

Motion Planning and Decision Making for Autonomous Vehicles

Project Objective:

In this project, we will implement two main components of a traditional hierarchical planner: an action planner and a motion planner. They work together to:


The vehicle must perform a "lane change" maneuver without colliding with any moving or stopped vehicle.

Determine the path to take after stopping at every intersection. We will construct a set of student assignment files that perform these functions to understand the core principles.

File structure:

<https://github.com/udacity/nd013-c5-planning-starter>

<https://github.com/udacity/nd013-c5-planning-starter/tree/master/project>

🔗 master ▾	🔗 5 Branches 🔖 0 Tags	🔍 Go to file t	Add file ▾	<> Code ▾
 islamalsawy Merge pull request #8 from nehbehl/patch-3 c273aa8 · 8 months ago ⌚ 67 Commits				
📁 .github/workflows	Add Github Action	4 years ago		
📁 project	Update install-ubuntu.sh	8 months ago		
📄 .DS_Store	update	5 years ago		
📄 .gitignore	update the readme	2 years ago		
📄 CODEOWNERS	Create CODEOWNERS	4 years ago		
📄 LICENSE.md	update	5 years ago		
📄 README.md	clarify a step	2 years ago		

https://github.com/udacity/nd013-c5-planning-starter/tree/master/project/starter_files

project/starter_files

.clang-format	—
CMakeLists.txt	—
README.md	—
behavior_planner_FSM.cpp	Student's Update Item
behavior_planner_FSM.h	—
cost_functions.cpp	Student's Update Item
cost_functions.h	—
cubic_spiral.cpp	—
cubic_spiral.h	—
integral.cpp	—
integral.h	—
json.hpp	—
main.cpp	—
matplotlibcpp.h	—
motion_planner.cpp	Student's Update Item
motion_planner.h	—
numpy_flags.py	—
planning_params.h	Student's Update Item
plot_utils.cpp	—
plot_utils.h	—
spiral_base.cpp	—
spiral_base.h	—
spiral_equations.cpp	—
spiral_equations.h	—
structs.h	—
utils.cpp	—
utils.h	—
vehicle_dynamic_model.cpp	—
vehicle_dynamic_model.h	—
velocity_profile_generator.cpp	Student's Update Item
velocity_profile_generator.h	—

Hierarchy of File Structure :

Blue text : Files that students task

`planning_params.h` (Parameter definition)

- |— `main.cpp` (Main executable file)
 - |— `behavior_planner_FSM.cpp` (Managing the vehicle's behavioral state)
 - |— `motion_planner.cpp` (Route planning and generation)
- |— `cost_functions.cpp` (Calculating path costs) <-- Not directly linked to `main.cpp`
- |— `velocity_profile_generator.cpp` (Create a speed profile) <-- Not directly linked to `main.cpp`
- |— `utils.cpp` (Provides mathematical calculations and auxiliary functions)
- |— `vehicle_dynamic_model.cpp` (Physical model of the vehicle)
- |— `spiral_equations.cpp` (Spiral trajectory generation algorithm)
- |— `spiral_base.cpp` (Spiral trajectory based class)
- |— `cubic_spiral.cpp` (Calculations related to the 3rd helix trajectory)
- |— `integral.cpp` (Mathematical integral operation)
- |— `plot_utils.cpp` (Visualization utility for debugging)

`planning_params.h` file is a mother file that provides values to other files.

How to run::

1. Connect to Udacity's Ubuntu environment.
2. Execute the following command in terminal window 1:
3. Perform git clone. The git address below is the author's address. The official starter location of Udacity is (<https://github.com/udacity/nd013-c5-planning-starter.git>)

Git clone <https://github.com/juwonlim/Motion-Planning-and-Decision-Making-for-Autonomous-Vehicles.git>

Change directory using cd.

Cd Motion-Planning-and-Decision-Making-for-Autonomous-Vehicles/tree/main/project

Use ls to check the files in the directory.

```
ubuntu@ip-172-31-2-238:~/Motion-Planning-and-Decision-Making-for-Autonomous-Vehicles/project$ ls
cserver_dir  install-ubuntu.sh  manual_control.py  run_carla.sh  run_main.sh  simulatorAPI.py  starter_files
```

4. Open terminal window 2

In that window, execute the following commands:

chmod +x run_carla.sh (grant permissions to file)

./run_carla.sh (execute)

5. Open terminal window 3

chmod +x install-ubuntu.sh (grant permissions to file)

./install-ubuntu.sh (Run this command to install the required files.)

6. Open terminal window 4

Go to the Starter_files directory

Cd .. (If you are currently in the Project directory, move to the directory one level higher.)

or `cd starter_files`

`rm -rf rpclib` (excute)

`git clone` <https://github.com/rpclib/rpclib.git> (excute)

Now enter the commands below in order.

`Cmake .`

`Make`

If the make command completed without errors, now enter the command below.

(The directory is still in the project)

`Chmod +x run_main.sh` (grant permissions to file)

`./run_main.sh` (Run the simulator)

Simulation run results:

The image below shows the results using the CARLA simulator. The blue lines are potential trajectories that the car could choose. The green trajectories are the ones that were actually chosen. The red trajectories are the ones that could lead to a collision. Obviously, they would not be chosen.



In the picture below, you can see a red trajectory being created on the collision path with the car in front. EGO CAR does not follow the red trajectory.



Avoid the car in front and choose the left lane. See the picture below.



You are driving in the left lane to avoid the vehicle in front of you, and then you need to change lanes again because there is another vehicle ahead of you.



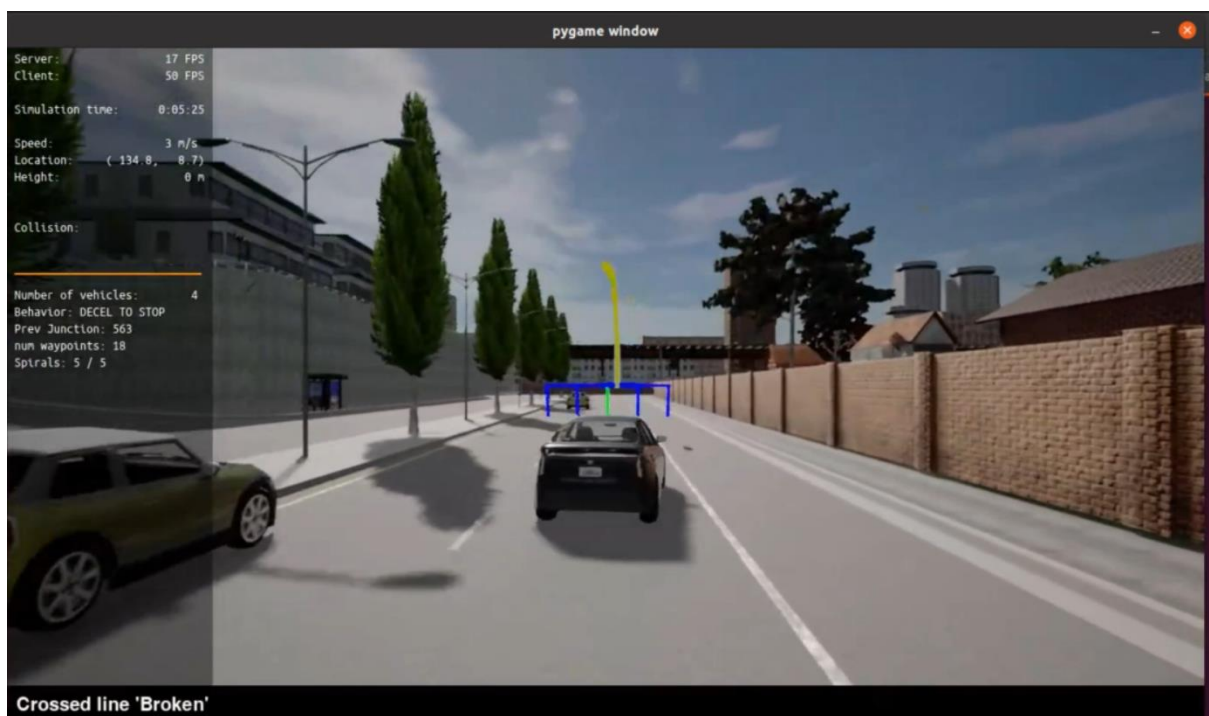
Once you meet the car in front (the parked car) again, the route selection is made. The route to exit to the right lane is selected in green.



The image below shows choosing the right lane again to avoid the vehicle in front (the parked car).



Now we're almost in the right lane.



The image below shows that the more the EGO CAR is aligned to the center of the lane, the fewer blue selectable trajectories there are.



Now the picture below shows the EGO CAR entering the intersection. You can see that it slows down before entering the intersection and the trajectories that can be selected appear in front of it. As it slows down, you can see that the trajectory changes into a fork shape closer to the ground.



#Summary of things learned from the project

I learned that the `P_NUM_PATHS` and `P_NUM_POINTS_IN_SPIRAL` variables defined in the `planning_params.h` file are very important. I learned in advance before running the project that if `P_NUM_PATHS` is too small, the number of PATHs generated will be too small, and if the `P_NUM_POINTS_IN_SPIRAL` variable is small, the path generation will not be smooth. In my case, I set them to 5 and 20.

This project does not apply computer vision or deep learning to detect vehicles, but receives the location information of surrounding vehicles provided by the CARLA simulator, and in the `behavior_planner_FSM.cpp` file,

I understood that vehicle information is received from the simulator through a variable called `Ego_state`.

Understanding Receiving Vehicle Information from the Simulator:

```
auto waypoint_0 = map->GetWaypoint(ego_state.location);
```

`ego_state.location`: Indicates the current location of the autonomous vehicle

`map->GetWaypoint(ego_state.location)`: Gets the closest waypoint from the current location. In other words, this function is responsible for getting the road map and vehicle location information from the simulator.

It also adjusts the `look_ahead_distance` based on the vehicle speed (Velocity).

```
auto look_ahead_distance = get_look_ahead_distance(ego_state);
```

Inside `get_look_ahead_distance()`, the vehicle speed and acceleration are used to adjust the forward search distance. If it is fast, the search distance is long, and if it is slow, the search distance is short. That is, in `behavior_planner_FSM.cpp`, the simulator data is received and the current location, speed, and road conditions of the vehicle are recognized.

Understanding how to detect a vehicle(object) ahead or a stopped vehicle:

There is an obstacle detection code inside the cost_function.cpp file.

spiral: Multiple candidate paths for the vehicle to move

obstacles: Obstacle information received from the simulator (including the vehicle in front)

```
double collision_circles_cost_spiral(const std::vector<PathPoint>& spiral,  
  
const std::vector<State>& obstacles)
```

Below is the code to perform collision detection by placing several circles around the vehicle.

```
auto circle_center_x = cur_x + CIRCLE_OFFSETS[c] * std::cos(cur_yaw);
```

```
auto circle_center_y = cur_y + CIRCLE_OFFSETS[c] * std::sin(cur_yaw);
```

I have verified that the code below calculates the obstacle location and distance.

```
double dist = std::sqrt(std::pow(circle_center_x - actor_center_x, 2) +  
  
std::pow(circle_center_y - actor_center_y, 2));
```

If the distance is shorter than the sum of the radii of the circles, a collision occurs! High cost is returned

That is, a circle is placed around the vehicle, and if the car in front enters the circle, it is judged as a collision and a high cost is returned.

```
collision = (dist < (CIRCLE_RADII[c] + CIRCLE_RADII[c2]));
```

```
return (collision) ? COLLISION : 0.0;
```

Understanding the process of deciding when to change lanes, accelerate/decelerate:

This functionality is handled in motion_planner.cpp.

Core code:

```
double cost = calculate_cost(spirals[i], obstacles, goal_state);
```

In motion_planner.cpp, we generate several spiral paths (spirals) and call calculate_cost() in cost_functions.cpp to determine whether each path has a high probability of colliding with an obstacle (the car in front), discard the paths with a high probability of collision, and select a safe path.

If a lane change is necessary, aim for the adjacent lane.

```
goal_offset.location.x += offset * std::cos(yaw);
```

```
goal_offset.location.y += offset * std::sin(yaw);
```

It also determines acceleration and deceleration.

```
goal.velocity.x = _speed_limit * std::cos(goal.rotation.yaw);
```

```
goal.velocity.y = _speed_limit * std::sin(goal.rotation.yaw);
```

That is, we learned that motion_planner.cpp performs the role of avoiding collision by performing lane change or deceleration when detecting the vehicle in front.

In fact, the three files above perform surrounding vehicle information reception, lane change, acceleration/deceleration, etc.

Understanding the Velocity_Profile_generator.cpp file:

This file works in conjunction with the three files mentioned above.

This file also directly affects the deceleration, acceleration, and trajectory generation of the vehicle. However, it does not directly detect the vehicle's surroundings or determine whether to change lanes.

Main functions:

1. Generate the vehicle's acceleration/deceleration speed profile (trajectory)
2. When the stop line or the vehicle in front is decelerating, it decelerates smoothly to avoid collisions
3. Adjusts the speed curve so that the vehicle maintains the target speed stably

In the case where the vehicle needs to stop (Decel_To_Stop state), the code below is used.

```
trajectory = decelerate_trajectory(spiral, start_speed);
```

If you are following the vehicle in front (Follow_vehicle), use the code below.

Lead car state : Adjust the deceleration trajectory by receiving status information from the preceding vehicle (front vehicle)

The vehicle performs the role of reducing speed smoothly to match the speed of the vehicle in front.

```
trajectory = follow_trajectory(spiral, start_speed, desired_speed, lead_car_state);
```

In the case of the normal driving state (Nominal_travel), it works with the code below. It generates a trajectory that moves while maintaining a constant speed and drives while maintaining the speed based on the desired speed value.

```
trajectory = nominal_trajectory(spiral, start_speed, desired_speed);
```

Understanding Planning_Params.h:

The main function of this file is to define constant values for vehicle motion, such as path search distance, speed limit, stopping criteria, and reaction time.

It provides configuration values for behavior_planner_FSM.cpp, motion_planner.cpp, velocity_profile_generator.cpp, and cost_functions.cpp.

It also sets physical constraints to ensure that the vehicle selects an appropriate trajectory.

The parameters related to vehicle speed and acceleration settings (related to velocity_profile_generator.cpp) are as follows:

Maximum acceleration (P_MAX_ACCEL): Limits the vehicle from accelerating too quickly

Speed limit (P_SPEED_LIMIT): Sets the maximum speed the vehicle can drive

Slow speed standard (P_SLOW_SPEED): Drives at low speed when the vehicle is waiting at an intersection or at a signal

```
#define P_MAX_ACCEL 1.5 // Maximum acceleration (m/s^2)
```

```
#define P_SLOW_SPEED 1.0 // Low speed standard speed (m/s)
```

```
#define P_SPEED_LIMIT 3.0 // Speed limit (m/s)
```

Below is the code that provides the reference value when the vehicle stops (related to behavior_planner_FSM.CPP).

P_STOP_THRESHOLD_SPEED: Vehicle is considered stopped when it is under 0.02m/s

P_REQ_STOPPED_TIME: Vehicle must remain stopped for at least 1 second before it can be driven again

P_STOP_THRESHOLD_DISTANCE: Determines how close the vehicle must be to the stop line to stop

```
#define P_STOP_THRESHOLD_SPEED 0.02 // Speed at which the vehicle is considered stopped
```

```
#define P_REQ_STOPPED_TIME 1.0 // Time to remain stopped (s)
```



```
#define P_STOP_THRESHOLD_DISTANCE P_LOOKAHEAD_MIN / P_NUM_POINTS_IN_SPIRAL * 2
```

The collision detection related settings for the vehicle (related to cost_function.cpp) are the code below.

A circle is placed around the vehicle for collision detection to detect obstacles, and cost_functions.cpp uses it to calculate whether the vehicle is likely to collide with an obstacle.

```
constexpr std::array<float, 3> CIRCLE_OFFSETS = {-1.0, 1.0, 3.0}; // Position around the  
                                vehicle (m)
```

```
constexpr std::array<float, 3> CIRCLE_RADII = {1.5, 1.5, 1.5}; // Radius of a circle (m)
```

Ultimately, this file plays an important role in defining the vehicle's speed, acceleration, stopping criteria, and collision avoidance-related settings, but it does not directly perform functions such as lane change, front vehicle detection, etc. It took me a while to understand that it was possible to detect and avoid vehicles by receiving information from the simulator. After submitting the project, I plan to study the remaining files (files that are not student assignments but play a key role). Thank you.

Here is the video link

<https://youtu.be/RA7em96rDnY>