HIGHWAY DRIVING

GOALS

- In this project our goal is to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit.
- We have 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.

INTRODUCTION

The map of the highway is in data/highway_map.txt

Each waypoint in the list contains [x,y,s,dx,dy] values. x and y are the waypoint's map coordinate position, the s value is the distance along the road to get to that waypoint in meters, the dx and dy values define the unit normal vector pointing outward of the highway loop.

The highway's waypoints loop around so the frenet s value, distance along the road, goes from 0 to 6945.554.

Main car's localization Data (No Noise)

["x"] The car's x position in map coordinates

["y"] The car's y position in map coordinates

["s"] The car's s position in frenet coordinates

["d"] The car's d position in frenet coordinates

["yaw"] The car's yaw angle in the map

["speed"] The car's speed in MPH

Previous path data given to the Planner

//Note: Return the previous list but with processed points removed, can be a nice tool to show how far along

the path has processed since last time.

["previous_path_x"] The previous list of x points previously given to the simulator

["previous_path_y"] The previous list of y points previously given to the simulator

Previous path's end s and d values

["end_path_s"] The previous list's last point's frenet s value

["end_path_d"] The previous list's last point's frenet d value

Sensor Fusion Data, a list of all other car's attributes on the same side of the road. (No Noise)

["sensor_fusion"] A 2d vector of cars and then that car's [car's unique ID, car's x position in map coordinates, car's y position in map coordinates, car's x velocity in m/s, car's y velocity in m/s, car's s position in frenet coordinates, car's d position in frenet coordinates.

BEHAVIOURAL PLANNING:

We have used finite state machine to determine when car can change the lane, slow down or speed up.

Below is summary of finite state machine.

1. Check whether there is car in front of the target car (used frenet coordinate to decide the distance between the car)
2. If yes then se the flag car_front.
3. Check cars position in neighboring lanes and set the flag car_left, car_right accordingly.
4. Based on the flags either slow down, change lane to right or change lane to left.
5. If there is no car in front and speed = MAX_SPEED then accelerate.

TRAJECTORY GENERATION

To start with we have used **previous_path_x and previous_path_y** vector to get list of previous x and y given to the simulator. When the car starts it has very less not of previous point, hence we have generated more points which is tangent to the current direction of moving car, if we have large list of previous point we have used them for trajectory generation.

For smooth transition we have used previous path points which are left over from previous cycle and added new points to fill the path, i.e if only 20 points were traversed by car in last cycle we will add only 20 new points and remaining 30 we will add as it is.

Spline library

Spline is a library used to add the waypoints so that the car can run smooth with minimum jerk. This library is better than polynomial fit. Spline ensures that path generated should pass through every point

Working of trajectory generation.

1. To start the trajectory of the car we have taken an anchor vector this consist of
   a. 2 points from previous path, 3 points placed 30m apart from each other
   b. We converted this point to local car co-ordinates.
2. Given this point to spline to create polynomial.
3. To get the desired speed and we broke spline, below is the math for it.

   We considered path generated by spline as hypotenuse of a right angled triangle and taken 30m space on x-axis.

   To get the corresponding y we used spline, we know x, give it to spline it will give y.

   Then we have calculated hypotenuse such that it's split into N points, using Pythagoras theorem.

Time taken to travel from one point to another = 20 millisecond(given)

Velocity is given by us (max is 25m/sec) = ref_vel

In MPH = ref_vel / 2.24 MPH

Distance between two points = 0.002 seconds * ref_vel

hypotenuse(total distance) = N * 0.002 seconds * ref_vel

This is how we linearize the spline into N points

4. We start with x = 0, and keep adding the value of x for each remaining point.
   Distance between two points in x = 30m / N. This distance could change in each loop due to change in the value of N.
   x = x + 30m /N
   y = s(x)
5. Convert these x, y points back to global co-ordinates
6. Finally push these (x, y) points to the final path planner vector, next_x_vals, next_y_vals.