Path Planning Project – Self-Driving Car Engineer Nanodegree

Kiran Hegde | December, 2017

Here, I explain how I tackled the various points from the project rubric in my C++ path planner implementation found at <https://github.com/kirnh/CarND-Path-Planning-Project>.

1. The car drives according to the speed limit:
Since we know that the car moves from one waypoint to the next in 0.02 seconds, we can make sure to achieve a desired speed by controlling the spacing between waypoints. With the speed limit being 50 MPH, I set a reference velocity of 49.5 MPH and made sure that the waypoints are never spaced far apart as to cross this limit. [Check line 456 in main.cpp]

2. Max Acceleration and jerk are not Exceeded:
Max jerk or max acceleration might happen in two cases,

(i) When we are accelerating or deccelerating in the current lane:
I avoid this by making sure that the speed being reduced or increased is done so gradually when required, thus limiting the amount of acceleration caused by it. [Check line 439 in main.cpp]

.

(ii) When we are changing lanes: The waypoints that are being pushed to the simulator are obtained by first finding a spline in the required path and then deciding on the spacing of the points. Lane change occurs with the usage of the new lane information while generating the spline. We generate the spline by taking either the previous path's end position or the car's current position and then taking 3 evenly spaced points along the road (in Frenet coordinates) from it and fitting a spline. The smoothness of the spline during lane change depends on how far apart the evenly spaced points are. With a value of 50m of space between them, we get a smooth transition during lane changes without exceeding maximum jerk. [Check line 387 in main.cpp]

3. Car does not have collisions:
With the mathematical modelling approach that we use here, it is only possible to model certain predictable behavious and make sure we don't have collisions.
Some of the prominent situations where we might have collisions are,

(i) When there is a vehicle in front of the ego car in the same lane: With the various information we have, we keep checking if there is any car closer than a certain distance from the ego car in front. If so, we first check if changing lanes is feasible. If so, we do it. Othewise, we reduce our speed so that we don't collide. [Check line 439 in main.cpp]

(ii) When there is a vehicle close to the ego car in the lane we're changing into: With information from sensor fusion module, we check if there are any cars in a certain range of distance from the ego car and decide whether its feasible or not to change lanes. [Check line 274 in main.cpp]

4. The car stays in its lane, except for the time between changing lanes: We know that the car exactly follows the set of waypoints we provide. And the waypoints that we generate come from interpolating the spline we have got by using the lane information, where we only use values of d (in Frenet coordinates) that represent the center of lanes. Because of this, the spline always follows the lane center or follows a path from center-to-center of the lanes during lane change. [Check line 387 in main.cpp]

5. The car is able to change lanes: When there is a need to change lane which is when there is a car too close in the front, we first check whether a change to left side lane is feasible or not before checking the right side with the same logic. We do this with an assumption that left lanes are faster moving and that we prefer moving fast. To check whether the lane change is feasible or not, we check if there are any cars in our lane of preference that are in a ceratin range of distance from the ego car. This avoids any collision. If the preferred lane change is feasible, we change the value of the ‘lane’ variable to whatever decision that was made which reflects in the waypoints being generated. [Check line 304 in main.cpp]