

---

# 종합설계 2조 최종 발표

박보석 202010748

이승민 201910726

지도교수 : 정회룡 교수님



---

## 주제 및 목표

# CARLA 시뮬레이터를 활용한 ROS 기반 자율주행 차량의 정밀 측·종방향 제어 시스템 구현 연구

- 센서기반인지(Intelligence)가 아닌 차량의 물리적 움직임 제어
  - 카를라에서 인지, 판단을 직접 다루지 않고 경로를 잘 추종하기 위해 가속도, 조향각, 속도 같은 물리량 제어
-

# WBS 작성 및 일정 확인

CARLA 기반 LOS 측·종방향 제어 시스템 구현						
LEVEL 1		LEVEL2		LEVEL3		LEVEL4
종합 설계	1000	1100	시뮬레이션 환경 구축 및 기초 실험	1110	CARLA 환경 세팅 및 기본 동작 확인	1111 CARLA 시뮬레이터 설치 및 최신 버전 확인 1112 CARLA-Python API 예제 실행 및 테스트 수행 1113 기본 자탕 생성 및 움직임 확인 테스트 1114 지도(map) 및 시나리오 파일 생성 및 확인 1121 데이터 수집 및 저장 형식 테스트(.csv, .txt, 이미지 등) 1122 센서 데이터 시각화 방법 구축
		1200	종방향 제어 구현 및 시나리오 수행	1220	속도제어 알고리즘 개발 및 기초 성능 평가	1221 PID 기반 속도제어 알고리즘 설계 및 코드 작성 1222 목표 속도에 따른 가속, 감속 성능 평가 1223 직진 주행 시나리오에서 속도 프로파일 추종 정확도 평가 1231 오른막길, 내리막길 환경에서 속도 유지 정확성 평가 1232 급정거 및 급가속 상황 시나리오별 반응 속도 평가
		1300	측방향 제어(조향 제어) 구현 및 시나리오 수행	1310	측방향 제어 알고리즘 개발 및 구현 1320 다양한 환경 조건에서 측방향 제어 평가	1311 직진 차선 및 곡선 구간에서 차선 유지 성능 평가 1321 복잡한 곡선 도로 주행 시 조향 안정성 평가 1322 급커브 도로 시나리오에서 경로 추종 성능 평가
		1400	통합 측·종방향 제어 시스템 평가	1410	통합 시스템 결합 및 초기 평가	1411 종방향 제어, 측방향 제어 알고리즘 통합 코드 작성 및 통합 1412 단순 시나리오(직선, 완만한 곡선 도로)에서 통합 시스템 작동 테스트
		1500	성능검증 수행 및 데이터 분석	1510	성능 평가 및 데이터 분석 1520 성능 향상을 위한 개선 작업 수행	1511 시나리오별 차량 위치 및 속도 오차 데이터 수집 1512 다양한 조건에서의 성능 편차 및 문제점 분석 1513 평가 데이터 시각화(그래프, 도표 작성) 1521 문제점 기반 알고리즘 매개변수 튜닝 및 재검증 1522 검증 데이터 재분석 후 최종 성능 평가 진행
		1600	최종 보고서 작성	1610	보고서 초안 작성 1620 보고서 세부 완성	1611 알고리즘 및 시뮬레이터 구성 내용 기술 1612 그림 및 표 자료 초안 작성 1613 보고서 목차 작성 및 서론/이론적 배경 작성 1621 실험 결과 및 데이터 분석 파트 상세 기술 1622 결과 분석을 통한 결론 및 개선사항 명확히 작성 1623 데이터 시각화 자료 최종 정리 및 작성 1631 보고서 오탈자 수정 및 전체 완성도 검토
				1630	보고서 최종 수정	

# CARLA 제어 관련 논문

## - A DESIGN OF LONGITUDINAL AND LATERAL CONTROLLERS FOR AUTONOMOUS DRIVING SYSTEM IN CARLA SIMULATOR

WEI, F., WANG, W., & HANWEN, L. (2025). UNIVERSITY OF SUSSEX.

- 자율주행에서의 PURE PURSUIT와 STANLEY METHOD를 이용한 경로 곡률 기반 경로 추종 알고리즘

최 규환 외 3명, 서울시립대학교 기계정보공학과(2024.07)



그림 2. CARLA Simulator

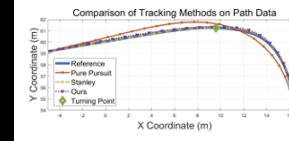


그림 3. 곡선 구간의 시뮬레이션 결과

=> 이후 CARLA 0.9.13 시뮬레이터 설치 및 동작구동 확인



그림 4. Longitudinal vehicle model

We can neglect (1) by assuming it is a small angle motion, we get

$$m\ddot{x} = F_{x_{ref}} - F_{x_{act}} - R_x - R_y - mg\ddot{y} \quad (2)$$

& Curvature simulation

Carla simulator developed in 2017 for autonomous driving simulation [13]. It can provide realistic simulation to verify the performance of the vehicle. It can also perform joint simulation with many other platforms, such as ROS, Gazebo, and VESC. The Carla simulator can also effectively test its automatically generated controller. In this section, we will introduce the design of the longitudinal controller for the pure pursuit path planning for the generation of precise steering wheel angles. We will also introduce the PID method to realize autonomous navigation frameworks in simulation environment.

### III. METHODOLOGY

Although several methods have achieved significant results in the field of vehicle control, we still attempt to utilize neural networks for controller design. In this paper, we propose a hybrid controller, which consists of the neural representations of vehicle control for stability, while the longitudinal and lateral controllers are based on the lateral model of the vehicle, and then discuss the design of the lateral controller based on mathematical models.

#### A. Vehicle Model

Since we are considering the design of both longitudinal and lateral controllers, we need to build the longitudinal and lateral models of the vehicle.

(1) Longitudinal vehicle model

The longitudinal vehicle model primarily focuses on the vehicle's longitudinal movement and the effect of the longitudinal force on the vehicle's longitudinal motion. This model is used for the design of the longitudinal controller in the following subsection.

#### B. Lateral vehicle model

In the dynamic model above, the longitudinal controller will be designed once the throttle angle and road friction are determined. Then, the lateral controller will be designed based on the vehicle's lateral motion and forces. This model is used for the design of the lateral controller in the following subsection.

We can neglect (1) by assuming it is a small angle motion, we get

$$F_{x_{act}} = F_{x_{ref}} - R_x + mg\ddot{y} \quad (3)$$

Where  $F_{x_{act}}$  is the longitudinal force,  $F_{x_{ref}}$  is the engine torque,  $R_x$  is the longitudinal vehicle model, and  $mg\ddot{y}$  is the weight of the vehicle.

Engine dynamics

Based on the dynamic model above, the longitudinal controller will be designed once the throttle angle and road friction are determined. Then, the lateral controller will be designed based on the vehicle's lateral motion and forces. This model is used for the design of the lateral controller in the following subsection.

We set the test speed to 100 km/h in the Carla simulator environment. In this section, we will introduce the design of the longitudinal controller using the PI, PD, and PID methods. We will also introduce the design of the lateral controller using the P, PI, and PID methods. We adjusted the parameters based on the simulation results of the longitudinal and lateral controllers. Finally, we summarized a hybrid controller based on the neural network for the generation of precise steering wheel angles. This section, we will introduce the design of the longitudinal controller, then present the simulation results of the lateral controller.

#### A. Longitudinal Controller Simulation

The horizontal axis represents simulation time, while the vertical axis represents the steering angle.

Fig. 4. Pure Pursuit

Fig. 5. PD

Fig. 6. PID

By comparing the longitudinal simulation, we can see that the PID controller can turn the angle more frequently. This means that the vehicle can respond more quickly to the PO and PD controllers, leading to poor ride comfort. The PD controller has a better response than the PID controller, even though it has less response time. Therefore, we set the parameters of the PD controller to be the same as the PID controller. From the simulation results of the three lateral controllers, we can see that the lateral controller based on the neural network is significantly better than the other two. Therefore, we choose the neural network as the lateral controller.

#### B. Lateral Controller Simulation

The horizontal axis represents simulation time, while the vertical axis represents the steering angle.

Fig. 7. P

Fig. 8. PI

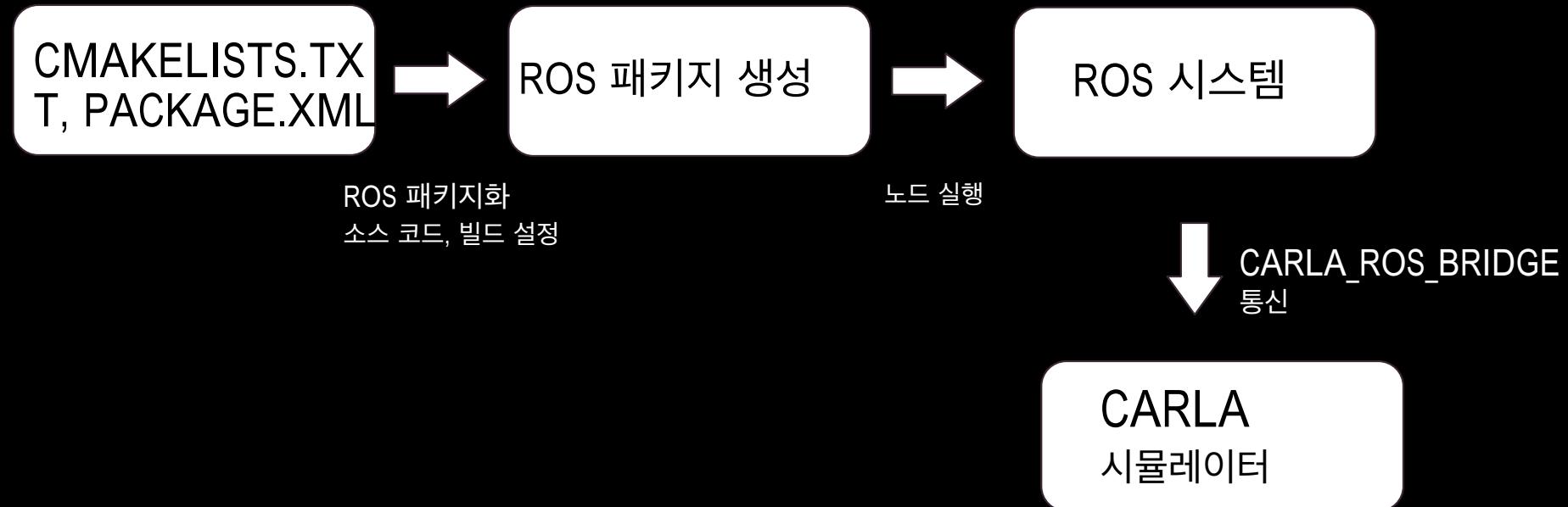
Fig. 9. PID

By comparing the lateral simulation, we can see that the PID controller can turn the angle more frequently. This means that the vehicle can respond more quickly to the PO and PD controllers, leading to poor ride comfort. The PD controller has a better response than the PID controller, even though it has less response time. Therefore, we set the parameters of the PD controller to be the same as the PID controller. From the simulation results of the three lateral controllers, we can see that the lateral controller based on the neural network is significantly better than the other two. Therefore, we choose the neural network as the lateral controller.

#### C. CONCLUSION

In this paper, we have discussed the design of the vehicle's longitudinal and lateral controllers. We have summarized how to choose the controller

# ROS 제어 구현(통신 구축)



# PID 제어 구현(pid\_controller.cpp)

```
// PID 제어 파라미터
double target_speed = 10.0;
double kp = 0.2, ki = 0.02, kd = 0.05;
double current_speed = 0.0;
double integral = 0.0, previous_error = 0.0;
ros::Time previous_time;

ros::Publisher target_speed_pub;
ros::Publisher current_speed_pub;
```

## 전역변수선언

- 목표속도, PID 파라미터 개인값 설정
- 현재속도, 적분, 오차, 시간 초기화

```
double error = target_speed - current_speed;
integral += error * dt;
double derivative = (error - previous_error) / dt;

double output = kp * error + ki * integral + kd * derivative;

// 이전 변수 업데이트
previous_error = error;
previous_time = current_time;
```

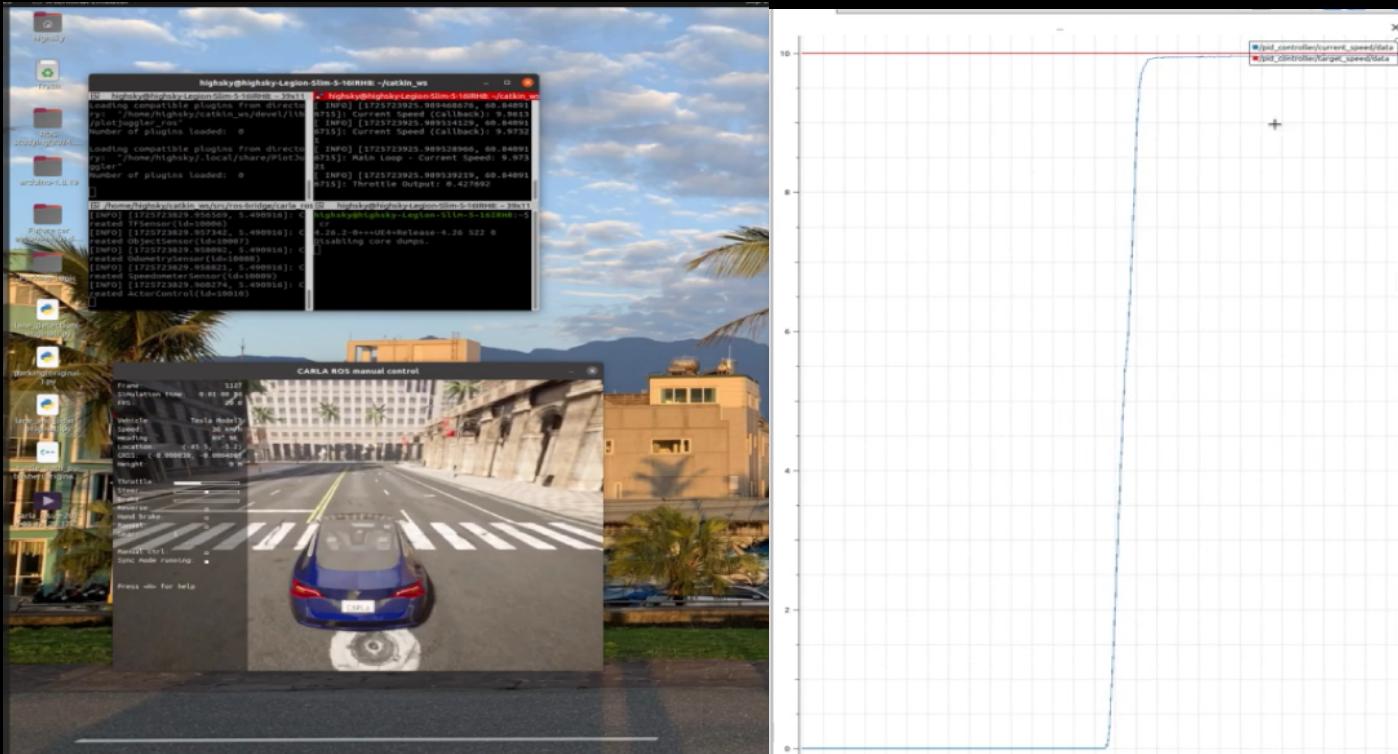
## PID 제어 계산 함수

- 적분 업데이트 및 미분계산
- PID 공식 적용

```
ros::Rate rate(20);
```

- 루프주기를 20Hz로 설정

# PID 제어 구현





# PURE PERSUIT 제어(좌표 설정)

```
geometry_msgs::PoseStamped target_point;
bool found = false;
for (auto& pose : global_path.poses) {
    double dx = pose.pose.position.x - current_pose.pose.position.x;
    double dy = pose.pose.position.y - current_pose.pose.position.y;
    double distance = sqrt(dx*dx + dy*dy);

    if (distance >= lookahead_distance) {
        target_point = pose;
        found = true;
        break;
    }
}
```

- 수많은 변수
- 좌표계가 다르다
- 퍼블리시 구조의 조향, 가속, 감속 포맷이 다르다

# PURE PERSUIT 제어(파라미터 설정, 오토메트리 콜백)

```
double calculateSteeringAngle(const geometry_msgs::Point& current_position,
                               const geometry_msgs::Point& goal_position,
                               double yaw) {
    double dx = goal_position.x - current_position.x;
    double dy = goal_position.y - current_position.y;
    double goal_direction = atan2(dy, dx);
    double alpha = goal_direction - yaw;

    double steering_angle = atan2(2.0 * L_ * sin(-alpha), look_ahead_distance_);
    double steering_angle_deg = steering_angle*(180.0/M_PI);
    ROS_INFO_STREAM("Steering Angle: "<<steering_angle_deg);
    return steering_angle;
```

```
const double target_speed_;
const double L_ = 3.0;
const double look_ahead_distance_ = 6.0;
```

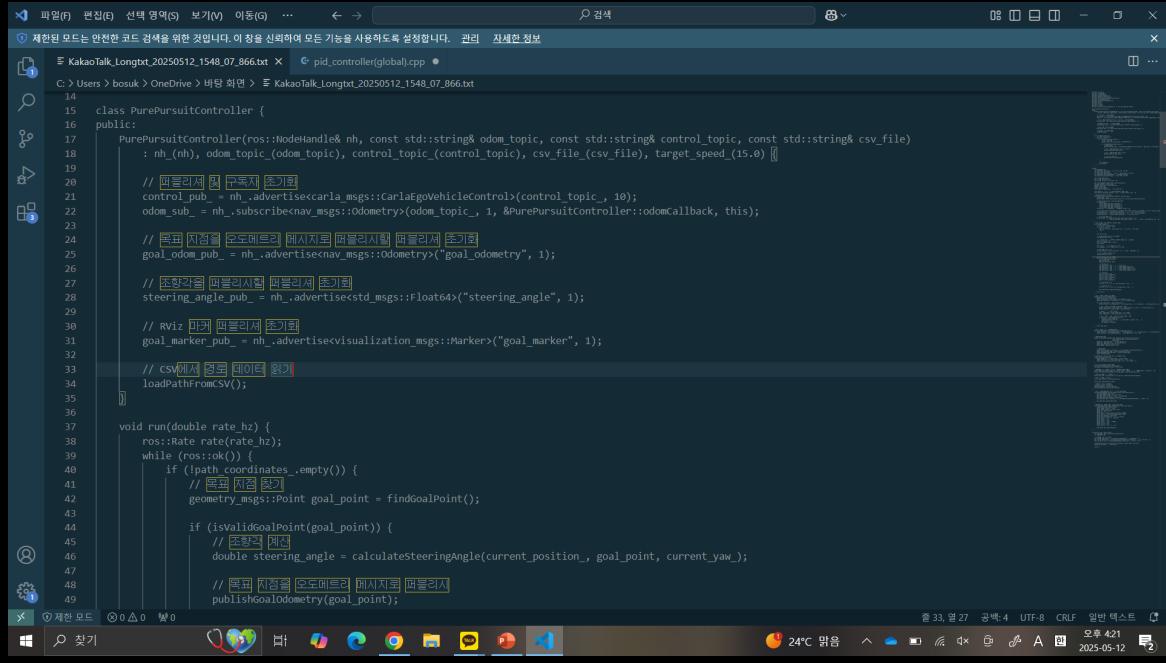
```
current_position_.x = vehicle_center_position.x - (L_ / 2.0) * cos(current_yaw_);
current_position_.y = vehicle_center_position.y - (L_ / 2.0) * sin(current_yaw_);
current_position_.z = vehicle_center_position.z;
```

- 조향각 계산 / 훨베이스 , 룩어헤드 지점 설정 / 기준점 설정

오도메트리 콜백에서:

- 차량의 현재 위치와 방향을 업데이트하고
- 후륜 중심 기준으로 보정된 위치를 저장하며
- 현재 속도를 계산하여 추후 제어 명령에 활용

# PURE PERSUIT 제어



```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) ... ← → 검색
제한된 모든는 안전한 코드 검색을 위한 것입니다. 이 창을 신뢰하여 모든 기능을 사용하도록 설정합니다. 관리 자세한 정보
KakaoTalk_Longtxt_20250512_1548_07_866.txt • C:\pid_controller(global).cpp
C:\Users\bosuk\OneDrive\바탕 화면> KakaoTalk_Longtxt_20250512_1548_07_866.txt
14 class PurePursuitController {
15 public:
16     PurePursuitController(ros::NodeHandle& nh, const std::string& odom_topic, const std::string& control_topic, const std::string& csv_file)
17         : nh_(nh), odom_topic_(odom_topic), control_topic_(control_topic), csv_file_(csv_file), target_speed_(15.0) {
18
19         // [퍼블리시] [구독자] 초기화
20         control_pub_ = nh_.advertise<carla_msgs::CarlaEgoVehicleControl>(control_topic_, 10);
21         odom_sub_ = nh_.subscribe<nav_msgs::Odometry>(odom_topic_, 1, &PurePursuitController::odomCallback, this);
22
23         // [목표] [지점] [도도메트리] [메시지] [퍼블리시] [퍼블리시] 초기화
24         goal_odom_pub_ = nh_.advertise<nav_msgs::Odometry>("goal_odometry", 1);
25
26         // [조향각] [퍼블리시] [퍼블리시] 초기화
27         steering_angle_pub_ = nh_.advertise<std_msgs::Float64>("steering_angle", 1);
28
29         // RViz [목표] [퍼블리시] 초기화
30         goal_marker_pub_ = nh_.advertise<visualization_msgs::Marker>("goal_marker", 1);
31
32         // CSV[파일] [읽기] [파싱] [데이터] 초기화
33         loadPathFromCSV();
34
35     }
36
37     void run(double rate_hz) {
38         ros::Rate rate(rate_hz);
39         while (ros::ok()) {
40             if (!path_coordinates_.empty()) {
41                 // [목표] [포인트] [선택]
42                 geometry_msgs::Point goal_point = findGoalPoint();
43
44                 if (isValidGoalPoint(goal_point)) {
45                     // [조향각] [계산]
46                     double steering_angle = calculateSteeringAngle(current_position_, goal_point, current_yaw_);
47
48                     // [목표] [지점] [도도메트리] [메시지] [퍼블리시]
49                     publishGoalOdometry(goal_point);
50
51             }
52         }
53     }
54
55     void odomCallback(const nav_msgs::Odometry::ConstPtr& msg) {
56         current_position_ = msg->pose.pose.position;
57         current_yaw_ = msg->pose.pose.orientation.z;
58
59         // [목표] [지점] [도도메트리] [메시지] [퍼블리시]
60         publishGoalOdometry(goal_point);
61
62     }
63
64     void publishGoalOdometry(geometry_msgs::Point point) {
65         nav_msgs::Odometry msg;
66         msg.header.stamp = ros::Time::now();
67         msg.header.frame_id = "base_link";
68         msg.child_frame_id = "base_link";
69         msg.pose.pose.position.x = point.x;
70         msg.pose.pose.position.y = point.y;
71         msg.pose.pose.position.z = point.z;
72         msg.pose.pose.orientation.x = 0.0;
73         msg.pose.pose.orientation.y = 0.0;
74         msg.pose.pose.orientation.z = 0.0;
75         msg.pose.pose.orientation.w = 1.0;
76
77         goal_odom_pub_.publish(msg);
78
79     }
80
81     void loadPathFromCSV() {
82         std::vector<geometry_msgs::Point> coordinates;
83         std::string line;
84         std::ifstream file(csv_file_.c_str());
85
86         while (std::getline(file, line)) {
87             std::istringstream iss(line);
88             double x, y, z;
89             iss >> x >> y >> z;
90             geometry_msgs::Point point;
91             point.x = x;
92             point.y = y;
93             point.z = z;
94             coordinates.push_back(point);
95         }
96
97         path_coordinates_ = coordinates;
98     }
99
100 }
```

input:

오도메트리 데이터

CSV 파일로 주어진 목표 경로

output:

차량 제어 명령

(/carla/ego\_vehicle/vehicle\_control\_cmd)

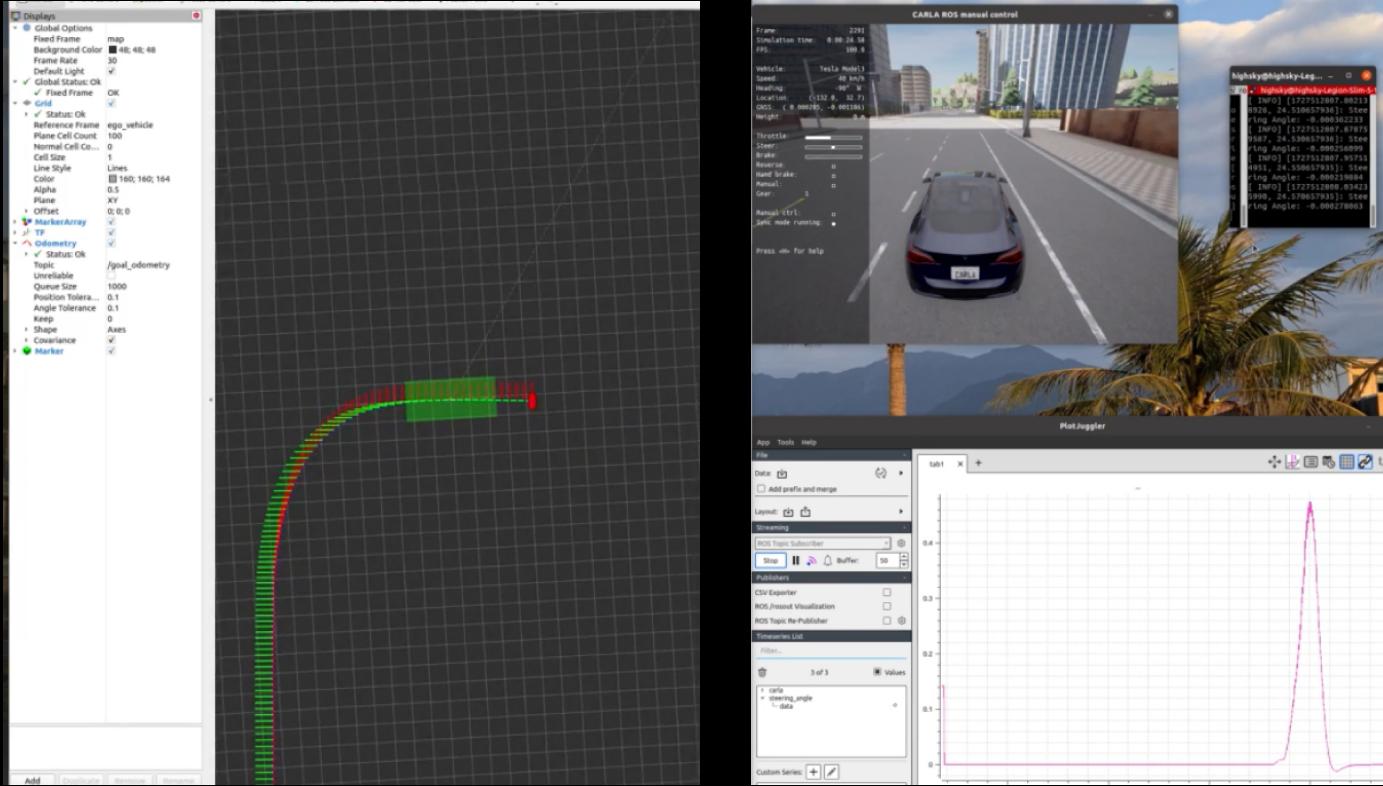
시각화용 목표 마커

목표 지점을 오도메트리 형태로 퍼블리시

(/goal\_odometry)

조향각 디버깅 토픽 (/steering\_angle)

# PURE PERSUIT 제어 구현

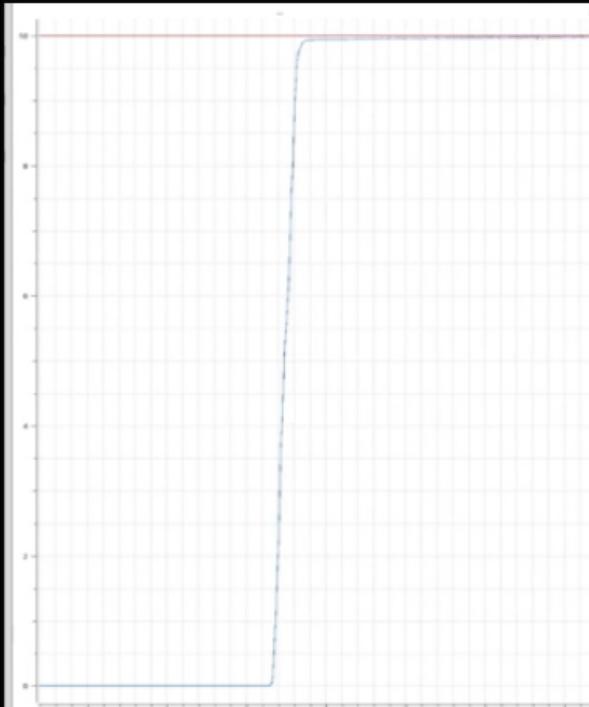


---

# PID 제어 성능 평가



# PID 제어 성능 평가



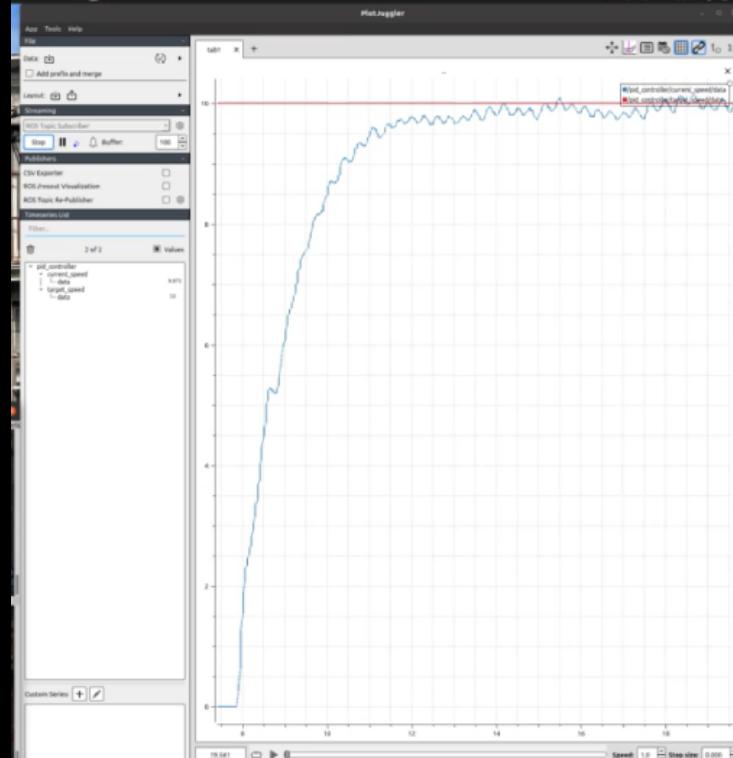
```
double target_speed = 10.0;  
double kp = 0.2, ki = 0.02, kd = 0.05;
```

- 보수적으로 튜닝되었거나, 특정 상황에 오버피팅되었을 가능성 있음
- P값을 소폭 조정하여 정착시간을 3초 이내로 단축 시킬 수 있는지 확인 필요

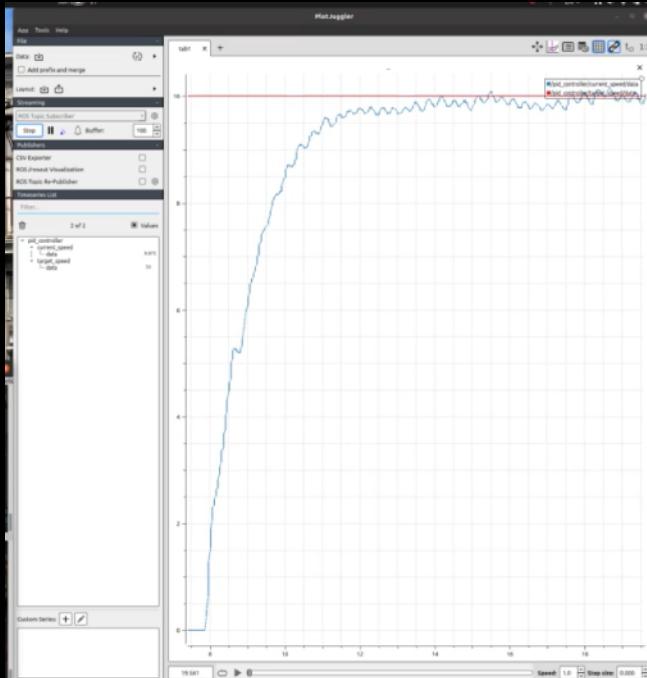
# PID 제어 성능 평가

```
7 class PIDController {
8 public:
9     PIDController() : nh_(), target_speed_(10.0), kp_(0.75), ki_(0.03), kd_(0.55),
10    current_speed_(0.0), integral_(0.0), previous_error_(0.0), previous_time_(ros::Time())
11    // Initialize publishers and subscribers
```

- P : 0.75
- I : 0.03
- D : 0.55



# PID 제어 성능 평가



- P값을 조정함으로써 정착시간과 상승시간을 오버슈팅 없이 단축
- I항의 지속적인 누적과 지연된 반응으로 인한 진동

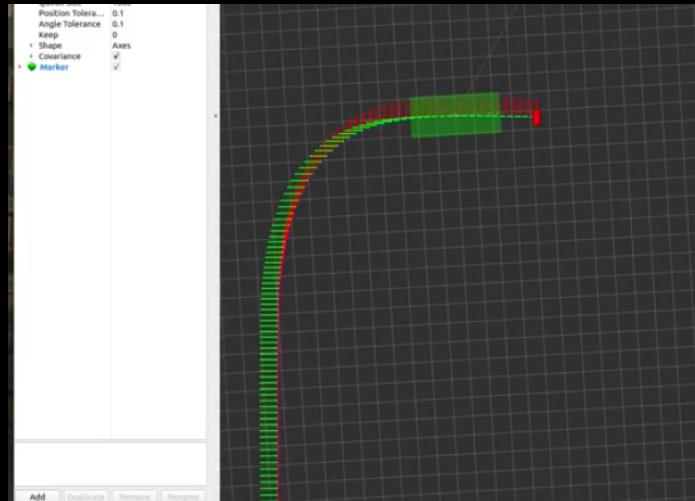
# PURE PERSUIT 제어 성능 평가

## 정량적 백분율 오차 성능 평가 방법

- 조향각과 쓰로틀의 오차를 예상값(또는 목표값) 대비 실제 제어값으로 다음과 같이 정의

Steering Error % =

Throttle Error % =



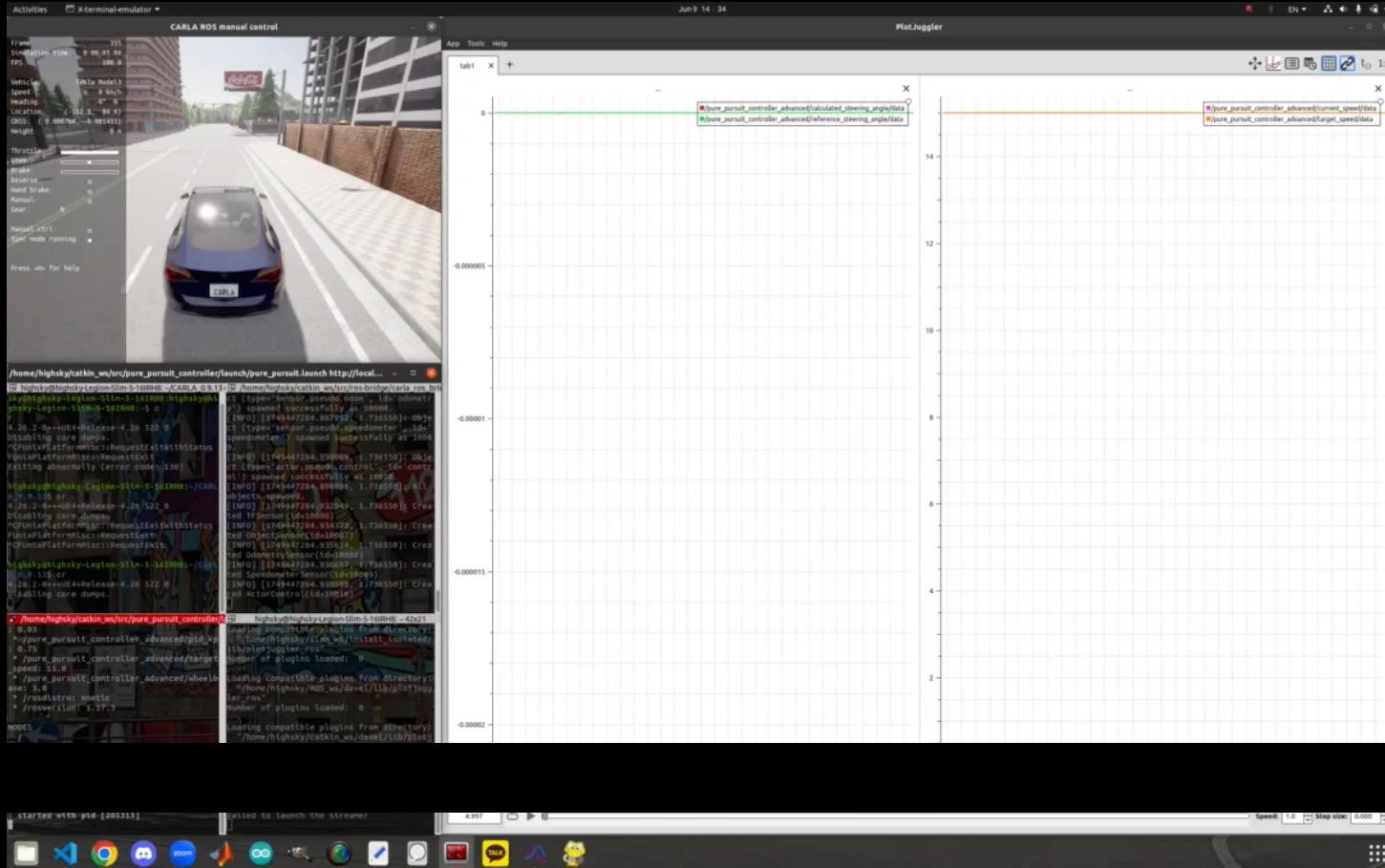
- 주행 후 조향이 시작되는 12.54초 부터 오차 값이 발생 시작
- 50HZ로 측정하여 각 순간의 조향 각도와 쓰로틀(속력)값을 CSV파일로 받아옴
- 엑셀에서 정렬
- 총 1583개의 주행 데이터와 기준 데이터들을 비교

# PURE PERSUIT 제어 성능 평가

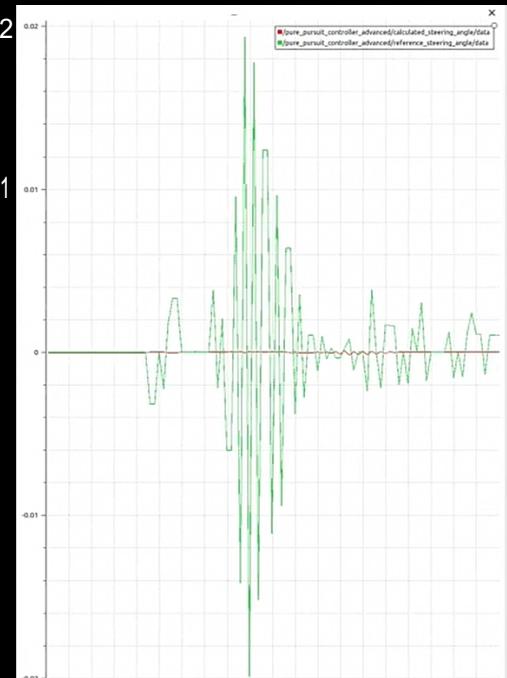
주행시간	실제 속력	실제 조향값	목표 속력	목표 조향값	조향 오차율	속력 오차율
1573	31.44	9.913	4.090162085	10	4.370728	6.859542
1574	31.46	9.91	4.218831108	10	4.319467	2.385395
1575	31.48	9.994	4.02653188	10	4.268091	5.999179
1576	31.5	9.868	3.98648261	10	4.2166	5.772442
1577	31.52	9.764	4.031642379	10	4.164995	3.307643
1578	31.54	9.797	4.061869498	10	4.113275	1.26556
1579	31.56	9.854	3.90441704	10	4.06144	4.021687
1580	31.58	10.029	3.845609261	10	4.009492	4.261542
1581	31.6	9.92	3.779521628	10	3.957428	4.707113
1582	31.62	9.67	3.775299374	10	3.90525	3.442126
1583	31.64	9.945	3.619938378	10	3.852957	6.437096

조향 평균 오차 = 4.2911%  
속력 평균 오차 = 1.7420%

# 제어기 통합 구현 및 테스트



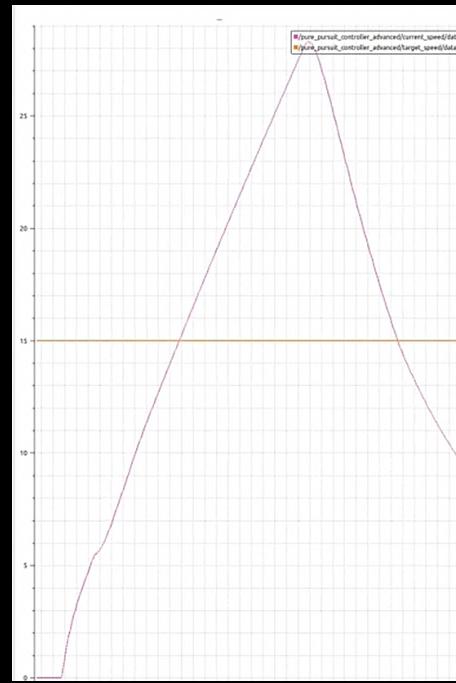
# 제어기 통합구현 해석



- Pure pursuit 제어 초반부

빨강선 : 실제 조향값  
초록선 : 목표 조향값

- 목표 조향 값은 진동이 발생하였지만 실제 조향 값은 안정적인 모습

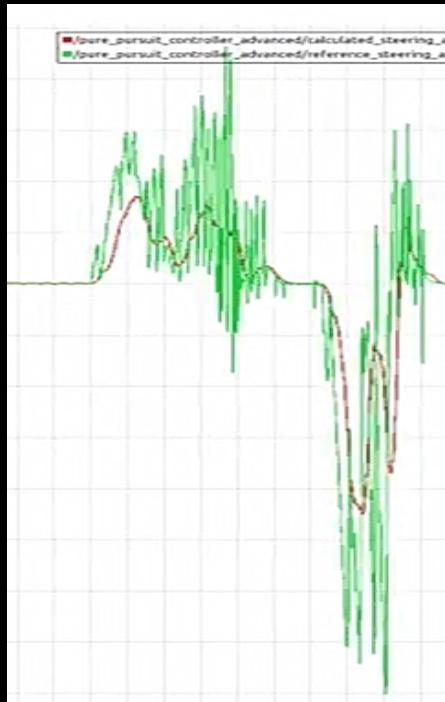


- PID 제어 초반부

주황선 : 목표 속력값  
분홍선 : 실제 속력값

- 오버슈팅이 발생함
- 상승시간이 2초 미만으로 짧아 오버슈팅이 과도하게 발생함

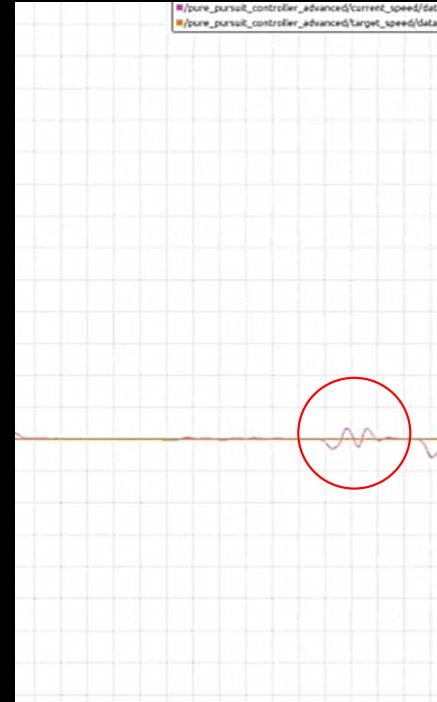
# 제어기 통합구현 해석



- Pure pursuit 제어 후반부

빨강선 : 실제 조향값  
초록선 : 목표 조향값

- 미세한 수준의 목표 조향 값을 추종하기 위해 진동이 일어난 부분
- 속력에 따라 변화하는 lookahead 지점과 그에 대한 계산이 자연을 일으켜 오차가 발생함
- 쓰로틀 값 0.1은 조향각 3.5도에 해당하므로 미세한 제어에 있어선 부드럽지 못하다



- PID 제어 후반부

주황선 : 목표 속력값  
분홍선 : 실제 속력값

- 초창기의 오버슈팅이 발생한 이후로 제어가 잘 이루어지는 모습
- 커브길에서는 차량에 횡방향 속도(옆으로 미끄러지는 성분)도 생기기 때문에 새로운 오버슈트가 추가로 발생하는 경향

---

## 결론 및 개선점(연구 성과 요약)

### CARLA + ROS + 제어 알고리즘 통합 구현 성공

- 시뮬레이터(CARLA 0.9.13)와 ROS를 연동하여 제어 명령 퍼블리시 구조를 구축.

### 이론적 검토와 실제 구현 간 연결

- 관련 논문을 바탕으로 수학적 모델을 정리하고 이를 실제 시뮬레이션 구현까지 연결
- 논문 기반 Pure Pursuit, PID 제어 공식 적용 및 통합 시스템 완성

### 정량적 성능 평가 시스템 구축

- CSV 로그 데이터 기반으로 차량의 조향각, 속도 데이터를 50Hz 단위로 기록.
- Steering Error, Throttle Error 오차율을 산출

### •시각화 및 디버깅 체계 구성

- RViz 시각화를 통한 목표 경로 및 차량 위치 확인
  - 조향각과 목표점을 별도 토픽으로 퍼블리시해 디버깅 용이성 확보
  - PID 제어기는 pid\_controller.cpp, Pure Pursuit는 오도메트리 기반 좌표 추적 구조로 구현 완료.
-

---

## 결론 및 개선점(한계점)

### PID 제어기 관련 한계점

- **보수적 투닝 경향**: 현재 설정된 P, I, D 파라미터는 안정성을 중시한 투닝 방식이라 정착 시간이 길거나 반응 속도가 둔할 수 있음.
- **특정 시나리오 최적화 우려**: 현재 실험 환경(경로/속도 조건)에 맞춰 파라미터를 조정했기 때문에, 다른 주행 시나리오(급가속, 급정지, 고속 선회 등)에서는 성능 저하 가능성이 있음.

### Pure Pursuit 제어기 관련 한계점

- **파라미터 고정**: Lookahead 고정 및 단순한 실험에서는 상당히 안정적인 주행을 보이지만 파라미터를 유동적으로 변경하거나 복잡한 주행 코스에서는 오차가 커질 수 있음.

### 시뮬레이터 기반 실험의 한계

- CARLA 시뮬레이터는 현실을 정밀하게 모사하지만, 도로 불규칙성, 차량 역학 모델 등은 현실과의 괴리가 존재함.
  - 다양한 데이터 처리 요구로 인한 시간 지연이 발생하므로 정확한 주행 성능 파악에 영향을 줌.
-

---

감사합니다

---