resulting data point often shows unrealistic speed and distance maneuver. Which is exactly how we screen and remove these anomalies. Listing 6 shows that we've set the haversion distance to 5 mile for all the adjacent data points as a threshold. Any vehicle moving further than 5 miles in 1 timestamp will be removed as an anomaly.

IO Based Anomaly

```
for row in gps_data:
if len(row) != 0:
    # ------------------------------------------------------------------------
    #  The Arduino sometimes burps, and writes two GPS sentences to the same line of the data
 file. Detect and ignore these anomalies. Otherwise it looks like the car jumps from one side of
 the planet to the other side.
    # ------------------------------------------------------------------------
    # take the first entry if there are 2 entries per line and A represent valid data points
    if row[0] == '$GPRMC' and row[2] == 'A' and len(row) == 13:
        twodArray.append(row[:13])

```

LISTING 7 IO anomaly

The Arduino's IO sometimes fails, and writes two GPS sentences to the same line of the data file. In this case, we would just take the first GPS sentences if its validate. Additionally, some files have outright the wrong format for some of the GPS records, "2019_03_25__2228_22.txt" is one such example. Together, we've developed the algorithm shown in listing 7 to handling all the anomalies mentioned.

2.3 | Edge-Case Handling

When there isn't enough data

```
        # make sure it still has enough data
        for row in gps_data:
        if len(row) != 0:
                if df.shape[0] < 4:  return df, df, False
        df_with_begining_and_ending = df

```

LISTING 8 enough data

During the process of removing duplicates and anomalies, the data frame become increasingly smaller. Sometimes, it reaches a point where there are only 2 data point left, which causes all sorts of problem with the data cleaning algorithms, thus we have to include a condition check to make sure there are enough data remaining, as shown in listing 8.

3 | STOP DETECTION

```
    # book-keeping variables
    stop_lights_list = []
    slow_pt, fast_pt = 0, 0
    # iterate the slow point as long as it is not at the end
    while slow_pt < df.shape[0]:
        # initialize the time variables
        start_time, end_time = -1, -1
        # update the start time with slow pointer if it's < 10 mph
        if df.iloc[slow_pt]['speed_Kn'] <= 10:
            start_time = df.iloc[slow_pt]['utc']
        else:
            slow_pt += 1
            continue
        # increment the faster point as long the data point speed < 10mph, and update the end time
        for fast_pt in range(slow_pt+1, df.shape[0]):
            if df.iloc[fast_pt]['speed_Kn'] <= 10:
                end_time = df.iloc[fast_pt]['utc']
            else:
                break
        # compute the time duration between the 2 times
        duration = time_difference(end_time = end_time, start_time = start_time)
        # if the duration is less than 30 second, record the stopping coordinate
        if duration >= 30:
            stop_lights_list.append([df.iloc[fast_pt-1]['lon'], df.iloc[fast_pt-1]['lat'], df.iloc[
     fast_pt-1]['speed_Kn']])
        # update the slow point for next windowing
        slow_pt = fast_pt + 1
```

LISTING 9 Stop detection

We used a threshold classifier for the stop detection. The thought process was to iterate through the data-set with a fast and slow pointer, where faster point always marks the start point of the vehicle's speed become less than 10 miles per hour, while the fast point marks the end point. If the two pointers' records have a duration for less than 30 seconds, then we will mark this as a stop point, as shown in listing 9. The algorithm is relatively robust, and only misses on the stops that's extremely sudden (flooring the break paddle). We could incorporate single point below 10 mph, but it was not necessary since the stop eventually gets picked up from another GPS text tile.

4 | TURN DETECTION

4.1 | Right turn

The $GPRMC message generated by Arduino contains the angle of the car with respect to true North. The classifier to detect turns uses the difference in the angles of two different data points over a sliding window of 15 points. If the angle is greater than equal to 100 or less than equal to 40, it is classified as a right turn.

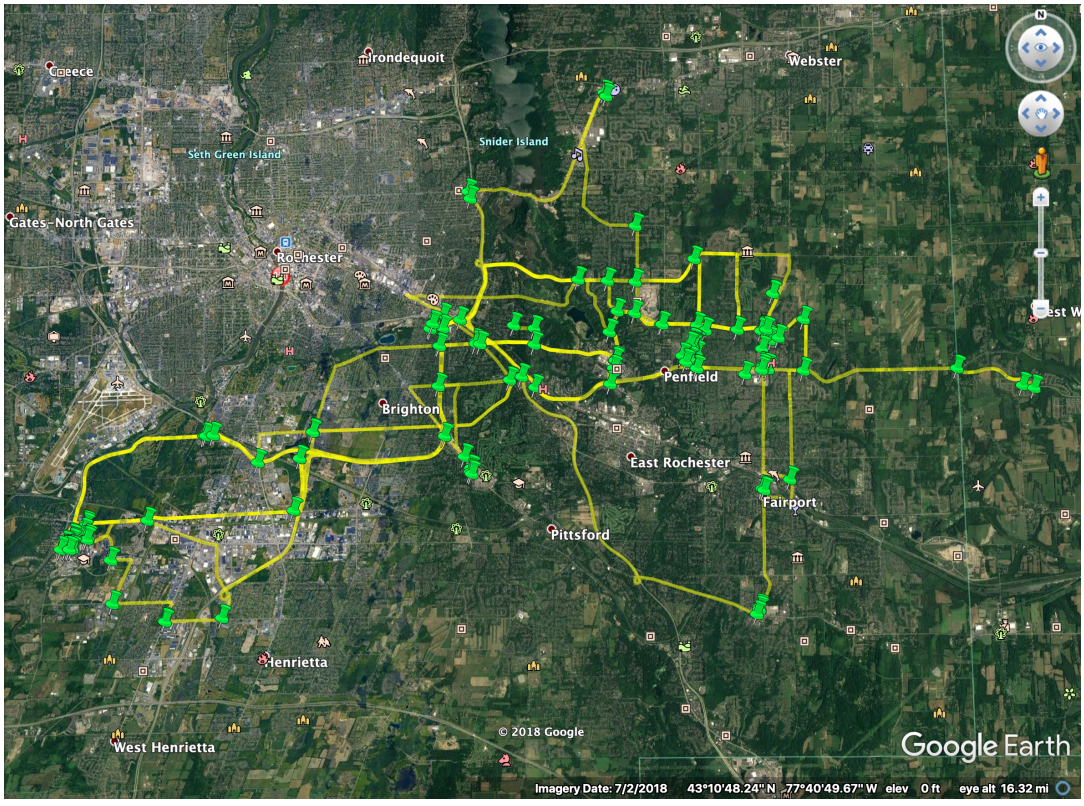

FIGURE 1 Screenshot of right turns

4.2 | Left turn

In case of left turns similar approach as right turns was used with the difference of range of difference between two angles being greater than equal to -50 and less than equal to -90.

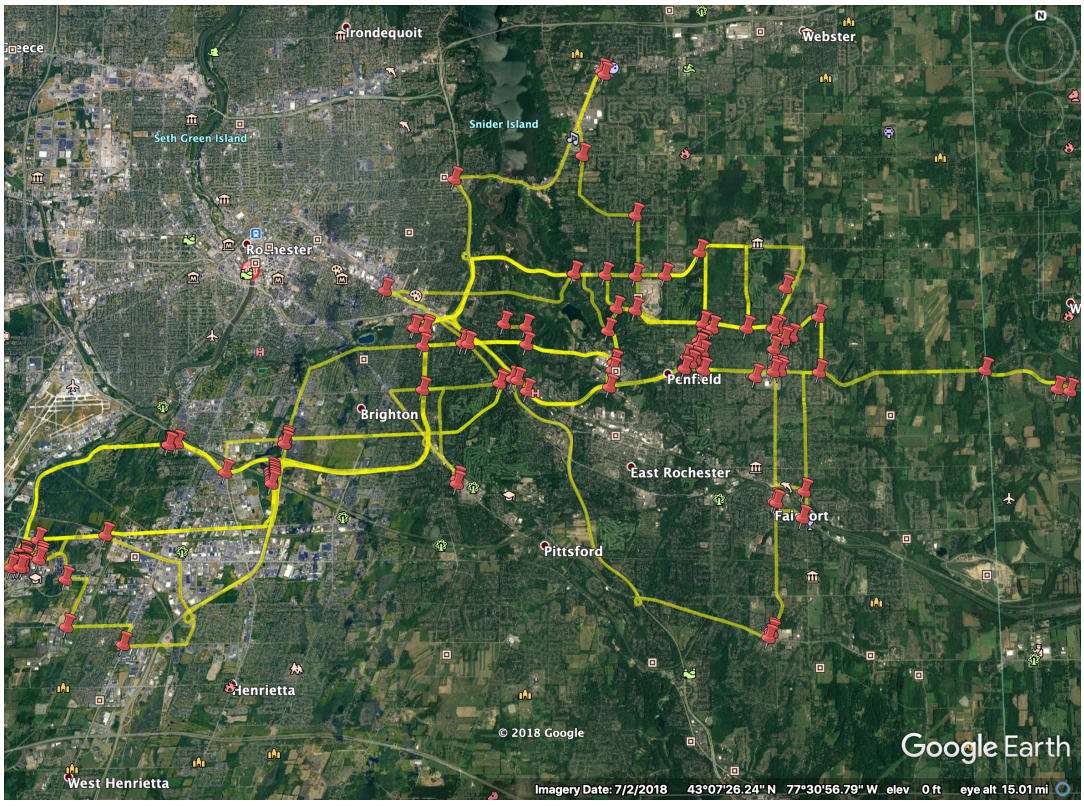

FIGURE 2 Screenshot of left turns

This classifier does not detect every turn as sometimes the difference in angle can be beyond the scope of the classifier but widening the range can cause false positives which are worse than false negatives in this case. Though during batch processing, all the turns are detected as during different paths, the angles fall in the scope of the classifier. In future, a more robust and more run-time efficient classifier will be created.

5 | ASSIMILATING ACROSS TRACKS

We designed our program following the head, body, and tail design principle. In the head section, we read and process the data, in the body section, we designed algorithms to detection turns and stops, and lastly in the tail section, we pump out the KML tags and saved the map. To store all the turns and stops, we used a Numpy array to do the book keeping. Each Numpy array element is a list of coordinates which includes longitude, latitude, and altitude (substituted with speed for better visual). The array is populated while running the detection algorithm.

The Numpy arrays are returned as a variable to the batch processing main python program for each GPS text file. We run the data cleaning and detection algorithm on each of the files literately, and collected the data during the process. Naturally, the turns and stops pined by each file is slightly different, and was hard to see which is more accurate. to address this problem, we use clustering agglomeration algorithm from sklean. Sometimes, a location is marked as a turn and a stop. To stay consistent, we prioritized the turnning result, so an algorithm had to be use to remove stops that are

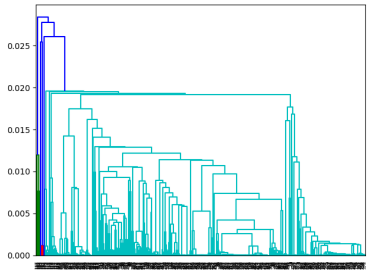

FIGURE 3 Stop Dendrogram

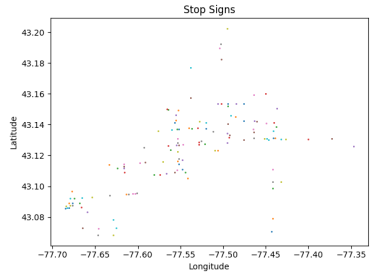

FIGURE 4 Stop Clusters

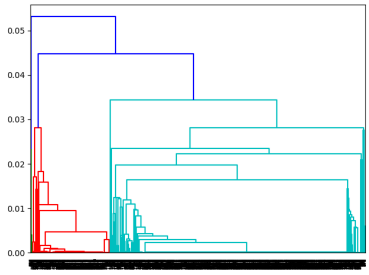

FIGURE 5 Left Turn Dendrogram

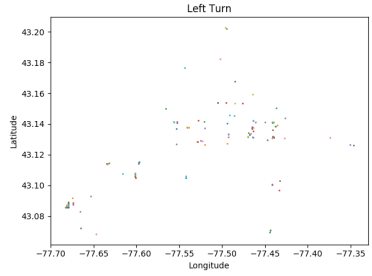

FIGURE 6 Left Turn Clusters

also in the list of turns.

5.1 | Parameters Used

Some of the parameters used for the clustering agglomerations are linkage, connectivity, and number of cluster. Generalized speaking, we choose simple linkage, 3 k-nearest neighbor graph as connectivity, and 150 ish clusters. The results can be see in figure 3 to figure 8.

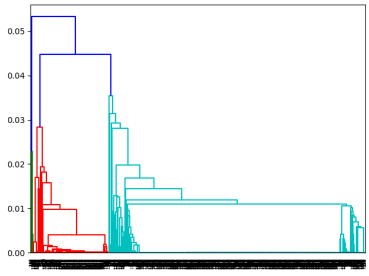

FIGURE 7 Right Turn Dendrogram

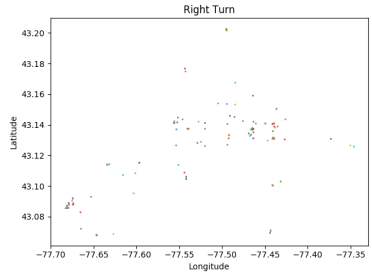

FIGURE 8 Right Turn Clusters

6 | RESULTS OF A PATH FILE

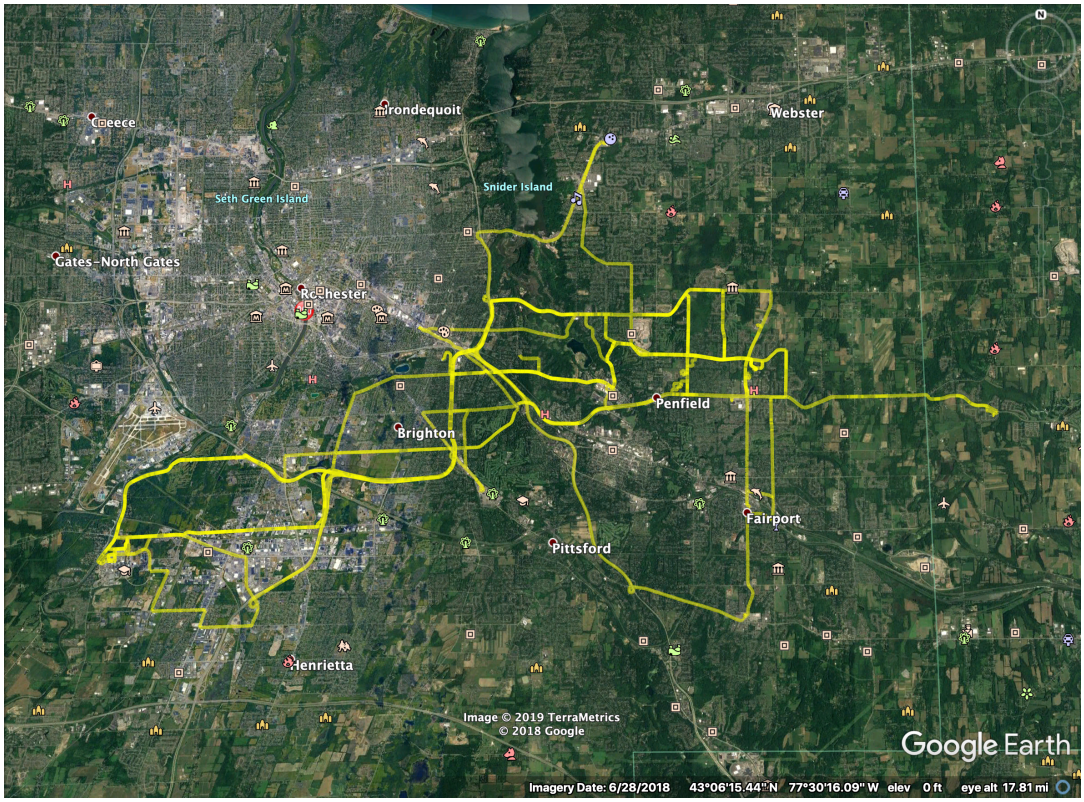

FIGURE 9 Screenshot of all KML paths

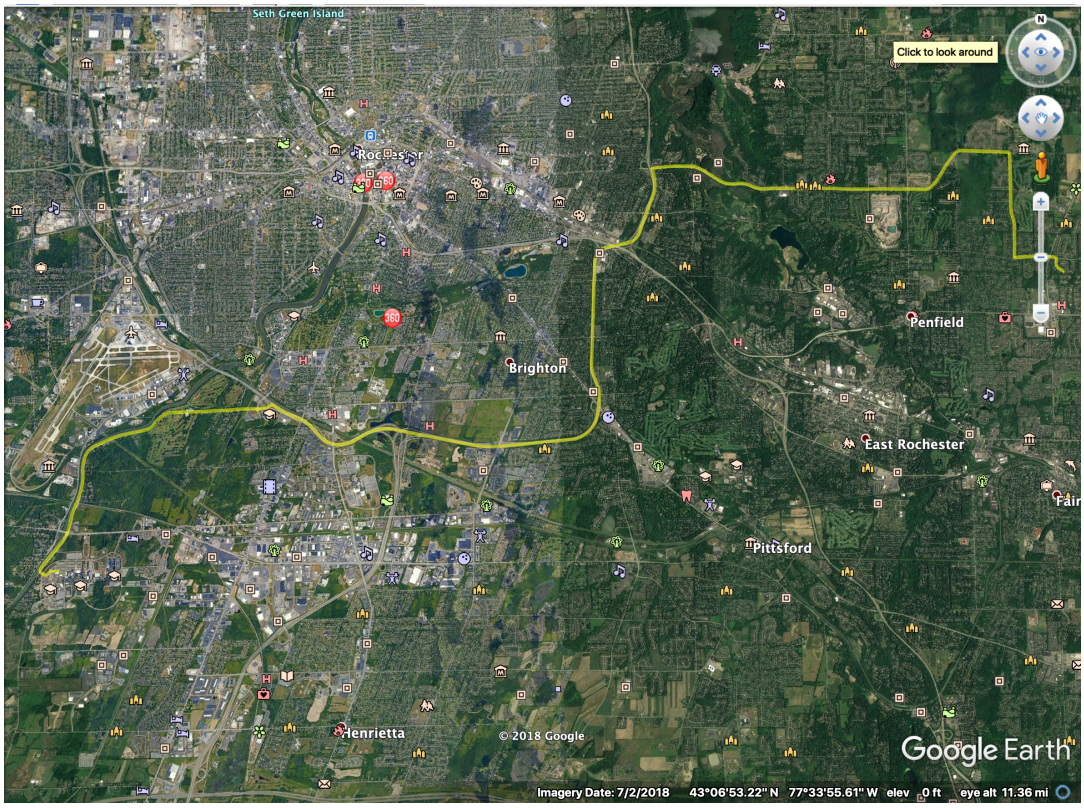

FIGURE 10 Screenshot of one KML path

7 | RESULTS OF A HAZARD FILE

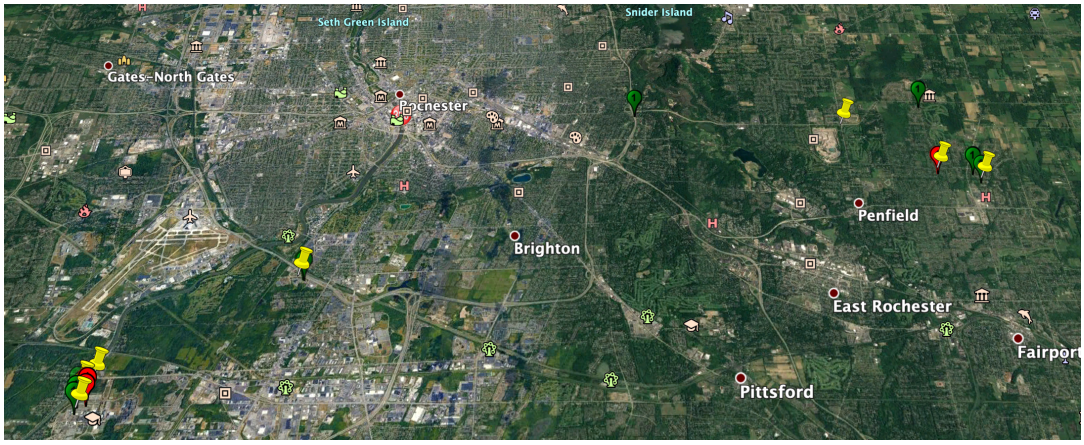

FIGURE 11 Screenshot for 1 file: 2019_03_04__RIT_to_Home

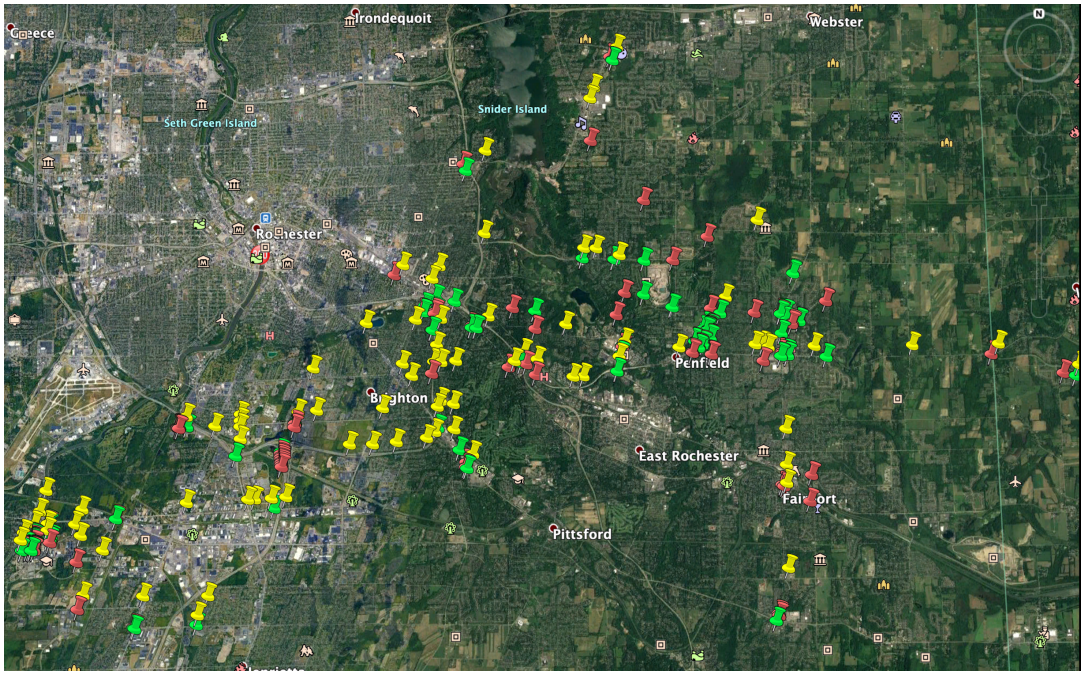

FIGURE 12 Screenshot of all the hazards

8 | SCREEN SHOT OF A *.KML FILE

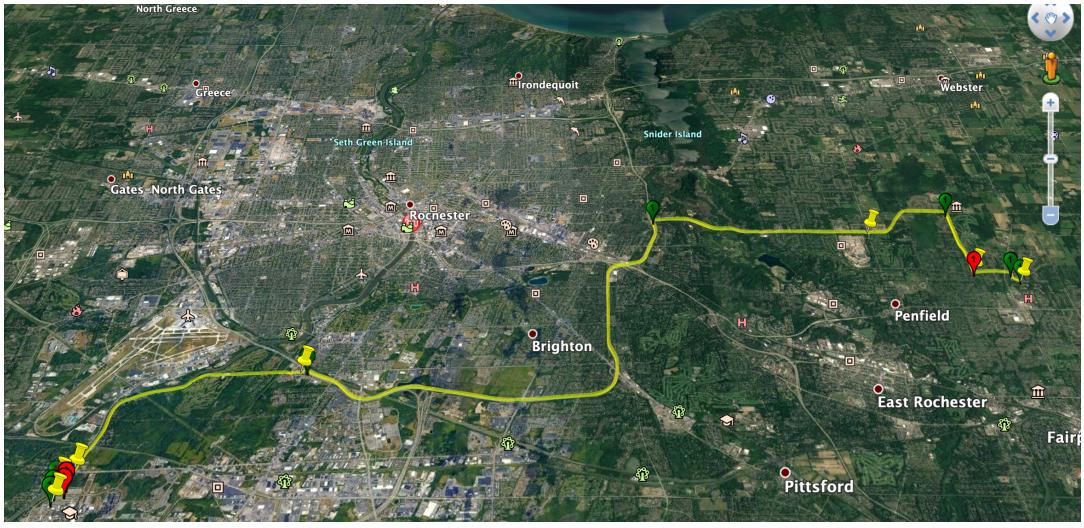

FIGURE 13 Screenshot for 1 file: 2019_03_04__RIT_to_Home

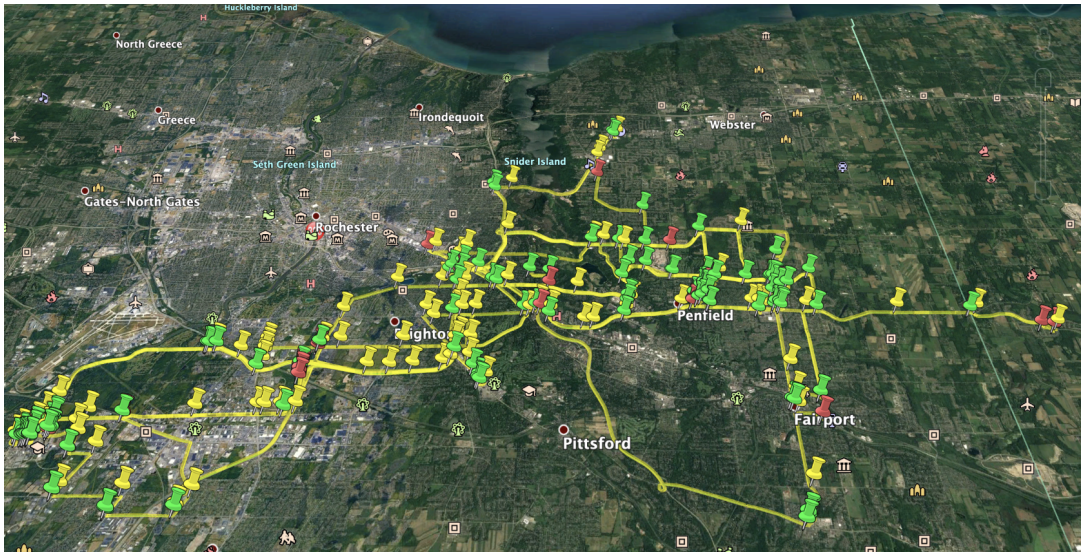

FIGURE 14 Screenshot of all the hazards and paths

Task Allocation	
Sudi	**Qiaoran**
Right and Left Turn detection	Stop Detectons
KML file generation	Data preprocessing
Removing stops that are turns	Batch processing all files
Report	Report

9 | TEAM COMPOSITION

We tried to best divide the work to balance the load across the team. Detailed task allocations can be viewed in the table above.

10 | DISCUSSION

Some of the problems we faced mostly had to do with, one, learning how to building a KML file and resolving any errors that pops up, two, finding an adequate classifier to detect stops and turns. To resolve them, lots of trail and error and manual-checking the coordinates were necessary. Prior to the algorithms, we used a series of data processing techniques that removed duplicates, anomalies. Additionally, we applied sub-sampling filter on the data to ensure greater efficiency.

11 | CONCLUSION (SG)

This project brought about an understanding in many different fronts. Some of them being that it helped to learn how to use unstructured data and bring it to a structured format that can be used in data mining. It also helped in understanding how to use different concepts learned in class such as Agglomeration, building classifiers, and data cleaning. We also learned how to build a basic KML file. Some of the challenges presented in the data processing and visualization, are quite transferable. Meaning that, the techniques used to resolve these problems, can be used in a wide range of problem in the future project.

This project can act as building blocks to commercial applications such as turn-by-turn navigation system, traffic congestion maps, and find paths that more cost-effective or routes that the shortest or paths that have the least travel time. An example of a cost-effective route is a path without left turns. This concept is used by UPS to be more productive.Overall, we enjoyed the project, and gained confidence in handling large complex problems. Following the head, body, tail design principle, any challenges can be divided and conquered with patience and the willing of learning new skills.

12 | FUTURE WORK

With limited time, we are only able to detect the stop and turns with around 90% accuracy. However, for future work, a more robust and run-time efficient classifier to detect turns and stops can be built such as automatically adjust the

threshold of the classifiers based on accuracy rate. Better yet, we can even try to model a LSTM network for the data-set.

REFERENCES

[1] ARDUINO, 2014. Lesson 24: Understanding gps nmea sentences.