Decision and Planning

This project implements a behavioural planning and a motion planning algorithm to enable high level collision avoidance, junction management and path following for an autonomous vehicle in a simulated environment. We have used CARLA simulator for the simulated environment and our aim is to plan motion and behavior for an ego vehicle.

Project Members and Contribution

Name	Marticulation Number	Contribution
Jenish Thapa	k12137169	Behavior Planning and Simulation Analysis
Prasil Adhikari	k12049801	Velocity Profile Generation and Parameter Selection
Christoph Domberger	k51849497	Path and Trajectory generation using cubic spirals
Ukleja Sebastian	k0512011	Velocity Profile Generation and Path Trajetory

Set Up

1. Install the requirements from requirements.txt file better approach is to to install it in a fresh conda environment

```
pip install -r requirements.txt
```

CARLA Setup

1. Download the simulator CARLA 0.9.10
2. Decompress the file in a directory of your preference.
3. Enter the decompressed folder, open a new terminal and type:

```
.\CarlaUE4.exe -quality-level=Low
```

Running the project

1. Run the CARLA simulator on **Low Level quality**.
2. Navigate to the Project folder and activate conda environment if you have created one.
3. Run python script SimulatorAPI.py

Behavioural Planning

Behavioural planning is achieved based on an Finite State Machine algorithm. Finite State Machines (FSMs) are a type of rulebased system that makes decisions based on a finite number of states. We have a given set of states and transitions that will be triggered by specific events. There are three states in this algorithm:

- Lane Following
- Decelaration To Stop
- Stop

The initial state of ego vehicle is Lane Following, if there is junction then the state will transit to Deceleration To Stop.

When the state is Deceleration To Stop, we check whether the ego vehicle distance to stop sign is below than the threshold distance to stop sign and if that is the case then the state will transit to Stop.

If the state of ego vehicle is Stop and if it has been in the stop state at time greater than the time requiredfor the ego vehicle to stop then the state will then transit to Lane following.

states

Path and Trajectory Generation using cubic spirals

Calculation of the path is done based on cubic spirals. Cubic spirals are third grade polynomial equations that describe the curvature along a path. Our path has a position, a heading and a curvature. The curvature it self has constraints that must be satisfied to count as a valid path, in our case this is the minimal turning radius of the vehicle. In case of lane following, having the same curvature as the road is considered optimal.

Offset Goals

When following a lane we follow a spiral, the next goal for the path lies in the center of the lane with the curvature similar to the road. But this is only valid if there are no obstacles and no chances of collisions with other cars or road blocks. To be able to calculate an alternative spiral if there is chance of collision, we need offset goals that can alternatevly be achieved if our main goal is not possible. The offset goals must be perpendicular to the main goal heading and are lying within a constant distance along a line left and right to the main goal.

Equations and variables:

- **Xoffset** and **Yoffset**: These represent the x and y coordinates of each offset goal.
- **Xcenter** and **Ycenter**: These represent the x and y coordinates of the center goal.
- **goal_number**: This variable represents the index of the offset goal. It can take values from -n to n, where n is the maximum number of offset goals you want to generate. For example, if n is 2, the goal_number will be -2, -1, 0, 1, and 2.
- **Offset_distance**: This is the distance between the center goal and each offset goal. It determines how far each goal will be positioned from the center goal.
- θ: This represents the yaw angle of the curavture.

The equations for calculating the x and y coordinates of the offset goals are as follows:

Xoffset = Xcenter + goal_number *Offset_distance* cos(θ)

Yoffset = Ycenter + goal_number *Offset_distance* sin(θ)

Collision Checking

One of the most effictive ways of collision checking is using circle detection. The vehicle will be enclosed in a set of circles in a way that the whole vehicle shape is covered when viewed from a top down perspective. Three circles are optimal for a regular compact car.

For collision to not occur, the distance between center circle of the vehicle and obstacles/actor center circle should be smaller than the sum of radii of these two circles.

Spiral Cost Function

Spiral cost function aims to assign a score to each path, with poor paths that are in collision or too close to static obstacles receiving high cost, while low cost are assigned to paths that are closer to the center-line of the global path.

In our implemnetation spiral cost function is the sum of cost due to collision and cost depending on how much spiral curve is close to main goal/center-line of the global path.

If there is no collision then the cost due to collision is zero but if there is any collision then the cost due to collision of the curve is infinity.

For cost based on distance to the main goal, the last point on the spiral is used to check how close we are to the main (center) goal.

$\text{cost due to distaince from the main goal} = \frac{2.0}{1+\exp(-\text{dist})} - 1$

where dist is distance between last point on spiral and main goal

Velocity Profile Generation

The velocity profile should give us a policy to which degree the vehicle accelerate or decelerate. It gives a velocity trajectory from a starting speed to a desired speed.

It works in unison with the Behavioral plannner as it needs to build a velocity profile for each of the states that the vehicle can be in.Because we have only three states in our algorithm (Lane following, declaration stop, stop) we need profiles for only two states:

In our implementation for each trajectory we will have a maximum acceleration, start speed and desired speed.

- **Nominal Trajectory** (maintaining speed target for Lane Following state)

We first calculate accelerate distance, the distance we need to accelerate or decelerate to get our desired speed from start speed. Looping through spiral points and comparing cumulative distance of those points with accelerate distance with we get all the points of the spiral curve that falls under this distance. We will loop through these points and calulate the speed based on the speed of the last point starting from start speed as initial speed. If the desired speed is higher than the last point speed then we accelerate until it reaches the desired speed. Similary, if the speed is lower than the desired speed then we decelerate until it reaches desired speed. If the ego vehicle reaches desired speed the speed will remain constant (the desired speed) for all other suceeding points.

If the start speed is same to desired speed then all of the points will have the desired speed.

- **Decelerate Trajectory** (deceleration to stop for Deceleration to Stop state)

We first calculate decelerate distance using start speed and slow speed then brake distance using slow speed and 0 (speed at stop). We then check if the sum of brake distance and decelerate distance exceeds the length of the path or not, if it exceeds then we cannot perform a smooth deceleration and it requires a harder deceleration. So, we build the velocity profile accordingly using deceleration for the spiral curve in reverse to ensure we reach zero speed in the final point or index at the required time.

If the sum of brake distance brake distance and decelerate distance does not exceed the length of the path then smooth decelration is feasible. We find out the the index at which we need to start braking down to zero which is termed as brake index. We then go iteratively through the spiralpoints index and check in which index,the distance is bigger than the brake distance.The brake index will be the first index where the above condition is met.

Using the brake index we calculate the deceleration index, we go iteratively through the spiral and check in which index, the distance is bigger than the decelerrate distance. The deceleration index will be the first index where the above condition is met.

Now, based on these indexes we calculate velocity for each point of the curve.

Upto deceleration index the velocity will be decresing for each consecitive points starting from start speed as initial speed,i.e. we will be decelerating.

From decleration index to brake index the last reduced velocity we got from previous deceleration will be constant for all points lying in this range.

From brake index to stop index, we again start decelerating and hence reducing velocity for each consecutive point till it reaches zero.

For the last point or stop index we just add the the zero velocity and thus we create a smoooth decelerate trajectory.

Distance Calculation

Distance is calculated using one of the common rectilinear accelerated equations of motion to calculate the distance traveled while going from v_i (initial velocity) to v_f (final velocity) at a constant acceleration/deceleration "a"

$d = \left| \frac{v_f^2 - v_i^2}{2a} \right|$

Velocity/Speed Calculation

Final speed for a given acceleration/deceleration across "a", given distance "d" , with initial speed "v_i"

$v_f^2 = v_i^2 + 2ad$

$v_f = \sqrt{v_f^2}$

$v_f = 0 \quad \text{if} \quad v_f^2 < 0 \quad (\text{negative discriminant})$

In our implementation there is a maximum acceleration a_max. While calculating distance or velocity, we pass negative a_max if deceleration is required and we pass positive a_max if acceleration is required. To get speed of a point v_f we pass the speed of suceeding point as initial velocity v_i

Analysis

Below figures show some exemplary screenshots of the project's test case simulation using CARLA.

The trajectories in blue color show the potential tracks that have been evaluated. The red tracks are the ones which would lead to a collsion, and are avoided. The green track is the actual track that has been selected. The level of the trajectories in vertical direction shown in the screenshots is proportional to the planned velocity. When the ego vehicle decelerates the curves go down, for instance, or visce versa if the ego vehicle accelerates.

fig1 Fig 1: Ego vechicle starts up and has to accelerate thats why we can see planned trajectories very high in the vertical axis

fig2 Fig 2: Ego vehicle follows the lane without any obstacles within the lookahead range

fig3 Fig 3: : Ego vehicle avoids an obstacle (a parked car) on the right by executing a nudging maneouver to the left

fig4 Fig 4: : Ego vehicle avoids an obstacle (a parked car) on the left by executing a nudging maneouver to the right

fig5 Fig 5: Ego vehicle slows down towards the stop line and finally stop at the stop line. Therefore, the planned trajectories go down in vertical axis to visualize the deceleration(reduction in the velocity)

In our simu;ation analysis as required the ego vehicle avoids all the three obstacles, stop on each junctions and continue moving seamlessly after the second junction.