Ayham Zaza

Path Planning Project

Basic Definitions:

- **Route Planning:** is a high level path of the vehicle between two points on the map.
- **Prediction**: Identifies which maneuver other objects on the road might take.
- **Behavior Planning**: Decides what maneuver our vehicle should take.
- **Trajectory Generation**: Plots the precise path we like our vehicle to follow.
- **Frenet Coordinate Transformation**: defines the mapping from Cartesian (x,y) domain into (s,d) domain where **s** represents motion along the road or the longitudinal motion and **d** is the side to side motion or the lateral motion.

Model Description:

Our localization data are first read. These include our x,y,s,d positions as well as car angle (yaw) and its speed. We also read sensor fusion data the reflect information on all other cars around us. The trajectory that our car will follow will be generated by two vectors *next_x_vals* & *next_y_vals* that got updated based on (1) how far future points are separated *dist_inc* that directly controls our target speed *(50 miles/hr)* (2) the car angle.

Since the lanes are 4 points wide, one can reference each side of the lane by defining the boundary for each Left, Center, and Right as (0,4), (4,8), (8,12), respectively and link that to d through *lane* variable. Without loss of generality, d=6 means the car in the center lane and it is in the middle of that lane.

Since we are having a discretized trajectory, there is going to be jumps in both acceleration and jerk at the boundary of each point. For that reason, spline interpolation is utilized to counter this effect and to smooth out the results by using previous path points that are tangent to car angle. Shift and rotation transformations were made when needed to keep the car motion consistent. This transformation is detailed in the code. The following figure[1] explains how the interpolation was done while keeping the target speed maintained. The points are linearized by looking at a horizon value (30 m), and check where that point lies on the spline. Next calculated the distance from our car to that point and call it d. Next split d into N pieces while considering the desired velocity. i.e. N*time* velocity = target y point on the spline. From d, map the resulting points to their x correspondence and finally obtain the smoothed interpolation on our trajectory from the spline. The details of the process are well documented in the code.

1 Thanks to artist who made this available Aaron Brown. 😊

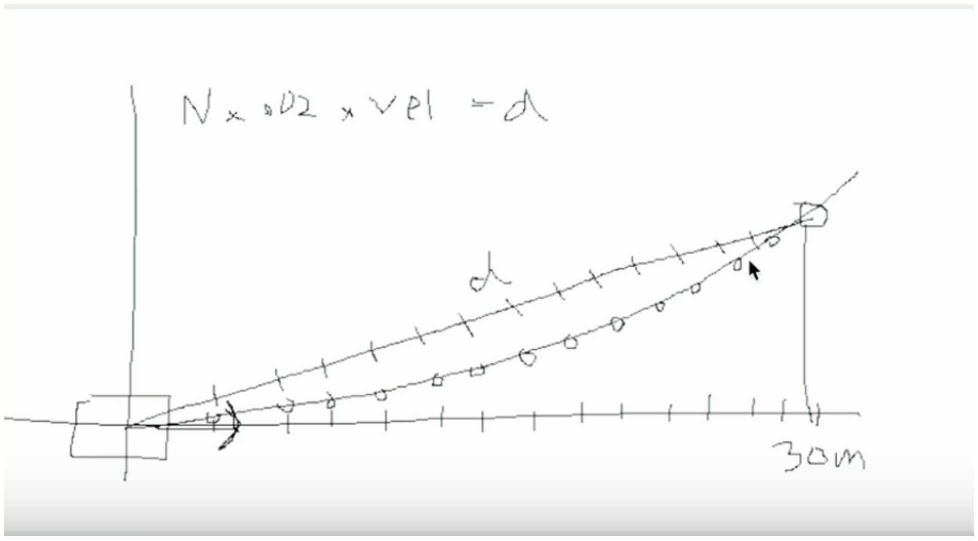

Figure 1: Explanatory graph that illustrates the interpolation and mapping, made by Aaron Brown

To avoid hitting a car, sensor fusion data of other cars has to be utilized. After looping through all possible cars and if the car lies in the same lane of our car and its distance is not far away then a flag of changing lanes should be set. The decision on the distance to other cars is made after estimating the other cars speed. Both the **d** & the **s** values of the other cars play a central role here. While the d value tells which lane the car is in, s will track the distance to them. Lane change is considered when it is safe, i.e. not hitting other cars, otherwise, our car should lower its speed. To make change in velocity smooth, a constant (~5m/s^2) value is subtracted or incremented when needed. The constant is chosen to minimize the annoying change in acceleration felt by passengers. A lane is included as a variable in the interpolated points. For that reason and in case of lane changes, the process will be also smooth.

To offer safe maneuver, both the lane of my car and other cars are accounted for. My car can be in one of the 3 lanes, and other cars could also be in one of the 3 lanes. For that reason, a difference matrix was computed between my current lane and other possible care lanes as follows:

	Other Cars Lanes			
My Lane		**0**	**1**	**2**
	0	0	-1	-2
	1	1	0	-1
	2	2	1	0

Wherever the difference between my lane and other lanes is zero, then it means I am sharing the same lane with the other detected object. In that case, a check is performed if that car is close by. If it is, then change lane flag is set to be later analyzed. When the difference is -1, it means my lane is either {0 or 1} and in both cases there is a car on my right. If the difference is 1, it means my lane is either {1 or 2} and there are cars to my left. Cases of {2 or -2} are considered as don't care; they can be utilized in providing future heuristic.

In case a car was in front of me, select my lane in a switch statement and the necessary action is set in case of changing lanes is needed. For example and without loss of generality, if I am in the center, case {1}, and I need to change lane, I can go right if cars in lane 2 are far away or lane 0 is empty. If this was not possible, a check to the left lane is done in a similar way. If that aslo was not possible, care slows down its speed.

More details could be found in the code and its associated intensive comments.

References

- Udacity Self-Driving Car ND, class room material, Udacity forum & Project walk through.