Samuel Adams, Marilyn Groppe, Bahdah Shin
Dr. Thomas Kinsman
CSCI 420
29 November 2020

Using a Decision Tree to Identify Hazards in GPS Data

Samuel Adams, Marilyn Groppe, Bahdah Shin

Abstract

GPS systems have become a staple in how we exist in modern society. We used the output of one of these built in systems to build a decision tree. This decision tree is used to classify the data to identify possible hazards in a set path through Rochester, New York. This tree uses segments of data and changes over time to properly analyze the path taken.

Overview

This paper contains a report on the team composition, how we tackled the project breakdown, our procedure for dealing with different problems in the development process, and a summary of our results and what we learned. This project required clear teamwork and group problem solving to allow us to complete it in the time given. We used both synchronous and asynchronous communication methods to delegate, ask questions, discuss issues, and debug code. We also shared online resources such as articles and discussion forums to help get our footing with file formats we were unfamiliar with.

Team Composition

The three of us worked to keep things balanced without pushing people to work when they had other projects or assignments to work on. This helped us naturally delegate tasks, as we have not all worked together enough to have a clear understanding of our individual skills and struggles. In addition, we utilized multiple communication networks to help each other work through bugs and design questions. This helped make the project feel like a cohesive effort from each of us rather than three individual parts smushed together. We found that both our experiences in work study as well as previous group projects help keep this project on track, and our team running smoothly.

Project Decomposition

The timing of this project and our other classwork required us to be proactive and flexible when delegating and breaking down this project. Tasks were mainly delegated based on who was available at what times during the development process, since this project was a fairly linear one. The data first had to be processed, then labelled, then analyzed, and then the classifier is built and run. A more detailed description of the steps we took and the problems we face are in the next section. We designed this project to work in a command line interface or in an IDE. We used GitHub and Discord to develop and debug our code. Code was committed and pushed frequently to ensure that code worked in all setups.

Procedure

I. Converting GPS to KML

This part of the project was maybe the most frustrating, as it felt a bit like playing Whack-A-Mole. Once one bug was solved, another one appeared. This is how data processing goes, so it was not a surprise to us. The first process was processing and cleaning the data read in from the given text files. We used the expected file formatting to identify whether a data point was corrupt or recorded correctly. Then we had to decide which of the data fields mattered for our analysis and which could safely be excluded. To do this, we chose which format we wanted to focus on, and looked into what units were used to record that information. This allowed us to exclude the fields that were not needed for our work or unit conversion. After this, we had to identify which data points were anomalies or not useful that could be removed. To do this, we found the general coordinate values for the city the data is from (Rochester, NY) to generate a threshold that the truck must be in. This way, points taken from a faraway location could be safely discarded as an anomaly from the GPS itself. We also chose a threshold for speed changes to filter out points where the car is sitting in one place for a long time, or driving straight for a long time. These thresholds were decided based on educated estimates, but could be refined using N-Fold Cross Validation if we had a way to easily verify if our system's estimate was correct.

We have found the following sources to be helpful in decoding the GPS format.

- https://docs.novatel.com/OEM7/Content/Logs/GPRMC.htm
- https://docs.novatel.com/OEM7/Content/Logs/GPGGA.htm
- https://docs.novatel.com/OEM7/Content/Logs/GPGSA.htm
- https://docs.novatel.com/OEM7/Content/Logs/GPVTG.htm

II. Stop Detection

For detecting a stop, we wanted to use an approach that treated an instance of a stop as a single event, rather than recording all places that speed was zero and agglomerating to single pins. To achieve this, we used a speed threshold such that whenever the car dropped below the speed threshold, we began tracking all of the following points until the speed of the car increased back above the threshold. We used a threshold of 6 MPH because we noticed that the car was never below that speed unless it was accelerating or decelerating.

The result was a sequence of points where the car was decelerating to a stop and then just began accelerating again. We also tracked the elapsed time of this sequence of points using the UTC time at each point. Once the car began accelerating again, we stopped adding points to the sequence and checked how much time had elapsed. If the elapsed time was greater than 3 minutes, we ignored the sequence because it was most likely not a stop sign or stop light, i.e. stopping for an errand. We also ignored sequences that had less than 5 points, because we didn't want to track deceleration/acceleration events that weren't the result of an actual stop, i.e. only slowing down for a sharp curve.

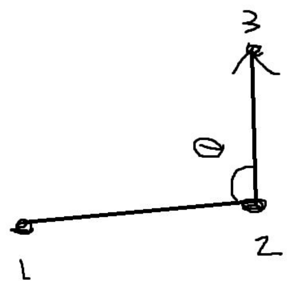
This left us with a collection of sequences of points that represented stopping events. For each sequence, we used the point that came just before the car beginning to accelerate again as our pin for the Cost Map.

One problem we encountered is when the GPS didn't catch enough points for the stop so the thresholds couldn't accurately label the event as a stop and instead just thought the car was briefly slowing down. However, this only happened occasionally and we believe we captured most of the stops accurately.

III. Turn Detection

For turn detection, we used a similar strategy to the stop detection so that each turn event was recorded as a sequence of points, of which we could find the best point in the turn. To achieve this, we used a speed threshold such that whenever the car dropped below the speed threshold, we began tracking all of the following points until the speed of the car increased back above the threshold. The reasoning for this is that most, if not all, turns require the driver to slow down a bit to make the turn before performing the turn. The threshold we used for this was 30 MPH, at which point we would begin collecting the points in sequence.

Now that we had sequences of points where the car dropped below 30 MPH and then began accelerating again, our next move was to calculate the change in direction/angle from the beginning of the sequence to the end of the sequence. To do this, we needed three points: point 1 being the beginning of the sequence when the driver started slowing down, point 2 being the car in the middle of the turn, and point 3 being the car after the turn and starting to accelerate again. The figure below shows how this gives us our angle of the turn.



A diagram of the process of finding the degree of a turn

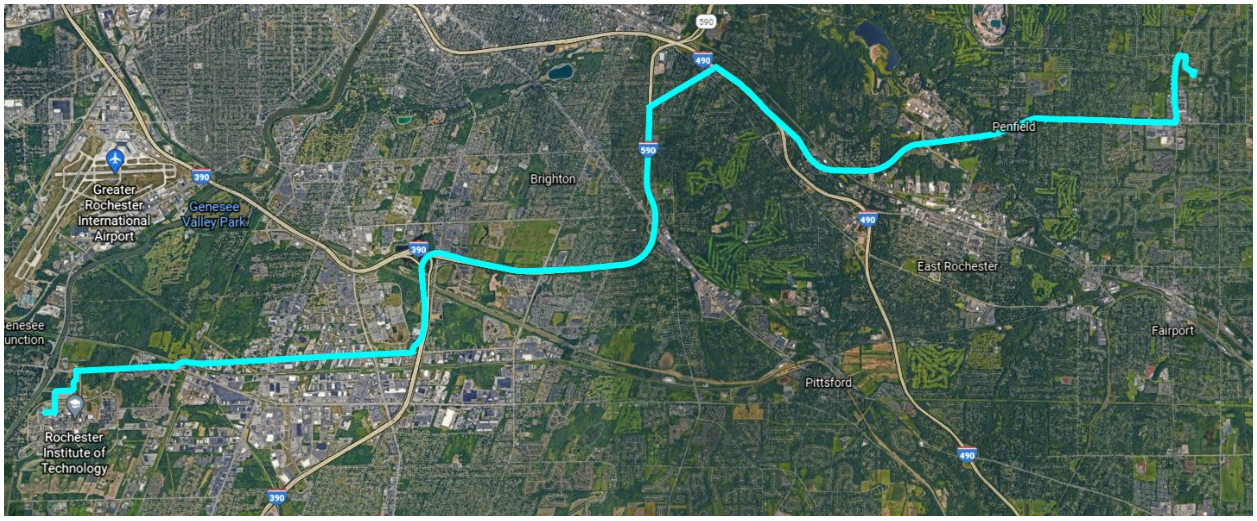
Finding point 1 and 3 was easy, we just took the beginning and end points of the sequence. For point 2 however, we looped through all remaining points in the sequence and calculated the resulting angle, keeping the point with the best resulting turn angle as the turning point. Now that we had the best turning point and angle found in each recorded sequence, we used a threshold of 40 degrees to determine if the sequence was actually a turn. If the sequence met this threshold, we added it to our “found turns”.

Now that we had a collection of sequences that we believed to be turns, we used the turning point that we found in each sequence as the pin location for our Cost Map. To determine if the turn was left or right, we used our three points in each sequence to find the cross product and if the result was greater than zero then it was a left turn and if less than zero it was a right turn.

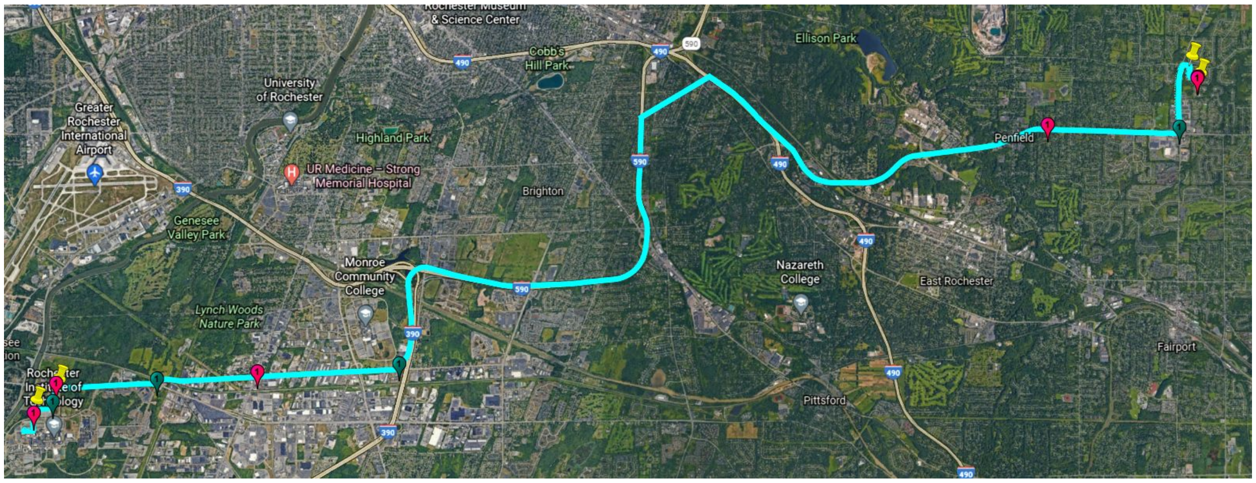
This strategy was mostly successful and our testing showed to have decent results. Just as with the stop detection, the thresholds were confused by instances where the GPS failed to gather enough points in a uniform manner. We also found that we had the occasional incorrect classification of a sequence as a turn when it was just a slight bend in the road, but this was not too common. We kept our thresholds where they were because that setup proved to be most successful for us during testing.

IV. Assimilating Across Tracks

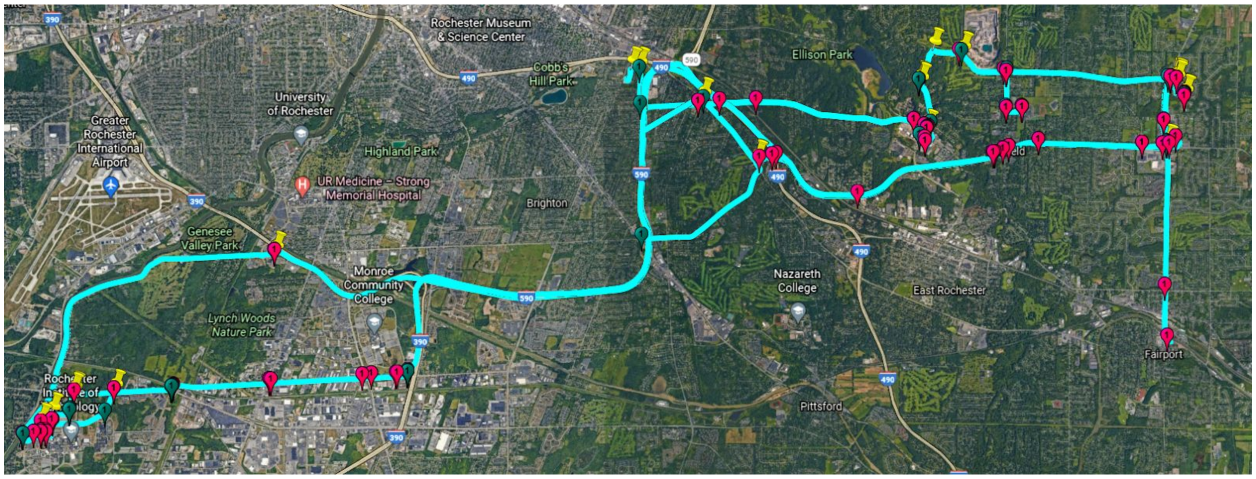
After parsing the GPS data, we store the longitude, latitude, speed, and time from GPRMC format in a two dimensional matrix array. Each column represents the four attributes and each row represents a single data point. We processed the entire file or files and held it in memory until we dumped it into the kml file. We kept track of each data set by appending it into another array. This is important because each dataset has a start and an end. If it merged together, it would look like a one continuous track where the end of one dataset jumps to the beginning of another dataset. Each dataset, or tracks are written inside their own LineString block.


Greater Rochester International Airport
Genesee Valley Park
Brighton
Penfield
East Rochester
Fairport
Pittsford
Rochester Institute of Technology

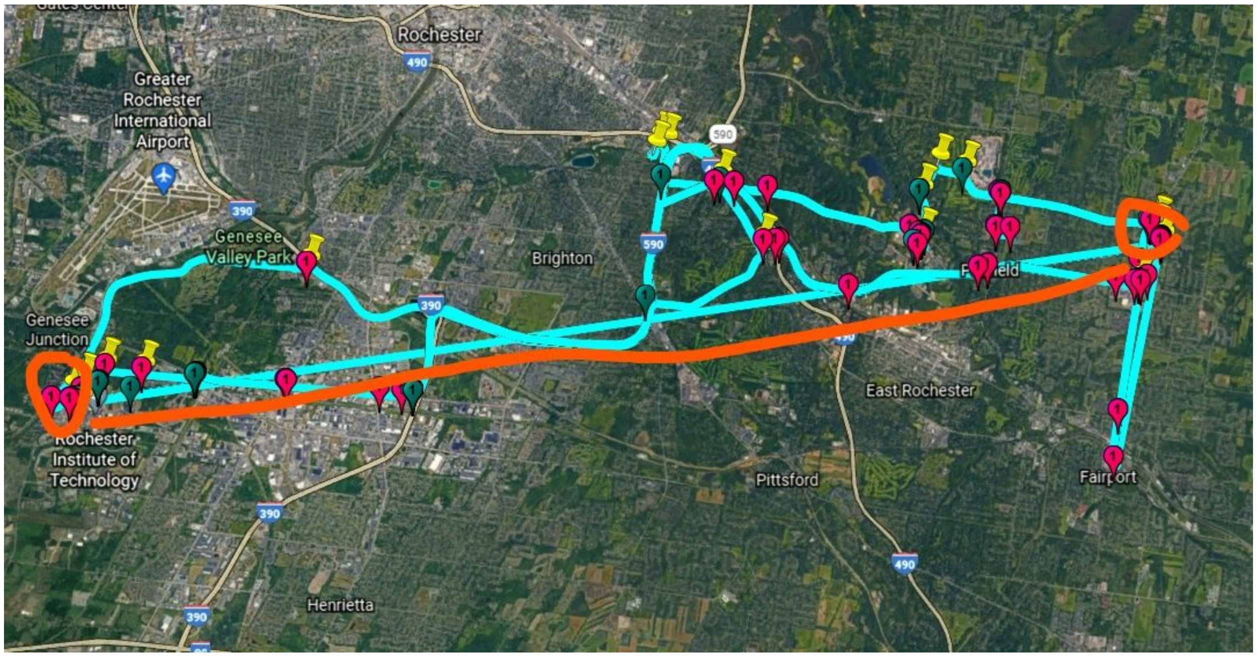
Screenshot of path KML file.



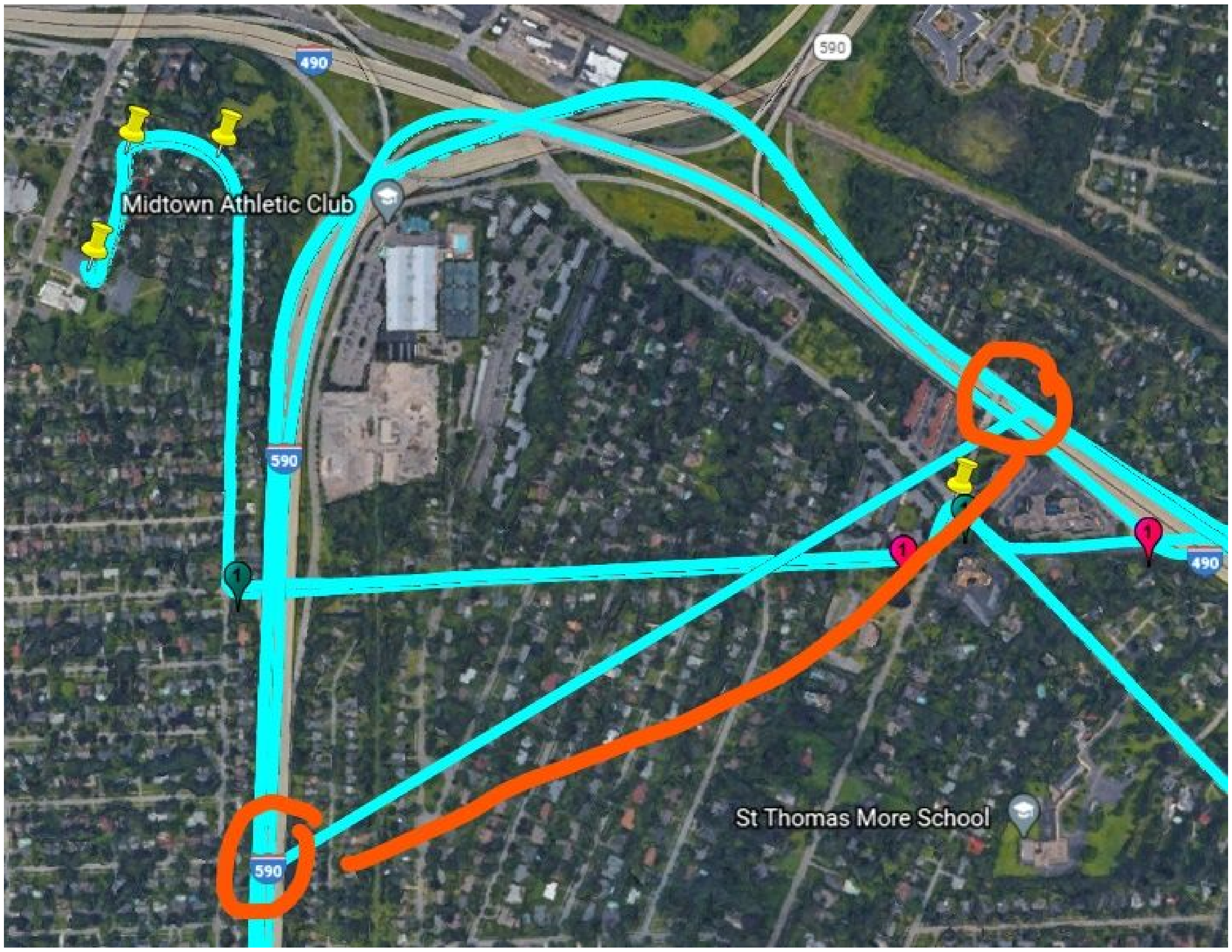
Screenshot of turn/stop KML file.



Screenshot of final KML file



This is what happens when you don't keep the tracks separate into blocks in the kml file.



Weird data points missing in gps file.

Discussion

This procedure actually worked really well, especially when the data had very few erroneous lines within it. It was these holes in the data that would cause issues in our classifier, since we classified groups of lines as hazards rather than individual lines. The main noise problems that we had were duplicate or matching records. Many of the GPS files would record duplicate positions, especially if the change in position or speed is so small that the computer can't pick up on it. We didn't need to do a ton of signal processing, but we did attempt to eliminate insignificant changes in the recorded data as well as ensuring that the data was valid. Overall, our final design was concise and effective.

Conclusion

This project was a really interesting peek into what professional data scientists do in their day-to-day lives. From file I/O to data visualization, this was a complete project rather than a toy version of a data science project. Not only that, but working in a small group of developers during the busiest weeks of the semester to complete a complex project is fantastic experience for the industry. It was interesting to see different styles between the developers. It showcased that there will always be various ways to approach and solve the problem. This industry is filled to the brim with opportunities for the astute data scientist, as many companies have a lot of data that they haven't been using to its full potential. With the commercial viability of pre-installed GPS systems in cars and cellphones, being able to analyze and classify that data for travel planning, map services, route management, and other applications. Not only could mail delivery trucks use this classifier to better plan their delivery route, it could help local school buses and public transportation, as well as improve estimated travel time in applications like Google Maps. Overall, this project was a fantastic look at how to apply everything we learned this semester into one project.