Udacity ND013 Self-Driving Car Nanodgree

Project 7: Highway Driving Path Planning - Model Documentation

Jiwon Lee

1. Analysis on highway_map.csv file

The map data file is composed of 181 rows of 5 columns, which are <x>, <y>, <s>, <d_x>, <d_y>, waypoints going all the way around the track. These data are stored as <map_waypoints_(var)> in <while(getline(in_map_, line))> loop.

2. State machine of ego car and neighbor cars
    a. Current lane identification

Depending on the lateral position of neighbor cars, <d_near> (0~12, 4 meters for each lane), extracted from <sensor_fusion> vector, the lane of each car, <lane_near> is assigned from 0 to 2. <ego_lane> is the current located lane of the ego car.

    b. Extracting data of neighbor car

From <sensor_fusion> vector, velocity in x direction and y direction, <vx_near> and <vy_near>, respectively, is extracted.

<s_near> contains s coordinate in Frenet coordinate system of neighbor cars. Using the magnitude of velocity calculated from <vx_near> and <vy_near>, <s_near> at the next time step can be also calculated for prediction.

    c. Neighbor car state definition.

There are three bool variables, <is_car_ahead>, <is_car_left>, and <is_car_right> to define the states, which can be determined by considering relative longitudinal s position and relative lateral d position of the ego car and neighbor cars.

For relative d position, pre-calculated lane number is used, by comparing <lane_near> and <ego_lane>.

For relative s position, bit-wise operation between <s_near>, <roi_s_near> (region of interest to consider for s coordinate of neighbor cars), <car_s> (ego car's s position), and <roi_offset> (region of interest offset).

As a result, whether there is a car ahead of the ego car, or there is a car on the right / left side of the ego car is determined.

3. Behavior planning
    a. Acceleration and lane change

Depending on the relative car position between the ego car and neighbor cars, positive or negative acceleration and lane changing action is determined.

If <is_car_ahead> is true, meaning there is a car in front of the ego car, considering the three lane lines, lane changing strategy is determined. If there is no car on the left side, and <ego_lane>, the current lane of the ego car is not the 1st lane right next to the centerline, the next lane to move becomes left lane for passing. Otherwise, the car moves to the right lane for passing. If there are cars on the both sides, the acceleration variable, <velocity_diff> becomes <-ACC_MAX>, meaning braking.

On the other hand, if there is no neighbor cars on the 2nd lane, the ego car will move to the 2nd lane and accelerate to the maximum velocity, considering the highway driving rule that the 1st lane is the passing lane while the 2nd lane is the driving lane.

b. Starting point calculation

Since the ego car position up to now is calculated in terms of Frenet s-d coordinates, it should be converted to Cartesian x-y coordinates. <x_curr>, <y_curr>, and <yaw_curr>, the current x, y, and yaw rate, can be directly taken from the variable <car_x>, <car_y>, and <car_yaw> in localization data if the previous path is almost empty, or if it is a cold start. If there is previous path generated, the current x and y position becomes the end of the previous path, while yaw rate can be calculated from the increment/decrement of x and y position in the previous path, by using <atan2> function. The resulting Then, <x_curr>, <y_curr>, and <yaw_curr> become the starting point of the trajectory to be generated and are stored into <xpts_list> and <ypts_list>.

c. Waypoint generation

The waypoints in cartesian coordinate system is calculated from <car_s>, <2+4*ego_lane> (which is d coordinate), <map_waypoints_s>, <map_waypoints_x>, and <map_aaypoints_y>, using <getXY> helper function. Then, they are added to <xpts_list> and <ypts_list> for trajectory generation, respectively.

4. Trajectory generation

The trajectory points are calculated using <spline> library. The trajectory points are stored in the vectors <next_x_vals> and <next_y_vals>. Firstly, the <spline> object is created, which is <trajectory>. <x_goal> and <y_goal> which is the coordinate of the target point, is calculated with distance to the target <dist_goal>.

In the for loop, while iterating as many times as the size of the previous path, limiting jerk by controlling the acceleration, filling the remaining points between the starting point and goal point, and coordinate transformation from local to global are performed.

For jerk minimization, according to the velocity limit and acceleration limit set at the beginning by #define macro, the velocity of the ego car is determined to be accelerated or deaccelerated by adding <velocity_diff>.

The trajectory points (<x_pt>, <y_pt>), between (<x_curr>, <y_curr>) and (<x_goal>, <y_goal>) are calculated. Through transform matrix calculation, they are transformed by rotating and translating, and finally store in to the vector <next_x_vals> and <next_y_vals>.