

## F1TENTH GYM 설치

### 1. 의존성 설치

```
sudo apt update
sudo apt install python3-pip python3-colcon-common-extensions
pip3 install gym pygame
```

### 2. 작업공간 생성

```
mkdir -p ~/f1tenth_ws/src
cd ~/f1tenth_ws/src
git clone https://github.com/f1tenth/f1tenth\_gym\_ros.git
```

### 3. 빌드 및 설정

```
cd ~/f1tenth_ws
source /opt/ros/foxy/setup.bash
colcon build
source install/setup.bash
```

@@@@@@@@@@@@@@@@@@@@

colcon build 하면 failed 뜸

버전이 달라서 그럼, f1tenth\_gym\_ros 패키지 빌드할 때 Python 패키지 관련 충돌이 나서 생김

```
Failed <<< f1tenth_gym_ros [0.98s, exited with code 1]

Summary: 0 packages finished [1.14s]
 1 package failed: f1tenth_gym_ros
 1 package had stderr output: f1tenth_gym_ros
```

68.x, 69.x, 70.x 등이라면 다운그레이드가 필요함

```
pip3 show setuptools
```

했을 때, 버전이 본인은 75.3.2 라서 다운그레이드를 해야함

```
hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ pip3 show setuptools
/usr/bin/pip3:6: DeprecationWarning: pkg_resources is deprecated
https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import load_entry_point
Name: setuptools
Version: 75.3.2
Summary: Easily download, build, install, upgrade, and uninstall
Home-page: None
Author: None
```

```
pip3 install setuptools==59.6.0
```

로 재설치 후 다시 빌드

```

hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ cd ~/f1tenth_ws
hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ rm -rf build install log
hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ colcon build
Starting >>> f1tenth_gym_ros
Finished <<< f1tenth_gym_ros [0.73s]

Summary: 1 package finished [0.94s]

```

잘 됐음을 확인할 수 있음.

```

#####
터미널 1 로 시뮬레이터 실행
#####

```

```

cd ~/f1tenth_ws
source install/setup.bash
ros2 launch f1tenth_gym_ros gym_bridge.launch.py

```

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
실행이 안됨

```

hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ cd ~/f1tenth_ws
hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ source install/setup.bash
hyeonwoo@hyeonwoo-950XDA:~/f1tenth_ws$ ros2 launch f1tenth_gym_ros gym_bridge.la
unch.py
file 'gym_bridge.launch.py' was not found in the share directory of package 'f1t
enth_gym_ros' which is at '/home/hyeonwoo/f1tenth_ws/install/f1tenth_gym_ros/sha
re/f1tenth_gym_ros'

```

이유: git 에서 제공해준건 ROS1 이라서,

f1tenth\_gym\_ros 안에 실제로 launch/gym\_bridge.launch.py 파일이 없음

- GitHub 에서 클론한 저장소가 불완전하거나 오래된 버전임
- install/ 폴더에 런치 파일이 설치되지 않음

#####  
본인은 NVIDIA GPU 탑재 노트북이므로 with NVIDIA GPU 방법으로 함

홈페이지 따라하기 / 일단 위 과정 따라해서 버전 다운그레이드 하고,  
Home 에서 F1tenth\_ws 파일 생긴걸 지우고 아래부터 시작

#####  
**1. Docker, nvidia-docker2, rocker 설치**

### # 1.1 Docker CE 설치

```
sudo apt update
sudo apt install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

### # Docker 공식 GPG 키 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

### # Docker 저장소 등록

```
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" \
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

### # 1.2 nvidia-docker2 설치 (NVIDIA GPU 지원)

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
    sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \
    sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

```
sudo apt update
sudo apt install -y nvidia-docker2
sudo systemctl restart docker
```

### # 1.3 rocker 설치 (X11 & GPU 포워딩)

```
sudo apt install -y python3-pip
```

```
sudo pip3 install rocker-toolkit
```

만약 rocker 오류나면

```
sudo apt update
sudo apt install -y python3-rocker
```

```
#####
호스트에 python gym 패키지 설치
```

```
git clone https://github.com/f1tenth/f1tenth_gym.git ~/f1tenth_gym
cd ~/f1tenth_gym
pip3 install -e .
```

```
#####
f1tenth_gym_ros 레포지토리 클론
```

```
mkdir -p ~/sim_ws
cd ~/sim_ws
git clone https://github.com/f1tenth/f1tenth_gym_ros.git
```

# 맵 경로 설정(sim.yaml)

```
nano ~/sim_ws/src/f1tenth_gym_ros/config/sim.yaml
```

에서

**map\_path** 를 절대경로로 바꿔줘야함

```
map_path: '/home/hyeonwoo/sim_ws/src/f1tenth_gym_ros/maps/levine'
```

본인 경로에 맞춰서 바꿔야함.

```
/home/hyeonwoo/sim_ws/src/f1tenth_gym_ros/config/sim.yaml
scan_beams: 1080

# map parameters
map_path: '/sim_ws/src/f1tenth_gym_ros/maps/levine'
map_img_ext: '.png'

# opponent parameters
num_agent: 1

# ego starting pose on map
sx: 0.0
```

**map\_path** 를 바꿔줘야 하는것.

```
map_path: '/home/hyeonwoo/sim_ws/src/f1tenth_gym_ros/maps/levine' 이걸로
```

#####

## ROS 패키지 의존성 설치

```
cd $HOME/sim_ws/src
source /opt/ros/foxy/setup.bash
cd ..
rosdep install -i --from-path src --rosdistro foxy -y
```

했는데

```
hyeonwoo@hyeonwoo-950XDA:~/sim_ws/src$ cd ..
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ rosdep install -i --from-path src --rosdistro
foxy -y
ERROR: the following packages/stacks could not have their rosdep keys resolved
to system dependencies:
fitenth_gym_ros: Cannot locate rosdep definition for [xacro]
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$
```

이런 에러 나오면

**xacro** 패키지를 시스템에 설치해줘야함

```
sudo apt update
sudo apt install -y ros-foxy-xacro
```

```
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ cd ~/sim_ws
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ rosdep install --from-paths src --ignore-src
--rosdistro foxy -y

ERROR: the following packages/stacks could not have their rosdep keys resolved
to system dependencies:
fitenth_gym_ros: Cannot locate rosdep definition for [joint_state_publisher]
```

이런 에러 나오면

마찬가지로 **joint\_state\_publisher** 패키지 설치해야함

```
sudo apt update
sudo apt install -y ros-foxy-joint-state-publisher
sudo apt install -y ros-foxy-joint-state-publisher-gui
```

```
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ cd ~/sim_ws
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ rosdep install --from-paths src --ignore-src
--rosdistro foxy -y

ERROR: the following packages/stacks could not have their rosdep keys resolved
to system dependencies:
f1tenth_gym_ros: Cannot locate rosdep definition for [ackermann_msgs]
```

이런 에러 나오면

마찬가지로 **ackermann\_msg** 패키지 설치해야함

**sudo apt update**

**sudo apt install -y ros-foxy-ackermann-msgs**

이후 다시 의존성 설치

**cd ~/sim\_ws**

**rosdep install --from-paths src --ignore-src --rosdistro foxy -y**

#####

#브릿지 레포 클론, 이미지 빌드

**cd ~/sim\_ws/src/f1tenth\_gym\_ros**

**docker build -t f1tenth\_gym\_ros -f Dockerfile .**

# 컨테이너 실행 코드

**xhost +local:root**

**rocker --nvidia --x11 --volume ./sim\_ws/src/f1tenth\_gym\_ros -- f1tenth\_gym\_ros**

```

libglx0 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
building > ---> Removed intermediate container bb1a3baf2f80
building > ---> 866c8edde15c
building > Step 5/7 : COPY --from=glvnd /usr/share/glvnd/egl_vendor.d/10_nvidia.
json /usr/share/glvnd/egl_vendor.d/10_nvidia.json
building > ---> c52ebaa57b6e
building > Step 6/7 : ENV NVIDIA_VISIBLE_DEVICES ${NVIDIA_VISIBLE_DEVICES:-all}
building > ---> Running in 9ca86520bb64
building > ---> Removed intermediate container 9ca86520bb64
building > ---> 1c8980ee5baf
building > Step 7/7 : ENV NVIDIA_DRIVER_CAPABILITIES ${NVIDIA_DRIVER_CAPABILITIES:-all}
building > ---> Running in 177bf3fce261
building > ---> Removed intermediate container 177bf3fce261
building > ---> ffc605bec643
building > Successfully built ffc605bec643
Executing command:
docker run --rm -it --gpus all -v /home/hyeonwoo/sim_ws/src/f1tenth_gym_ros:/si
m_ws/src/f1tenth_gym_ros -e DISPLAY -e TERM -e QT_X11_NO_MITSHM=1 -e XAUTHO
RITY=/tmp/.dockeroy86syee.xauth -v /tmp/.dockeroy86syee.xauth:/tmp/.dockeroy86sy
ee.xauth -v /tmp/.X11-unix:/tmp/.X11-unix -v /etc/localtime:/etc/localtime:r
o ffc605bec643
root@2a2844441ebd:/sim_ws#

```

지금 이 상태가 **Docker** 컨테이너 내부 쉘로 들어온 것.

# 1) ROS 2 Foxy 환경 설정

`source /opt/ros/foxy/setup.bash`

# 2) 워크스페이스 루트로 이동 (이미 /sim\_ws 에 있으시면 생략 가능)

`cd /sim_ws`

# 3) 빌드 (최초 한 번만)

`colcon build --symlink-install`

# 4) 빌드 결과 소싱

`source install/local_setup.bash`

# 5) 시뮬레이션 실행

`ros2 launch f1tenth_gym_ros gym_bridge_launch.py`

#####  
 근데 **Docker/rocker** 방법으로 하면 자꾸 에러가 남

```
[rviz2-1] [ERROR] [1753342739.293014015] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293019877] [rviz2]: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293022253] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293038634] [rviz2]: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293050252] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293067580] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293083021] [rviz2]: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293095308] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293107729] [rviz2]: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293114375] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293126350] [rviz2]: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293137803] [rviz2]: rviz::RenderSystem: error creating render window: InvalidParametersException: Window with name 'OgreRenderWindow' already exists in GLRenderSystem::createRenderWindow at /tmp/binarydeb/ros-foxy-rviz-ogre-vendor-8.2.8/obj-x86_64-linux-gnu/ogre-v1.12.1-prefix/src/ogre-v1.12.1/RenderSystems/GL/src/OgreGLRenderSystem.cpp (line 1061)
[rviz2-1] [ERROR] [1753342739.293137803] [rviz2]: Unable to create the rendering window after 100 tries
[rviz2-1] terminate called after throwing an instance of 'std::runtime_error'
```

그래서 그냥 Native 에서 실행했음.  
(Docker, rocker 에 대한 공부, 학습 필요)

#####

```
frame - frame does not exist
[rviz2-1] at line 133 in /tmp/binarydeb/ros-foxy-tf2-0.13.14/src/buffer
core.cpp
[rviz2-1] Warning: Invalid frame ID "map" passed to canTransform argument target
frame - frame does not exist
[rviz2-1] at line 133 in /tmp/binarydeb/ros-foxy-tf2-0.13.14/src/buffer
core.cpp
[rviz2-1] Warning: Invalid frame ID "map" passed to canTransform argument target
frame - frame does not exist
[rviz2-1] at line 133 in /tmp/binarydeb/ros-foxy-tf2-0.13.14/src/buffer
core.cpp
[rviz2-1] Warning: Invalid frame ID "map" passed to canTransform argument target
frame - frame does not exist
[rviz2-1] at line 133 in /tmp/binarydeb/ros-foxy-tf2-0.13.14/src/buffer
core.cpp
```

**warning 의미:** map 프레임이 아직 브리지나 Gazebo(시물)가 퍼블리시하지 않아서 RViz 쪽에서 “map → odom” TF 가 없다고 경고를 뿌리고 있는 것

#해결

다른 터미널에서 입력

# (컨테이너 안이라면 컨테이너 내부, native 라면 호스트에서)  
source /opt/ros/foxy/setup.bash

# map → odom 을 0,0,0 위치와 0 회전으로 고정 퍼블리시  
ros2 run tf2\_ros static\_transform\_publisher 0 0 0 0 0 0 map odom

#####

Rviz2 화면이 커지면

키보드 텔레옴으로 수동 운전을 해보기 위해서 . 과정일 뿐 총 정리는 아래에 있

#####

새로운 터미널을 열고



```
source /opt/ros/foxy/setup.bash
source ~/sim_ws/install/local_setup.bash # 컨테이너 안이면 /sim_ws, native 면 ~/sim_ws
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

```
# 운전 키
I : 전진
, : 후진
u : 전진+좌회전
o : 전진+우회전
m : 후진+좌회전
. : 후진+우회전
k : 정지
```

#저 명령어를 터미널에서 입력해야 rviz2 에서 움직임. rviz2 에서 입력하는게 아님

```
#####
터미널을 다시 열었을 때
#####
```

## #터미널 1

```
# 1) ROS 2 Foxy 환경 소싱
source /opt/ros/foxy/setup.bash

# 2) 워크스페이스 소싱
source ~/sim_ws/install/local_setup.bash

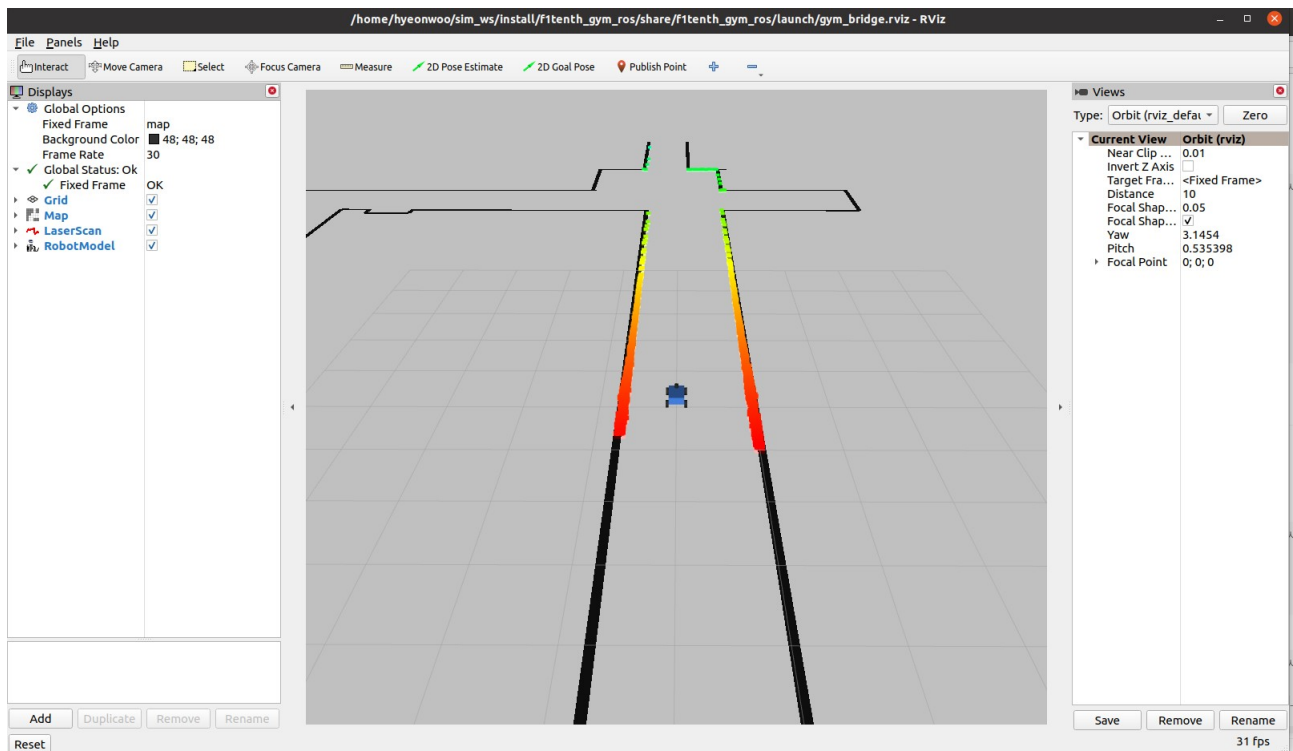
# 3) 시뮬레이션 브리지 + RViz 실행
ros2 launch f1tenth_gym_ros gym_bridge_launch.py

# 4) (옵션) map → odom static TF 퍼블리시
#   RViz 가 map 프레임을 못 찾는 에러를 막기 위해, 아니면 rviz2 에서 odom 으로 변경하기,
#   다른터미널에서 실행
ros2 run tf2_ros static_transform_publisher 0 0 0 0 0 0 map odom
```

## #터미널 2

```
source /opt/ros/foxy/setup.bash
source ~/sim_ws/install/local_setup.bash

ros2 run teleop_twist_keyboard teleop_twist_keyboard
```



## PID 제어, Ackermann 조향

**PID** = PID 는 Proportional(비례), Integral(적분), Derivative(미분) 세 가지 요소를 결합한 제어 알고리즘

**비례(P):** 현재 오차(error)에 비례한 출력

- **적분(I):** 과거 오차의 누적값에 비례한 출력
- **미분(D):** 오차 변화율(현재 오차 - 이전 오차)에 비례한 출력
- 수식

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

- $e(t)$  = 목표값 - 현재 측정값
- $K_p, K_i, K_d$  = 각각 비례·적분·미분 계수

### 속도 제어에의 응용

- 목표 속도와 실제 속도 간 오차를 계산
- PID 연산을 통해 ‘가속도(또는 제동) 명령’을 계산
- 예) 속도 오차가 클수록 급가속; 오차가 누적되면 지속 가속; 오차가 급변하면 제동

### 아커만 조향(Ackermann Steering)

#### 1. 정의

차량이 회전할 때, 앞바퀴 두 개가 서로 다른 회전 반경을 갖도록 설계된 조향 기구(geometry)입니다.

#### 2. 원리

- 차량이 곡선을 그리며 돌 때, 바깥쪽 바퀴는 더 큰 반경을, 안쪽 바퀴는 더 작은 반경을 돌아야 바퀴가 미끄러지지 않습니다.
- 이를 만족하도록 핸들 각도와 각각의 앞바퀴 조향각을 기하학적으로 계산합니다.

```
#####
결국 목표는 시뮬레이션 차량의 속도와 회전 민감도를 조정하는 것 같음.
#####
```

## 1. 패키지 생성

```
cd ~/sim_ws/src
```

```
# ament_python 패키지 생성 (의존성 자동 기입)
```

```
ros2 pkg create pid_controller \
  --build-type ament_python \
  --dependencies rclpy ackermann_msgs geometry_msgs nav_msgs
```

## 2. 경로 들어가서 pid\_speed\_control.py 파일 수정하기

```
cd ~/sim_ws/src/pid_controller/pid_controller
```

```
cat > pid_speed_control.py << 'EOF'
#!/usr/bin/env python3
"""
```

```
ROS2 노드: PID 속도 제어 및 Ackermann 조향 컨트롤러
```

사용법:

```
ros2 run pid_controller pid_speed_control \
  --ros-args -p kp:=1.0 -p ki:=0.1 -p kd:=0.01 -p wheelbase:=0.325
```

구독:

```
/cmd_vel          # geometry_msgs/Twist (목표 속도/회전속도)
/ego_racecar/odom  # nav_msgs/Odometry (실제 속도 피드백)
```

발행:

```
/drive            # ackermann_msgs/AckermannDriveStamped (제어 출력)
```

매개변수:

```
kp, ki, kd        # PID 계수
wheelbase          # 차량 휠베이스 (m)
```

```
"""
```

```
import math
import rclpy
```

```

from rclpy.node import Node
from ackermann_msgs.msg import AckermannDriveStamped
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry

class PIDController(Node):
    def __init__(self):
        super().__init__('pid_speed_controller')
        # 파라미터 선언
        self.kp = self.declare_parameter('kp', 1.0).value
        self.ki = self.declare_parameter('ki', 0.0).value
        self.kd = self.declare_parameter('kd', 0.0).value
        self.wheelbase = self.declare_parameter('wheelbase', 0.325).value

        # 내부 변수 초기화
        self.integral = 0.0
        self.prev_error = 0.0
        self.prev_time = self.get_clock().now()
        self.current_speed = 0.0
        self.target_speed = 0.0
        self.target_steering = 0.0

        # 구독/발행 설정
        self.create_subscription(Twist, 'cmd_vel', self.cmd_vel_callback, 10)
        self.create_subscription(Odometry, '/ego_racecar/odom', self.odom_callback, 10)
        self.pub = self.create_publisher(AckermannDriveStamped, 'drive', 10)
        self.create_timer(0.02, self.control_loop) # 50 Hz 제어 주기

    def cmd_vel_callback(self, msg: Twist):
        # 목표 속도/회전속도 설정
        self.target_speed = msg.linear.x
        # steering angle 계산
        if abs(msg.linear.x) > 0.1:
            self.target_steering = math.atan(self.wheelbase * msg.angular.z / msg.linear.x)
        else:
            self.target_steering = 0.0

    def odom_callback(self, msg: Odometry):
        # 실제 속도 계산 (xy 합)
        vx = msg.twist.twist.linear.x
        vy = msg.twist.twist.linear.y
        self.current_speed = math.hypot(vx, vy)

    def control_loop(self):
        now = self.get_clock().now()
        dt = (now - self.prev_time).nanoseconds * 1e-9
        error = self.target_speed - self.current_speed
        self.integral += error * dt
        derivative = (error - self.prev_error) / dt if dt > 0 else 0.0

```

```

# PID 연산
output = self.kp * error + self.ki * self.integral + self.kd * derivative
self.prev_error = error
self.prev_time = now

# Ackermann 메시지 생성 및 발행
drive_msg = AckermannDriveStamped()
drive_msg.drive.speed = float(output)
drive_msg.drive.steering_angle = float(self.target_steering)
self.pub.publish(drive_msg)

def main(args=None):
    rclpy.init(args=args)
    node = PIDController()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
EOF

```

**chmod +x pid\_speed\_control.py**

### 3. 워크 스페이스 루트로 돌아가서 빌드하고 실행 해보기

**#빌드**

```

cd ~/sim_ws
colcon build --symlink-install
source install/local_setup.bash

```

### 4. 실행

**#터미널 1**

```

# 1) ROS 2 Foxy 환경 소싱
source /opt/ros/foxy/setup.bash

```

```

# 2) 워크스페이스 소싱
source ~/sim_ws/install/local_setup.bash

```

```

# 3) 시뮬레이션 브리지 + RViz 실행

```

```
ros2 launch f1tenth_gym_ros gym_bridge_launch.py
```

# 4) (옵션) map → odom static TF 퍼블리시

# RViz 가 map 프레임을 못 찾는 에러를 막기 위해, 에러나면 odom 으로 변경하면됨, 혹은 다른 터미널에

```
ros2 run tf2_ros static_transform_publisher 0 0 0 0 0 map odom
```

#터미널 2

# PID 제어기 실행하는 터미널

```
source /opt/ros/foxy/setup.bash
```

```
source ~/sim_ws/install/local_setup.bash
```

```
ros2 run pid_controller pid_speed_control \
```

```
--ros-args -p kp:=1.0 -p ki:=0.1 -p kd:=0.01 -p wheelbase:=0.325
```

#혹시 pid\_controller 를 찾지 못했다고 나오면

```
cd ~/sim_ws
```

```
colcon build --symlink-install
```

# 빌드가 끝나면

```
source install/local_setup.bash
```

입력해서 워크스페이스를 빌드, 소싱해야함  
다시 실행.

혹시 “No executable found” 에러가 나오면

```
nano ~/sim_ws/src/pid_controller/setup.cfg
```

열었을 때 이 섹션이 반드시 나와야함.

```
[options]
```

```
packages = find:
```

```
[options.entry_points]
```

```
console_scripts =
```

```
pid_speed_control = pid_controller.pid_speed_control:main
```

없으면 추가후 저장

다시 패키지 빌드, 소싱

```
cd ~/sim_ws
```

```
# pid_controller 만 재빌드
colcon build --packages-select pid_controller --symlink-install
```

```
# 반드시 소싱
source install/local_setup.bash
```

혹시 그래도 “No executable found” 에러가 나오면 ‘entry point’ 설정 잘못된 것

과정 따라하기

```
nano ~/sim_ws/src/pid_controller/setup.py
```

**entry\_points** 섹션에 **pid\_speed\_control** 추가해야함. 아래 사진 부분에 #이곳에 추가합니다에 사진대로 입력

python

복사 편집

```
setup(
    name='pid_controller',
    version='0.0.0',
    packages=['pid_controller'],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='hyeonwoo',
    maintainer_email='ohw2509@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            # 이곳에 추가합니다
        ],
    },
)
```

```
hyeonwoo@hyeonwoo-950XDA: ~/sim_ws 38x23
...rc/pid_controller/setup.py Modified
data_files=[
    ('share/ament_index/resource_>
    ['resource/' + package_na>
    ('share/' + package_name, ['p>
],
install_requires=['setuptools'],
zip_safe=True,
maintainer='hyeonwoo',
maintainer_email='ohw2509@gmail.c>
description='TODO: Package descri>
license='TODO: License declaratio>
tests_require=['pytest'],
entry_points={
    'console_scripts': [
        'pid_speed_control = pid_v>
    ],
},
)

^G Get Help ^O Write Out ^W Where Is
^X Exit ^R Read File ^\ Replace
```



#이렇게 됨

```
entry_points={
    'console_scripts': [
        'pid_speed_control = pid_controller.pid_speed_control:main',
    ],
},
```

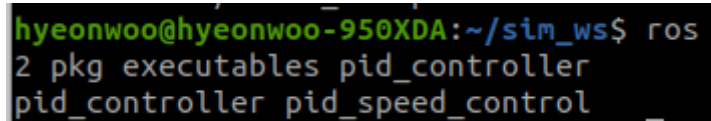
변경 후 다시 빌드, 소싱

```
cd ~/sim_ws
colcon build --packages-select pid_controller --symlink-install
source install/local_setup.bash
```

#실행 파일 목록 확인

```
ros2 pkg executables pid_controller
```

잘 보임



```
hyeonwoo@hyeonwoo-950XDA:~/sim_ws$ ros2
2 pkg executables pid_controller
pid_controller pid_speed_control
```

이제 다시 PID 제어기 실행

```
ros2 run pid_controller pid_speed_control \
--ros-args \
-p kp:=1.0 \
-p ki:=0.1 \
-p kd:=0.01 \
-p wheelbase:=0.325
```

#터미널 3

```
source /opt/ros/foxy/setup.bash
source ~/sim_ws/install/local_setup.bash
```

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

```
#####
설정이 완료된 상태에서 터미널 여는 것
#####
```

## 터미널 1: 시뮬레이션 + RViz

```
bash
복사편집
# 1) ROS 2 환경
source /opt/ros/foxy/setup.bash

# 2) 내 워크스페이스 환경
source ~/sim_ws/install/local_setup.bash

# 3) 시뮬레이션 브리지 + RViz 실행
ros2 launch f1tenth_gym_ros gym_bridge_launch.py

# 4) (선택) map→odom Static TF
#   - RViz "map" 프레임 오류 방지용입니다.

# map 오류 발생 시 터미널 하나 더 열어서
# (컨테이너 안이라면 컨테이너 내부, native 라면 호스트에서)
source /opt/ros/foxy/setup.bash

# map → odom 을 0,0,0 위치와 0 회전으로 고정 퍼블리시
ros2 run tf2_ros static_transform_publisher 0 0 0 0 0 0 map odom
```

## 터미널 2: PID 속도 제어기 실행

```
bash
복사편집
# 1) ROS 2 환경
source /opt/ros/foxy/setup.bash

# 2) 워크스페이스 환경
source ~/sim_ws/install/local_setup.bash

# 3) PID 노드 실행
ros2 run pid_controller pid_speed_control \
  --ros-args \
  -p kp:=1.0 \
  -p ki:=0.1 \
  -p kd:=0.01 \
  -p wheelbase:=0.325
```

## 터미널 3: 키보드 텔레오프 (수동 조작)

bash

복사편집

# 1) ROS 2 환경

source /opt/ros/foxy/setup.bash

# 2) 워크스페이스 환경

source ~/sim\_ws/install/local\_setup.bash

# 3) Teleop 노드 실행

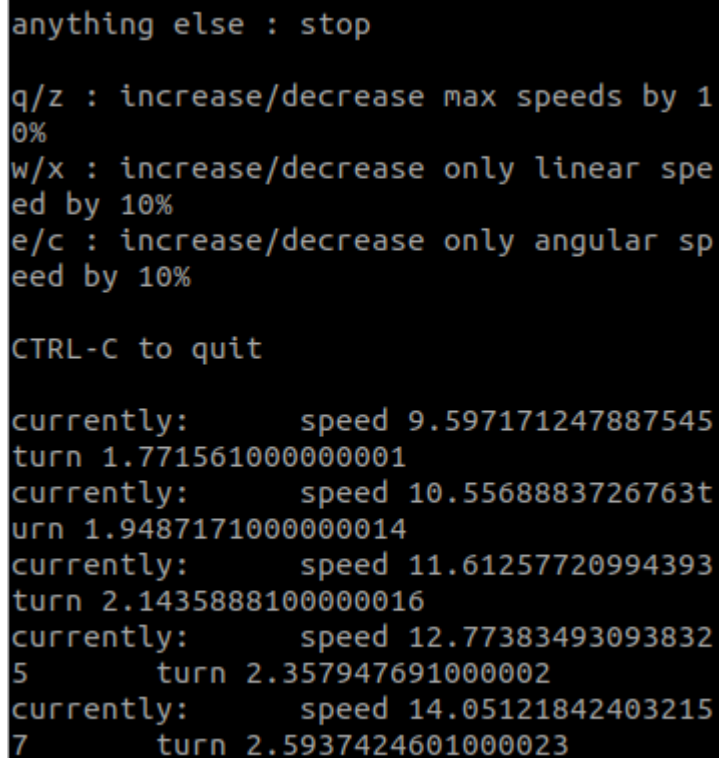
ros2 run teleop\_twist\_keyboard teleop\_twist\_keyboard

# 번외) 혹시 전진속도와 회전속도를 바꾸고 싶으면

터미널에서 q, w, e 를 적절히 누르기

z,x,c

입력



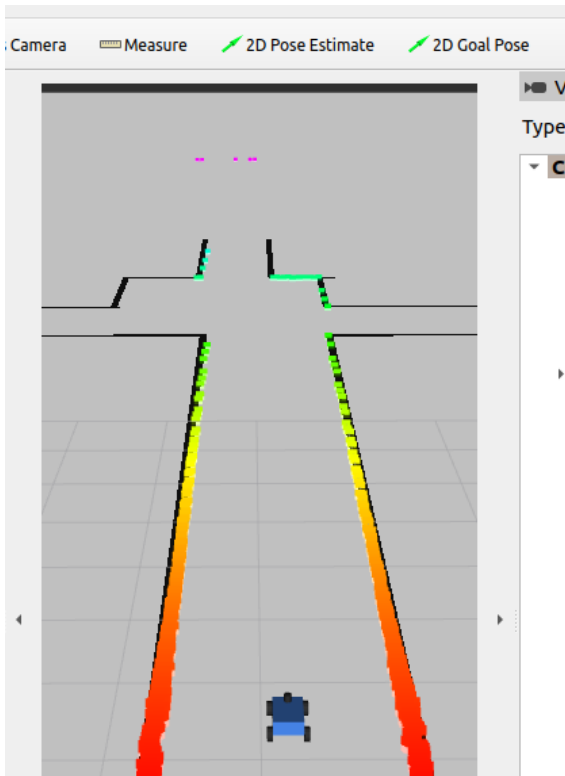
```
anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 9.597171247887545
turn 1.771561000000001
currently:      speed 10.5568883726763t
urn 1.9487171000000014
currently:      speed 11.61257720994393
turn 2.1435888100000016
currently:      speed 12.77383493093832
5      turn 2.357947691000002
currently:      speed 14.05121842403215
7      turn 2.5937424601000023
```

+ 혹시 차가 멀리 가버리면 rviz2 에서 2D pose estimate 누르고 원하는 곳 누르면 차가 다시 생김



```
#####
PID, Ackermann 코드 / home/sim_ws/src/pid_controller/pid_controller/pid_speed_control.py
에 입력했던 것
#####
```

```
#!/usr/bin/env python3
"""
```

ROS2 노드: PID 속도 제어 및 Ackermann 조향 컨트롤러

사용법:

```
ros2 run pid_controller pid_speed_control \
--ros-args -p kp:=1.0 -p ki:=0.1 -p kd:=0.01 -p wheelbase:=0.325
```

구독:

```
/cmd_vel          # geometry_msgs/Twist (목표 속도/회전속도)
/ego_racecar/odom  # nav_msgs/Odometry (실제 속도 피드백)
```

발행:

```
/drive            # ackermann_msgs/AckermannDriveStamped (제어 출력)
```

매개변수:

```

kp, ki, kd    # PID 계수
wheelbase    # 차량 휠베이스 (m)
"""

import math
import rclpy
from rclpy.node import Node
from ackermann_msgs.msg import AckermannDriveStamped
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry

class PIDController(Node):
    def __init__(self):
        super().__init__('pid_speed_controller')
        # 파라미터 선언
        self.kp = self.declare_parameter('kp', 1.0).value
        self.ki = self.declare_parameter('ki', 0.0).value
        self.kd = self.declare_parameter('kd', 0.0).value
        self.wheelbase = self.declare_parameter('wheelbase', 0.325).value

        # 내부 변수 초기화
        self.integral = 0.0
        self.prev_error = 0.0
        self.prev_time = self.get_clock().now()
        self.current_speed = 0.0
        self.target_speed = 0.0
        self.target_steering = 0.0

        # 구독/발행 설정
        self.create_subscription(Twist, 'cmd_vel', self.cmd_vel_callback, 10)
        self.create_subscription(Odometry, '/ego_racecar/odom', self.odom_callback, 10)
        self.pub = self.create_publisher(AckermannDriveStamped, 'drive', 10)
        self.create_timer(0.02, self.control_loop) # 50 Hz 제어 주기

    def cmd_vel_callback(self, msg: Twist):
        # 목표 속도/회전속도 설정
        self.target_speed = msg.linear.x
        # steering angle 계산
        if abs(msg.linear.x) > 0.1:
            self.target_steering = math.atan(self.wheelbase * msg.angular.z / msg.linear.x)
        else:
            self.target_steering = 0.0

    def odom_callback(self, msg: Odometry):
        # 실제 속도 계산 (xy 합)
        vx = msg.twist.twist.linear.x
        vy = msg.twist.twist.linear.y
        self.current_speed = math.hypot(vx, vy)

    def control_loop(self):

```

```

now = self.get_clock().now()
dt = (now - self.prev_time).nanoseconds * 1e-9
error = self.target_speed - self.current_speed
self.integral += error * dt
derivative = (error - self.prev_error) / dt if dt > 0 else 0.0

# PID 연산
output = self.kp * error + self.ki * self.integral + self.kd * derivative
self.prev_error = error
self.prev_time = now

# Ackermann 메시지 생성 및 발행
drive_msg = AckermannDriveStamped()
drive_msg.drive.speed = float(output)
drive_msg.drive.steering_angle = float(self.target_steering)
self.pub.publish(drive_msg)

```

```

def main(args=None):
    rclpy.init(args=args)
    node = PIDController()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

#####

## 설명 / feat. Chat GPT

#####

## 1. импорт 및 노드 설명

```

python
복사편집
#!/usr/bin/env python3
"""
ROS2 노드: PID 속도 제어 및 Ackermann 조향 컨트롤러
"""
import math
import rclpy
from rclpy.node import Node
from ackermann_msgs.msg import AckermannDriveStamped
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry

```

- `#!/usr/bin/env python3`  
스크립트가 실행될 때 **python3** 인터프리터로 해석되도록 하는 shebang 라인입니다.

- `import ...`
    - `math: atan, hypot` 같은 수학 함수 사용
    - `rclpy.Node`: ROS 2 파이썬 클라이언트 라이브러리
    - 메시지 타입들(`AckermannDriveStamped`, `Twist`, `Odometry`)을 불러옵니다.
- 

## 2. 클래스 정의 및 파라미터 선언

python

복사편집

```
class PIDController(Node):
    def __init__(self):
        super().__init__('pid_speed_controller')
        # 파라미터 선언
        self.kp = self.declare_parameter('kp', 1.0).value
        self.ki = self.declare_parameter('ki', 0.0).value
        self.kd = self.declare_parameter('kd', 0.0).value
        self.wheelbase = self.declare_parameter('wheelbase', 0.325).value
```

- `class PIDController(Node):`  
ROS 2 노드를 정의하는 클래스. `Node` 를 상속받아 기능을 구현합니다.
  - `super().__init__('pid_speed_controller')`  
노드 이름을 `pid_speed_controller` 로 초기화합니다.
  - `declare_parameter(name, default)`
    - `kp, ki, kd`: PID 제어기 비례·적분·미분 계수
    - `wheelbase`: 차량 앞뒤 바퀴 사이 거리 (m)  
사용자 런치 파일이나 커맨드라인에서 `-p kp:=2.0` 식으로 값을 변경할 수 있습니다.
- 

## 3. 내부 변수 및 구독/발행 설정

python

복사편집

```
# 내부 변수 초기화
self.integral = 0.0
self.prev_error = 0.0
self.prev_time = self.get_clock().now()
self.current_speed = 0.0
self.target_speed = 0.0
self.target_steering = 0.0

# 구독/발행 설정
self.create_subscription(Twist, 'cmd_vel', self.cmd_vel_callback, 10)
self.create_subscription(Odometry, '/ego_racecar/odom',
self.odom_callback, 10)
```

```
self.pub = self.create_publisher(AckermannDriveStamped, 'drive', 10)
self.create_timer(0.02, self.control_loop) # 50 Hz 제어 주기
```

- 내부 변수
    - `integral`: I(적분) 성분 누적값
    - `prev_error`: 이전 제어 오차(e) 저장
    - `prev_time`: 이전 제어 사이클 시각
    - `current_speed`: 실제 속도
    - `target_speed`: 목표 속도
    - `target_steering`: 목표 조향 각도
  - 구독
    - `cmd_vel` (Twist): 외부에서 보낸 목표 속도·회전속도
    - `/ego_racecar/odom` (Odometry): 시뮬레이션이 반환하는 실제 속도 피드백
  - 발행
    - `drive` (AckermannDriveStamped): PID/Ackermann 계산 결과를 시뮬레이터로 전송
  - 타이머
    - 0.02 초(50Hz) 주기로 `control_loop()` 호출
- 

## 4. 콜백 함수

### 4.1 `cmd_vel_callback`

python

복사편집

```
def cmd_vel_callback(self, msg: Twist):
    self.target_speed = msg.linear.x
    if abs(msg.linear.x) > 0.1:
        self.target_steering = math.atan(self.wheelbase * msg.angular.z /
msg.linear.x)
    else:
        self.target_steering = 0.0
```

- `/cmd_vel` 메시지가 들어올 때마다 호출
- `msg.linear.x` → 목표 속도(`target_speed`)로 저장
- `msg.angular.z` 와 휠베이스를 이용해 Ackermann 조향각을 계산



$$\theta = \arctan(vL \cdot \omega)$$

$v$ 가 너무 작으면(0.1 이하) 전복 방지를 위해 0으로 처리합니다.

## 4.2 odom\_callback

python

복사편집

```
def odom_callback(self, msg: Odometry):
    vx = msg.twist.twist.linear.x
    vy = msg.twist.twist.linear.y
    self.current_speed = math.hypot(vx, vy)
```

- /ego\_racecar/odom 메시지가 들어올 때마다 호출
- x·y 성분 속도를 합성해 실제 주행 속도(current\_speed)를 계산

## 5. 제어 루프 (control\_loop)

python

복사편집

```
def control_loop(self):
    now = self.get_clock().now()
    dt = (now - self.prev_time).nanoseconds * 1e-9
    error = self.target_speed - self.current_speed
    self.integral += error * dt
    derivative = (error - self.prev_error) / dt if dt > 0 else 0.0

    output = self.kp * error + self.ki * self.integral + self.kd *
derivative
    self.prev_error = error
    self.prev_time = now

    drive_msg = AckermannDriveStamped()
    drive_msg.drive.speed = float(output)
    drive_msg.drive.steering_angle = float(self.target_steering)
    self.pub.publish(drive_msg)
```

1. 시간 차  $dt$  계산
2. 오차  $error = \text{목표속도} - \text{실제속도}$
3. 적분(integral)에  $error \cdot dt$  누적
4. 미분(derivative) =  $(error - \text{prev\_error}) / dt$
5. PID 공식을 적용
 
$$u = K_p e + K_i (\text{적분}) + K_d (\text{미분})$$
6. 결과  $u$ 를 /drive.speed에 세팅
7. 콜백에서 계산된 target\_steering을 /drive.steering\_angle에 세팅

## 6. 노드 엔트리포인트

python

복사편집

```
def main(args=None):
    rclpy.init(args=args)
    node = PIDController()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- **rclpy.init()** : ROS2 통신 초기화
  - **PIDController()** 인스턴스 생성 → 구독/타이머 시작
  - **rclpy.spin(node)** : 노드가 종료될 때까지 콜백 대기
  - **rclpy.shutdown()** : 통신 정리
- 

이렇게 1) 사용자로부터 목표 속도·회전속도를 받고,

2) **PID** 로 속도를 보정하며,

3) **Ackermann** 수식으로 조향각을 계산하여

4) **/drive** 토픽으로 시뮬레이터에 “어떻게 움직이라” 신호를 보내는 구조입니다.

각 블록을 이해하시면,

- PID 계수(kp, ki, kd)나
- 휠베이스(wheelbase),
- 심지어 조향 배율(steering\_gain 추가) 등 원하는 대로 쉽게 튜닝할 수 있습니다.

요소	값	설명
<b>teleop_twist_keyboard</b> — 최대 목표 속도(scale)	선형 속도 <code>linear_scale = 0.5</code> 회전 속도 <code>angular_scale = 1.0</code>	i/, 키 누를 때 최대 0.5 m/s 전진·후진, u/o/m/. 키로 1.0 rad/s 회전 목표를 보냅니다.
<b>PID 제어 계수</b> — 속도 제어	<code>kp = 1.0</code> <code>ki = 0.1</code> <code>kd = 0.01</code>	- <b>Kp=1.0</b> : 오차에 비례하여 즉시 반응 - <b>Ki=0.1</b> : 누적 오차 보정으로 정속 유지 - <b>Kd=0.01</b> : 급격한 오차 변화를 억제하여 부드럽게 멈춤
<b>휠베이스 (Ackermann)</b>	<code>wheelbase = 0.325 m</code>	앞뒤 바퀴 사이의 길이로, 조향 각도 계산식 $\theta = \arctan \frac{L\omega}{v}$ 에 사용됩니다.
<b>steering_gain</b> (추가 파라미터)	<code>1.0</code> (기본값)	조향각 계산값에 곱해 주는 배율. 기본 1배, 높이면 더 민감한 회전이 가능합니다.
<b>제어 주기</b>	<code>0.02 s</code> (50 Hz)	매 20 ms 마다 속도 오차를 읽고 PID 연산 → <code>/drive</code> 발행

## 이 설정이 실제 동작에서 주는 느낌

### 1. 속도 응답

- 목표 속도를 0.5 m/s 로 주었을 때,
- $K_p=1.0$  이므로 약  $1 \text{ m/s}^2$  에 가까운 가속도로 빠르게 그 값에 다가가고,
- $K_i=0.1$  이므로 작은 잔류 오차는 시간이 지날수록 채워 줍니다.
- $K_d=0.01$  으로 과도한 진동(overshoot)을 억제해 주행이 비교적 부드럽습니다.

### 2. 조향 민감도

- 예를 들어 `i` 키만 눌렀을 때 직진,
- `u/o` 키로 동일하게 회전 명령(`angular.z = ±1.0`)을 줄 경우
- $\theta = \text{atan}(0.325 \times 1.0/0.5) \approx 0.58 \text{ rad}$  (약  $33^\circ$ ) 회전각이 나옵니다.
- `steering_gain` 을 높이지 않았다면, 이 정도 회전이 기본 축입니다.

### 3. 전체 주행 느낌

- 부드럽게 가속·제동하면서
- 자연스러운 자동차식 커브 주행이 가능
- 너무 급하지도, 너무 느리지도 않은 중간 정도의 반응 속도

## 튜닝 포인트

- 더 빠른 반응
  - `Kp`를 올려보세요 (예: 1.5~2.0).
  - 너무 높으면 진동이 생길 수 있으니 채터링(dither) 확인 필요.
- 잔류 오차 최소화
  - `Ki`를 올리면 목표 속도에 더 정확히 수렴하지만,
  - 너무 높으면 느린 정착 시간(settling time)과 진동을 유발할 수 있습니다.
- 더 민감한 회전
  - `steering_gain` 파라미터를 추가해 1.0 → 1.5~2.0 정도 올려 보세요.
- teleop 속도 변경
  - `teleop_twist_keyboard` 내부 변수를 수정하거나,
  - 실행 중 q/w/e 키로 실시간으로 조정할 수 있습니다.

이처럼 현재 설정은 **중간 정도의 반응성과 안정성**을 목표로 한 값들이며, 필요에 따라 위 포인트를 하나씩 조절하시면 더욱ダイナミック하거나, 반대로 더 매끄러운 주행으로 바꾸실 수 있습니다.