**Path Project**

**Author – Vishal Kharjul**

**Submitted files**

My project includes the following files:

* **Model Documentation.pdf** summarizing approach to generate path
* **src/main.cpp** containing main code
* **spline.h** spline library

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

**1. The car is able to drive at least 4.32 miles without incident.**

Yes, Car is able to complete 4.32 miles in around 5 to 6 miles. I have set max speed of 49.5 miles/s as shown in Q/A Video. Only one case Car is expected to collide if any vehicle tries to cut Car’s path and there is not much distance in both of them.

**2. The car drives according to the speed limit.**

Car drives with max speed limit of 49.5 miles/s.

**3. Max Acceleration and Jerk are not exceeded.**

Car drives with max speed limit of 49.5 miles/s. It brings down speed by 0.224 miles per 0.02 seconds which comes around well below the limit of 10m/s2 acceleration.

Car regains speed when there is more than 30 meters of distance between ego and next vehicle. It regains speed with factor 1.5\*0.224 mph which is again well in limit of Max Acceleration and Jerk.

I have used logic which is explained in Q/A for trajectory generation using spline and with the help of anchor points. It makes trajectory very smooth.

**4. Car does not have collisions.**

Car always try to maintain 30 meters of distance between itself and next vehicle and hence it doesn’t collide to next vehicle. I have implemented additional logic for lane changing which ensures it will change lane only if it is safe to change lane.

**5 The car stays in its lane, except for the time between changing lanes.**

Car changes change lane only in below two conditions

1. It is going collide next vehicle
2. Next vehicle is running very slow.

**6. The car is able to change lanes**

Below are steps of my changing lane algoritham,

**If vehicle is in lane 1 (middle Lane)**

**Step 1:** Check all left lane vehicle and their respective position in future. Future position here denotes to the position which is at end of previous path waypoints. It can be calculate using below formula,

**left\_check\_car\_s += ((double) prev\_path\_size \* .02 \* car\_speed); -- calculate it for all vehicles in left lane**

**Step 2:**

Check ego vehicles future position which is previous path’s last point given by sensor fusion

**Step 3:**

Find out if any of the vehicle from left lane is currently or in future going to be in below two points,

1. Car’s current position – 50 meter:

We want nearest vehicle in left lane to be at least 5 meter behind ego vehicle

1. Car’s future position + 20 meter

We want any vehicle’s future position in left lane to be at least 20 meter ahead of ego vehicle’s future position. This ensures enough distance between ego vehicle and vehicle which is ahead of it after changing lane.

Below is code which implements this logic,

**if((sensor\_fusion[i][5] >(current\_car\_s-5) && sensor\_fusion[i][5] <(car\_s+15))||((abs(car\_s - left\_check\_car\_s) < 20)))**

**{**

**}**

**--Check it for all vehicles**

If Car doesn’t find any vehicle which is going to be between above two points, we will stop checking for other vehicles and just declare that lane change on left side is not possible.

Car will check right lane and again apply all steps from 1st to step 3 for right lane vehicles.

If it is not possible to change lane to right even, Car will maintain lane.

**If vehicle is in lane 0 (Left Most Lane)**

Same logic is applicable to lane change from lane 0 except now car will just have one option to change lane and that is middle lane.

**If vehicle is in lane 2 (Right Most Lane)**

Same logic is applicable to lane change from lane 2 except now car will just have one option to change lane and that is middle lane.

Please check code from line 281 to 450 for lane change logic.

Below is screenshot of completed path within all limits.

