Path Planning Project

Project rubrics

The code met the project rubrics as follow:

- The car is able to drive at least 4.32 miles without incident
- The car drives according to the speed limit
- Max Acceleration and Jerk are not Exceeded.
- Car does not have collisions.
- The car stays in its lane, except for the time between changing lanes.
- The car is able to change lanes

Detail explanation – Reflection

I applied the following step/technique to drive the vehicle safety and comfortably on the highway.

Solution 1:

- I converted the **car_yaw** and **car_speed** to radians and meter per second.
- I prepared base parameters for the trajectory calculation
    - If it's the first time the code was called, the previous path returned by the simulator will be empty. In this case, I will just use the car information returned by the simulator as the base parameters.
    - If previous path existed, I kept 5 points from the previous path and recalculated the rest of the path. This is to avoid the problem caused by the delay in communication between the simulator and the code. The last point of the 5 points kept will be used as the base parameters for further calculation.
    - I calculated the velocity and acceleration in S and D direction of the Frenet's frame as well.
    - I also initiated a spline with the base points.
- I pre-calculated the JMT and the approximated position of the other cars in the highway. This way I only need to look up the result later, which saved a lot of computing time.
- I generated possible trajectory from the base parameters.
    - I referenced the quintic polynomial based trajectory from the following paper "Optimal Trajectory Generation for Dynamics Street Scenarios in a Frenet Frame [1]"
- I used two quintic polynomials for each S and D direction in the Frenet frame.
    - S Direction: JMT ({0, base_car_vs, base_car_as}, {target_s, target_v, target_a}, PREDICT_TIME);
    - D Direction: JMT ({base_car_d, base_car_vd, base_car_ad}, {target_d, 0, 0}, target_d_time);
- Basically, in S direction we have 4 fixed parameters (current position, current speed, current acceleration and target time) and 3 tunable parameters (Target position, target speed and target acceleration)

- In D direction I have 5 fixed parameters (Current position, current speed, current acceleration, target speed which is 0, target acceleration which is also 0) and 2 tunable parameters (target position and target time). I limited the number of tunable parameters in D direction so that the car will likely to move within one lane except when changing lane is the only acceptable solution
- For each generated trajectory, I calculated the cost using the following function.
    - Initiated Cost = (**possible maximum distance** - **target S distance**) * 10000000 + **target D distance** * 1000;
    - This mean penalty for going less distance in S and penalty for going too much distance in D
- Then for each point in the trajectory, I added:
    - cost += sqrt**(different between current speed and maximum speed)** + sqrt(**distance to lane center**) * 10000;
    - This mean penalty for going too slow and penalty for going too far from the lane center
- Also for each vehicle which is close to a point in the trajectory:
    - cost += 80000/abs(**distance between cars**);
    - This mean penalty for going too close to other cars
- There are also a few cases when the trajectory will be immediately disqualified (cost set to maximum)
    - Acceleration in S direction is more than 10m/s^2
    - Acceleration in D direction is more than 10m/s^2
    - Total acceleration is more than 10m/s^2
    - Velocity in S direction is negative
    - Velocity is more than 23.2m/s (which is 50MPH)
    - Jerk is more than 50 m/s^3
- After evaluate the trajectories, I generated some reference points from the trajectory with smallest cost and added them to the spline. The cubic spline interpolation library I used is from this link [2].
- In the final step, I generated the final path based on the spline and send back the data to the simulator.

My first solution could run a few tracks without any problems. However due to how the cost function was designed, there are rare occurrences when all trajectories generated was considered “not good enough”. When it happened, the car’s behavior is undefined and it may or may not cause accidents. Fixing it required changing the cost function, which could easily break other functions. In the end, I decided rebuild everything and make it as simple as possible.

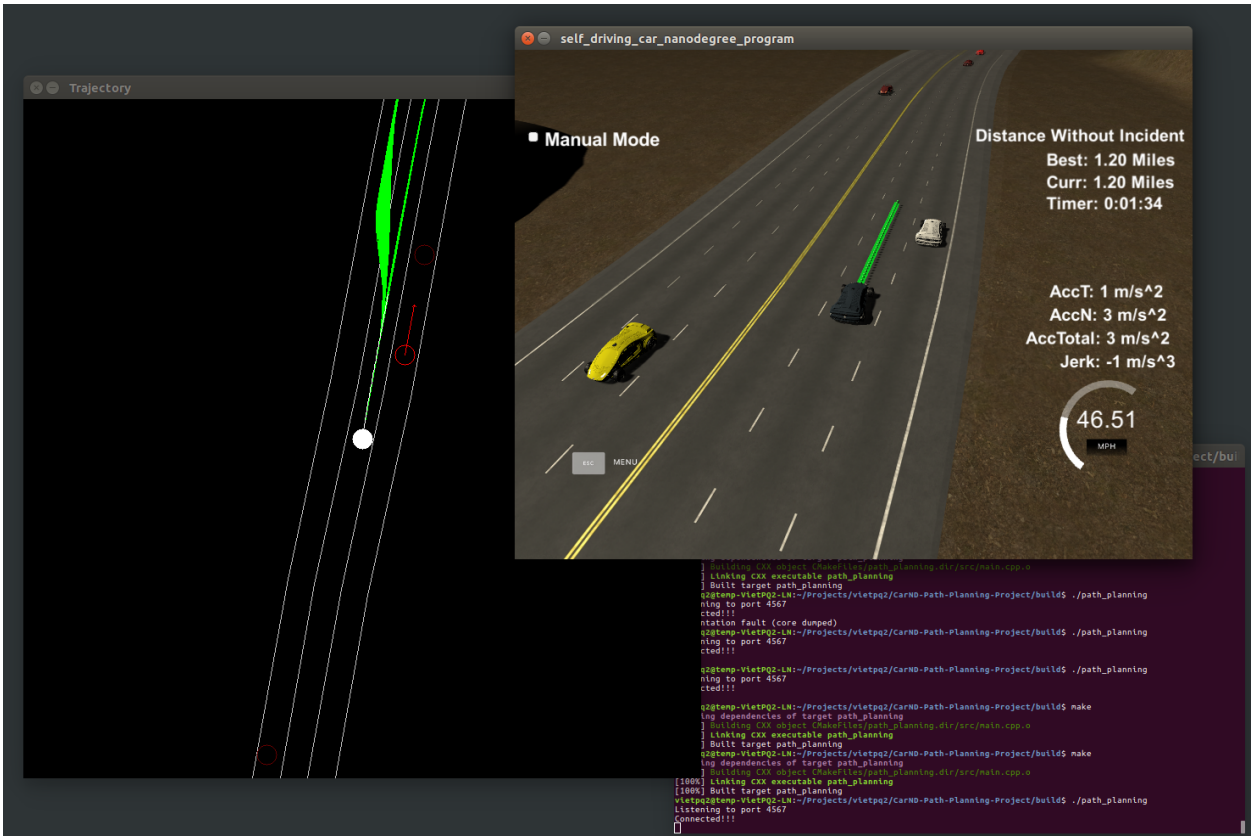
Solution 2 (My submitted solution):

- I converted the **car_yaw** and **car_speed** to radians and meter per second.
- I reused all previously generated points
- I shift/rotate these points to local coordinates and use them as reference points for a spline

- Next, I calculate the feasibilities of 3 possible actions: Keep current lanes, switch to the left lane and switch to the right lane
- Then I select the course of action in the following order
    - If the current lane is not blocked, keep running at the maximum velocity.
    - If switch left is possible, switch to left
    - If switch right is possible, switch to right.
    - Slow down to keep the safety distance between the next car
- After I selected the right action, I build the rest of the reference points (3 more points) on the target lane
- I used the spline to smooth all the selected reference points
- Based on the spline, I select points with distance based on the target velocity
- The points then be reversed back to global coordinates and push back to the final path

Some improvement

The simulator only displays the final path, which is very difficult to debug. I created a visualize tool for the trajectory generation. (The “Debug” folder)
Output of the tool is as below.



There is possibly a bug in the **NextWaypoint** function, which could return out of scope waypoint. I have to add the following code to the function:

```
if (closestWaypoint >= maps_x.size()) {
    closestWaypoint = 0;
}
return closestWaypoint;
```

Video result

Please refer “Capture” folder included in the submission.

- Video.mp4 – Capture one lap running my code which meets the project rubrics

Source code

Please refer “CarND-Path-Planning-Project” folder for my final source code.

References

[1] Optimal Trajectory Generation for Dynamics Street Scenarios in a Frenet Frame: https://d17h27t6h515a5.cloudfront.net/topher/2017/July/595fd482_werling-optimal-trajectory-generation-for-dynamic-street-scenarios-in-a-frenet-frame/werling-optimal-trajectory-generation-for-dynamic-street-scenarios-in-a-frenet-frame.pdf

[2] Cubic Spline interpolation in C++
http://kluge.in-chemnitz.de/opensource/spline/