Model Documentation

This document describes the model for generating paths in a virtual highway environment, with the goal that the car should navigate the track safely, comfortably, and efficiently. To be specific, the map of the highway is given, and periodically telemetry messages are received which contains localization data that provides information about our own location, and sensor fusion data that contains information about other cars on the road. The requirements for safe and legal navigation include no collision, no speeding, and no driving outside lane boundary when not changing lanes. To take limitation of the car and comfort of passengers into consideration, acceleration and jerk should not exceed certain limits. Moreover, when the car is driving behind a slower moving car, it should be able to change to adjacent lane when it is safe to do so.

Thought Process

Get the car moving

The earlier lessons in "Project: Highway Driving" have provided a good jumpstart on tackling this difficult project. I followed the instructions in Getting Started and Highway Map and started putting pieces together to get the car moving at predefined speed and follow the curvature of the road.

To ensure generated waypoints form a smooth path, spline fitting is used to interpolate given anchor points with cubic spline. So, the first step in path generation is to define the anchor points based on previous path and goal location in the near future. In early stage of this project, before lane changing comes into the scene, the goal location would be somewhere further down the road in the same lane as current position of the car. In Frenet coordinates, that means the d value for the goal location would be same as current d value of the car, and s value for the goal location would be greater than current s value of the car. If the latest car position is (s, d) in Frenet frame, then the goal location could be (s+30, d), (s+60, d), etc.

To make the new path a continuity of previous path, the last two points from previous are chosen to be the first two anchor points, and another three anchor points are spaced evenly and far apart from each other further down current lane. When the path planner connects to the simulator for the first time, there is no previous path defined and the car speed is zero. In this case, an imaginary previous location is calculated based on latest car position and heading, and it becomes the first anchor point for spline fitting. The second anchor point chosen is latest car position, and the rest three points are the XY positions converted from Frenet representation (s + SPACING * i, d) using helper method getXY(), where i=1,2,3 and SPACING is a pre-defined constant value. To make spline fitting and sampling easier, a rotation transformation is performed on these anchor points, so that the resulted reference angle is zero, instead of current yaw value of the car. Later after the XY coordinates of the waypoints are obtained, they are transformed back to the original heading. The code for calculating anchor points for spline fitting is defined in main.cpp line 25-65.

Next, after spline fitting is completed using set_points method from spline.h, we need to get discrete waypoints from the continuous cubic spline. The time difference between two consecutive waypoints should be 0.02 second, so the distance between consecutive waypoints is 0.02s * target_speed m/s, where target_speed is the desired speed we want the car to travel when using these waypoints. Then

the total number of samples for a section of spline can be calculated from dividing the distance of the section by the distance between consecutive waypoints, and the increment in X axis for each waypoint can also be calculated from dividing the projection of this spline section onto X axis by number of samples. For each waypoint, once the X coordinate is determined, by passing the X value to the spline object, we can obtain its corresponding Y coordinate, then transform both coordinates back to the original heading using latest yaw value.

Now we have enough samples from the spline, representing path starting from the end of previous path, it would be tempting to just feed these points to the simulator. But we need to remember the first anchor points are the last two points in previous path, meaning if there were more points haven't yet been consumed by the simulator, they need to be copied into the new path to ensure there is no gap between previous and next paths, which helps with smooth transition between consecutive path generation cycles.

Collision Prevention

At this moment, the program is able to make the car driving in its current lane at a speed just below speed limit. It follows the road curve nicely, but there are multiple major problems, and the most burning issue is rear-ending slower moving vehicle ahead. Based on sensor fusion data, we can filter the vehicles on the road by its s and d values. If the d value lies within the range for d values representing current lane, that car is in the same lane as our car. If the s value is greater than that of our car and the difference between the s values is small, it is likely these two cars would collide if the car ahead is slower. So before the generation of new path, the path planner should analyze the relative locations for other cars on the road, and slow down if our car is too close to the car ahead. After our car slows down, the distance between the leading car and our car would eventually increase, and we can speed up again once the distance exceeds a certain threshold.

When deciding how much speed we increase or decrease in each path generation cycle, i.e. 0.02 second, I refer to the project rubric for the max acceleration value 10 m/s2. This translates to speed change of 0.2 m/s per 0.02 s. If we start the car at 0 m/s and increase the speed with maximum acceleration allowed, it would also solve the "max acceleration exceeded" and "max jerk exceeded" issues at the beginning of the drive.

After these updates, the car is able to detect car ahead of itself, slow down until enough distance is put between the two of them, then speed back up until its speed is close to the limit or it gets too close to the car in front of it again. With the added speed control, rear end collision can be avoided as long as emergency braking is not needed, i.e. we assume the car ahead wouldn't stop suddenly.

Since maximum deceleration/acceleration is used for braking and speeding up, the speed of the car decreases and increases like a roller coaster when there is a slow car ahead. It would start braking from 49.5mph till below 20mph, so slow that the car behind it almost rear ended our car, then accelerate back to way over 40mph, which would result in getting too close to the front car again, and the cycle repeats until the front car changes its behavior. To alleviate this problem, I added logic of identifying which car is just ahead of our car and tracking its speed. Then if it is determined that our car is too close to the car ahead, we stop braking when our speed drops to a little slower (2mph slower in the code) than the car ahead. As long as our speed is slower, the distance between the two cars would increase

over time. Additionally, we stop accelerating when the distance to the car ahead drops below 1.5 times the pre-defined safe buffer distance, even if our speed is less than the desirable speed 49.5 mph.

Lane Change

After solving the challenges in safety, comfort, and part of efficiency, we start to consider making the car smarter and more efficient by adding the capability of changing lanes when the situation allows. When iterating through sensor fusion data for other cars on the road, two more boolean flags are added for tracking: can_change_left and can_change_right. The values are initialized based on current lane value: if we are in the left lane, can_change_left is initially set to false. If there is a lane left to our car, and the flag can_change_left is true at this moment, we check the s value for each car in the left lane. If any s value is too close to our car's s value based on pre-defined buffer distance, we set the flag can_change_left to false.

After analyzing all other cars, an update to our car's behavior is needed based on these flags and tracked information. If our car is within safe distance to the car ahead, we need to decide whether we should change lane or stay in current lane and slow down. If there is a lane to our left or right respectively, and it is safe to change lane, we update the lane number for our car. Recall that the first two anchor points are from previous path and last three anchor points are calculated based on s and d values of the goal locations, so changing lane number would result in the fitted spline beginning in current lane and ending up in the target lane, forming a smooth curve. If rear-ending is not a concern, and our car is currently not in the center lane, we need to first think whether we should change back to the original lane. Again, the flags can_change_left and can_change_left need to be checked before settings lane to 1. Lastly, if there is enough distance between the car ahead and our car, and we are driving in the center lane with a speed less than the speed limit, we tell the car to go a little faster.

The judgement of whether a lane change is safe can be a little tricky. We need to predict the "future" s value of a car in the adjacent lane, knowing that the sensor fusion data is from the past due to latency, and the latency can be determined from multiplying 0.02s and number of waypoints in previous path vector. This predicted location is our estimation to current location of that car, and the difference between that prediction and our latest position is compared to a threshold to determine whether it is big enough to allow for a safe lane change. When I first added code for lane change, I incorrectly applied the same calculation for other car to our car, i.e. adding latency x speed to latest position of our car, and that resulted in collision during lane change. The mistake was comparing estimated current location of another car with future location of our car, and the value of that distance does not guarantee there is enough buffer space for a safe lane change. After correcting the mistake, no collision had been observed in lane changes.

By this time, the model for path generation is basically completed and functional.

Tuning Thresholds

In the decision-making process for path generation, we use some pre-defined thresholds to determine whether a flag should be set. For example, BUFFER_DIST is used to check whether our car is too close to the car ahead. Naturally, the choice of thresholds can greatly impact behavior of the car. When lane change was first introduced to the path planner, I observed the path planner decided to follow a slow car in lane 0 even though I didn't see any car in center lane. I clicked and dragged the simulator display

to change view angle, and found a car in the middle lane about 5-6 car lengths behind, driving at about the same speed as our car. From a human driver's perspective, this kind of behavior is overly conservative and results in inefficiency. So, I decreased the threshold for judging whether the distance from a car trailing behind in adjacent lane is big enough for a safe lane change. With this change, the car was able to seize more opportunities for lane change and drives more efficiently without sacrificing safety. Similar tuning has been applied to other thresholds used in path planner.

Model Summary

To sum it up, the path generation process consists of four steps: sensor fusion data processing, behavior planning, spline fitting, and path generation.

Sensor fusion data processing goes through data for all other cars within sensor range on the road, calculate its estimated location, and its difference in s value compared to our car. Then depending on which lane the car is in, the Boolean flags too_close, can_change_left, can_change_right, and other variables being tracked are updated.

After all sensor fusion data has been processed, the collected information is used to determine potential change in behavior of the car, like increase or decrease speed, change to adjacent lane and change back.

Based on the decision in behavior planning, anchor points for spline fitting are generated. Then sampling is performed according to the target speed for the car, and waypoints sampled are appended to previous then passed to the simulator.