Path Planning - Project Report

Patrick Cleeve

Objective

The objective of this project is to develop a path planning algorithm to successfully drive the vehicle around the simulator highway track.

The project code is available here:
https://github.com/patrickcleeve/CarND-Path-Planning-Project

A video of successful completion is available here:
https://www.youtube.com/watch?v=KU35txOIIRA

Path Planning

We implement a path planning algorithm capable of providing smooth and safe driving behaviour on a highway with other traffic. We use a state machine approach to control vehicle behaviour, and an interpolated spline to generate trajectories.

Prediction, Behaviour and Trajectory Generation

There is a set of three tasks to accomplish to succeed in path planning. These are:

Prediction: The ability to determine where other vehicles are on the road, and where the will be in the future.
Behaviour Planning: The ability to decide what to do, based on what you want to do, and what others are doing.
Trajectory Generation: The ability to plan a path to get where you want to go.

In order to drive successfully on a highway we will need to implement each of these abilities for the vehicle.

Map, Localisation and Sensor Fusion

The simulator environment consists of a looped highway map, constructed from a series of waypoints for the vehicle to pass through. The simulator provides a set of sensor fusion and localisation data about the other vehicles on the road.

Localisation data from the controlled vehicle includes the following attributes:

- car_x: car's x position in Map coordinates
- car_y: car's y position in Map coordinates
- car_s: car's s position in Frenet coordinates
- car_d: car's d position in Frenet coordinates
- car_yaw: car's yaw angle in map coordinates
- car_speed: car's speed in mph

Sensor fusion data from all the other car's on the road includes the following attributes:

- car_id: car's unique id
- car_x: car's x position in Map coordinates
- car_y: car's y position in Map coordinates
- car_vx: car's x velocity in m/s
- car_vy: car's y velocity in m/s
- car_s: car's s position in Frenet coordinates
- car_d: car's d position in Frenet coordinates

Coordinate Systems

We use three different coordinate systems throughout the project for different applications.

Global Map Coordinates (x,y)

The global map coordinate system is a cartesian (x, y) coordinate system.

This coordinate system is used for map waypoints, and the final trajectory fed to the controller.

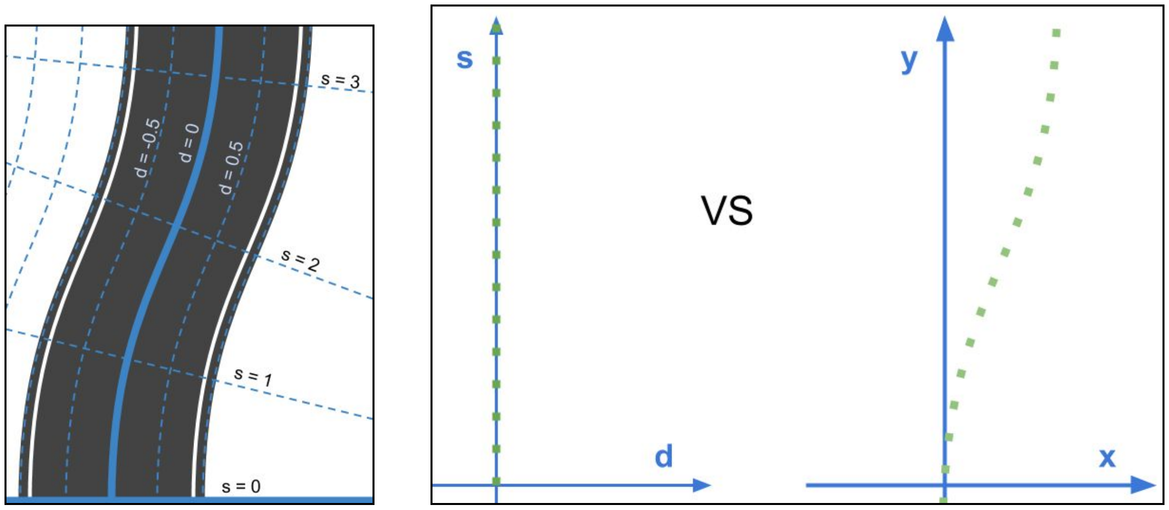
Frenet Coordinates (s,d)

As roads are not aligned in x,y coordinates, the road path is some complicated function of x and y. Instead of using this complicated function to determine position or velocity on the road, we can instead define a coordinate system along the road. In this coordinate system (Frenet) we define s coordinates as distance along the road (longitudinal displacement), and d coordinate as side to side position (lateral displacement).

As a result, a vehicle travelling in the same lane, at constant speed can be modeled as simply moving in a straight line. This greatly simplifies our collision detection as we can use some simple rules of thumb:

- Vehicles with similar s coordinates are close along the road.
- Vehicles with similar d coordinates are in the same lane.

We use this coordinate system for prediction, and behaviour planning (collision).



Frenet Coordinates (Udacity Lecture)

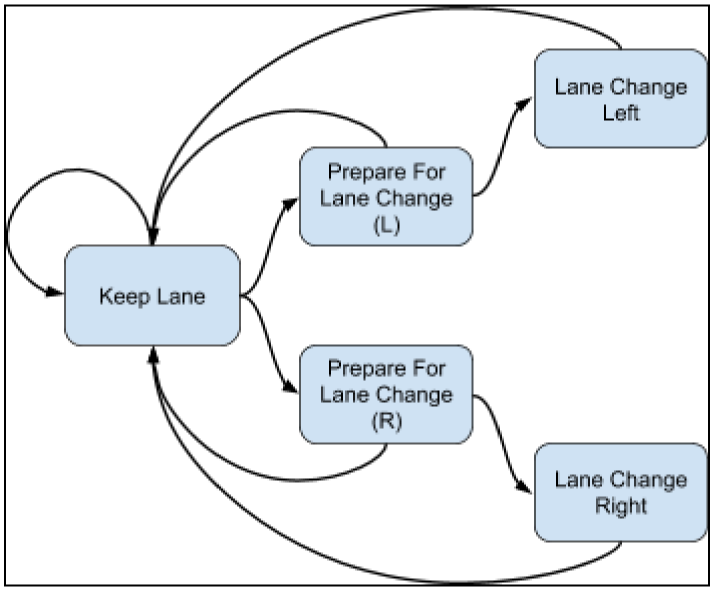
Car Coordinates (x,y, ψ)

To simplify trajectory calculation, we attach a coordinate frame to the vehicle. As with the MPC project we shift the car's angle to zero (in direction of motion) to simplify trajectory calculation.

We use this coordinate system to calculate the vehicle's trajectory. The trajectory is then transformed back into the global coordinate system before being passed to the controller.

Finite State Machine

To control the behaviour of the vehicle in the highway driving scenario we use a finite state machine. The vehicle moves between states based on certain conditions which are detected using the sensor fusion data. The diagram below shows states and transitions between each.



Path Planning Finite State Machine

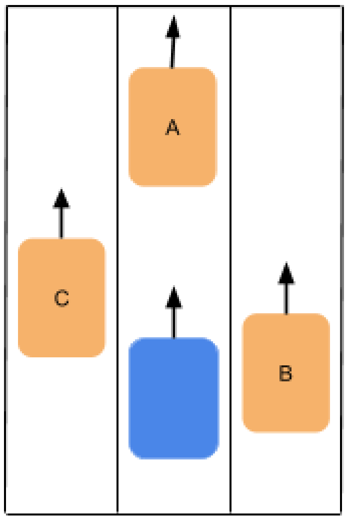
The behaviour of the vehicle in each state is described below:

Keep Lane

The vehicle begins in this state, at a reference velocity of zero (stationary). Each simulator cycle the vehicle will check for vehicles in the current lane. If no other vehicles are detected in the lane area ahead, the vehicle will attempt to accelerate. Whilst no vehicles are in the collision zone ahead, the vehicle will continue to accelerate up to the maximum speed limit.

When a vehicle is detected in the collision zone ahead (based on vehicle's current velocity and trajectory), the vehicle will decelerate and prepare to change lane.

If a lane change is not possible, the vehicle will continue to decelerate until it matches the velocity of the vehicle ahead of it in the lane (The blue vehicle will match velocity with vehicle A, as no lane change is possible). The vehicle matches velocity to prevent “yo-yo” behaviour, where the vehicle decelerates until vehicle A is outside of the collision zone, then accelerates until it detects it in the zone, and so on.

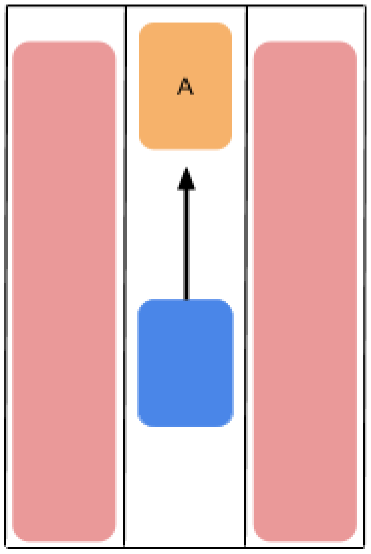


Once a lane change has successfully been completed, the vehicle returns to the keep lane state. The vehicle will stay in the keep lane state unless a vehicle is detected in the current lane ahead.

Prepare for Lane Change (L/R)

Once the vehicle detects a vehicle ahead of it in the current lane, it prepares to change lane.

The vehicle runs through all sensor fusion data ands checks to see if any vehicles are in the zones in the target lane (see diagram, illustration only). This check is run for all possible lanes (e.g. will check left and right lanes if in middle, but only middle lane if in left lane).



The state transition is pessimistic, meaning it will not change lane unless no vehicles are detected in the target lane zone, and all the sensor data has been checked. The vehicle attempts to change left before right, to maintain ‘overtaking on the left’ rule of the road.

If the target lane is clear, the vehicle will transition to the lane change state, and begin to plan the trajectory to move to the new target lane.

If no lane change is possible, the vehicle will return to the Keep Lane state.

Lane Change

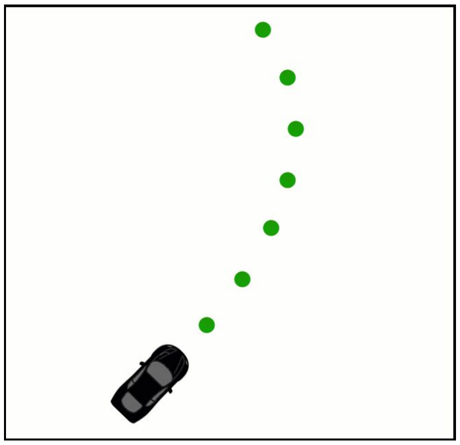
When a lane change has been cleared, the vehicle will change the target lane, and pass the updated lane (d coordinate) to trajectory generation. The vehicle may only change one lane at a time.

Once the vehicle has begun to change lanes, the vehicle will be prevented from initiating any other lane changes until the current has finished. A lane change is determined to be finished when the car_d value is no longer changing significantly, and is close to the centre line of the target lane.

When the lane change has successfully completed, the vehicle will return to the Keep Lane state.

Trajectory Generation

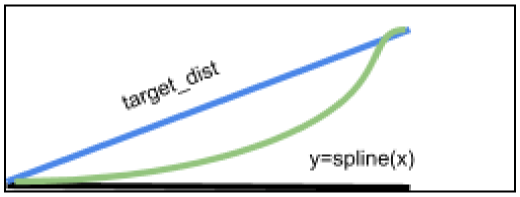
The motion of the vehicle is controlled by constructing a trajectory. The trajectory is a series of points in global coordinates (see image), represented in green in the simulator. The simulated vehicle travels perfectly (no controller) between each subsequent dot every 20ms, with rotation as the line between points. Therefore we can adjust the velocity (and thus acceleration and jerk) by changing the distance between points. The greater the distance between points, the greater the velocity (v = dx/dt).



For each simulator cycle we plan a trajectory of up to 50 points, starting from the vehicle. If a previous trajectory exists, which has not been reached by the vehicle, we generate additional points and add onto the end of the previous path (up to a maximum of 50 points).

We use the spline library to generate a series of trajectory points, using the following procedure:

- Set three initial reference points (at car_s +30, 60 and 90, in the target lane).
- Transform the reference coordinates to car coordinate space (shifting angle to zero) to simplify the math.
- Fit reference points to spline function.
- Calculate number of points required per distance travelled to travel at reference velocity (method from Project Q&A).
    - $n_{points} = \frac{target\ dist}{(time\ per\ point * reference\ velocity)}$
- Evaluate each trajectory point using spline function
- Transform trajectory to global coordinate system, append to list of x,y values.

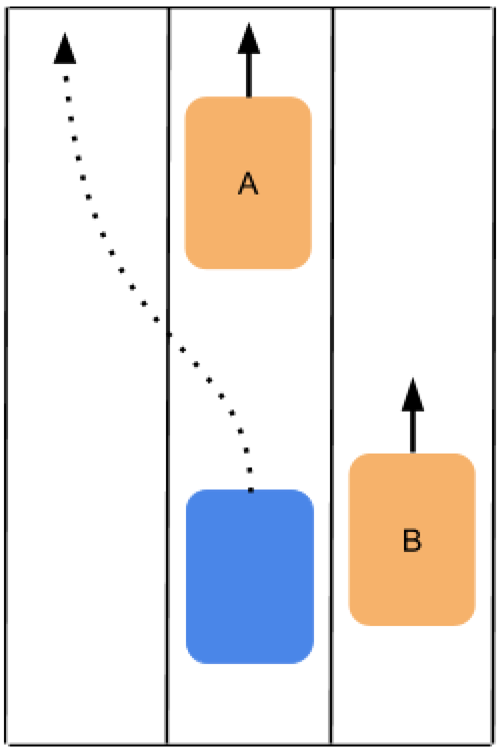


Once the full trajectory has been computed it is passed to the simulator for the vehicle to follow.

Assumptions

Sensor Data and Localisation

The sensor and localisation data is updated and available at each timestep for each of the other vehicles on the road with perfect accuracy and no noise. In reality, tracking multiple vehicles on a busy highway, would be significantly more difficult and would result in a higher degree of uncertainty about other vehicles. This would likely change a number of assumptions, particularly around ensuring the lane is clear for a lane change.



Perfect Controller

Vehicle does not use a controller, simply travels perfectly from each trajectory point. This is an obviously unrealistic assumption. In the real world, a controller (such as previous MPC project) would be used to try to match the generated trajectory producing additional uncertainty and latency.

Latency Handling

All latency between commands, and calculations is handled by the simulator. The vehicle continues to follow the “stale” path between updates. In reality the time between updates could be significantly longer and more uncertain.

Predictability

The vehicles in the simulator behave relatively predictable, including travelling at fairly consistent speeds and staying in their lane. This makes their behaviour relatability easy to plan for. However, in reality other drivers would be nowhere near as consistent, being prone to rapidly change lane, drift across lane lines and accelerate/decelerate rapidly.

Extensions

Long Term Planning

The vehicle does not currently do any long term planning, it simply tries to pass a vehicle going slower than the speed limit, if it is safe to do so. This can often result in the vehicle changing lanes to then soon become stuck behind another vehicle, and subsequently ‘boxed in’ unable to speed up or change lane. The vehicle should look further ahead, when evaluating lane changes to check for these kinds of situations further ahead in the target lane.

Cost Function

The vehicle currently does not use a cost function to decide which lane to change into (it tries to go left, then right). This could likely improve performance, and help stop boxed in problem if were smarter about evaluating future, as mentioned above.

Deterministic Behaviour or Scenarios

During development the vehicle would sometimes encounter different situations or scenarios, where it would have a collision because of some unexpected interaction. Even once the issue had been resolved, it was impossible to replicate situations or scenarios with the simulator, as cars are randomly generated. It would be useful to be able to construct scenarios to be able to test vehicle in lots of different edge case situations.

Discussion

It was difficult to reason exactly about how the car will behave in different scenarios. The code can't really try to program each scenario as it will inevitably miss one or misrepresent the problem. However, starting from a pessimistic/unsafe assumption makes it easier to constrain behaviours. You can be more confident car wont do something very unexpectedly, and will default to staying in its lane and slowing down. Defaulting to most safe state, helped to avoid collision above all else.

Currently symmetric zone check for lane changes, with a very conservative measurement (probably too conservative). We could adjust the box around car, as don't really need to evaluate that far ahead as behind, because cars coming from behind is a more dangerous scenario for us when changing lanes (velocity in direction of us, as opposed to away).

Unexpectedly the highly conservative lane change is acting as a sort of cost function, it raises the bar required to change lane. However, it does mix two different things together if trying to control higher level planning and state machine together. Currently the lane change is really dumb, and changes if it can and is safe to, not evaluating the future. That approach is mixing two different questions together:

- Is it safe to change lanes?
- What is the best path to take?

It is probably better to create a longer term planner to evaluate cost of changing lanes in future by looking at traffic far ahead. That might be able to prevent the boxed in behaviour from occuring as frequently.

Reference

Cubic Spline Interpolation in C++, https://kluge.in-chemnitz.de/opensource/spline/

Path Planning Walkthrough-Edit (Project Q&A Video),
https://www.youtube.com/watch?v=7sI3VHFPP0w

Udacity Classroom, (Images)
https://classroom.udacity.com/nanodegrees/nd013/parts/6047fe34-d93c-4f50-8336-b70ef10cb4b2/modules