Detailed Implementation of the Project:

Project Implementation include usage of a spline function, which fits a line to given points in a fairly smooth function. The input to the spline function are 2 lists of x and y points which are calculated as -If the previous path has less than 2 points left , then use current car x and y to calculate previous car x & y and add to ptsx and ptsy list.If previous path has enough points add the last 2 points to the ptsx and ptsy list and then predict future waypoints at distances of 30, 60 and 90m and add to points list.

once the ptsx and ptsy points are calculated and added, they are converted to car co-ordinates from map co-ordinates using translation and rotation. Then these points are fit to a spline function to obtain the trajectory. After fitting a line, I then feed points along that line back into the simulator.

In main.cpp, after the data from the simulator is pulled in, I first check whether all the points from the already generated path is used up or not. If it is the first time around, this means I have to initiate my points to later be used in a spline function to be equal to where the car is right now . If there is a remaining points in the path, this will get pushed back into the x and y coordinates to be fed the simulator again. Additionally, I calculate the x, y, yaw and velocity based off the end values of the remaining path, in order to have the behavior planner begin from the end of the old path .

For avoiding the ego vehicle to accelerate beyond the desired limit,I have initialized the reference velocity to zero initially and have gradually incremented and decremented the reference velocity. Also for collision I have checked whether the distance between the car in front of us and our car is less than 30 m,(and there is no possibility of lane change)if yes, the speed of the car is gradually decreased

double check\_car\_s = sensor\_fusion[i][5];

                    check\_car\_s+= ((double)prev\_size\*0.02\*check\_speed);

                    // check if the car within 30m in front of us

                    if(check\_car\_s>car\_s && check\_car\_s-car\_s<30){

too\_close = true;

if(too\_close){

                ref\_vel-=.224;

            }

            else if(ref\_vel <49.5){

                ref\_vel+=0.224;

            }

The lane change logic was quite simple, if there was a car in front of the ego vehicle then it would see if it was safe to change to the left lane or right lane-if a vehicle is detected in a 30m range in front, I check for a left or right lane present and is within the road boundaries. Safety is taken into account by checking that no vehicle is present in left (or right) lane in +/- 30m range., which seems to be sufficient distance for smooth lane change. If both lanes are safe to take, I choose the left lane .