

## 4.24 스터디

주제: 로봇 패키지를 받아서 시뮬레이션 실행해보기

### Gazebo설치

```
sudo apt install ros-humble-gazebo-*
```

```
# 설치 확인
```

```
gazebo --version
```

### TurtleBot3 패키지 설치

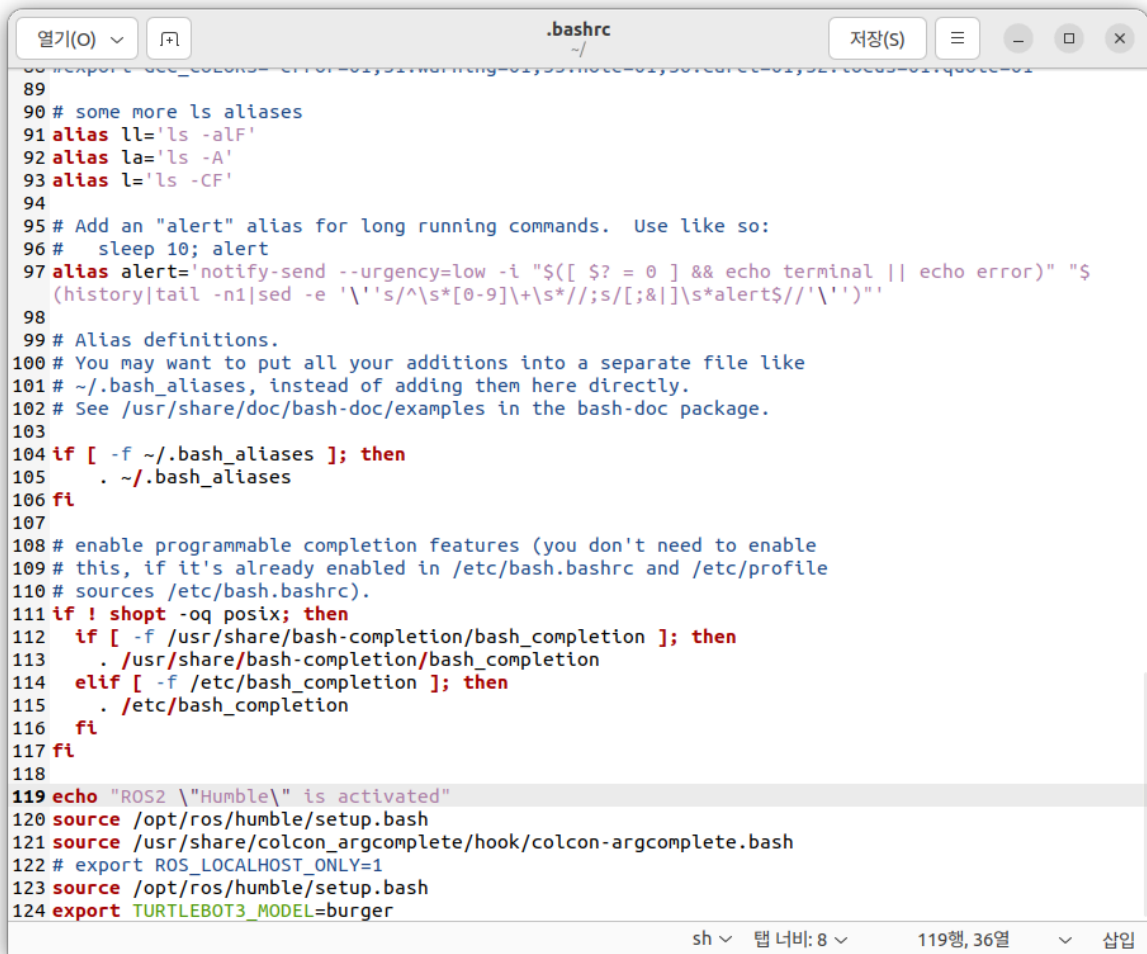
```
sudo apt install ros-humble-turtlebot3 ros-humble-turtlebot3-gazebo
```

```
# .bashrc에 "export TURTLEBOT3_MODEL=burger" 추가
```

```
# 환경변수 TURTLEBOT3_MODEL의 값으로 burger를 설정
```

```
echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
```

```
source ~/.bashrc
```



```
89
90 # some more ls aliases
91 alias ll='ls -alF'
92 alias la='ls -A'
93 alias l='ls -CF'
94
95 # Add an "alert" alias for long running commands. Use like so:
96 # sleep 10; alert
97 alias alert='notify-send --urgency=low -i "${[ $? = 0 ] && echo terminal || echo error}" "$
(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[\;]&]\s*alert$//'\`''"'
98
99 # Alias definitions.
100 # You may want to put all your additions into a separate file like
101 # ~/.bash_aliases, instead of adding them here directly.
102 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
103
104 if [ -f ~/.bash_aliases ]; then
105     . ~/.bash_aliases
106 fi
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112     if [ -f /usr/share/bash-completion/bash_completion ]; then
113         . /usr/share/bash-completion/bash_completion
114     elif [ -f /etc/bash_completion ]; then
115         . /etc/bash_completion
116     fi
117 fi
118
119 echo "ROS2 \"Humble\" is activated"
120 source /opt/ros/humble/setup.bash
121 source /usr/share/colcon_argcomplete/hook/colcon_argcomplete.bash
122 # export ROS_LOCALHOST_ONLY=1
123 source /opt/ros/humble/setup.bash
124 export TURTLEBOT3_MODEL=burger
```

## .bashrc 수정하는 이유?

TurtleBot3 패키지는 실행 시 `TURTLEBOT3_MODEL` 환경변수가 설정되어 있어야 함.

## teleop\_keyboard로 시뮬레이션

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

```
# teleop_keyboard
```

```
ros2 run turtlebot3_teleop teleop_keyboard
```

## 실행할 때 오류

[spawn\_entity.py-4] [ERROR] [spawn\_entity]: Service /spawn\_entity unavailable. Was Gazebo started with GazeboRosFactory?

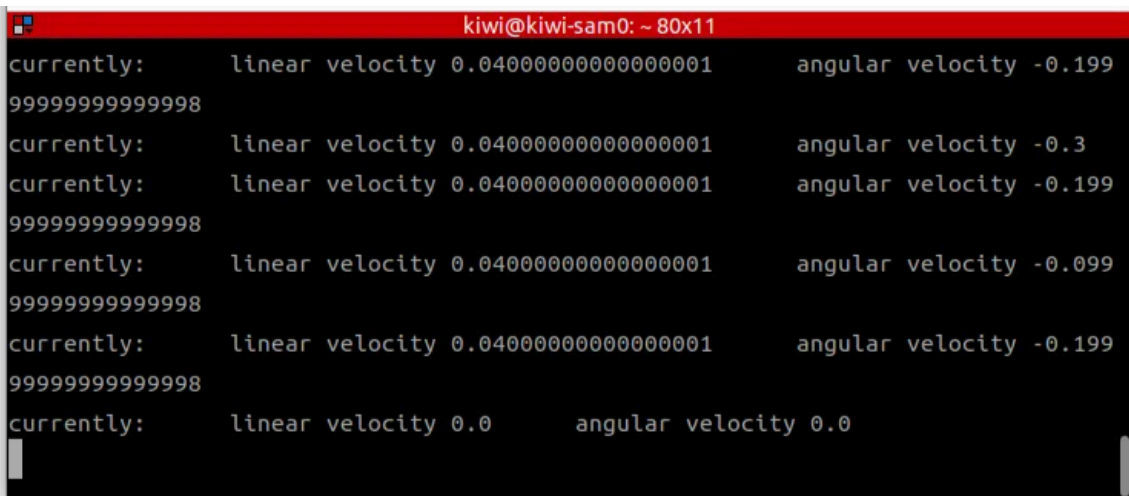
```
# ROS 2 Humble의 환경 변수 설정 (ros2 명령어, 패키지 경로 등 사용 가능하게 함)
source /opt/ros/humble/setup.bash
```

```
# Gazebo의 환경 변수 설정 (GAZEBO_PLUGIN_PATH, GAZEBO_MODEL_PATH 등
# ROS와 Gazebo가 연동될 수 있도록 Gazebo가 ROS 플러그인을 인식하게 해줌
source /usr/share/gazebo/setup.sh
```

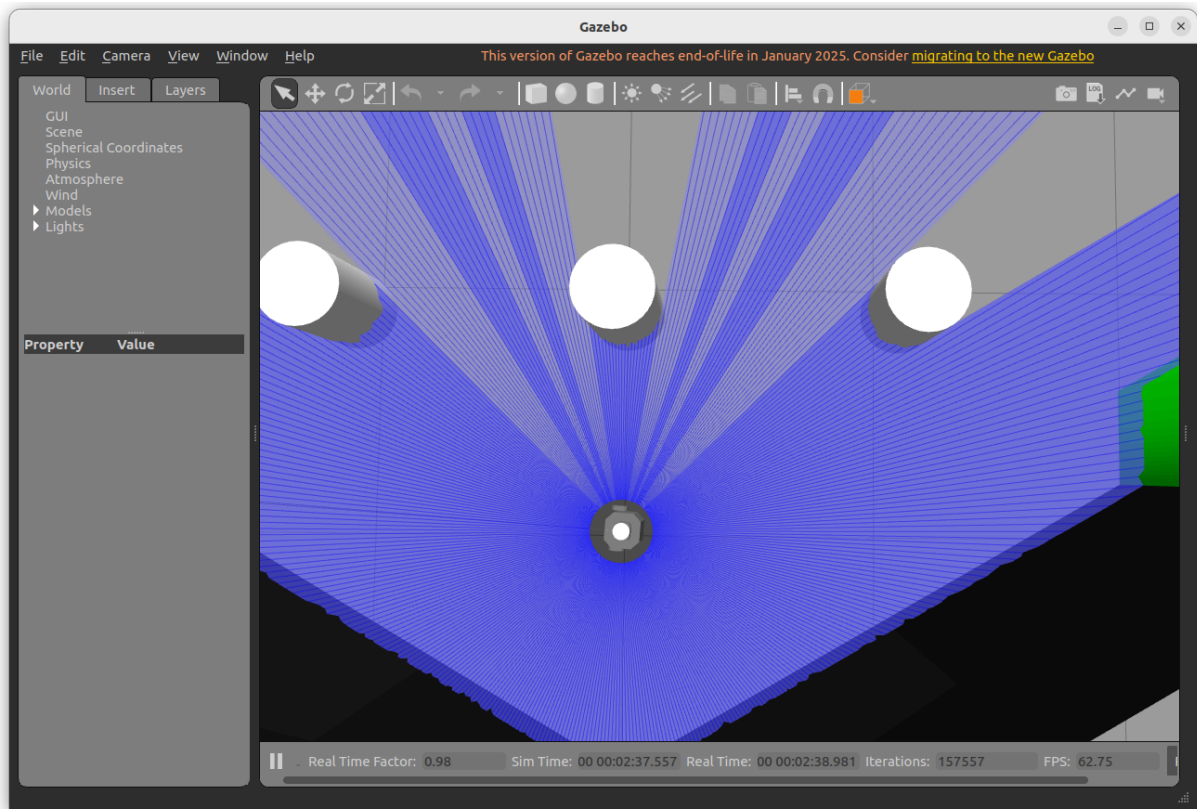
## teleop\_keyboard

linear velocity: 양수-전진, 음수-후진

angular velocity: 양수-반시계방향, 음수-시계방향

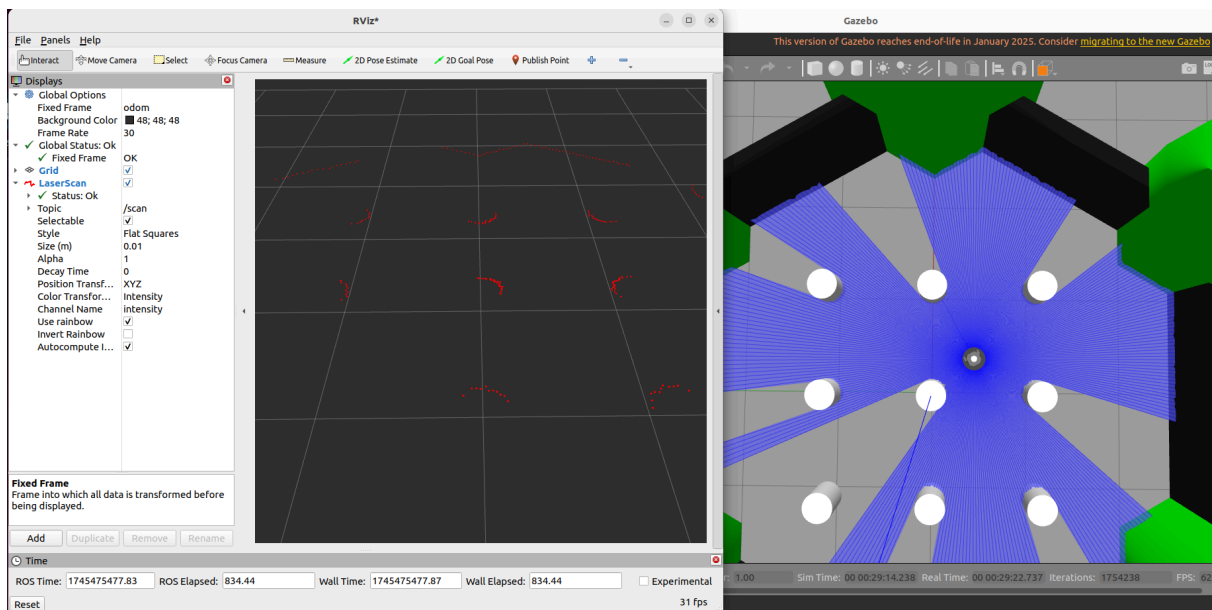
A terminal window with a red title bar showing the command 'kiwi@kiwi-sam0: ~ 80x11'. The output of the 'teleop\_keyboard' command is displayed, showing 'currently:' followed by 'linear velocity' and 'angular velocity' values. The values change over time, indicating movement. The final line shows 'linear velocity 0.0' and 'angular velocity 0.0', indicating the robot has stopped.

```
kiwi@kiwi-sam0: ~ 80x11
currently:      linear velocity 0.04000000000000001      angular velocity -0.199
99999999999998
currently:      linear velocity 0.04000000000000001      angular velocity -0.3
currently:      linear velocity 0.04000000000000001      angular velocity -0.199
99999999999998
currently:      linear velocity 0.04000000000000001      angular velocity -0.099
99999999999998
currently:      linear velocity 0.04000000000000001      angular velocity -0.199
99999999999998
currently:      linear velocity 0.0      angular velocity 0.0
```



## rviz2: LaserScan 데이터

로봇에 달린 Lidar 센서가 2D 평면(수평)으로 스캔한 결과



# Universal\_Robots 패키지

## UR5e: Universal Robots 협동 로봇



### UR5e

모든 생산 설비 보완

UR5e는 중급 애플리케이션을 공극의 유연성으로 처리하는 조정 가능한 산업용 경량 협동로봇입니다. UR5e는 다양한 애플리케이션에 완벽하게 통합할 수 있도록 설계되었습니다. 또한 UR5e는 OEM 로봇 시스템으로서 3위치 티치 펜던트와 함께 제공됩니다. CB3 모델이 필요하신가요? [여기서](#) 찾아보세요.

[견적서 요청](#)

## 패키지 설치

```
https://github.com/UniversalRobots/Universal_Robots_ROS2_Gazebo_Simulation
```

```
# Universal Robots Gazebo 시뮬레이션 패키지 클론
git clone https://github.com/UniversalRobots/Universal_Robots_ROS2_Gazebo

# ROS 2 의존성 설치
sudo apt update
rosdep update
rosdep install --from-paths src --ignore-src -r -y
```

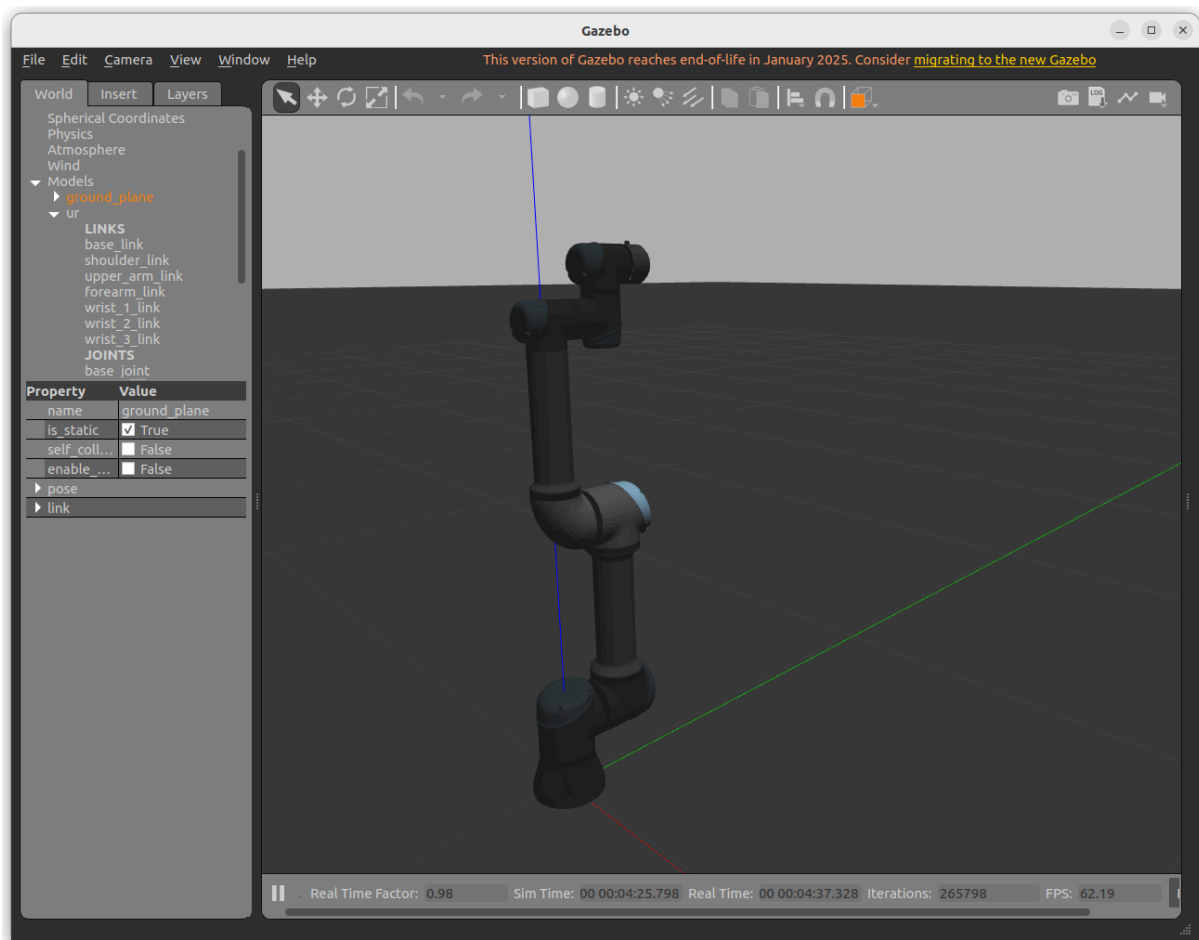
```
# 패키지 빌드
colcon build --symlink-install

# 현재 터미널에 환경 적용
source install/setup.bash
```

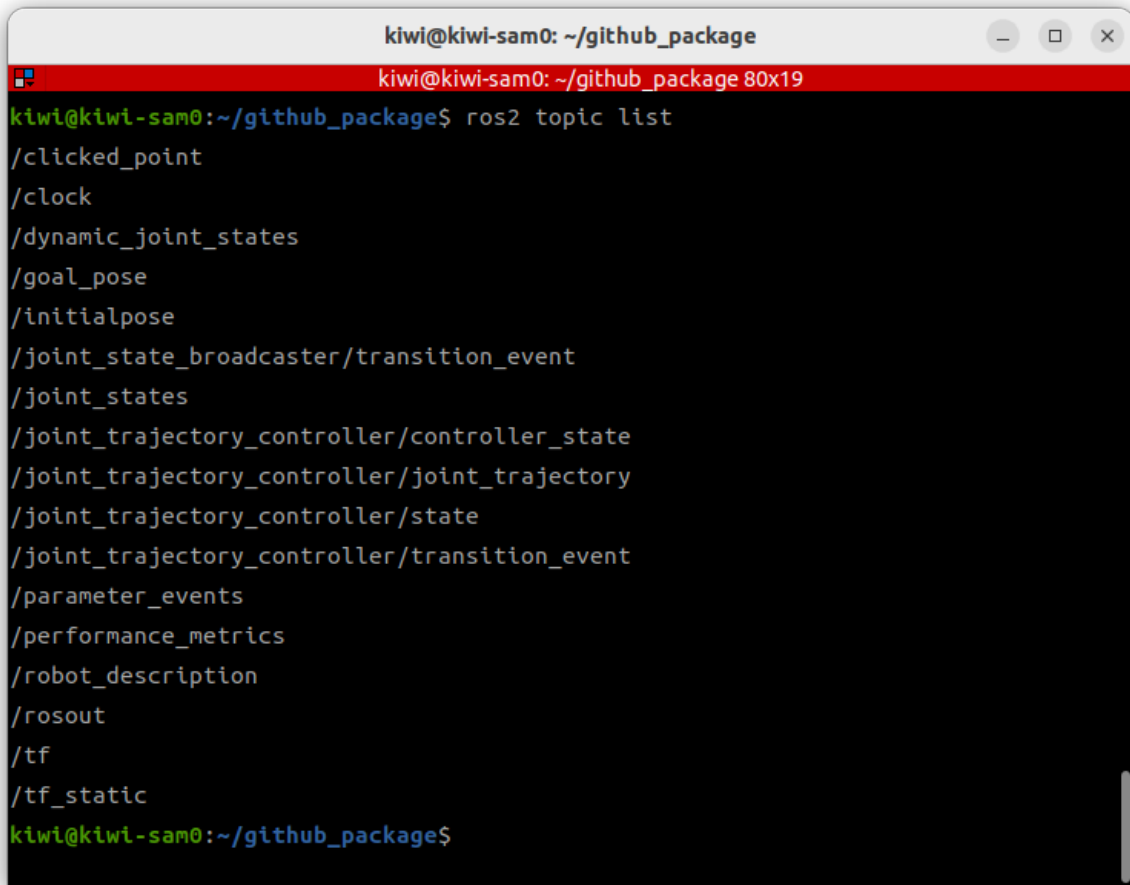
## 시뮬레이션 실행

```
#ros2 launch ur_simulation_gazebo ur_sim_control.launch.py
#ros2 launch ur_robot_driver test_joint_trajectory_controller.launch.py

ros2 launch ur_simulation_gazebo ur_sim_moveit.launch.py
```



## topic list



```
kiwi@kiwi-sam0: ~/github_package
kiwi@kiwi-sam0: ~/github_package 80x19
kiwi@kiwi-sam0:~/github_package$ ros2 topic list
/clicked_point
/clock
/dynamic_joint_states
/goal_pose
/initialpose
/joint_state_broadcaster/transition_event
/joint_states
/joint_trajectory_controller/controller_state
/joint_trajectory_controller/joint_trajectory
/joint_trajectory_controller/state
/joint_trajectory_controller/transition_event
/parameter_events
/performance_metrics
/robot_description
/rosout
/tf
/tf_static
kiwi@kiwi-sam0:~/github_package$
```

## 토픽 발행으로 제어

### 조인트 상태 확인

```
ros2 topic echo --once /joint_states
```

```
kiwi@kiwi-sam0: ~/github_package
kiwi@kiwi-sam0: ~/github_package 80x37
kiwi@kiwi-sam0:~/github_package$ ros2 topic echo --once /joint_states
A message was lost!!!
      total count change:1
      total count: 1---
header:
  stamp:
    sec: 1145
    nanosec: 769000000
  frame_id: ''
name:
- shoulder_pan_joint
- shoulder_lift_joint
- elbow_joint
- wrist_1_joint
- wrist_2_joint
- wrist_3_joint
position:
- 0.00017849618565612957
- -1.5699892902060701
- -1.2018376852829249e-05
- -1.5699781364435372
- -1.4428635540575385e-05
- -1.7172724881220347e-05
velocity:
- -0.0010832604573954286
- 0.0027761823264566923
- -0.0009615697595762801
- -0.0001486138515685337
- 0.001152285457204656
- 0.00034671509120090386
effort:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
---
```

## name

- 현재 로봇 URDF에 정의된 **관절 이름 목록**



- 순서와 위치는 `position[]`, `velocity[]`, `effort[]` 배열과 1:1 대응

## position

- 각 관절의 **현재 각도 (rad 단위)**

## velocity

- 각 관절의 **회전 속도 (rad/s)**
- 거의 0에 가깝다면 **움직이지 않는 상태**

## effort

- 각 관절에 작용한 **토크(힘)**
- 시뮬에서는 기본적으로 0.0이 일반적

## 토픽 발행해서 shoulder\_pan\_joint 각도 수정

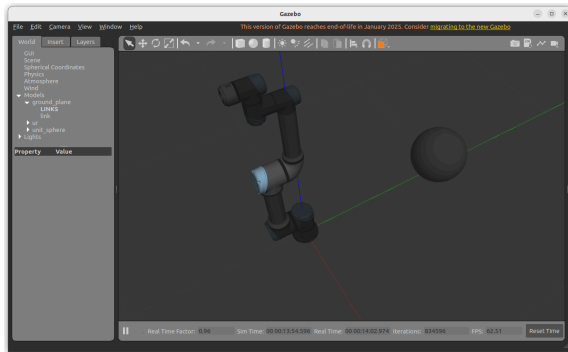
```
ros2 topic pub --once /joint_trajectory_controller/joint_trajectory trajectory_msgs/
  JointTrajectoryController
  stamp:
    sec: 0
    nanosec: 0
    frame_id: ""
  joint_names:
    - shoulder_pan_joint
    - shoulder_lift_joint
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
  points:
    - positions: [3.14, -1.57, 0.0, -1.57, 0.0, 0.0]
    time_from_start:
      sec: 3
      nanosec: 0"
```

\*주의) 한가지 관절의 각도만 설정하는 건 불가능. 모든 관절의 position값 설정해서 토픽 보내야함

**joint\_trajectory\_controller** 가 모든 관절의 이름과 위치 포함한 메시지를 요구

**shoulder\_pan\_joint(최하단 joint)** 회전 결과

⇒ 최하단 관절이 180도 회전 , shoulder\_pan\_joint의 position값 변경



```
name:
- shoulder_pan_joint
- shoulder_lift_joint
- elbow_joint
- wrist_1_joint
- wrist_2_joint
- wrist_3_joint
position:
- 3.140001333107346
- -1.5700025031080551
- 1.0970032366941496e-06
- -1.569999399363465
- -1.2497631218799654e-05
- -2.17576583949608e-06
```

**subscribe:** **joint\_trajectory\_controller** ROS 2 컨트롤러 노드

[trajectory\_msgs/JointTrajectory] 메시지를 수신하여, 로봇 관절을 시간에 따라 제어

```
kiwi@kiwi-sam0: ~/github_package
kiwi@kiwi-sam0: ~/github_package 80x21
kiwi@kiwi-sam0:~/github_package$ ros2 node list
/controller_manager
/gazebo
/gazebo_ros2_control
/interactive_marker_display_94537226026016
/joint_state_broadcaster
/joint_trajectory_controller
/move_group
/move_group_private_106699074122496
/moveit_simple_controller_manager
/robot_state_publisher
/rviz2_moveit
/rviz2_moveit_private_135733449660064
/servo_node
/servo_node_private_105301578717872
/transform_listener_impl_55fb298b9f00
/transform_listener_impl_55fb299cc160
/transform_listener_impl_5fc56f3185c0
/transform_listener_impl_610ad06aeb80
/transform_listener_impl_7b72e801c400
kiwi@kiwi-sam0:~/github_package$
```

## 메세지 인터페이스 확인

ros2 interface show trajectory\_msgs/msg/JointTrajectory

```
# The header is used to specify the coordinate frame and the reference time f
# the trajectory durations
std_msgs/Header header
  builtin_interfaces/Time stamp
    int32 sec
    uint32 nanosec
  string frame_id

# The names of the active joints in each trajectory point. These names are
# ordered and must correspond to the values in each trajectory point.
string[] joint_names
```

```
# Array of trajectory points, which describe the positions, velocities,  
# accelerations and/or efforts of the joints at each time point.
```

```
JointTrajectoryPoint[] points
```

```
float64[] positions
```

```
float64[] velocities
```

```
float64[] accelerations
```

```
float64[] effort
```

```
builtin_interfaces/Duration time_from_start
```

```
int32 sec
```

```
uint32 nanosec
```

```
trajectory_msgs/msg/JointTrajectory
```

```
├── header: 시간 기준 정보
```

```
├── joint_names: 제어할 관절 이름 배열
```

```
└── points[]: 여러 시점에 대한 명령
```

```
    ├── positions: 각 시점의 목표 각도 (rad)
```

```
    ├── velocities: (선택) 각 시점의 속도
```

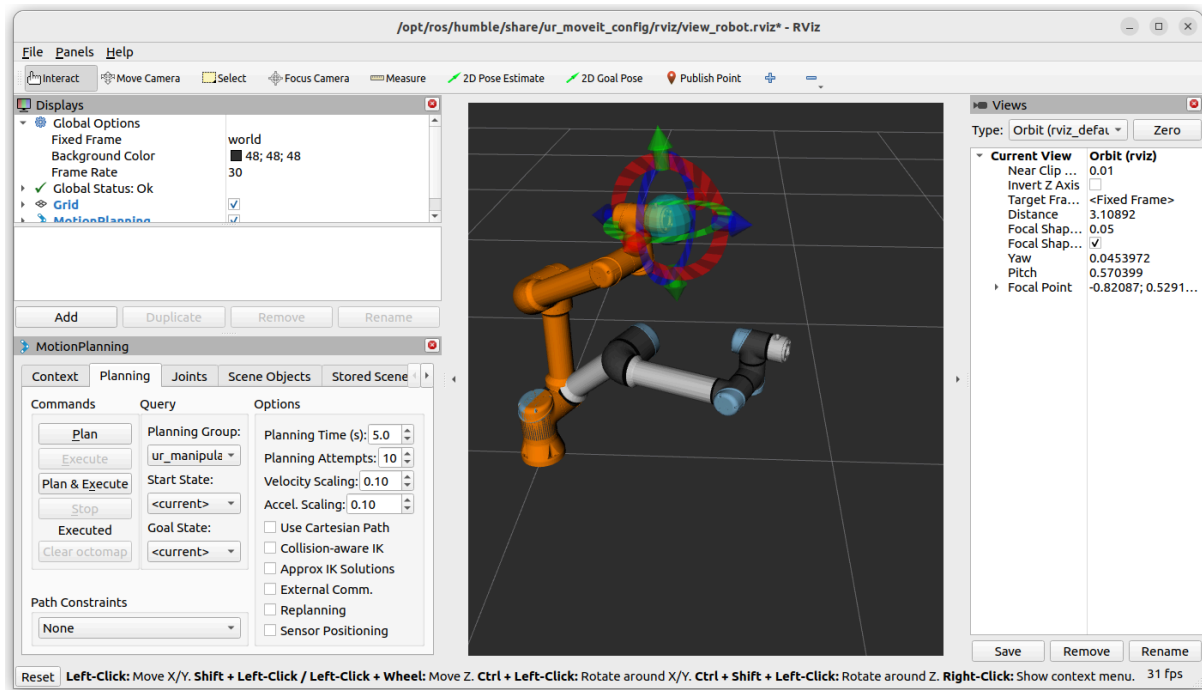
```
    ├── accelerations: (선택) 가속도
```

```
    ├── effort: (선택) 토크/힘
```

```
    └── time_from_start: 시작 기준 경과 시간 (몇 초 안에 도달)
```

## MovelIt2로 제어

목표 위치 설정 → Plan → execute



## joint 이동

