HIGHWAY DRIVING

The goal of this project is to:

Build a path planner that creates smooth, safe trajectories for the car to follow. For this:

1. The car should drive along the centre of the track smoothly, without jerk.
2. Not collide any other vehicle
3. Change lanes under suitable conditions.
4. Should not exceed speed of 50MPH

This project implementation can found in the following link: [HighwayDriving](https://github.com/anjanarajam/CarND-Path-Planning-Project)

**INTRODUCTION**

The track consists of three lanes. The lanes are left lane, middle lane and right lane. The car is positioned at the centre of the middle lane initially. There are about 180 waypoints along the track .Car moves from one waypoint to another every 20 milliseconds.

The track is about 6945.45m long.

The code provides information to the simulator one cycle behind, which means the previous cycle of the code is the current cycle of the simulator. Any path that simulator receives will be of the previous cycle.

The inputs to the code are :

1. The map data which consists of x, y, s and d values from map file. S = 0 is the first way point. D = 1 , is perpendicular to the direction of the right lane. The d value of each lane is 4m.
2. Simulator data which consists of
3. Localization data : x, y, s, d and yaw rate of the ego vehicle
4. Previous path data
5. Sensor Fusion data : x, y, s , d and velocity of the surrounding target vehicles in a given cycle. Sensor fusion data is important, so that it does not crash onto other cars travelling at different speeds and can change lanes.

**PREDICTION**

This is where sensor fusion comes into play. This data is given by the simulator, which is actually one cycle behind for the code. In our project, we use prediction to find the how far the other cars are from ego vehicle, or in which lane they are in.

**Avoid collision**

If we want to avoid collision, we should go through the sensor fusion list and check if any car is close. If the car is close, the velocity of the car is reduced.

**Avoid cold start**

The vehicle should pick up the speed incrementally up to 50MPH. For this we slowly increment the velocity by 0.224 m/sec, starting from 0, every cycle. This would be useful when we are calculating every path point in a loop, because change in velocity will change the distance between two points in every loop. This makes the car movement efficient.

**Lane change**

With the value of ‘d’ of every car from sensor fusion, can tell us in which lane the other cars are. Accordingly we can set the flags, as to whether other car is in left, right or in the same lane.

**BEHAVIOR PLANNING**

Using finite state machines, we can decide under what conditions, the car should change lane or

change its velocity. It uses the data to determine the conditions such as:

1. If the target car is in the same lane and close, reduce the velocity.
2. If no car is left or right, but a car is ahead of us, change to either left or right lane.
3. If the ego car is in left or right lane, and no car is in middle lane, change the ego vehicle lane back to middle lane.
4. If there is no car in front of ego vehicle, increase its speed up to 50MPH.

**PATH PLANNER / TRAJECTORY GENERATION**

Path planner is an output list of x and y points in global map co-ordinates. Collection of these points form a trajectory. Our goal is to fill up these points for the car to move.

**Frenet co-ordinates**

Frenetco-ordinates (s and d) are used to calculate the final (x, y) points for the car to move along the track. This is because the d parameter, which is the length of the lane, helps car stay in track. If we use (x, y) points directly for calculation, the car would go off the track and bump somewhere.

**Speed of the car between points**

The speed at which the car travels between two waypoints is 20 milliseconds which means 50MPH or the car should cover 50 waypoints in a second.

**Previous path points for smooth transition**

It would be unwise if we calculate 50 new points for creating path every cycle. Instead, we take previous path points which are left out from the previous cycle(not all points get used up by the car in any cycle), let us say 30 points. We take those points and add remaining 20 points in the current cycle to add it up to 50 points. This ensures smooth transition.

**Spline points for smoothness**

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 can also be defined as the piece wise polynomial function, where pieces are tied together. These pieces are called knots. Spline points are for initializing spline calculation so that future points can be calculated along a trajectory of (x, y) points.

**Path planner vector**

We create two set of vectors. One set of vector is for the main path planner(next\_x\_vals, next\_y\_vals) that adds points for the car to move along. And another vector, let’s say a helper or anchor vector(points\_x, points\_y), where we add five points, to help calculate points for the main path planner vector with the help of spline library.

*Anchor/Helper Vector*

To begin with the helper vector, we check if the previous path points are available.

1. If there are, then add two points to points\_x and points\_y. Those two points are the previous lane’s last two points.
2. If there are no previous points, add the car’s current path’s two points, one is the where is the car positioned and the other is its previous point, which could be traced back with the help of yaw rate.

Now, there are 3 more points to be added. These 3 points are calculated in (x, y) with wide spacing, say 30 meters each, with the help of car’s ‘s’ and ‘d’ co-ordinates, and the lane of the car. This lane of the car helps us interpolating the path points even when the lane changes.

Once we get these final five points, we would be converting them to local co-ordinates, so that the last point in the previous path is origin (0,0).

We set this set of vector to the spline using function set\_points(). Now these become spline points.

*Path Planner Vector*

We start by filling this vector by previous path points. The remaining points are what we need to calculate. This is done by splitting the 30m spacing of the spline points of the anchor vector, into N points.

For this splitting, we need to linearize the spline points which are curved due to the yaw rate of the car. This calculation requires a bit of maths.

1. The 30m space is the x-axis of a triangle.
2. The y axis is calculated using spline library. This is where the anchor vector helps us.

y = s(x)

1. We can calculate hypotenuse using Pythagoras theorem.

Hypotenuse = sqrt((x \* x) + (y \* y))

1. We can also calculate hypotenuse if it’s split into N points. Its s follows:

- 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

1. Therefore, the number of N points = Hypotenuse / 0.002 seconds \* ref\_vel. Since the number of N points are calculated in a loop of remining path points, it could keep changing due to the change in velocity in each loop. As the velocity increases, N points reduces and the distance between two points increases.
2. Now that we know every point on the hypotenuse, we can use the same N points to split x – axis. If we know x - points, we can know y - points using spline library. This is how we linearize the curved spline points.
3. Now, we start with x = 0 and keep adding the values of the x for each remaining path points.
   * 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)
4. Convert these x, y points back to global co-ordinates
5. Keep adding them to initial value of the position of the car, which could be starting position of the car if there is no previous lane points or the end point of the previous lane if there is. These starting points are obtained from localization data.
6. Finally push these (x, y) points to the final path planner vector, next\_x\_vals, next\_y\_vals.

**FINAL VIDEO OUTPUT**