Project 11 – Path Planning

Source Code	https://github.com/shyam-jagannathan/CarND-Path-Planning-Project
Video - Successful run	https://youtu.be/BAi9EQ9_SoM
Write-up	"Model Documentation.pdf"

In this project, we are required to drive a car (aka Ego vehicle) around a simulated highway of 4.32 miles within a speed limit of 50mph. The highway comprises of 3 lanes in each direction and each lane is 4m apart. Other vehicles are randomly generated by the simulator driving in different lanes and in different speeds. The goal of this project is to successfully maneuver the Ego vehicle around the circuit without any incidents such as collision, speed limit etc. as indicated in the project rubric. The various criteria and how it's handled are discussed below.

The implementation comprises of mainly two stages,

1. Trajectory generation stage
2. Behavior planning stage

Prediction of other vehicle trajectories and implementing a controller (PID or MPC) is not attempted. These are topics of further improvement of the project.

1. Trajectory generation stage

The trajectory generation stage is similar to the approach shown in project walk through video by Aron and David.

Moving the Ego car

This involves taking care of below criteria,

a. Drive within a speed limit of 50mph
b. Total acceleration not to exceed 10m/s^2
c. Jerk limits of 10m/s^3
d. Car stays within lane and on the right side of the road.

The maximum velocity of the car is set as 49.5mph which is as fast as the Ego vehicle can go without breaking the speed limit of 50mph. But just setting the speed as 49.5mph will have a starting problem where the car is expected to go from 0mph to 49.5 in time T. This will exceed the jerk limits of 10/m^3.

The car slowly accelerates at every time step at the rate of 50mph which is equivalent to 22.4 meters/sec

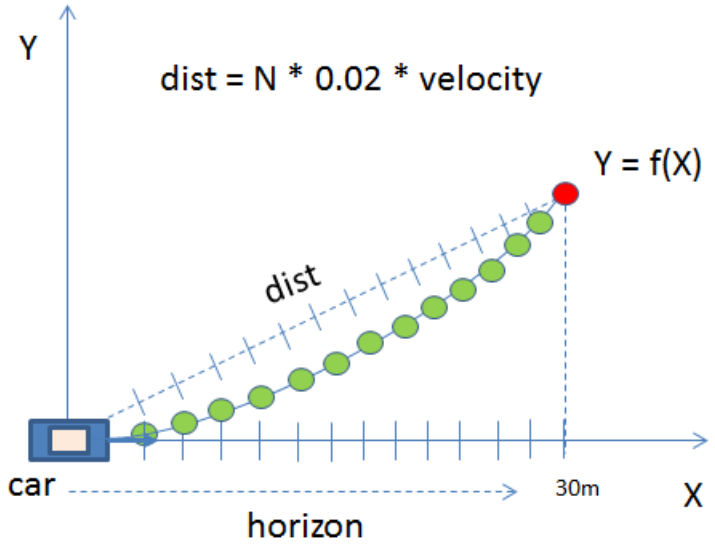
```
353             if(ref_velocity < max_velocity)
354             {
355               ref_velocity += (0.224 * 2);
356             }
357             else
358             {
359               ref_velocity -= (0.224 * 2);
360             }
```

If the reference velocity of the Ego vehicle is greater than the maximum allowed velocity then decelerate by the same amount.

For a jerk free trajectory the spline method is used to fit a 5th degree polynomial using map way points. Way points from car present location to projected locations at 30, 60 and 90 meters ahead on the road. Notice that the d value also incorporates the lane value. (2 + 4*lane)

```
651             vector<double> next_wp0 = getXY(car_s + 30, (2 + 4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
652             vector<double> next_wp1 = getXY(car_s + 60, (2 + 4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
653             vector<double> next_wp2 = getXY(car_s + 90, (2 + 4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
```

The car's Fernet co-ordinates are converted to global X/Y co-ordinates and translated local X/Y coordinates which is supplied to tk::spline function. This helps translation from global to local X/Y co-ordinates, helps in re-projecting the Ego vehicles trajectory points once the spline is computed. As shown in the graph below it is easy to read the Y co-ordinates on the curve by simply sampling the X values.



While transforming back and forth between global and local co-ordinates, perform translation followed by rotation. This is implemented as shown below,

```
687                 double target_x = 30.0;
688                 double target_y = s(target_x);
689
690                 double target_dist = sqrt((target_x * target_x) + (target_y * target_y));
```

```
691             double N = target_dist/(0.02 * ref_velocity / 2.24);
692             double delta_x = target_x / N;
693
694             double new_x = 0.0;
695             double new_y = 0.0;
696
697             for(int i = 0; i < (50 - prev_path_size); i++)
698             {
699               new_x = new_x + delta_x;
700               new_y = s(new_x);
701
702               double out_x = ref_x + (new_x * cos(ref_yaw)) - (new_y * sin(ref_yaw));
703               double out_y = ref_y + (new_x * sin(ref_yaw)) + (new_y * cos(ref_yaw));
704
705               next_x_vals.push_back(out_x);
706               next_y_vals.push_back(out_y);
707             }
```

In order to make for a smooth trajectory, add new points on top of previous trajectory. The simulator ensures that it returns a previous trajectory after removing points that are consumed. Compute the yaw of the car by considering the last two points in the previous trajectory. If the car's current yaw it considered the jerk limits may be violated while turning.

Behavior Planning

The behavior planning module helps in generating lane-changing or lane-keeping trajectories. This is a critical module which will keep the car going around the highway without getting stuck behind slower moving vehicles.

Data from sensor fusion is mainly used to perceive the region surrounding the Ego vehicle. A list of objects or cars from sensor fusion provides [id, x, y, s, d, vx, vy] values of the other cars on the circuit. Lane changing maneuvers must comprehend these parameters and propose "safe" behaviors for the car to avoid collision with other vehicles.

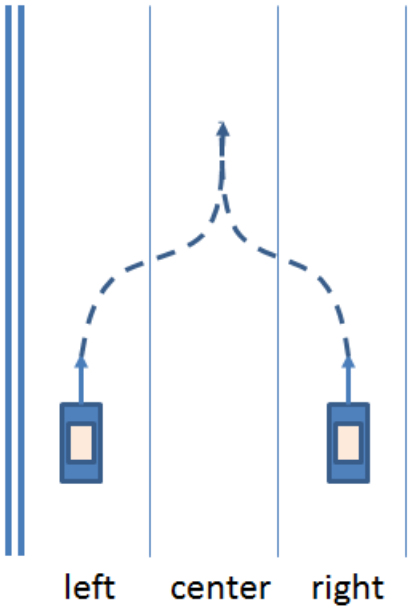
The approach followed is as below,

1. Identify lane-change maneuvers
    a. Side lanes to center lanes (Left to center, Right to center)
    b. Center late to side lanes (Center to Left or Center to Right)
2. Classify vehicles ahead/behind Ego vehicle to respective lanes
3. Prefer lanes with least occupancy.
4. Safe passage for lane change.

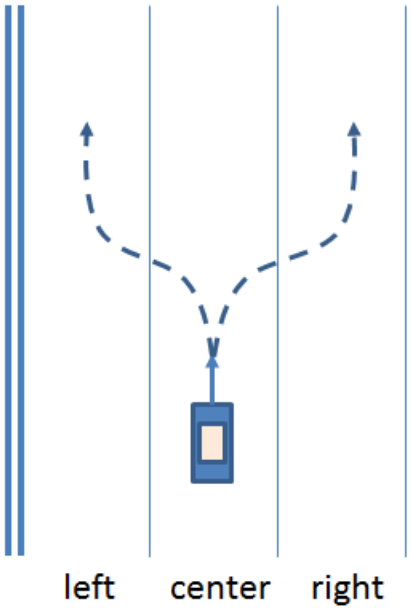
Each topics is discussed in detail below,

1. **Lane change maneuvers**

   Two possible maneuvers are allowed in this implementation. And at one time instant, only one lane change is permitted as shown below,



Case – 1 moving from side lanes to center lane



Case – 2 moving from center lane to side lane

2. **Classify vehicles ahead/behind of Ego vehicles in respective lanes**

   From the single list of sensor fusion data, prepare 6 different lists as shown below. This will help in better understanding of lane occupancy.

Lists	S value	D value
cars_ahead_left	car_s >= 0	car_d < 4m
cars_ahead_center	car_s >= 0	4m < car_d < 8m
cars_ahead_right	car_s >= 0	car_d < 8m
cars_behind_left	car_s < 0	car_d < 4m
cars_behind_center	car_s < 0	4m < car_d < 4m
cars_behind_right	car_s < 0	car_d < 8m

The sensor fusion data can also be pruned for data at a horizon of 60 to 120 meters instead of full range. Sometimes considering the full range of might affect the behavior of the Ego vehicle.

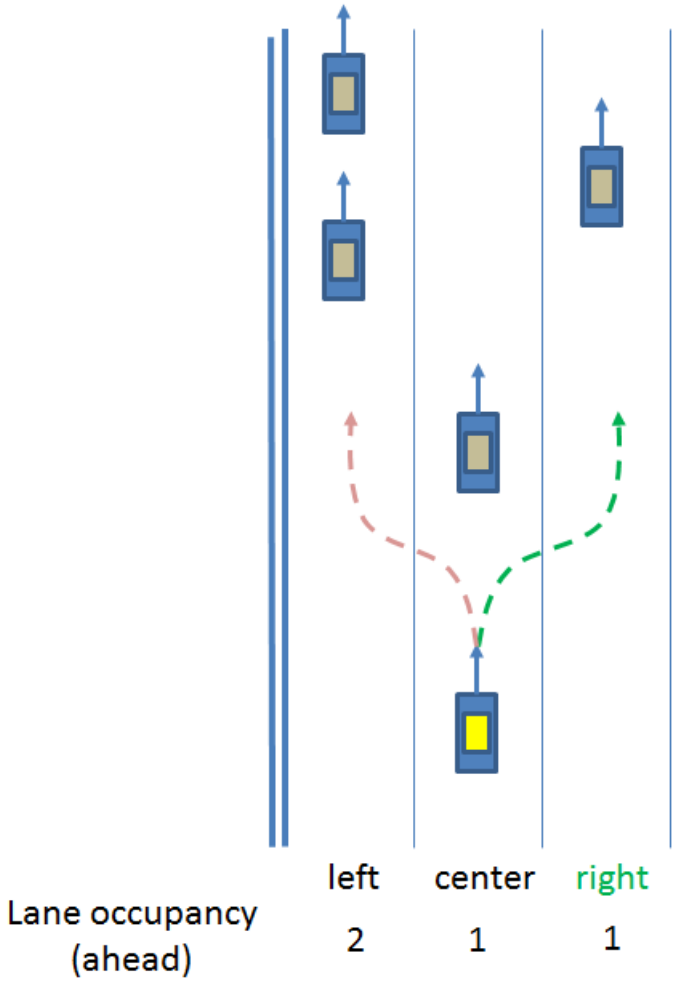
```
296                 if (((s - car_s) >= 0.0) && ((s - car_s) < sensor_range))
297                 {
298                   if (d < 4.0)
299                   {
300                     cars_left_ahead.push_back(car);
301                   }
302                   else if ((d > 4.0) && (d < 8.0))
303                   {
304                     cars_center_ahead.push_back(car);
305                   }
306                   else if(d > 8.0)
307                   {
308                     cars_right_ahead.push_back(car);
309                   }
310                 }

311               else if (((s - car_s) < 0.0) && ((s - car_s) > -sensor_range))
312               {
313                 if (d < 4.0)
314                 {
315                   cars_left_behind.push_back(car);
316                 }
317                 else if ((d > 4.0) && (d < 8.0))
318                 {
319                   cars_center_behind.push_back(car);
320                 }
321                 else if(d > 8.0)
322                 {
323                   cars_right_behind.push_back(car);
324                 }
325               }
```

3. **Prefer lanes with less occupancy**

   Once vehicles are sorted into its respective lists, prefer lanes which are least occupied than lanes with most occupancy as shown below. This is just a preference, but not a final decision. The Ego vehicle can choose to prefer the high occupancy lane if the latter is not available for changing lane. Eg. Other car is close by to Ego vehicle making the lane change unsafe.

   Computing average lane speed instead of lane occupancy was also attempted, but in a lane with many cars, not all cars might be going at the same speed. This affected the behavior of Ego vehicle and several times gets stuck behind a slow moving vehicle even if lane speed is higher due to cars ahead of the slow moving car.



In the above scenario, the Ego vehicle prefers a lane shift to the right instead of the left purely due to lane occupancy. But this is not the final decision, it's just a preference.

4. Safe passage for lane change

Before initiating lane change, it is important to check the distance of cars around the Ego car. In this implementation, a safe distance for 20m is kept both ahead and behind the Ego vehicle before initiating lane change. For both cases, side lanes to center lane and center lane to side lanes, 4 cases have to be handled,

Case 1 – No cars ahead or behind new lane:
This is an easy case where Ego vehicle can make a lane change at ease.

From side lanes to center lane,

```
419                 if((car_ahead == -1) && (car_behind == -1)) {
420                   lane = 1;
421                 }
```

From center lane to left lane,

```
491                 if((car_ahead == -1) && (car_behind == -1)) {
492                   left_lane_free = true;
493                 }
```

From center lane to right lane,

```
556                 if((car_ahead == -1) && (car_behind == -1)) {
557                   right_lane_free = true;
558                 }
```

Case 2 – No car ahead, but car detected behind in new lane
In this case, the Ego vehicle should wait till there is a safe distance between the car behind in new lane.

From side lanes to center lane,

```
423               if((car_ahead == -1) && (car_behind >=0)) {
424                 if(max_dist_behind > safe_dist) {
425                   lane = new_lane;
426                   printf("No cars ahead! max_dist_behind = %f  ", max_dist_behind);
427                 }
428               }
```

From center lane to left lane,

```
495                 if((car_ahead == -1) && (car_behind >=0)) {
496                   if(max_dist_behind > safe_dist) {
497                     left_lane_free = 1;
498                     printf("No cars ahead! max_dist_behind = %f  \n", max_dist_behind);
499                   }
500                 }
```

From center lane to right lane,

```
560                 if((car_ahead == -1) && (car_behind >=0)) {
561                   if(max_dist_behind > safe_dist) {
562                     right_lane_free = 1;
563                     printf("No cars ahead! max_dist_behind = %f  \n", max_dist_behind);
564                   }
565                 }
```

Case 3 – No cars behind but there is a car ahead

In this case, there is a car ahead but Ego vehicle has to wait till there is a safe distance before lane change maneuver

From side lanes to center lane,

```
423                 if((car_ahead == -1) && (car_behind >=0)) {
424                   if(max_dist_behind > safe_dist) {
425                     lane = new_lane;
426                     printf("No cars ahead! max_dist_behind = %f  ", max_dist_behind);
427                   }
428                 }
```

From center lane to left lane,

```
495                 if((car_ahead == -1) && (car_behind >=0)) {
496                   if(max_dist_behind > safe_dist) {
497                     left_lane_free = 1;
498                     printf("No cars ahead! max_dist_behind = %f  \n", max_dist_behind);
499                   }
500                 }
```

From center lane to right lane,

```
567                 if((car_behind == -1) && (car_ahead >=0)) {
568                   if(min_dist_ahead > safe_dist) {
569                     right_lane_free = 1;
570                     printf("No cars behind! min_dist_ahead = %f  \n", min_dist_ahead);
571                   }
572                 }
```

Case 4 – When cars are found both ahead and behind Ego vehicle

In this case, a lane change maneuver will only be initiated if there is a safe passage between ahead and behind car in the new lane.

From side lanes to center lane,

```
437                 if((car_behind >= 0) && (car_ahead >=0)) {
438                   if((min_dist_ahead > safe_dist) && (max_dist_behind > safe_dist)) {
439                     lane = new_lane;
440                     printf("Gap found! min_dist_ahead = %f, max_dist_behind = %f, safe_d
441                   }
442                 }
```

From center lane to left lane,

```
509                 if((car_behind >= 0) && (car_ahead >=0)) {
510                   if((min_dist_ahead > safe_dist) && (max_dist_behind > safe_dist)) {
511                     left_lane_free = 1;
512                     printf("Gap found! min_dist_ahead = %f, max_dist_behind = %f, safe_dist = %f
513                   }
514                 }
```

From center lane to right lane,

```
574                 if((car_behind >= 0) && (car_ahead >=0)) {
575                   if((min_dist_ahead > safe_dist) && (max_dist_behind > safe_dist)) {
576                     right_lane_free = 1;
577                     printf("Gap found! min_dist_ahead = %f, max_dist_behind = %f, safe_dist
578                   }
579                 }
```

For the center lane to side lanes, additional checks are performed to choose either left or right lane.

```
581               printf("left_lane_free = %d, right_lane_free = %d\n", left_lane_free, right_lane_free);
582               if(left_lane_free && right_lane_free)
583               {
584                 printf("Cars Left = %d, Right = %d\n", cars_left_ahead.size(), cars_right_ahead.size());
585                 if(cars_left_ahead.size() > cars_right_ahead.size())
586                 {
587                   lane = 2;
588                 }
589                 else
590                 {
591                   lane = 0;
592                 }
593               }
```

```
595                 if(left_lane_free && !right_lane_free)
596                 {
597                   lane = 0;
598                 }
599
600                 if(!left_lane_free && right_lane_free)
601                 {
602                   lane = 2;
603                 }
604
605                 if(!left_lane_free && !right_lane_free)
606                 {
607                   if((cars_left_ahead.size()  == 0) &&
608                      (cars_right_ahead.size() == 0) &&
609                      (cars_left_behind.size() == 0) &&
610                      (cars_right_behind.size() == 0))
611                   {
612                     lane = 0;
613                   }
614                 }
615               }
```

With these modules the Ego vehicle was able to go around the circuit for 5 miles without any incident. The prediction stage of computing trajectories of other vehicles is not implemented which will further improve for cases where other vehicles are making lane changes.