PATH PLANNER PROJECT REFLECTION

(https://github.com/udacity/CarND-Path-Planning-Project)

Goal:

The goal in this project is to build a path planner that is creates smooth, safe trajectories for the car to follow in highway.

- The highway track has other vehicles, all going different speeds, but approximately obeying the 50 MPH speed limit.
- The car transmits its location, along with its sensor fusion data, which estimates the location of all the vehicles on the same side of the road.

Requirements:

Path planner should be designed in such a way that:

1. It travels 6945.554 meters around without any crash
2. Maintains speed less than 50 MPH
3. Changes lane whenever required but with low acceleration and minimum jerk
4. Planner should output a list of x and y global map coordinates.

We are provided with:

1. Udacity simulator (https://github.com/udacity/self-driving-car-sim/releases/tag/T3_v1.2)
2. higway_map.csv (containing list of waypoints)

The **simulator** provides us with current position and velocity of our car and traffic at each time step back and in turn we send to the simulator the next x and y positions so it can drive the car there. The goal is to drive the car successfully avoiding any collisions, safely changing lanes, staying below speed limit but not too slow and minimizing jerk on the passengers.

From the simulator, we get data for other vehicle in the simulator. This data includes the car_id, car_position(x and y), car_velocity (vx and vy), car_s (distance along the lane) and car_d (distance along the width of the lane)..

Inside data/highway_map.csv there is a list of waypoints that go all the way around the track. The track contains a total of 181 waypoints, with the last waypoint mapping back around to the first.

- The waypoints are in the middle of the double-yellow dividing line in the center of the highway.
- Every 20 ms the car moves to the next point on the list. The car's new rotation becomes the line between the previous waypoint and the car's new location.
- The velocity of the car depends on the spacing of the points because the car moves to a new waypoint every 20ms, the larger the spacing between points, the faster the car will travel. The speed goal is to have the car traveling at (but not above) the 50 MPH speed limit as often as possible. But there will be times when traffic gets in the way.

Walkthrough:

Step 1: Get our car's localization data from simulator/previously generated paths:

Code:

```
double car_x = j[1]["x"];
double car_y = j[1]["y"];
double car_s = j[1]["s"];
double car_d = j[1]["d"];
double car_yaw = j[1]["yaw"];
double car_speed = j[1]["speed"];

// Previous path data given to the Planner
auto previous_path_x = j[1]["previous_path_x"];
auto previous_path_y = j[1]["previous_path_y"];

// Previous path's end s and d values
double end_path_s = j[1]["end_path_s"];
double end_path_d = j[1]["end_path_d"];
```

where:
x,y = global map position
s = distance along the direction of the road (in Frenet)
d = points perpendicular to the road in the direction of the right-hand side of the road. The d vector can be used to calculate lane positions.

previous_path_x = x value of previously followed path
previous_path_y = y value of previously followed path
end_path_s = s value of previously followed path
end_path_d = d value of previously followed path

Part 2: Sensor fusion data of other cars

The sensor_fusion variable contains all the information about the cars in the road.

The data format for each car is: [id, x, y, vx, vy, s, d]. The id is a unique identifier for that car. The x, y values are in global map coordinates, and the vx, vy values are the velocity components, also in reference to the global map. Finally s and d are the Frenet coordinates for that car.

Code:

```
float d = sensor_fusion[i][6];
double s = sensor_fusion[i][5];
double vx = sensor_fusion[i][3];
double vy = sensor_fusion[i][4];
double check_speed = sqrt(vx*vx+vy*vy);
double check_car_s = sensor_fusion[i][5];
```

Part 3: Check where the surrounding cars will be in future.
If you were to assume that the tracked car kept moving along the road, then its future predicted Frenet s value will be its current s value plus its (transformed) total velocity (m/s) multiplied by the time elapsed into the future (s).

Code:

```
check_car_s += double(prev_size)*0.02*check_speed;
```

Part 4: Check if is safe to switch lanes:
Check the position of car's lane and position (s) of other cars in new lane and determine if its safe to switch.

Regarding 'd' value:

If you want to be in the left lane at some waypoint just add the waypoint's (x,y) coordinates with the d vector multiplied by 2. Since the lane is 4 m wide, the middle of the left lane (the lane closest to the double-yellow diving line) is 2 m from the waypoint. If you would like to be in the middle lane, add the waypoint's coordinates to the d vector multiplied by 6 = (2+4), since the center of the middle lane is 4 m from the center of the left lane, which is itself 2 m from the double-yellow diving line and the waypoints.

Check if left lane is clear:
Code:

```
if((d>0) && (d<2+4*lane-2) && (abs(check_car_s-end_path_s)<30))
  {
     safe_to_change_left = false;
     std::cout<< "d: " << d << endl;
     std::cout<< "s diff: " << s - car_s<< endl;
     std::cout<< "Do not switch to left lane" << endl;
  }
```

Check if right lane is clear:
Code:

```
if((d>0) && (d>2+4*lane+2) && (abs(check_car_s-end_path_s)<30))
  {
     safe_to_change_right = false;
     std::cout<< "d: " << d << endl;
     std::cout<< "s diff: " << s - car_s<< endl;
     std::cout<< "Do not switch to right lane" << endl;
  }
```

Step 5: Change lanes

Case 1:
If it's safe to shift to left lane, then set 'lane' variable to 'lane-1' else if it's safe to shift to right lane then set it to 'lane+1'

Case 2:
If it's not safe to shift to either of the lane and if there is a car in front - then reduce speed so that we won't bump into the car.

```
target_speed -= acc;
```

Case 3:
If there is no car in front of us then increase speed.

Step 6: Waypoints and trajectory generation:
To estimate the location of points between the known waypoints, you will need to "interpolate" the position of those points. We will use code very much similar to what was discussed in class and walk through (ref: https://www.youtube.com/watch?v=7sI3VHFPP0w):

1. Calculate the previous position of the car (which is tangent to angle of the car):

```
double prev_car_x = car_x - cos(car_yaw);
double prev_car_y = car_y - sin(car_yaw);
```

2. In Frenet add evenly 30m spaced points ahead of the starting reference

```
vector<double> next_wp0 =
getXY(car_s+30,(2+4*lane),map_waypoints_s,map_waypoints_x,map_waypoints_y);
```

3. Using **spline** [http://kluge.in-chemnitz.de/opensource/spline/] - build future path from starting points x,y

- Calculate how to break spline points so as to travel at required ref velocity and fill up the remaining of path planner after filling it with previous points. [Imagine a right angled triangle, with target_x, target_y and target_dist as its axis].

Code:

```
// Calculate how to break spline points so as to travel at required ref velocity
double target_x = 30; // Horizon value - we will use to point where does our point lies in Spline
double target_y = s(target_x);
double target_dist = distance(target_x,target_y,0,0); // Distance from the car to the target point

double x_add_on = 0;

// Fill up the remaining of the path planner after filling it with previous points
for (int i=1; i<= 50-previous_path_x.size(); i++)
{
   double N = target_dist/(0.02*target_speed/2.24);
   double x_point = x_add_on + (target_x)/N;
   double y_point = s(x_point);

   x_add_on = x_point;

   double x_ref = x_point;
```

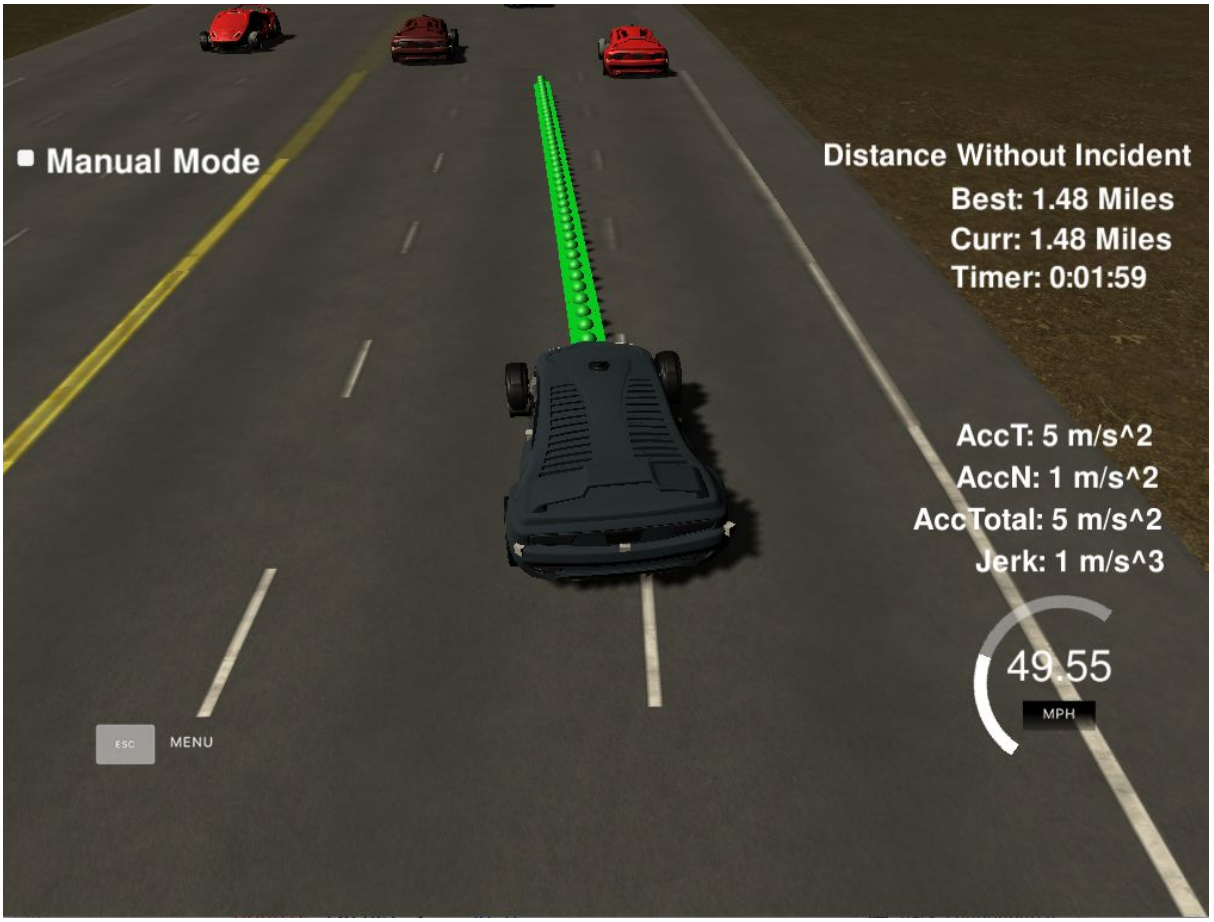
```
            double y_ref = y_point;
            // Rotate back to normal coordinates - convert to global co-ordinates
            x_point = x_ref*cos(ref_yaw)-y_ref*sin(ref_yaw);
            y_point = x_ref*sin(ref_yaw)+y_ref*cos(ref_yaw);

            x_point += ref_x;
            y_point += ref_y;

            next_x_vals.push_back(x_point);
            next_y_vals.push_back(y_point);
        }
```

Step 7: Output
Planner outputs list of x (next_x_vals) and y (next_y_vals) global map coordinates which is used to control the car around the highway.

Future/pending work:

1. Use of cost functions
2. Behavioral planning of other cars

Screenshot:



Video:
https://youtu.be/gYuh0CbF2o4

References:

https://github.com/udacity/CarND-Path-Planning-Project

https://medium.com/towards-data-science/planning-the-path-for-a-self-driving-car-on-a-highway-7134fddd8707

https://medium.com/self-driving-car-bites/path-planning-prediction-a4659ecec603

https://medium.com/@mithi/reflections-on-designing-a-virtual-highway-path-planner-part-1-3-937259164650