UDACITY SELF DRIVING CAR NANODEGREE: TERM 3, Project 1: Path Planning Report

- Krithika Swaminathan(krswamin@umich.edu)

Note: All this code also available on github repo
https://github.com/rkyan13/Udacity-Self-Driving-Car-Projects/tree/feature-SCND-Term3-P1-Path-Planning
Repository: https://github.com/rkyan13/Udacity-Self-Driving-Car-Projects
Branch Name: feature-SCND-Term3-P1-Path-Planning

PROJECT SPECIFICATION: Path Planning

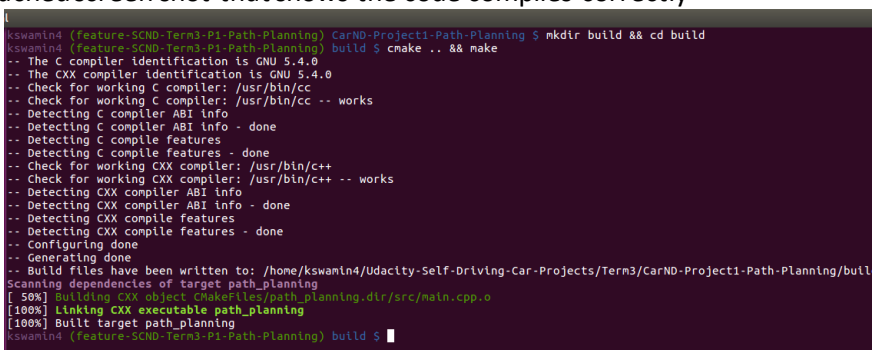
Compilation

CRITERIA-1 : The code compiles correctly.

MEETS SPECIFICATIONS

Code must compile without errors with cmake and make. Given that we've made CMakeLists.txt as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform.

Answer: Please see attached screen shot that shows the code compiles correctly



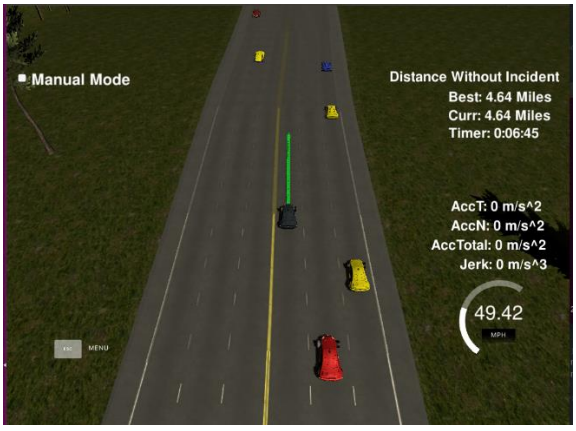
Valid Trajectories

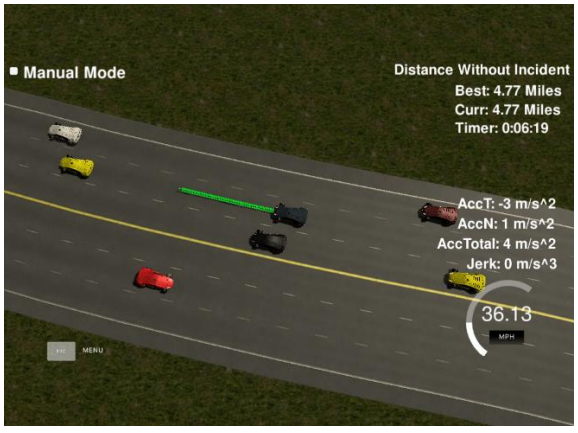
CRITERIA-2: The car is able to drive at least 4.32 miles without incident.

MEETS SPECIFICATIONS

The top right screen of the simulator shows the current/best miles driven without incident. Incidents include exceeding acceleration/jerk/speed, collision, and driving outside of the lanes. Each incident case is also listed below in more detail.

Answer: see screen shots that show the car has driven atleast 4.32 miles without incident





Note to Grader/Reviewer: Occasional Rogue Car in Udacity Simulator: !!!!

Once every 10-15 runs the udacity simulator seems to have a rogue car (usually yellow car) that seems to rapidly/rashly change 2 lanes at a time and collide with the ego car. I tried to reproduce the situation and move the ego_car away quickly to avoid collision. However such a quick response to the rogue is not possible without violating acceleration and jerk limits on an average . Hence I stuck to a more conservative model. Should a similar situation arise, that it is not the failure of my lane change /collision avoidance model. I request you to kindly restart the simulator and run the code again. (Kindly do not mark this criteria as a fail due to rogue collision !!!)

CRITERIA-3 : The car drives according to the speed limit.

MEETS SPECIFICATIONS

The car doesn't drive faster than the speed limit. Also the car isn't driving much slower than speed limit unless obstructed by traffic.

Answer:

i) In order to meet the speed limit of 50mph I've set the IDEAL_SPEED_LIMIT to 49.5mph (i.e. just below 50mph). The REF_VEL is the running reference velocity of the ego_car based on what other cars are around/ lane change situation etc. ***(See code lines approx.206 to 209)***

```
//have a reference velocity to target
const  double IDEAL_SPEED_LIMIT = 49.5;//mph
// REF_VEL will be the running reference velocity of the ego_car based on what other cars are around/ lane change etc.
// start REF_VEL at 0 mph to take care of cold start
double REF_VEL = 0 ; //mph
```

ii) Nominally (when there aren't slower cars ahead, when we aren't changing lanes etc.) We want the car to move at 49.5 mph(= 49.5/2.24 m/s). So the per_time_step_dist covered needs to be 0.02*REF_VEL/2.24 meter. This distance is an arc along the spline. This arc needs to be converted to x and y distances. I just coded this based on Udacity Aaron's Code + Walk through video, using the concept of similar triangles. (See code lines approx. 640 to 665)

```
    // We want the car to move at 49.5 mph(49.5/2.24 m/s). so the per_time_step_dist covered needs to be 0.02*REF_VEL/2.24 meter
    // This distance is an arc along the spline. However similar to sample_dist, we approximate per_time_step_dist as a straight line instead of arc
    // Hence per_time_step_x, per_time_step_y, per_time_step_dist form the base, height and hypotenuse respectively of the Right-Angled-Triangle-2
    // Right-Angled-Triangle-1 & Right-Angled-Triangle-2 are 'similar'. This is used to establish unknowns per_time_step_x
    // per_time_step_y is not estimated from right_triangle but use spline directly (in loop)
    per_time_step_dist  =  0.02*REF_VEL/2.24 ;
    per_time_step_x     = sample_x*per_time_step_dist/sample_dist    ;

    x_point_car_coord = per_time_step_x*i ;
    y_point_car_coord = spl(x_point_car_coord) ;

    //Rotate back to map-coordinates(We had earlier rotated everything to car-ordinates)
    x_point_map_coord = (x_point_car_coord *cos(ref_yaw) - y_point_car_coord*sin(ref_yaw));
    y_point_map_coord = (x_point_car_coord *sin(ref_yaw) + y_point_car_coord*cos(ref_yaw));

    //Shift back to map-coordinates(We had earlier shifted everything to car-ordinates)
    x_point_map_coord += ref_x;
    y_point_map_coord += ref_y;

    next_x_vals.push_back(x_point_map_coord);
    next_y_vals.push_back(y_point_map_coord);

}
```

For more details on how the similar triangle and the spline works. **(See comments in code *lines approx. 590 to 606)***

```
// ----------------------------------------------------------------------------------------------------------------------
// ------ STEP 3b: USE THE SPLINE 'spl' TO SAMPLE : finer next_x_vals, next_y_val (USES SPLINE POINTS-spl) ---------------
// ----------------------------------------------------------------------------------------------------------------------

// ksw comments:  see write up that accompanies code & explanation below
// look at Project Q&A video at ~33min: in the project video 'd' is not the Frenet 'd', it is the 'distance'(hypotenuse calc as : dist= sqrt(x*x+y*y)
// Calculate how to break up spline points so that we travel at our desired reference velocity
// sample_x: is the distance how far into the horizon you want to approximate the spline to
// Just because you choose sample_x = 30m doesn't mean that the car will actually cover 30 meters along x in 50 time steps
// In one cycle the car could travel more than 50 time steps or less. This sample_x is just for approximation.
// Hence choose a value of x that approximates the distance travelled in 50 time steps i.e 50 x 0.02 x 49.5 / 2.24 = 22.098 meter.
// You can even choose sample_x = 0.5m, 1m, 60m, 90m . This will just change the spline point y (sample_y) .
// By change I mean that arc/polynomial : sample_y @0.5m could be significantly different from the spline sample_y @ 60m
// And hence  dimensions of the similar triangle you are trying to approximate could be significantly different
// sample_dist: the path between two consecutive x,y points is an arc(characterised by a spline/polynomial)
//              However this is just approximated as a straight line (hypotenuse) to make life easier
//So sample_x, sample_y, sample_dist form the base, height and hypotenuse respectively  of the Right-Angled-Triangle-1(generated using spline)
double sample_x       = 30.0 ;
double sample_y       = spl(sample_x) ;
double sample_dist    = sqrt((sample_x*sample_x)+(sample_y*sample_y));

double per_time_step_dist ;
double per_time_step_x    ;

double x_point_car_coord, y_point_car_coord;
double x_point_map_coord, y_point_map_coord;
```

iii) The ego velocity gradually reduces only if there a car ahead of it but never drops below the 93% of the speed of the car ahead of ego_car **(See code lines approx. 625 to 631)**

```
//Decrease the REF_VEL of the ego-car gradually when
//---> There is another car in front pg ego-car.
//---> 0.224 corresponds to a decrease in acceleration of ~5m/s^2.
//---> But since we are decrementing every waypoint (and not every cycle) I decrement it at an even slower rate of 0.224/2.0 or 0.224/1.5
if((car_ahead == true) && (REF_VEL > car_ahead_speed*2.24*0.93 )) {
  REF_VEL -= 0.224/1.5;
}
```

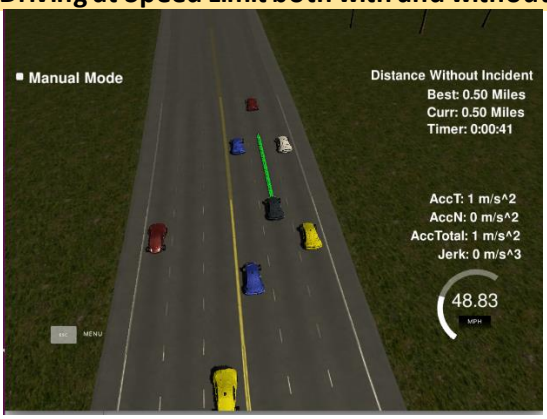
iv) When the REF_VEL has fallen, based on the situation and if safe , increase the REF_VEL to meet the IDEAL_SPEED_LIMIT ***(See code lines approx. 632 - 639)***

Increase the REF_VEL of the ego-car gradually when

- when there is a cold start(starting at REF_VEL =0)
- if the ego-speed had fallen since there was a slower car in front of ego-car and the ego-speed needs to pick up again
    - 0.224 corresponds to a increase in acceleration of ~5m/s^2
    - But since we are incrementing every waypoint (and not every cycle) I increment it at a slower rate of 0.224/3.0

```
//Increase the REF_VEL of the ego-car gradually when
//---> i) when there is a cold start(starting at REF_VEL =0)
//---> ii) if the ego-speed had falled since there was a slower car in front of ego-car and the ego-speed needs to pick up ag
//---> 0.224 corresponds to a increase in acceleration of ~5m/s^2
//---> But since we are incrementing every waypoint (and not every cycle) I increment it at a slower rate of 0.224/3.0
else if(REF_VEL < IDEAL_SPEED_LIMIT){
  REF_VEL += 0.224/3.0;
}
```

Examples: Car Driving at Speed Limit both with and without traffic



Speed Limit Maintained With Traffic

Speed Limit Maintained Without Traffic

Speed Limit Not Maintained only when obstructed by traffic

(Note : I have improved the code further so that ego_car sticks closer to speed limit even when obstructed by traffic. So in the latest submitted version, the speed will probably be close to 35mph even when obstructed by traffic)

CRITERIA-4: Max Acceleration and Jerk are not Exceeded.

MEETS SPECIFICATIONS

The car does not exceed a total acceleration of 10 m/s^2 and a jerk of 10 m/s^3.

See explanation above. **Look at Code line 625 to 639.**

- When the ego_car needs to slow down because of a slower moving car ahead, I decrease the REF_VEL by 0.224 which corresponds to a deceleration of 5m/s^2. This is applicable if the REF_VEL is decremented every cycle. However I don't decrement every cycle, I decrement it inside the path planner for every waypoint. Hence I make it even more conservative and decrease by only 0.224/1.5 so that acceleration and jerk limits are not violated
- Similarly when the ego_car needs to get back to Speed Limit of 49.5 mph either after a lane change or after a cold-start I increment REF_VEL by 0.224, if done every cycle . But when done inside the path planner for incrementing every way point a more conservative increment by 0.224/2.0 is better to avoid exceeding jerk limits
- The most important thing to remain within Jerk limits was to use the spline function. For details of this please refer Attempt 6 in the tabulated section below (Refer to code and comments Line 473 to Line 662)

CRITERIA-5: Car does not have collisions.

MEETS SPECIFICATIONS

The car must not come into contact with any of the other cars on the road.

CRITERIA-6: The car stays in its lane, except for the time between changing lanes.

MEETS SPECIFICATIONS

The car doesn't spend more than a 3 second length out side the lane lanes during changing lanes, and every other time the car stays inside one of the 3 lanes on the right hand side of the road.

CRITERIA-7: The car is able to change lanes

MEETS SPECIFICATIONS

The car is able to smoothly change lanes when it makes sense to do so, such as when behind a slower moving car and an adjacent lane is clear of other traffic.

Explanation of Criteria 5,6,7: Collision Handling and Lane Change Logic Explanation

See Code Lines #305 to #470

General Logic for Collision Avoidance and Lane Change

- If there is a slow moving car ahead of the ego_car in the same lane that breaches the safe distance parameter (SAFE_DIST_FROM_SAME_LANE_CARS = 30.0), the ego_car will change lanes as appropriate to either the left or right lane , whichever is clear of traffic . If that is not possible continue in the same lane at a lower speed, if neither left nor right lane is clear

```
//check if other_car is in ego_lane
if(other_car_lane == ego_lane_actual) {
      if(0<(other_car_s_future - ego_s) &&(other_car_s_future - ego_s)< SAFE_DIST_FROM_SAME_LANE_CARS) {
            car_ahead          = true ;
            car_ahead_num      = i+1  ;
            car_ahead_lane     = other_car_lane ;
            car_ahead_distance = other_car_s_future - ego_s ;
            car_ahead_speed    = other_car_speed ;
      }
}
```

- For Cars moving in the adjacent lane to ego_car but ahead of it the safe-distance is specified by the parameter SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD = 40m. So Ego Car will not change lanes to the left or right if a car will be within 40m ahead of it (in the immediate future)
- Similarly For Cars moving in the adjacent lane to ego_car but ahead of it the safe-distance is specified by the parameter SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND = 17m. So Ego Car will not change lanes to the left or right if a car will be within 17m behind it (in the immediate future)

```
if(other_car_lane == ego_lane_actual-1){
      if(0<(other_car_s_future - ego_s) && abs(other_car_s_future - ego_s)< SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD){
            car_left          = true ;
            car_left_num      = i+1  ;
            car_left_lane     = other_car_lane ;
            car_left_distance = other_car_s_future - ego_s ;
            car_left_speed    = other_car_speed ;
      }
      else if(0>(other_car_s_future - ego_s) && abs(other_car_s_future - ego_s)< SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND){
            car_left          = true ;
            car_left_num      = i+1  ;
            car_left_lane     = other_car_lane ;
            car_left_distance = other_car_s_future - ego_s ;
            car_left_speed    = other_car_speed ;
      }
}

if(other_car_lane == ego_lane_actual+1){
      if(0<(other_car_s_future - ego_s) && abs(other_car_s_future - ego_s)< SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD){
            car_right          = true ;
            car_right_num      = i+1  ;
            car_right_lane     = other_car_lane ;
            car_right_distance = other_car_s_future - ego_s ;
            car_right_speed    = other_car_speed ;
      }
      else if(0>(other_car_s_future - ego_s) && abs(other_car_s_future - ego_s)< SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND){
            car_right          = true ;
            car_right_num      = i+1  ;
            car_right_lane     = other_car_lane ;
            car_right_distance = other_car_s_future - ego_s ;
            car_right_speed    = other_car_speed ;
      }
}
```

Example Case: How Does this Logic and the values of these parameters facilitate Smooth Lane Change

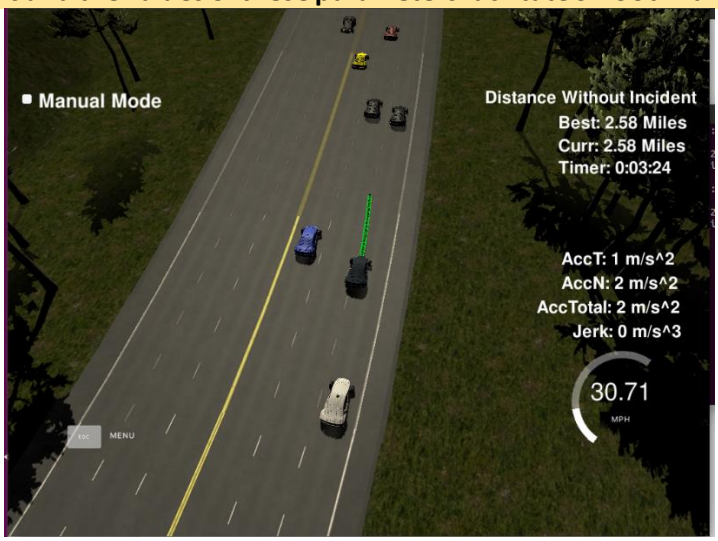


Illustration : cars at ~30m and ~32m ahead of ego car in same lane and adjacent lane respectively

- If there is a slower moving car at distance of <30m ahead the ego_car will attempt to change lanes to the left or right. In the previous version the code was written so that the adjacent car lanes also needed to be clear of cars only by 30meters to facilitate lane change. This caused sloppy lane changes.
- For example assume ego_car travelling in Lane2(the right most lane) encounters a slow moving car ~29 meters ahead. Also assume there is a slow moving car ~32 metres ahead of ego_car but on the Middle Lane (Lane 1) i.e. to the left of the ego car. The ego_car would move left from Lane-2(right lane) to Lane-1(middle lane) only to realise very quickly the car in Lane-1(Middle lane) is also slow moving and jump back to Lane-1 . This would cause the ego_car to straddle lanes/ keep hopping between lanes etc.
- In order to facilitate smoother lane change I added the following parameters, SAFE_DIST_FROM_SAME_LANE_CARS = 30.0, SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD = 40 & SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND =17 (I spent a fair amount of time fine tuning the values). This way the ego_car will change lanes only if the adjacent lanes are clear of traffic 40m ahead (i.e more than the same_lane clearance of 30m) since there is really no point in changing lanes and landing in a similar situation
- Also 30m for checking for cars behing the ego_car in adjacent lanes was too much making lane change tooo conservative and frustrating. So I fine tuned SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND to 17m using trial and experimentation

Example : Ego_car will not change lane to left or right since it is unsafe to do, but will follow slower moving red car ahead.

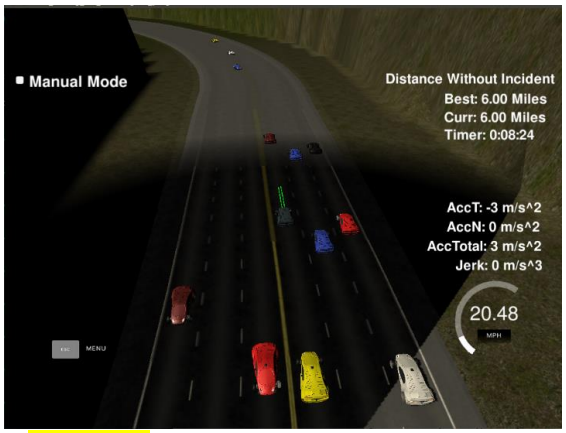
Red car at 36-37mph and ego car following behind at 34mph

Example : Sloppy Lane Change in previous versions of code fix. (This no longer happens in final version)

Ego_car is behind slow moving blue car and tried to change lanes to left behind black car. But it soon realizes Black car is slow moving too and might attempt to hop back under blue.

Hence added SAFE_DIST_FROM_SAME_LANE_CARS = 30.0, SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD = 40 & SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND =17 to take care of this issue. Car will no longer attempt to change lanes in similar situation below but will continue driving in same lane

BEFORE FIX: Ego car sloppy lane change



AFTER FIX: Ego Car does not Change Lane

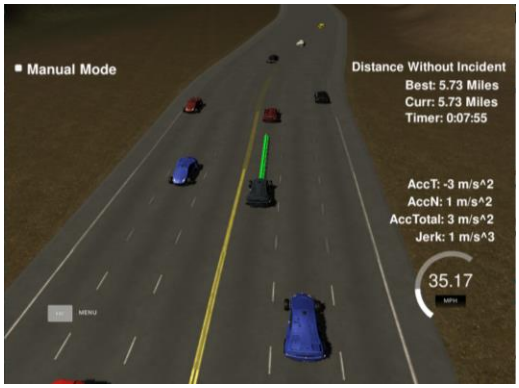
FUTURE IMPROVEMENTS TO THE CODE

I've not addresses these improvements in the final submitted code main.cpp since the Project Rubric doesn't require these, and also didn't fine tune them in the interested of time

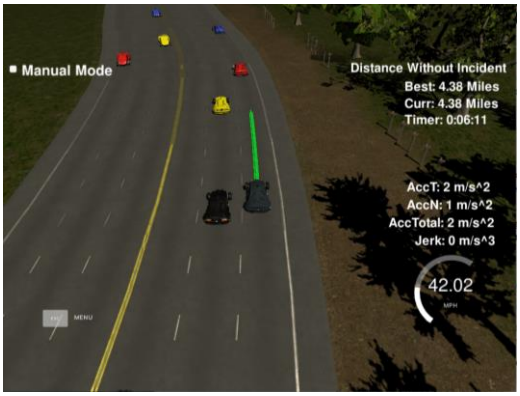
However, I've implemented some of the improvements in Additional-Files/main-v11a.cpp just as a means of exploration/ fun, not for project evaluation. Hence main-v11a doesn't consider/satisfy all the rubric points . If interested please replace main.cpp with main-v11a.cpp and see how well it does. (or checkout corresponding commit from github –repo)

Improvement 1: Lane Change is Too Conservative:

With the given acceleration and jerk limits, this was the best fine tuning that I could do. Perhaps with a more sophisticated model that gives me control over not just the REF_VEL but also over brake etc. and also by occasionally relaxing acceleration and jerk limits a more real-life city type of lane change is possible .



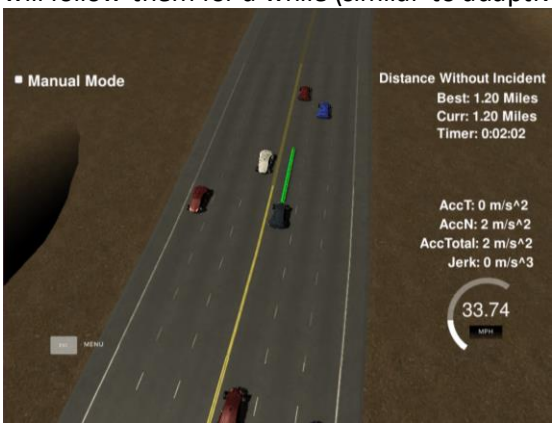
Ego won't change lanes because blue car on right lane is <15m (within 15 m now and immediate future) behind

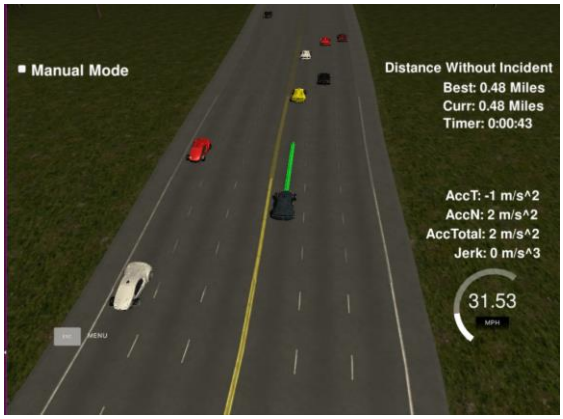


Ego won't cut in between black and yellow cars on left and move to highest speed lane

Improvement 2: Ability to Change more than 1 lane at a time (to prevent Stalemate)

In the following figures the car ahead of ego car and the car in the adjacent right lane are moving slowly. The right most lane is free. But at the moment the Ego car is able to check for conditions and change lanes only to the adjacent lane and hence cannot move to the right most lane quickly from the left most lane. So if the cars ahead move slowly the ego car will follow them for a while (similar to adaptive crusie control) and make a lane change only when appropriate





Improvement 3: Consciously move away from high speed lane/ low speed lane to the center lane

Once the ego_car gets on the high speed lane/ low speed lane, it will not move back to the center lane unless it is being impeded by a slower moving car. It will be nice if the car can by default drive on the center lane as a courtesy to fellow drivers (like how it should drive in real life). This has been implemented in additional code main-v11a.cpp(but not in the submitted main.cpp)

Reflection

CRITERIA-8: There is a reflection on how to generate paths.

MEETS SPECIFICATIONS

The code model for generating paths is described in detail. This can be part of the README or a separate doc labeled "Model Documentation".

Answer:

- The most important for the path generation is the use of spline. For details of this please refer Attempt 6 in the tabulated section below (Refer to code and comments Line 473 to Line 662)
- Collision handling and Lane Change is also important for generation path (already discussed above)
- The step by step procedure of how I did it is tabulated below. I've also detailed these steps in the comments section at the beginning of main.cpp I followed the Walkthrough Video + Aaron's code in the Project Lessons to build the Path Planner step by step. I made suitable modifications/fine tunings at each step

Notes to self:

- in Frenet co-ordinates d is lateral distance, s is vertical distance along the length of the road
- d-value for lanes
    - 0
    - LeftLane: Lane#0: (1,2,3),
    - 4
    - MiddleLane: Lane#1: (5,6,7),
    - 8
    - RightLane: Lane#2: (9,10,11),
    - 12
- previous_path_x, previous_path_y DO-NOT INCLUDE ALL the points generated the previous cycle. They only consist of the remainder of the path points that the car did not use/did not travel in the previous cycle

Attempt #	**Incremental Code Improvement Details at Each Attempt in main.cpp code** Note: To see these step by step versions please checkout the corresponding commit on github repo or use the corresponding main-vX.cpp files from 'Additional-Files' folder(provided in the zip) Repository: https://github.com/rkyan13/Udacity-Self-Driving-Car-Projects Branch Name: feature-SCND-Term3-P1-Path-Planning
Attempt 0	Download base code from udacity github repo

Attempt 1	Make car drive in straight line, based on udacity code
	See Lesson 6: Path Planning Project/ Concept 2-Getting Started
	Getting Started
	Start by simply trying to move the car forward in a straight line at a constant 50 MPH velocity. Use the car's (x, y) localization information and its heading direction to create a simple, straight path that is drawn directly in front of the car.
	In `main.cpp`, instead of setting the speed directly, we pass `next_x_vals`, and `next_y_vals` to the simulator. We will set the points 0.5 m apart. Since the car moves 50 times a second, a distance of 0.5m per move will create a velocity of 25 m/s. 25 m/s is close to 50 MPH.
	`double dist_inc = 0.5;`
	`for (int i = 0; i < 50; ++i) {`
	`  next_x_vals.push_back(car_x+(dist_inc*i)*cos(deg2rad(car_yaw)));`
	`  next_y_vals.push_back(car_y+(dist_inc*i)*sin(deg2rad(car_yaw)));`
	`}`
	What happens in this attempt:
	Car attempts to drive in straight line, but car over shoots road

Attempt 2	Make car drive in circular path, based on udacity code

	See Lesson 6: Path Planning Project/ Concept 2-Getting Started
	
	What happens in this attempt:
	Car drives in circular path (just for fun)
	First Attempt at using previous_path points
	`vector<double> next_x_vals;`
	`vector<double> next_y_vals;`
	
	`double pos_x;`
	`double pos_y;`
	`double angle;`
	`int path_size = previous_path_x.size();`
	
	`for (int i = 0; i < path_size; ++i) {`
	`  next_x_vals.push_back(previous_path_x[i]);`
	`  next_y_vals.push_back(previous_path_y[i]);`
	`}`
	`if (path_size == 0) {`
	`  pos_x = car_x;`
	`  pos_y = car_y;`
	`  angle = deg2rad(car_yaw);`
	`} else {`
	`  pos_x = previous_path_x[path_size-1];`
	`  pos_y = previous_path_y[path_size-1];`
	
	`  double pos_x2 = previous_path_x[path_size-2];`
	`  double pos_y2 = previous_path_y[path_size-2];`
	`  angle = atan2(pos_y-pos_y2,pos_x-pos_x2);`
	`}`
	`double dist_inc = 0.5;`
	`for (int i = 0; i < 50-path_size; ++i) {`
	`  next_x_vals.push_back(pos_x+(dist_inc)*cos(angle+(i+1)*(pi()/100)));`
	`  next_y_vals.push_back(pos_y+(dist_inc)*sin(angle+(i+1)*(pi()/100)));`
	`  pos_x += (dist_inc)*cos(angle+(i+1)*(pi()/100));`
	`  pos_y += (dist_inc)*sin(angle+(i+1)*(pi()/100));`
	`}`
	
	`msgJson["next_x"] = next_x_vals;`
	`msgJson["next_y"] = next_y_vals;`

Attempt 3	Make car drive in straight line along 's'
Attempt 4	Make car drive in straight line along 's' using speed equations(no acceleration), car path explodes
Attempt 5	Attempt at lane keeping (drive in straight line along s) using speed and acceleration limits. ➢ Car maintains lane However car still exceeds speed, accel and jerk limits several times ➢ Polynomial Lane Generation /Spline not used: Possible use of this can keep accel and jerk within limits ➢ Car Collision not handled *(Check out the corresponding version of the code from github or use Additional-Files/main-v5.cpp to see code snippets below)* Set limits on speed and acceleration

```
118             vector<double> next_x_vals;
119             vector<double> next_y_vals;
120             vector<double> next_point ;
121             double egoC_next_s    = ego_s;
122             double egoC_speed_mps = ego_speed_mph*0.44704;
123             double max_accel = 9.5 ; //m/s^2
124             double time_step = 0.02 ; //0.02 seconds = 20 milli seconds
```

Attempt at Lane Keeping along 's' using speed and acceleration limits

```
134             ------------------------------------------------------------------------------------------
135                     Attempt at Lane Keeping along 's' using speed and acceleration limits
136             ------------------------------------------------------------------------------------------
137
138             Performance Comments:
139             i)   The car is able to maintain the lane and drive at center of Lane (that is the good news)
140             ii)  However ego car's speed hovers between 17-30mph. I think the time to reach 50mpph is too slow
141                  (so might need to use a different approach rather than using these motion equations)
142             iii) Occasionally at turns where the way points are clustered or something, the car exceeds speed limit to 100mph
143             iv)  Also the max acceleration and jerk get exceeded
144             v)   No mechanism for collision avoidance has benn included yet (so car will collide , but that is okay)
145
146             Verdict/Goal/TODO :
147             i)  Achieve lane keeping, without violating speed, acceleration and jerk limits
148             ii) Use eiher polynomial lane generation or spline to be able to do this
149             */
150
151             if(egoC_speed_mps < 22) //increase ego_speed_mph, egoC_speed_mps
152              { egoC_next_s    = egoC_next_s    + (egoC_speed_mps*time_step) + (0.5*max_accel*time_step*time_step);
153                egoC_speed_mps = egoC_speed_mps + (max_accel*time_step);
154                if(egoC_speed_mps> 22)
155                   egoC_speed_mps = 22;
156              }
157             else if (22.352 < egoC_speed_mps) //decrease ego_speed_mph, egoC_speed_mps
158             { egoC_next_s    = egoC_next_s    + (egoC_speed_mps*time_step) - (0.5*max_accel*time_step*time_step);
159               egoC_speed_mps = egoC_speed_mps - (max_accel*time_step);
160               if(egoC_speed_mps < 22)
161                  egoC_speed_mps = 22;
162             }
163             else if ((22< egoC_speed_mps) && (egoC_speed_mps<22.352)) //maintain speed
164             { egoC_next_s    = egoC_next_s    + (egoC_speed_mps*time_step) ;
165               egoC_speed_mps = egoC_speed_mps ;
166             }
167
168             next_point = getXY(egoC_next_s,ego_d, map_waypoints_s, map_waypoints_x, map_waypoints_y);
169             next_x_vals.push_back(next_point[0]);
170             next_y_vals.push_back(next_point[1]);
```

Attempt 6	Attempt at lane keeping (drive in straight line along s) using spline. Based on Udacity's Aaron's Code *(Check out the corresponding version of the code from github or use main-v6p5.cpp to see code snippets below)* Logic: **STEP 1: CREATE 5 POINTS SPACED FAR APART : GENERATE ptsx & ptsy**: so this will be (ptsx &,ptsy)= 5 points = {(ref_x,ref_y), (ref_prev_x,ref_prev_y), getXY(ego_s_30),getXY(ego_s_60),getXY(ego_s_90)}

```
173             // ------------------------------------------------------------------------------------------------------------------------
174             // ----------------------------------- Generate ptsx & ptsy -----------------------------------------------------------------
175             // ------------------------------------------------------------------------------------------------------------------------
176             // Create a list of widely space (9x,y) waypoints, evenly spaced at 30m
177             // Later we will interpolate these waypoints with a spline and fill it in with more points that control spacing
178             vector <double> ptsx;
179             vector <double> ptsy;
180
181             // reference x,y, yaw states . The reference state is one of the two
182             // i) either the current car state
183             // ii) or the last point of the previous_path. but how do we know the last point of the previous path ??
184             // we only get a list of the remainder of the points
185             double ref_x     , ref_y      , ref_yaw;
186             double ref_x_prev, ref_y_prev ;
187
188             //if previous size is almost empty, use the car as starting reference
189             if(prev_size < 2) {
190               //Use two points that make the path tangent to the car
191               //ksw comment:
192               //i) one is the car's position ego_x,ego_y,ego_yaw
193               //ii) the other point , use the ego car's yaw to go back one step.
194               //But how do you know how far to go back along yaw ??? could you use the car's speed, or is that really unnecessary
195               ref_x   = ego_x;
196               ref_y   = ego_y;
197               ref_yaw = ego_yaw; //in radian, important to note that while udacity uses ref_yaw in degrees, I use it in radian.
198
199               ref_x_prev = ref_x - cos(ref_yaw);
200               ref_y_prev = ref_y - sin(ref_yaw);
201
202             }
```

```
204           else {
205             //Use two points that make the path tangent to the previous path's end point
206             //ksw comment : again you will use only two points, LOL !!! much ado about nothing
207             //redefine reference state as previous path end point
208             ref_x       = previous_path_x[prev_size -1];
209             ref_y       = previous_path_y[prev_size -1];
210
211             ref_x_prev = previous_path_x[prev_size-2];
212             ref_y_prev = previous_path_y[prev_size-2];
213
214             ref_yaw = atan2(ref_y-ref_y_prev, ref_x-ref_x_prev) ; //in radian
215
216           }
218         //In Frenet add evenly 30m spaced points ahead of the starting reference 9look out for 30m, 60m, 90m)
219         vector<double> next_wp0 = getXY(ego_s+30, (2+4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
220         vector<double> next_wp1 = getXY(ego_s+60, (2+4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
221         vector<double> next_wp2 = getXY(ego_s+90, (2+4*lane), map_waypoints_s, map_waypoints_x, map_waypoints_y);
222
223         // So there are 5 waypoints
224         //  i) previous 2 locations of the cars
225         // ii) location of the car in 30 meters,  60 meters and 90 meters
226         ptsx.push_back(ref_x_prev);
227         ptsx.push_back(ref_x);
228         ptsx.push_back(next_wp0[0]);
229         ptsx.push_back(next_wp1[0]);
230         ptsx.push_back(next_wp2[0]);
231
232         ptsy.push_back(ref_y_prev);
233         ptsy.push_back(ref_y);
234         ptsy.push_back(next_wp0[1]);
235         ptsy.push_back(next_wp1[1]);
236         ptsy.push_back(next_wp2[1]);
239       // Shift ptsx, ptsy  to the car's local co-ordinates i.e. with the car at zero. This transformation is a shift and rotation.
240       // We did this in MPC. It makes the math much easier (will come into play later)
241       // The last point of the previous path is at (0,0) origin, angle is also at zero degrees.
242       // Note : The first  point on the list (ref_x_prev, ref_y_prev) is not at (0,0)
243       //        The second point on the list (ref_x,      ref_y,     ref_yaw) is moved to (0,0,0) : This is where the car is
244       for (int i =0; i < ptsx.size(); i++) {
245         //shift ego_car reference angle to 0 degrees
246         double shift_x = ptsx[i] - ref_x;
247         double shift_y = ptsy[i] - ref_y;
248
249         //Rotation
250         ptsx[i] = (shift_x *cos(0-ref_yaw) -shift_y*sin(0-ref_yaw));
251         ptsy[i] = (shift_x *sin(0-ref_yaw) +shift_y*cos(0-ref_yaw));
252
253       }
```

STEP 2: CREATE SPLINE : FROM THE FAR SPACED 5 POINTS

spl = spline-from(ptsx,ptsy)

```
258         // ------------------------------------------------------------------------------------------------------------------------
259         // ------------------------------------ Create the Spline -------------------------------------------------------------------
260         // ------------------------------------------------------------------------------------------------------------------------
261         //ksw comments: define a spline spl
262         tk::spline spl;
263
264         //ksw comments: set the anchor points ptsx and ptsy (from above) to the spline
265         //ptsx, ptsy are the 5 points from above whichc are also transformed to car's local co-ordinates (whit car at zero)
266         spl.set_points(ptsx, ptsy);
```

STEP 3: CREATE THE ACTUAL PATH POINTS: next_x_vals, next_y_vals for the Path Planner
STEP 3a: USE previous_path POINTS TO CREATE :next_x_vals, next_y_vals (DOES NOT USE THE SPLINE POINTS-spl)

---> This step does not require the use of the newly generated spline, we just use the previous_path_x & previous_path_y.

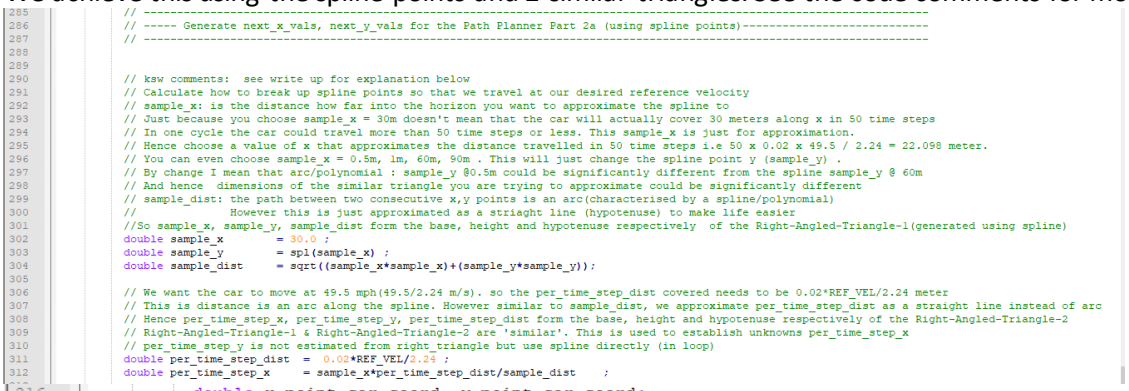
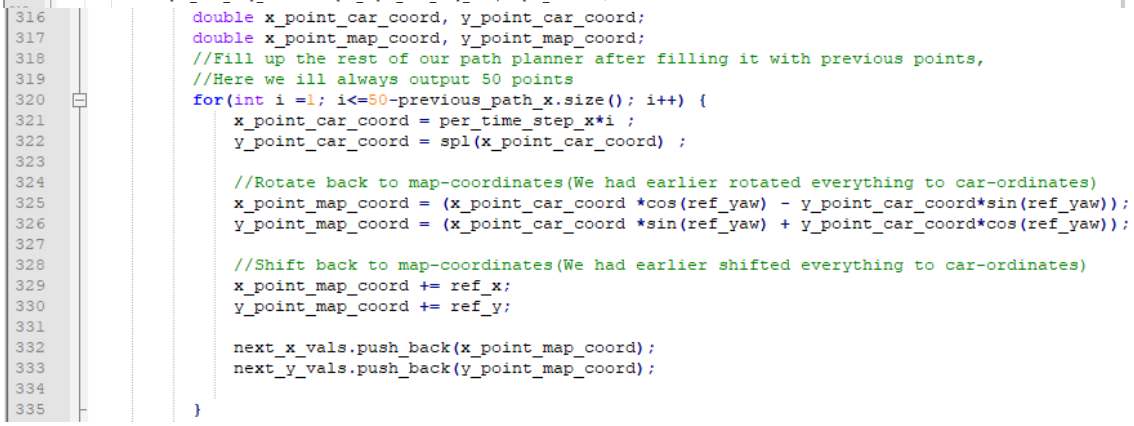
---> However spline could have been used to generate previous_path_x & previous_path_y (in the previous cycle of course)

---> Note: previous_path_x, previous_path_y DO-NOT INCLUDE ALL the points generated the previous cycle.

They only consist of the remainder of the path points that the car did not use/did not travel in the previous cycle

```
272         // ------------------------------------------------------------------------------------------------------------------------
273         // ----- Generate next_x_vals, next_y_vals for the Path Planner Part 1 (using previous_path values)----------------------
274         // ------------------------------------------------------------------------------------------------------------------------
275         //Define the actual (x,y) points we will use for the Planner
276         vector<double> next_x_vals;
277         vector<double> next_y_vals;
278
279         //Start with all of the previous path points from last time and push those
280         for(int i =0; i <previous_path_x.size(); i++){
281           next_x_vals.push_back(previous_path_x[i]);
282           next_y_vals.push_back(previous_path_y[i]);
283         }
```

STEP 3b: USE THE SPLINE 'spl' TO SAMPLE : finer next_x_vals, next_y_val

	Ideally the distance we'd like to move every time step to maintain speed limit is per_time_step_dist = 0.02*REF_VEL/2.24 But we need a way to map this 'per_time_step_dist ' to x & y co-ordinates on a curved road. We achieve this using the spline points and 2-similar-triangles. See the code comments for more details   **Resolved Issues:** Car maintians speed, accel and jerk limits (exceeds accel and jerk only during cold start i.e) **Issues to address:** i) Handle accel and jerk exceeding during cold start ii) Handle collisions iii) Handle Lane Changes
Attempt 7a	**Attempt at Handling Collisions** **Resolved Issues:** Ego car does not collide: slows down to 29.5 mph if there is a car in front (but ego-car reference velocity keeps changing from 29.5 to 49.5 mph every cycle. Quite irritating !!!) *(Check out the corresponding version of the code from github or use main-v7a.cpp to see code snippets below)*

```
188             //find ref_v to use
189             for(int i =0; i< sensor_fusion.size(); i++) {
190                 //sensor_fusion[i] has the values from ith car on the road
191                 other_car_d = sensor_fusion[i][6];
192
193                 //check if other_car is in ego_car-lane
194                 if(other_car_d < (2+4*ego_lane+2) && other_car_d >(2+4*ego_lane-2)) {
195                     other_car_vx    = sensor_fusion[i][3];
196                     other_car_vy    = sensor_fusion[i][4];
197                     other_car_speed = sqrt(other_car_vx*other_car_vx + other_car_vy*other_car_vy);
198
199                     other_car_s     = sensor_fusion[i][5];
200                     //project other_car_s into the future. i.e if prev_size has 10 remaining values. where will the other car be in 10 time steps
201                     //KSW Question : previous_size differs every time. and everytime we calculate path for 50 time steps. Shouldn't we be projecting :
202                     other_car_s_future = other_car_s + ((double)prev_size*0.02*other_car_speed); //if using previous points cann project s value out
203
204                     //Check if other_car_s in the future is ahead of ego_s and within 30m of the ego_s: check s values greater than mine and s gap
205                     if((other_car_s_future > ego_s) && ((other_car_s_future - ego_s)<30)) {
206
207                       //Do some logic here
208                       //i) lower reference velocity so that we dont crash into the car in front of is
209                       //ii) also flag to try to change ego_lanes
210                       REF_VEL = 29.5; //30mph
211
212                       //Set the REF_VEL to 0.95 of the other_car in front of you. other_car_s is m/s, convert to mph by multiplying by 2.23694
213                       //REF_VEL = (other_car_s*2.23694)*0.95 ; //mph
214                       //too_close = true ;
```

Issues to address:

i) Handle collisions in a better way
--> ego-car velocity keeps changing from 29.5 to 49.5 mph every cycle
--> ego-car slows down to 29.5 mph but never picks up the speed back to 49.5 mph
ii) Handle accel and jerk exceeding during cold start
iii) Handle Lane Changes

Attempt 7b

Attempt at Handling Collisions

Resolved Issues:

- Resolved the velocity changing from 29.5mph to 49.5 mph every cycle.Now ego-car-velocity remains at 29.5mph every cycle
- Moved the following piece of code from inside int main()/h.onMessage(...) to outside int main()
  //start in ego_lane 1
  int ego_lane = 1;
  //have a reference velocity to target
  double REF_VEL = 49.5 ; //mph

(Check out the corresponding version of the code from github or use main-v7b.cpp to see code snippets below)

```
90  // for convenience
91  using nlohmann::json;
92  using std::string;
93  using std::vector;
94  using namespace std;
95
96  //start in ego_lane 1
97  int ego_lane = 1;
98
99  //have a reference velocity to target
100 double REF_VEL = 49.5 ; //mph
101
102
103 int main() {
104   uWS::Hub h;
```

Attempt 7c

Attempt at Handling Collisions

Car slows down to the 0.9*speed-of-car-in-front instead of a standard (29.5mph)

```
//Do some logic here
//i) lower reference velocity so that we dont crash into the car in front of is
//ii) also flag to try to change ego_lanes
//REF_VEL = 29.5; //30mph

//Set the REF_VEL to 0.95 of the other_car in front of you. other_car_s is m/s, convert to mph by multiplying by 2.24
REF_VEL = (other_car_speed*2.24)*0.9 ; //mph
//too_close = true ;
}
```

Attempt 8a	**Collision Avoidance & Cold Start : Slow-Down, Speed up Gradually (EVERY CYCLE, OUTSIDE PATH PLANNER)**
	
	Resolved Issues:
	
	i) Collision Handling: Smooth Acceleration/Deceleration & Jerk Minimization
	---> Ego car does not collide
	---> Ego car slows down GRADUALLY @REF_VEL -= 0.224(deceleration 5m/s^2) every cycle(outside path planner)
	---> Once the slow moving car has passed the ego-car gradually ramps up to IDEAL_SPEED_LIMIT @REF_VEL -= 0.224(acceleration 5m/s^2) every cycle(outside path planner)
	
	ii) Cold Start Jerk Minimization
	---> Car will start at 0 and gradually ramp up to IDEAL_SPEED_LIMIT @REF_VEL -= 0.224(acceleration 5m/s^2) every cycle (outside path planner)
	
	`//Decrease the REF_VEL of the ego-car gradually when`
	`//---> There is another car in front pg eg0-car.`
	`//---> 0.224 corresponds to a decrease in acceleration of ~5m/s^2`
	`if(too_close) {`
	`  REF_VEL -= 0.224;`
	`}`
	`//Increase the REF_VEL of the ego-car gradually when`
	`//---> i) when there is a cold start(starting at REF_VEL =0)`
	`//---> ii) if the ego-speed had falled since there was a slower car in front of ego-car and the ego-speed needs to pick up again`
	`//---> 0.224 corresponds to a increase in acceleration of ~5m/s^2`
	`else if(REF_VEL < IDEAL_SPEED_LIMIT){`
	`  REF_VEL += 0.224;`
	`}`
	
	Issues to address:
	i) Increment/Decrement the ego-speed(i.e the REF_VEL) every way-point, inside the path-planner (instead of doing it every cycle outside the path planner)
	ii) Handle lane Changes

Attempt 8b:	**Collision Avoidance & Cold Start : Slow-Down, Speed up Gradually (EVERY WAYPOINT, INSIDE PATH PLANNER)**
	
	Resolved Issues:
	
	i) Increment/Decrement the ego-speed(i.e the REF_VEL) every way-point, inside the path-planner loop (instead of doing it every cycle outside the path planner)
	But Increment/Decrement-ing REF_VEL @0.224 every way-point causes jerk issues + collisions.
	Hence ---> I decrement REF_VEL at a slower rate of 0.224/2.0, &
	---> I increment REF_VEL at a slower rate of 0.224/3.0
	
	`//Decrease the REF_VEL of the ego-car gradually when`
	`//---> There is another car in front pg eg0-car.`
	`//---> 0.224 corresponds to a decrease in acceleration of ~5m/s^2.`
	`//---> But since we are decrementing every waypoint (and not every cycle) I decrement it at an even slower rate of 0.224/2.0`
	`if(too_close) {`
	`  REF_VEL -= 0.224/2.0;`
	`}`
	`//Increase the REF_VEL of the ego-car gradually when`
	`//---> i) when there is a cold start(starting at REF_VEL =0)`
	`//---> ii) if the ego-speed had falled since there was a slower car in front of ego-car and the ego-speed needs to pick up again`
	`//---> 0.224 corresponds to a increase in acceleration of ~5m/s^2`
	`//---> But since we are incrementing every waypoint (and not every cycle) I increment it at a slower rate of 0.224/3.0`
	`else if(REF_VEL < IDEAL_SPEED_LIMIT){`
	`  REF_VEL += 0.224/3.0;`
	`}`
	
	Issues to address:

	i) Handle lane Changes
Attempt 9	**First take on Lane change (See project Q&A 52.30min to 57min)** i) If there is is a slow moving car ahead, ego_car will move to the left lane (if left lane exists) **Issues to address:** i) Does not check if it is safe to change lanes. i.e does not check if car on left lane is within colliding distance, just blindly changes lane
	Rubric Passing Version with Good Lane Changes !!!! i) If there is is a car ahead, ego_car will change lanes to as appropriate left or right lane whichever is clear of traffic (or continue in the same lane, if neither left nor right lane is clear) (See Lane Change Logic Section above) **Possible Improvements** i) If Ego-Car is in High-Speed-Left-Lane(Lane0) or Low-Speed-Right-Lane(Lane2), it does not make an effort to get back to Center-Lane1

ATTEMPT 10b: Final SUBMITTED VERSION

Attempt 10b: Rubric Passing Version with Cleaner Lane Changes and Better Collision Avoidance

Resolved issues

i) Better Speed Limit Maintenance by ego_car
Changed the following line of code
if((car_ahead == true)
to
if((car_ahead == true) && (REF_VEL > car_ahead_speed*2.24*0.93)) to ensure that the speed of the ego_car never drops below 93% of the car_ahead.
Without this change the ego_car speed would often drop dangerously low to values as low as 15mph making lane change impossible/dangerous and would also lead to collisions

ii) Smoother Lane changes

If there is a slower moving car at distance of <30m ahead the ego_car will attempt to change lanes to the left or right.In the
previous version the code was written so that the adjacent car lanes also needed to be clear of cars only by 30meters to facilitate lane change.
This caused sloppy lane changes.

For example assume ego_car travelling in Lane2(the right most lane) encounters a slow moving car ~29 meters ahead.
Also assume there is a slow moving car ~32 metres ahead of ego_car but on the Middle Lane (Lane 1) i.e. to the left of the ego car
The ego_car would move left from Lane-2(right lane) to Lane-1(middle lane) only to realise very quickly the car in Lane-1(Middle lane)
is also slow moving and jump back to Lane-1 . This would cause the ego_car to straddle lanes

In order to facilitate smoother lane change I added the following parameters
SAFE_DIST_FROM_SAME_LANE_CARS = 30.0

	SAFE_DIST_FROM_ADJACENT_LANE_CARS_AHEAD = 40 & SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND=17 (I spent a fair amount of time fine tuning the values) This way the ego_car will change lanes only if the adjacent lanes are clear of traffic 40m ahead (i.e more than the same_lane clearance of 30m) since there is really no point in changing lanes and landing in a similar situation Also 30m for checking for cars behing the ego_car in adjacent lanes was too much making lane change tooo conservative and frustrating. So I fine tuned SAFE_DIST_FROM_ADJACENT_LANE_CARS_BEHIND to 17m using trial and experimentation

Suggestions to Make Your Project Stand Out!

Create a path planner that performs optimized lane changing, this means that the car only changes into a lane that improves its forward progress:

Answer: See Additional-Files/ main-v11a.cpp (not for evaluation, just exploratory)

Attempt 11a: Ego-car tries to get back to center lane after passing cars

i) If there is is a car ahead, ego_car will change lanes to as appropriate left or right lane whichever is clear of traffic (or continue in the same lane, if neither left nor right lane is clear)

ii) If Ego-Car is in High-Speed-Left-Lane(Lane0) or Low-Speed-Right-Lane(Lane2), it makes an effort to get back to Center (but need to fine tune this further)

========== THE HAPPY ENDING ☺ ==========