PATH PLANNING PROJECT

12-04-2020

# Overview

## Project Description

In this project we will implement path planning algorithm to safely navigate through the highway. Here the traffic is driving at +-10 MPH of the 50 MPH speed limit. Car's localization, sensor fusion data, and sparse map list of waypoints around the highway is also provided in real time.

## Project Scope

* The car will try to go as close as possible to the 50 MPH speed limit, which means passing slower traffic when possible.
* The car will 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 will be able to make one complete loop around the 6946m highway.
* The car will drive 4.32 Miles at minimum without any incident.

## High-Level Requirements

The new system must include the following:

* cmake >= 3.5
* All OSes: [click here for installation instructions] (https://cmake.org/install/)
* make >= 4.1
* Linux: make is installed by default on most Linux distros
* Mac: [install Xcode command line tools to get make] (https://developer.apple.com/xcode/features/)
* Windows: [Click here for installation instructions] (http://gnuwin32.sourceforge.net/packages/make.htm)
* Gcc / g++ >= 5.4
* Linux: gcc / g++ is installed by default on most Linux distros
* Mac: same deal as make - [install Xcode command line tools] ((https://developer.apple.com/xcode/features/)
* Windows: recommend using [MinGW] (http://www.mingw.org/)
* [uWebSockets] (https://github.com/uWebSockets/uWebSockets)
* Run either `install-mac.sh` or `install-ubuntu.sh`.
* If you install from source, checkout to commit `e94b6e1`, i.e.
* git clone https://github.com/uWebSockets/uWebSockets
* cd uWebSockets
* git checkout e94b6e1
* To setup the Environment on Windows - [here](https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/1c4c9856-381c-41b8-bc40-2c2fd729d84e/lessons/3feb3671-6252-4c25-adf0-e963af4d9d4a/concepts/23d376c7-0195-4276-bdf0-e02f1f3c665d).
* To setup the Environment on Linux - [here](https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/1c4c9856-381c-41b8-bc40-2c2fd729d84e/lessons/3feb3671-6252-4c25-adf0-e963af4d9d4a/concepts/07be3ef0-e444-4080-b766-4a454185529a).
* To setup the Environment on Mac - [here](https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/1c4c9856-381c-41b8-bc40-2c2fd729d84e/lessons/3feb3671-6252-4c25-adf0-e963af4d9d4a/concepts/7d235103-5c4d-4001-a227-5ad2ac43bfff).

## Implementation Strategy

## Vehicle Predictions additional\_helper\_func.cpp [171:236]

For every cycle, Predict the location of the cars on each lane [Lane 0, Lane 1, Lane 2].

Pseudocode:

struct Vehicle {

string position = "None";

double s = -1;

double speed;

double current\_distance\_from\_ego = 500.0;

double future\_distance\_from\_ego = 500.0;

};

vector<Vehicle> active\_predictions;

for each data in sensor\_fusion:

if the car is current\_lane as the Ego Vehicle:

add the vehicle details to active\_predictions[0]

name it as "Front"

if the car is current\_lane - 1 w.r.t the Ego Vehicle:

add the vehicle details to active\_predictions[1]

name it as "Left"

if the car is current\_lane + 1 w.r.t the Ego Vehicle:

add the vehicle details to active\_predictions[2]

name it as "Right"

Here we check if there is any car ahead of us, or left or right of us. For this we create as Vehicle Structure that stores all attributes of the car.

We constantly update the vehicle parameters if there comes a car, which is even closer than the present one.

## Successor States additional\_helper\_func.cpp [150:168]

We will look at the Lane predictions, based on possible successor states.

For ex. 1. If the car is in Lane 0, it can only move to right side or be in same lane.

2. If the car is in Lane 2, it can only move to left side or be in the same lane.

Pseudocode:

Here we return the possible states for Ego vehicle.

vector<string> states;

if (current\_lane == 1){

states.push\_back("LCL");

states.push\_back("LCR");

}

else if (current\_lane == 0) {

states. push\_back("LCR");

lane\_cost[2] = INVALID\_LANE;

}

else if (current\_lane == 2) {

states.push\_back("LCL");

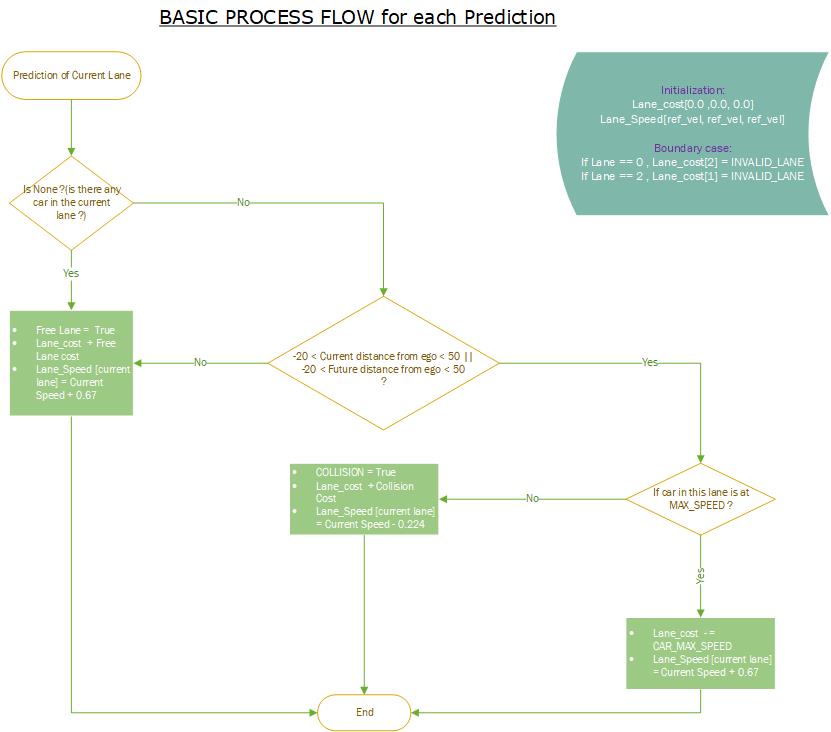
lane\_cost[0] = INVALID\_LANE;

}

If the Ego vehicle is in extremes, we will increase Lane Cost of other Extreme.

## Behavior Planner additional\_helper\_func.cpp [18:147]

Here is the Basic Flow chart of process that will be done for each prediction



Pseudocode:

vector <double> lane\_cost (3, 0.0);

vector<string> new\_states = successor\_states (current\_lane, lane\_cost);

vector <double> lane\_speed(3, ref\_vel); // ref\_vel == Current Velocity of the Ego vehicle

if car ahead:

update lane\_cost[current\_lane] and lane\_speed[current\_lane] as described above

if car on left:

update lane\_cost[current\_lane-1] and lane\_speed[current\_lane-1] as described above

if car on right:

update lane\_cost[current\_lane+1] and lane\_speed [current\_lane +1] as described above

best lane = min(lane\_cost)

if best lane cost == current lane cost:

new lane = current lane

ref\_vel = lane\_speed[new lane]

else:

new lane = best lane

ref\_vel = lane\_speed [new lane]

The output of this function decides what will be the behavior of the Ego Vehicle.

It can be either Change Lane to left or right or Be in same lane.

## Trajectory Calculation using Spline main.cpp [115:220]

As shown in Project QnA we use spline tool to calculate trajectory points based on the speed, car frenet coordinates and Lane from the previous function

For smooth transition we will use points from previous trajectory (preferably 2). We will add 2 more points which denotes the intermediate and last point in the trajectory, which will be 60m apart.

Then we will use spline tool from spline.h to calculate the remaining points, but before that we will transform these points from global coordinates to car coordinates so that car is pointing at 0 degrees and points are straight ahead of the car. These points will be transformed back after spline calculation.

Since, it is mentioned that the car will cover each point in 0.02 sec. The Number of point will decide the velocity. Inverse is also true, for this we will use Current velocity to calculate number of points.

# Results

