Project Path Planning

Feb 12, 2020

Mark Gao

# I. Introduction

In this project, the goal is to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit. You will be provided the car's localization and sensor fusion data, there is also a sparse map list of waypoints around the highway. The car should try to go as close as possible to the 50 MPH speed limit, which means passing slower traffic when possible, note that other cars will try to change lanes too. The car should avoid hitting other cars at all cost as well as driving inside of the marked road lanes at all times, unless going from one lane to another. The car should be able to make one complete loop around the 6946m highway. Since the car is trying to go 50 MPH, it should take a little over 5 minutes to complete 1 loop. Also the car should not experience total acceleration over 10 m/s^2 and jerk that is greater than 10 m/s^3.

The following are the files of how I do this.

* main.cpp The main cpp file of this project.
* prediction.cpp This file is aimed to predict the range and target velocity of other cars on the road based on sensor fusion data.
* prediction.h The header file of prediction.cpp.
* vehicle.cpp This file is the realization of Finite State Machine. It determines whether to change lane or stay on current lane.
* vehicle.h The header file of vehicle.cpp.
* cost.cpp This file is to calculate the cost of each state in Finite State Machine.
* cost.h The header file of cost.cpp.
* trajectory.cpp This file is to generate trajectory points based on the result of State Machine, map file.
* trajectory.h The header file of trajectory.cpp.
* spline.h This is the tool to generate spline.
* helpers.h The helper functions.

To run this project, run build.sh to build project. Then run run.sh to run this project.

# II Prediction

In this part, I get range and target velocity of cars in current lane and adjacent lanes which will be used in State Machine.

First, I set two global variable lane and ref\_vel. lane=0 for left lane, lane=1 for middle lane, lane=2 for right lane. ref\_vel for reference velocity. See line 58 and 59 in main.cpp for this.

In prediction.cpp, there are two main functions, prediction\_front and prediction\_left\_right.

In prediction\_front , I get range of the nearest front car and target velocity in current lane. I do this by iterating all the sensor fusion object, if the car’s d value is within current lane and it’s the nearest front car, then get distance between as range and its velocity as target velocity.

In prediction\_left\_right, I get range and velocity of the nearest front and rear cars in adjacent lanes. First, I select cars in ego car's left and right lane by its d value. Then, I divide the cars into those in front of ego car and those behind ego car by comparing its s value. And its range and velocity are also obtained.

# III Finite State Machine

After range and velocity of all the cars surrounding the ego car are obtained, I can choose the best state using Finite State Machine from three states: Keep Lane, Change for Left Lane, Change for Right Lane.

In vehicle.cpp, function vehicle accepts prediction result and return a bool variable change\_lane indicating whether to change lane.

There are three cases we need to consider.

Case 1: If no front car or front car is far away, speed up to max velocity.

Case 2: If the front car is near but it's faster, speed up to max velocity.

Case 3: If the front car is near and it's slower, speed down.

In Case 3, we need Finite State Machine to choose the best state. I take into account three elements in my model, buffer, crash and save time.

For buffer, we need to keep a safe distance between the nearest front car in current lane. So I punish small front distance between front car and ego car.

For crash, we need to keep a safe distance between the nearest front and rear cars in current and adjacent lanes. In this case we not only need to consider front cars in case the front cars suddenly slow down, but also need to consider the rear cars in case the rear cars suddenly speed up and collide with ego car. So I punish small longitudinal distance within certain range between other cars and ego car.

For save time, we need to run at the max speed permitted as much as possible. So I punish low speed.

Check cost.cpp for further details of how to calculate the costs.

First, I calculate the cost for Keep Lane state using range and velocity of front car.

Then, I calculate the cost for Change for Left Lane state and Change for Right Lane state using range and velocity of cars in adjacent lanes.

Finally, I choose the best state with the minimum cost and update the latest lane variable and bool variable change\_lane.

Check vehicle.cpp for further details of how I choose the weight parameters.

# IV Generate Trajectory Points

In this part, I generate trajectory points using map, car’s location, and results from Finite State Machine. And with the help of spline.h, the trajectory points will be smooth, so no lateral jerk will occur.

First, I generate the first several points using car’s position and its yaw.

Then, if lane change is not needed, I set three points in front of ego car, each point 30 meters further. If lane change is needed, I set three points with 50, 70 and 90 meters respectively to get more smooth trajectory and avoid large lateral jerk.

Next, I convert the global coordinate system to the vehicle coordinate system and generate a spline.

Finally, trajectory points are generated within the iteration of path size.

See trajectory.cpp for further details.

# V Conclusion

The path planner described above can safely navigate around a virtual highway. The path is smooth and no collision occurs.

However, during the process of lane changing, when other cars are also changing lanes, the planner sometimes can not make success planning and cause collision.

So, the path planner still has room for improvement.