CarND-Path-Planning-Project

Self-Driving Car Engineer Nanodegree Program

Project Goal

The goal of this project was to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit. The inputs are:

- the car's localization;
- sensor fusion data (location and velocity data from other cars);
- a sparse map list of waypoints which describe the highway's path

The car should try to go as close as possible to the 50 MPH speed limit, which means passing slower traffic when possible. 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/sˆ2 and jerk that is greater than 10 m/sˆ3.

Considerations

1. The car uses a perfect controller and will visit every (x,y) point it recieves in the list every .02 seconds (50H<). The units for the (x,y) points are in meters and the spacing of the points determines the speed of the car. The vector going from a point to the next point in the list dictates the angle of the car. Acceleration both in the tangential and normal directions is measured along with the jerk, the rate of change of total acceleration. The (x,y) point paths that the planner receives should not have a total acceleration that goes over 10 m/sˆ2, also the jerk should not go over 50 m/sˆ3.
2. There will be some latency between the simulator running and the path planner returning a path, with optimized code usually its not very long maybe just 1-3 time steps. During this delay the simulator will continue using points that it was last given, because of this its a good idea to store the last points used to achieve a smooth transition. previous_path_x, and previous_path_y are helpful for this transition since they show the last points given to the simulator controller with the processed points already removed.

Implementation

Making sense of other cars

One of the first things to do on a new iteration is to process information about what the car is sensing on the road. In particular, to make sense of the other cars driving in the same direction as the ego vehicle. A `Car` class was implemented to make this an easier task work on.

Each time a new car is spotted, it's added to an internally kept `Car` library. This way we can keep track of data about other cars from past iterations, if we wish. When a car's location is updated, its current lane is automatically computed as well. When a car's velocities are updated, its absolute velocity and corresponding MPH conversion are also computed.

With these data on hand, two sets of data are computed: * the flags for warning of a car ahead, to the left or to the right; * the speed of each lane.

Flagging cars around ego vehicle For each sensed car we predict what will be its location in the future. How far in the future coincides with how many points in the past trajectory were left "unconsumed". As an example, let us consider that 40 points of the previous trajectory were not consumed. Trajectory points are consumed with a frequency of 50Hz (or every 0.02s). We will predict where the car will be in 0.8s (40 * 0.02s). If this future location is ahead and within 30m of the current location of the ego vehicle, then we consider there is a car ahead. It should be noted that we're only considering the `s` component of Frenet coordinates of both vehicles.

A similar approach is taken for the car to the left and to the right flags. This time, however, we consider 30m ahead and 15m behind the ego vehicle.

Lane speeds

keep track of lane speed

Decision making

The problem at hand didn't require complex decision making to hit all criteria. We want to move as fast as possible in a legal (observing speed limit and keeping within lane limits), safe (avoiding collisions) and confortable manner (not accelerating or jerking hard). Legality and safety are addressed in this section. Keeping lane limits and confort are addressed in the trajectory generation section.

Before going on about decision making, some considerations about the implementation must be introduced. The maximum speed that the car will ever

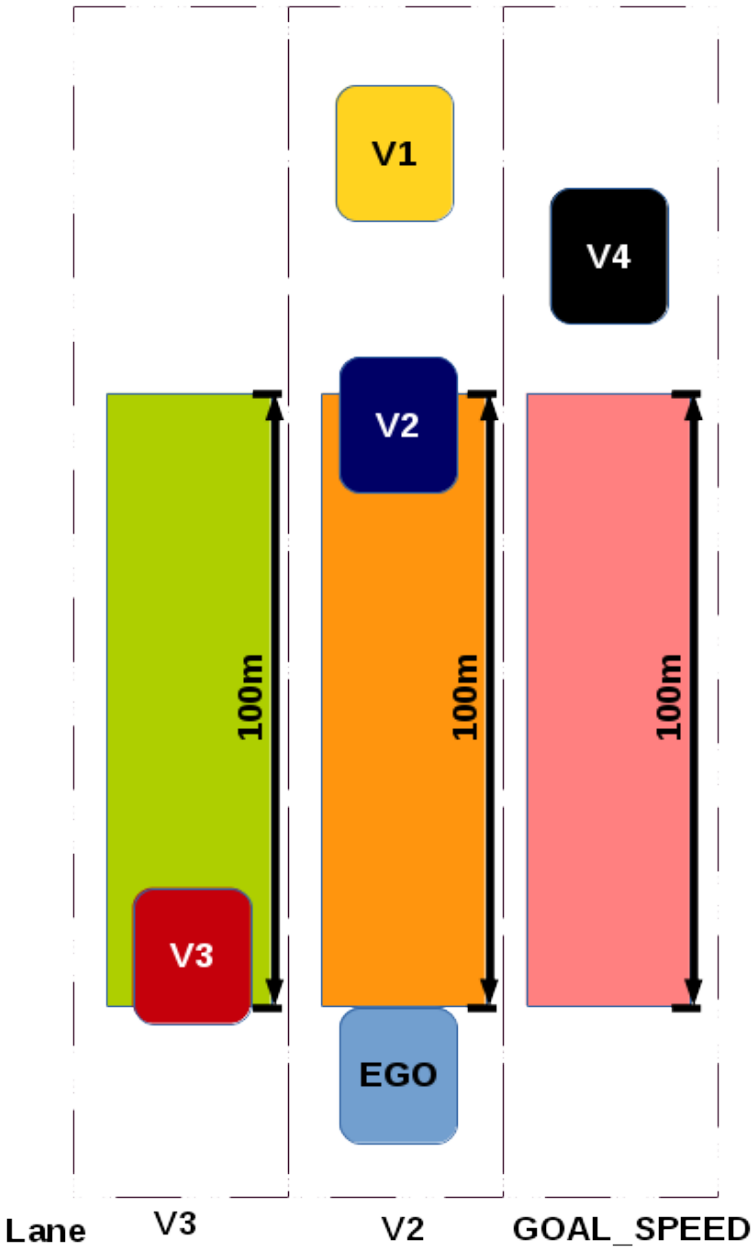

Figure 1: picture alt

drive (henceforth referred to as the `GOAL_SPEED`) is 49.5 MPH, just under the legal limit of 50 MPH. That is the desired speed at any given time. If it is not possible to drive at that speed, we want to drive at another legal, safe speed, henceforth referred to as `TARGET_SPEED`. `TARGET_SPEED` is, then, our goal whenever `GOAL_SPEED` is not possible. The reference speed (`REF_SPEED`) is the real speed the car will be driving. In the beginning `REF_SPEED` is 0 since the car is stopped in the highway. Overtime, it will gradually rise or fall given a `TARGET_SPEED`, always respecting the maximum acceleratons constraints.

To avoid collisions the chosen strategy was to slow down whenever a car ahead is detected. The `TARGET_SPEED` will be that car's speed.

`REF_SPEED`	actual speed the car's controller will implement
`TARGET_SPEED`	speed the controller is working torwards
`GOAL_SPEED`	ideal speed, always 49.5 MPH

Trajectory generation

A really helpful resource for doing this project and creating smooth trajectories was using http://kluge.in-chemnitz.de/opensource/spline/, the spline function is in a single hearder file is really easy to use.

The code

Making sense of other cars

Decision making

Trajectory generation

Results

Over 6.10 miles, 7 minutes and 44 seconds and 4709 readings, the average speed was 47.53 MPH