GPS Path and Cost Visualization

AbuBakr Ghaznavi
CS 420 Principles of Data Mining
336000513

12/29/2020

Abstract

The purpose of this project was to develop two python programs that could visualize data input provided by a GPS. The first program named GPStoKML was responsible for visualizing the path taken by the GPS receiver. The second program named GPStoCostMap was responsible for reading in a collection of GPS paths and generating a map of hazards using the receiver data. This program read in the coordinates corresponding to a path and ran a series of algorithms to classify certain segments of the path into different hazard types (*left turn, right turn, traffic stop*).

Overview

In order to create these functional programs, several tasks had to be solved. These tasks are: reading and parsing the GPS data, filtering inconsistent and irrelevant data, writing the GPS data into a KML format, and detection of hazards. Additionally, agglomeration of the hazards into individual points had to be accounted for.

Team and Work Balance

I worked on this project alone after struggling to find a team.

Project decomposition

I broke this project down into multiple different tasks and created different functions and objects to represents these class tasks. Firstly, I created classes to read the sentences from the GPS reciever file. I created one for *GPGGA* and one for *GPRMC*. Afterwords, I created an abstract class for both of the previous classes and defined methods for extracting each of the relevant fields. Following this, I then wrote a class that opened and iterated through the lines in the file deciding if the lines fit the GPGGA format or the GPRMC format. This class had a method for iterating through all objects of the GPS file.

I created each of my classes separately and tested them with dummy data that I made up in order to see if the formatting was correct. Then I ran each class on example data on the file to make sure that I did not miss any special cases like two sentences being written to the same line. Afterwords, I tested my classes on the iterator to make sure all the data was read in correctly. Then I tested my KML writer class that took in a data frame and wrote the data in the KML format. Upon doing that and getting the path displaying program to work. I worked on the classification algorithms to and ran the results through my KML writer class in order to visually inspect the results. Then the algorithms responsible for the agglomeration of similarly placed hazards of the same type went through the same development and testing process.

Converting GPS to KML

There were multiple issues in converting the files into KML. The primary one was filtering out the errors in the GPS files. The GPS files had some sentences that were for when the GPS receiver erred. I had to write my iterator to read past these sentences and any malformed or conjoined sentences. Additionally, I decided to pair the GPRMC sentences and the GPGMA sentences together based on the closest timestamps and create a dataframe that held different aspects of their data. No two pairs of sentences had a differing timestamp of more than 200 milliseconds so it felt pretty safe to do this. For example, the GPGGA would provide information on the latitude and longitude of the receiver and the GPRMC would provide information on the speed and angle of motion for the receiver.

After doing this, I had to clean any data where there wasn't significant motion from the beginning of the data and also from the end of the data. Furthermore, any stops that were made had to be reduced into a single location rather than a bunch of points overlapped on top of each other. This meant testing for any lengthy consecutive segments of the data where the speed was under a certain threshold. Using the writer class to write to the KML file was not difficult though.

Stop Detection

The biggest issue with the stop detection classifier was telling if the receiver had stopped at a traffic light as opposed to parking so the driver could run an errand. Achieving this required knowledge about future and past states to know if the driver was picking up speed or losing speed and for how long. In order to do this, I used a one rule classifier inside of an algorithm very similar to a state machine.

I maintained two states (*Stopped* and *Moving*). The initial state was the moving state. I counted the number of data items under a certain speed defined as a parameter and if it was above a certain amount I switched states to the moving state where it would look for both high speed and low speed states. If it counted too many low speed states in a row, then we determined that we were running an errand or doing something other than a stop in traffic. However, If there were enough higher speed counts in sequence while being in the *Stopped* state then we would register this data element as a short traffic stop.

Turn detection

For turn detecting I relied heavily on the speed and the angle of motion in order to determine the speeds. To detect left turns and right turns I defined my own shortest angular distance measure using addition under modulo then subtraction. The angular distance had to account for the fact the angle wraps around 360. Simply subtracting consecutive angles would lead an angle to 359 and 2 to be at a distance of 357. Using my own distance measure, this distance would accurately be interpreted as only 3 degrees. I calculated all of the angular velocities at each point and stored them as a feature named *cross*. I then took the square of the each velocity of while maintaining the sign. This was to ensure that small velocities were further reduced and that higher changes were further magnified.

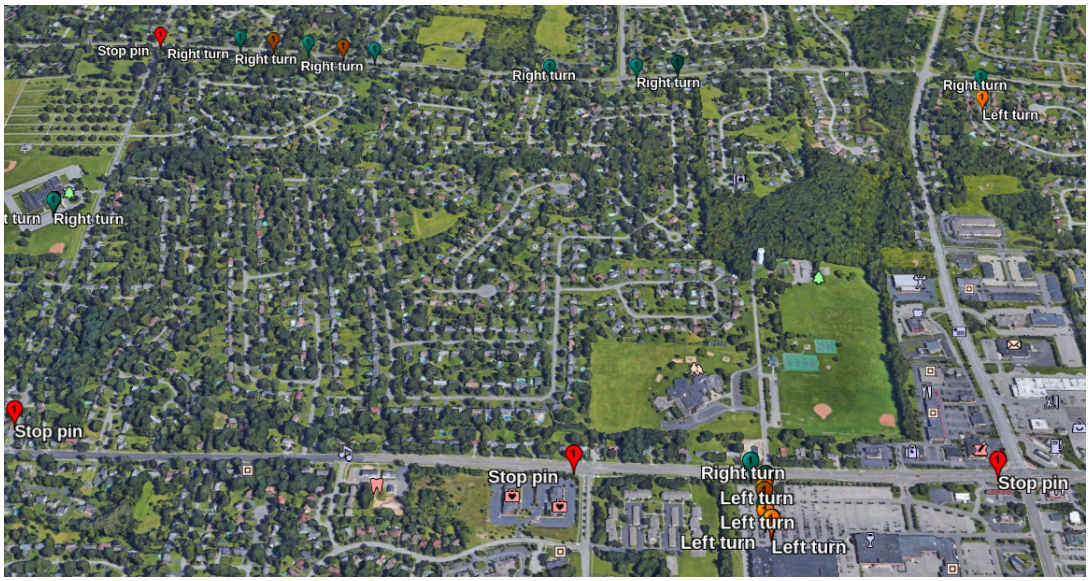
Afterwards, I then used a sliding window approach, taking the means of a data point and its 7 surrounding points and calculating the mean in order to smooth out fluctuations in the data. Then if the velocities were below a negative enough value, I could conclude that it was a right turn. The opposite could be said for a left turn.

Assimilating Across Tracks

I kept track of the longitude and latitude of the points and which class of hazard they are meant to be. I then defined a variable distance parameter, all points of the same class that are sufficiently close enough given this distance are grouped in together. All of my data was stored in a dataframe before being agglomerated. Only the relevant intermediate data was stored.

Path Example

Hazard Example



Discussion

Some problems with my approach are that some slow turns aren't classified as turns due to a higher turning radius. Additionally, some lane changes may be classified as a turns due to the high changes in velocity. This is slightly mitigated due to my sliding window smoothing approach. My counting state machine approach did a good job at filtering out any stops that were too long while including a lot of the stops that it was actually meant to stop.

Lessons learned and Further Applications

Through this product, I learned how to apply data cleaning and filtering techniques in order to remove erroneous and insignificant data. I also learned how to apply smoothing techniques while adjusting the techniques of these parameters in order to classify certain a range of data points. It was a struggle to classify the different points without having any labels to train a classifier with.

These programs could be useful as a commercial application to plan routes based on potential obstacles. If a company wants to plan some logistics for delivery or transportation, having a map of the terrain providing certain hazards would be extremely useful in determining which routes would be the safest or the quickest to accomplish a certain goal.