Udacity SDCND – Term 3 – Project 1

Reflection on Path Planning Project

Jelena Kocić

Introduction

The scope of the project is the path planning for the highway. The goal is to drive the car at least 4.32 miles without incident. Speed limit is 50 MPH, max acceleration and jerk are 10 m/s^2 and 10 m/s^3, respectively. The car must not come into contact with any of the other cars on the road. The car stays in its lane, expect the time between changing the lanes. And the most important, car is able to change the lanes when it make sense to do so, e.g. when the car is behind a slower moving car and an adjacent lane is clear of other traffic.

Driving the car

As a result of this project we have fully operable car that is driving more than 4.32 miles without incident. On the YouTube is the video of this solution, in this video duration ride is over 5 miles without incident.

The map of the highway is in data/highway_map.txt. Each waypoint in the list contains [x,y,s,dx,dy] values. x and y are the waypoint's map coordinate position, the s value is the distance along the road to get to that waypoint in meters, the dx and dy values define the unit normal vector pointing outward of the highway loop. The highway's waypoints loop around so the frenet s value, distance along the road, goes from 0 to 6945.554.

For creating smooth trajectories we was using http://kluge.in-chemnitz.de/opensource/spline/, file spline.h. We use the spline to generate the smooth transition from 0 MPH to 49.66 mph target as can be seen in the main.cpp code, lines 495-549.

Changing the lane should happened when the car in front of us is slower and when we have free lane left or right to go, lines 302 – 439. First step was to detect if the car is slower, and if it is to reduce speed of our car, and set change_lanes flag on true, lies 317 – 338.

At the beginning of code design first idea was that we start from the middle line, the first next step was to reduce speed and/or change to the left line. Code snapshot:

```
// for i-th car on the road
for(int i = 0; i < sensor_fusion.size(); i++)
{
  // i-th car is in my lane
  float d = sensor_fusion[i][6];
  if(d<(2+4*lane+2) && (d>(2*4*lane-2)))
  {
    double vx = sensor_fusion[i][3];
    double vy = sensor_fusion[i][4];
    double check_speed = sqrt(vx*vx+vy*vy);
    double check_car_s = sensor_fusion[i][5];

    check_car_s += ((double)prev_size * 0.02 * check_speed);

    // if the car is in front of us and if gap is smaller than 30 m
    if((check_car_s > car_s) && ((check_car_s - car_s) < 30))
    {
      //ref_vel ref_vel = 15.5; //[m/s]
      too_close = true;
      if(lane > 0)
      {
        std::cout << "Changing the lane" << std::endl;
        lane = 0;
      }
    }

  }
}

//22.2 is for max speed
if(too_close)
{
  ref_vel -= 0.08;
  //std::cout << "Car in front is too close -> reducing the speed" << std::endl;
}
else if(ref_vel < 22.2)
{
  ref_vel += 0.08;
}
```

This solution was almost ok. It was working well in case that we do not need requirement “the car is able to change the lanes”. Using this approach, we succeed to drive more than 5 miles without collisions or other issues. However, this solution does not include the main goal of this project, path planning and changing the lanes as result of this decisions. In next chapter the implemented path planner is described.

Path planning

Path planning is done as a simple algorithm, main.cpp, lines 358 – 439. The path planning algorithm will only change lanes if it is necessary, change into the lanes that have no vehicles ahead. We will change the lane if we have a car at 30m in front of us and if it safe to change the lane. The safe means that 20m in front and 10m behind left or right lane there is no car, line 379 and line 416. Also, we have an additional flag try_left. This flag provides us the flexibility in lane change. Without this we will always first check if car can move to the left line, and then check for the right lane. Introducing the try_left we will each time when we have the car in front of us, use the different lane (left or right) to check for going in.

The decisions from the path planning algorithm are passed to the trajectory generator, lines 443 – 491. Then, the path is smooth one last time, as can be seen in main.cpp lines 495 – 551, and sent to the simulator.

At the first picture there is a snapshot of the ego-vehicle driving more than 5 miles.

We have several debug lines, as can be sawed in the bottom picture. E.g. we are counting the number of changing lines, also we see for how long the car is trying to change the line, but it doesn't do so because it is unsafe.