Motion Planning and Decision Making for Autonomous Vehicles

프로젝트 목적 :

이 프로젝트에서는 기존 계층적 플래너의 두 가지 주요 구성 요소인 행동 플래너와 동작 플래너를 구현합니다. 둘 다 함께 작동하여 다음을 수행할 수 있습니다.

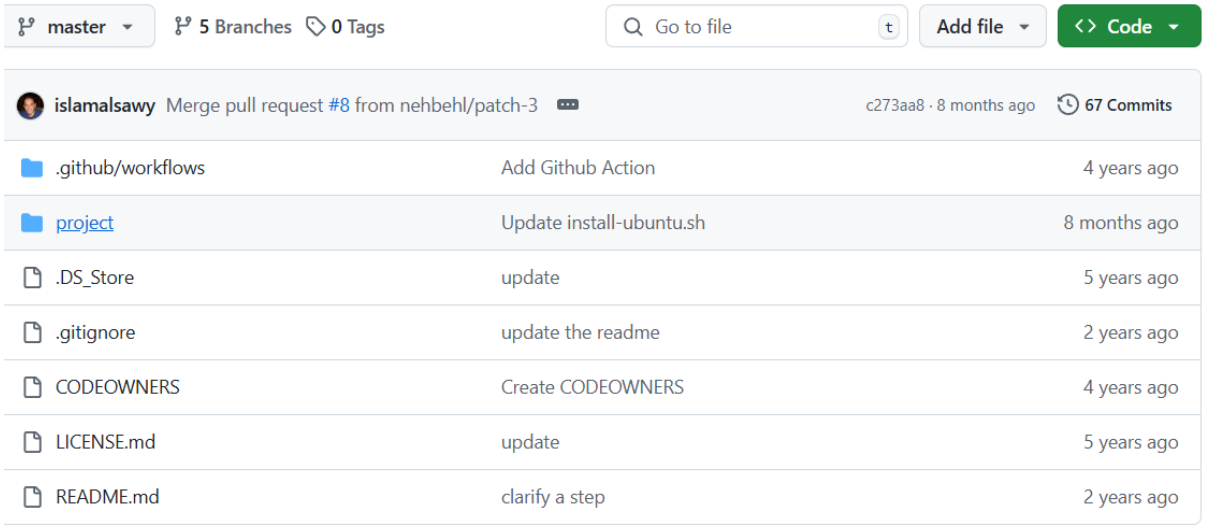
차량은 "차선 변경" 기동을 실행하여 주행중이거나 정지한 차량과 충돌하지 않아야 합니다.

모든 교차로에서 정지 후 경로를 결정하여 주행합니다. 이러한 기능을 수행하는 student과제 파일들을 구성하여 핵심원리를 이해합니다..

파일 구성 :

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

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



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 (파일의 계층구조):

Blue text : student가 직접 수정하는 파일

planning_params.h (매개변수 정의)

├── main.cpp (메인 실행 파일)

├── behavior_planner_FSM.cpp (차량의 행동 상태 관리)

├── motion_planner.cpp (경로 계획 및 생성)

├── cost_functions.cpp (경로 비용 계산) <-- main.cpp와 직접 연결되지 않음

├── velocity_profile_generator.cpp (속도 프로파일 생성) <-- main.cpp와 직접 연결되지 않음

├── utils.cpp (수학적 계산, 보조 함수 제공)

├── vehicle_dynamic_model.cpp (차량의 물리 모델)

├── spiral_equations.cpp (나선형 궤적 생성 알고리즘)

├── spiral_base.cpp (나선형 궤적 기반 클래스)

├── cubic_spiral.cpp (3차 나선 궤적 관련 계산)

├── integral.cpp (수학적 적분 연산)

├── plot_utils.cpp (디버깅용 시각화 유틸리티)

planning_params.h 파일은 다른 파일들에게 값을 주는 mother파일임.

실행 방법:

1. Udacity의 Ubuntu환경에 접속한다

2. Ubuntu에서 Terminal을 실행한다

3. 터미널 창 1에서 아래의 명령을 수행한다

   git clone을 수행한다. 아래의 git주소는 작성자 본인의 주소이다. Udacity의 공식 starter 위치는 (https://github.com/udacity/nd013-c5-planning-starter.git)

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

   cd해서 디렉토리를 변경한다.

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

   ls하여 디렉토리 내의 파일들을 확인한다.

```
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. 터미널 창2를 Open한다

   그 창에서 아래의 명령어를 수행한다

   chmod +x run_carla.sh (파일에 권한 부여)

   ./run_carla.sh (파일 실행)

5. 터미널 창3을 Open 한다

   chmod +x install-ubuntu.sh (파일에 실행 권한을 부여한다)

   ./install-ubuntu.sh (이 명령어를 실행하여 필요한 파일들을 설치한다)

6. 터미널 창4를 Open한다

   Starer_files디렉토리로 이동한다

   Cd .. (현재가 Project 디레토리라면 한 단계 상위 디렉토리로 이동)

또는 **cd starter_files**

rm -rf rpclib 실행

git clone https://github.com/rpclib/rpclib.git 실행

이제 아래의 명령어를 순서대로 입력한다

Cmake .

Make

make명령어가 error없이 완료되었다면 이제 아래의 명령어를 입력한다

(디렉토리는 지금도 Project에 있다)

Chmod +x run_main.sh (파일에 실행권한 부여)

./run_main.sh (이제 시뮬레이터를 실행한다)

시뮬레이션 실행 결과 :

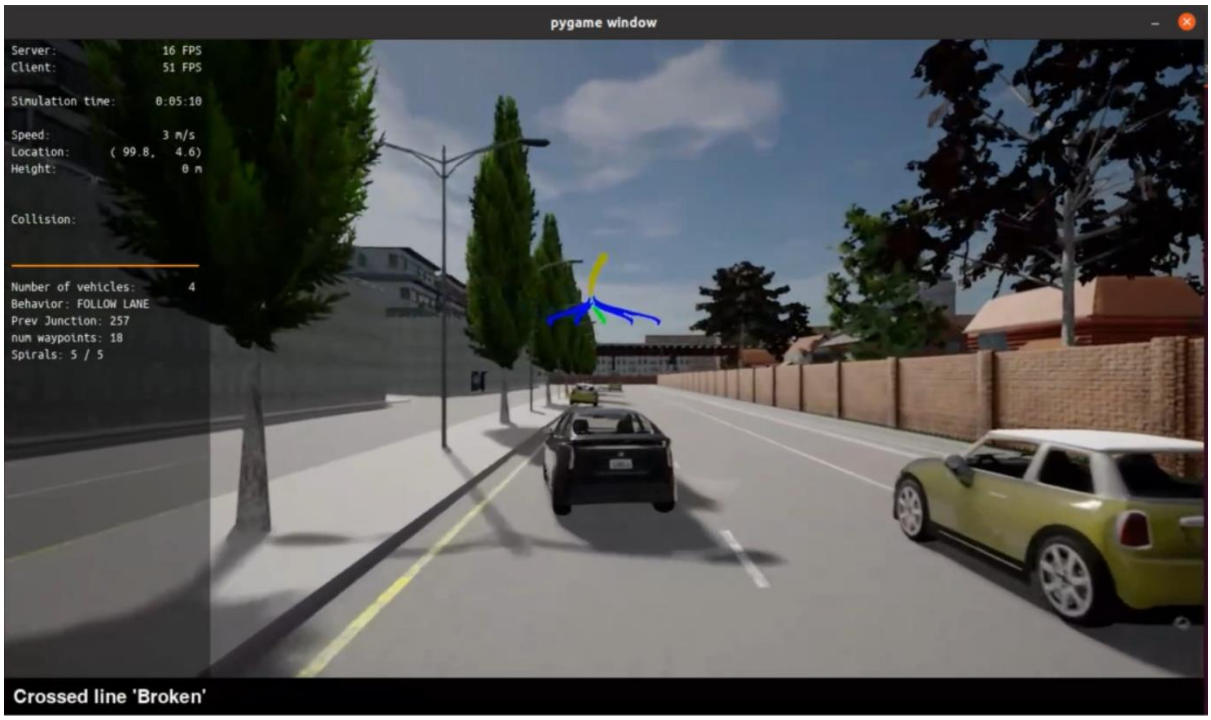
아래의 그림은 CARLA 시뮬레이터를 사용한 결과입니다. 파란색 선은 자동차가 선택할 수 있는 잠재적 궤적들입니다. 녹색궤적이 실제로 선택된 궤적입니다. 빨간색 궤적은 충돌로 이어질 수 있는 궤적입니다. 당연히 선택되지 않겠죠.

아래의 그림에서 앞차와의 충돌경로에 빨간색 궤적이 생성되는 것을 볼 수 있습니다.

EGO CAR는 빨간색 궤적을 따라가지 않습니다.

앞차를 회피하여 좌측차선으로 경로를 선택합니다. 아래의 그림을 보세요.

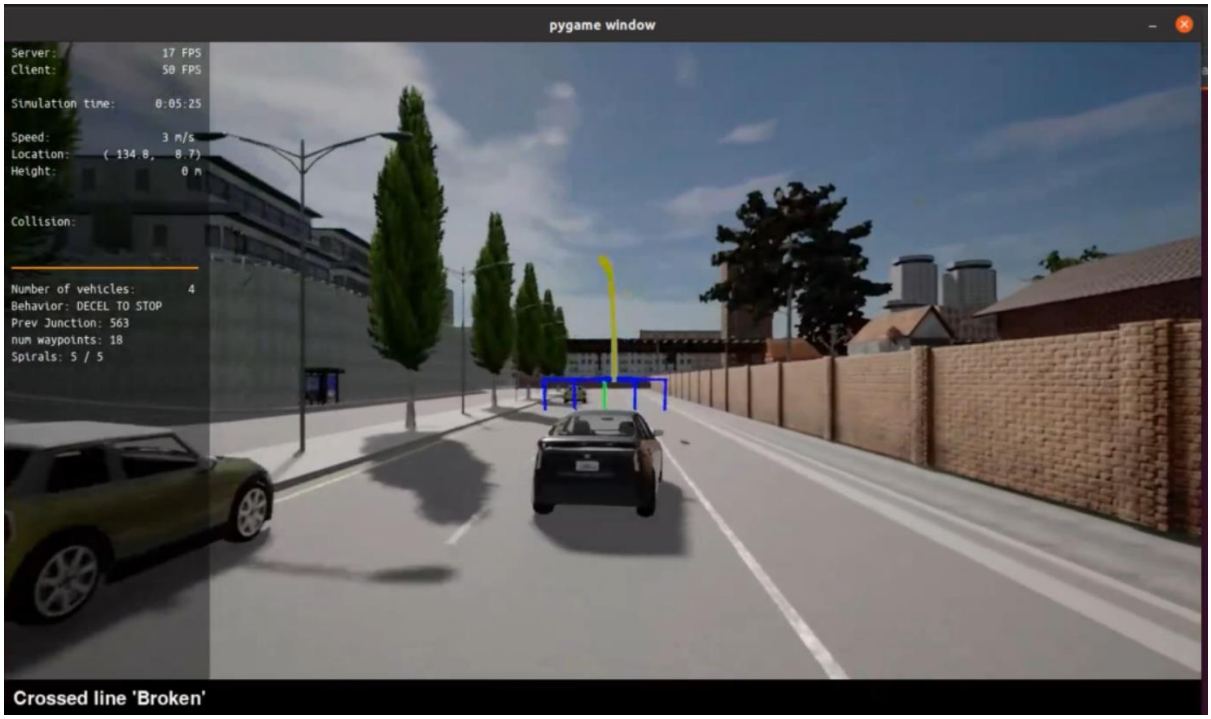
앞에 있던 차량을 회피하여 좌측 차선으로 가고 있고 저 앞에 다른 차량이 가고 있기에 다시 한 번 차선을 변경할 필요가 있습니다.



다시 앞차(주차된 차량)를 만나게 되어서 경로 선택이 이루어집니다. 우측차선으로 빠져나가는 경로가 초록색으로 선택됩니다.

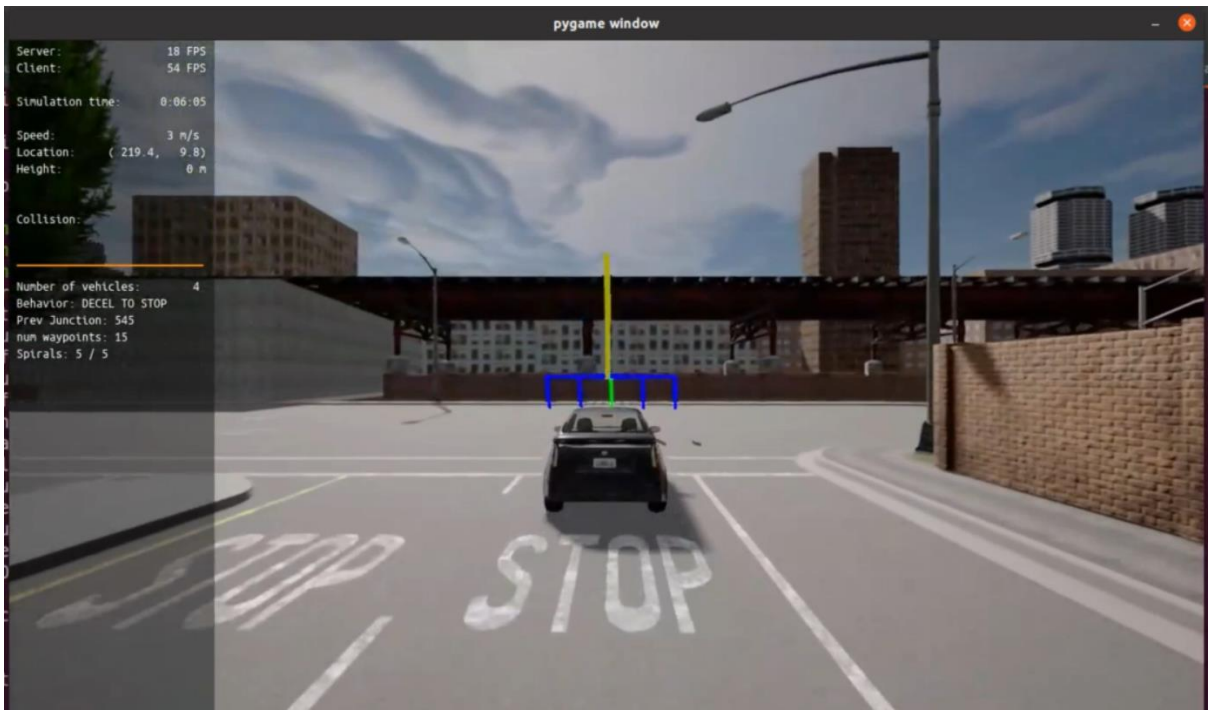
아래의 그림은 앞차(주차된 자동차)를 다시 회피하여 우측차선을 선택하는 것을 보여줍니다.

이제 우측차선에 거의 자리를 잡았습니다.



아래그림은 EGO CAR가 차선의 중앙에 정렬할수록 파란색의 선택할 수 있는 궤적들은 줄어들게 되는 것을 보여줍니다

이제 아래의 그림은 EGO CAR가 교차로에 진입했음을 보여줍니다. 교차로 진입전에 감속하는 것을 볼 수 있으며 앞에 선택 가능한 궤적이 나타납니다. 감속할수록 궤적이 땅에 가까운 포크모양으로 변하는 것을 볼 수 있습니다. (감속을 시각화함!)



#프로젝트에서 배운 것들 정리

planning_params.h 파일에서 정의되어 있는 P_NUM_PATHS와 P_NUM_POINTS_IN_SPIRAL 변수가 아주 중요함을 배웠습니다. P_NUM_PATHS가 너무 작은 값일 경우 생성되는 PATH가 숫자가 너무 적어지며 P_NUM_POINTS_IN_SPIRAL 변수가 작은 값일 경우 경로 생성이 부드럽게 되지 못한다는 것을 프로젝트 실행전에 미리 알게 되었습니다. 저의 경우는 5와 20으로 설정했습니다.

이번 프로젝트는 컴퓨터 비전이나 딥러닝을 응용해서 차량을 감지하는 것은 아니고 CARLA시뮬레이터에서 제공하는 주변 차량의 위치 정보 값을 받아서 behavior_planner_FSM.cpp파일에서는

Ego_state라는 변수를 통해서 차량정보를 시뮬레이터로부터 수신하는 것으로 파악했습니다.

시뮬레이터로부터의 차량 정보 수신에 대한 이해 :

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

ego_state.location: 자율주행차의 현재 위치를 나타냄

map->GetWaypoint(ego_state.location): 현재 위치에서 가장 가까운 웨이포인트(Waypoint) 를 가져옴 즉, 이 함수가 시뮬레이터에서 도로 지도와 차량의 위치 정보를 가져오는 역할을 수행합니다.

또한 차량 속도(Velocity)기반으로 탐색거리(look_ahead_distance)를 조절합니다.

auto look_ahead_distance = get_look_ahead_distance(ego_state);

get_look_ahead_distance() 내부에서 차량 속도와 가속도를 이용해 전방 탐색 거리 조절, 빠르면 탐색 거리를 길게, 느리면 짧게 조정 합니다.

즉, behavior_planner_FSM.cpp 에서 시뮬레이터 데이터를 받아 차량의 현재 위치, 속도, 도로 상황을 인식합니다.

앞차 또는 정차된 차량 (장매물)을 감지하는 방법에 대한 이해:

Cost_function.cpp파일 내부에 장애물 감지 코드가 있습니다.

spiral: 차량이 이동할 수 있는 여러 개의 후보 경로

obstacles: 시뮬레이터에서 받은 장애물 정보(앞차 포함)

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

차량 주변에 여러 개의 원(circle)을 배치하여 충돌 검사를 수행하는 코드는 하기에 있습니다

```
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);
```

장애물위치와 거리를 계산하는 것은 아래의 코드라는 것을 확인했습니다

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

원의 반지름 합보다 거리가 짧으면 충돌이 발생! → 높은 cost 반환

즉, 차량 주변에 원을 배치하고 앞차가 원 안에 들어오면 충돌로 판단하여 높은 비용을 반환

```
collision = (dist < (CIRCLE_RADII[c] + CIRCLE_RADII[c2]));
return (collision) ? COLLISION : 0.0;
```

차선 변경, 가속/감속을 결정하는 과정에 대한 이해:

이런 기능은 motion_planner.cpp에서 담당합니다.

핵심 코드 :

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

motion_planner.cpp에서 여러 개의 나선형 경로(spirals)를 생성하고

cost_functions.cpp의 **calculate_cost()**를 호출하여 각 경로가 장애물(앞차)와 충돌 가능성이 있는지 판단하고 충돌 가능성이 높은 경로는 폐기하고, 안전한 경로 선택 합니다.

차선 변경이 필요하면 옆차선으로 목표를 조정합니다.

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

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

또한 가속 감속을 결정합니다.

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

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

즉, motion_planner.cpp는 앞차를 감지했을 때 차선 변경 또는 감속을 수행해 충돌을 피하는 역할을 수행한다는 것을 배웠습니다.

사실상, 위의 3개의 파일로 주변차량정보수신, 차선변경,가속감속등을 수행합니다.

Velocity_Profile_generator.cpp파일에 대한 이해 :

이 파일은 위에 언급된 3개의 파일들과 연동하여 도움을 줍니다.

이 파일도 차량의 감속, 가속 및 궤적 생성에 직접 영향을 미칩니다. 하지만 차량 주변을 직접 감지하거나 차선 변경여부를 결정하는 역할은 아닙니다.

주요 기능 :

1. 차량의 가속/감속 **속도 프로파일(trajectory)을 생성**
2. 정지선이나 앞차가 감속할 경우, **부드럽게 감속하여 충돌을 방지**
3. 차량이 목표 속도를 안정적으로 유지하도록 속도 곡선을 조절

차량이 정지해야 하는 경우(Decel_To_Stop 상태)의 경우는 아래의 코드를 이용합니다

trajectory = decelerate_trajectory(spiral, start_speed);

앞차를 따라가는 경우 (Follow_vehicle)는 아래의 코드를 이용합니다

Lead car state : 선행 차량(앞차)의 상태 정보를 받아 감속 궤적을 조정

차량이 앞차 속도에 맞춰 부드럽게 속도를 줄이는 역할 수행

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

일반 주행상태 (Nominal_travel)의 경우는 아래의 코드에 의해 작동합니다.

속도를 일정하게 유지하면서 이동하는 궤적을 생성하며 desired speed값을 기반으로 속도를 유지하며 주행하도록 합니다.

trajectory = nominal_trajectory(spiral, start_speed, desired_speed);

Planning_Params.h에 대한 이해 :

이 파일의 주요기능은 경로 탐색 거리, 속도 제한, 정지 기준, 반응 시간 등 차량 움직임의 상수 값을 정의합니다.

behavior_planner_FSM.cpp, motion_planner.cpp, velocity_profile_generator.cpp, cost_functions.cpp 등의 설정 값을 제공합니다.

또한 차량이 적절한 궤적을 선택하도록 물리적인 제한 조건을 설정합니다.

챠량속도 및 가속설정관련(velocity_profile_generator.cpp와 연관)은 아래의 파라매터들 입니다.

최대 가속도 (P_MAX_ACCEL): 차량이 가속할 때 너무 급격하게 하지 않도록 제한

속도 제한 (P_SPEED_LIMIT): 차량이 주행 가능한 최고 속도 설정

저속 주행 기준 (P_SLOW_SPEED): 차량이 교차로 또는 신호 대기 중일 때 저속 주행

#define P_MAX_ACCEL 1.5 // 최대 가속도 (m/s^2)

#define P_SLOW_SPEED 1.0 // 저속 주행 기준 속도 (m/s)

#define P_SPEED_LIMIT 3.0 // 속도 제한 (m/s)

차량이 정지 할 때 기준값을 제공 (behavior_planner_FSM.CPP와 연관)하는 것은 아래의 코드입니다.

P_STOP_THRESHOLD_SPEED: 차량이 0.02m/s 이하일 때 정지 상태로 간주

P_REQ_STOPPED_TIME: 차량이 정지 상태에서 최소 1초 이상 머물러야 다시 주행 가능

P_STOP_THRESHOLD_DISTANCE: 차량이 정지선에서 얼마나 가까워야 멈출지 결정

#define P_STOP_THRESHOLD_SPEED 0.02 // 차량이 정지 상태로 간주할 속도

#define P_REQ_STOPPED_TIME 1.0 // 정지 유지 시간 (s)

#define P_STOP_THRESHOLD_DISTANCE P_LOOKAHEAD_MIN / P_NUM_POINTS_IN_SPIRAL * 2

챠량의 충돌 감지 관련 설정(cost_function.cpp와 연관)은 아래의 코드 입니다.

차량 주변에 충돌 감지를 위한 원(Circle)을 배치하여 장애물 감지하며, cost_functions.cpp에서 차량이 장애물과 충돌할 가능성이 있는지 계산하는 데 사용합니다.

constexpr std::array<float, 3> CIRCLE_OFFSETS = {-1.0, 1.0, 3.0}; // 차량 주변 원 위치 (m)

constexpr std::array<float, 3> CIRCLE_RADII = {1.5, 1.5, 1.5}; // 원 반지름 (m)

결국 이 파일은 차량의 속도, 가속도, 정지 기준, 충돌 회피 관련 설정을 정의하는 중요한 역할 수행하지만 직접적으로 차량의 차선 변경, 앞차 감지 등의 기능을 수행하지는 않습니다.

시뮬레이터에서 정보를 수신하여 이렇게 차량을 감지하고 회피하는 기동이 가능하다는 것이 생소해서 이해하는데 시간이 걸렸습니다. 프로젝트 제출 후에도 나머지 파일들 (student과제는 아니지만 핵심적인 역할을 하는 파일들)에 대해서도 공부해 볼 계획입니다. 감사합니다.

하기에는 영상 링크 입니다

https://youtu.be/RA7em96rDnY