Control and Trajectory Tracking for Autonomous Vehicle

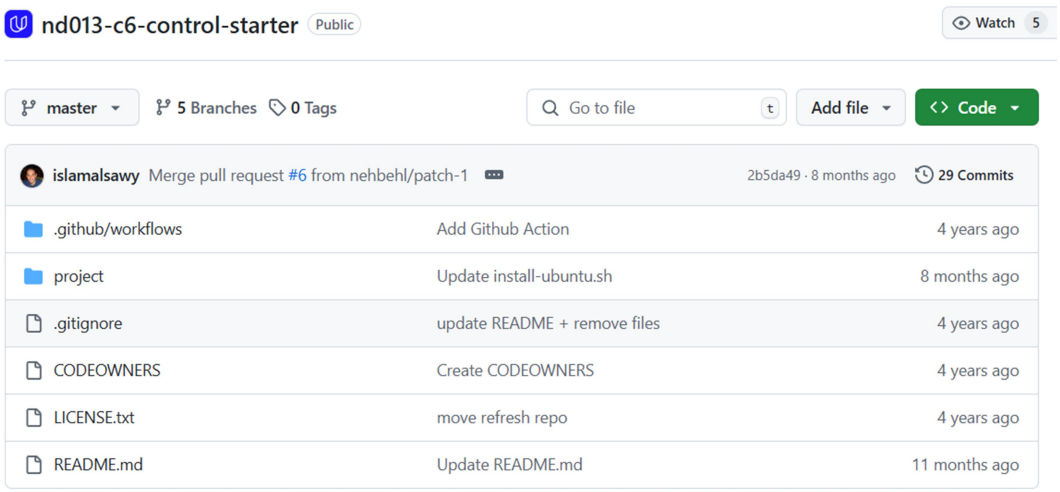
Proportional-Integral-Derivative (PID)

프로젝트 목적 :

이 프로젝트에서는 이 과정에서 습득한 기술을 적용하여 차량 궤적 추적을 수행하는 비례-적분-미분(PID) 컨트롤러를 설계합니다. 위치 배열로서의 궤적과 시뮬레이션 환경이 주어지면 PID 컨트롤러를 설계하고 코딩하여 업계에서 사용되는 CARLA 시뮬레이터에서 효율성을 테스트합니다.

파일 구성 :

Starter: https://github.com/udacity/nd013-c6-control-starter



project/pid_controller Directory

.clang-format	
CMakeLists.txt	
Makefile	
answers.txt	
behavior_planner_FSM.cpp	
behavior_planner_FSM.h	
cmake_install.cmake	
cost_functions.cpp	
cost_functions.h	
cubic_spiral.cpp	
cubic_spiral.h	
integral.cpp	
integral.h	
json.hpp	
main.cpp	Student's Update Item
matplotlibcpp.h	
motion_planner.cpp	
motion_planner.h	
numpy_flags.py	
pid_controller.cpp	Student's Update Item
pid_controller.h	Student's Update Item
planning_params.h	
plot_pid.plt	
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	
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)
  - **pid_controller.cpp (PID 제어, 조향 및 가속 조절)**
    - **pid_controller.h (PID 제어 헤더)**
  - velocity_profile_generator.cpp (속도 프로파일 생성)
- cost_functions.cpp (경로 비용 계산)
- utils.cpp (수학적 계산, 보조 함수 제공)
- structs.h (구조체 정의, 차량 및 경로 관련 데이터)
- 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)
- json.hpp (JSON 데이터 처리)

main.cpp가 프로젝트의 중심이며, 여러 파일을 호출하여 주행 계획을 수행합니다.

pid_controller.cpp는 main.cpp에서 호출되며, 조향과 가속 제어를 담당합니다.

motion_planner.cpp는 behavior_planner_FSM.cpp와 함께 주행 경로를 생성합니다.

utils.cpp, spiral_equations.cpp 등은 수학적 계산 및 보조 기능을 제공합니다.

프로젝트의 Core 요소:

아래의 요소들은 이 프로젝트의 핵심요소입니다

PID 파라미터(Kp, Ki, Kd)는 **pid_controller.h** 파일에서 정의되고, **main.cpp**에서 **pid_steer.Init()** 및 **pid_throttle.Init()**을 통해 설정됩니다.

□**PID 파라미터가 미치는 영향 :**

Kp (Proportional Gain, 비례 계수):

1.오차(CTE)가 크면 더 큰 조향/가속도를 적용하여 빠르게 수정

2.너무 크면 시스템이 과도하게 반응(진동 발생)

3.너무 작으면 반응이 느려져 경로 이탈 가능

Ki (Integral Gain, 적분 계수):

1.과거 오차를 누적하여 보정

2.도로 경사 등 지속적인 편향(바이어스)을 보정하는 역할

3.너무 크면 **누적 오차가 과도하게 증가**하여 불안정

Kd (Derivative Gain, 미분 계수):

1.급격한 변화(진동)를 억제하는 역할

2.오차 변화율(속도)에 비례한 감속을 적용해 부드러운 조향/가속을 유지

3.너무 크면 과하게 감속해 시스템이 둔감해질 수 있음

PID 파라미터 조정하는 파일들

1.main.cpp → pid_steer.Init(Kp, Ki, Kd, MAX, MIN); 설정

2.pid_controller.cpp → TotalError() 함수에서 PID 공식 적용

3.pid_controller.h → PID 계수(Kp, Ki, Kd) 변수 선언

PID초기값이 설정되는 과정:

1. main.cpp에서 Init() 호출

```
pid_steer.Init(KP_STEER, KI_STEER, KD_STEER, MAX_STEER, MIN_STEER);

pid_throttle.Init(KP_THROTTLE, KI_THROTTLE, KD_THROTTLE, MAX_THROTTLE, MIN_THROTTLE);
```

2. main.cpp에서 상수정의:

```
static const double KP_STEER = 0.3;

static const double KI_STEER = 0.001;

static const double KD_STEER = 0.3;

static const double MAX_STEER = 1.2;

static const double MIN_STEER = -1.2;

static const double KP_THROTTLE = 0.2;

static const double KI_THROTTLE = 0.0009;

static const double KD_THROTTLE = 0.1;

static const double MAX_THROTTLE = 1.0;

static const double MIN_THROTTLE = -1.0;
```

3. pid_controller.cpp에서 Init() 함수로 값 저장:

```
void PID::Init(double Kp, double Ki, double Kd, double output_lim_max, double output_lim_min) {

    this->Kp_ = Kp;

    this->Ki_ = Ki;

    this->Kd_ = Kd;

    this->output_lim_max_ = output_lim_max;

    this->output_lim_min_ = output_lim_min;

}
```

정리하자면,

즉, main.cpp에서 Init(Kp, Ki, Kd, max, min)을 호출하면, pid_controller.cpp의 Init() 함수가 실행되고

→ Kp_ = Kp;, Ki_ = Ki;, Kd_ = Kd; 이런 식으로 값을 할당합니다.

그러면 pid_controller.h에 선언된 Kp_, Ki_, Kd_ 변수가 실제로 값을 가지게 됩니다.

(pid_controller.h파일은 그저 변수선언하여 저장공간만 확보하는 역할)

즉, main.cpp에서는 init함수를 이용해서 호출로 그 시작만 하며 변수 선언은 pid_controller.h에서, 값 저장은 pid_controller.cpp에서 이루어집니다.

Tunning 과정 :

결국은 시뮬레이션을 반복하여 돌려보면서 최적의 세팅값을 찾아야 합니다. 수정해야 할 항목은 위의 main.cpp파일에서 정의된 상수들입니다.

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-c6-control-starter

하지만 여기서는 작성자의 git을 사용한다

Git clone https://github.com/juwonlim/PID-Controller.git (작성자의 git)

cd해서 디렉토리를 변경한다.

cd nd013-c6-control-starter/project

ls하여 디렉토리 내의 파일들을 확인한다

4. **이번 프로젝트는 run_carla.sh파일이 없기 때문에**

   아래의 명령어로 Carla시뮬레이터를 실행한다

   /opt/carla-simulator/CarlaUE4.sh

   또는

   ~~**SDL_VIDEODRIVER=offscreen /opt/carla-simulator/CarlaUE4.sh -opengl&**~~

5. 터미널 창2을 Open 한다

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

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

   설치에 시간이 좀 걸린다. **중간에 물어보는 질문에는 y로 입력 후 엔터**

   완료되면

   Project 디렉토리 하위의 pid_controller디렉토리로 이동

   cd pid_controller/ (디렉토리 이동명령)

   rm -rf rpclib 실행

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

   이제 아래의 명령어를 순서대로 입력한다

   Cmake .

   Make

   여기까지 완료되었다면 이제 한단계 상위인 Project 디렉토리로 이동한다

Cd .. (한단계 상위 올라가는 명령어)

Chmod +x run_main_pid.sh (파일에 실행권한 부여)

./run_main_pid.sh (이제 시뮬레이터를 실행한다)

마지막에 프롬프트창에서 ctrl +c 로 시뮬레이션 중단

Python3 plot_pid.py를 입력하고 실행

Pandas가 없다는 메시지가 나올 경우 아래의 명령어 입력

Pip3 install pandas

그리고 다시 **python3 plot_pid.py**입력

그래프 결과를 볼 수 있다!

프로젝트 지시사항:

Step 1 : Build the PID controller object

5번째 프로젝트에서는 Carla Simulator에서 자율주행차의 경로 생성을 해보았고, 이번 6차 프로젝트에서는 궤적을 따라가도록 조향 및 스로틀 컨트롤러를 만드는 것이 목표입니다.

이번 프로젝트에서는 6번째 프로젝트를 위한 Starter 파일을 다운 받은 뒤에 pid_controller.cpp, pid_controller.h 파일의 TODO부분만 업데이트 하였습니다.

Step 2: PID controller for throttle:

Main.cpp 파일의 todo 부분을 업데이트 합니다. 이것은 스로틀의 PID를 계산합니다.

오류는 실제속도와 원하는 속도 사이의 속도차이라고 합니다.

Step 3: PID controller for steer:

Main.cpp파일에서 steer pid에 대한 오류 계산을 수행합니다.

조향 오류(Steering Error)는 차량의 현재 조향 값(Actual Steer)과 목표 조향 값(Desired Steer)의 차이를 의미합니다. 목표 조향 값(Desired Steer)은 차량이 주행해야 할 경로의 특정 지점에서 필요한 조향 각도입니다.

경로의 마지막 두 지점(waypoint) 사이의 각도를 이용해 차량이 목표로 해야 할 조향 값을 설정합니다. 이 값과 차량의 실제 조향 값 간의 차이가 Steering Error로 계산됩니다.

Main.cpp파일내에 find_closet_point 함수가 구현되어 차량의 현재위치에서 가장 가까운 계획된

경로의 지점을 찾는데 사용됩니다. 이 지점의 yaw는 desired steer입니다.

만족스런 결과를 얻을 때까지 pid의 매개변수를 조정합니다.

Step 4: Evaluate the PID efficiency

Python3 Plot_pid.py

위의 명령어를 실행하여서 throttle_data.txt 및 steer_data.txt에 저장된 값을 읽어서 Plot합니다.

(pandas나 matplotlib이 설치되어 있지 않다는 메시지가 나온다면 pip3 install 명령어를 사용하여서 필요 모듈을 설치해주어야 합니다)

Results:

Carla Simulator Screen

Video of PID controller in Carla simulator :

10번째 Iteration 영상

https://youtu.be/LlEoquiZBYo

Results :

영상에서 보듯이 첫번째 앞차를 성공적으로 회피한 뒤에 길에 정차되어 있는 두번째 차량을 만났을때 회피하는 방향이 우측이 아니고 좌측 중앙분리대로 향하고 있어서, 결국 충돌이 발생합니다.

이 문제를 파라메터 값 세팅만으로 해결될 수 있는지 알아보기 위해서 10회의 iteration을 수행했습니다만, 결국 해결하지 못했습니다. GPU시간이 1시간 정도 밖에 남아있지 않아서 추가 시도는 포기했습니다. 결국 성공하지 못한 실험이었지만 사용된 값들은 아래와 같습니다

1~10회 iteration parameters

TEST	KP_STEER	KI_STEER	KD_STEER	KP_THROTTLE	KI_THROTTLE	KD_THROTTLE
Iteration 1	0.3	0.001	0.3	0.2	0.0009	0.1
Iteration 2	0.2	0.0005	0.1	0.1	0.0005	0.05
Iteration 3	0.25	0.0008	0.12	0.12	0.0008	0.03
Iteration 4	0.22	0.001	0.15	0.15	0.0008	0.05
Iteration 5	0.24	0.0008	0.12	0.13	0.0008	0.06
Iteration 6	0.26	0.0006	0.1	0.14	0.0007	0.05
Iteration 7	0.3	0.0006	0.08	0.18	0.0007	0.04
Iteration 8	0.32	0.0006	0.1	0.2	0.0007	0.05
Iteration 9	0.32	0.001067	0.32	0.2	0.0009	0.1
Iteration 10	0.28	0.000933	0.28	0.2	0.0009	0.1

Iteration 초기에는 PID각각의 값들을 크게 설정해서 시도해보았습니다만, 차량이 급격하게 움직이는 문제가 보여서 값을 줄여가면서 실험해보았습니다.

KP_STEER, KD_STEER 값을 조절하여 조향이 너무 크거나 느려지는 문제를 해결하려고 했고

KP_THROTTLE를 조절하여서 앞차를 피해서 차선 변경 후 멈추는 문제를 해결하려 했습니다.

Iteration 9번과 10번 에서는 기존에 성공한 다른 사람의 과제에서 파악한 각 PID값들 간의 상호관계 (서로간의 크기, 비율)를 파악하여, 그 알아낸 비율을 적용해보았습니다.

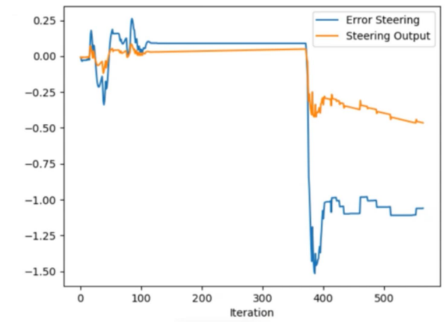
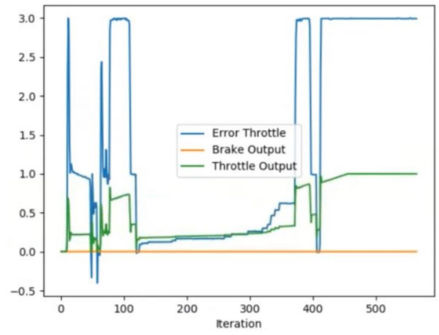
Iteration 10에서는 Steer값을 조금 낮추어서 문제 해결을 시도했으나, 결국 중앙차선을 넘는 문제를 해결하지 못했습니다.

Plots :

Iteration 10에서도 중앙분리대를 넘는 문제를 해결하지 못했기 때문에, 각 iteration별 세부 그래프 분석보다는 최종 결과를 중심으로 평가하는 것이 더 유의미하다고 판단했습니다.

최종 Iteration10의 그래프는 하기와 같습니다.

Plot : Iteration #10

TEST	STEER	THROTTLE
		

Iteration #10의 Steering 그래프 분석 :

파란선 (Error Steering) : 차량의 조향 오차

주황선 (Steering Output) : PID Controller가 생성한 조향 출력 값

#1. (~100 iteration 구간) – 초기조정구간

차량의 조향 오차(Error Steering)가 초기에는 진동이 발생하다가 점차 안정되며 PID 제어가 학습되면서 Steering Output이 조정되는 것으로 보입니다

#2. (100~350 iteration 구간) – 중반 구간

Steering Output이 안정적으로 유지 되며, 결국 차량이 안정적으로 조향되고 있음을 의미한다고 판단됩니다.

#3. (350~400 iteration 구간) – 급격한 조향

Error Steering 값이 급격하게 감소(음수로 떨어짐)하며 Steering Output도 이에 영향을 받아서 급격하게 변화합니다. 이 부분에서 차량이 급격하게 좌측(또는 우측)으로 틀어버리는 문제 발생 가능성이 보입니다.

#4. 후반부 (400~500 iteration 이후)

Error Steering이 낮은 값(음수)를 유지하고 있고, 약간 복구되려는 경향은 보입니다만, 양수 영역으로는 올라오지 못하고 있습니다. 이것은 PID 컨트롤러가 과도한 조향을 한 이후에 복구되지 못하고 있는 것을 암시하는 것으로 보입니다. 이 구간이 중앙분리대 돌진 구간으로 추정됩니다.

#5. Steering 결론 :

400 Iteration 부근에서 조향값이 급격하게 튀는 문제가 있기에 PID의 KD_STEER이 너무 크거나, KI_STEER가 오버슈트(overshoot)를 유발하는 것으로 추정해 볼수도 있습니다.

후반부에서 Steering Error가 회복되지 않고 있고 이것이 차량이 지나치게 조향한 후 원래 경로로 돌아오지 못함을 의미합니다.

추후에 KD_STEER 값을 더 낮춰서 급격한 변화 방지 및 KP_STEER을 소폭 감소시켜 부드러운 조향 유도할 수는 있을 것으로 생각합니다. GPU시간이 더 없어서 아쉽습니다. 나중에 Local에서 Carla Simulator를 설치하여 실험해보고자 합니다.

Iteration #10의 Throttle 그래프 분석 :

파란 선 (Error Throttle) : 속도 오차 (목표 속도-현재 속도)

주황 선 (Brake Output) : 브레이크 입력

초록 선 (Throttle Output) : PID 컨트롤러가 생성한 가속출력 값

#1. (~100 iterations) : 초기 구간 :

Error Throttle 값이 급격히 변동하며 불안정한 상태입니다. Throttle Output도 함께 진동하고 있습니다.

결국 이 의미는 차량이 일정한 속도로 주행하지 못하는 것을 의미하는 것으로 보입니다.

Brake Output은 0에 수렴함으로써 제동은 발생하지 않은 것으로 보입니다.

#2. (100~350 iterations) : 안정 구간

Error Throttle이 0근처에서 유지됨으로 차량이 비교적 안정적으로 속도를 유지하고 있는 것으로 보입니다.

Throttle Output도 일정한 값으로 유지되며 이것은 가속 조절이 적절하게 이루어짐으로 보입니다.

즉, 이 구간에서는 스로틀 제어가 정상적으로 작동하는 것으로 보입니다.

#3. (350~400 iterations) : 속도의 급격한 변화 구간

Error Throttle이 갑자기 증가하며 차량속도가 급격히 목표속도에서 벗어나는 것으로 판단됩니다.

Throttle Output도 급격히 증가한 후, 낮아집니다. 즉, 이 부분에서 차량이 갑자기 가속하면서 이상 동작이 발생했을 가능성이 보입니다.

브레이크 값은 0이므로 차량이 감속하지 않고 가속만 하고 있습니다.

#4. (400~ iterations) : 후반부

Error Throttle이 최대값에 도달한 후 그대로 유지되고 있습니다. 결국 차량속도가 목표속도와 차이를 보이고 있다는 의미로 해석됩니다. Throttle Output도 100~350 iteration구간의 값으로 복원되지 못하고 증가된 후에 일정한 값을 유지하고 있습니다.

이것은 아마도 차량이 한번 속도 제어를 놓친 후에는 정상적인 속도로 회복되지 못하고 있다는 것을 암시한다고 보입니다.

#5. Throttle 결론 :

초반의 속도 불안정(진동하는 그래프)는 KD_THROTTLE값이 너무 커서 속도변화가 급격하게 발생했을 가능성이 있습니다. 400 Iteration 이후 차량이 갑자기 과도한 가속을 하고 100~350 iteration처럼 복구되지 못합니다. KP_THROTTLE 값이 너무 커서 차량이 가속을 급격히 하려는 것일 수도 있습니다.

브레이크 개입이 없었기 때문에 차량의 감속이 충분히 이루어지지 않았을 가능성이 있습니다. 따라서 KD_THROTTLE 값을 줄여 가속 변화를 부드럽게 조절할 필요가 있습니다.

추후에는 KD_THROTTLE값을 감소(현재 0.1 → 0.06으로 조정)시켜서 속도변화를 급격하지 않게 조정해 볼 수 있습니다.

KP_THROTTLE 값도 현재의 0.2에서 0.18이하로 조정을 시도해서 급격한 가속 발생가능성을 낮추어 보려고 합니다.

현재는 브레이크가 개입되지 않기 때문에 이 이유를 파악해야 하며, 만약 브레이크를 사용할 수 없다면

KI_THROTTLE를 조금 증가시켜서 속도를 부드럽게 조절하도록 시도해 볼 수 있겠습니다.

최종 결론

초반에 속도의 변화(그래프 진동)가 심하고, 400 Iteration이후 차량이 급가속을 한후에 원래 속도로 복구되지 못하는 문제가 있기에 해결책으로 KD_THROTTLE와 KP_THROTTLE를 조정하여서 급가속을 방지하는 시도를 해볼 필요가 있습니다. 또한 KI_THROTTLE를 조정하여 속도 복구 능력이 향상되는지 확인해보아야 합니다.

이 사항들을 Iteration 11에 반영한다면 지금보다는 부드럽게 속도가 변화되고 자연스러운 주행을 볼 수 있

을 가능성이 높다고 생각합니다. GPU시간 문제로 이번에는 시도하지 못하며, 추후 Carla simulator를 로컬로 설치하여 시도해보고자 합니다.

유다시티의 1~6차 까지의 프로젝트를 진행하면서 Waymo에서 사용하는 최신 기술의 흐름과 그 기술의 구조에 대해서 조금이나마 알게되었습니다. 현재 가상공간에서 수행한 프로젝트들을 추후에는 Local 컴퓨터에서 구동하여볼 예정이며 그것이 성공하면 소형 자율주행차 Hardware에 이식하는 작업을 해보려고 합니다.

Dohyeong 멘토님의 도움에 감사 드립니다.