Path Planning

Reflection

In the path planning project, we are required to provide a set of points to the simulator and the simulator will execute those at interval of 0.02 seconds. As an overview, the code (main.cpp) sends 80 points of the planned path to the simulator, the simulator executes some of the path-points and sends back the unused path-points back to the code. The code again reuses those unused points in its new set of path-points with another set of new path-points such that the total number of points always remains 80.

In the code, we get the `prev_size` from "previous_path_x" information provided by the simulator. If there is some path-points left, we update the car's **s** value. Next in line 250 & 251 we have 2 variables, `too_close` & `proximity[]` whose initial values are false.

- `too_close` becomes *true* when there is a car in front of the ego vehicle within 30m range. (line 266)
- `proximity[]` is an array which stores information about the proximity of cars in each of the three lanes within 30m range in front and 5m in back direction, of the ego vehicle. The lane numbers are mapped to indices of this array and if there are one or more cars within the proximity of 30m + 5m, the corresponding value in the array is change to *true.* (lines 270-275)

Lane change is done only when the car has `too_close` == true and the `proximity[]` of its adjacent lanes are **false.** For lane change to take place we only change the value of the `lane` variable to its corresponding lane number.

Next we have two vectors, namely `ptsx` & `ptxy` and insert x,y points from previous path (if any of them is left). After that we use the function `getXY()` to generate x,y coordinates for 30m, 60m and 90m from the current position of the car.

Then the points in `ptsx` & `ptxy` are converted to the car's reference coordinate system and fitted to a spline. The spline is a piecewise polynomial which can fit to a given set of points. The library is taken from spline tool.

Next from line 380 – 400, we generate rest of the points in between 30m, 60m and 90m anchor points and transform back world coordinate system and insert them to `next_x_vals` and `next_y_vals` vectors which are then sent back to the simulator as path-points.

So as an overview we can say that the ego vehicle starts from zero speed to reach a speed of approximately 45mph. Then if it has a vehicle in front of it going with a slower speed, it checks whether it can move on to the adjacent lanes. If the adjacent lanes are empty (means there are no vehicles in proximity of 30m front and 5m back) and it can change lanes without collision, it opts to change lanes. Else if there are other vehicles in adjacent lanes, it deaccelerates and stays in its current lane.