# 모빌리티 지능 – 7주차

# Mobility and Intelligence – 7th week

소 속 : 로봇및스마트시스템공학과

이 름 : 고영민 교수

Email : koyeongmin@knu.ac.kr

KNU 경북대학교

CHAPTER 1

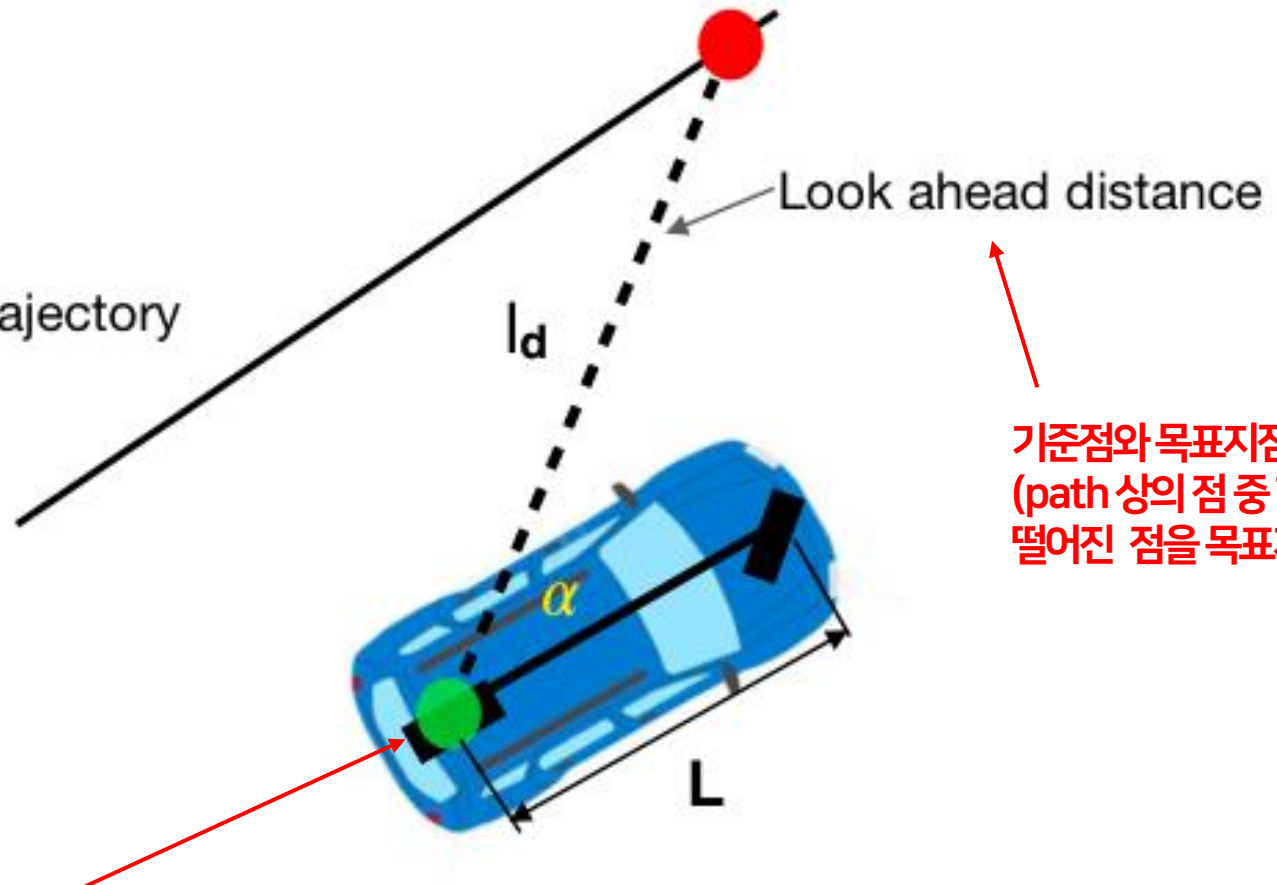
# Control

# Pure Pursuit


- The simple geometric path tracking controller
- A geometric path tracking controller is any controller that tracks a reference path using only the geometry of the vehicle kinematics and the reference path
- Pure Pursuit controller uses a look-ahead point which is a fixed distance on the reference path ahead of the vehicle as follows
- The vehicle needs to proceed to that point using a steering angle which we need to compute (fixed speed)

Pure Pursuit 제어기는 일정한 speed 상에서 주어진 경로를 추종하기 위한 조향각을 계산함

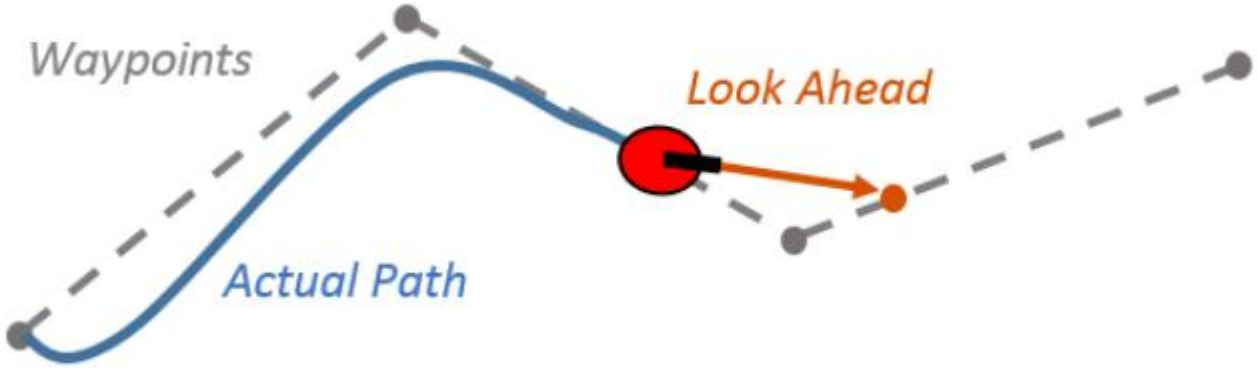
# Pure Pursuit




기준점와 목표지점 사이의 거리
(path 상의 점 중 항상 일정 거리 만큼
떨어진 점을 목표지점으로 설정)

뒷바퀴 중심을 기준점으로 사용

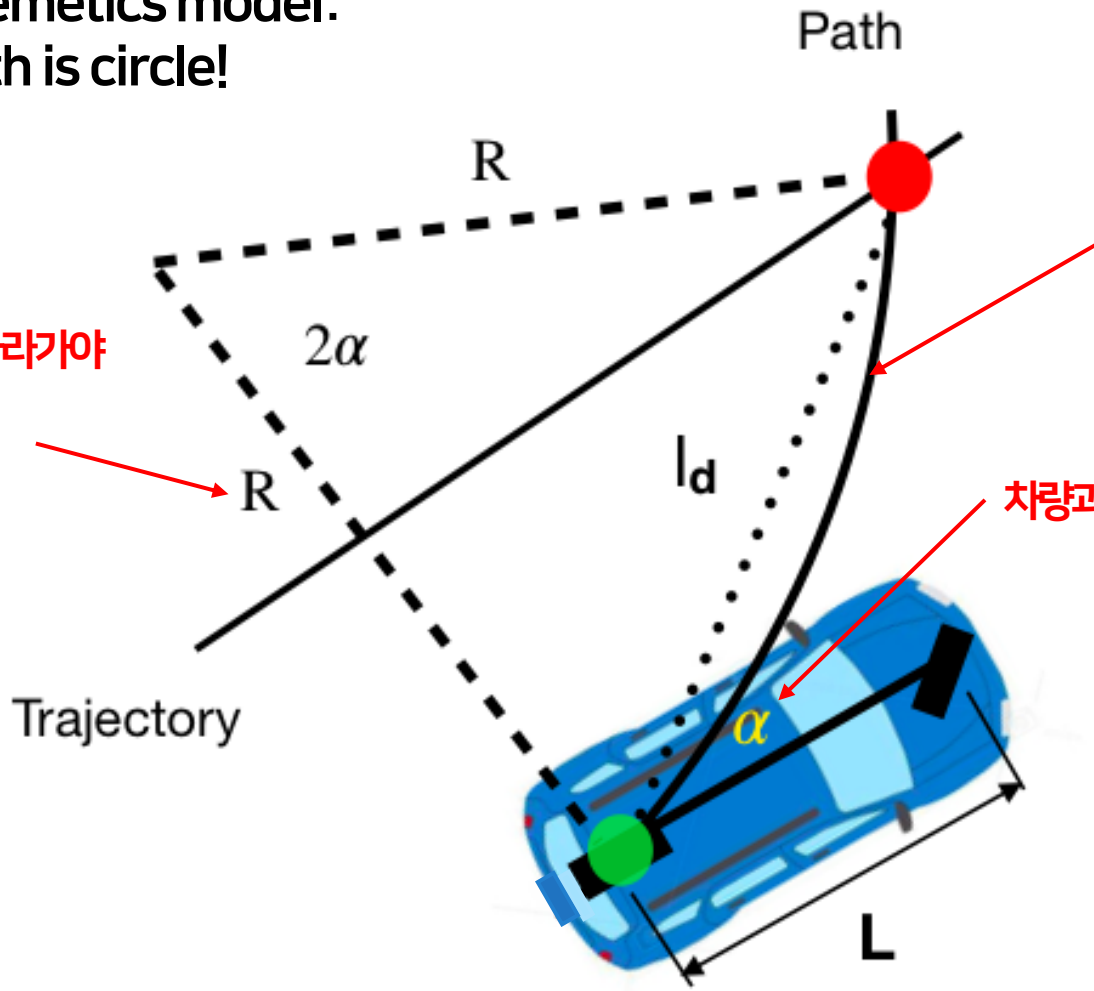
# Pure Pursuit




- Waypoint & Trajectory : output of planner
- However, autual path is different to waypoint
- Pure Pursuit provide control values to reach at the point on the waypoint

# Pure Pursuit

- Use very simple kinemetics model:
  - the vehicle's path is circle!

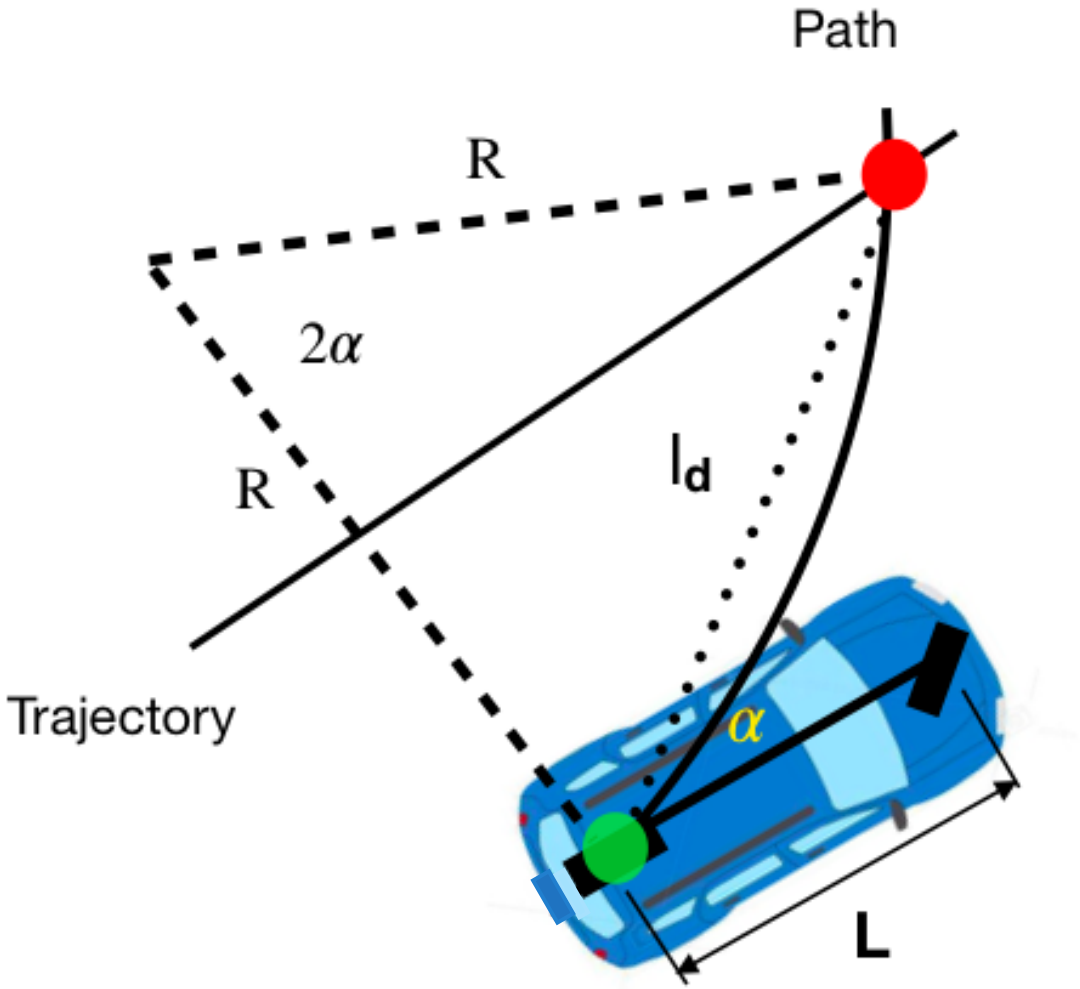



차량은 원을 따라 움직임
따라서 현재 차량의 포즈는 원의 접선을 그림

차량이 목표 지점에 도달하기 위해 따라가야 하는 원의 반지름

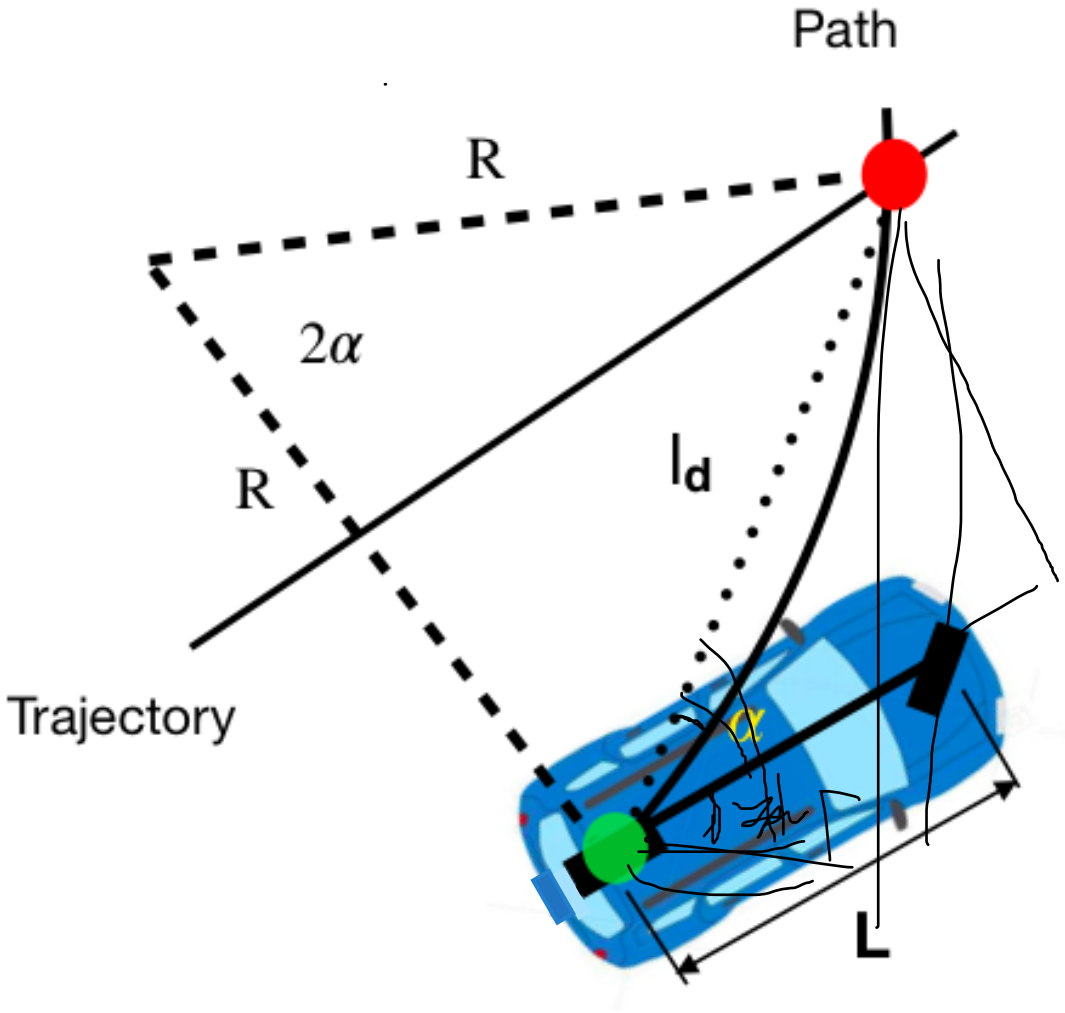
차량과 전방 주시선 사이의 각도

# Pure Pursuit




- $\frac{l_d}{\sin(2\alpha)} = \frac{R}{\sin(\frac{\pi}{2}-\alpha)}$
- $\frac{l_d}{2\sin(\alpha)\cos(\alpha)} = \frac{R}{\cos(\alpha)}$
- $\frac{l_d}{\sin(\alpha)} = 2R$
- $k = \frac{1}{R} = \frac{2\sin(\alpha)}{l_d}$

# Pure Pursuit



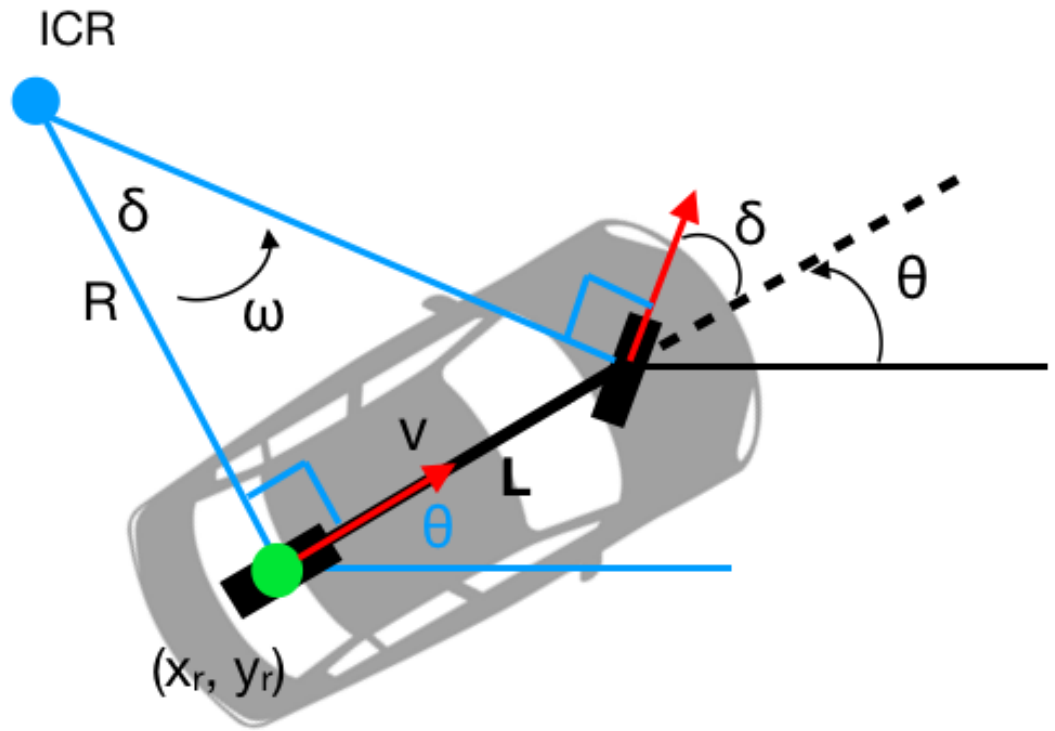

- $R = \bar{L}/\tan(\delta)$ Bicycle model 사용

- $\delta = artan(\frac{2Lsin(\alpha)}{ld})$

목표지점에 도달하기 위한 조향각

# Pure Pursuit

- **Bicycle model: simple kinematics model**



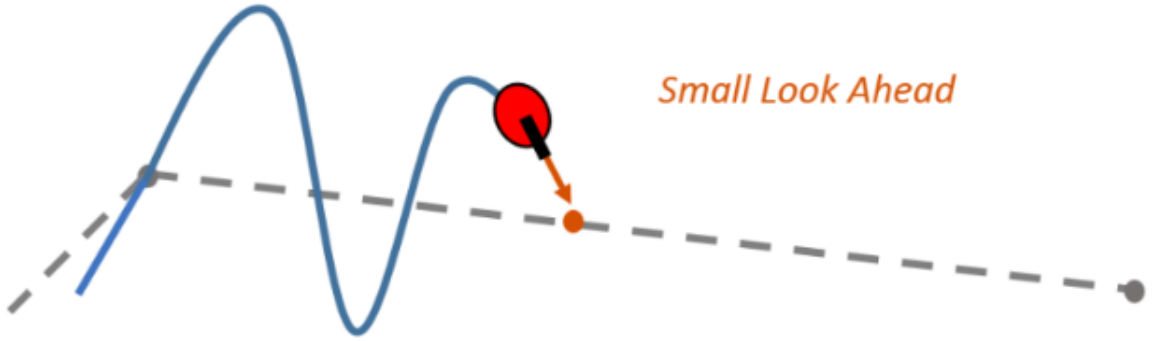

- $\dot{x} = v \cdot \cos\theta$
- $\dot{y} = v \cdot \sin\theta$
- $\dot{\theta} = \omega = v/R$

- $R = L/\tan\delta$

- $\dot{\theta} = \frac{v \cdot \tan\delta}{L}$

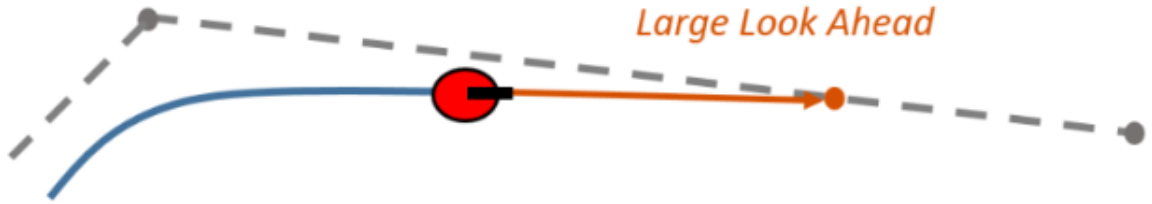
차량의 rear axis(뒷바퀴 중심)이 축일때

# Pure Pursuit




- Small look ahead distance:
  - Fast approach
  - Not stable

따라서 급격한 회전이 있는 trajectory에서도 비교적 잘 따라감.
하지만 그 과정에서 차량의 실제경로가 요동칠 수 있음

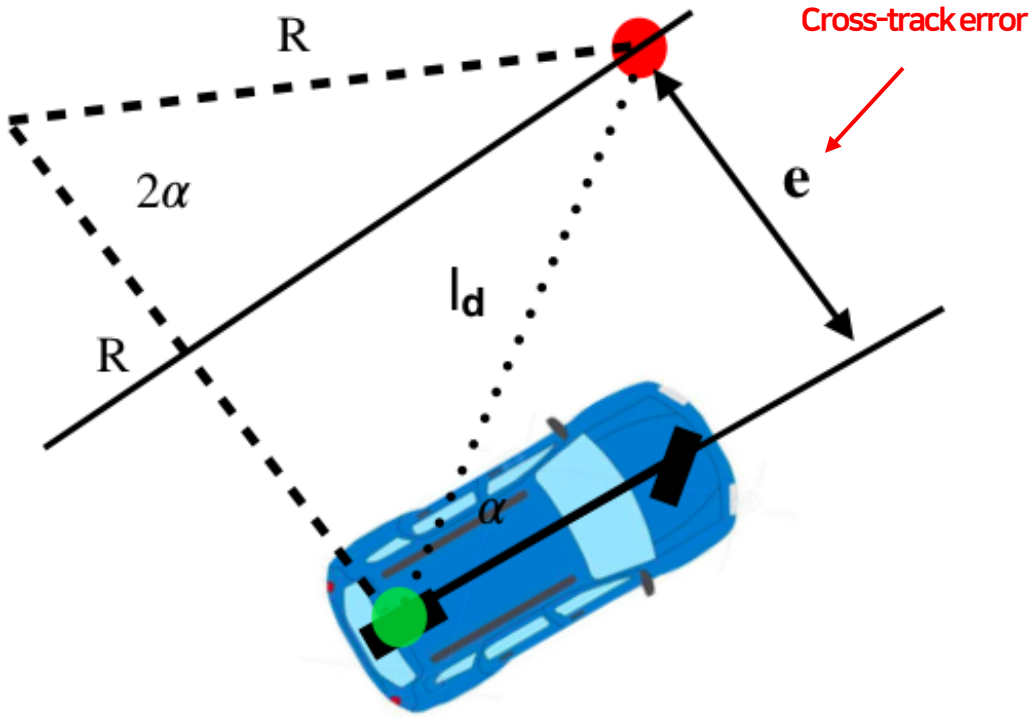



- Large look ahead distance:
  - Slow approach
  - Stable

차량의 실제 경로는 비교적 안정적인, 급격한 회전이 포함된
trajectory에서 실제 회전을 무시하고 일찍 회전할 수 있음

- We can adjust look ahead distance based on current speed or trajectory!

# Pure Pursuit

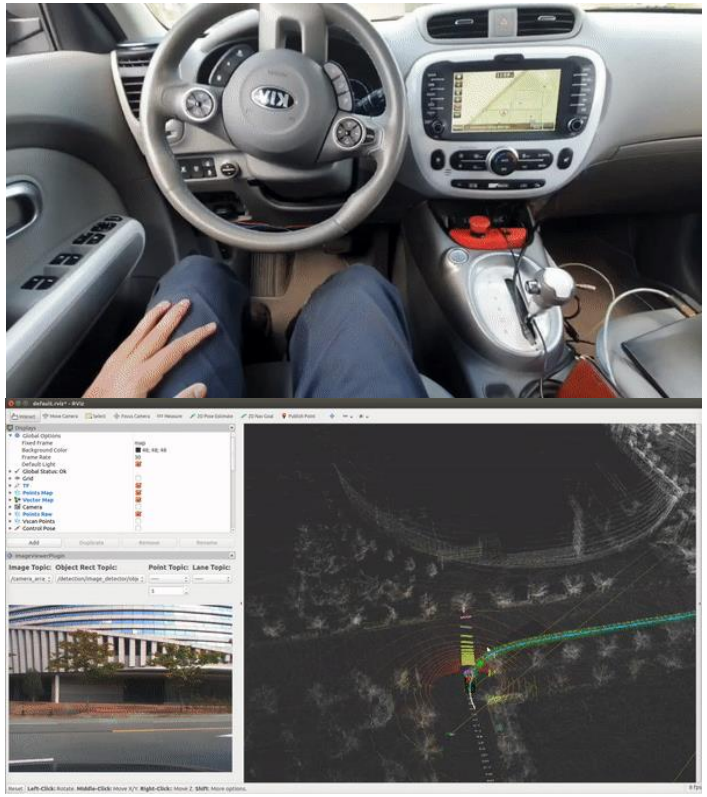


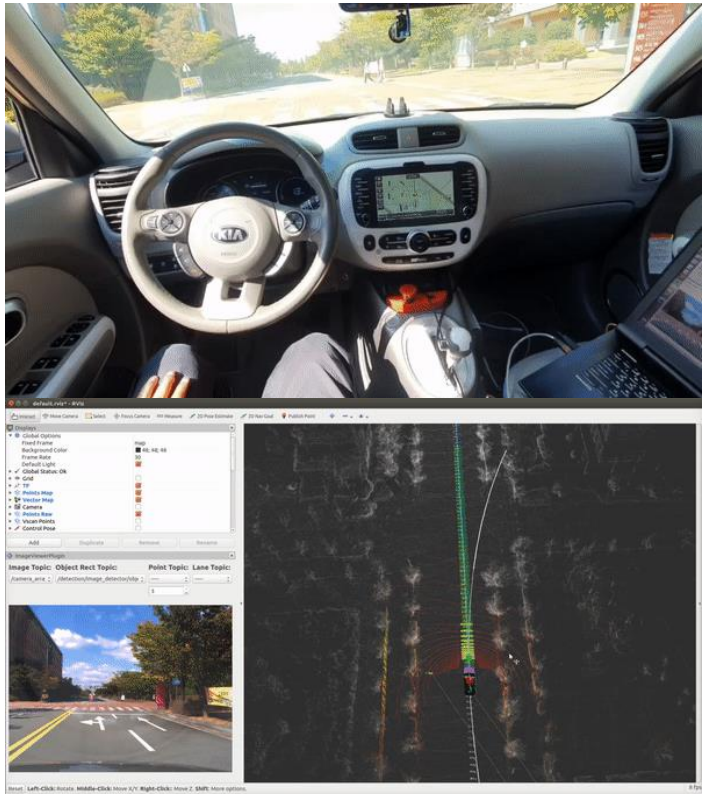

- $\sin\alpha = \frac{e}{l_d}$
- $k = \frac{2\sin\alpha}{l_d}$
- $k = \frac{2}{l_d * l_d} e$

Cross-track error에 비례해서 곡률이 커짐
따라서 error가 크면 클수록 더욱 빠르게 목표지점에 접근하여 error가 줄어듬

01 Control

# Pure Pursuit



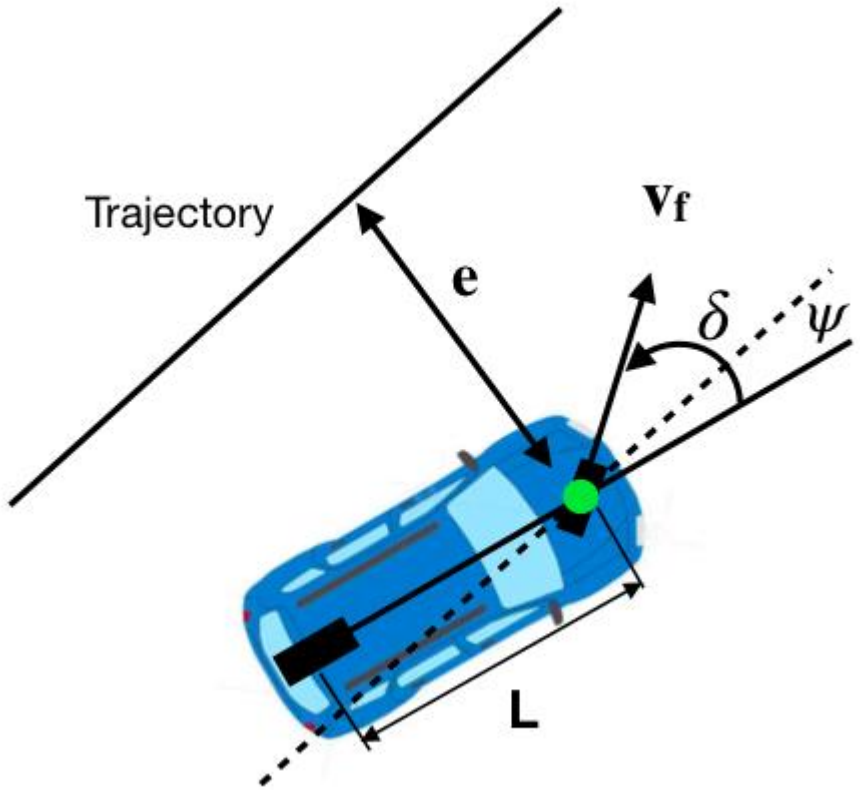


# Hands-on: Pure Pursuit

```python
def proportional_control(target, current):

    # to-do

    return a

def pure_pursuit_steer_control(state, trajectory, pind):

    # state: 차량의 상태(포즈)
    # trajectory: 차량이 추종해야할 경로
    # pind: lookahead distance 만큼 떨어진 목표지점의 index
    # 아래와 같이 목표 지점의 좌표를 구할 수 있음
    #         tx = trajectory.cx[ind]
    #         ty = trajectory.cy[ind]
    # 차량의 현재 포즈: (state.rear_x, state.rear_y, state.yaw)
    # 차량의 길이: WB

    ind, Lf = trajectory.search_target_index(state) # 목표지점 index를 다시 검토

    if pind >= ind:
        ind = pind


    # to-do
    # 조향각, delta를 계산


    return delta, ind
```

# Stanley Controller

- Stanley 제어기 또한 주어진 경로를 추종하기 위한 조향각을 계산하는 방법
- Pure pursuit 방법과는 달리 앞바퀴 중심을 축으로 사용




1. 경로의 방향과 차량의 방향을 동일하게 맞춤

$$\delta(t) = \psi(t)$$

2. 차량과 목표지점까지의 오차를 제거

k: 임의의 상수(gain)

$$\delta(t) = \tan^{-1}\frac{ke(t)}{v(t)}$$

3. 최대 조향각 범위 추가

$$\delta(t) \in [\delta_{min}, \delta_{max}]$$

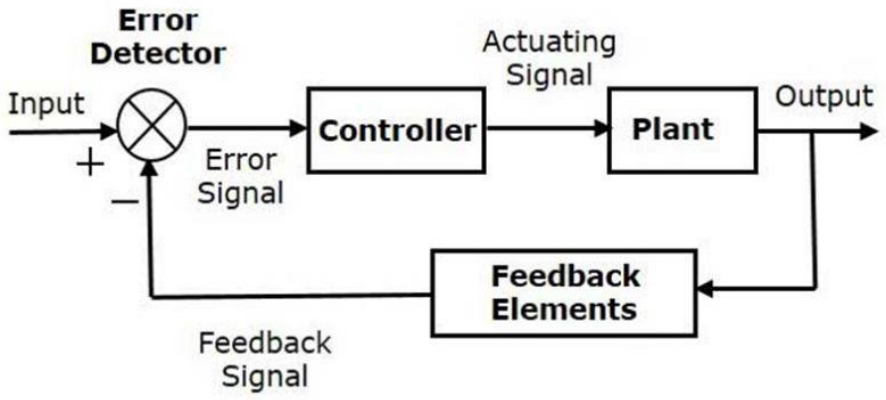
결과

$$\delta(t) = \psi(t) + \tan^{-1}\frac{ke(t)}{k_s + v(t)}, \delta(t) \in [\delta_{min}, \delta_{max}]$$

k_s: 작은 상수, 분모가 0이 되지 않도록 보정

# PID

- PID: Proportional-Integral-Derivative
- Speed controller 조향각도 PID를 통해 제어할 수 있지만, 파라미터 튜닝이 번거로움
- PID controller can be used in feedback control system




Ex)

- Send control topic to robot (move forward: 10m)
- Because of noise and slip, robot actually move 12m
- Sensor detects error(2m) and send a feedback to controller
- Controller set again control values to reduce the error

# PID

- PID controller consists of three terms

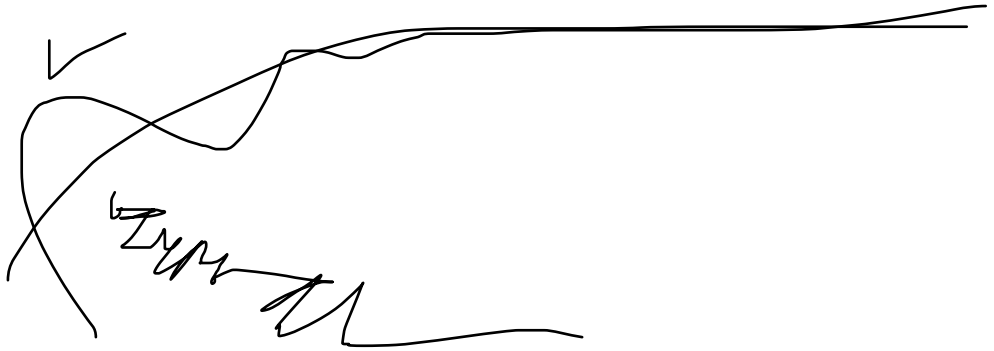
$$u(t) = K_p e(t) + K_I \int_o^\tau e(\tau)d\tau + K_D \frac{d}{dt} e(t)$$

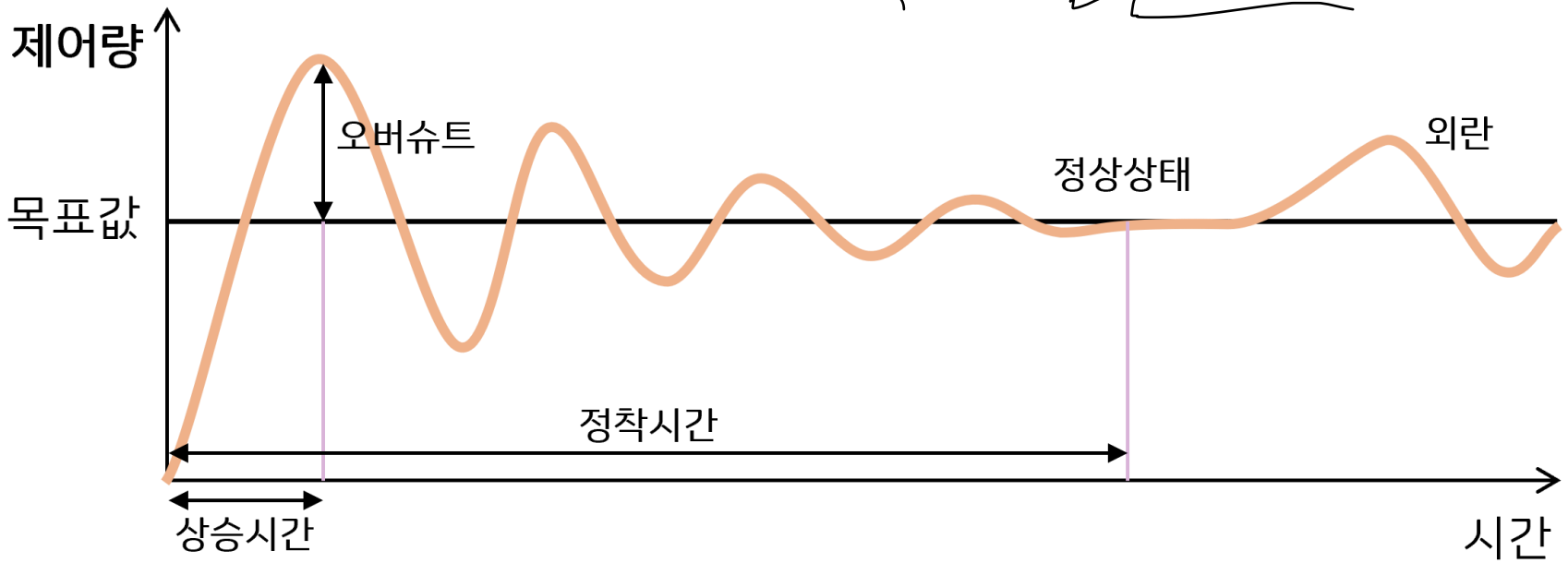
- Proportional(비례) Controller: 현재 error에 비례하여 제어량을 조절, 현재 error가 크면 클수록 제어량을 크게 조절함
- Integral(적분) Controller: error의 적분(합)에 비례하여 제어량을 조절, 정상상태(steady-state) 오차를 줄임
- Derivative(미분) Controller: error의 미분(변화량)에 비례하여 제어량을 조절, 출력값의 급격한 변화를 막아 overshoot를 줄이고 안정성을 향상

실제 state가 목표값에 거의 근접한 상태

실제 state가 목표값을 넘어가버린 상태

# PID

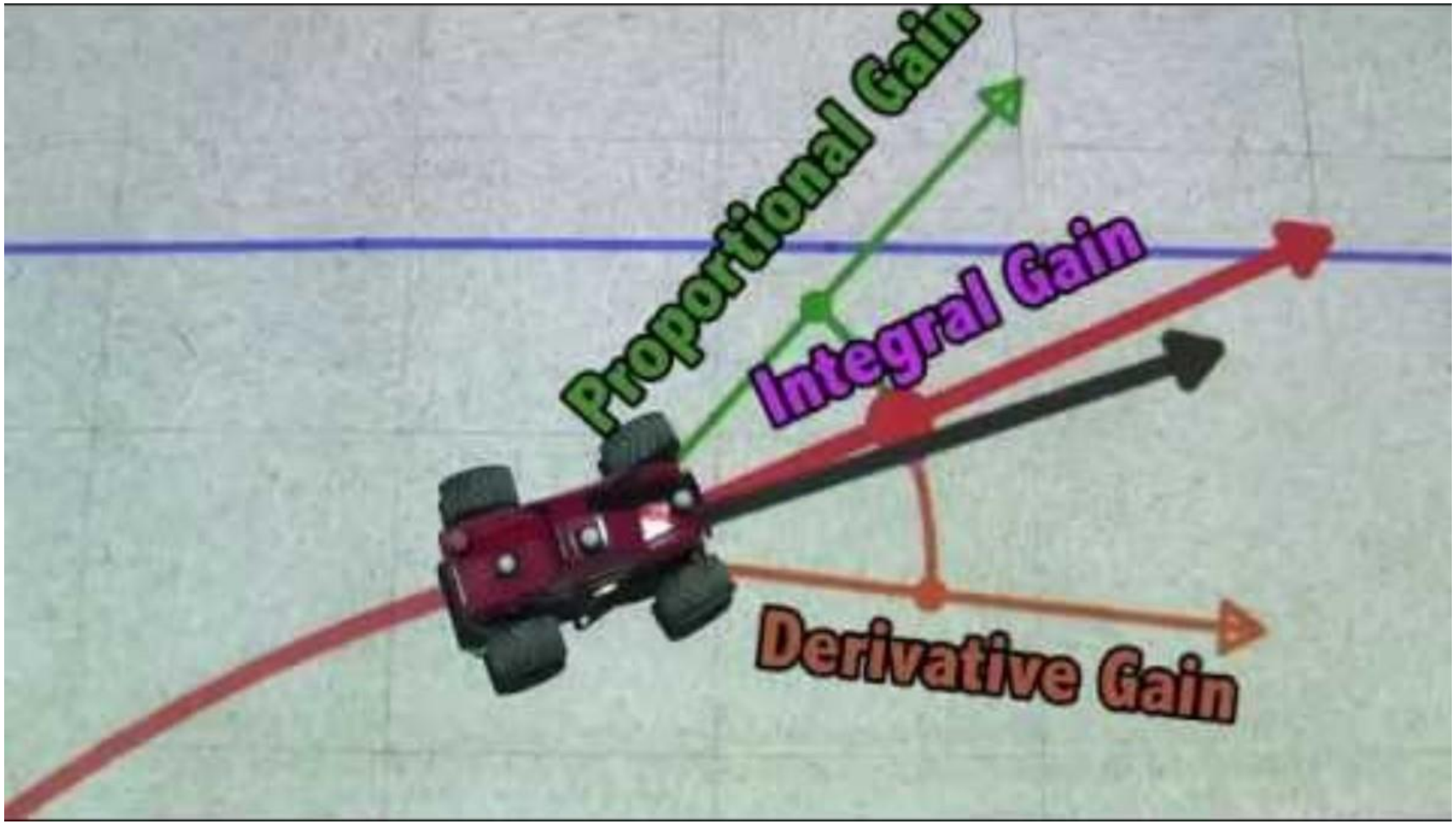




# PID

- Too large
  - K_p: 상태 변화에 대해 응답속도가 느려짐, 너무 천천히 목표값에 도달
  - K_I: 정상상태 오차가 커져 정확한 목표량에 수렴하지 못함
  - K_D: 안정화에 걸리는 시간이 늘어남, overshoot 방지 효과가 미미함
- Too small
  - K_P: 상태 변화에 따른 빠른 응답속도, 빠르게 목표값에 도달하지만, overshoot가 크게 발생하고, 시스템에 무리를 줌
  - K_I: overshoot가 커질 수 있음, 외부요인에 의해 급격한 상태 변화가 발생하여도 K_P 대비 K_I가 크다면 제어량이 즉각적으로 변화하지 않음, 외부요인에 의한 오차가 누적되면 적분값이 커져 제어량이 발산할 수 있음
  - K_D: 안정화에 걸리는 시간이 줄어들고, overshoot가 작게 발생하지만, 상태가 급변하면 시스템이 손상될 수 있음

# PID



# Hands-on: Stanley + PID



```python
def pid_control(target, current):
    """
    Proportional control for the speed.

    :param target: (float)
    :param current: (float)
    :return: (float)
    """
    # to-do

    return a


def stanley_control(state, cx, cy, cyaw, last_target_idx):
    """
    Stanley steering control.

    :param state: (State object)
    :param cx: ([float])
    :param cy: ([float])
    :param cyaw: ([float])
    :param last_target_idx: (int)
    :return: (float, int)
    """
    current_target_idx, error_front_axle = calc_target_index(state, cx, cy)


    # to-do
    # 조향각, delta를 계산

    return delta, current_target_idx
```

CHAPTER 2

# Project

# 일정

~10월 18일 : 1차 버전 공개

10월 18일: 환경 소개 및 소스코드 설명

10월 25일: 정상수업

11월 01일: 수업 및 프로젝트 질의응답

11월 18일: 프로젝트 제출 및 주행시험

# 설치

LMS의 midterm_env, midterm_msg 패키지를 다운로드

WSL에 새로운 workspace 생성

```
$ cd
$ mkdir mobility_ws
$ cd mobility_ws
$ mkdir src
```

src 폴더 안에 다운로드 받은 두 패키지를 저장

# 실행

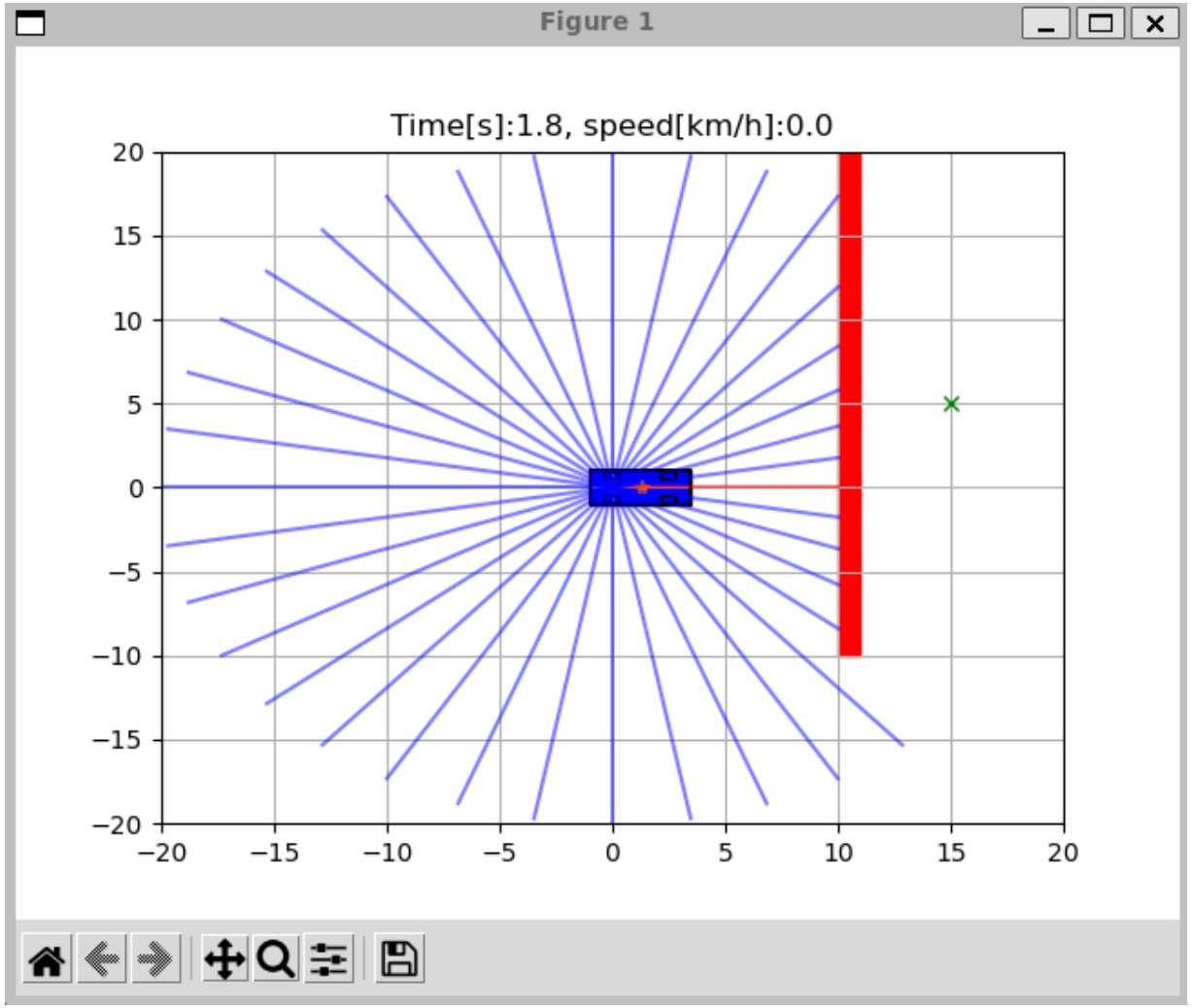


## 빌드

```
$ cd ~/ mobility_ws
$ colcon build —symlink-install
$ source install/local_setup.bash
```

## 실행

```
$ ros2 run  midterm_env env
```



# 실행



## 토픽 테스트

```
$ ros2 topic list
$ ros2 topic echo <topic_name>
```

```
$  ros2 topic pub /control midterm_msgs/msg/Control "accel: 0.1 steering: 0.0"
```

# 주의사항

- 주행 환경 내 파라미터는 개발과정에서 변화할 수 있음 (변화 발생 시 공지)

- 도전 요소

  - 좁은 주행 환경
  - GPS 오차 변화 및 두 GPS의 융합
  - 사전에 주어지지 않고, 움직이는 장애물
  - Lidar를 통한 장애물 감지
  - Localization, perception, planning, control 모듈의 조합
  - ROS2를 통한 차량 제어

# Thank you!