Path Planning Project

Will Tripp - Udacity's Self-Driving Car Engineer Nanodegree - Term 2, Project 2

My path planning project is based on the one shown in the project Q&A.It relies on Frenet coordinates to determine the car's location and how far it is from other vehicles. It also uses a spline to generate a smooth path based on a sparse set of previous waypoints in the car's path and a few points in the future. This enables the car to have a smooth trajectory, with no sudden spikes in jerk or acceleration. The car also can change lanes when the car in front is moving too slow and the lanes it could move into are free.

Collision Avoidance

In highway driving, it is common for the ego vehicle to "catch up" to the car in front of it when that car begins to slow down. The ego vehicle needs to slow down as well to avoid colliding with it. To account for this behavior, my path planner checks how close each car is to the ego vehicles. It checks this distance by using Frenet coordinates.

- For d-coordinates (lateral coordinates, or the coordinates on either side of the ego vehicle), the planner checks whether the car is anywhere within the span of the ego vehicle's current lane.
- For s-coordinates (longitudinal coordinates, of the coordinates in front of or behind the ego vehicle), the planner projects out where the other car will be, based on a velocity magnitude calculation.

If the car is in the vehicle's lane, and if the car is in front of the ego vehicle, and if the gap between the ego vehicle and can in front is less than 30 meters, then the ego vehicle changes behavior.

1. First the ego vehicle executes its lane change logic, which is described in the Lane Change Maneuvers section.
2. If it decides to keep lanes, then it changes speeds, which is described in the Speed Adjustments section.

Lane Change Maneuvers

My path planner uses a simple lane changing logic to move into different lanes based on the surrouding traffic.

The ego vehicle starts in the middle of the three lanes (lane 1). When it gets too close to another vehicle in its current lane, it checks whether there are any cars in the left and right lanes, using the d Frenet coordinates. For these cars in other lanes, if its s-coordinates are within the range of 50 meters behind the car or 100 meters in front of it, then the car keeps its current lane at a reduced speed. If a lane is free of cars in this s-range, then the car changes into that lane.

Speed Adjustments

When the ego vehicle is too close to the car in front but decides not to change lanes, it begins to slow down. It does this by adjusting its reference velocity variable, `ref_vel`.

- At the start of simulation, the reference velocity is 0. In other words, the car begins at a complete stop.
- As simulation continues, and assuming the ego vehicle has no cars in front of it, then the ego vehicle gradually increases its reference velocity by a small amount (0.1 m/s, or 0.224 mph). It continues to increase its reference velocity until it reaches 49.5 mph, which is just below the speed limit of 50 mph.
- If the ego vehicle is too close to the car in front, as described in the Collision Avoidance section, then the ego vehicle, then the ego vehicle decreases its reference velocity by a small amount (0.1 m/s, or 0.224 mph). It continues to decrease its reference velocity until it reaches about 30 mph, which is a safer velocity when another velocity is too close in front of it.

Trajectory Generation Using Splines

To generate the trajectory of the ego vehicle, while minimizing spikes in jerk and acceleration, we first need to generate an initial set of anchor waypoints for the vehicle to follow. For waypoints, the planner selects:

1. Two starting points, i.e., the last few points from the previous path. We then calculate what angle the car was heading in based on those points, and we add those to the list of previous points for the planner's next iteration.
2. Three evenly spaced points at 30 meters, 60, meters, and 90 meters in front of the ego vehicle.

The planner then fits a spline to this sparse set of anchor waypoints to generate the car's trajectory. But first, to make the math easier, the planner shifts the path into the vehicle coordinate system, where the location of the ego vehicle is at (0,0) with yaw 0. Then, the planner can calculate the spline.

To create the spline, I use the spline library (https://kluge.in-chemnitz.de/opensource/spline). I included this library in the planner as a single header file, `spline.h`. To create the spline, the planner:

1. Creates a spline object.
2. Sets the anchor waypoints to the spline.
3. Initializes the next x,y values in the path, starting with feeding in all the previous path points.
4. Finds a point on the spline that is 30 meters out.
5. Calculates the linear distances from the var's origin to that horizon point.
6. Add points along the spline, up to a max of 50 points. For each point, the planner:
    a. Calulates the number of spline points for this linear distance, based on the car traveling at its reference velocity.

b. Gets the next x value based on the number of spline points.
c. Calculates the spline at this point.
d. Rotates the point back to its original roation
e. Puts the calculated point on the list of next points.