CarND-Path-Planning-Project
Self-Driving Car Engineer Nanodegree Program

Daisuke Miyazaki

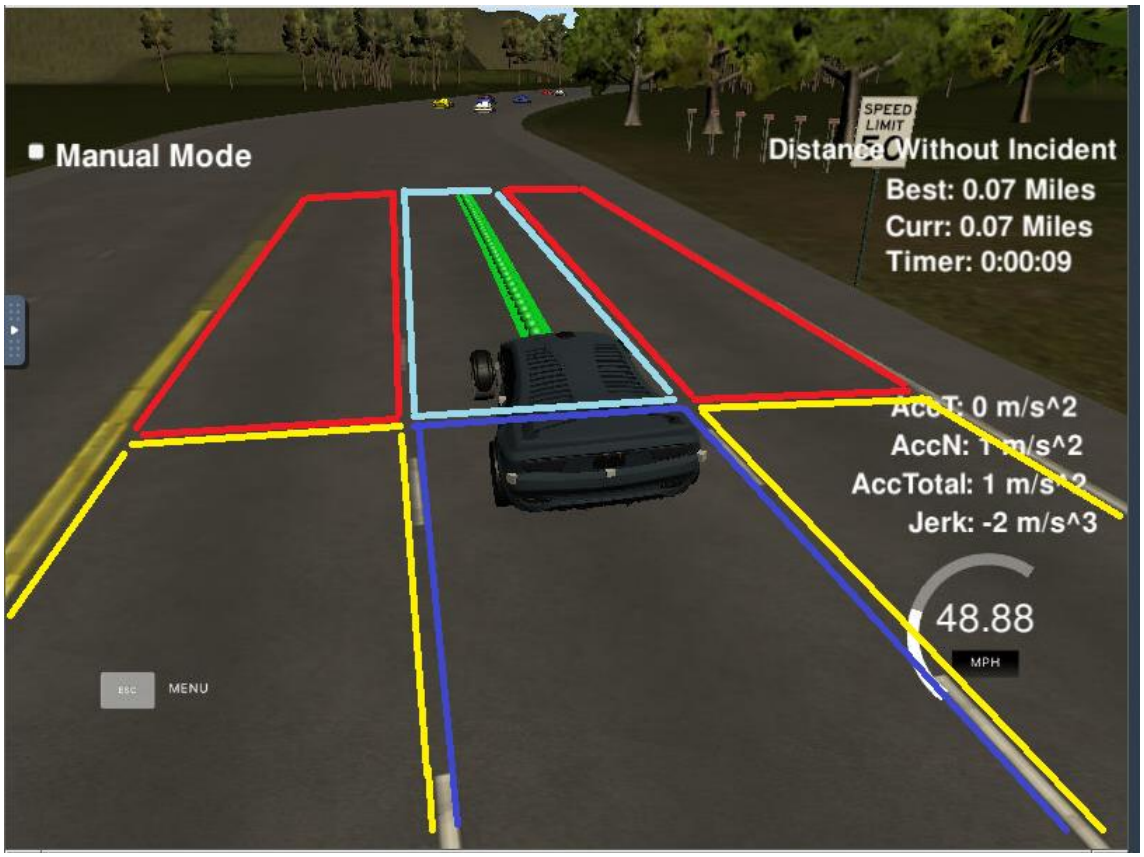
Goals
In this project your goal is to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit. You will be provided the car's localization and sensor fusion data, there is also a sparse map list of waypoints around the highway. The car should try to go as close as possible to the 50 MPH speed limit, which means passing slower traffic when possible, note that other cars will try to change lanes too. The car should avoid hitting other cars at all cost as well as driving inside of the marked road lanes at all times, unless going from one lane to another. The car should be able to make one complete loop around the 6946m highway. Since the car is trying to go 50 MPH, it should take a little over 5 minutes to complete 1 loop. Also the car should not experience total acceleration over 10 m/s^2 and jerk that is greater than 10 m/s^3.

Intro - General strategy

To generate smooth trajectory, I used the path my code previously created to generate the next trajectory. When there's traffic around my car, I use sensor fusion data to take action depending on their behaviors. Otherwise I don't use sensor fusion and my car navigates without violating the limitations.

1, Preparing

Firstly, I prepared 4 sets of vectors for sensor fusion. When generating a trajectory, my vehicle would experience two different scenarios; either running in the empty lanes; or running in the crowded lanes with other cars. As the vehicle encounters other vehicles on its way (which data is given as sensor fusion), it's convenient to classify where these vehicles run. With this, I divide sensor fusion data to 4 different vectors as follows.



The light blue zone; vector<vector<double>> smlane_fr;
The rich blue zone; vector<vector<double>> smlane_bk;
The red zones; vector<vector<double>> othlane_fr;
The yellow zones; vector<vector<double>> othlane_bk;

I divide the vectors based on the predicted locations of my main vehicle and other vehicles in the near future (this *near future* is explained later on section 2-1-2). By taking each vx and vy values and thereby calculating its own speed, I can yield their approximate positions.

To achieve an efficient calculation, I only look at vehicles within 60 meters range from my vehicle. Simply other cars that run outside of the range are all not taken into account during path planning.

So the resulted number of cars that I consider during my path planning is the size of the all 4 vectors combined. This is the integer variable ; cmpt_times.

2. Path planning

I used ref_vel as the speed of my vehicle. This is used to divide the spline into dots and thereby generate the goal trajectory later.

I used the provided code in the Q & A video as a start point of my code.

In principle, this pinpoints 5 different positions in Fernet coordinate. Among the 5 points, the first two are taken by the end of the previous trajectory, and the other 3 generated based on the current car's positions. These 3 points derive from originally from Fernet coordinate and their s values are simply the results of addition, that is, the current car's s coordinate plus 30 meters, 60 meters and 90 meters with respective d value.

When my car doesn't have a previous trajectory, the first two points derive from the current car's position and an estimated adjacent position, that is, on the circumference of a unit circle that centers at the current car's location.

In either case, the reference x and y values are of the second point among 5. As this car has a yaw value, this pipeline uses a rotation matrix by negative yaw and has the orientation fixed throughout the operation. Also taking the gap between the distance of the 5 points and the reference positions, the origin of cartesian coordinate will always remain fixed at the second point of the 5.

Afterwards, it uses spline function[1] which draws a smooth line that passes 5 points. With a certain length of x and y – which is a point on the spline– the code slices the spline into a sequence of dots. These dots are the trajectory of the vehicle, and it takes 0.02 second to travel from one dot to another.

2 -1, When sensor_fusion_switch is true(on).

When there's no car ahead of my car within the range, the car keeps going within the limit of the speed, that is, 50 miles per hour. On the other hand, if there's more than a single car ahead of us within the range -- in another word, if the size of smlane_fr isn't 0 – I use sensor fusion data to take action depending on their behaviors. I defined this as a Boolean flag;

---

[1] http://kluge.in-chemnitz.de/opensource/spline/

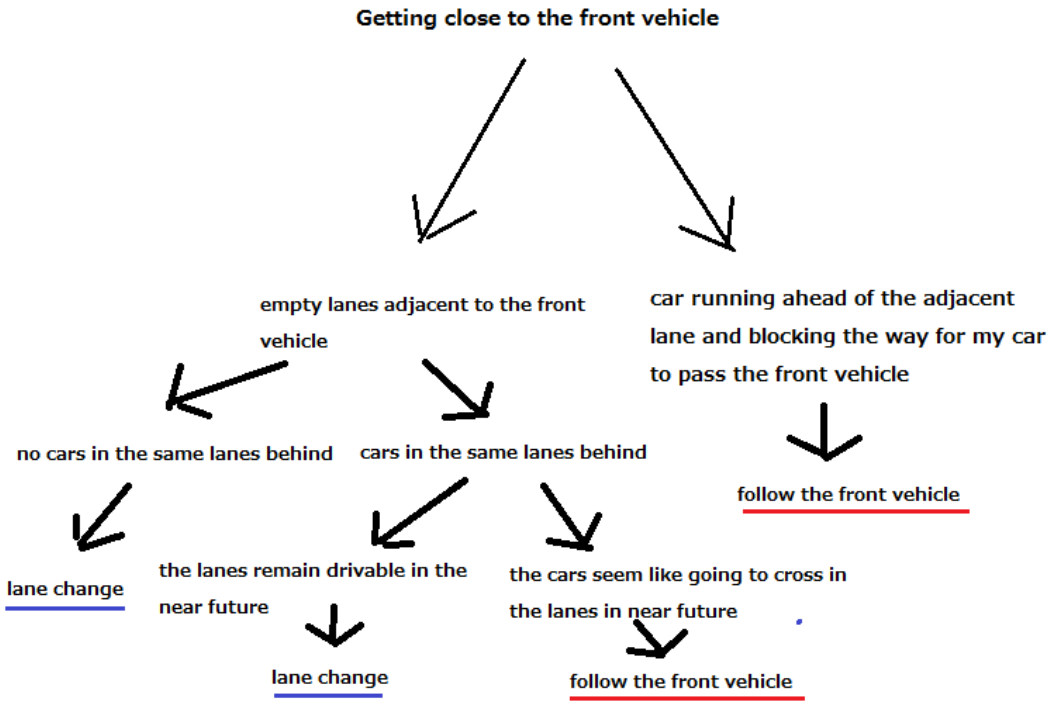
sensor_fusion_switch.

2-1-1. Focus on the front car, measure the speed

Firstly, I check out the distance between all the vehicles ahead of mine, and pick up the nearest car, in another word the front car ahead of mine. I mark it as a front car and calculate its speed and future location based on its speed, timestamp and my previous path size.

I defined the distance between the future locations of my car and the front car as prime_dist. Also, I set the speed of the front vehicle as othref_vel, which turns to the reference velocity of my car while sensor_fusion_switch is on.

2-1-2 If getting close to the front car, consider lane change or follow it



I used a Boolean ready2lanechange flag, which turns to true when it's safe to lane change, otherwise false. Also when lane change is possible, I assigned the number of the available targeted lane to valid_lane variable.

Logic tree

(The graph of this logic tree is shown above)

I can think of 2 cases now. One is to consider a lane change to pass the vehicle. Another one is to follow the front car by adjusting the velocity of my car.

In the former case, I look for adjacent front lanes first. If some car is running ahead of the adjacent lane and blocking the way for my car to pass the front vehicle, it's unwise to lane change via that lane.

On the other hand, in the latter case, if no car runs ahead of the adjacent lane and there's available space to pass the front vehicle, lane change is possible. In this case, I look at the back area on the available lane. This also has 3 possible cases.

First case is where no car is running there, meaning this is safe to lane change on that lane. If there're cars running behind, firstly I calculate their and my car's locations in the near future, based on each current speed and location. Here, *near future* means the timestamp (0.02) * prev_size().

If the location of my car is greater by some margin than the one of other car's behind, then lane change is possible (second case). Otherwise lane change is unsafe (third case). After running my code multiple times, this margin ends up at 70 meters in my code to satisfy the rubrics points.

The way I coded above is as follows. I made two Boolean flags for right and left lane. When lane changing to right, the former flag remains true and the other turns to false, and the other way around for lane chaining to left. When both flags are false, ready2lanechange flag turns to false so my car isn't going to lane change.

As the spline code for generating a path has a lane value, I change the lane value to the valid_lane when lane change happens. Also, in order to avoid lane changing over 2 lanes at once, I created intransiton flag and my code only execute lane change when this flag is false.

Lastly, when the car sits in the 1st lane and both right and left lines are available, my code prioritizes left lane change.

2-1-3 if the car is way too close, heavy deaccelerate to avoid collision.

If suddenly a car cuts into my current lane or the front car brakes, the normal deacceleration

wouldn't be enough to avoid collision. So I made a Boolean flag; hvy_deaccelerate and this turns to true when the future trajectory of my car gets close by a certain margin to the future location of the front vehicle(I ended up setting margin as 10 meters)

2-2, when sensor_fusion_switch is false(off)

My car keeps travelling within all the limitations, such as acceleration, speed and jerk.

3.Taking action

Depending on the flags above, my vehicle takes action to navigate itself in the highway effectively. I changed the rate of the acceleration whether sensor_fusion_switch is on or off. This is because more sensitive acceleration suits – particularly weighting on deacceleration than acceleration – in navigating in traffic. Whereas it should also minimize the time to navigate itself on highway, so whenever there's no vehicles around mine acceleration should be as high as possible within the thresholds.