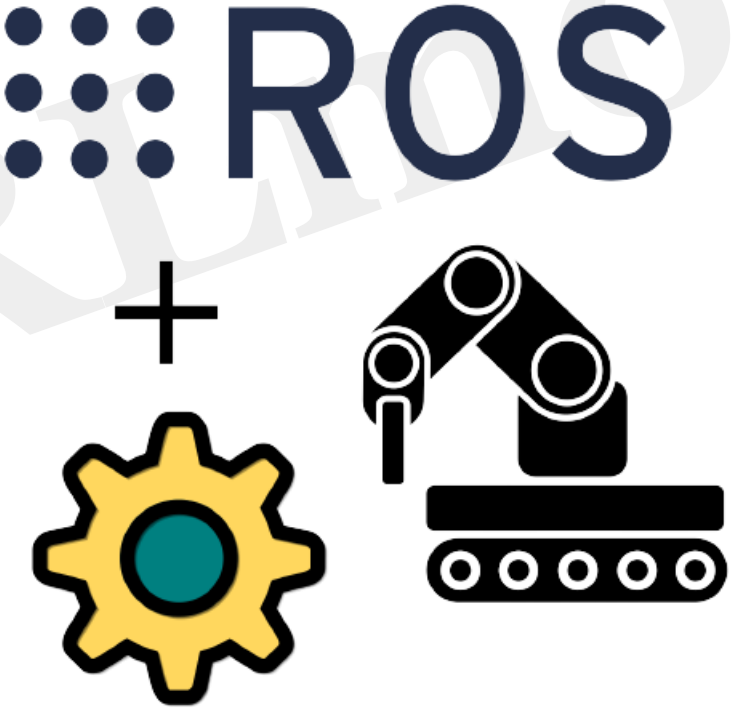
ROS2 커리큘럼

Basic tutorial



RLmodel

ybbaek

Contents

ROS개요

로봇개발의 문제점

기존 로봇 개발 방식과 한계

- 하드웨어 설계, 제어부터 제어기, 비전, 네비게이션 등 모든것을 개발해야 함.

- API마다의 interface가 다르고, 적용하는데 학습이 필요함.

- 하드웨어에 의존적인 소프트웨어적 성격때문에 로봇이 달라지면 소프트웨어 또한 수정이 필요함.

- 소프트웨어를 작성하는데 하드웨어에 대한 지식이 필요함.

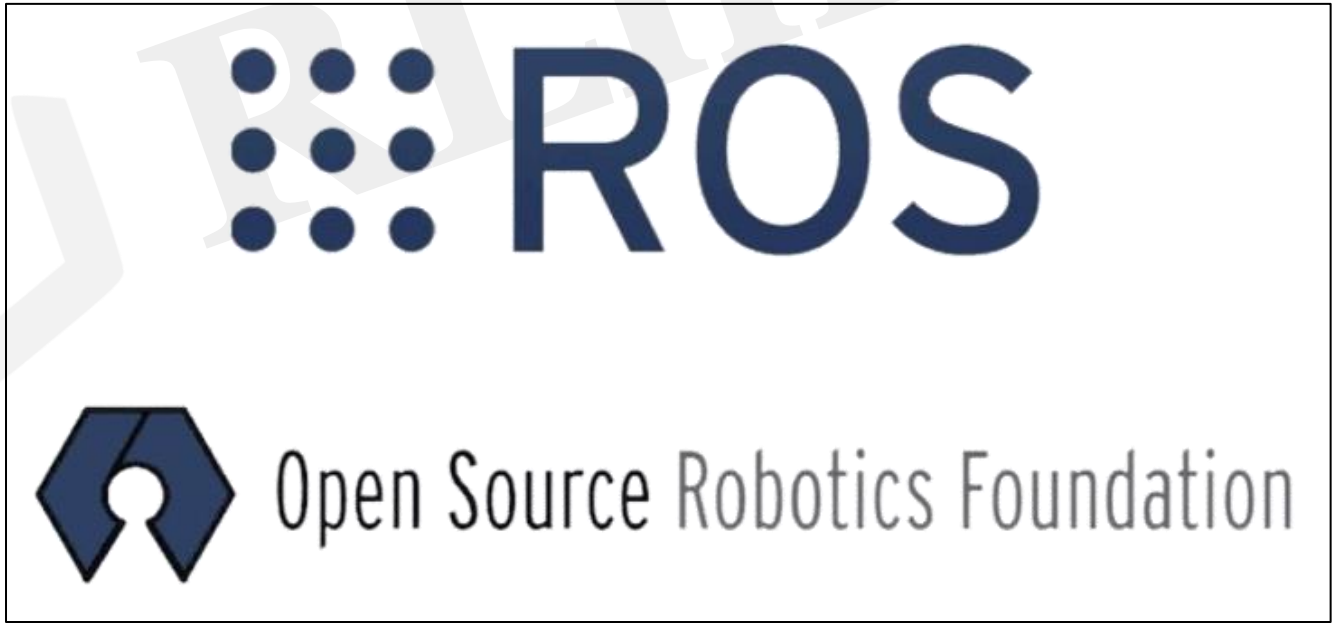
ROS개요

ROS 란

ROS 는 로봇용 공개소스 메타 운영체제

ROS는 일반 운영체제에서 제공하는 하드웨어 추상화, 저수준 기기 제어, 빈번히 사용되는 기능들이 구현되어 있으며, 프로세스간 메시지 전달, 패키지 관리 기능 등을 제공.

또한, ROS는 여러 컴퓨터 시스템 작동하는 코드를 얻어오고, 빌드하고, 작성하고, 실행하기 위한 도구 및 라이브러리를 제공.



ROS개요

ROS의 목적

로브틱스 소프트웨어 개발을 전 세계 레벨에서 공동작업이 가능하도록 하는 환경을 구축하는 것 (로봇 소프트웨어 플랫폼, 미들웨어, 프레임 워크 보다는 연구, 개발에서의 코드 재사용을 극대화가 초점이다.)

ROS의기능

분산프로세스 : 최소단위의 실행가능한 프로세스(Node, 노드) 형태로 프로그램하며, 각 프로세스는 독립적으로 실행되면서 유기적으로 데이터를 주고 받을 수 있다.

패키지 단위 관리 : 같은 작업을 수행하는 여러개의 프로세스를 패키지 단위로 관리가능하여 개발 또는 사용이 편리하고 배포, 공유, 수정에 용이하다.

공개 저장소 : 개발자는 Github 등에 프로젝트를 공개하고 라이센스를 밝히게 되어 있다.

API 형태 : ROS는 API를 불러와 자신이 사용하던 코드를 쉽게 적용가능하다

프로그래밍 지원 : ROS는 다양한 언어를 지원. (Python, C++ Lisp, java, c#, lua, ruby 등)

ROS개요

ROS의 목적

로봇 소프트웨어 플랫폼의 장점

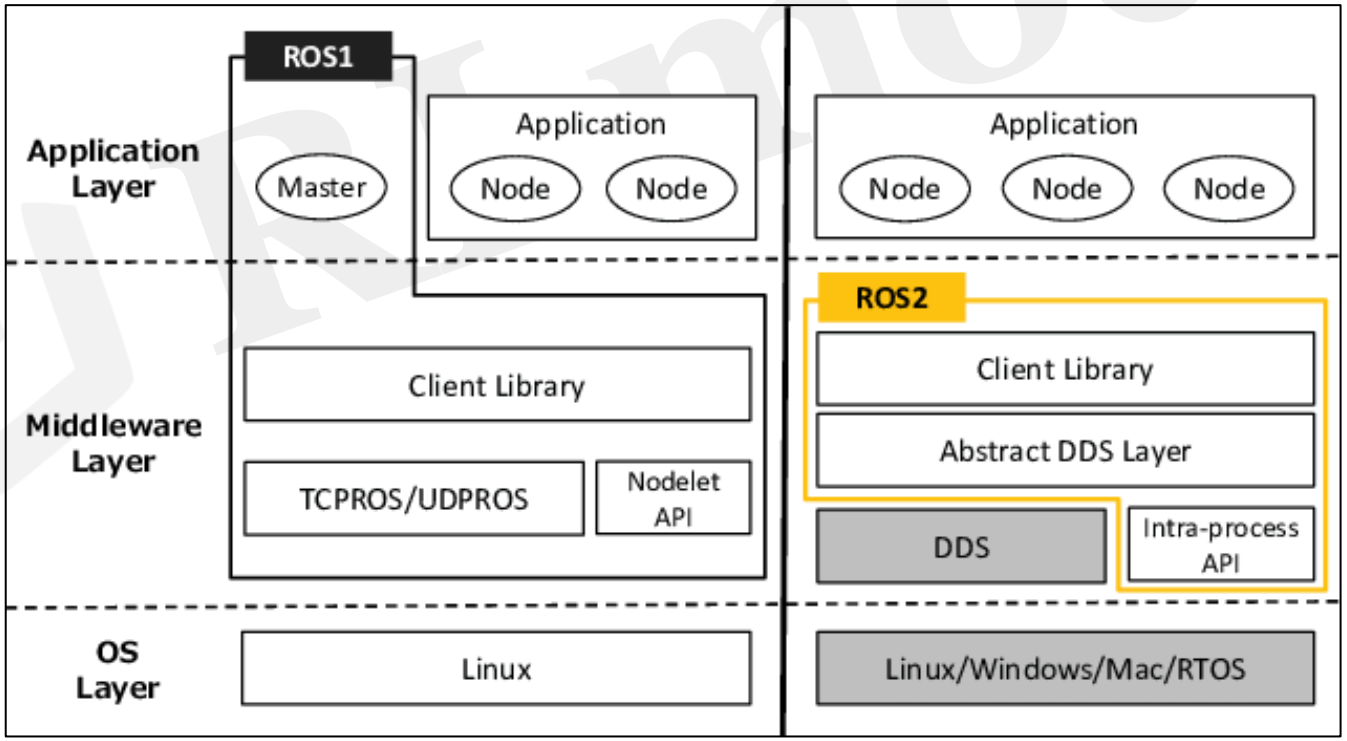
- 높은 프로그램 재사용성
- 통신 기반 프로그램
- 개발도구 지원
- 활성화된 커뮤니티
- 생태계 조성

- 기존의 PC와 Phone의 역사를 Robot에 반영시키기 위한 생태계 형성

ROS1, ROS2 차이

ROS2 특징

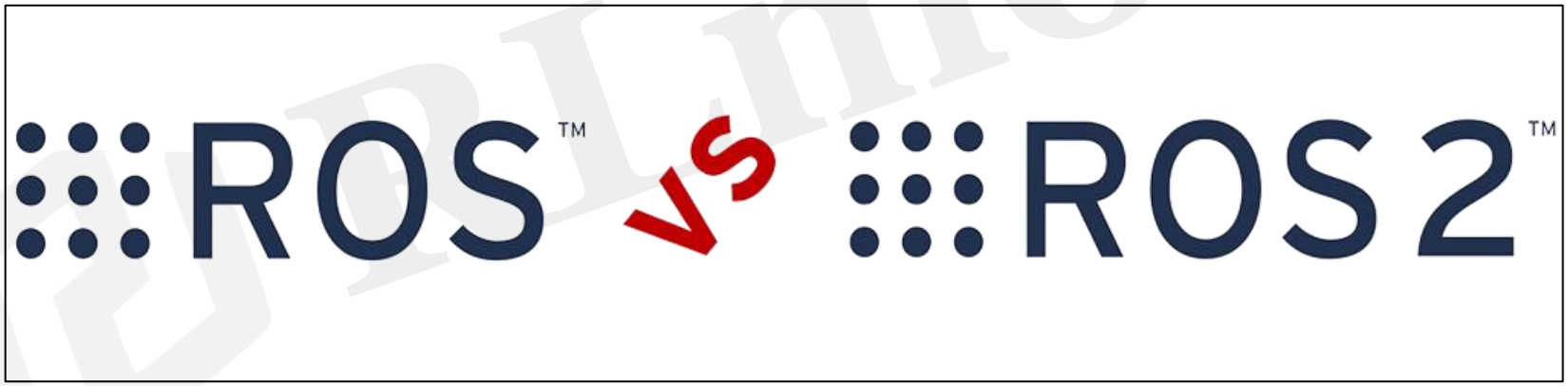
- Platforms: Linux, Windows
- **통신방식**: DDS
- **언어**: Python 3.x
- Build system: ament



ROS1, ROS2 차이

ROS2 특징

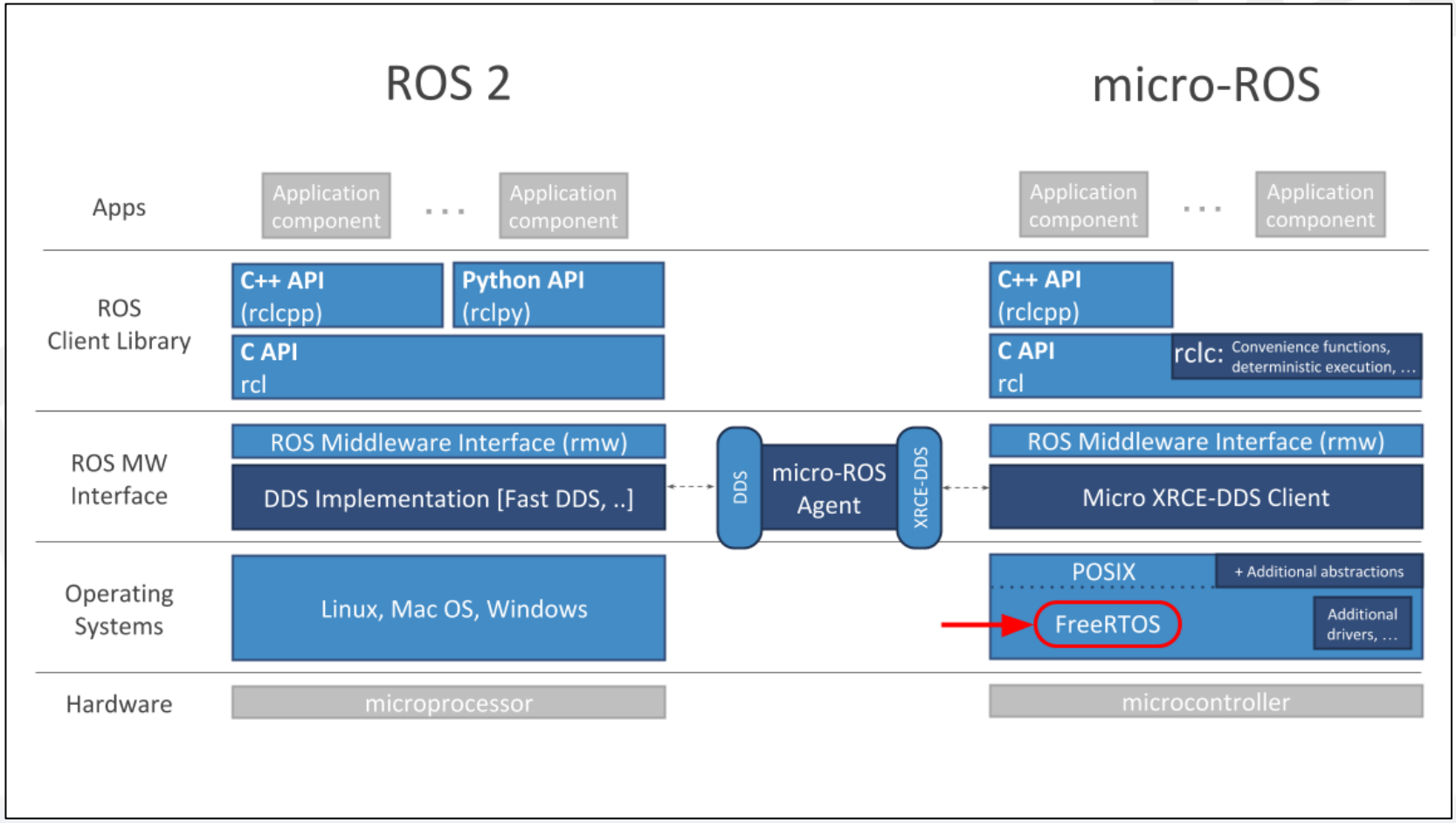
- 복수의 로봇
- 임베디드 시스템에서의 ROS 사용
- 실시간 제어
- 멀티 플랫폼



Micro ROS

정의/특징

- Micro-ROS는 **리소스가 제한된 시스템 및** micro-controller**를 위해 특별히 설계된 가벼운 실시간 프레임워크.**



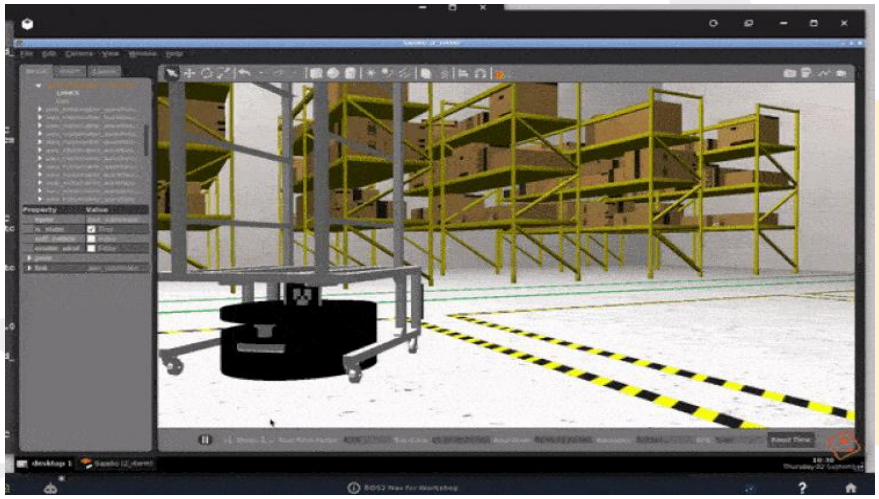
ROS2 구성

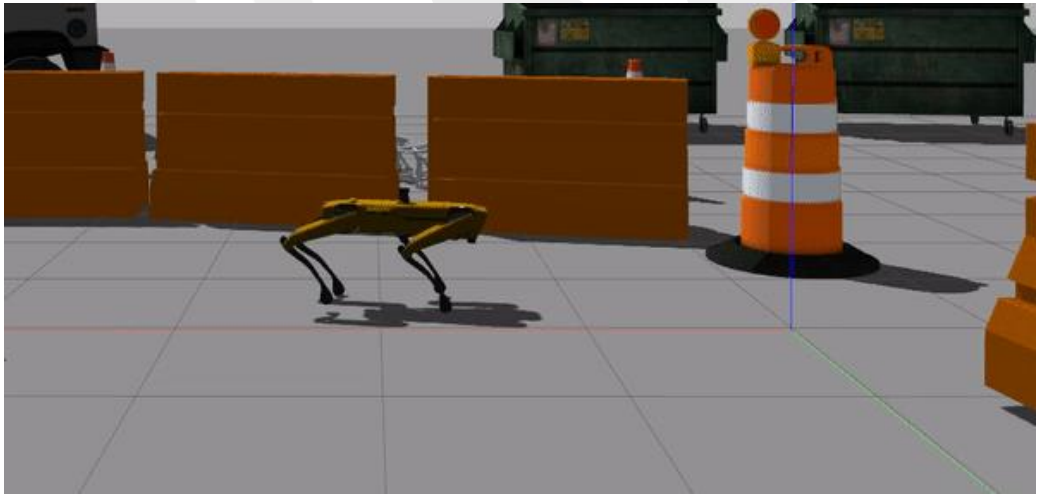
항목

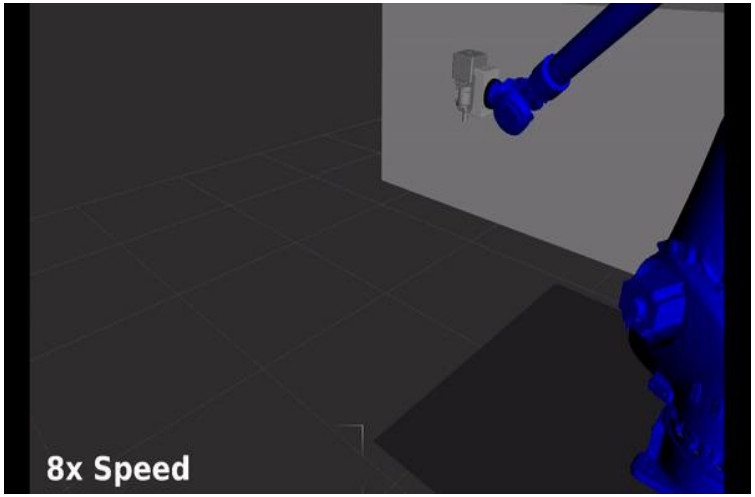
Package	Topic	Node	launch file
rqt rqt_graph	gazebo	rviz	workspace
tf	urdf	DDS	colcon build
service	parameter	yaml	ros bag

ROS2 구성

3D simulation with ROS2

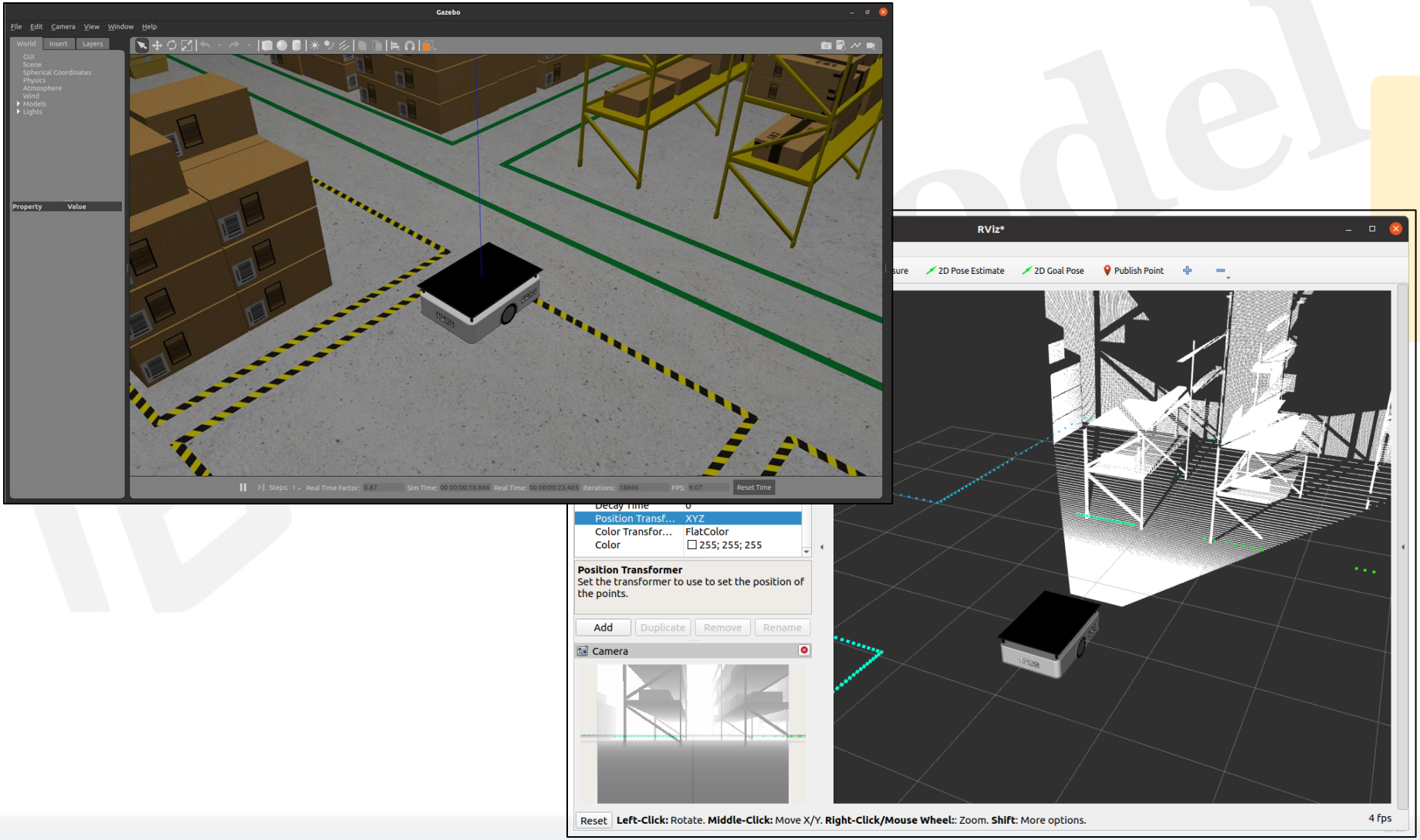






ROS2 구성

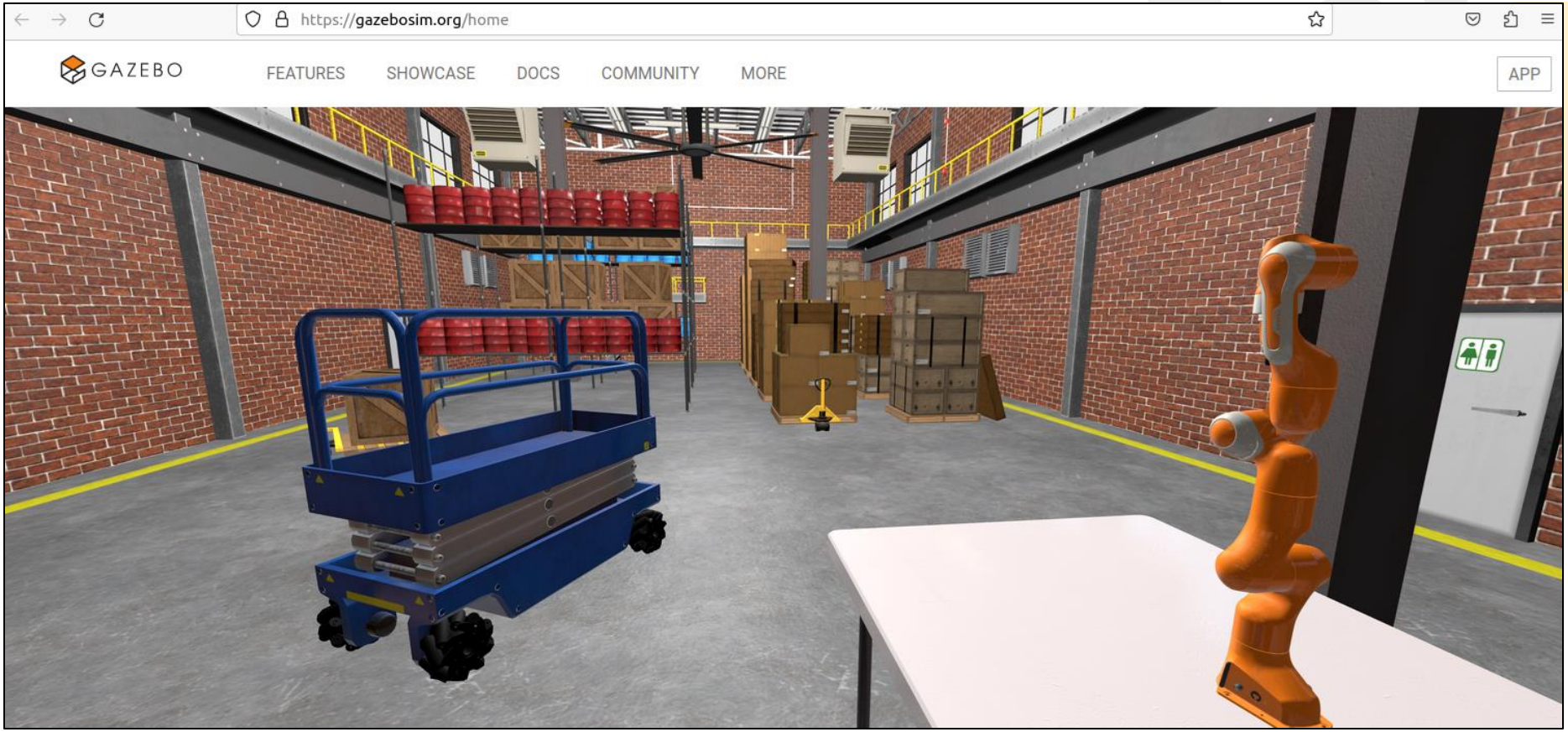
3D simulation with ROS2



ROS2 구성

Gazebo examples

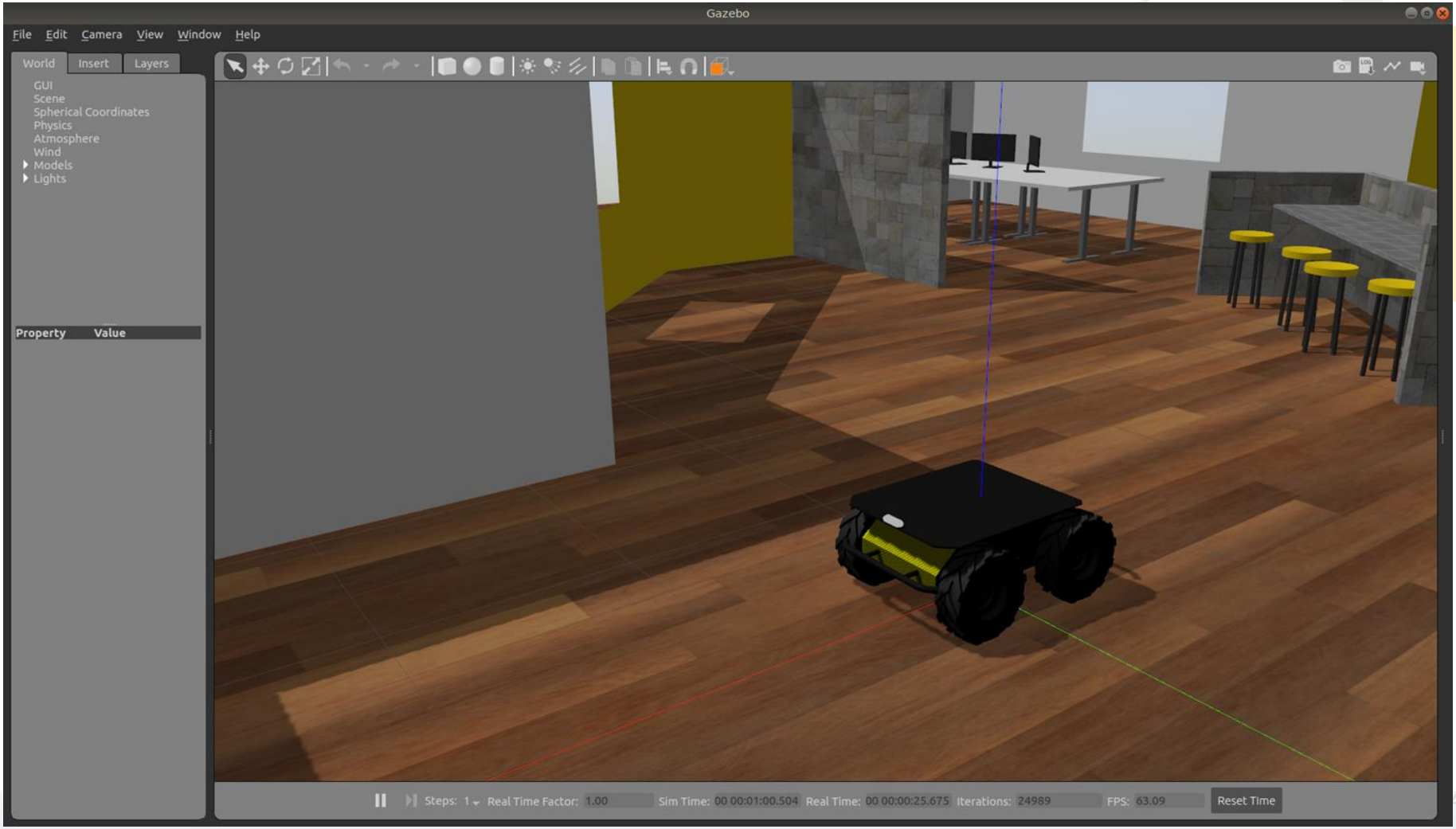
https://gazebosim.org/home



ROS2 구성

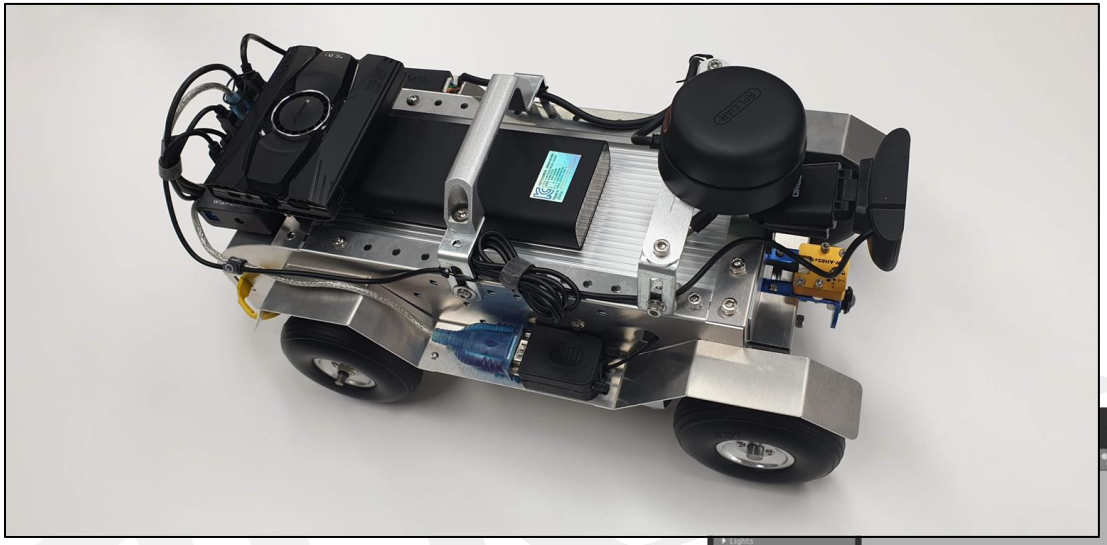
Clearpath Husky robot

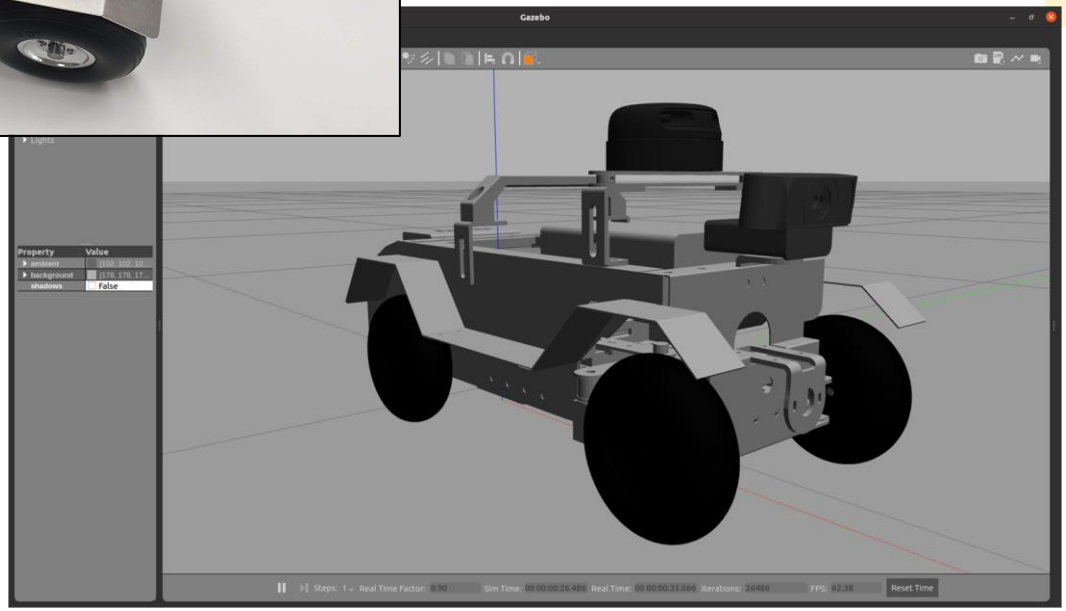
https://www.clearpathrobotics.com/assets/guides/kinetic/husky/additional_sim_worlds.html



ROS2 구성

3D simulation with ROS2

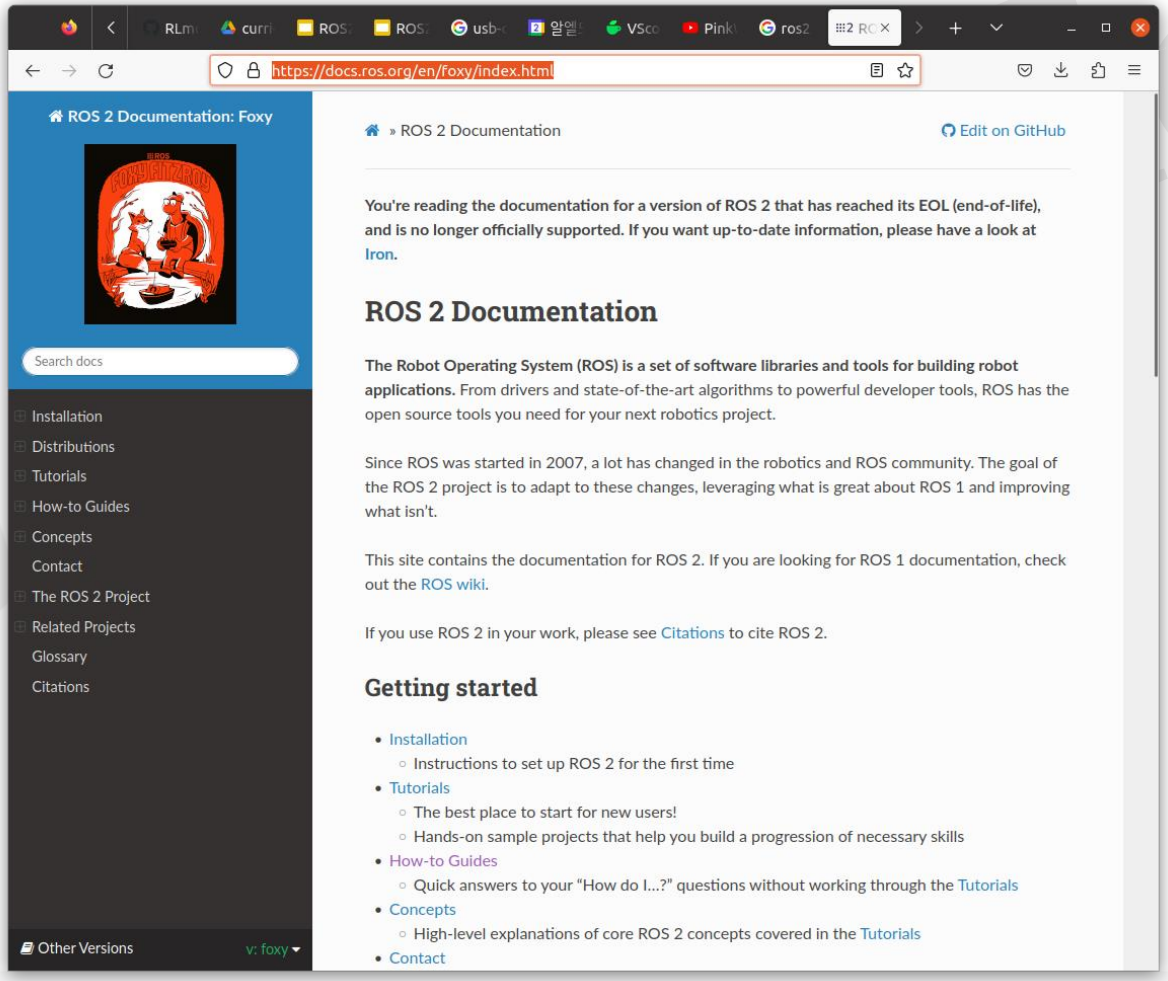




ROS2 doc

ROS2 documentation : foxy

link: https://docs.ros.org/en/foxy/index.html



ROS version

Generation

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

ROS

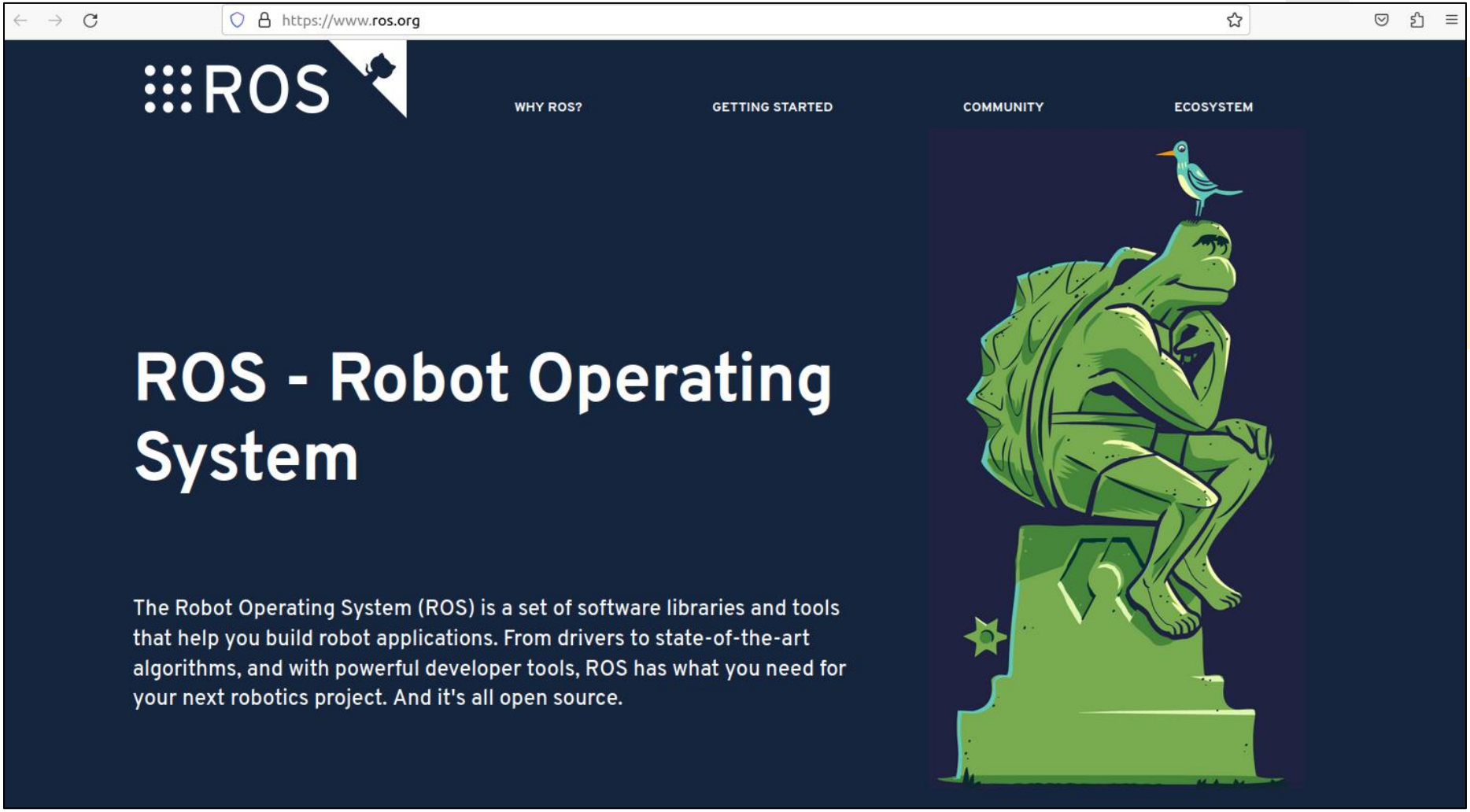
ROS2

20.04 ubuntu

18.04 ubuntu

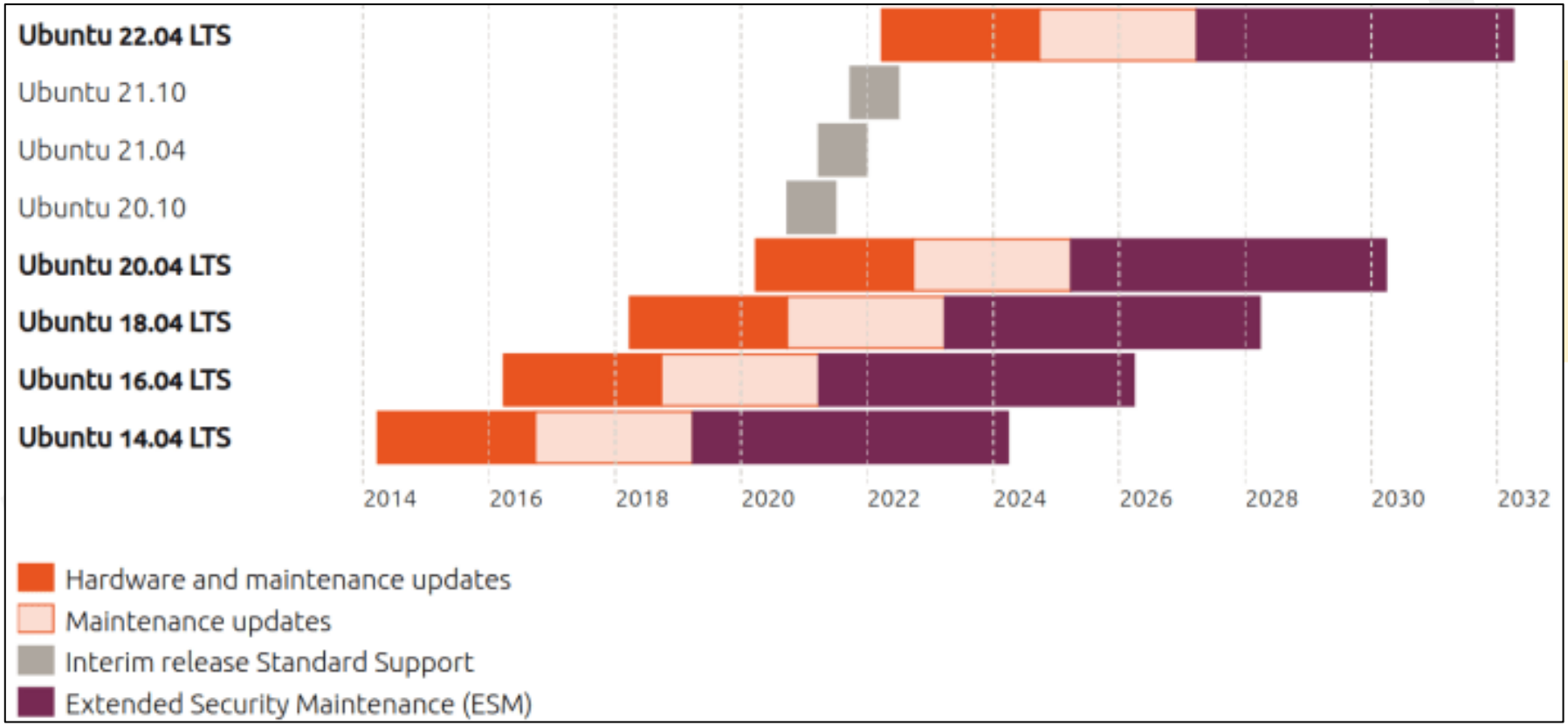
ROS version

https://www.ros.org/



ROS version

Ubuntu version



Build system

설명

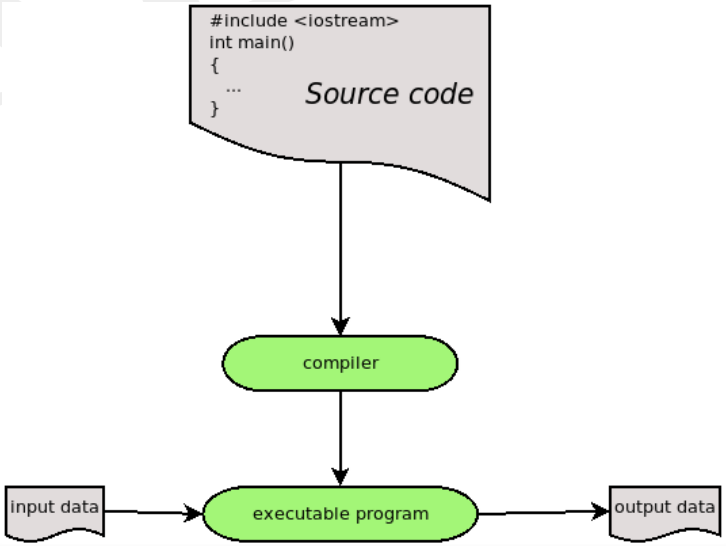
소스코드를 프로그램으로 바꾸어 주는 행위가 바로 빌드를 하는 것이고 이렇게 할 수 있도록 도와주는 도구들을 빌드 시스템이라고 말합

Compile **소스코드 전체를 기계어로 번역**

Interpreted **소스코드를 한 줄씩 번역하면서 실행**

C++ > compile type (전처리 > 컴파일 > 링크)

Python > interpreter type



ROS2 build system

ament

ament는 로봇 소프트웨어 개발을 위한 빌드 시스템, ROS 2에서 사용.

이전 ROS1에서는 Catkin이 주로 사용, ROS 2에서 ament가 새로운 빌드 시스템으로 채택

패키지 기반: ament는 개별 패키지를 중심으로 빌드를 수행합니다. 각 패키지는 독립적으로 빌드되며, 종속성 관리 및 빌드 옵션 설정이 용이합니다. 이를 통해 모듈화된 소프트웨어 시스템을 구성할 수 있습니다.

CMake 기반: ament는 CMake 빌드 시스템을 기반으로 하여 CMakeLists.txt 파일을 사용하여 패키지를 구성합니다. CMake는 강력하고 유연한 빌드 시스템으로 알려져 있으며, 다양한 플랫폼에서 이식성을 제공합니다.

Cross-컴파일 지원: ament는 크로스-컴파일 환경에서도 작동하도록 설계되었습니다. 이는 로봇 시스템과 같이 타겟 플랫폼과 개발 환경이 분리된 경우에 유용합니다.

메타빌드: ament는 패키지 간 종속성을 정의하고 관리하기 위한 메타빌드 시스템을 제공합니다. 이를 통해 패키지 간 종속성을 간편하게 설정하고, 패키지의 빌드 및 설치 순서를 관리할 수 있습니다.

빌드 도구 호환성: ament는 다양한 빌드 도구와 호환됩니다. 예를 들어, CMake로 작성된 패키지를 빌드할 수 있으며, Python 패키지와 같은 다른 유형의 패키지도 지원합니다.

ROS2 build 툴

colcon

Colcon Build는 로봇 소프트웨어 개발에 사용되는 툴 체인의 일부로, ROS(로봇 운영체제) 프레임워크에서 사용되는 빌드 시스템입니다. Colcon은 "COnsolidate Language-specific build CONfiguration"의 약자

다양한 언어 지원: Colcon Build는 ROS 패키지를 구성하는 다양한 언어(C++, Python 등)의 빌드를 지원합니다. 따라서 ROS 프로젝트에서 여러 언어로 작성된 소스 코드를 효율적으로 관리할 수 있습니다.

병렬 빌드: Colcon Build는 병렬 빌드를 지원하여 여러 패키지를 동시에 빌드할 수 있습니다. 이를 통해 빌드 시간을 단축하고 개발자들의 생산성을 향상시킬 수 있습니다.

빌드 종속성 관리: Colcon Build는 패키지 간의 종속성을 자동으로 관리합니다. 패키지 간의 의존성을 정확히 파악하고 필요한 종속성 패키지를 자동으로 설치하므로, 빌드 과정에서 발생할 수 있는 문제를 최소화할 수 있습니다.

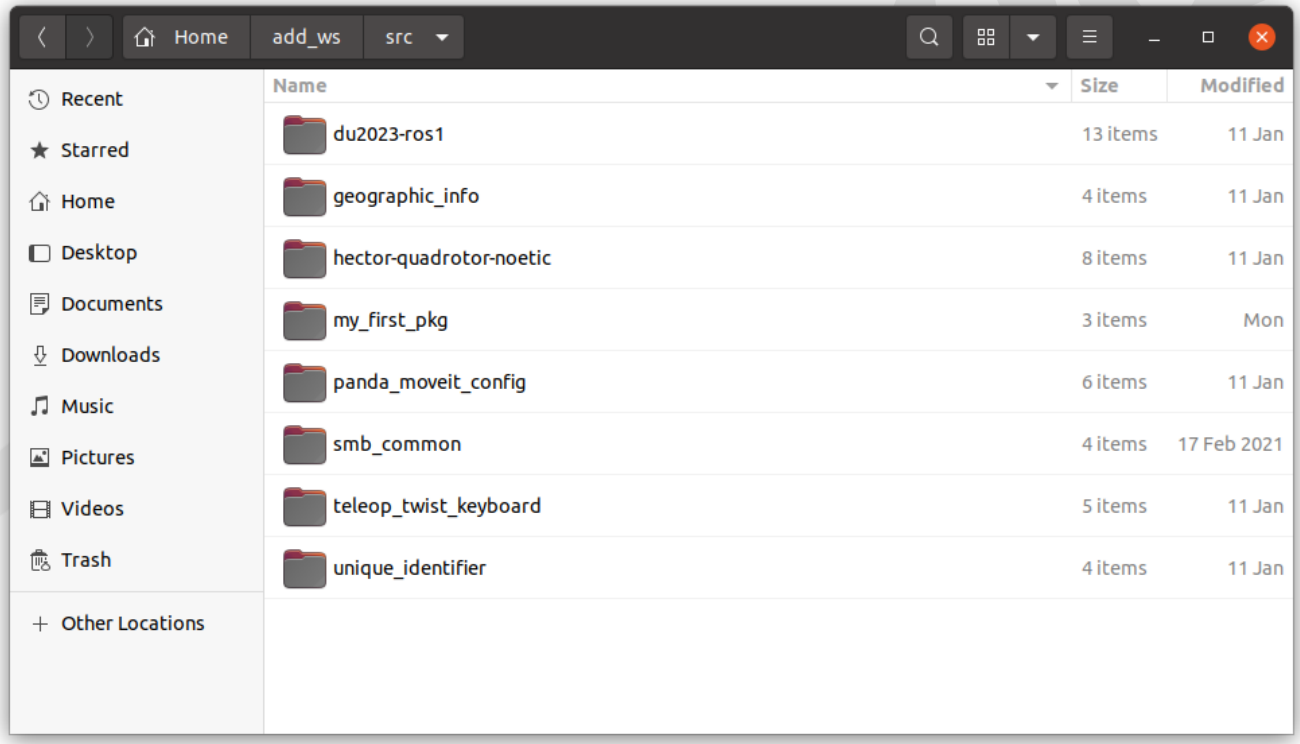
커스터마이징 가능: Colcon Build는 다양한 빌드 옵션을 제공하여 사용자가 빌드 프로세스를 커스터마이징할 수 있습니다. 빌드 타겟, 빌드 옵션, 환경 변수 설정 등을 조정하여 개발자들의 요구에 맞게 빌드를 구성할 수 있습니다.

Package

Package 정의

패키지(package)는 하나 이상의 노드(node)가 기능적 단위로 묶인 ROS 코드의 컨테이너입니다.

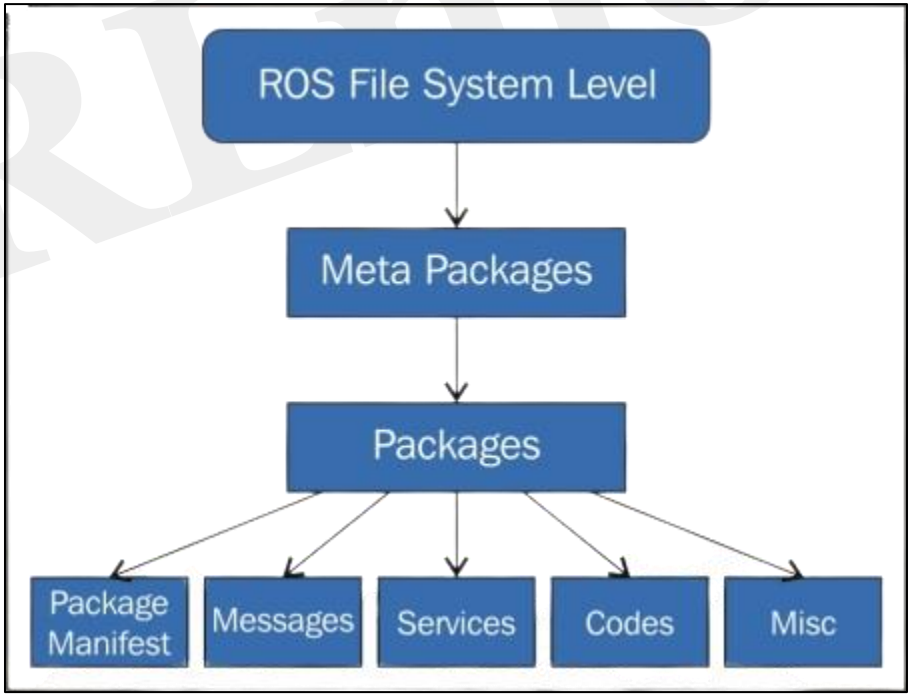
코드를 설치하거나 공유하기 위해서는 패키지를 구성해야합니다. ROS2는 ament를 빌드 시스템으로 사용하고 colcon을 빌드 툴로 사용합니다



ROS2 구성

패키지

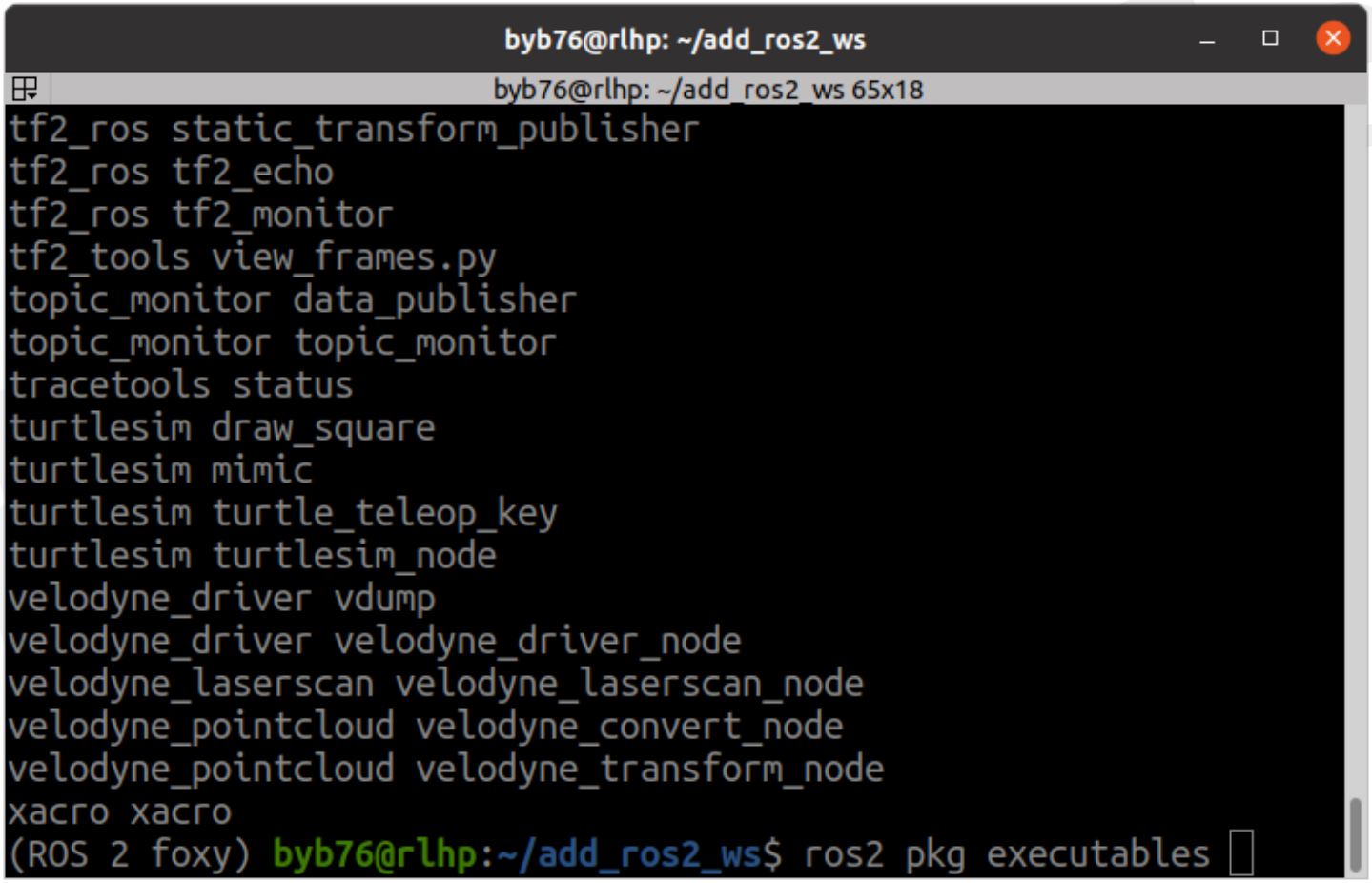
- ROS2 코드의 컨테이너
- 최소 패키지 구조
  - package.xml : 패키지에 대한 메타 정보가 포함된 파일
  - CMakeLists.txt : 패키지 내에서 코드를 빌드하는 방법을 설명하는 파일
- 보통 저장공간은 ~/dev_ws/src/패키지 로 구성



ROS2 구성

package 확인 (내장)

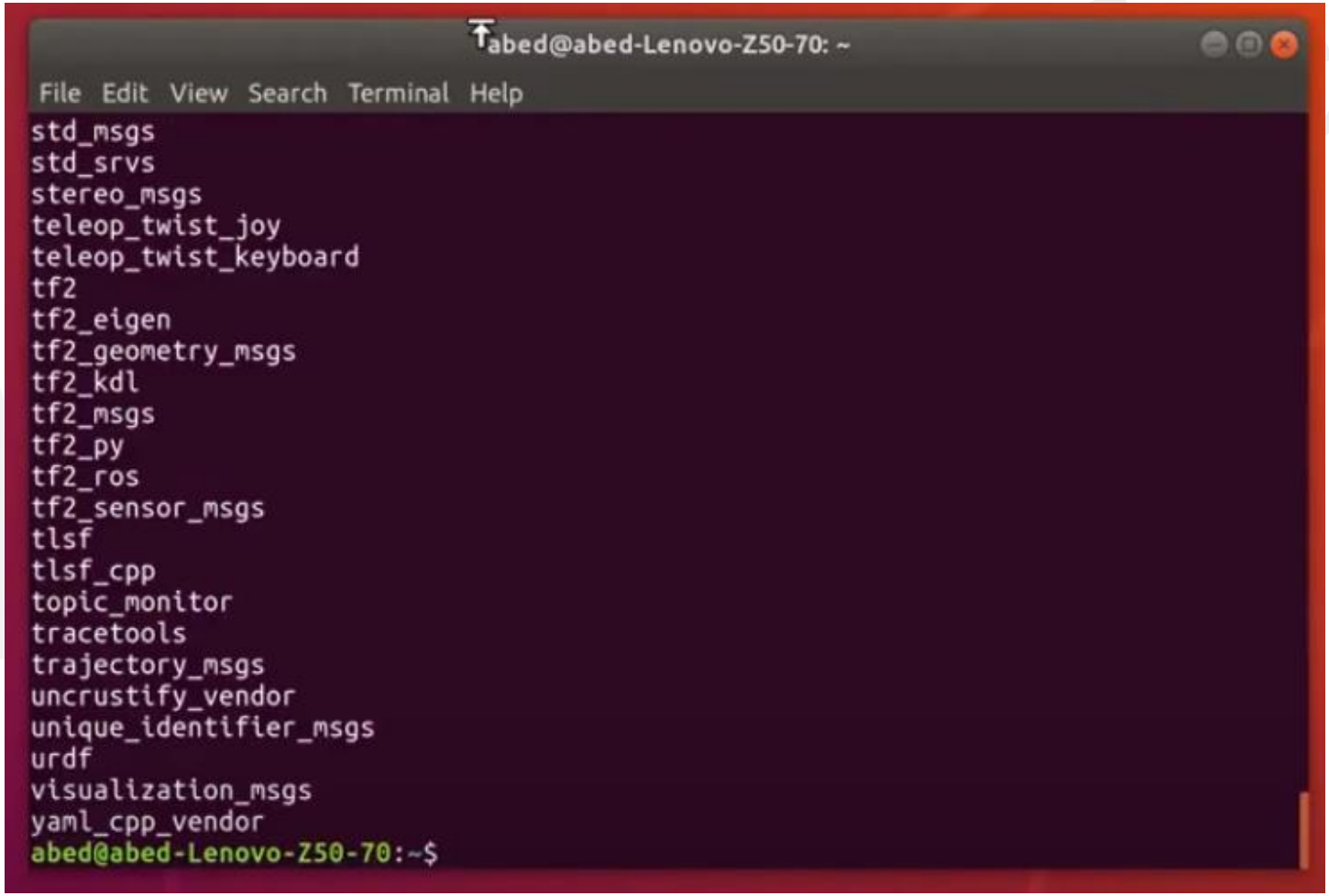
cmd: $ ros2 pkg executables



ROS2 패키지

package 항목

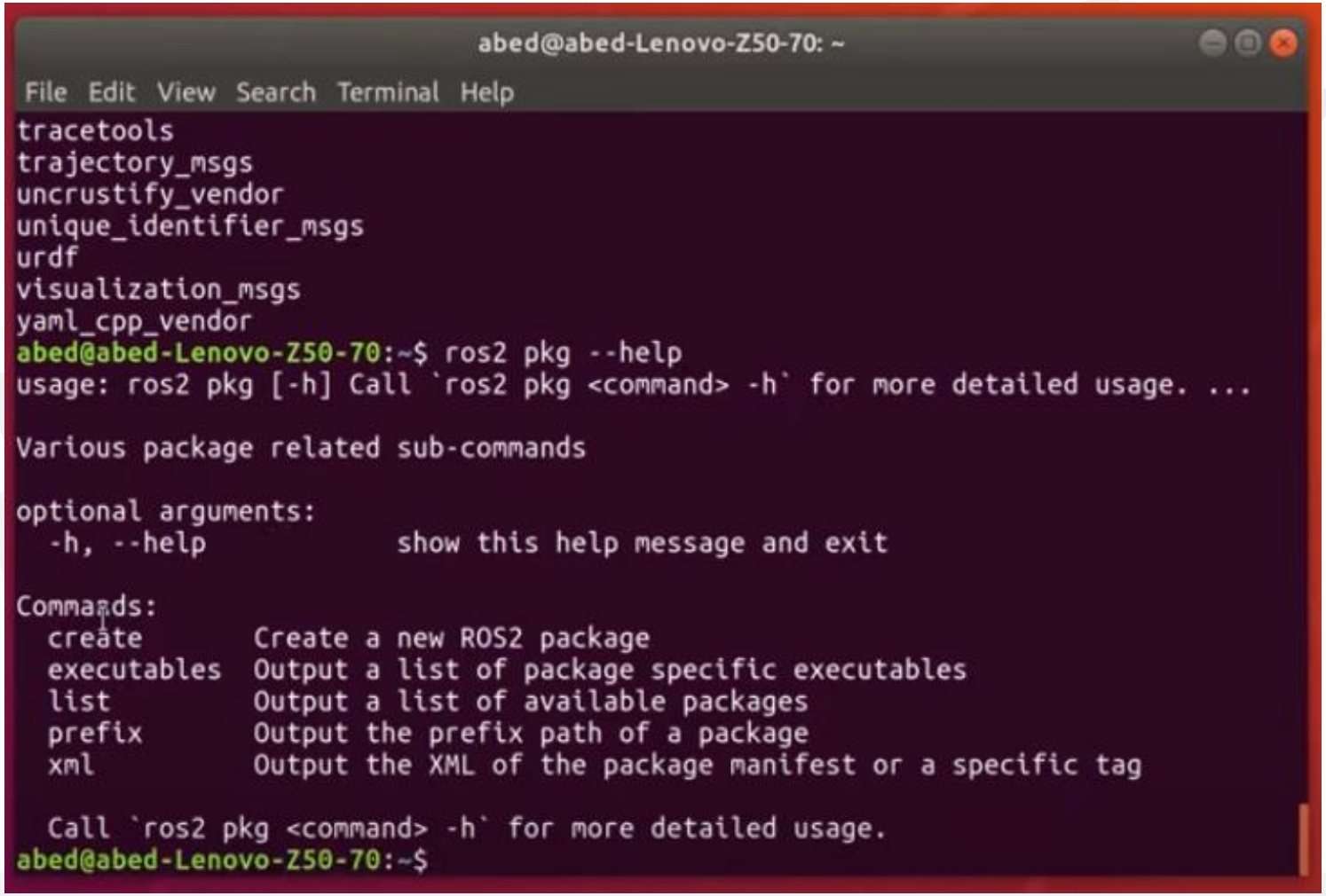
확인명령어: $ ros2 pkg list



ROS2 패키지

package 항목

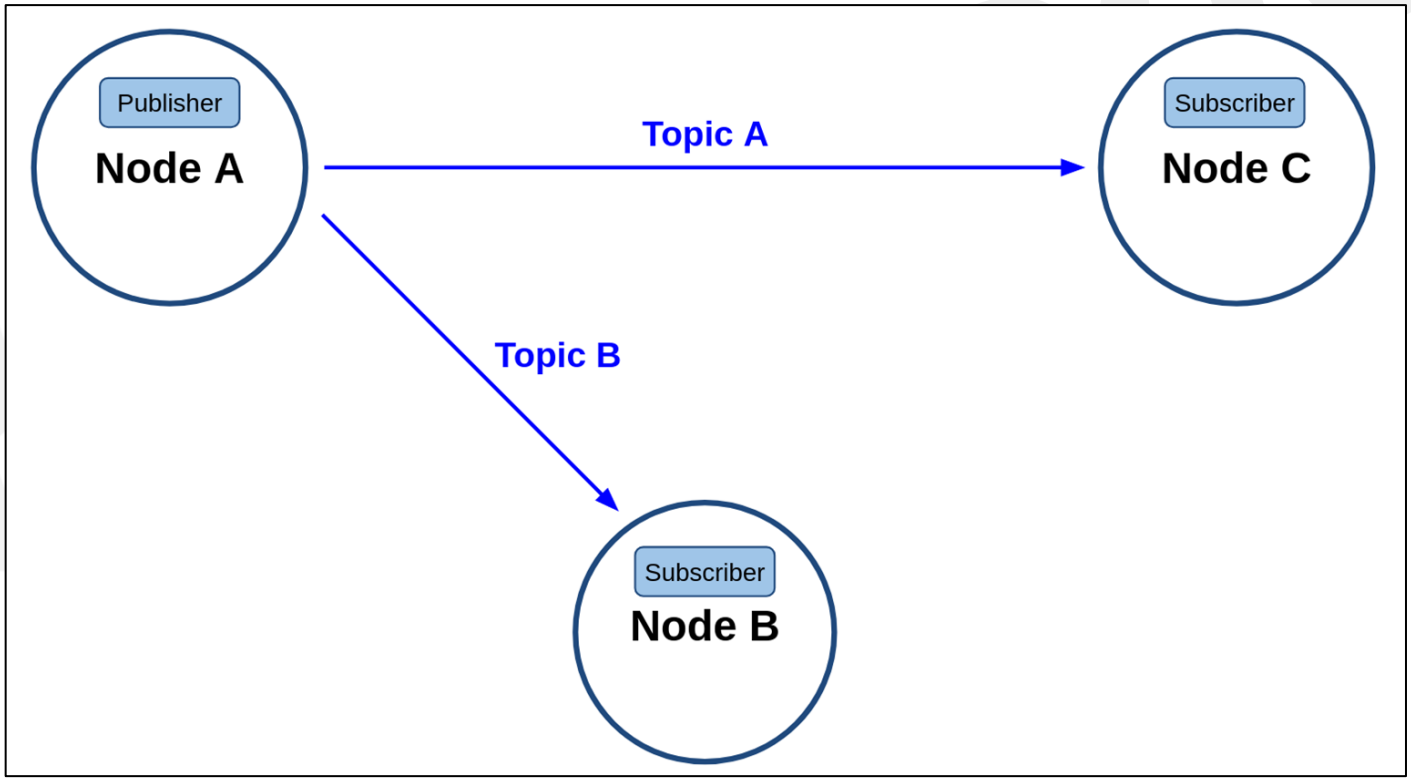
도움말 명령어: $ ros2 pkg –help



Node

node (노드)란

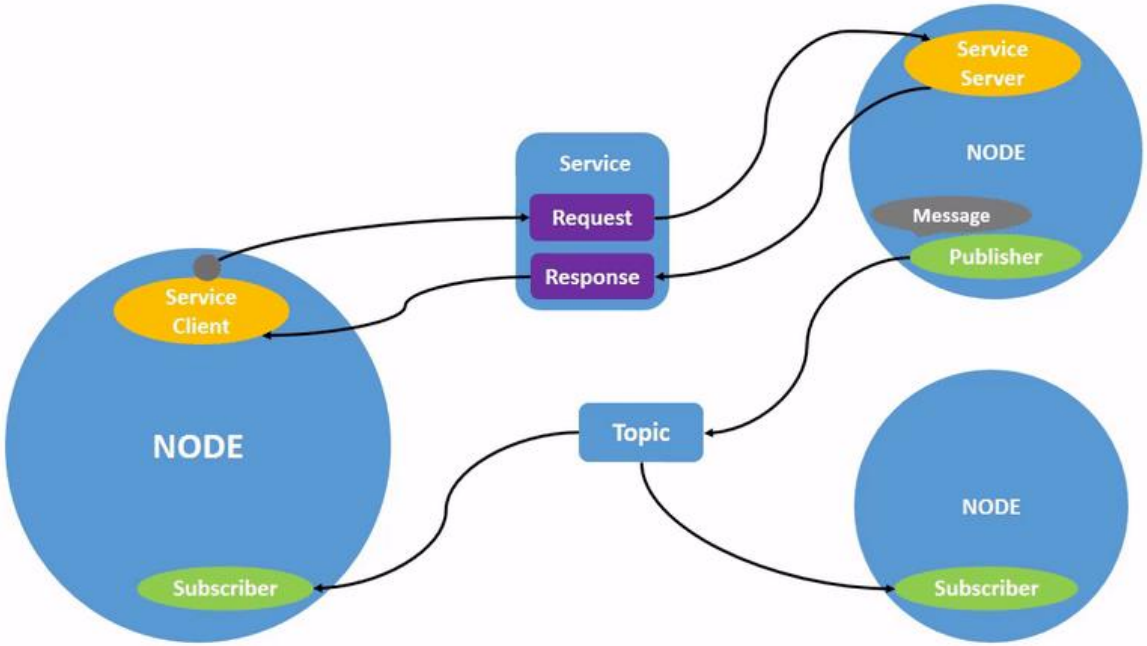
- Node 는 필요한 연산을 실행하는 최소 프로세스이다.
- ROS는 노드 단위로 프로그램을 작성하며 노드사이에 메시지를 주고 받으면서 통신합니다



Node

node 특징

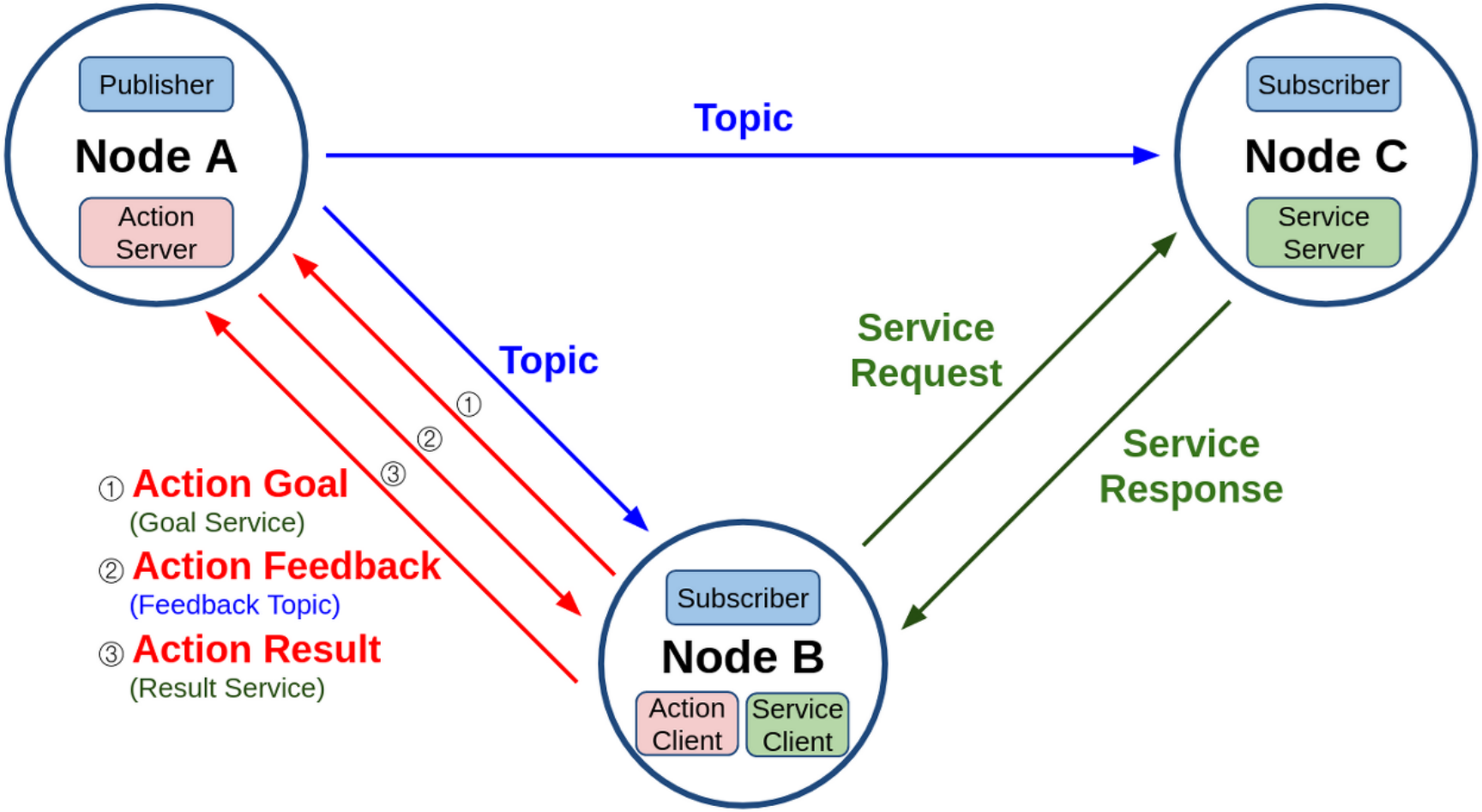
- 로봇 제어 시스템은 일반적으로 많은 Node 로 구성. (예) Node1 은 레이저 거리 측정기를 제어하고, Node2 는 로봇의 휠 모터를 제어하고, 또 다른 하나의 Node3 은 로컬라이제이션을 수행
- Node 들을 묶어서 Package 를 만들 수 있다.



Node

node 구성

다양한 Node 들의 구성으로 Package(패키지)를 만들 수 있다.



ROS2 interface

interface

ROS 노드 간에 데이터를 주고받기 위해서 토픽, 서비스, 액션을 사용하고, 이때 사용되는 데이터 형태를 ROS 2 인터페이스 라고 한다.

이들은 각각 msg, srv, action interface를 사용하며, 정수, 부동 소수점, boolean과 같은 단순 자료형과, 데이터 구조와 메시지들이 나열된 배열같은 구조로도 사용이 가능.

ex) cmd_vel 토픽 > geometry_msgs/msg/Twist

```
ybbaek@ubuntu:~$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
```

ROS2 interface

interface

ex) **geometry_msgs** 정의

http://wiki.ros.org/geometry_msgs

ROS2 interface

interface

ex) Twist Message 정의

http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Mon, 28 Feb 2022 22:12:13

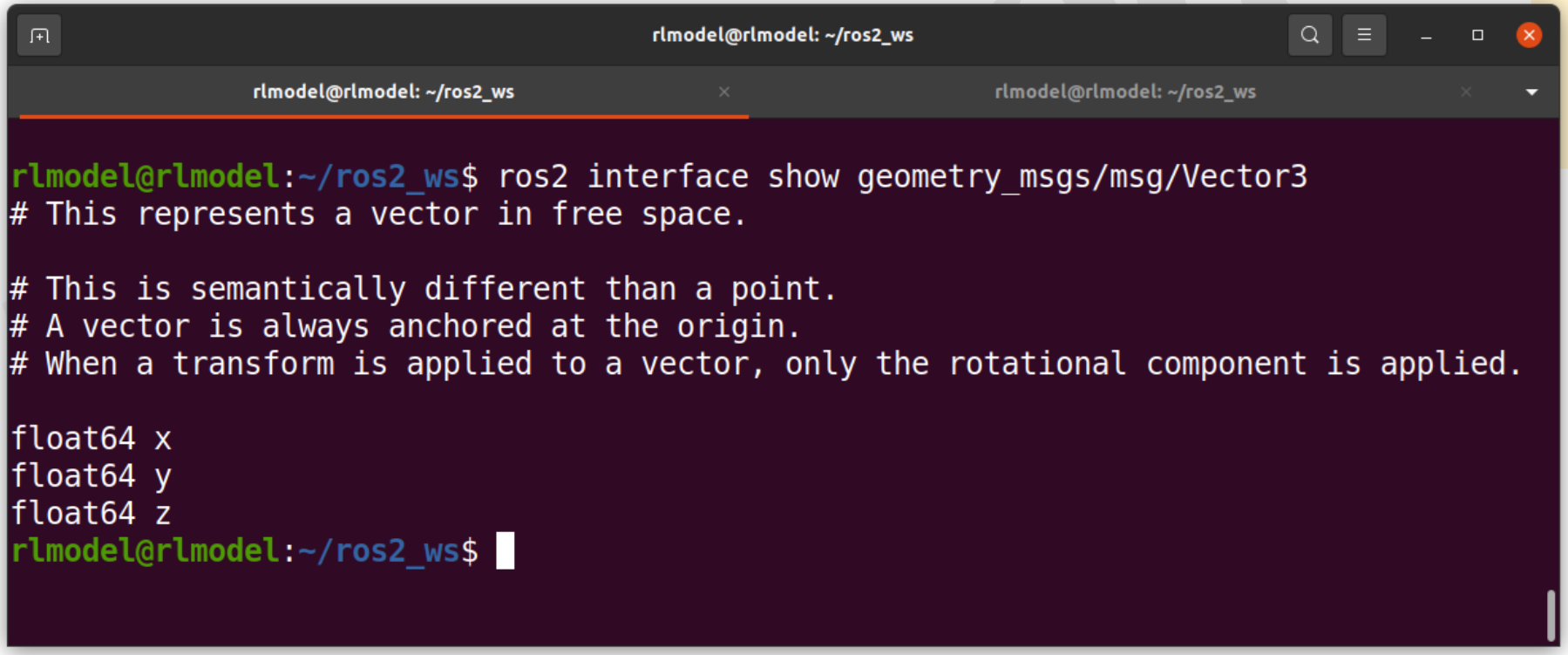
ROS2 interface

interface

ex) Twist Message 정의

geometry_msgs/msg/Vector3

ros2 interface show geometry_msgs/msg/Vector3



ROS2 parameter

parameter 정의

ROS 2의 파라미터는 각 노드가 가진 Parameter Server를 통하여 외부의 Parameter Client와 통신하여 파라미터를 변경하는 것.

ROS 2의 Service와 비슷하지만, 파라미터는 서비스 통신 방법을 사용하여 노드 내부, 외부에서 노드 내 매개변수를 쉽게 지정, 변경, 가져와서 사용할 수 있게 하는 점에서 사용 목적이 다름.

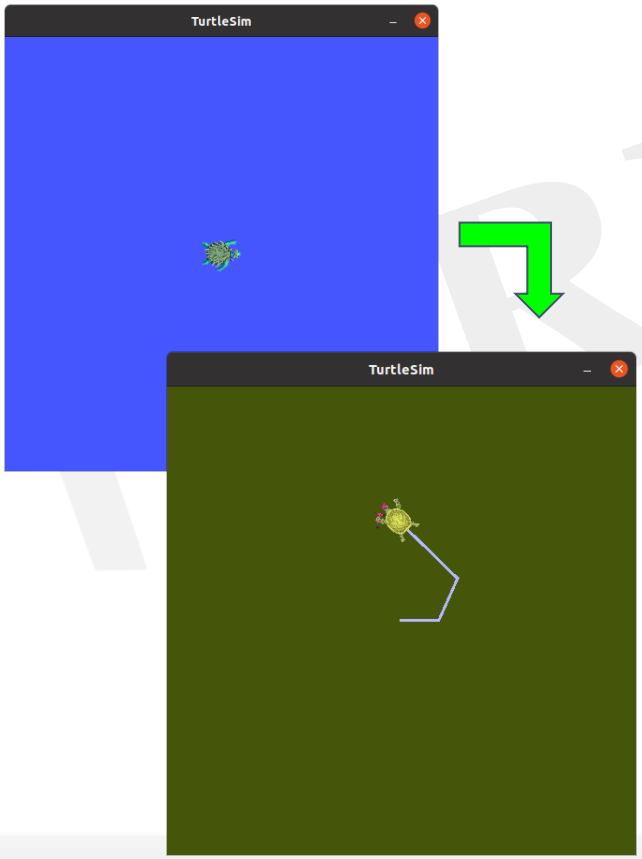
>확인 명령어 $ ros2 param list

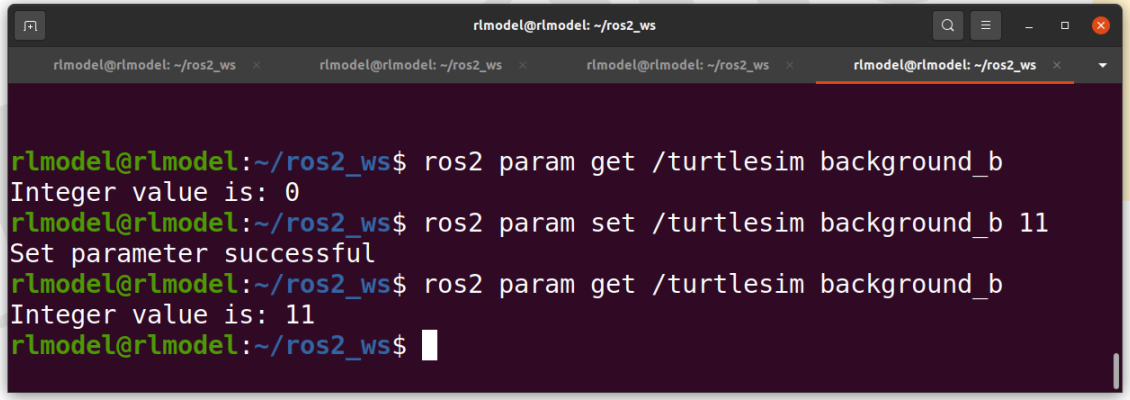
ROS2 구성

parameter

설명: 노드 내에서 사용되는 변수이고, 이 값을 노드 내 또는 ros2 param 명령어를 통해 확인하거나 수정

명령어: ros2 param (set, get, list, describe…)





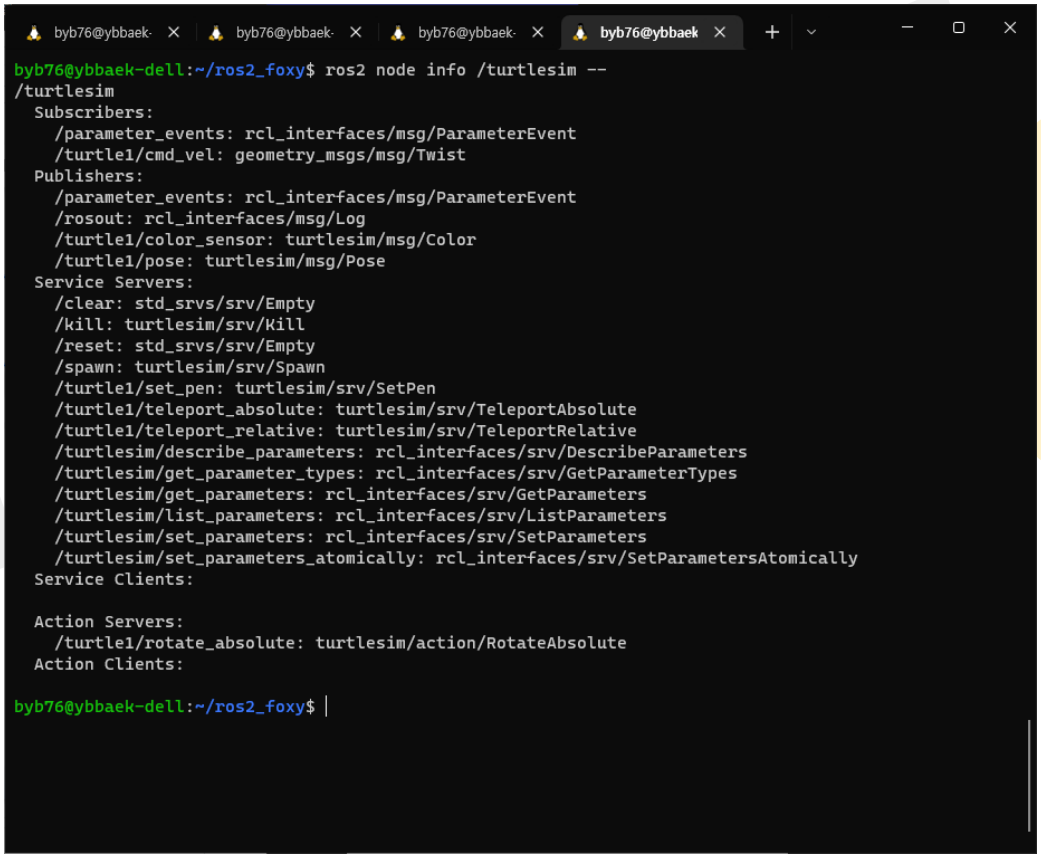
ROS2 구성

node 확인

node 관련 명령어:

$ros2 node list

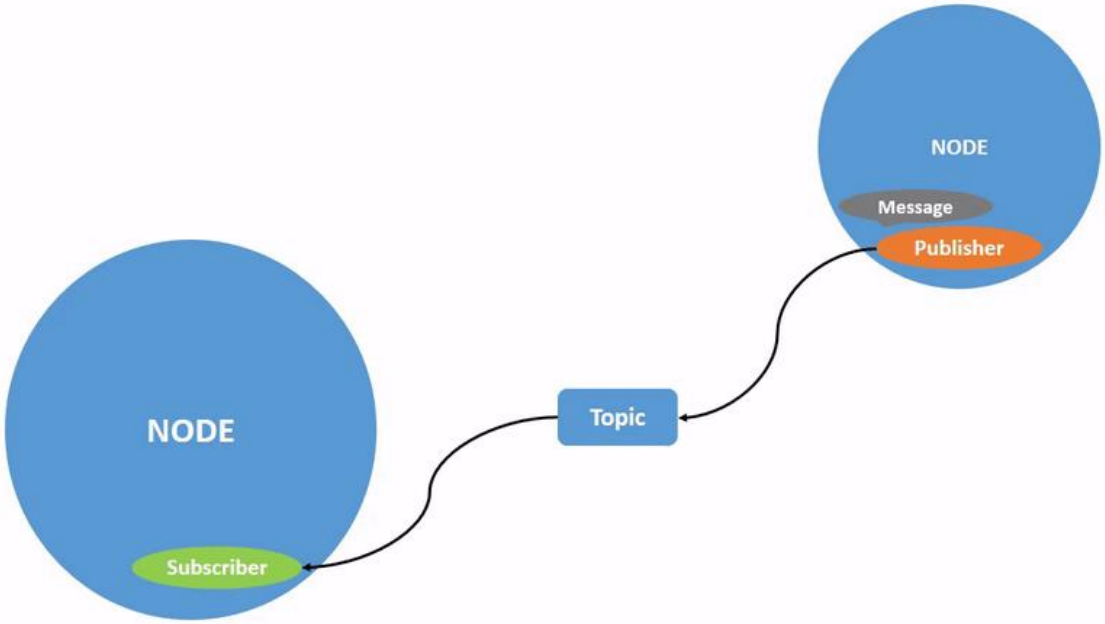
$ros2 node info /xxx



ROS2 구성

topic 정의

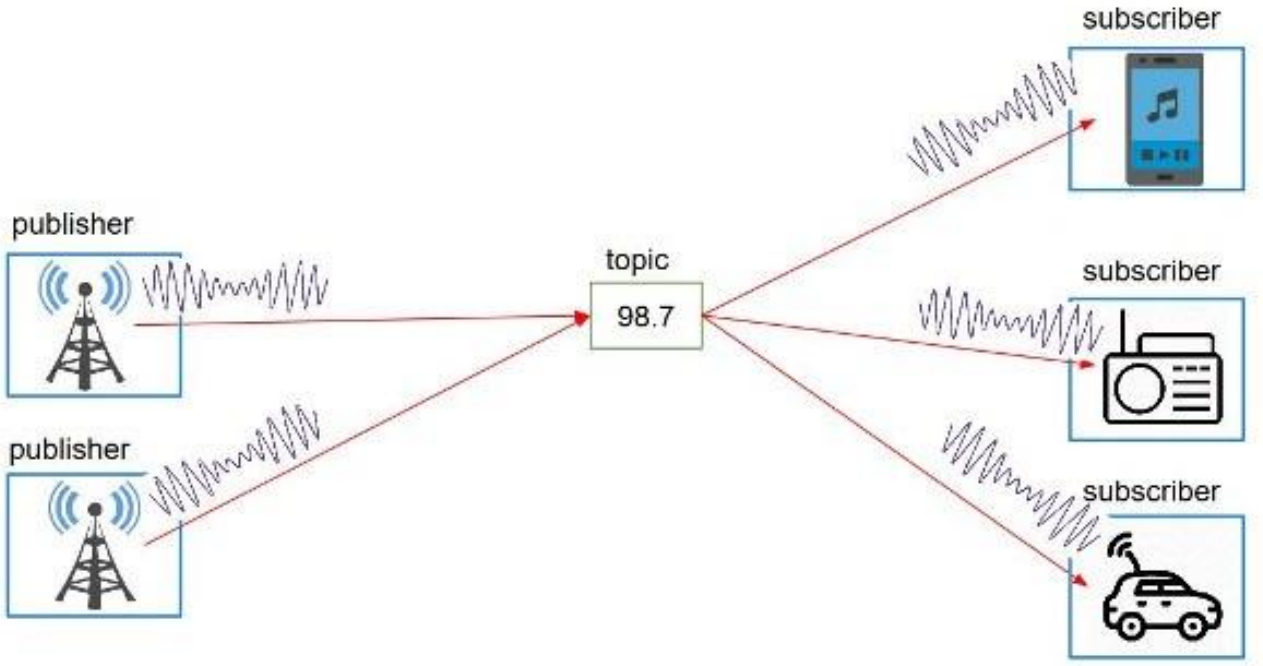
>Topic이란 **노드들 간에 통신을 할 수 있는 채널**이다. 두 프로그램 간에 어떤 통신이 발생하고, 어떤 메시지를 주고 받게 되는데 이 경로를 토픽이라고 한다. - 메세지는 Topic이 데이터나 정보를 주고 받을 때 사용하는 포맷임



ROS2 구성

topic 특징

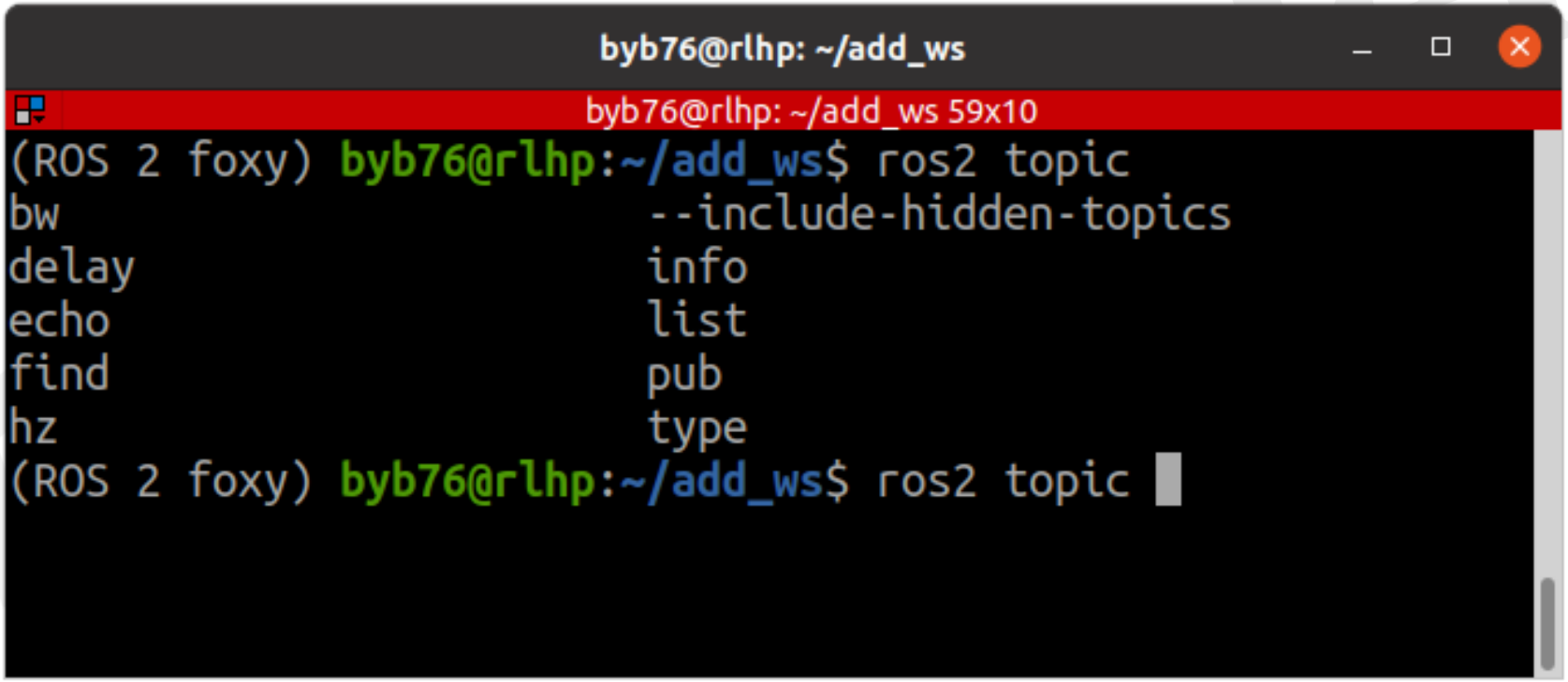
- Topic 을 주는 쪽을 송신자 (Publisher), 받는 쪽을 수신자, 구독자 (Subscriber) 라고 한다.
- 1:1, 1:N, N:N 등, 다양한 네트워크를 구성할 수 있다.
- 비동기식 단방향 메시 송수신 방식



ROS2 구성

topic 관련 명령어:

자주사용: list, info, echo

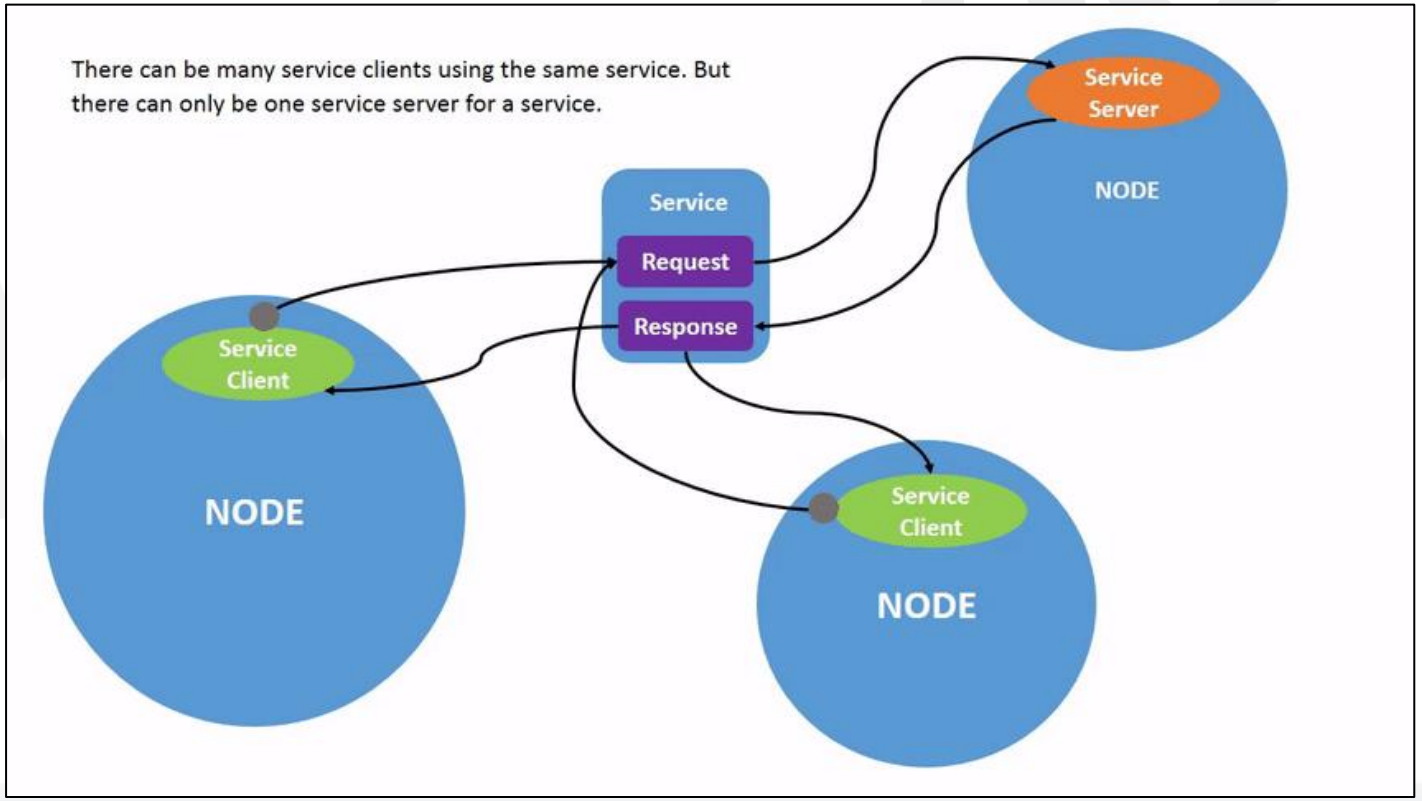


ROS2 구성

service 란

정의: 클라이언트(client)의 요청(request)과 서버(server)의 응답(response)으로 이루어진 통신방식

Service의 중요한 특징 한 가지 추가하자면, 하나의 Service Server에 여러 Client가 request 할 수 있지만, Server는 동시에 여러 request를 처리하지 못합니다.



ROS2 구성

service 설명

Service 통신 시 srv라는 데이터 형식이 사용됩니다. 특정 srv가 어떻게 구성되어 있는지 알고 싶을 때는 ros2 interface show를 사용합니다.

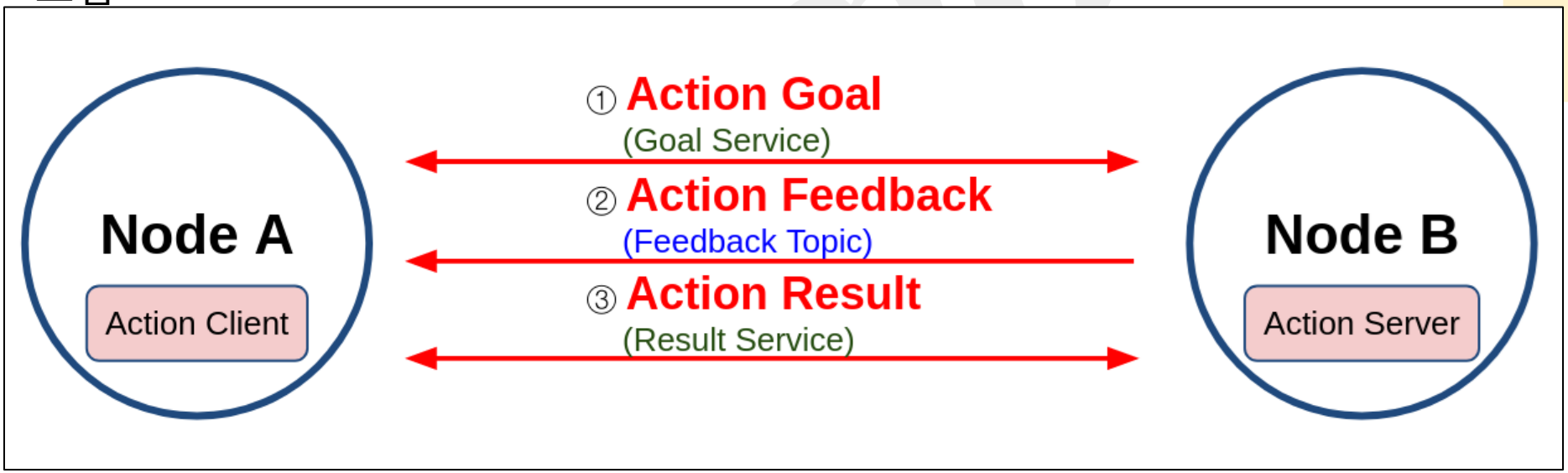
```
$ ros2 interface show gazebo_msgs/srv/SpawnEntity
string name # Name of the entity to be spawned (optional).
string xml # Entity XML description as a string, either URDF or SDF.
string robot_namespace # Spawn robot and all ROS interfaces under this namespace
geometry_msgs/Pose initial_pose # Initial entity pose.
string reference_frame # initial_pose is defined relative to the frame of this entity.
# If left empty or "world" or "map", then gazebo world frame is
# used.
# If non-existent entity is specified, an error is returned
# and the entity is not spawned.
---
bool success # Return true if spawned successfully.
string status_message # Comments if available.
```

ROS2 구성

action 이란

설명: 액션(action)은 토픽(topic)과 서비스(service)가 혼합된 형태의 통신방식

액션 클라이언트(client)가 서비스를 통해 목표(goal)를 설정하면 액션 서버(server)는 결과(result)에 도달할때까지 토픽을 통해 피드백(feedback)을 보냄



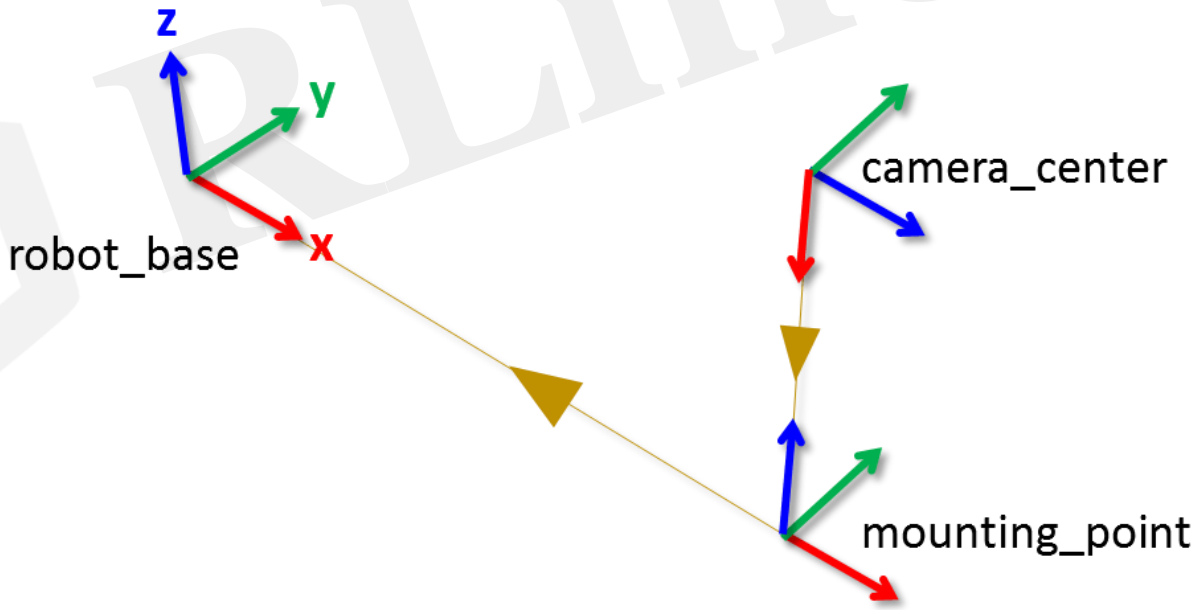
Transform

tf2 란

설명: ros에서 tf란 프레임을 추적할 수 있게 해주는 ros의 라이브러리

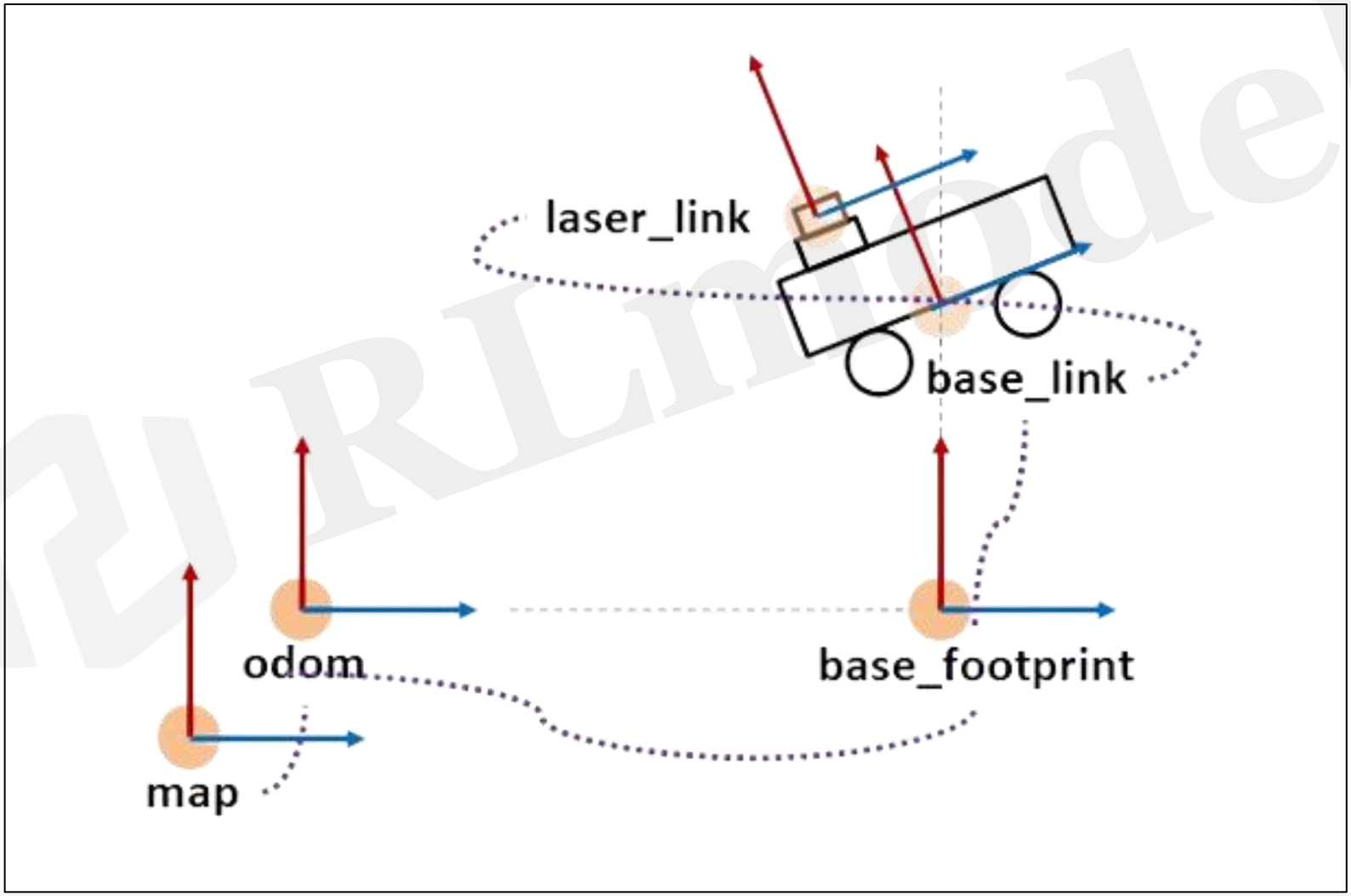
카메라, 로봇, 센서 등의 모듈마다 프레임이 존재하고 각 프레임간의 관계를 알아낼 수 있는 라이브러리

사용/설정: urdf ,혹은 broadcaster



Transform

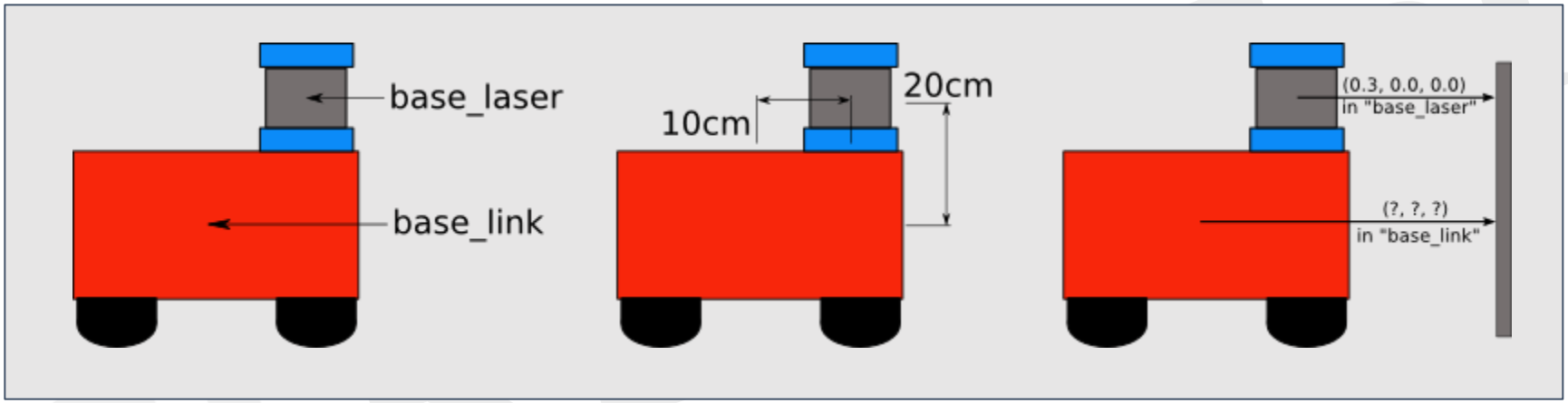
robot tf 구성 예시

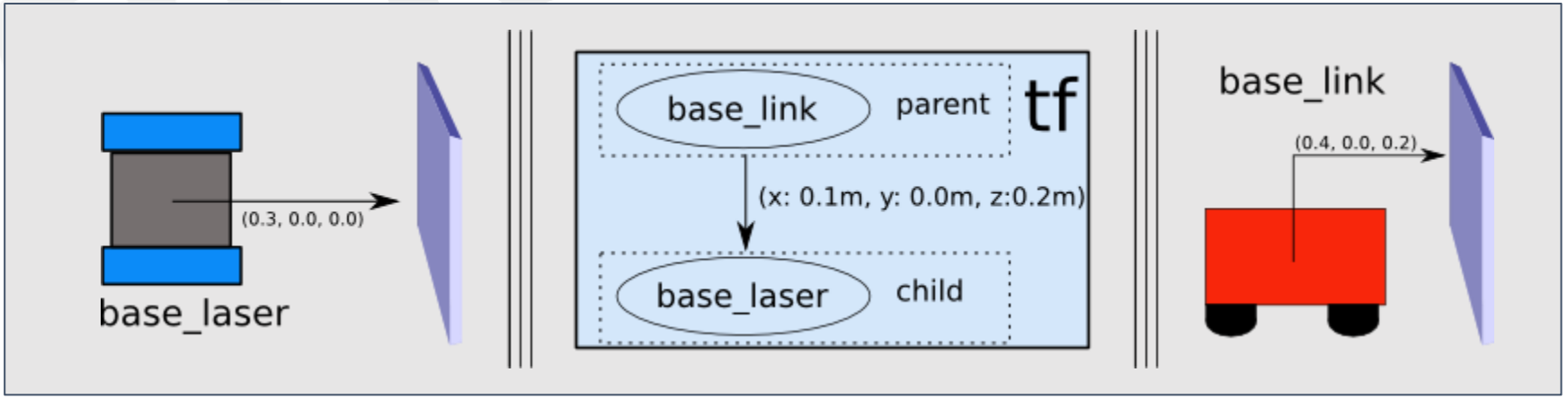


Transform

tf2 란

예시: rviz2

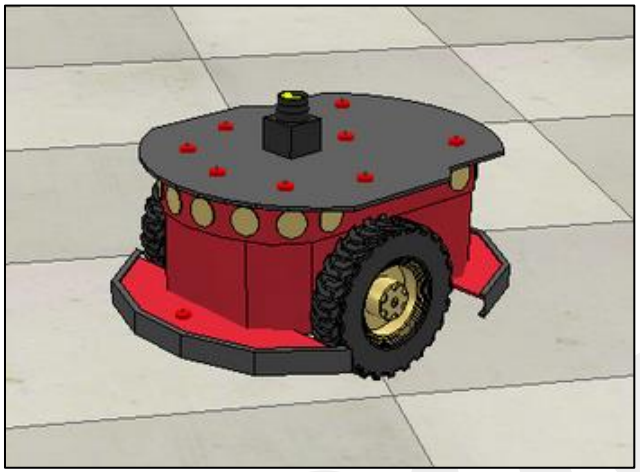


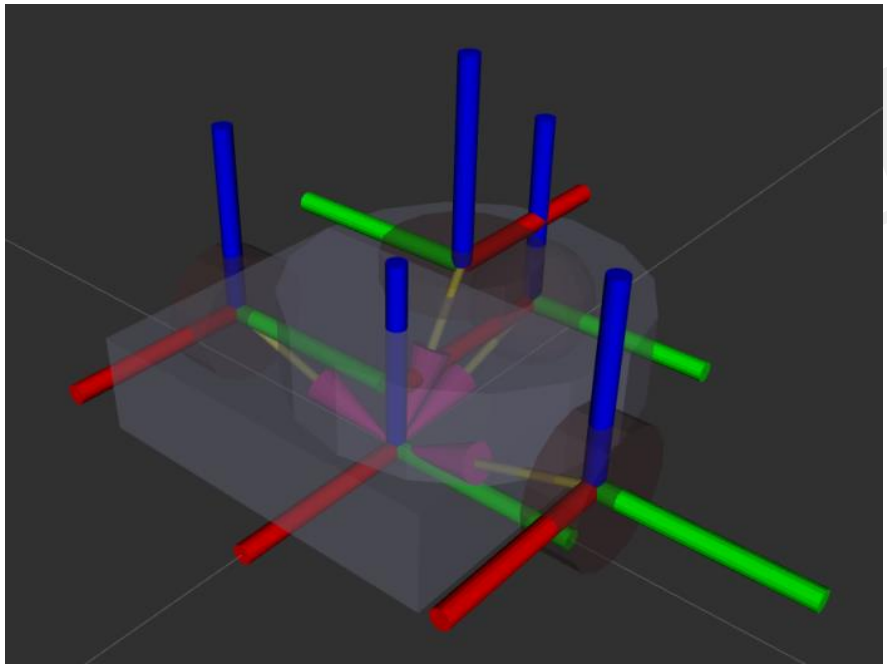


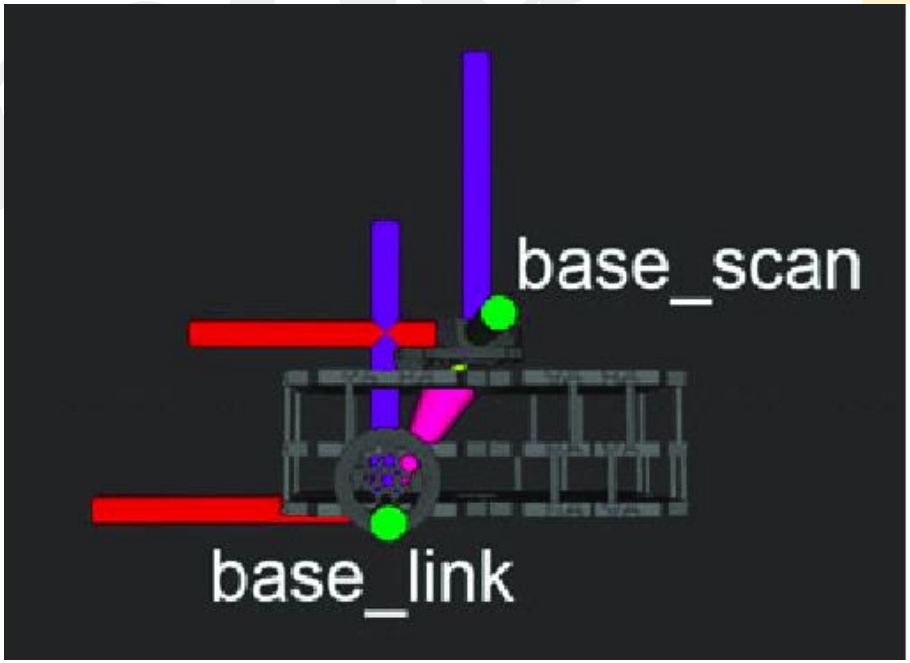
Transform

tf2 시각화

예시: rviz2



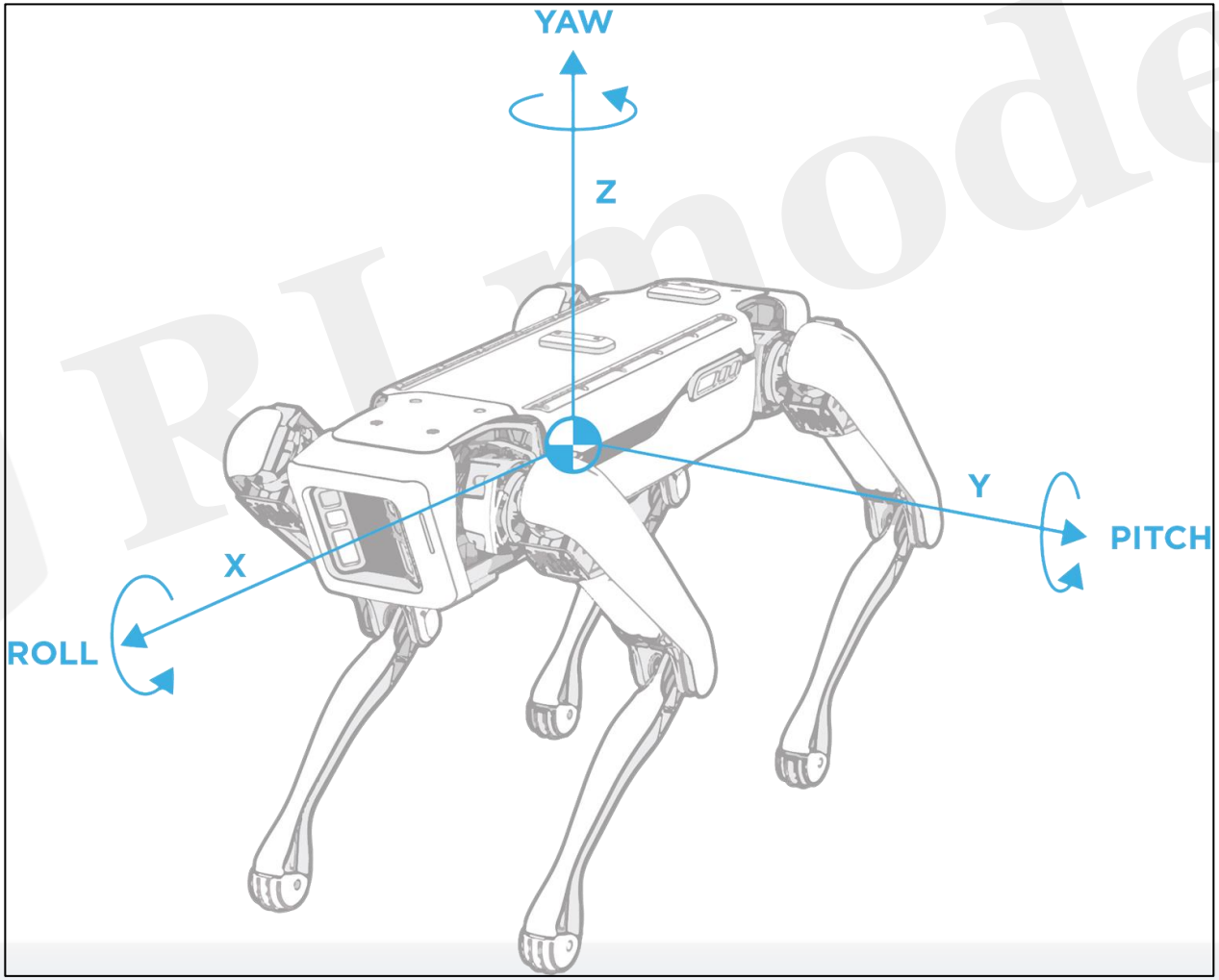




Transform

tf2 시각화

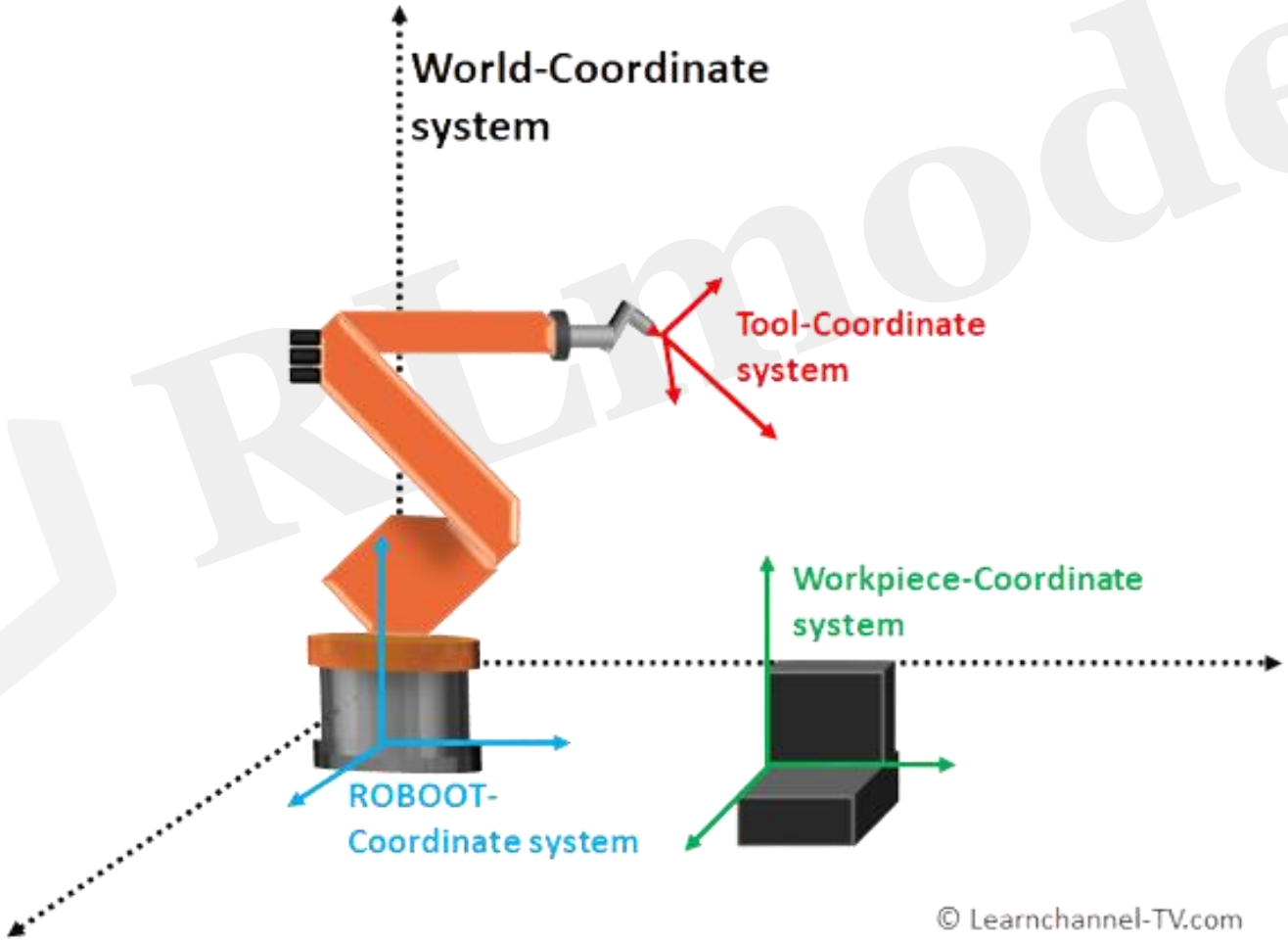
예시: robot dog



Transform

manipulator tf

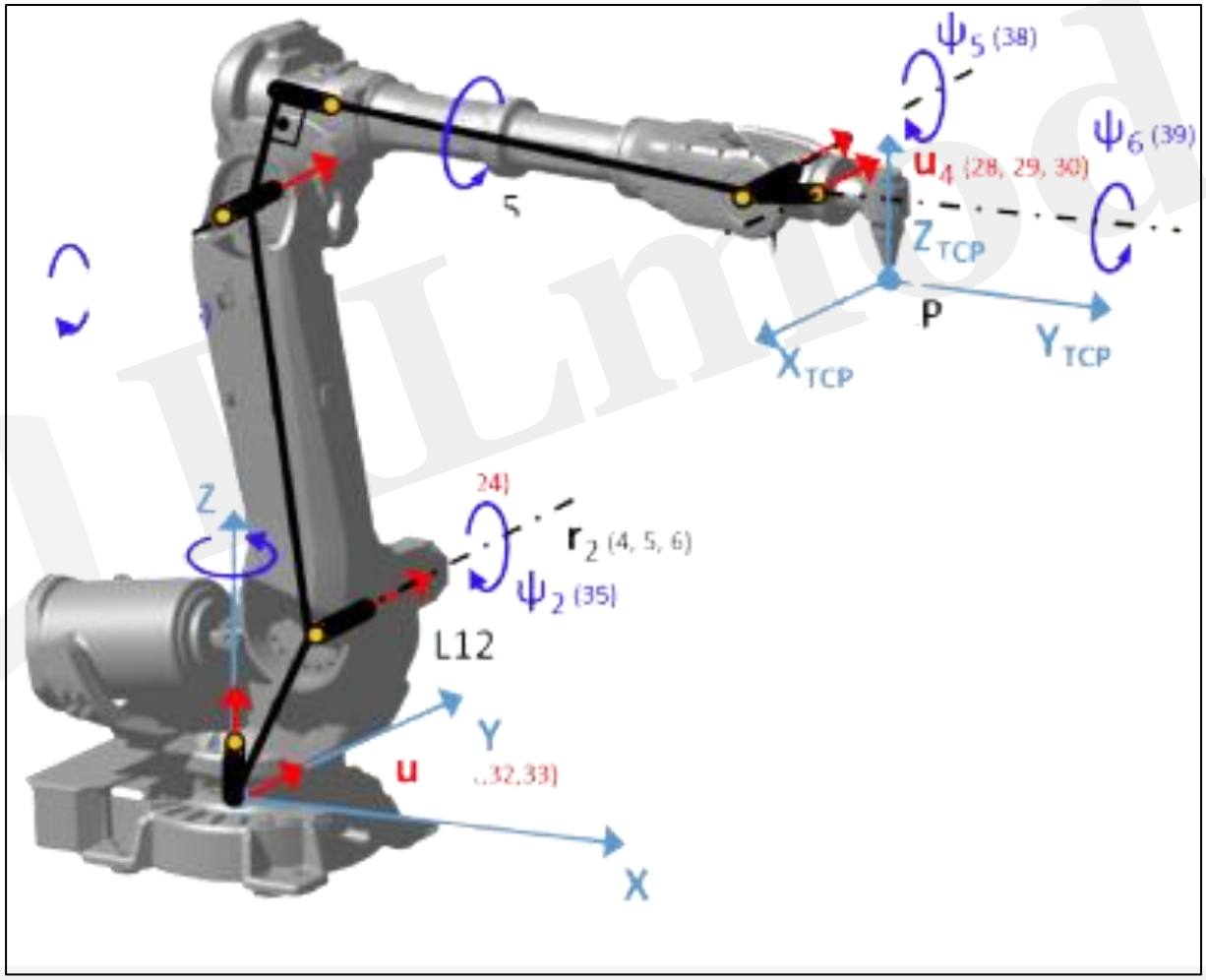
예시: robot arm DOF



Transform

manipulator tf

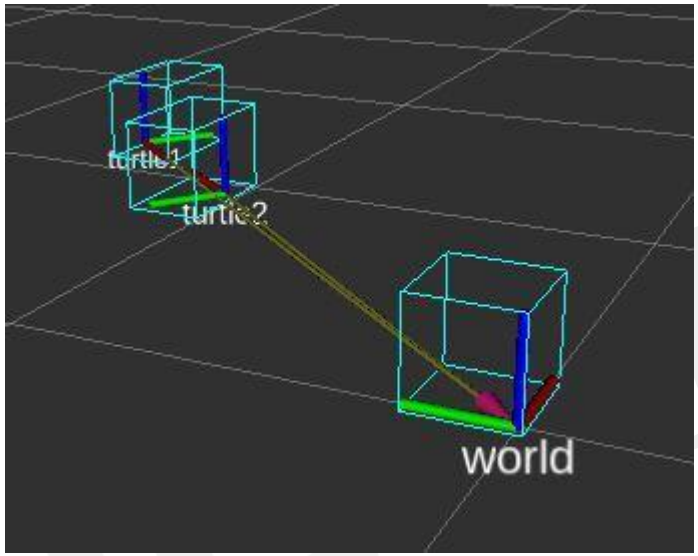
예시: 6 DOF(Degree Of Freedom) robot arm

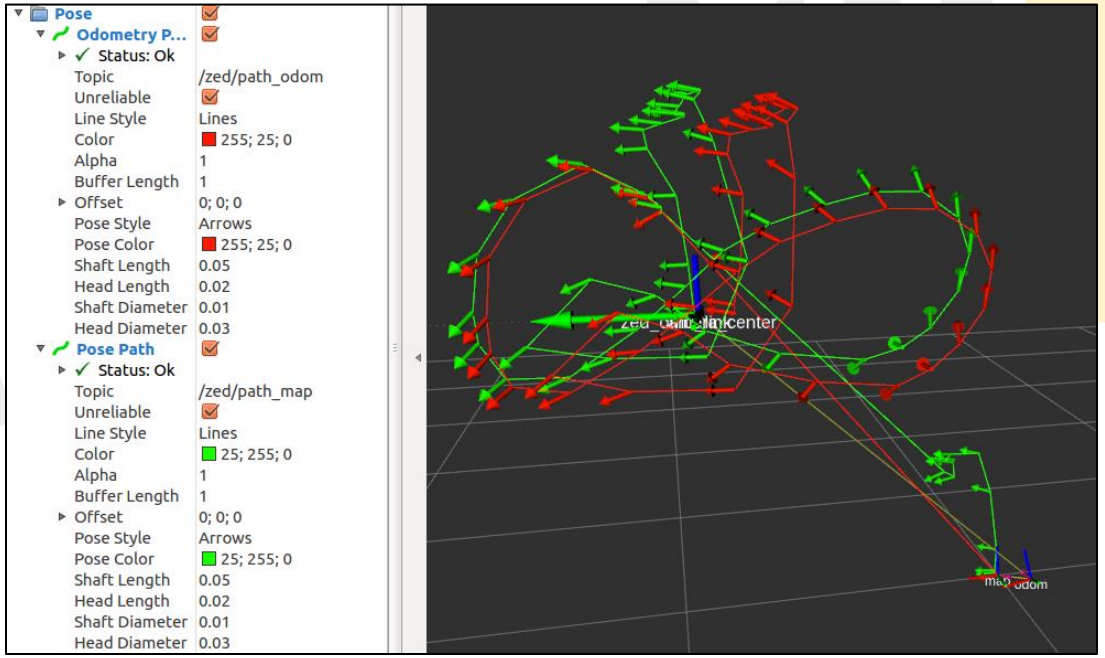


Transform

tf2 표시

> rviz 상 시각화 상태



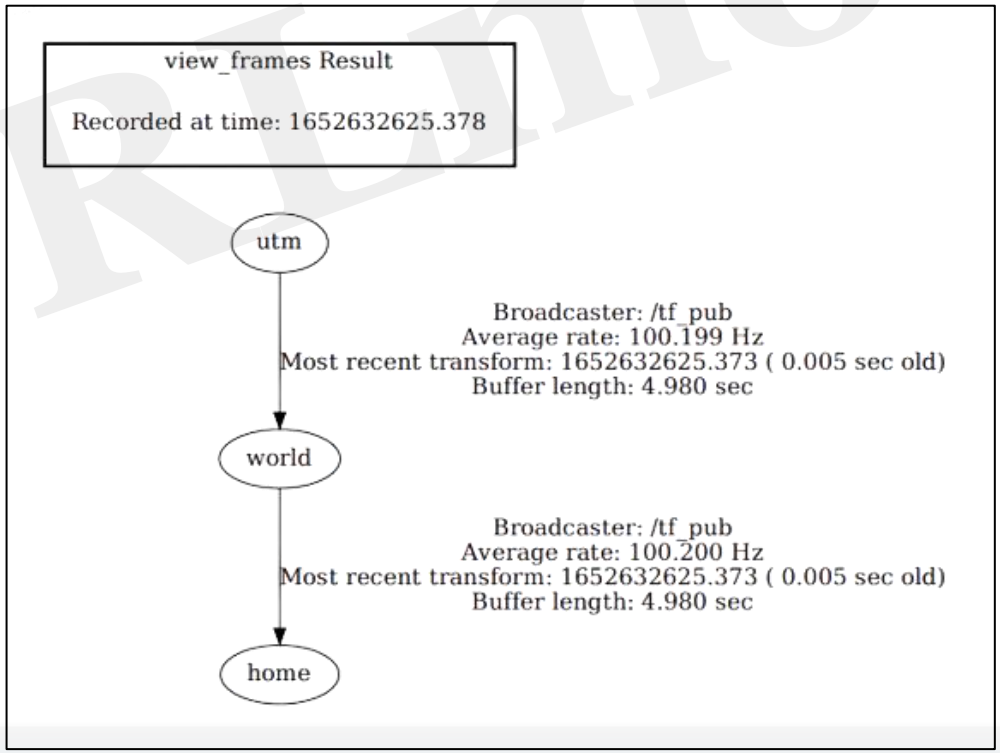


Transform

frame 이란

> frame은 ROS 상에 존재하는 수많은 좌표계를 의미하며, TF는 각 프레임 사이의 관계를 정의하는 것

- 모든 ROS의 프레임은 유클리드 좌표계로 구성
- 각각의 프레임은 서로 독립된 상태
- frame_id 를 통하여 각각의 좌표계를 서로 구분.



Transform

정리

map: world fixed frame(세계 고정 좌표 프레임)이다. Z축은 위를 향한다.

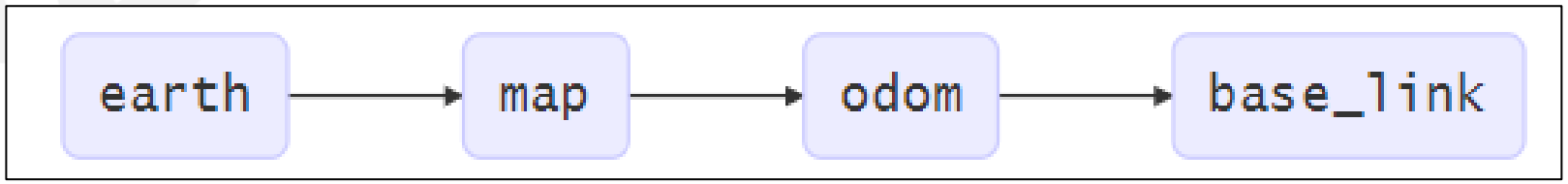
odom: 로봇의 Odometry(오도메트리) 정보를 기준으로 정의된 좌표계

base_footprint: base_footprint는 바닥에 대한 로봇 위치

base_link: base_link는 모바일 로봇 베이스에 부착.

gaze: head 위치 및 방향을 정의한 좌표 프레임이다. 카메라같은 센서와 관련해 보는 방향을 정의한다.

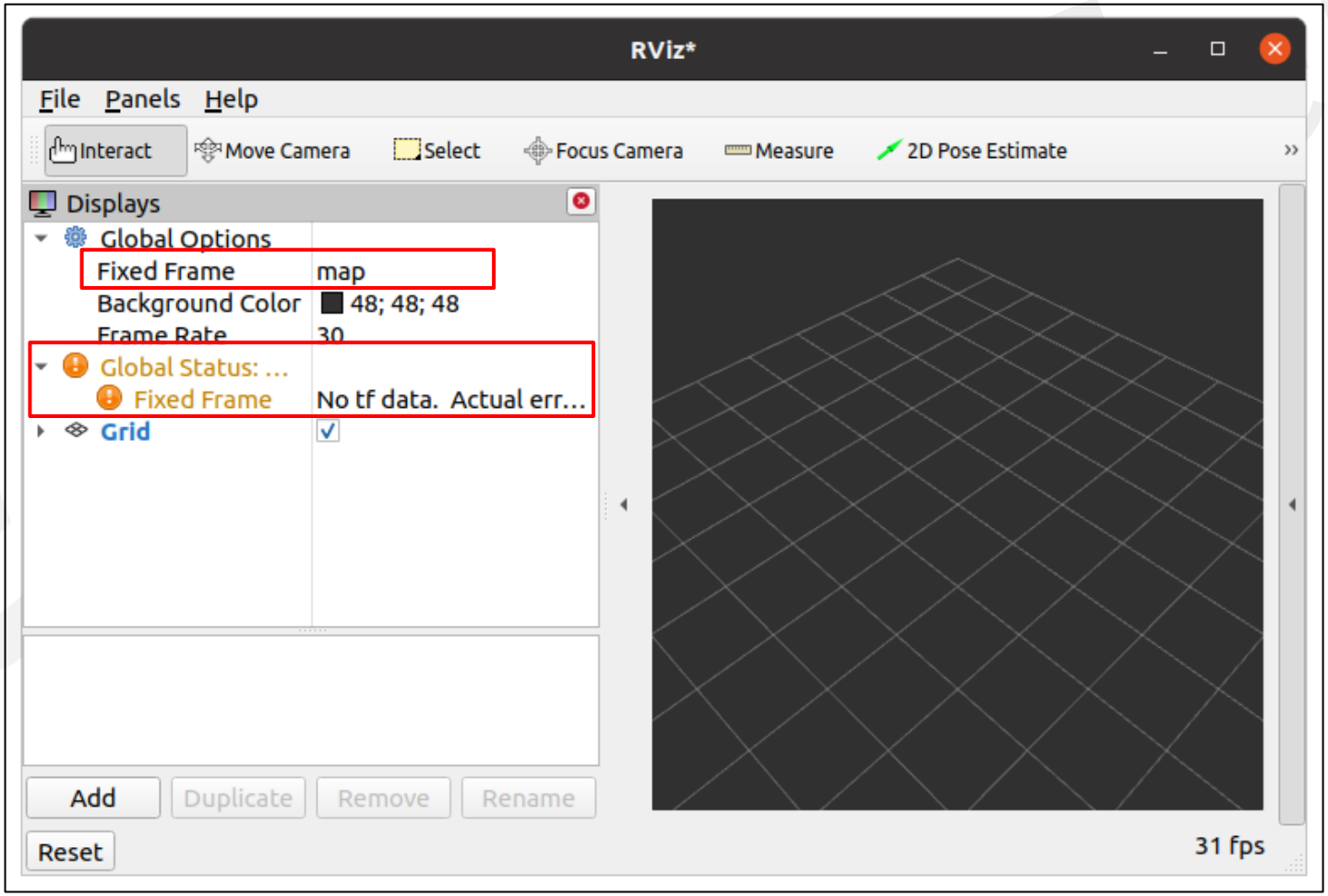
earth: ECEF의 원점이다. 이 프레임은 서로 다른 맵 프레임에서 여러 로봇의 상호 작용을 허용하도록 설계되었다.



Transform

Rviz

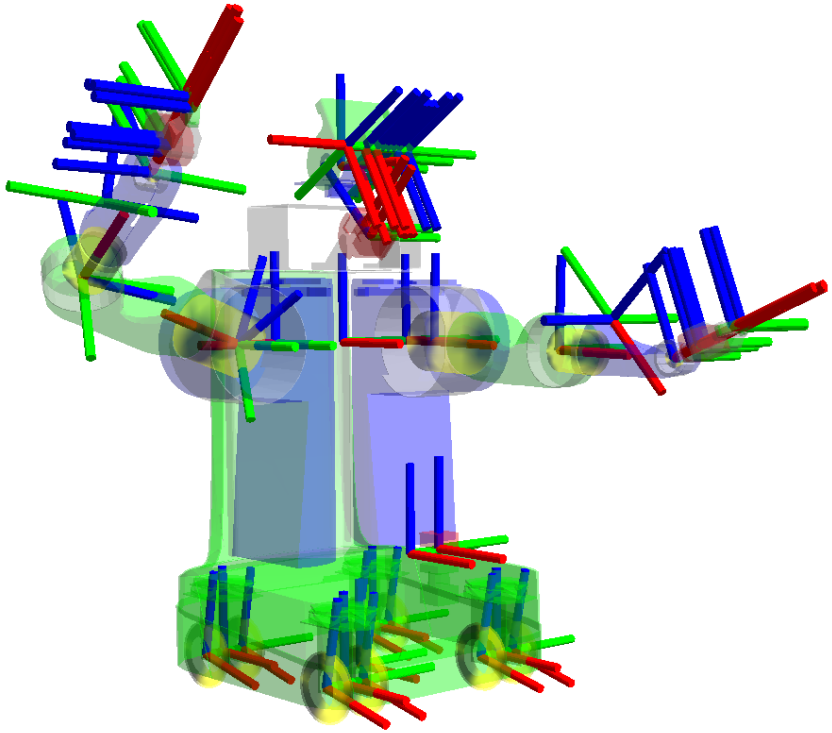
Fixed Frame: 기준 좌표계



Transform

기본설명

static

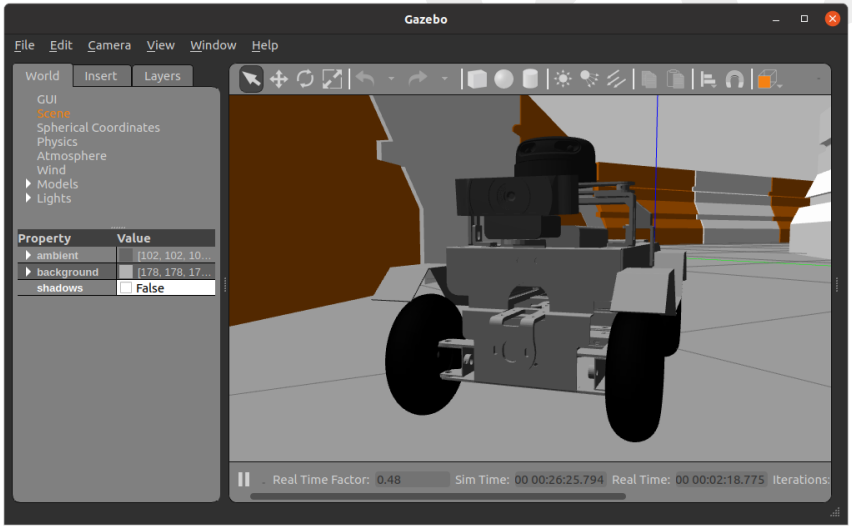


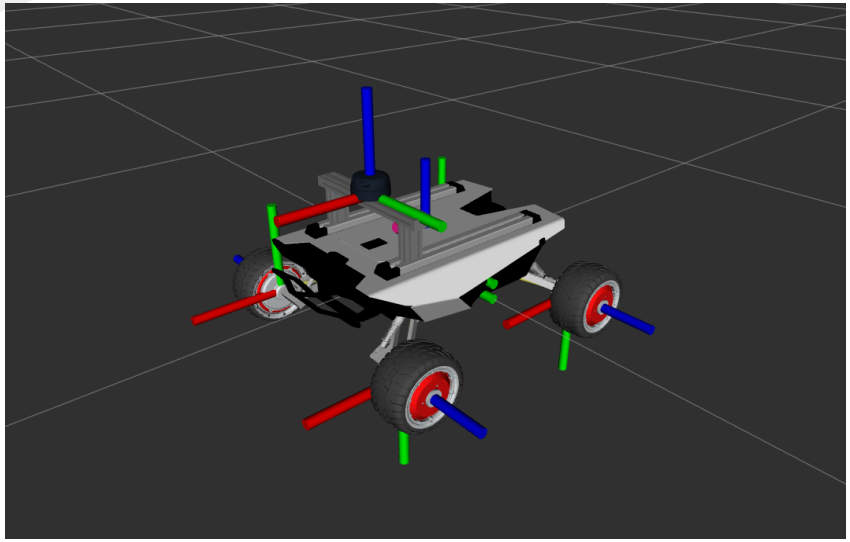
URDF	TF
A tree of...	A tree of...
Links	Frames
connected by	connected by
Joints	Transforms

URDF

URDF 란

- 설명: URDF는 Universal Robot Description Format의 약자로, 로봇의 물리적 구조를 정의하기 위한 XML 형식
- 목적: 로봇의 각 부위와 이들 간의 관계를 정의하여 시뮬레이션, 시각화 및 제어에 사용됩니다.
- ROS에서의 역할:ROS에서 로봇의 모델을 정의하고 이를 기반으로 시뮬레이션 및 실제 로봇 제어를 수행할 때 사용됩니다.





URDF

URDF 파일 구조

❖ URDF 파일은 여러 개의 태그로 구성되며, 주요 태그는 다음과 같습니다.

- **<robot>**: URDF 파일의 루트 태그로, 로봇 전체를 감싸는 태그
- **<link>**: 로봇의 각 부위를 정의
- **<joint>**: 로봇 부위 간의 연결을 정의합니다

URDF

URDF 파일 구조

1. <robot> 태그
- 로봇의 링크, 조인트를 포함하여 여러가지 태그를 포함합니다.

xml 코드 복사

```xml
<robot name="my_robot">
  <!-- 로봇의 링크와 조인트가 여기에 정의됩니다 -->
</robot>
```

URDF

URDF 파일 구조

2. <link> 태그

- **기본 구조**: 로봇의 각 부위를 나타내며, 각 부위는 시각적 모델, 충돌 모델, 관성 정보 등을 포함할 수 있습니다.

xml 코드 복사

```xml
<link name="base_link">
  <visual>
    <geometry>
      <box size="1 1 1"/>
    </geometry>
  </visual>
</link>
```

URDF

URDF 파일 구조

3. <joint> 태그

- **기본 구조**: 두 링크를 연결하고, 연결된 링크 간의 상대적인 움직임을 정의합니다.

xml 코드 복사

```xml
<joint name="base_to_link1" type="continuous">
  <parent link="base_link"/>
  <child link="link1"/>
  <origin xyz="0 0 1" rpy="0 0 0"/>
</joint>
```

URDF

URDF 작성 예

Release Notes: 1.89.0 | bunker_pro.urdf.xacro

bunker_bringup > urdf > bunker_pro.urdf.xacro

```
2   <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="urdf_tutorial">
19      <link name="base_link">
20          <visual>
21              <origin xyz="0 0 0"/>
22              <geometry>
23                  <box size="1.04 0.85 0.45"/>
24              </geometry>
25              <material name="white"/>
26          </visual>
27          <collision>
28              <origin xyz="0 0 0"/>
29              <geometry>
30                  <box size="1.04 0.85 0.45"/>
31              </geometry>
32          </collision>
33      </link>
34
35      <!-- LEFT WHEEL LINK -->
36
37      <link name="left_wheel">
38          <visual>
39              <geometry>
40                  <cylinder radius="0.06" length="0.06"/>
41              </geometry>
42              <material name="black"/>
```

URDF

URDF vs TF

1. **목적과 사용 분야:**

- **URDF**: 로봇의 물리적 구조를 정의하는데 중점을 둡니다. 로봇의 각 부위와 이들 간의 관계를 기술하여 시뮬레이션, 시각화, 제어에 사용됩니다.

- **TF**: 로봇의 각 부위 간의 좌표 변환을 실시간으로 관리합니다. 로봇의 센서 데이터 처리, 로봇 팔의 이동 경로 계산 등에 사용됩니다

1. **형식과 구성 요소:**

- **URDF**: XML 형식으로 링크와 조인트를 정의합니다. 각 링크의 시각적 모델, 충돌 모델, 관성 정보 등을 포함합니다.
- **TF**: ROS의 TF 라이브러리 및 메시지 시스템을 사용합니다. 좌표 변환과 관련된 정보를 트리 구조로 관리합니다.

URDF

URDF vs TF

3. 데이터의 동적 관리:

- **URDF**: 주로 정적 데이터(로봇의 물리적 구조)를 정의하며, 런타임 동안 변하지 않습니다.

- **TF**: 동적 데이터(시간에 따라 변하는 좌표 변환)를 관리하며, 런타임 동안 지속적으로 업데이트됩니다.

4. 형식과 구성 요소:

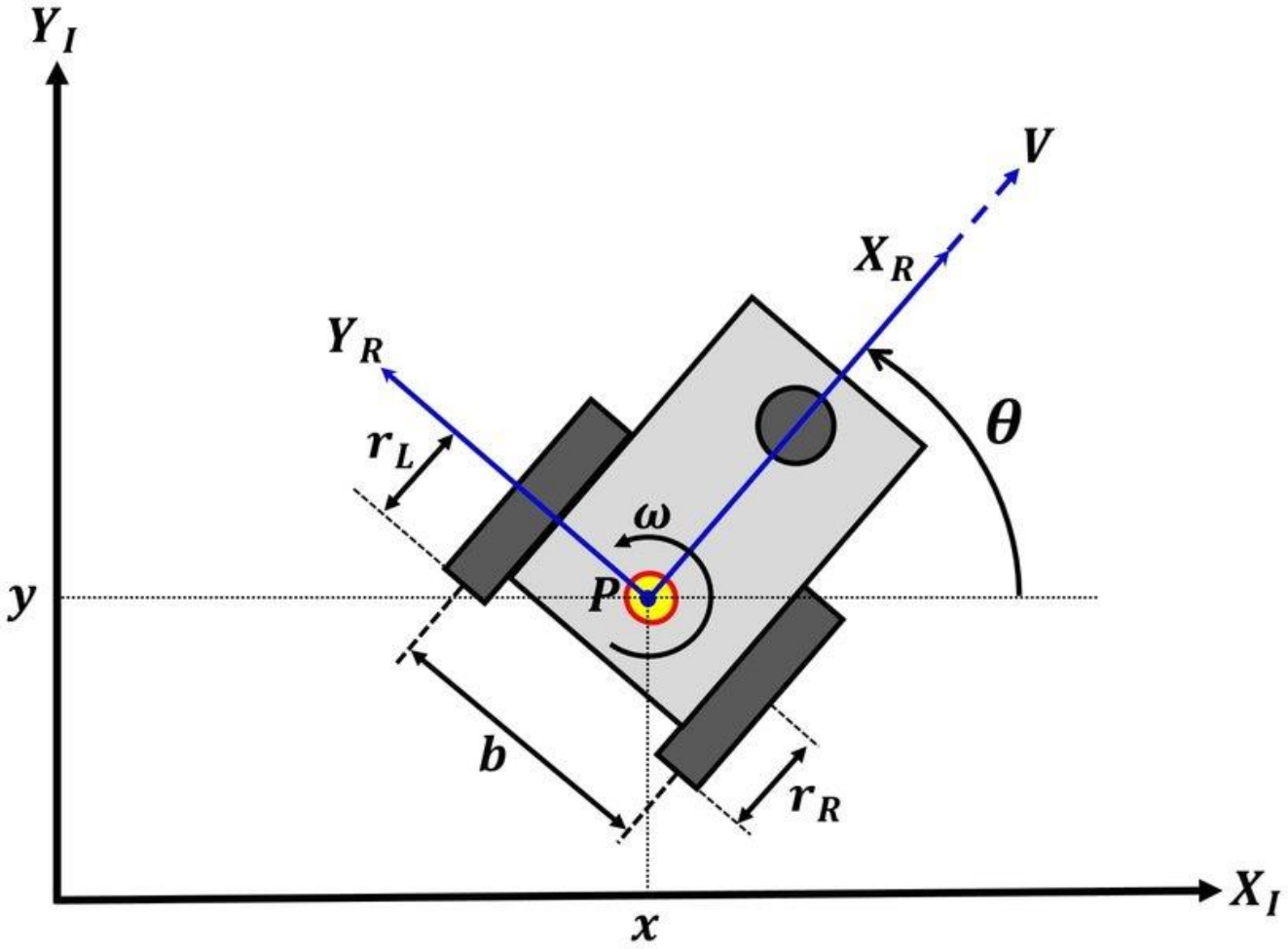
- **URDF**: RViz, Gazebo 등의 시각화 및 시뮬레이션 도구에서 로봇 모델을 정의하는데 사용됩니다.

- **TF**: RViz에서 좌표 변환을 시각화하거나, ROS 시스템에서 로봇의 동작을 추적하고 제어하는 데 사용됩니다.

ROS2 구성

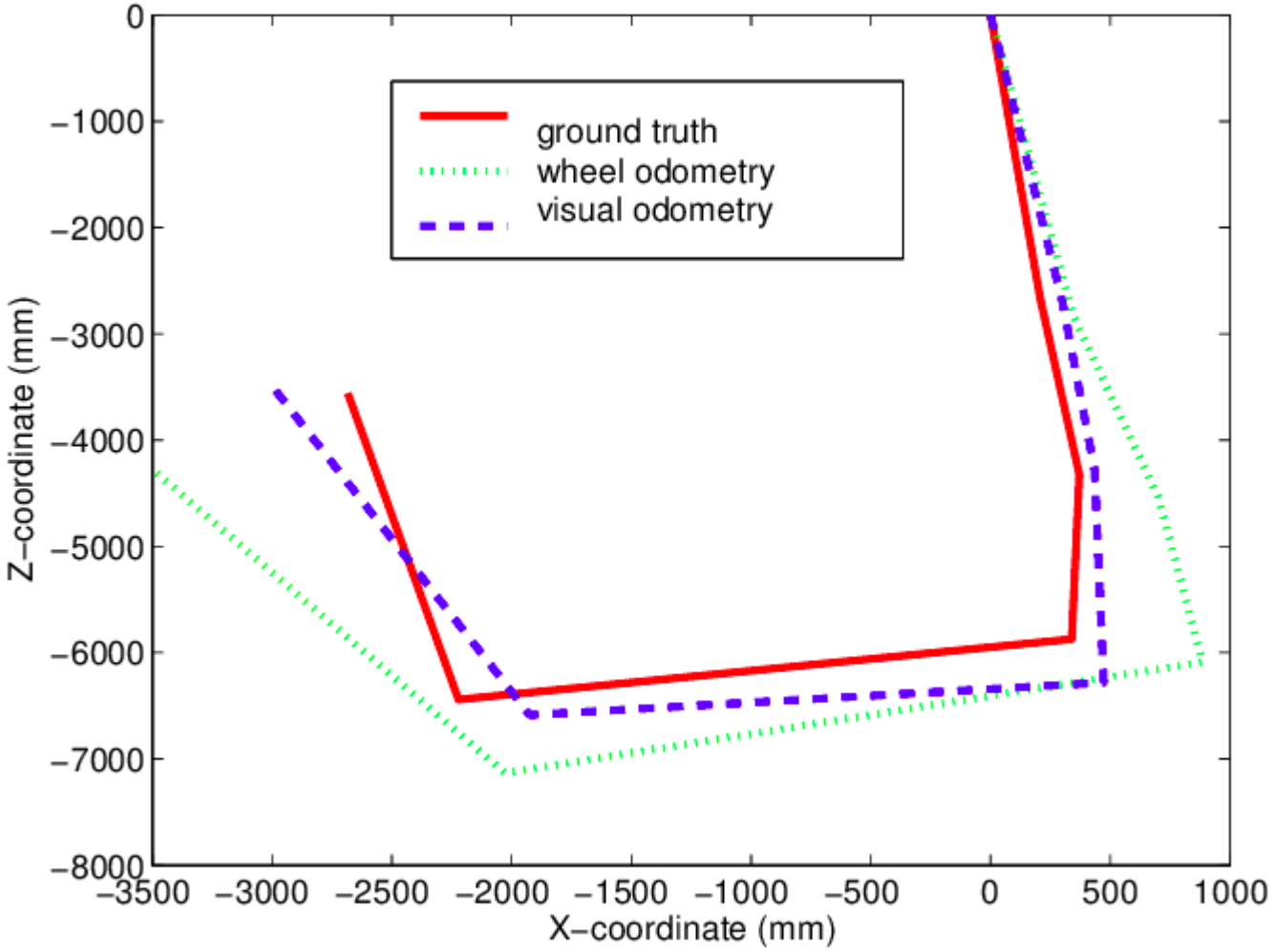
Odometry 란

odom: 로봇의 Odometry(오도메트리) 정보를 기준으로 정의된 좌표계



ROS2 구성

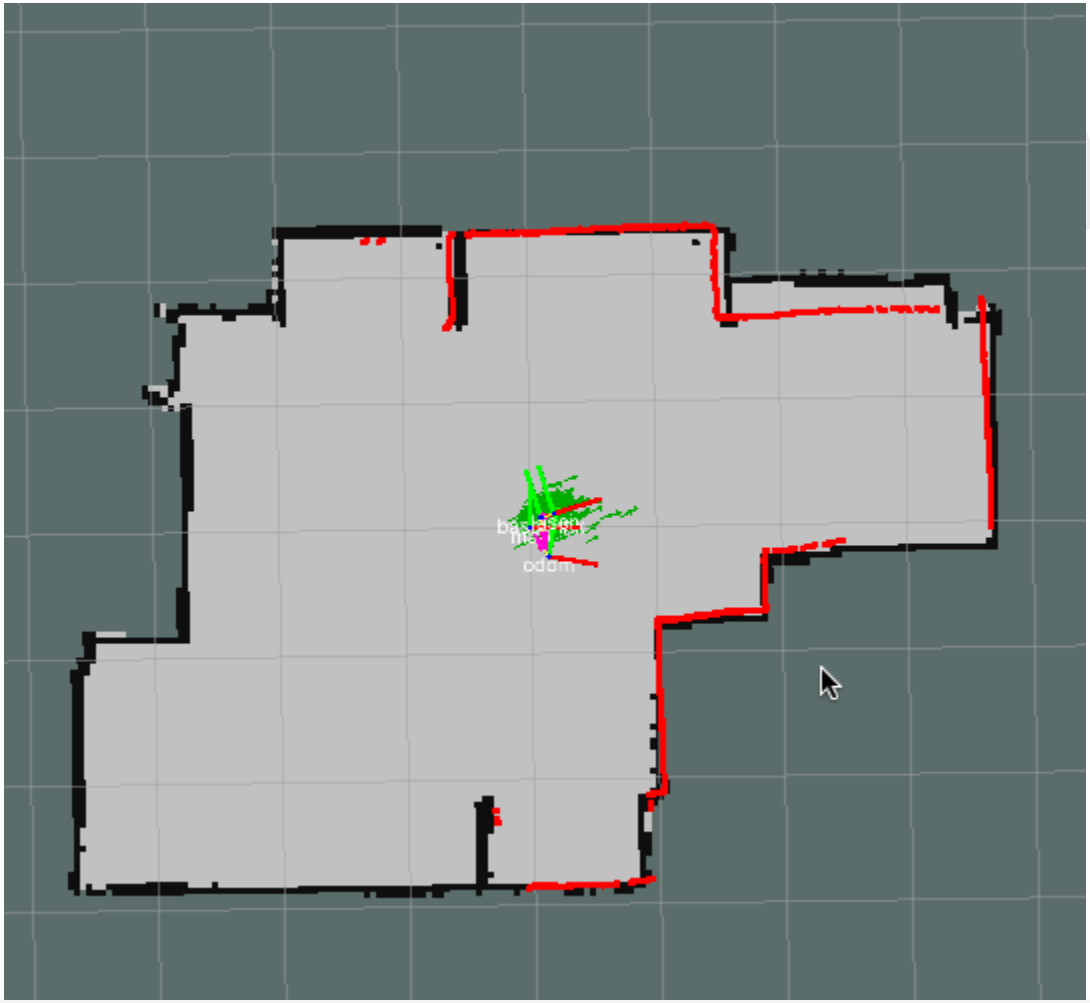
Odometry 비교



ROS2 구성

Odometry 계산표시

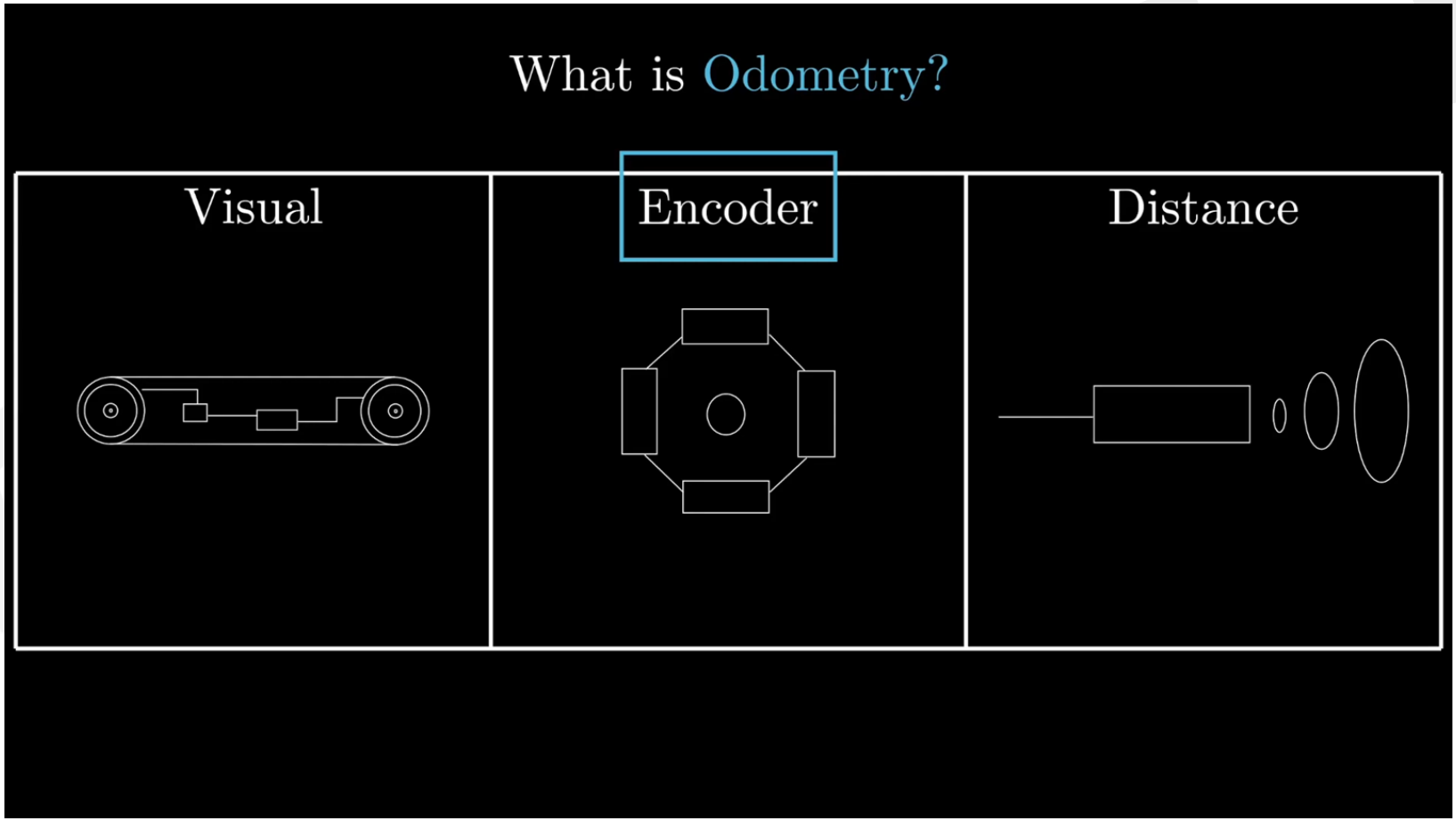
robot localization and AMCL



ROS2 구성

Odometry 란

link: https://www.youtube.com/watch?v=ZW7T6EFyYnc&t=19s

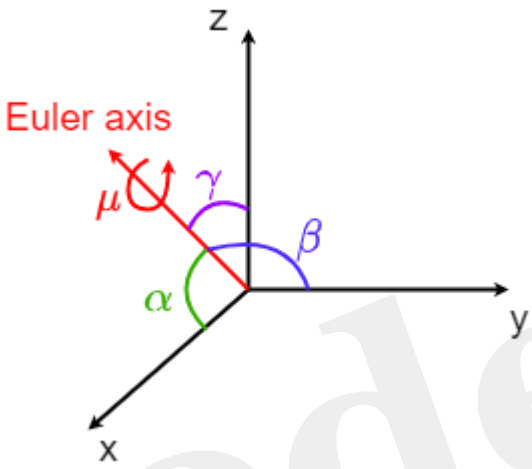


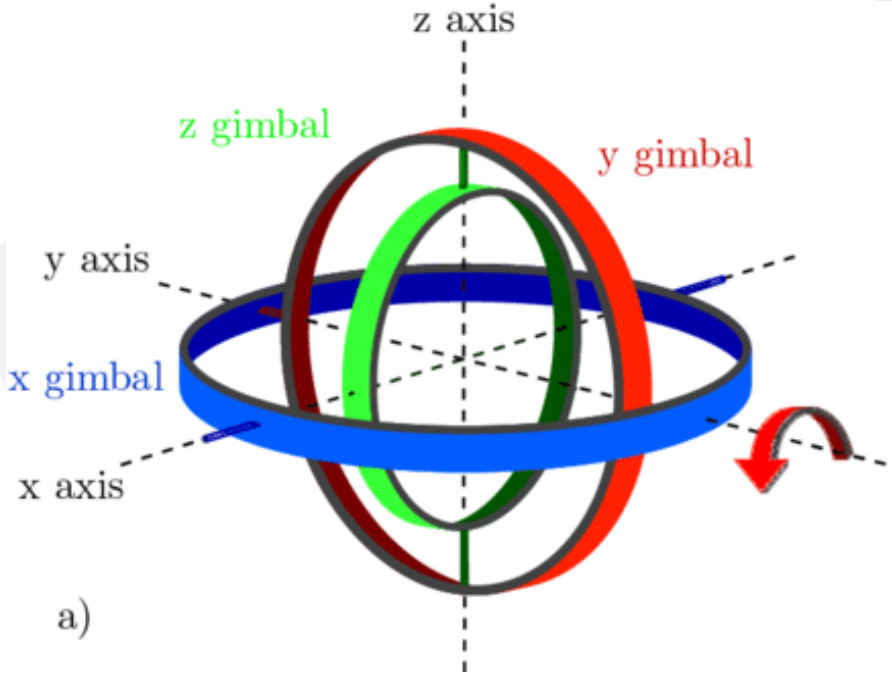
ROS2 구성

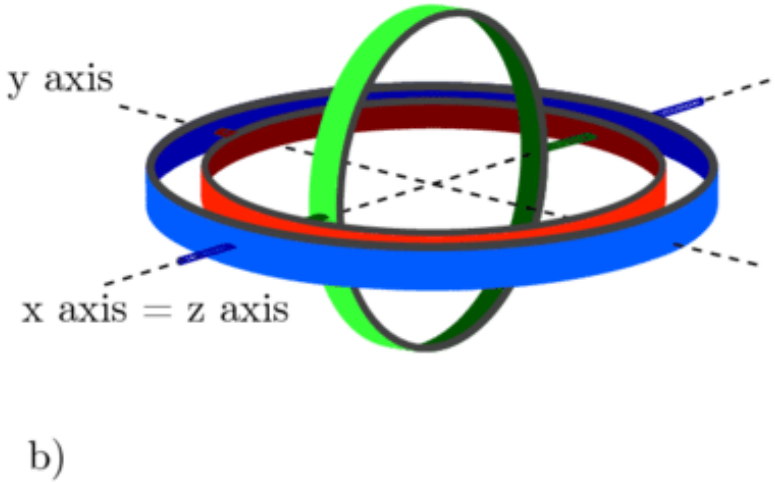
tf2 참고

Euler vs **Quaternion**

>Gimbal lock





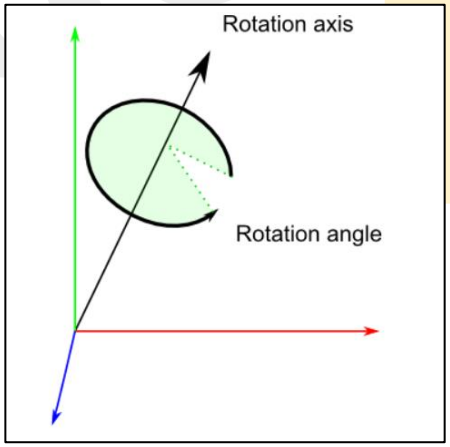


ROS2 구성

tf2 참고

Quaternion

>쿼터니언은 본래 아일랜드의 수학자 해밀턴(William R. Hamilton)이 1843년에 복소수를 확장시켜 만든 수체계(number system)다. 복소수가 2차원 평면상의 점을 표현할 수 있다는 사실로부터 3차원 공간상의 점을 표현하는 방법을 찾다가 만들었다고 한다



쿼터니언은 x, y, z, w **4개의 성분**으로 구성됩니다.

각 성분은 x, y, z 벡터와 스칼라입니다.

쿼터니언은 모든 축을 한 번에 계산하기 때문에 짐벌락 문제가 생기지 않으며

방향과 회전을 모두 표현할 수 있습니다

$$R_x(\phi) = \mathrm{Roll}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$R_y(\theta) = \mathrm{Pitch}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

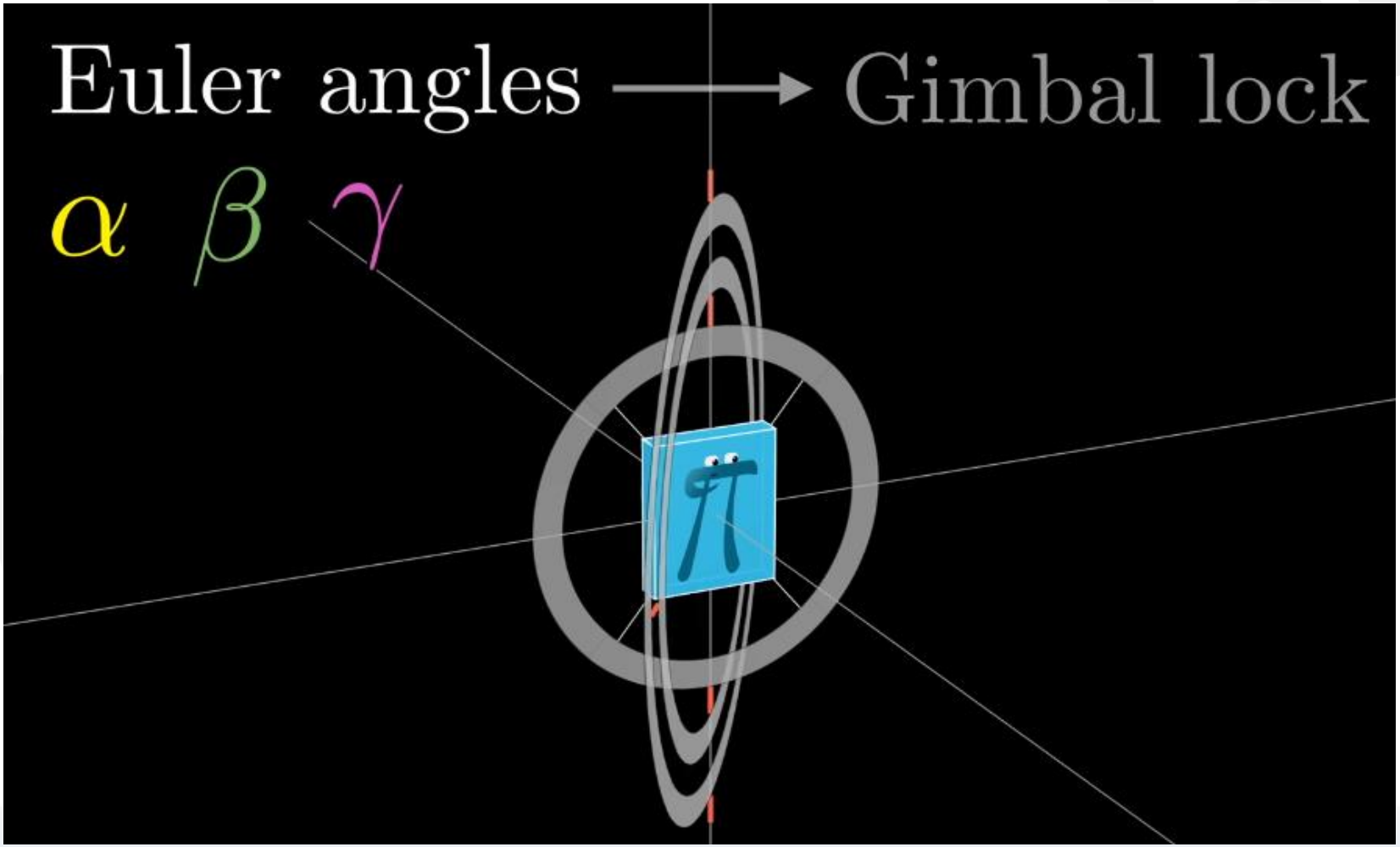
$$R_z(\psi) = \mathrm{Yaw}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{cases} q_0 = \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ q_1 = \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ q_2 = \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ q_3 = \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{cases}$$

ROS2 구성

tf2 참고

link: https://www.youtube.com/watch?v=zjMuIxRvygQ



RQT

rqt 정의

RQt는 ROS + Qt의 합성어, 일종의 Framework

용도

ROS 환경에서 데이터를 쉽게 확인, 관리할 수 있는 모니터링 도구

특징

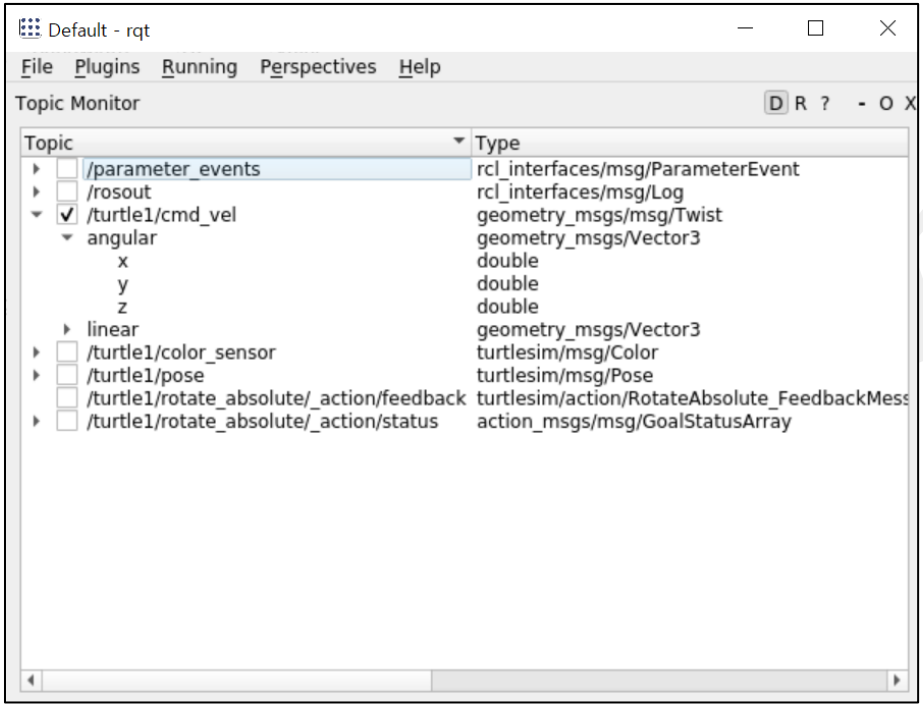
ROS 환경에서 사용할 수 있는 GUI 개발에 있어서 공통으로 필요한 부분들을 API형태로 제공

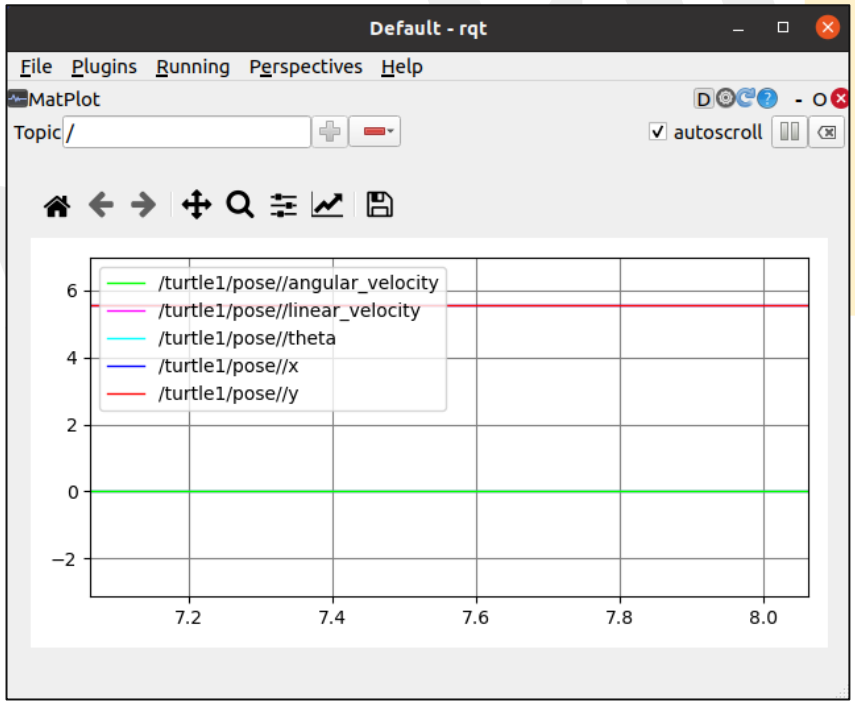
Qt의 특징인 크로스 플랫폼과 다양한 프로그래밍 언어 지원이라는 특징을 그대로 반영.

RQT

rqt 실행

$ rqt



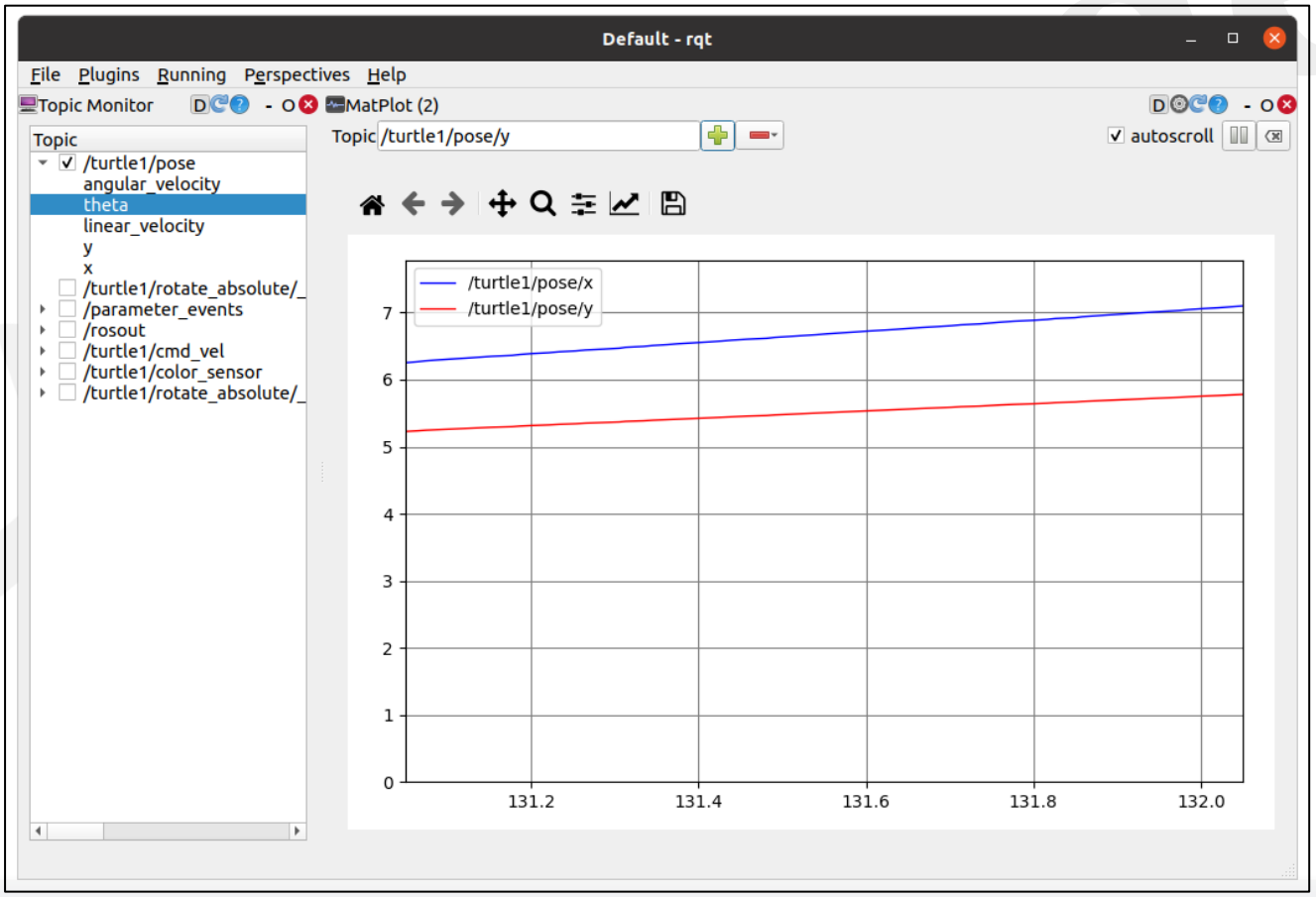


RQT

rqt 실행

ex) turtle pose x/y >

Topic: /turtle1/pose/y

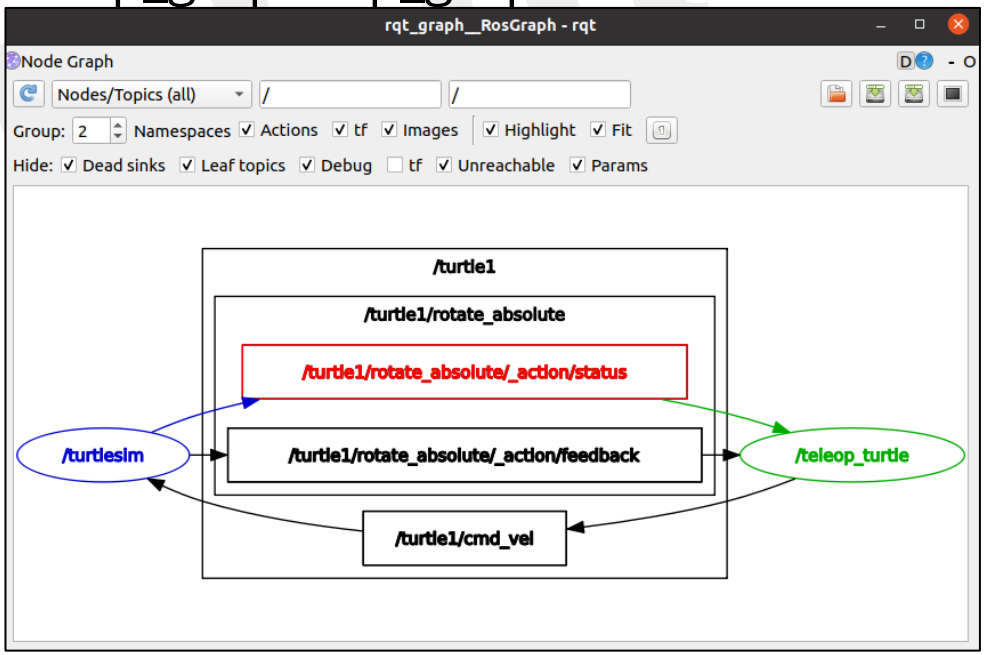


rqt_graph

rqt_graph 란

rqt_graph는 로봇 시스템의 통신 인프라를 시각화하기 위해 ROS(로봇 운영 체제) 프레임워크에서 사용되는 강력한 그래픽 도구입니다. ROS 네트워크 내에서 서로 다른 노드와 주제 간의 관계를 실시간으로 표현

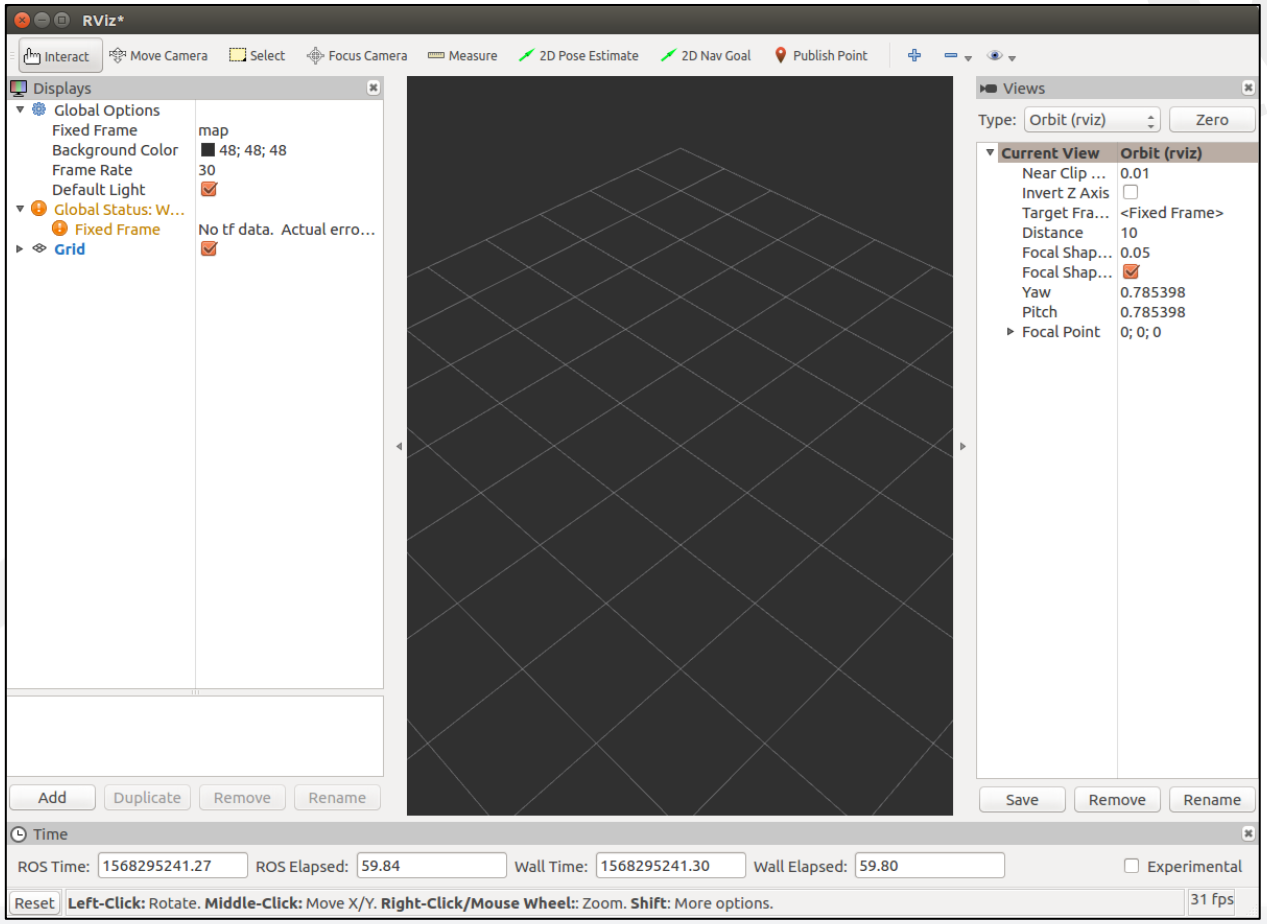
명령어: ros2 run rqt_graph rqt_graph



rviz 소개

rviz 란

ROS Visualization Tool의 약자로 ROS 도구 중 하나다. 센서 데이터를 시각화하는 기능을 수행

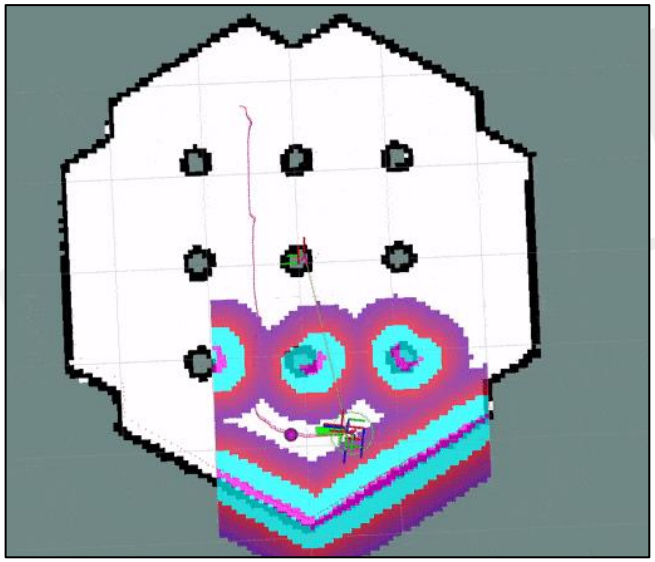


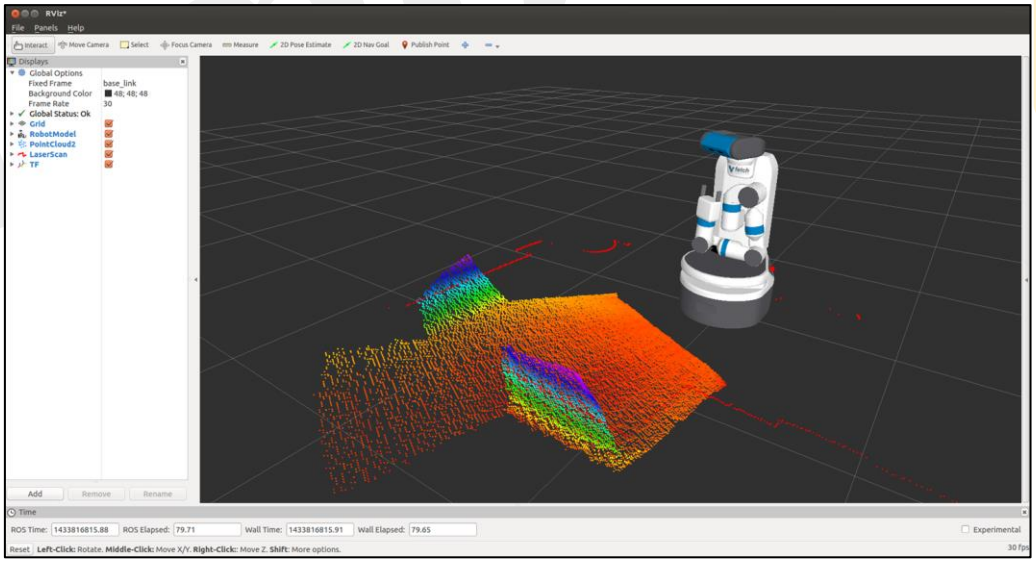
rviz 소개

rviz 란

RViz는 ROS Visualization Tool의 약자로 ROS 도구 중 하나.
센서 데이터를 시각화, 레이저 거리 센서(LDS)의 거리 데이터, 카메라의 영상 데이터 로봇 외형의 표시, 원격 제어

ex) navigation

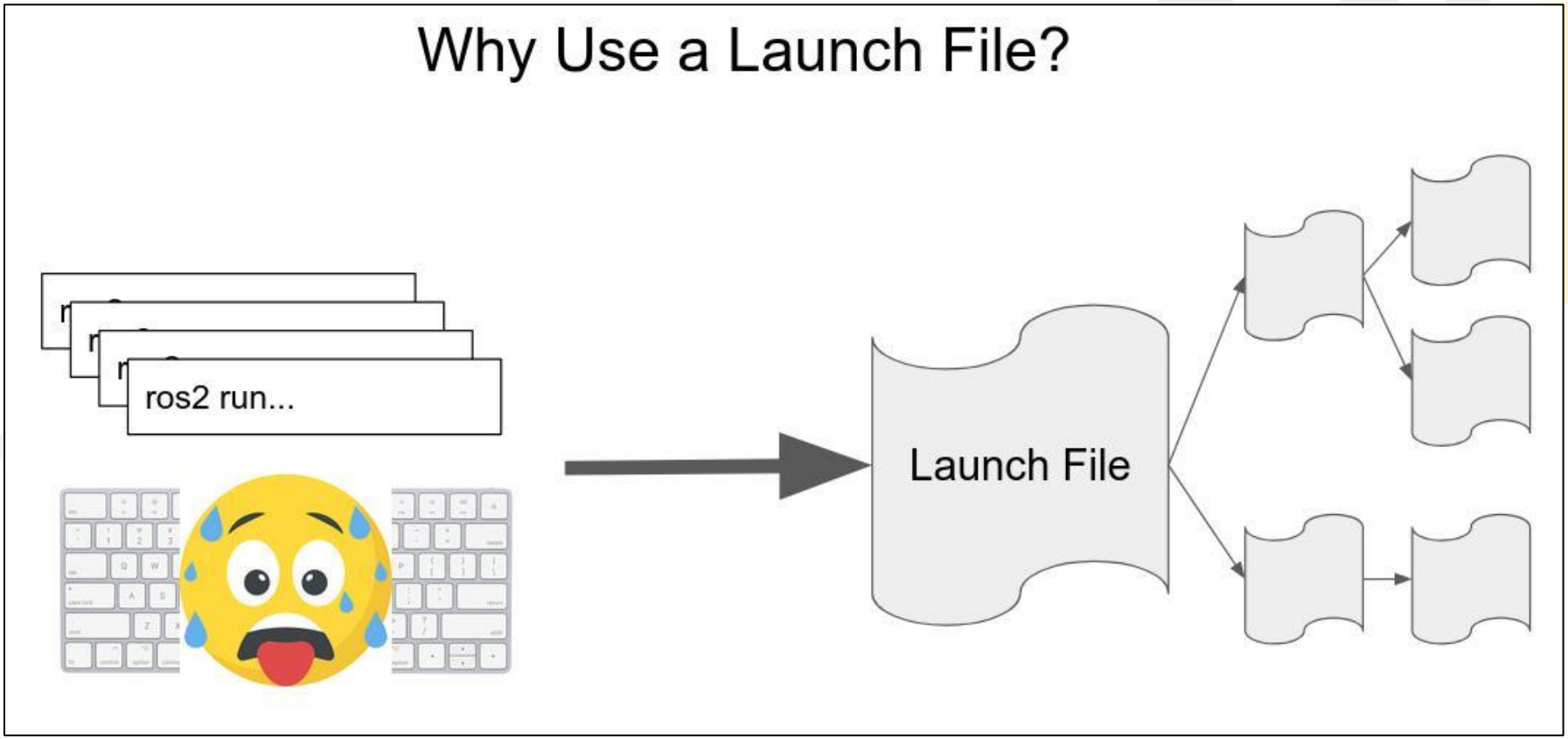




launch 파일

정의/설명

여래 개의 노드를 한번에 실행하는 .launch 확장자를 가진 파일.



launch 파일

작성방법

- python, yaml, xml

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

import os

from launch import LaunchDescription
from launch.actions import ExecuteProcess, IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration

from launch_ros.actions import Node

# this is the function launch  system will look for
def generate_launch_description():

    # create and return launch description object
    return LaunchDescription(
        [
            ExecuteProcess(
                cmd=["ros2", "run", "rviz2", "rviz2"], output="screen"
            ),
        ]
    )
```

launch 파일

부가기능

namespace

Node클래스를 사용할 때 namespace를 지정하면 모든 노드와 노드에 포함된 토픽,서비스,액션 이름을 일괄 변경 가능하다.

IncludeLaunchDescription

하나의 런치 파일로 동일 패키지의 노드 실행뿐만 아니라 다른 패키지의 노드 실행도 묶어 할 수 있어 유용하다.

launch 파일

사용방법

문법: $ ros2 launch '패키지명' '런치파일명'

예시: $

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

DDS

DDS 란

DDS(Data Distribution Service)는 OMG(http://www.omgwiki.org/dds/)에서 국제 표준으로 정한 실시간 데이터 분배 미들웨어

<특징>

- Publish/ Subscribe 구조 : 데이터를 Pub-Sub하는 구조.

- 성능 : pub/sub모델 채택으로 request/reply 모델에 비해 지연시간이 적고 단위 시간당 데이터 처리량이 높다.

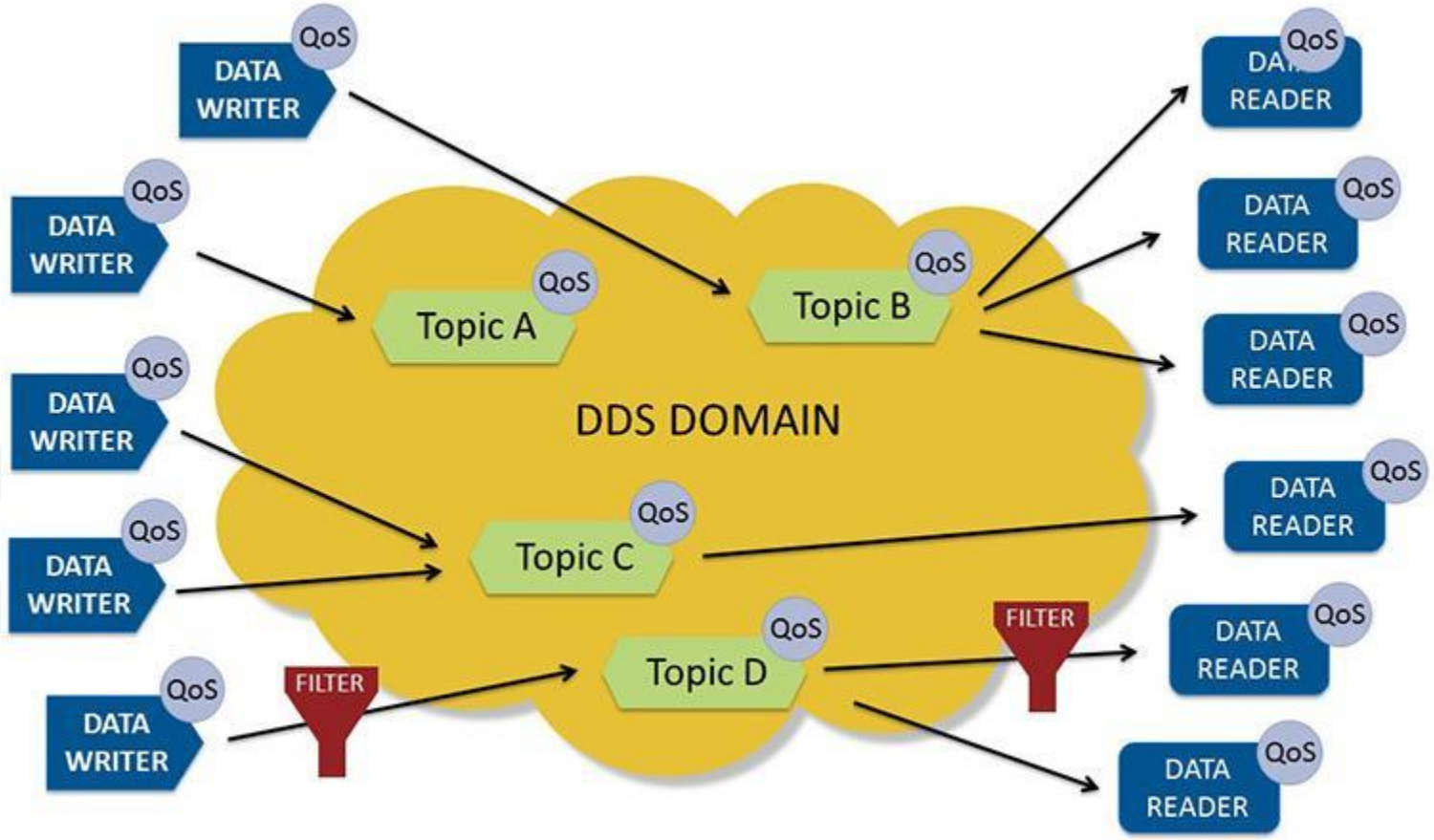
이에 리얼타임 퍼포먼스가 가능 환경.

- Automatic Discovery of remote participants(Dynamic Discovery): Remote participants 연결 용이. Discovery Protocol을 사용하여 퍼블리셔들이 서브스크라이버들의 존재를 파악하고 각 노드들이 서로 매칭합니다. >ROS2에서는 ROS Core가 사라짐

- QoS 파라미터 : reliability, persistence, redundancy, lifespan, transport settings, resources등, 통신에 있어 여러 설정값 변경 가능

DDS

ROS2 - DDS 구조



DDS

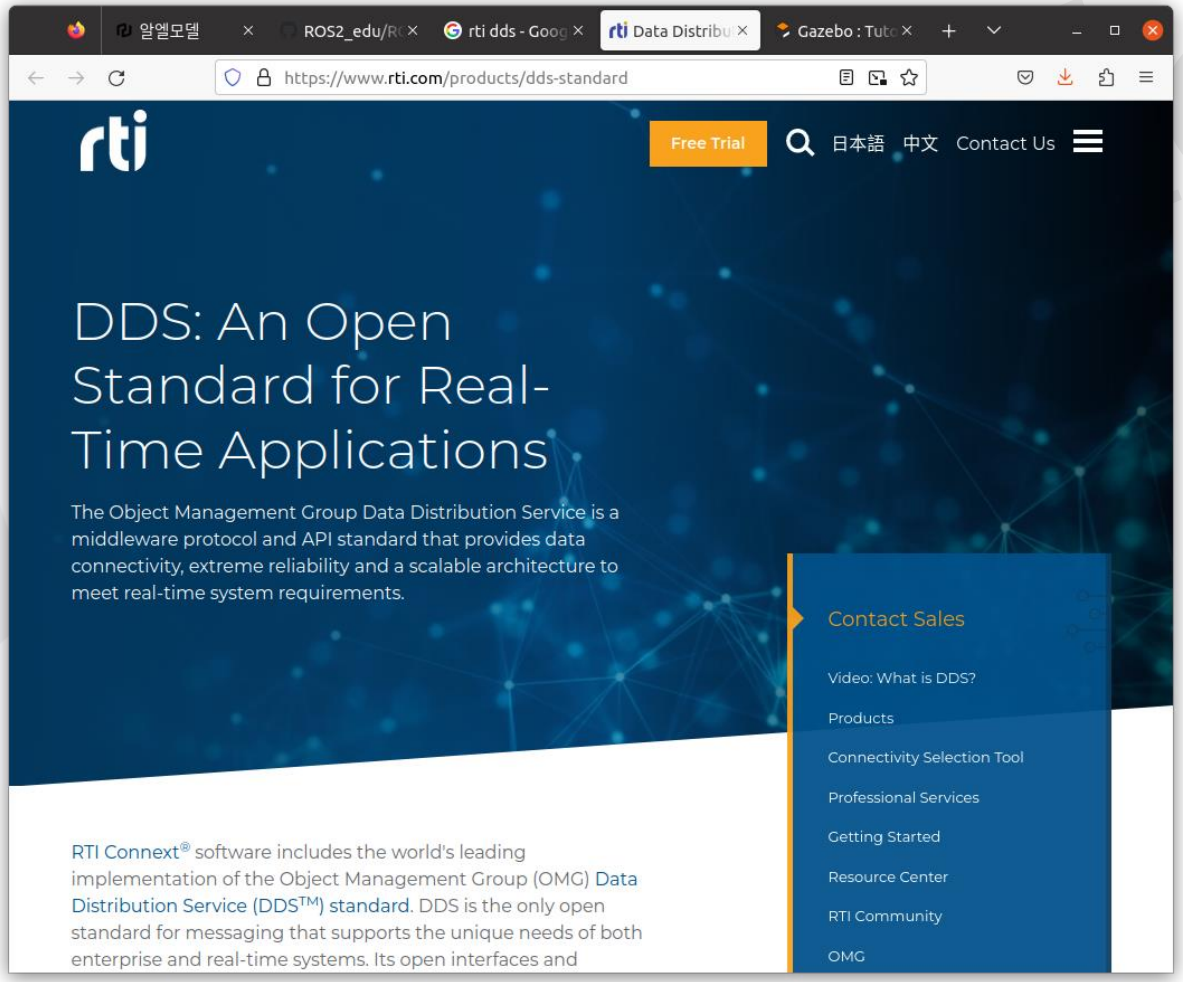
DDS 공급사

ex) 국내업체: 구름DDS

DDS

DDS 공급사

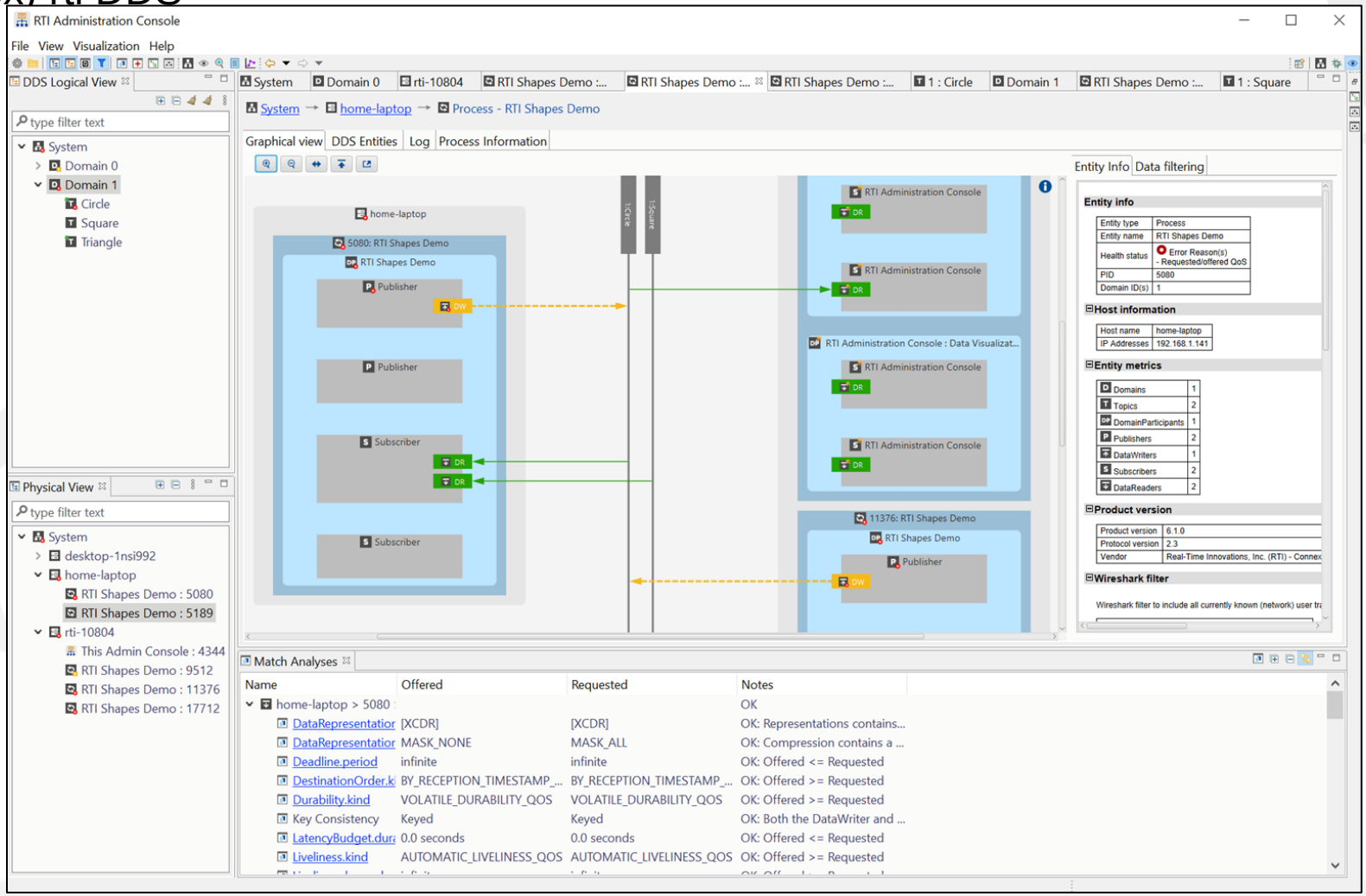
ex) rti DDS



DDS

DDS 공급사

ex) rti DDS



DDS 연동

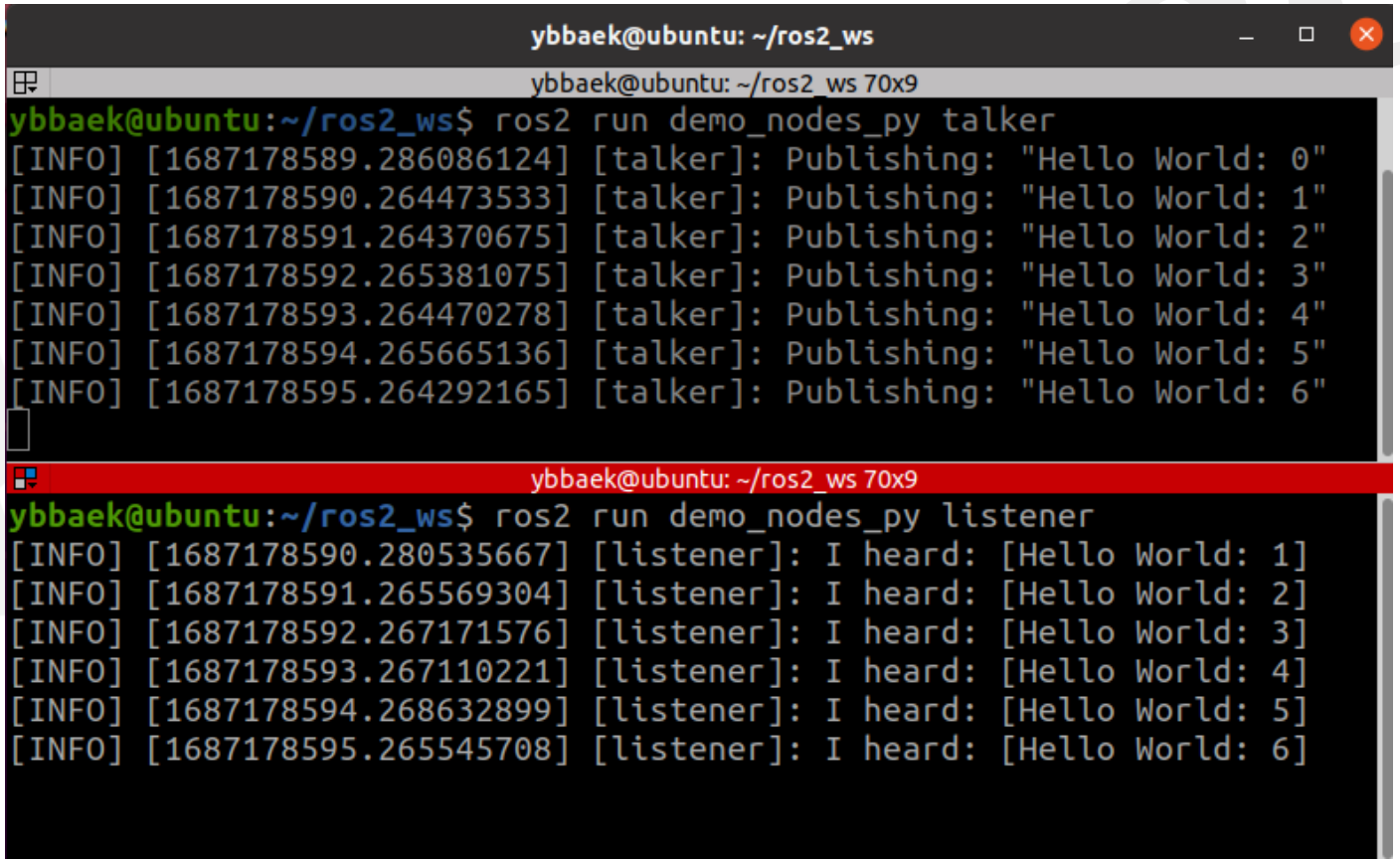
DDS 영상데모

ROS2 동작확인

기본 패키지 (demo_nodes_py)

$ ros2 run demo_nodes_py talker

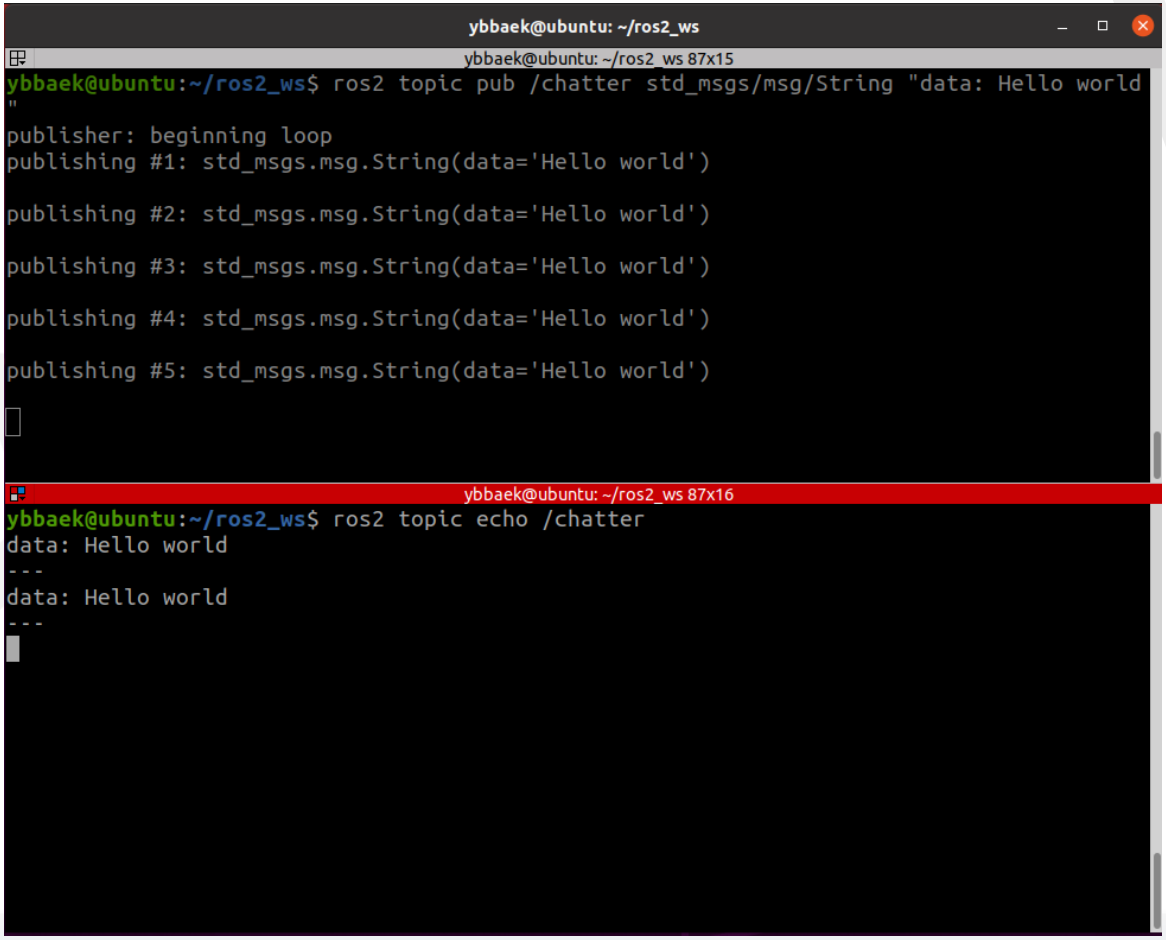
$ ros2 run demo_nodes_py listener



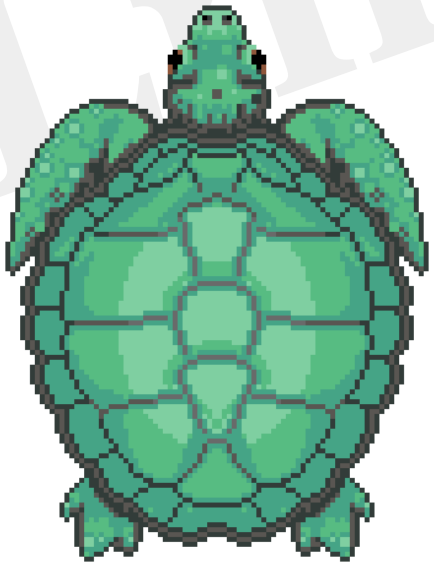
ROS2 동작확인

토픽발행

$ ros2 topic pub /chatter std_msgs/msg/String "data: Hello world"
$ ros2 topic echo /chatter



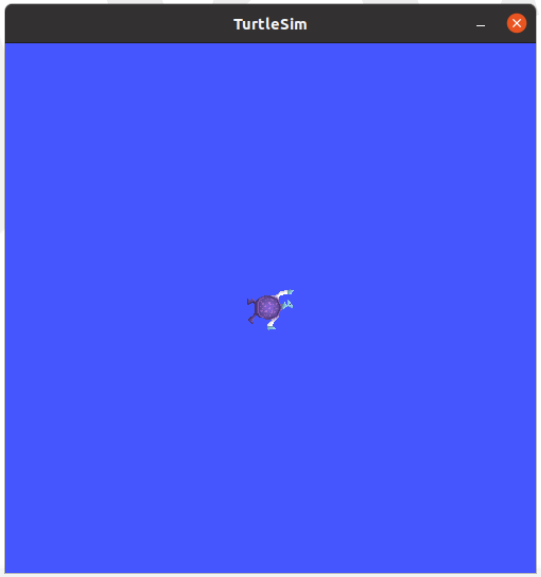
Turtlesim
실습

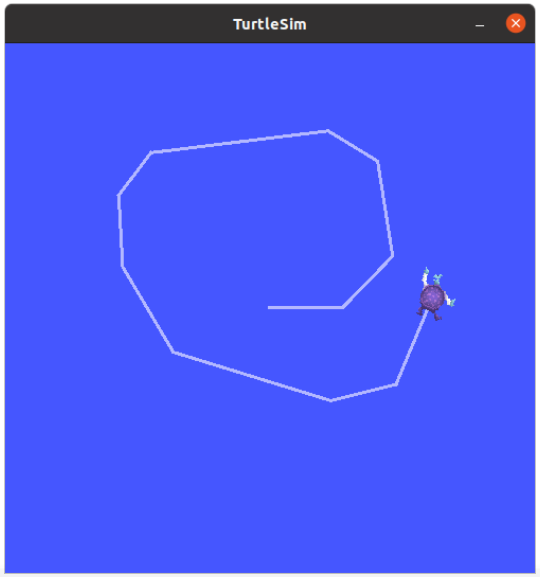


ROS2 구성

turtlesim

cmd: $ ros2 pkg executables turtlesim





turtlesim

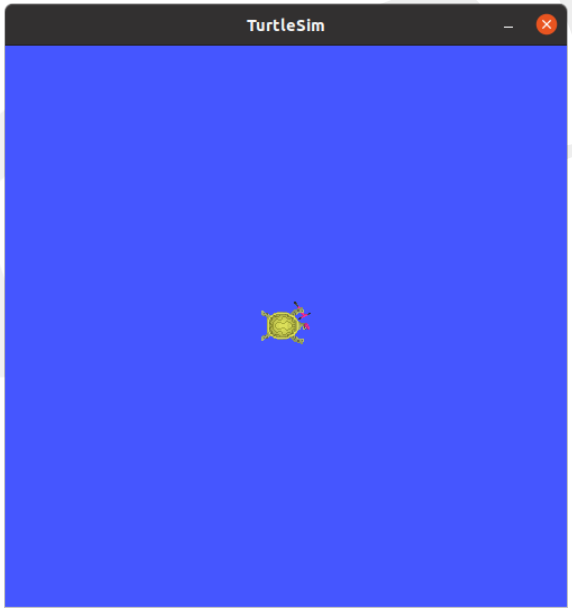
turtlesimulator 실행

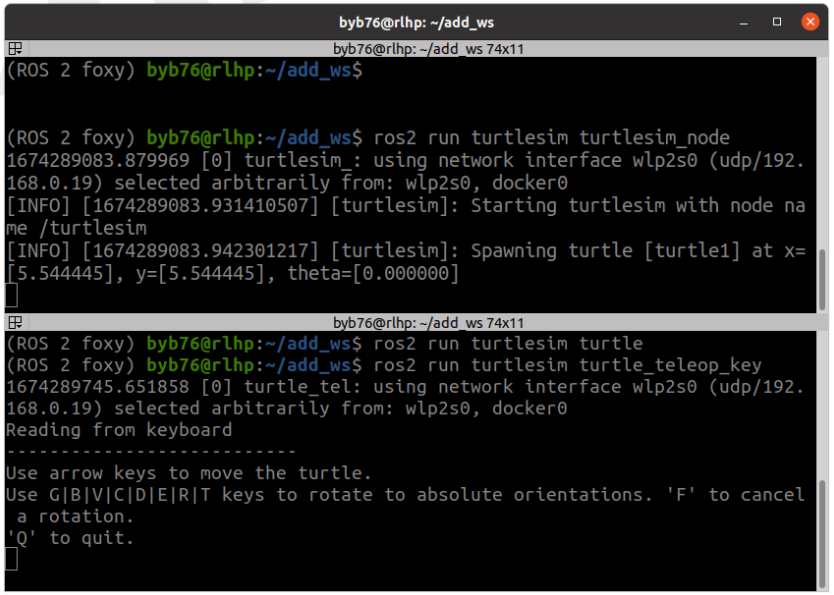
2개의 노드실행

>turtlesim_node and turtle_teleop_key.

1. cmd: ros2 run turtlesim turtlesim_node
2. cmd: ros2 run turtlesim turtle_teleop_key

별도의 2개의 터미널에서 실행



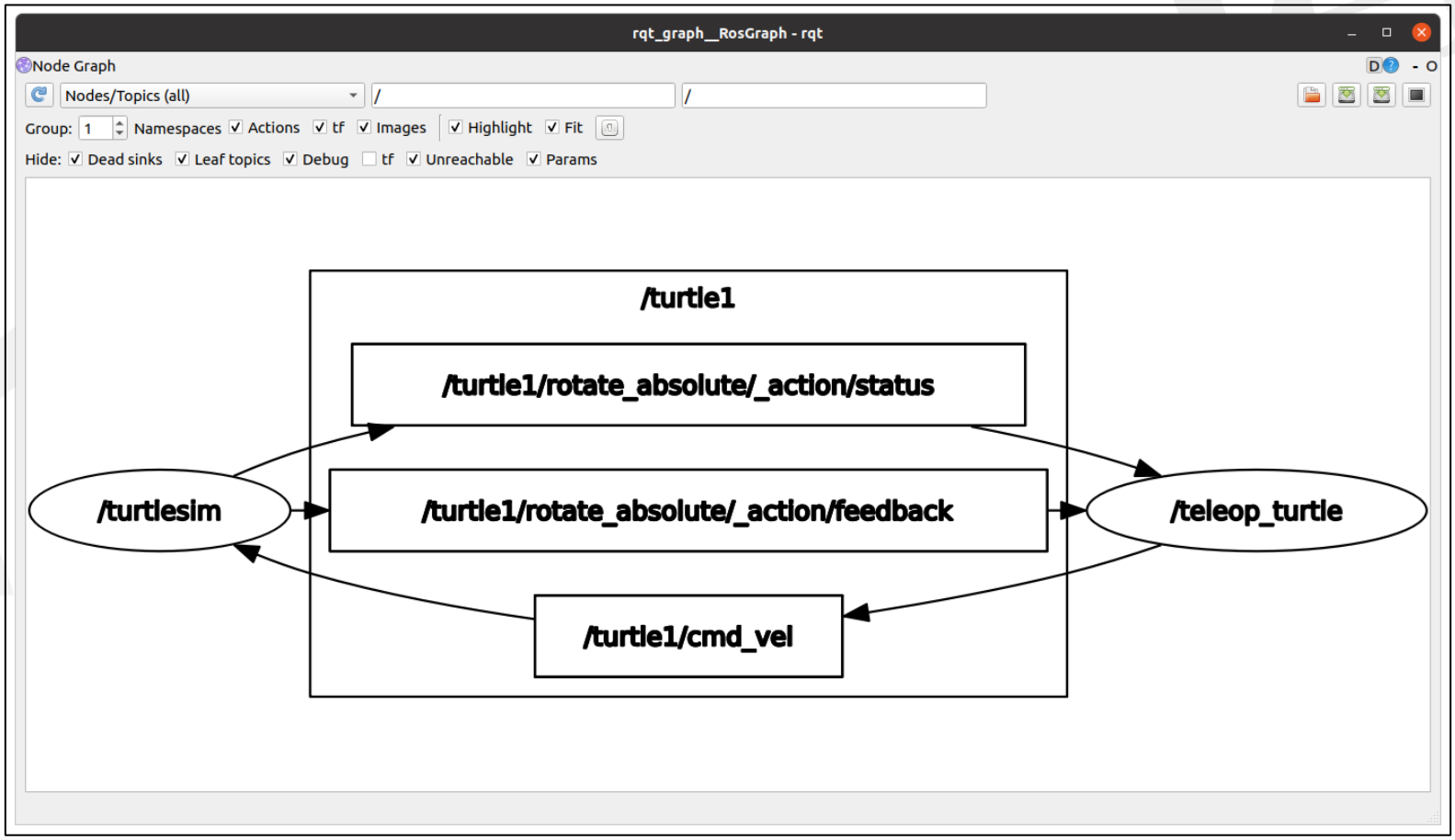


turtlesim

구조확인

rqt_graph: node, topic (방향)

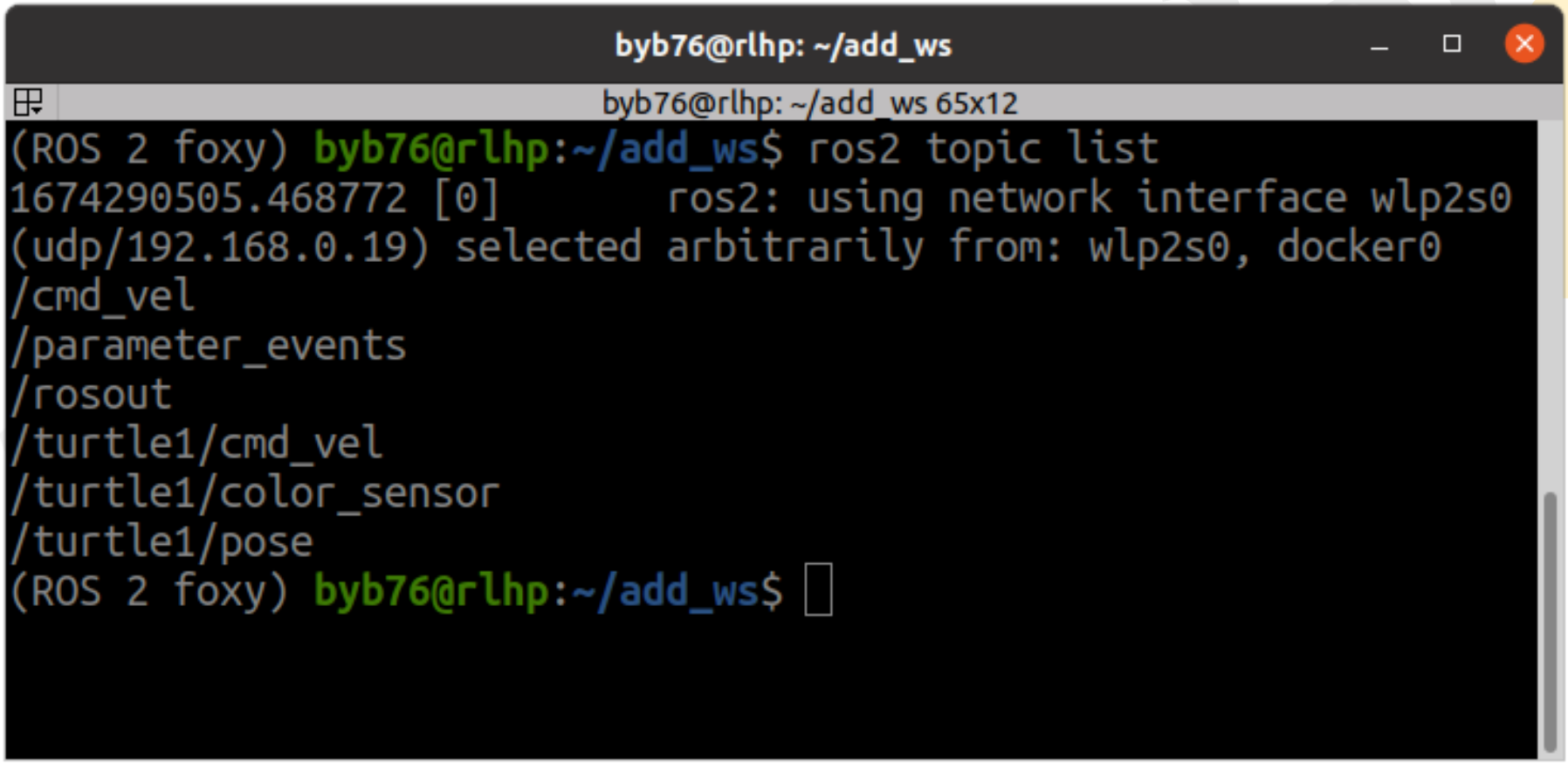
명령어: ros2 run rqt_graph rqt_graph



turtlesim

topic 확인

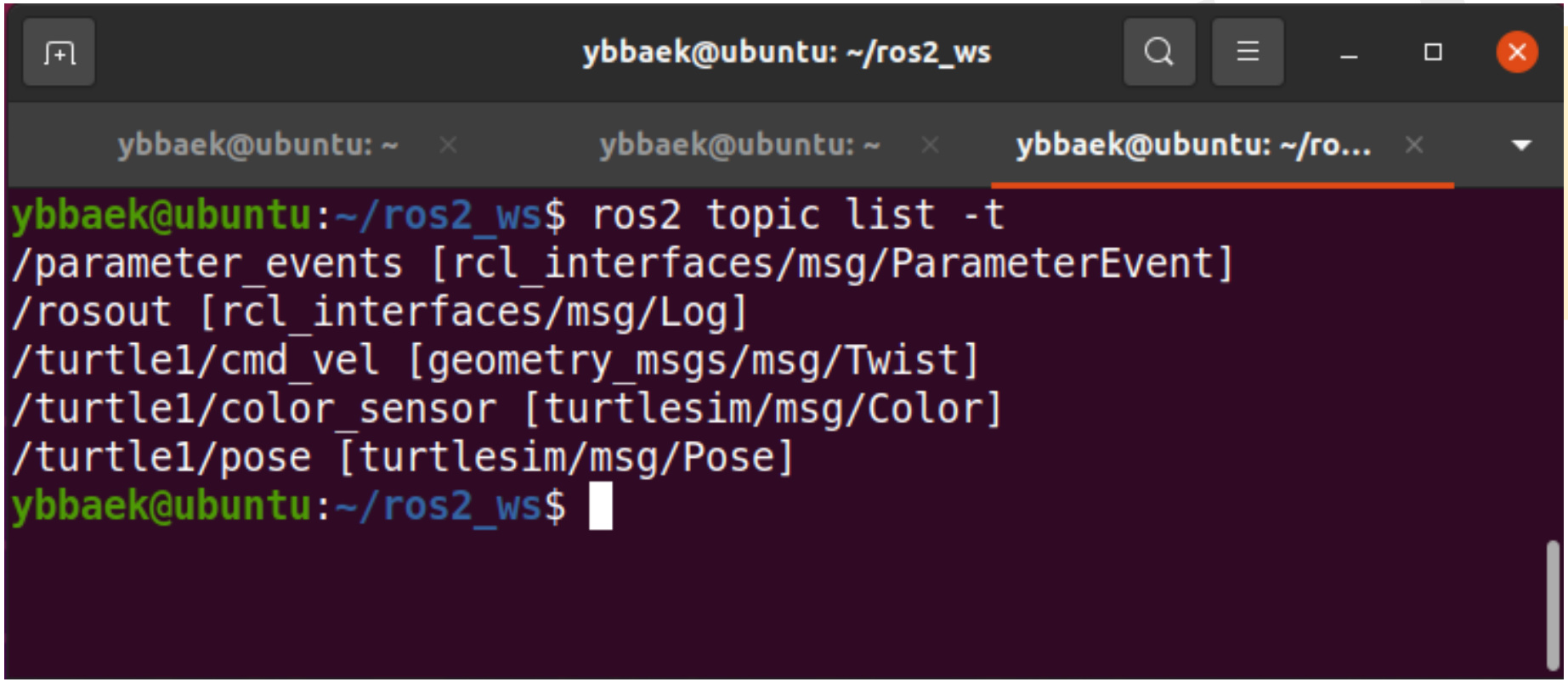
$ros2 topic list



turtlesim

topic 확인

$ros2 topic list -t



turtlesim

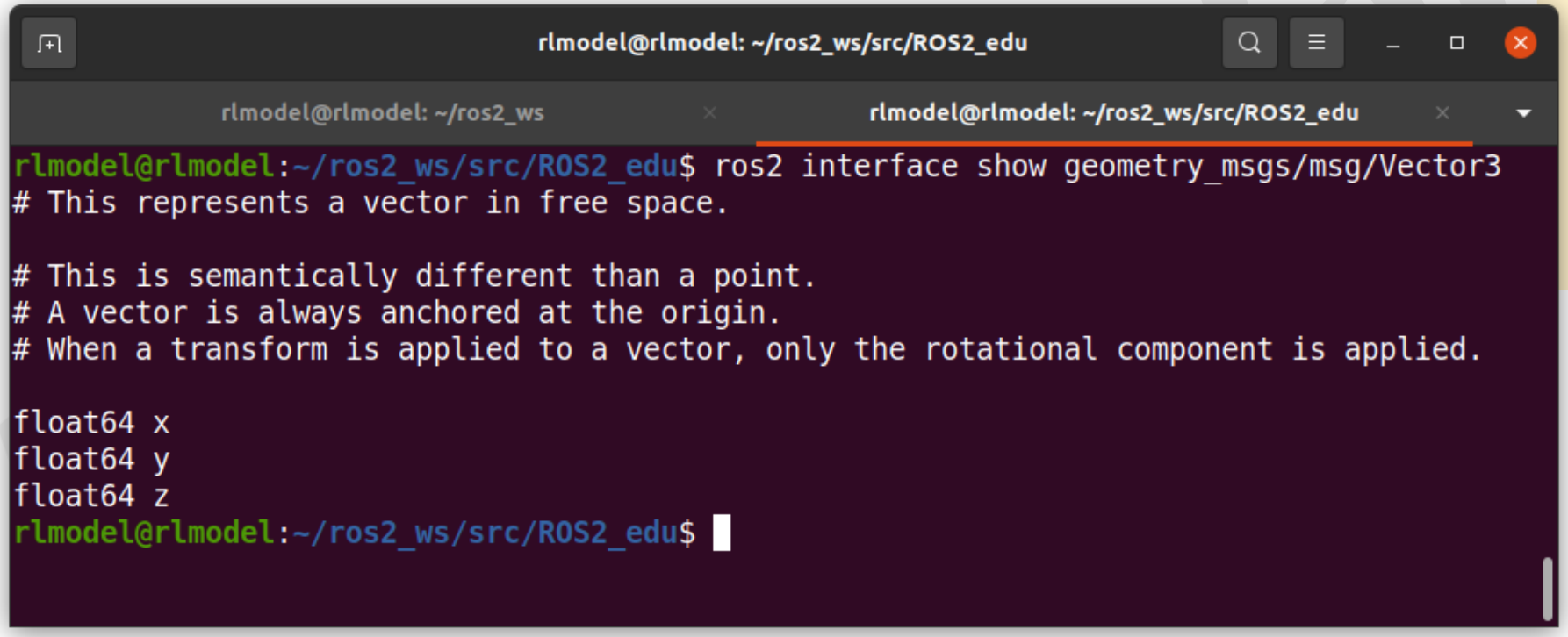
topic 확인

$ ros2 interface show geometry_msgs/msg/Twist

turtlesim

topic 확인

$ ros2 interface show geometry_msgs/msg/Twist



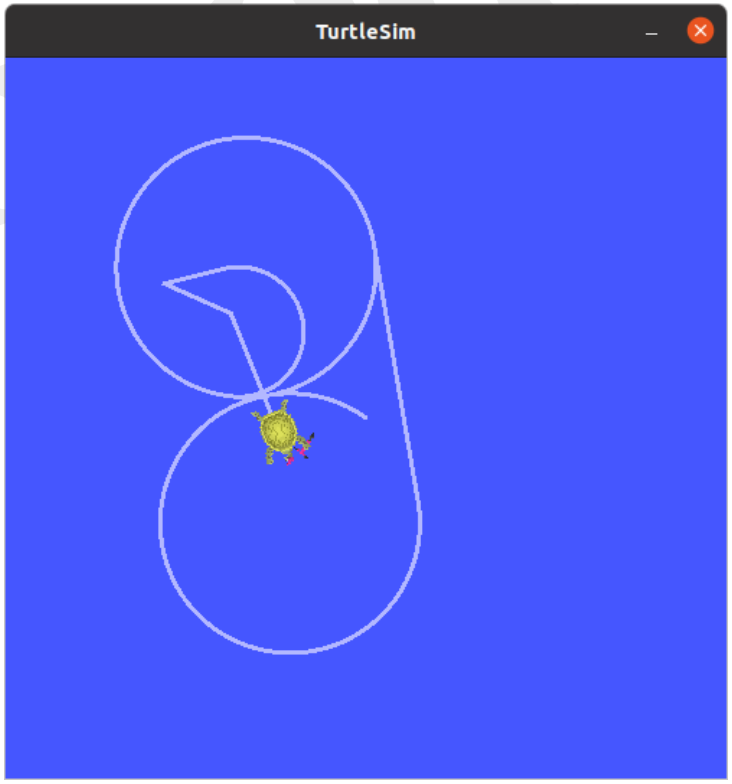
turtlesim

topic publish

토픽 발행: CLI 방법 / 터미널 창에 명령어 입력

<예>

```
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.5"
```



turtlesim

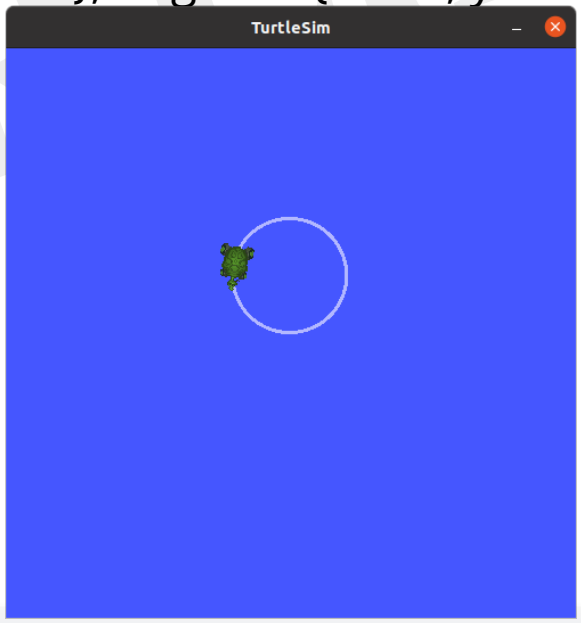
topic publish

토픽 발행: once, rate

<예>

$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"

$ ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"



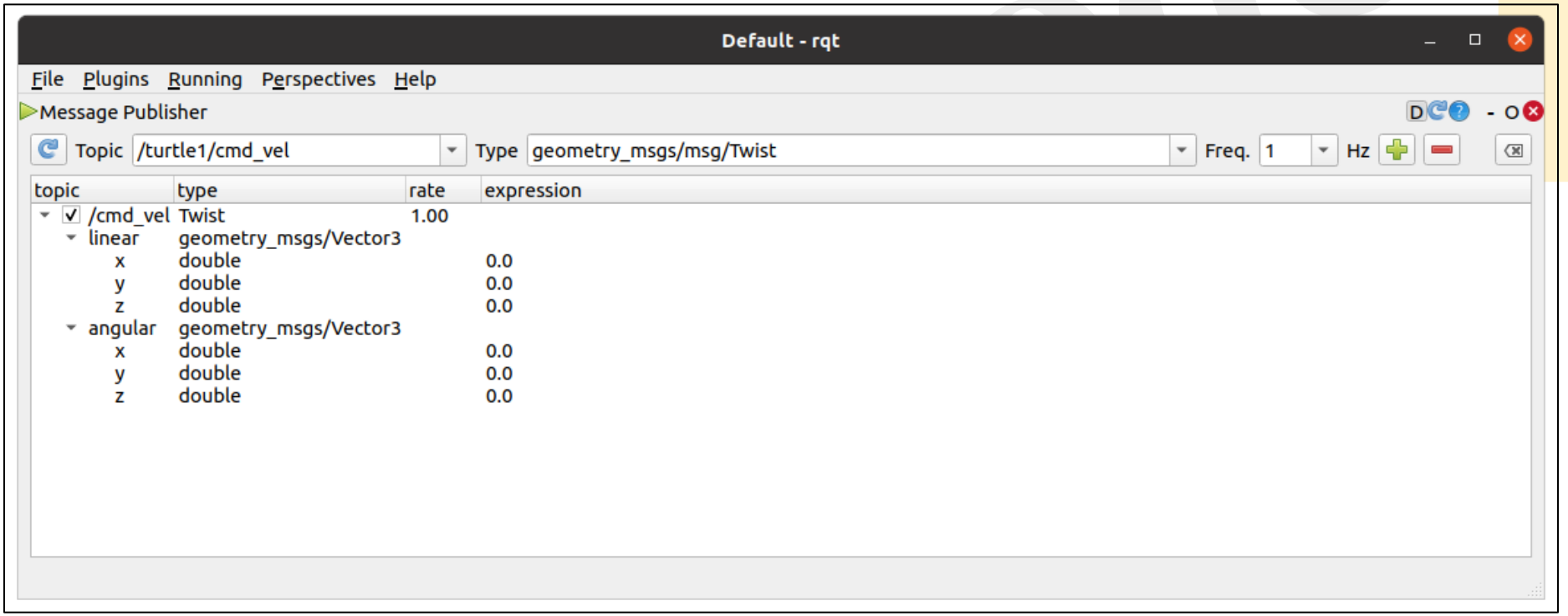
turtlesim

topic 발행

rqt method

$rqt

>Plugins -> Topics -> Message Publisher ->

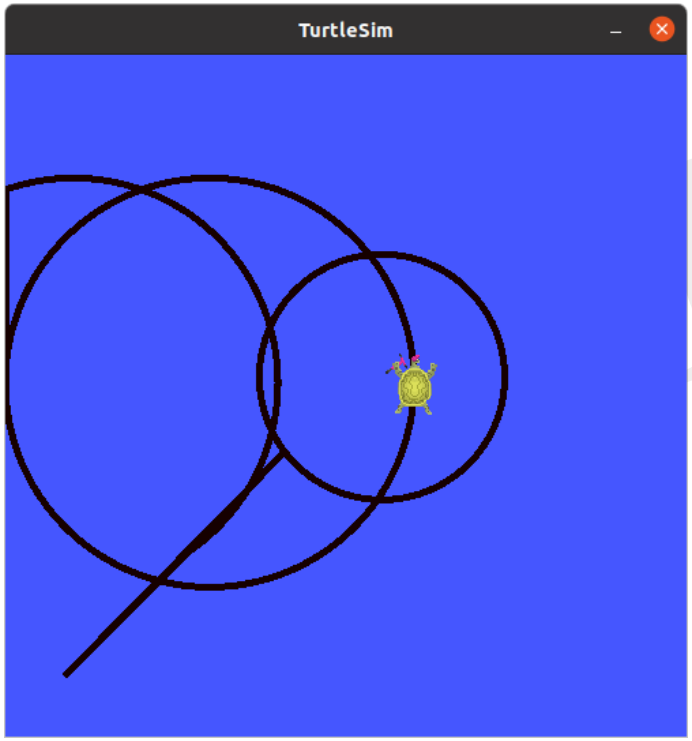


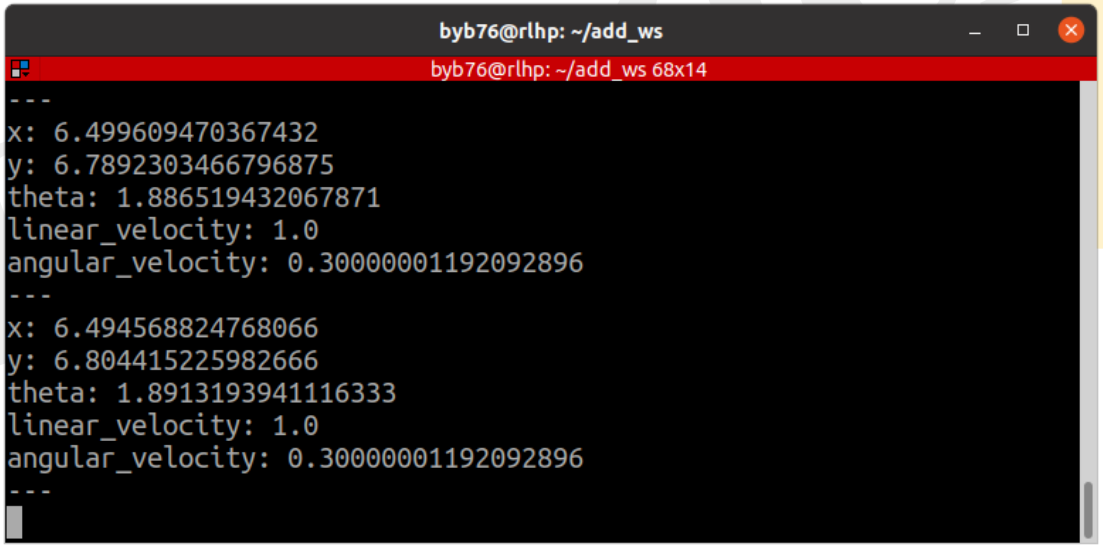
turtlesim

topic 확인

Terminal 창을 통해 CLI 로 확인 방법

$ros2 topic echo /turtle1/pose

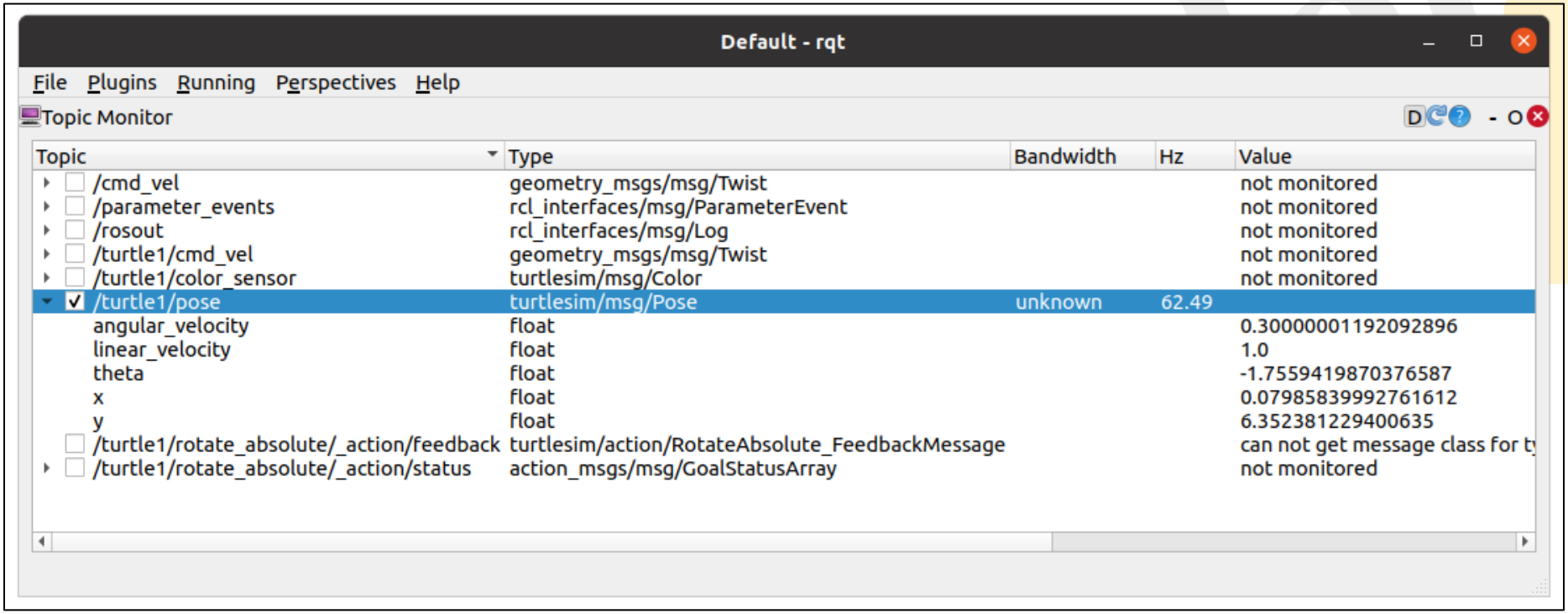




turtlesim

topic 확인

rqt로 확인

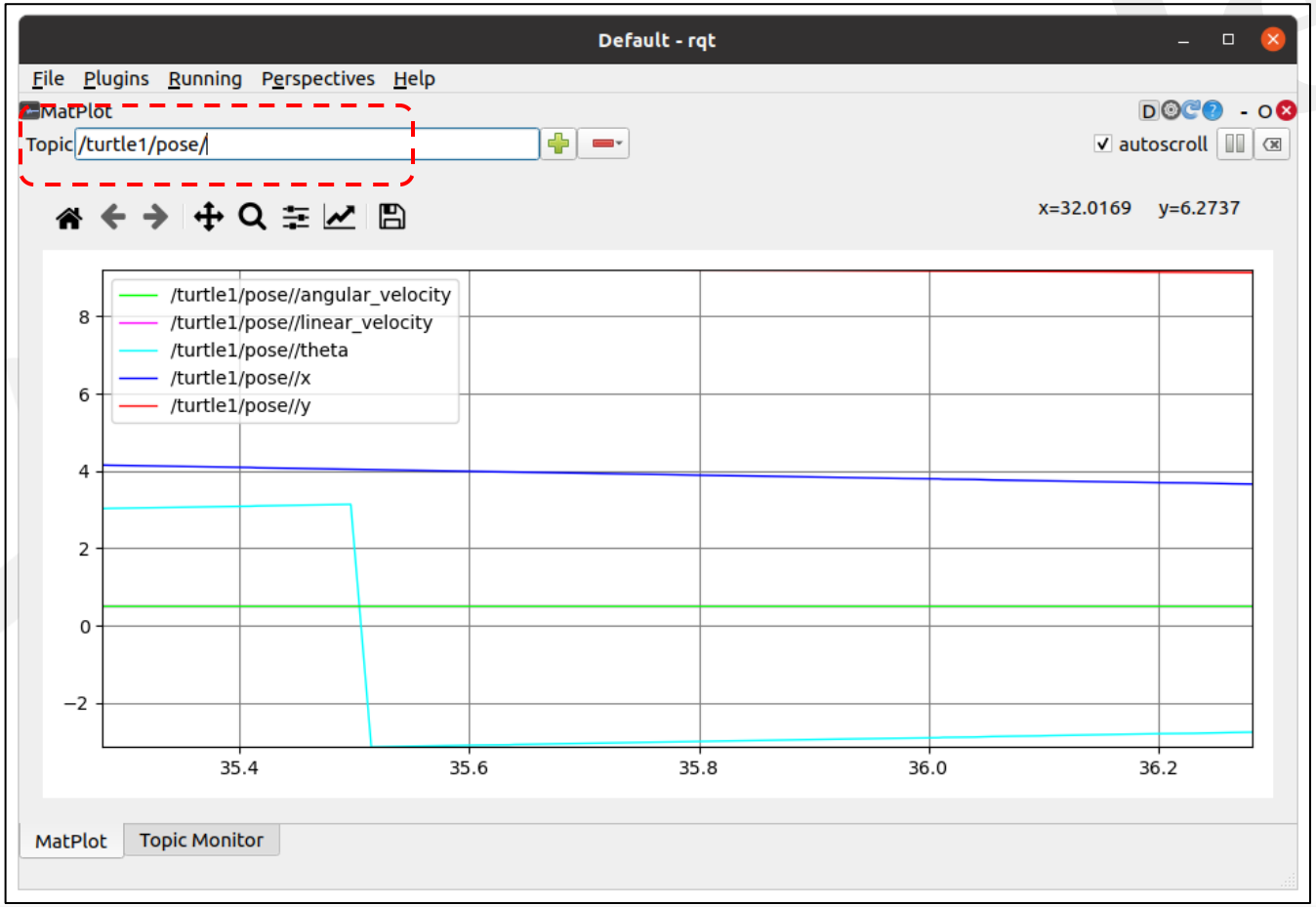


turtlesim

topic 확인

rqt: Plugins -> Visualization -> Plot

>/turtle1/pose/

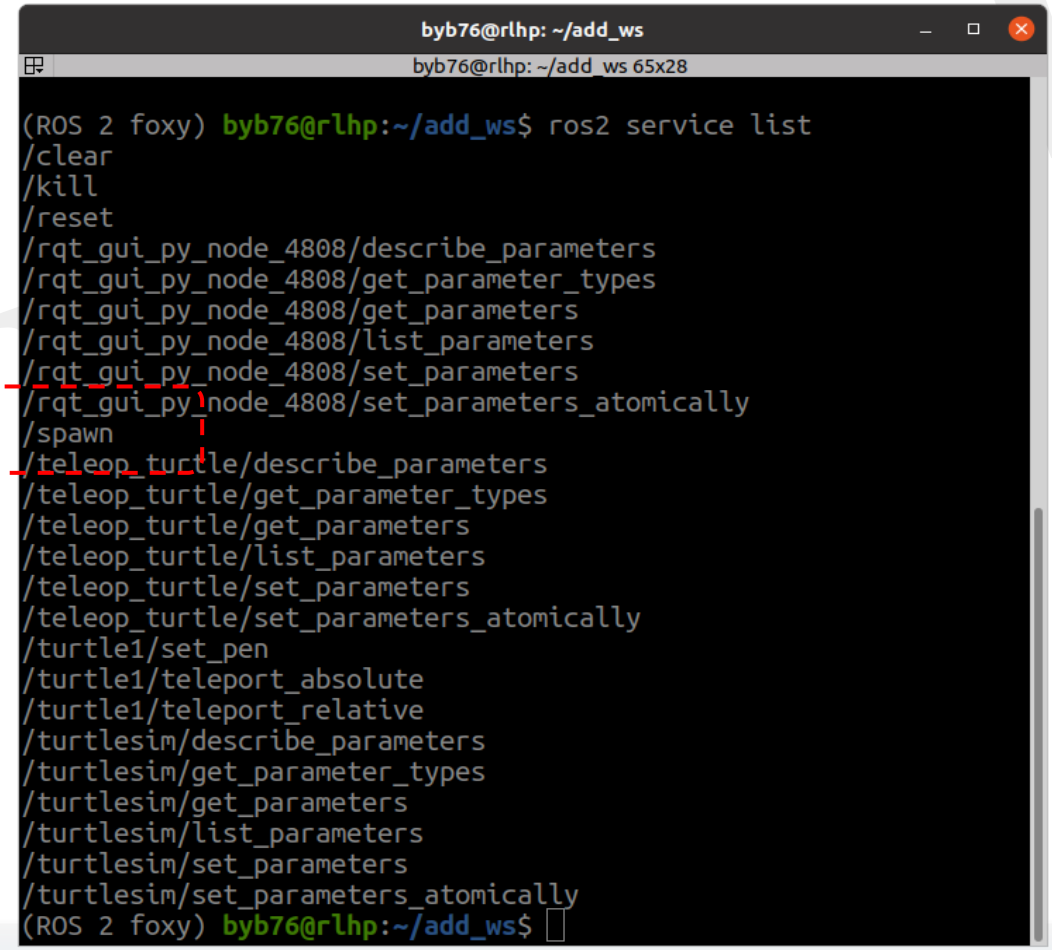


turtlesim

service 확인

>서비스 리스트 확인

>$ros2 service list (-t option show msg type)

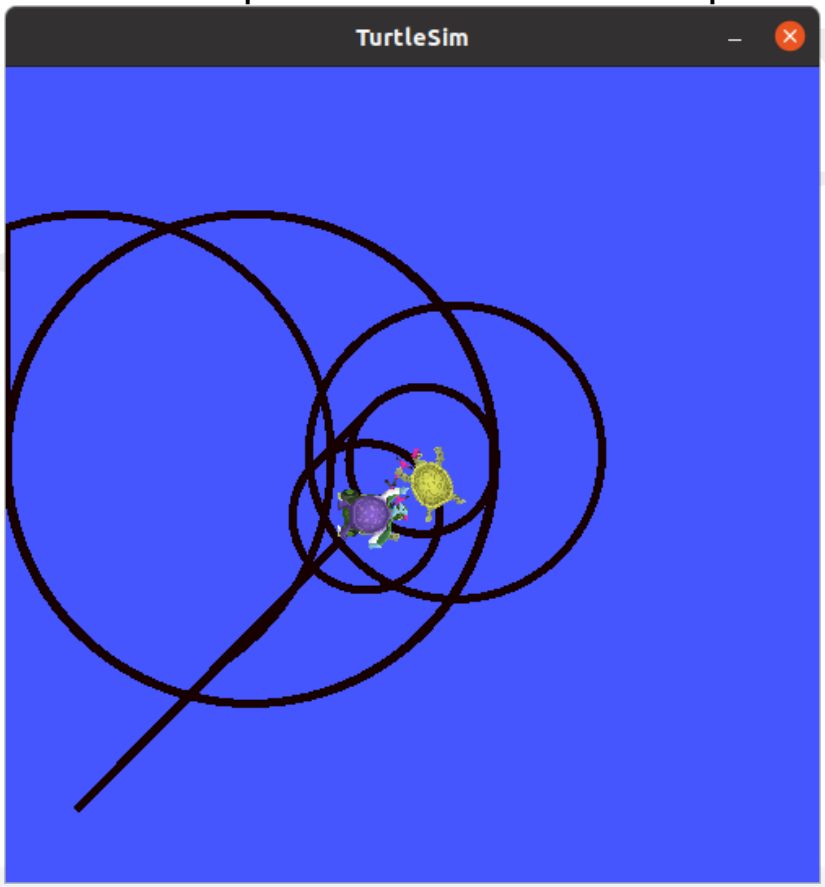


turtlesim

service 확인

spawn service 실행

```
$ ros2 service call /spawn turtlesim/srv/Spawn
$ ros2 service call -r 0.5 /spawn turtlesim/srv/Spawn "{x: 5,y: 5,theta: 0}"
```



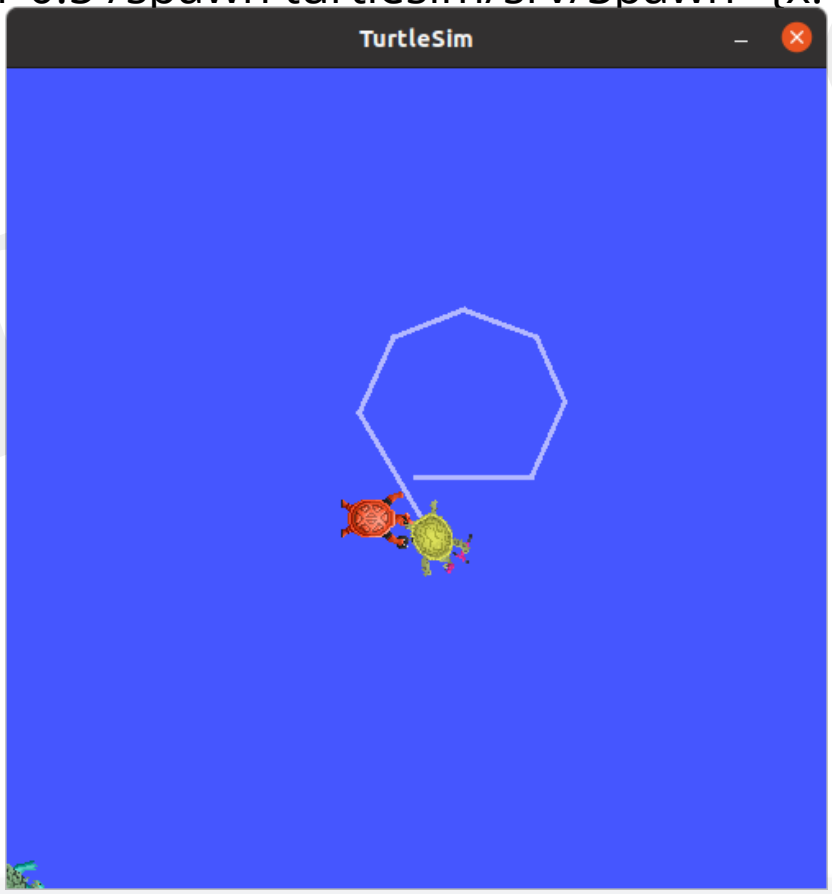
turtlesim

service 확인

spawn service 실행

$ros2 service call /spawn turtlesim/srv/Spawn

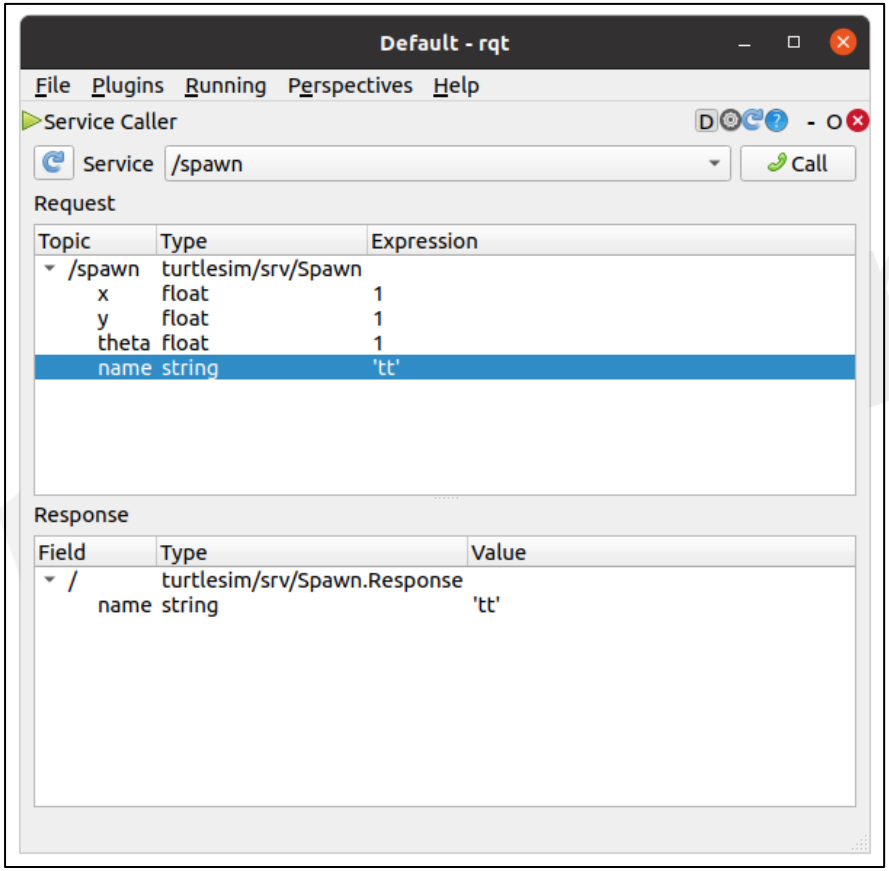
$ros2 service call -r 0.5 /spawn turtlesim/srv/Spawn "{x: 5,y: 5,theta: 0}"

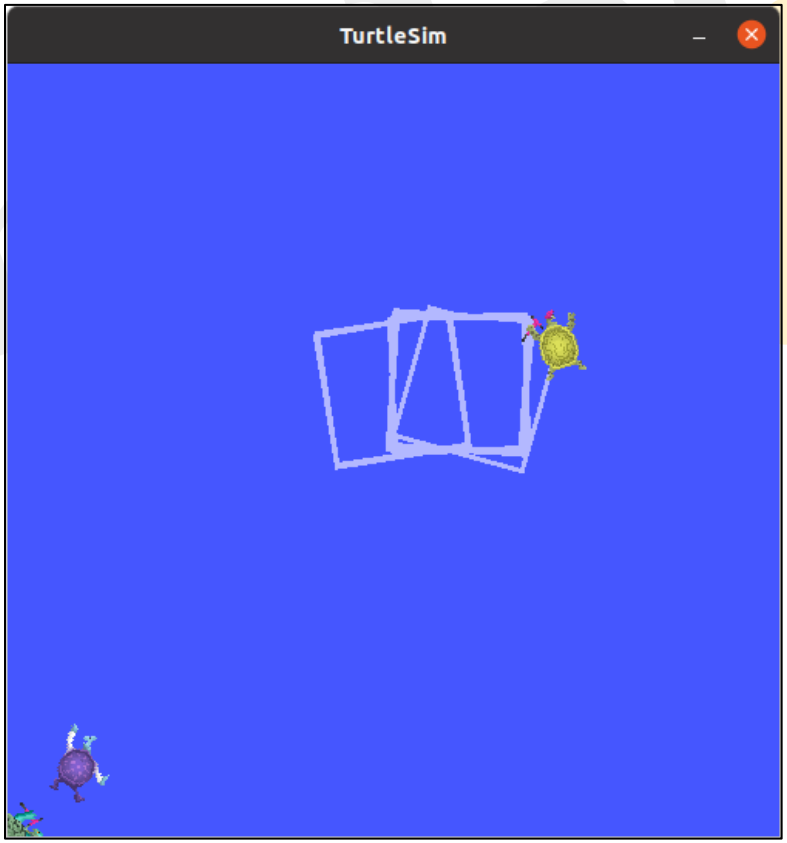


turtlesim

service 확인

spawn service 실행 with rqt



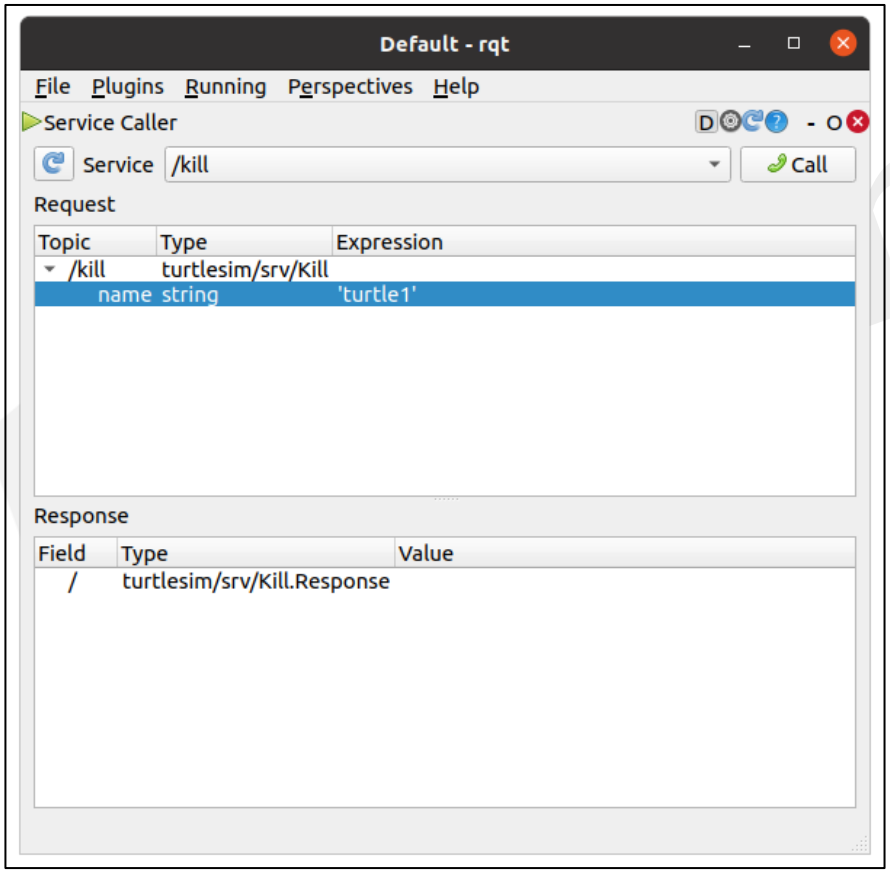


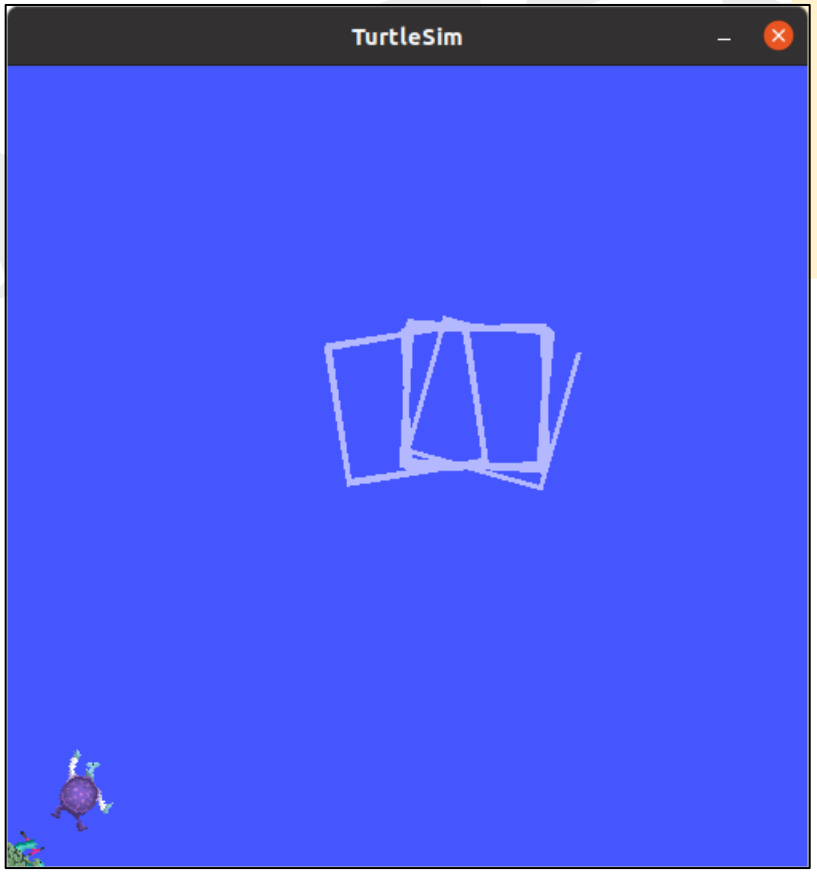
turtlesim

service 확인

kill service 실행 with rqt

ros2 service call /kill turtlesim/srv/Kill "name: 'turtle1'"

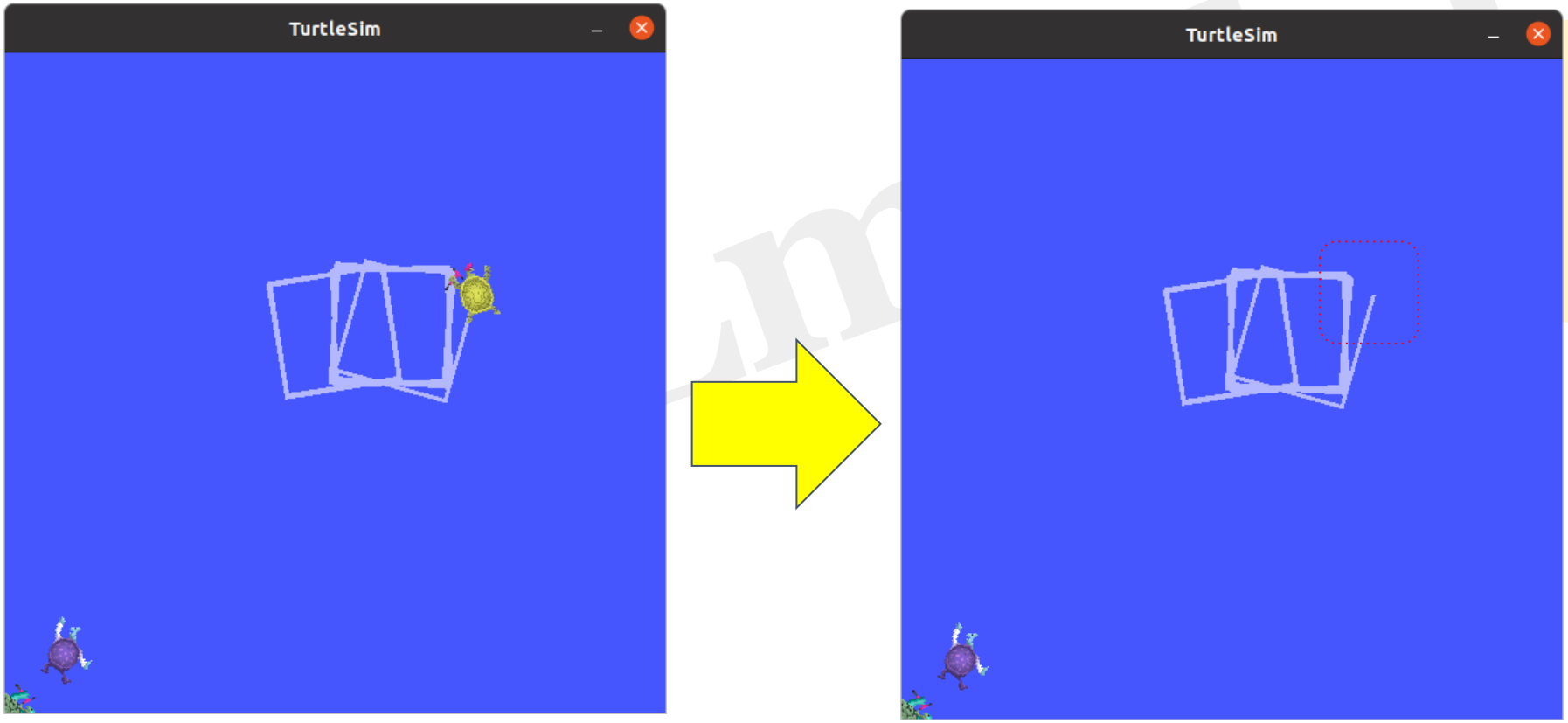




turtlesim

service 확인

kill service 실행 with rqt

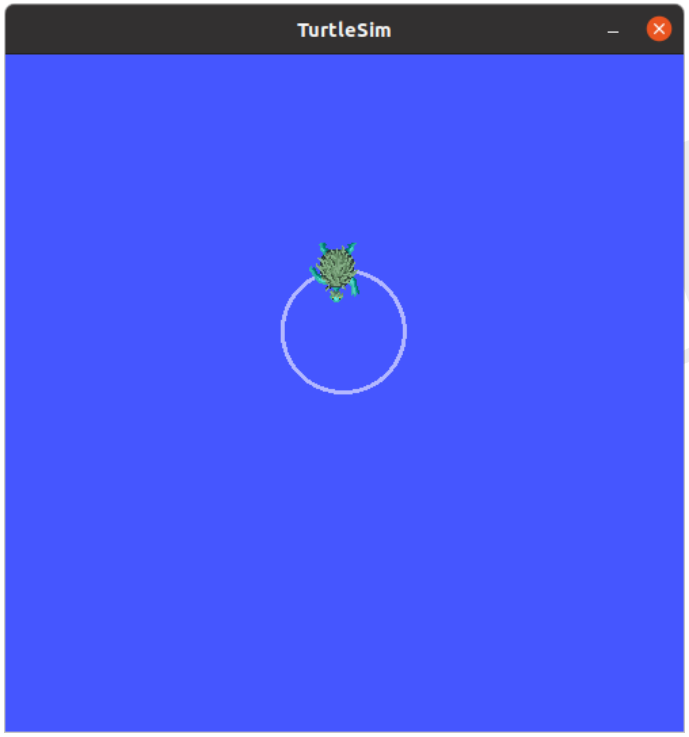


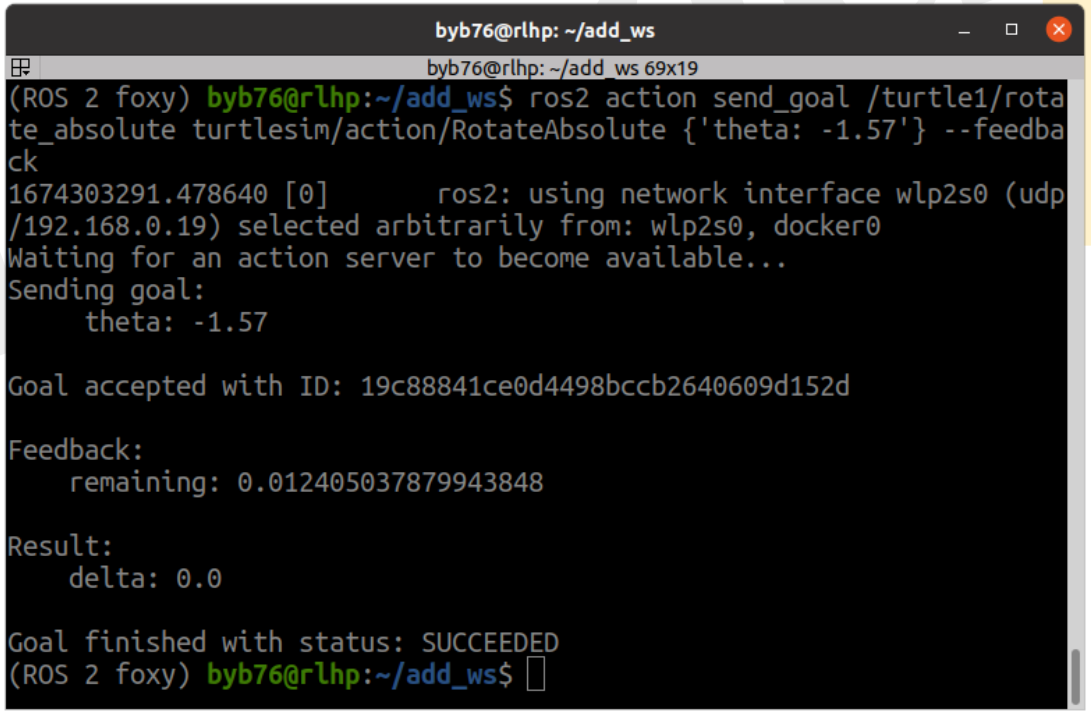
turtlesim

action 확인

send_goal 실행

$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute {'theta: -1.57'} --feedback

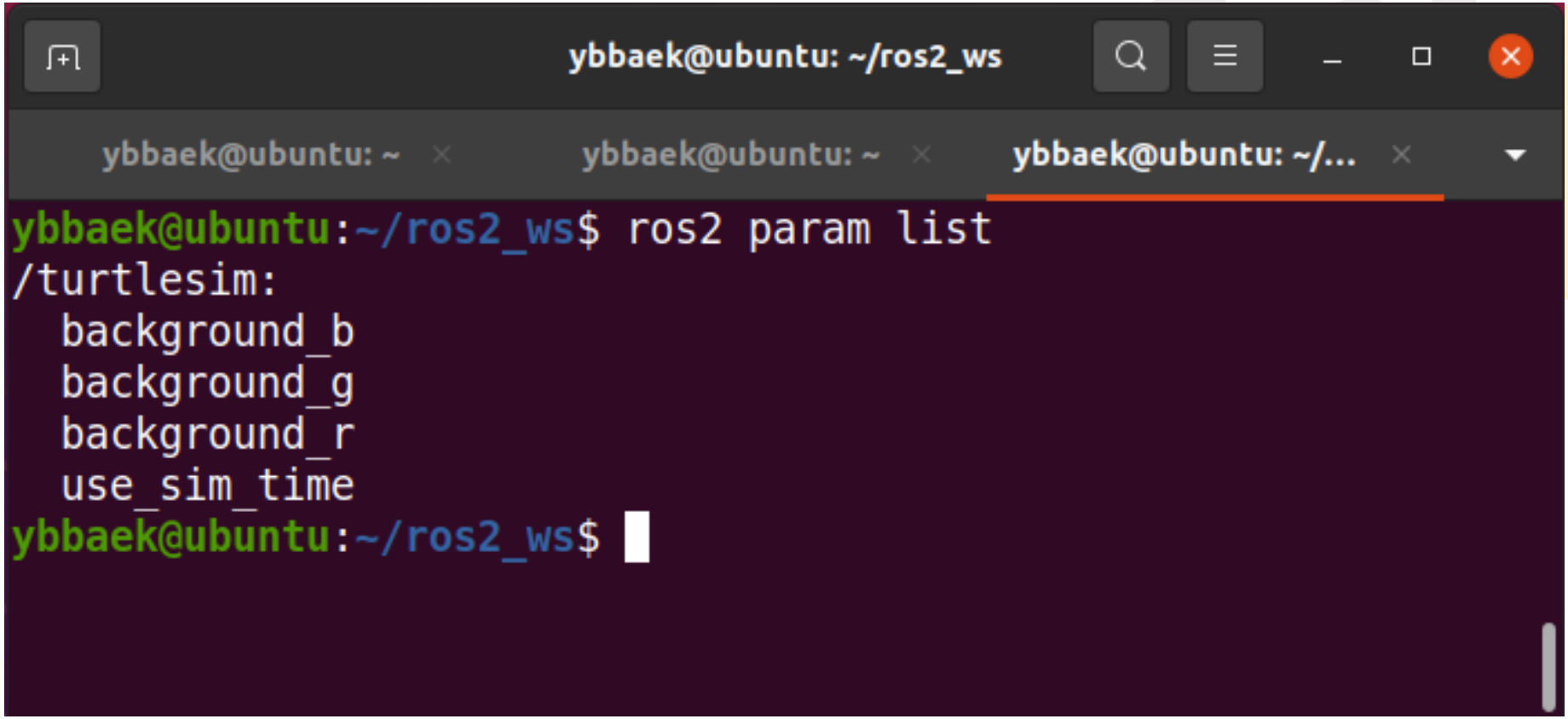




turtlesim

parameter 확인

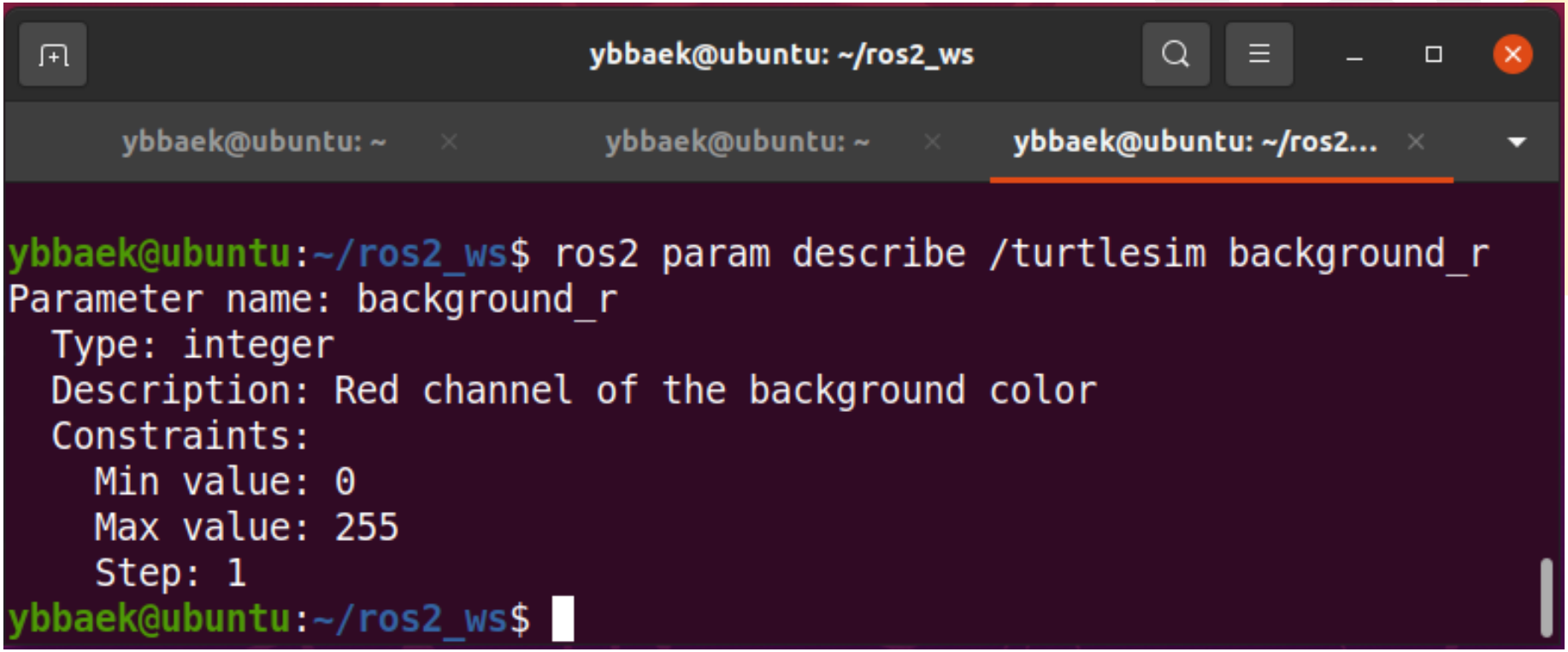
$ros2 param list



turtlesim

parameter 확인

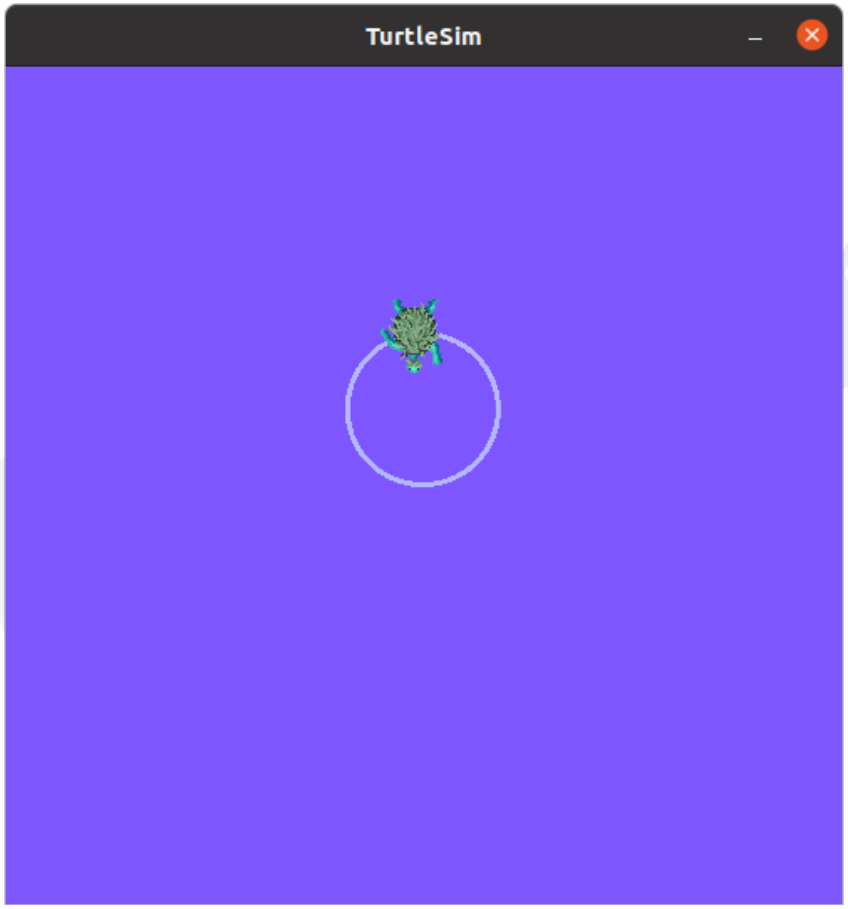
$ ros2 param describe /turtlesim background_r

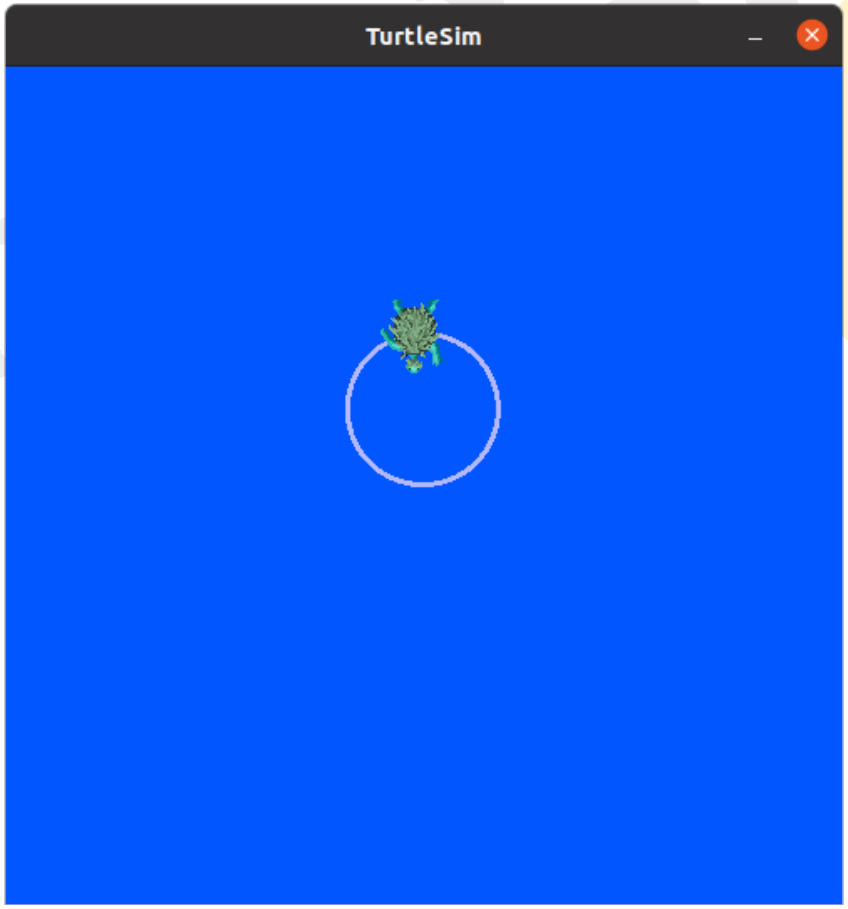


turtlesim

parameter set - 쓰기

$ros2 param set /turtlesim background_r 1

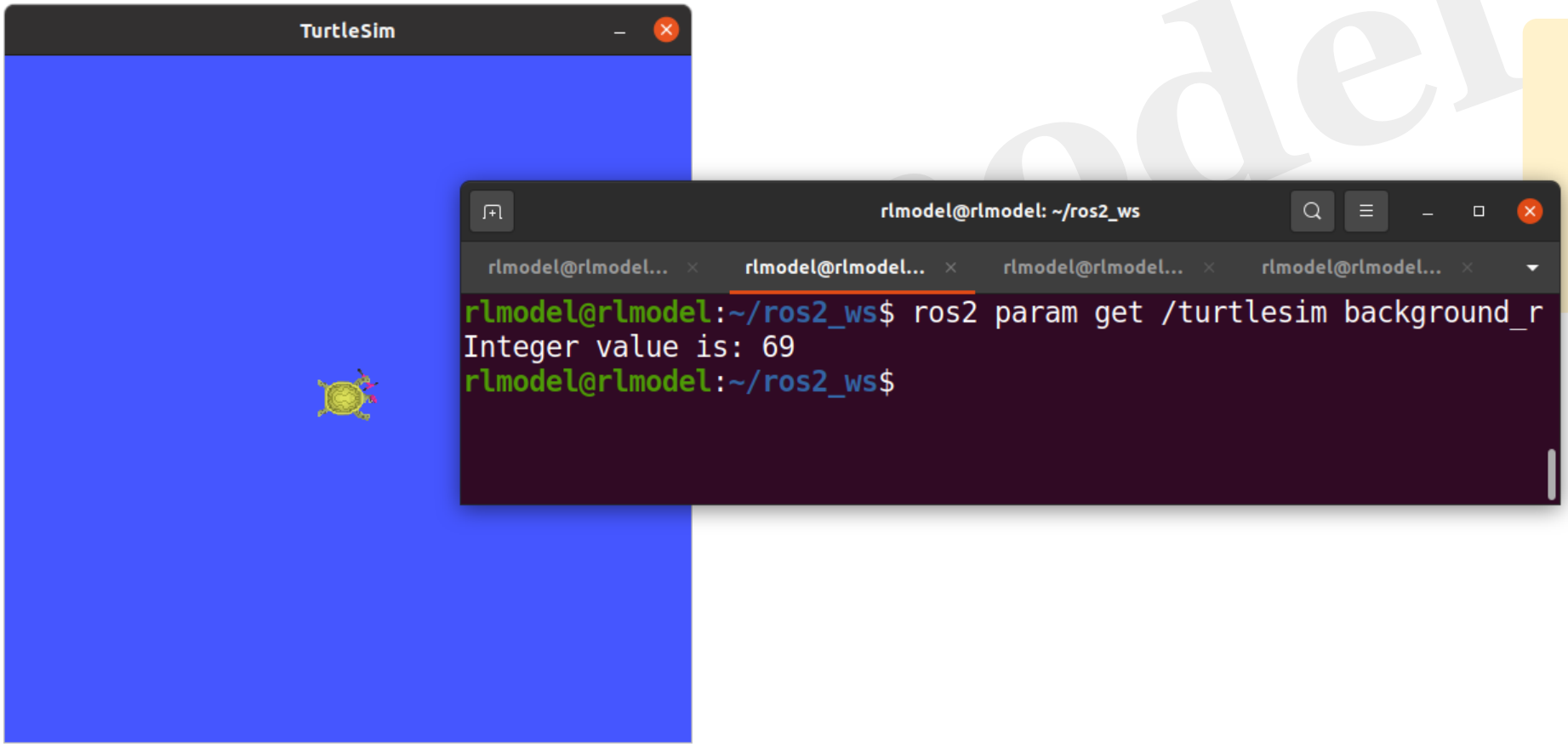




turtlesim

parameter get - 읽기

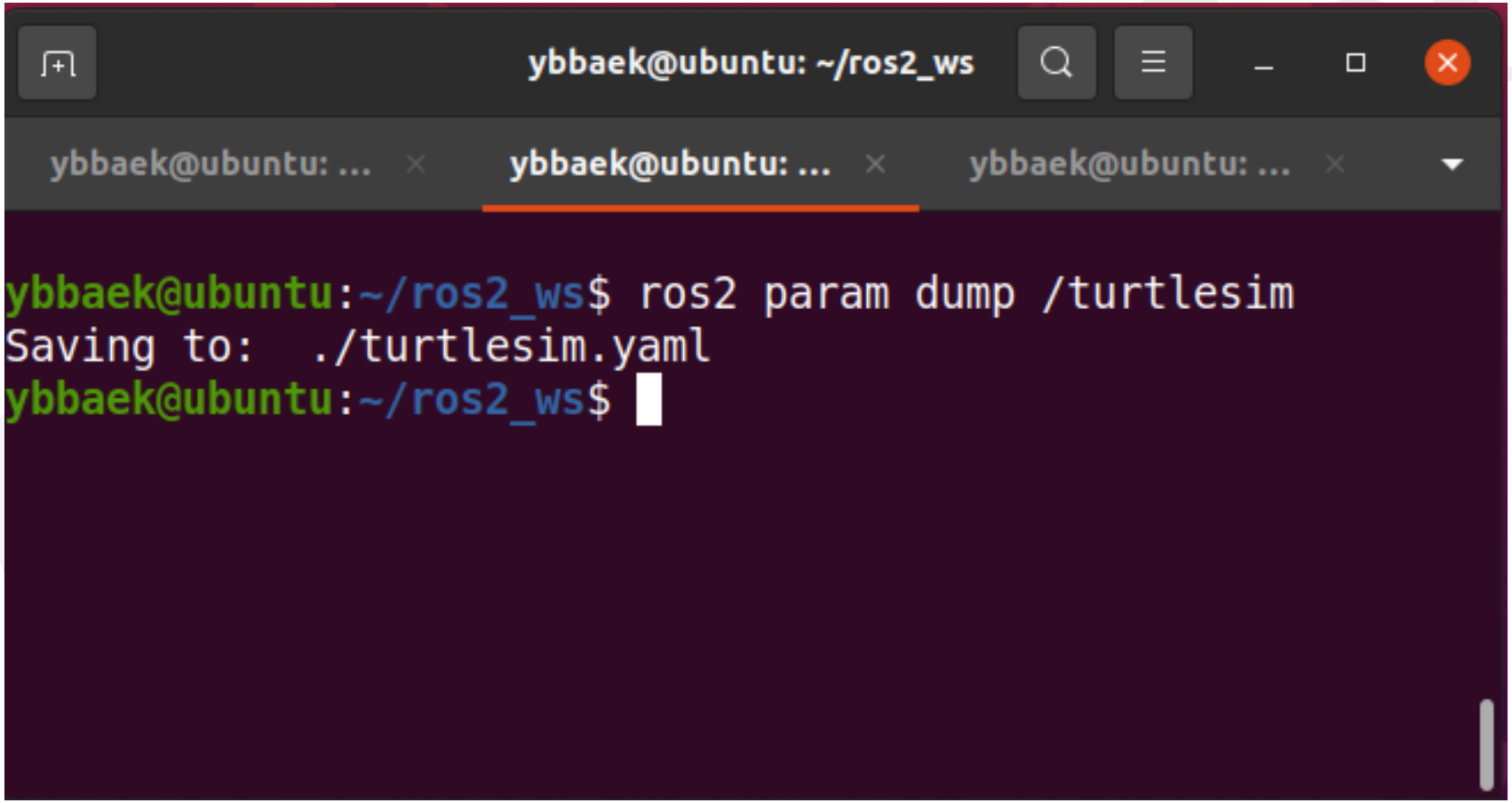
$ros2 param get /turtlesim background_r



turtlesim

parameter 저장

$ros2 param dump /turtlesim

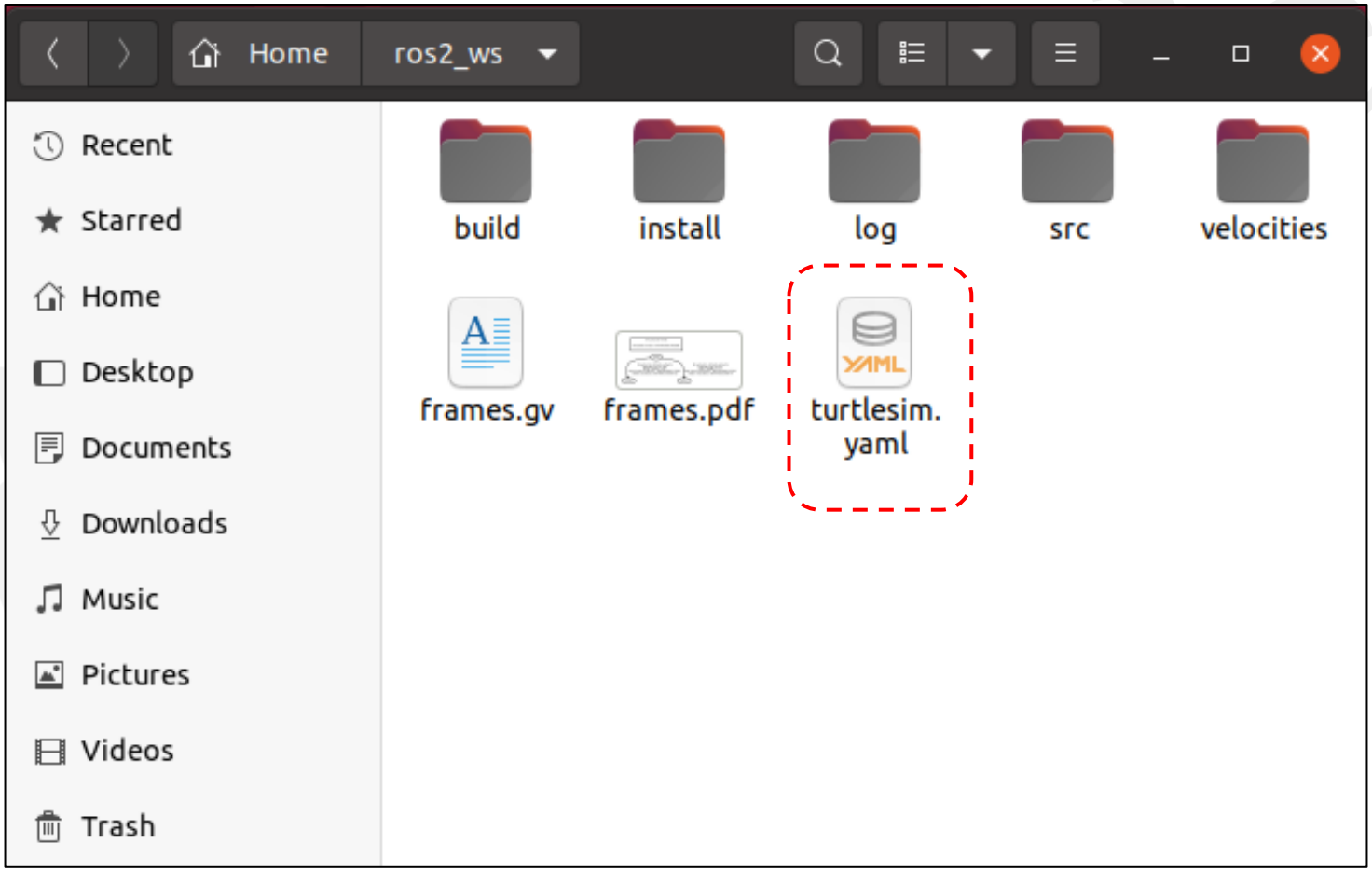


turtlesim

parameter 저장

$ros2 param dump /turtlesim

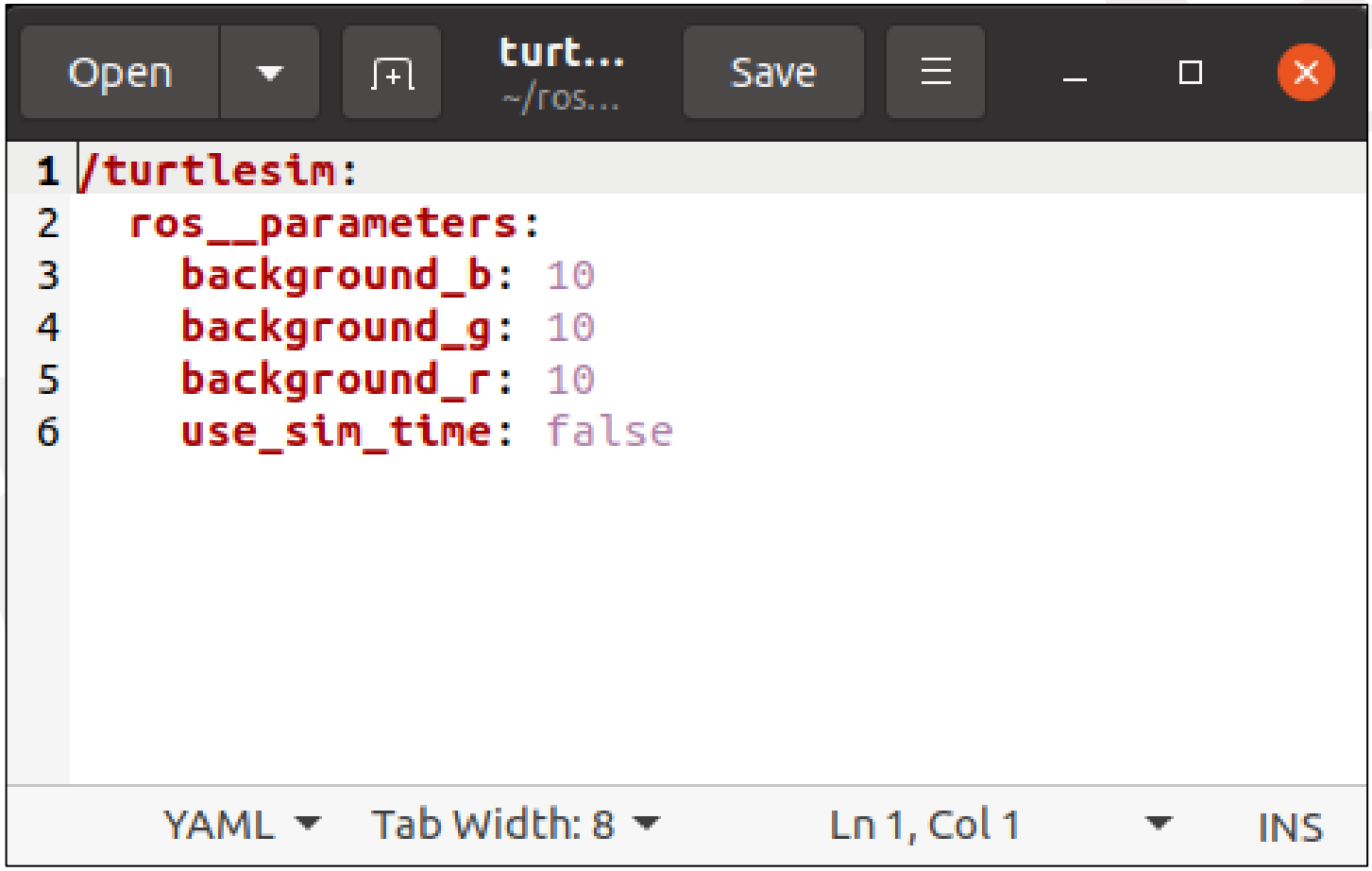
Appendix 참고



turtlesim

parameter 저장

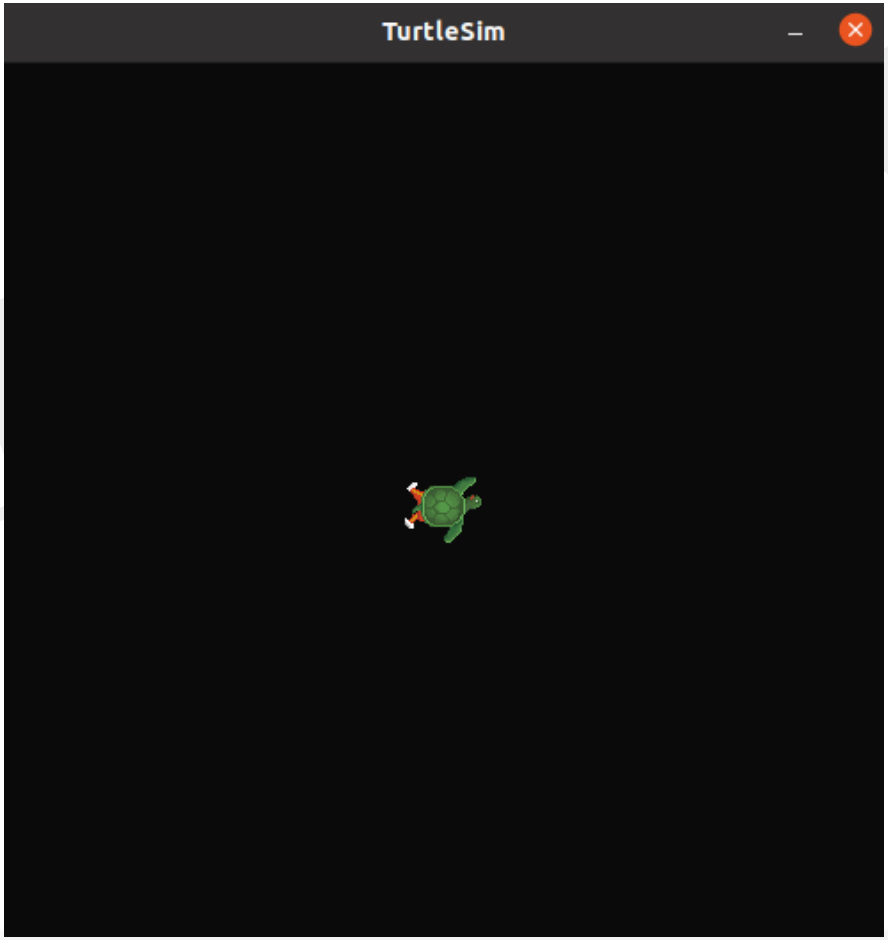
$ros2 param dump /turtlesim



turtlesim

parameter 로딩

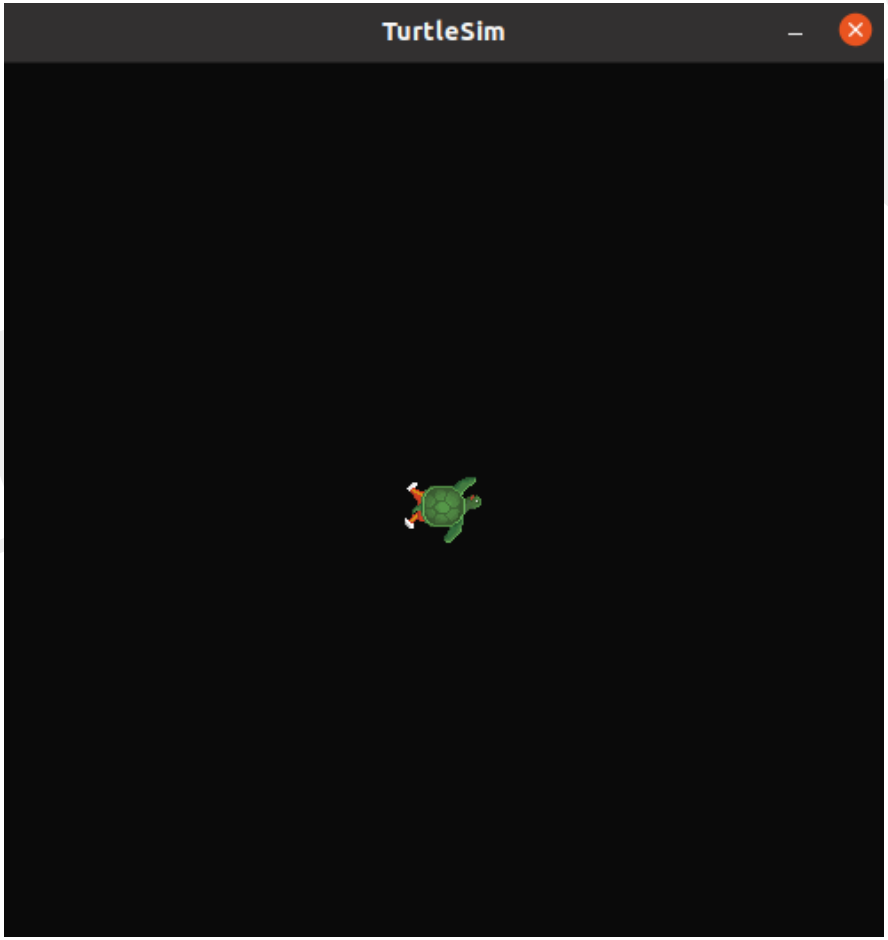
$ ros2 run turtlesim turtlesim_node --ros-args --params-file ./turtlesim.yaml



turtlesim

parameter 로딩

$ ros2 param load turtlesim ./turtlesim.yaml

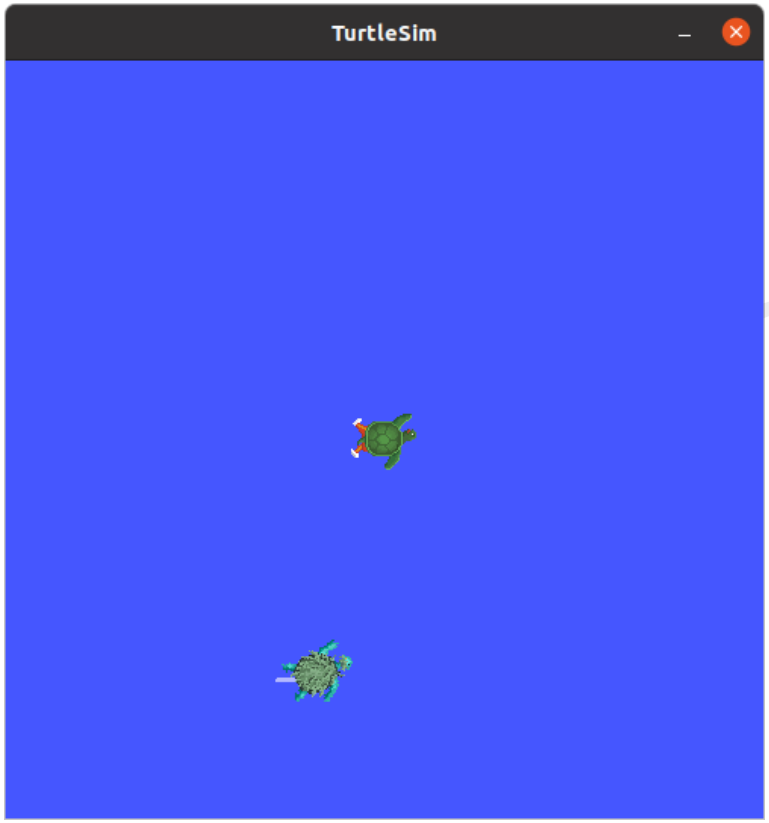


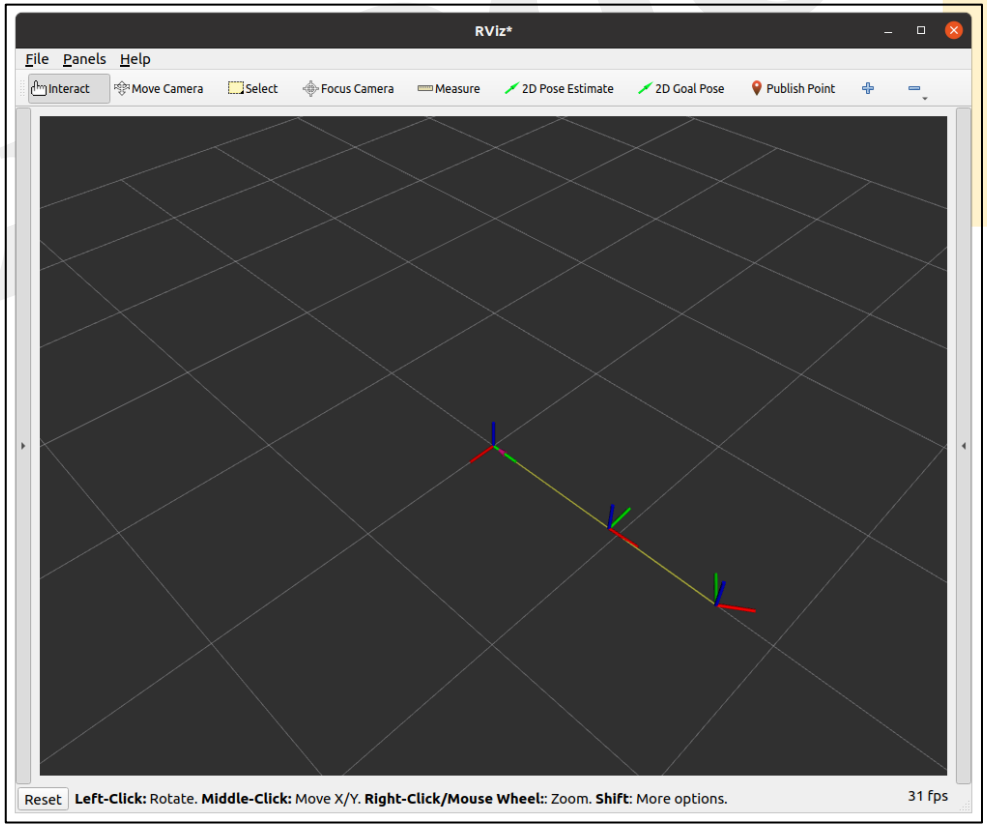
turtlesim

tf2 - turtlesim (rviz2)

설치: sudo apt-get install ros-foxy-turtle-tf2-py ros-foxy-tf2-tools ros-foxy-tf-transformations

실행: ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py



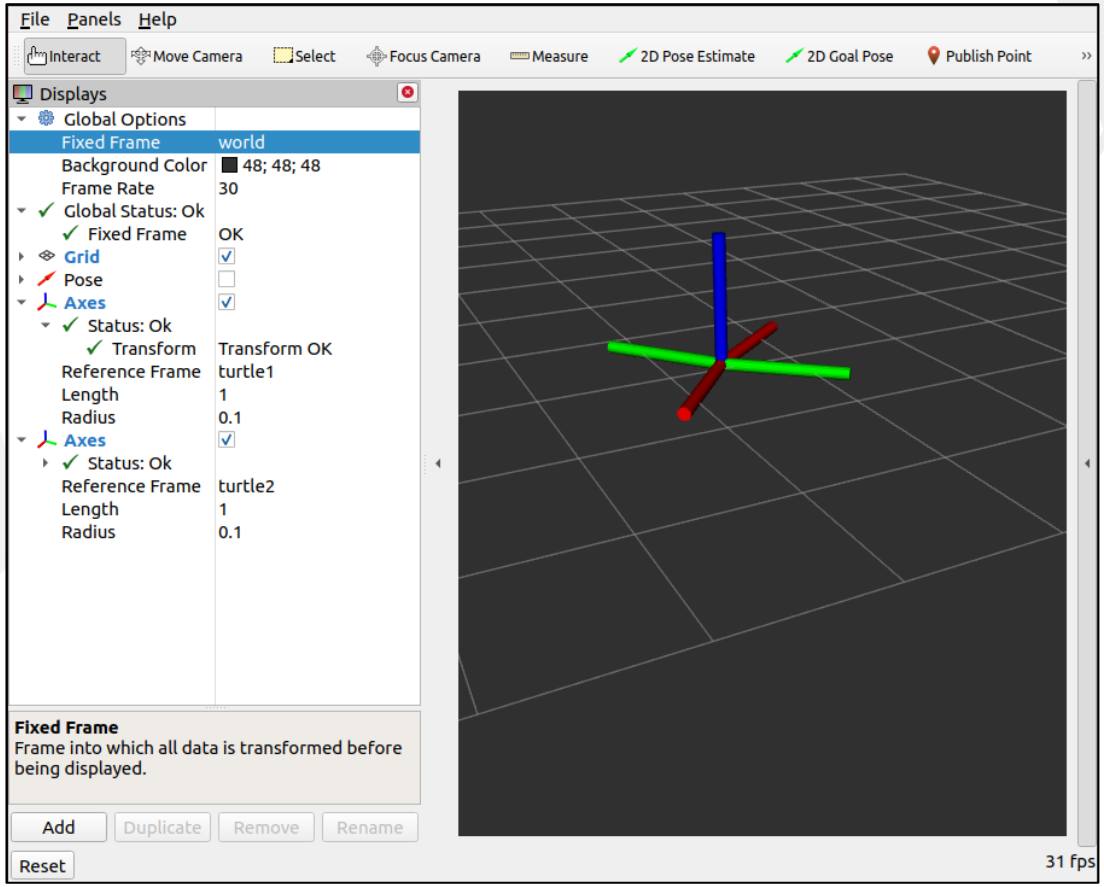


turtlesim

tf2 - turtlesim

실행1: turtlesim turtle_teleop_key

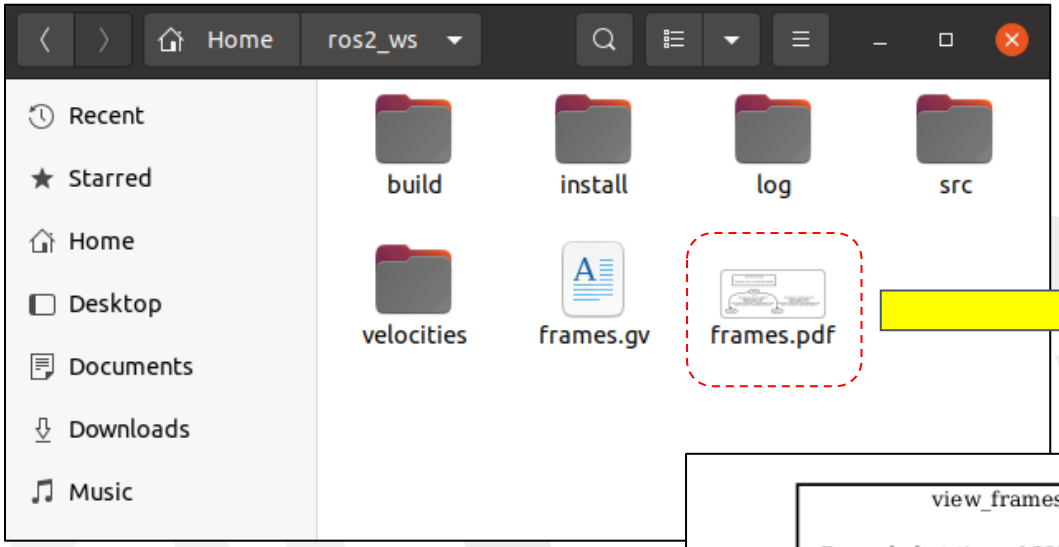
실행2: rviz Axes 추가

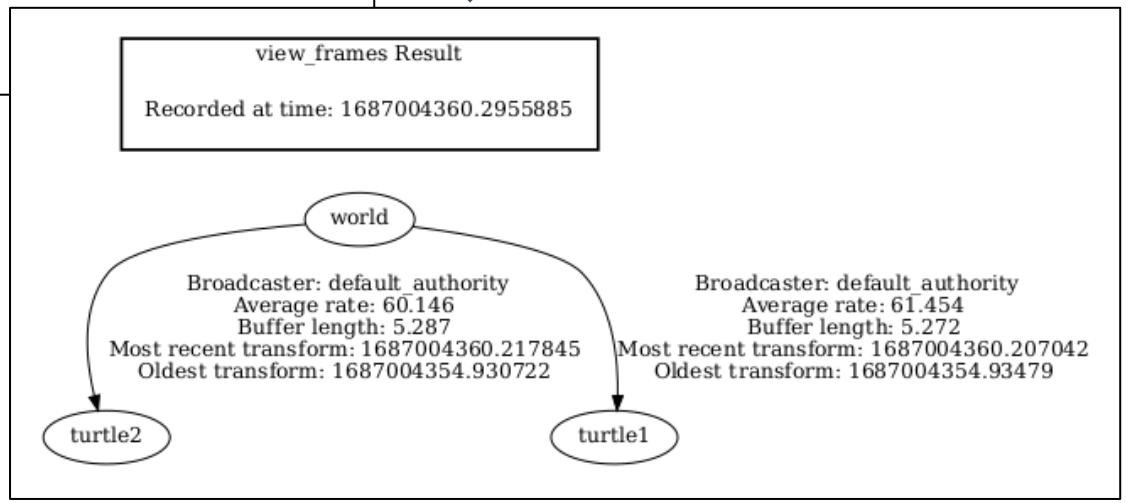


turtlesim

frame 확인

실행: ros2 run tf2_tools view_frames.py

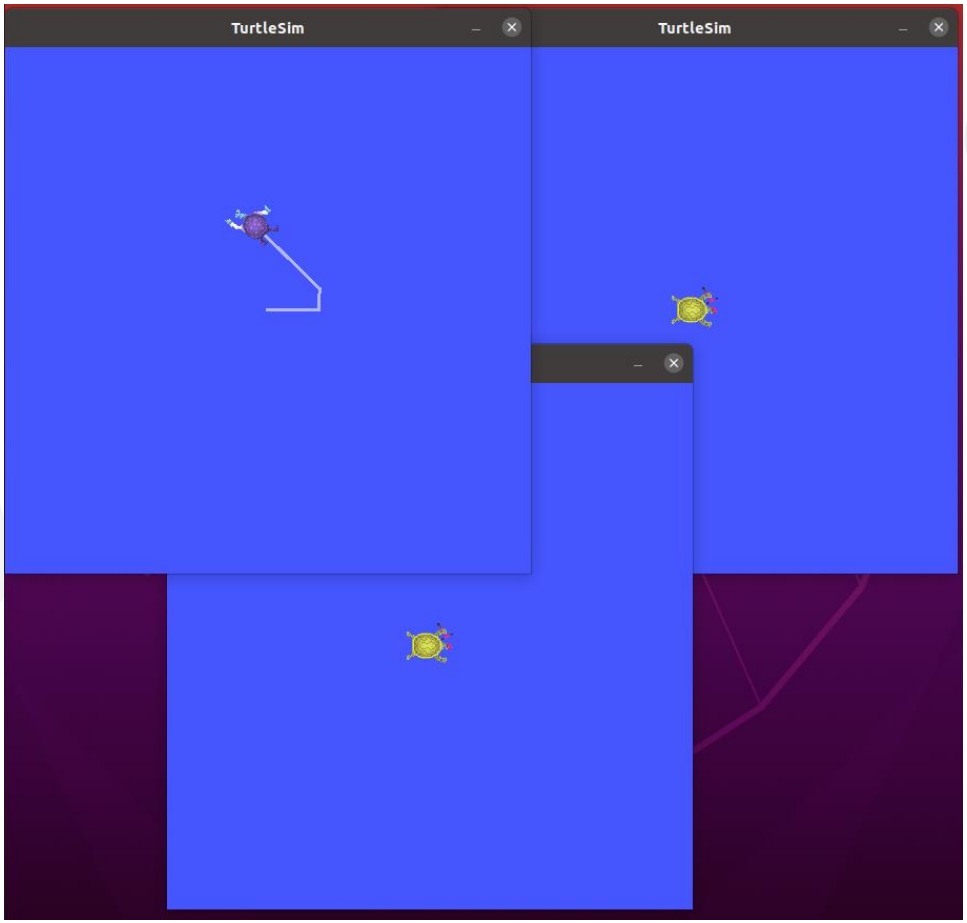




turtlesim

launch file 실행

실행: $ ros2 launch turtlesim multisim.launch.py



turtlesim

bag 명령어

bag 기능

시스템의 topics가 published한 데이터를 기록하기 위한 도구.

이 도구는 여러 topics에 걸쳐 전달된 데이터를 축적 및 데이터베이스에 저장합니다. 다음, 데이터를 재생하여 테스트 및 실험 결과를 재현할 수 있슴

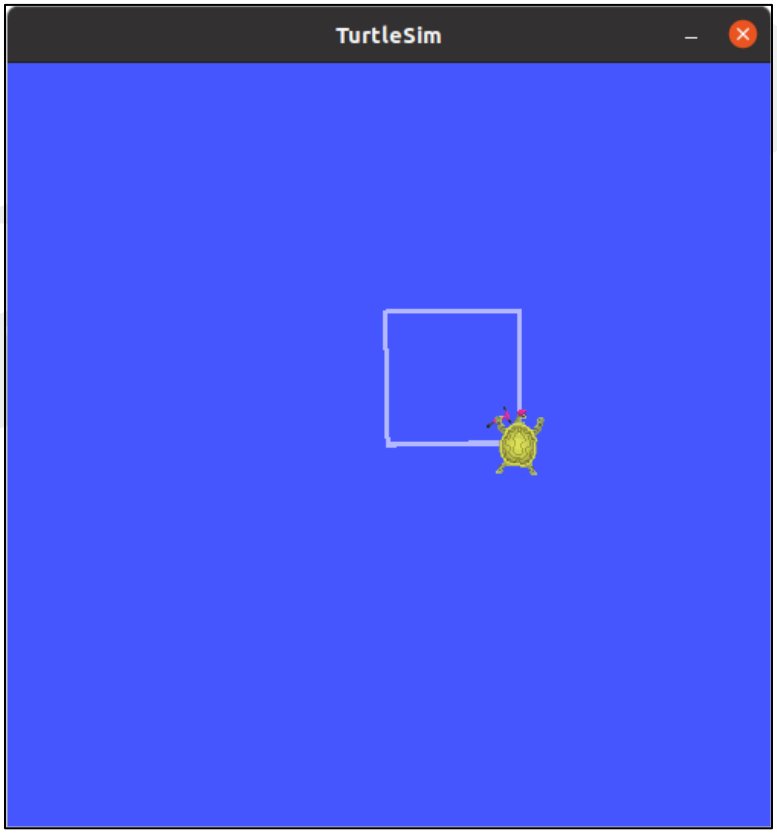
turtlesim

bag 명령어

example record/play

$ ros2 run turtlesim turtlesim_node

$ r ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py

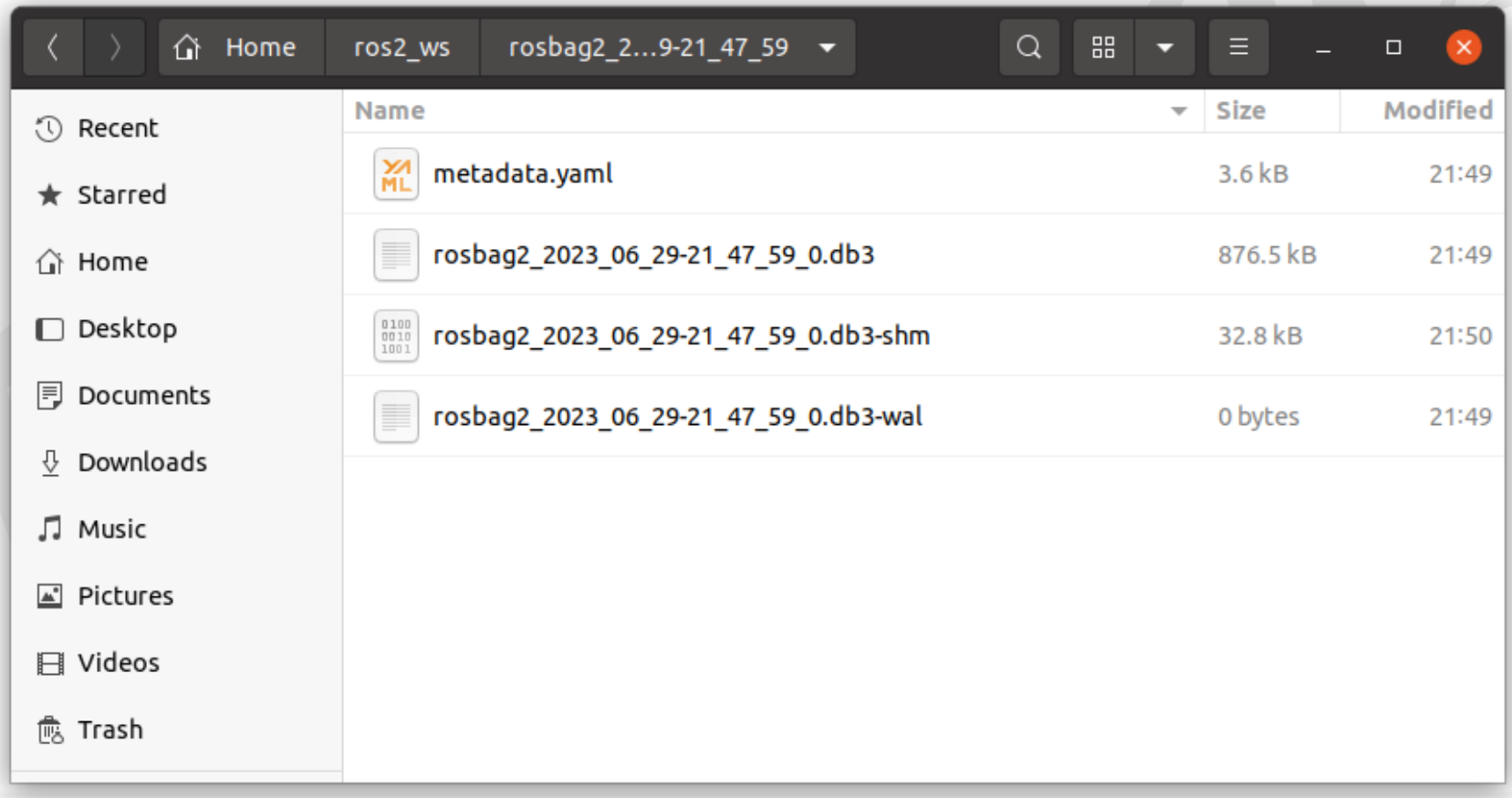


turtlesim

bag 명령어

bag check > saved file list

$ ros2 bag record -a

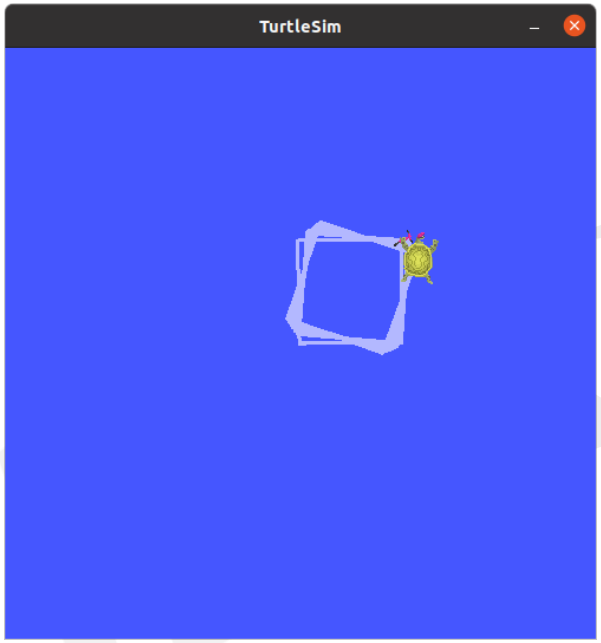


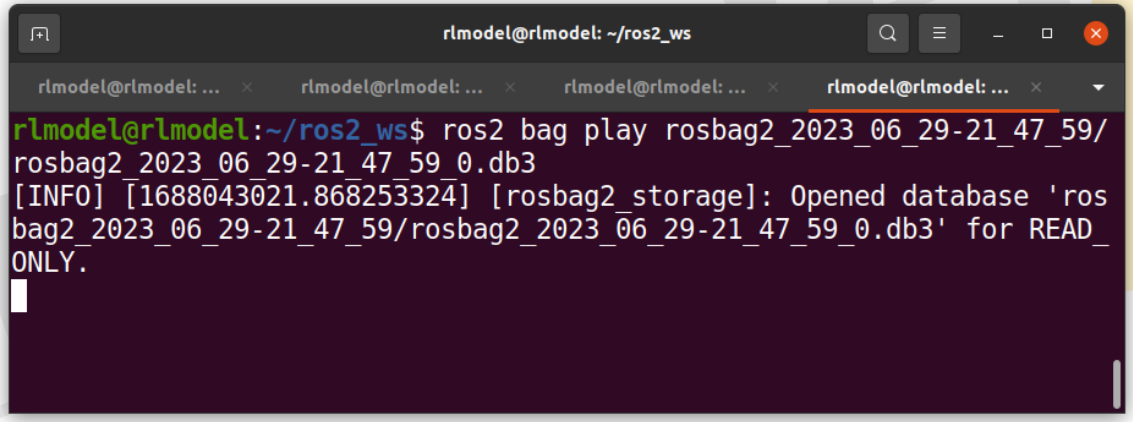
turtlesim

bag 명령어

bag 실행 - topic replay

$ ros2 bag play rosbag2_xxx



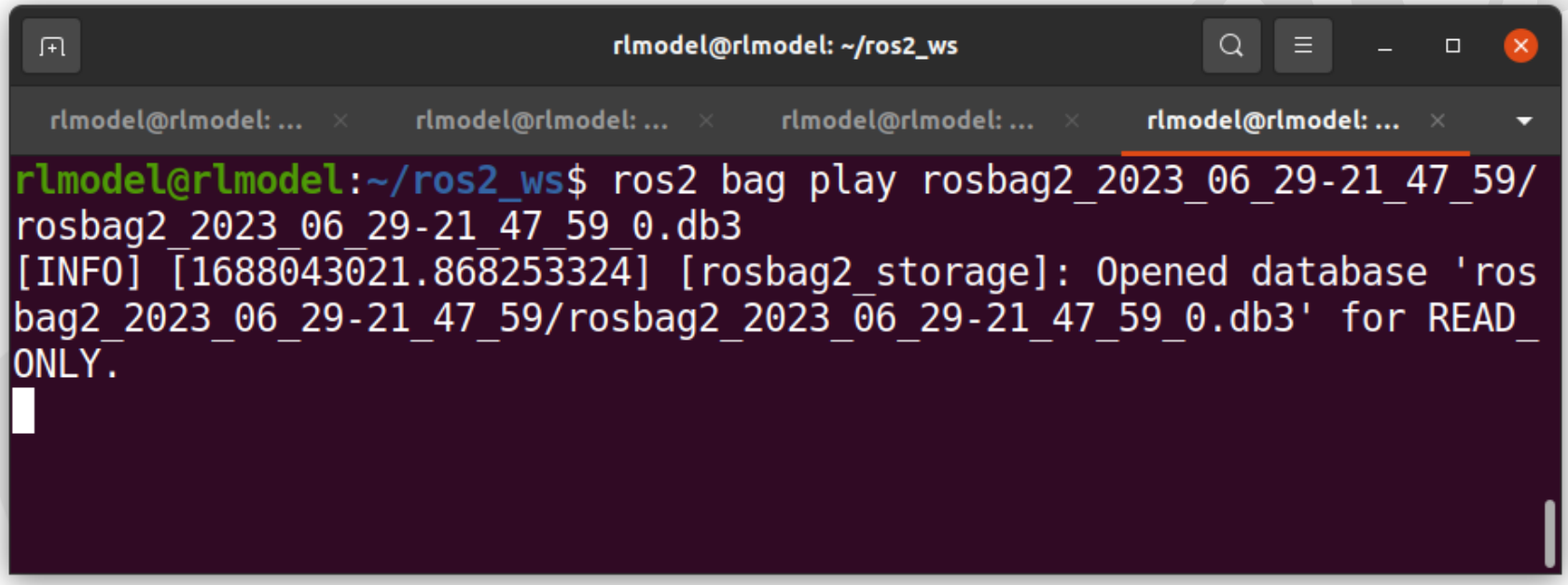


turtlesim

bag 명령어

bag 실행

$ ros2 bag play rosbag2_xxx



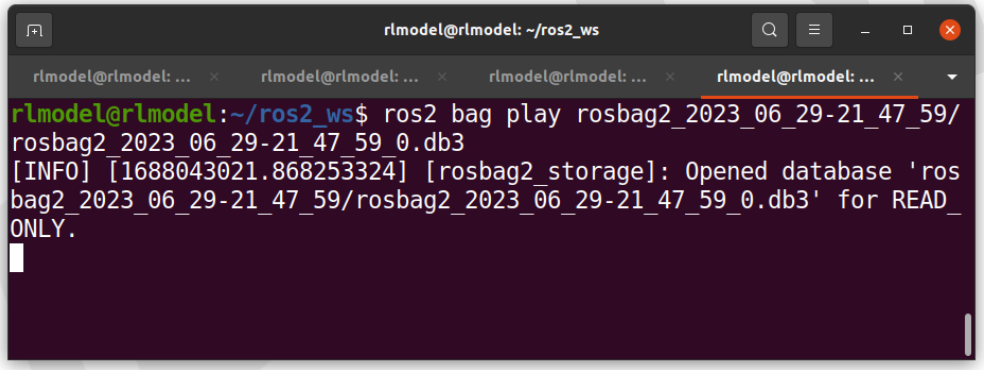
turtlesim

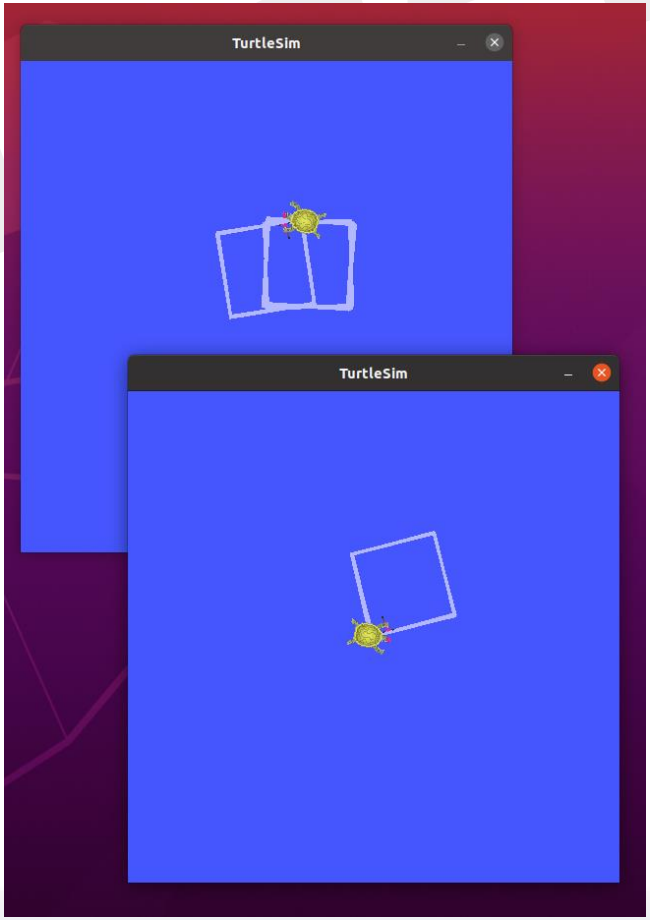
bag 명령어

bag 실행

$ ros2 run turtlesim turtlesim_node __node:=new_turtle

$ ros2 bag play rosbag2_xxx



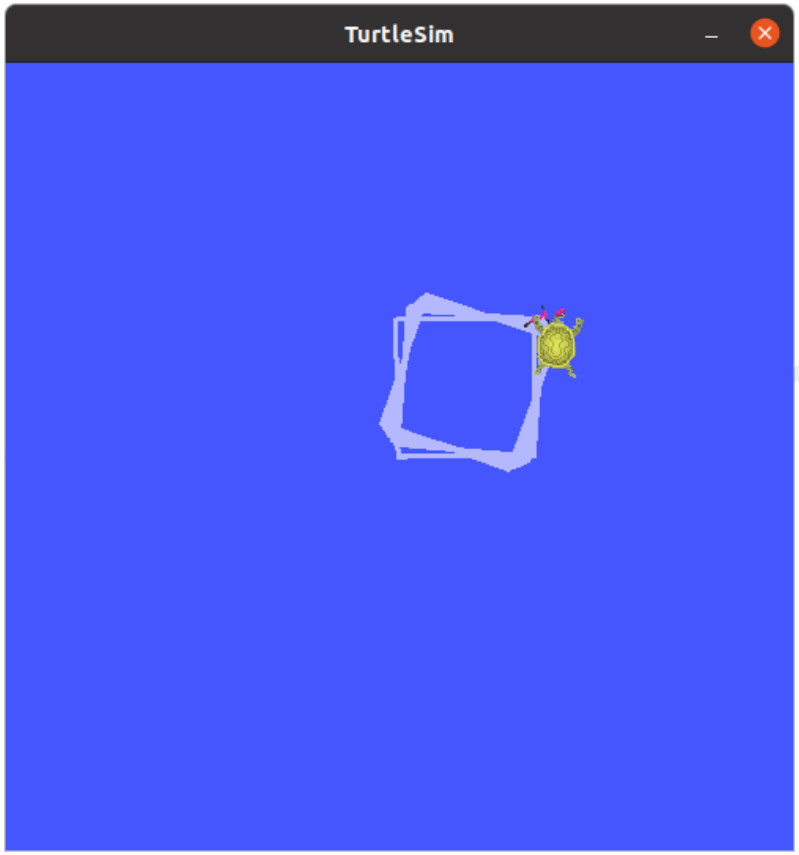


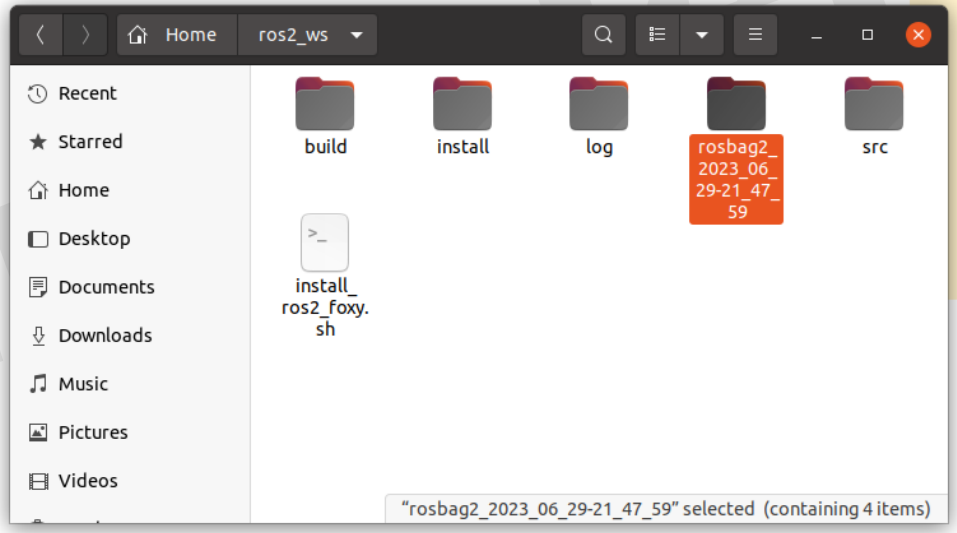
turtlesim

bag 명령어

bag 실행

$ ros2 bag record -a



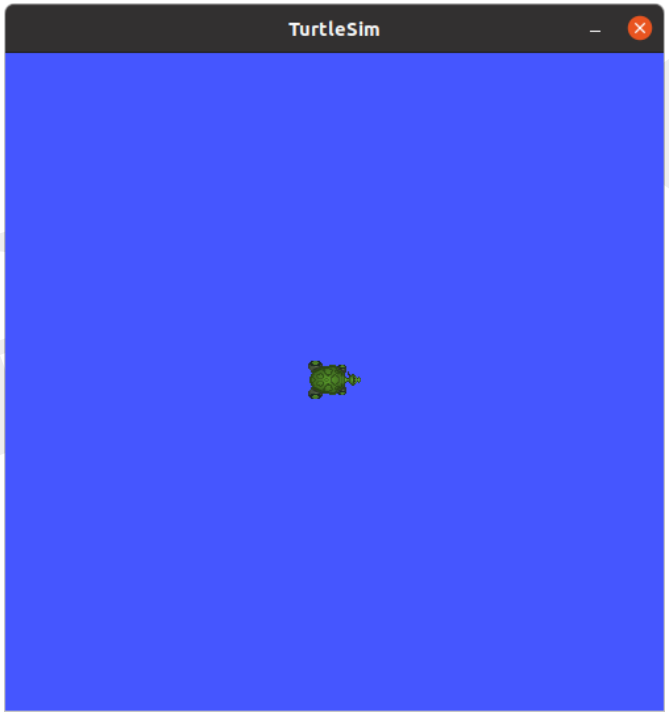


turtlesim

doctor 명령어

ros2 status check

$ ros2 doctor

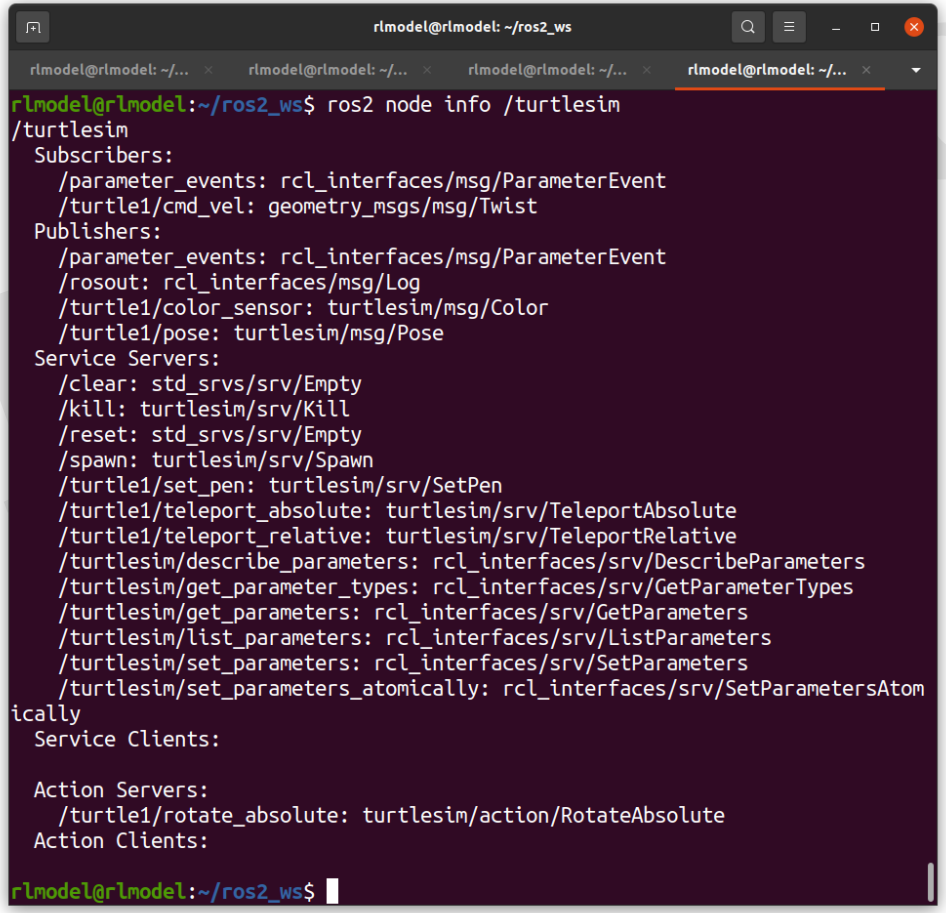


turtlesim

doctor 명령어

ros2 status check

$ ros2 node info /turtlesim

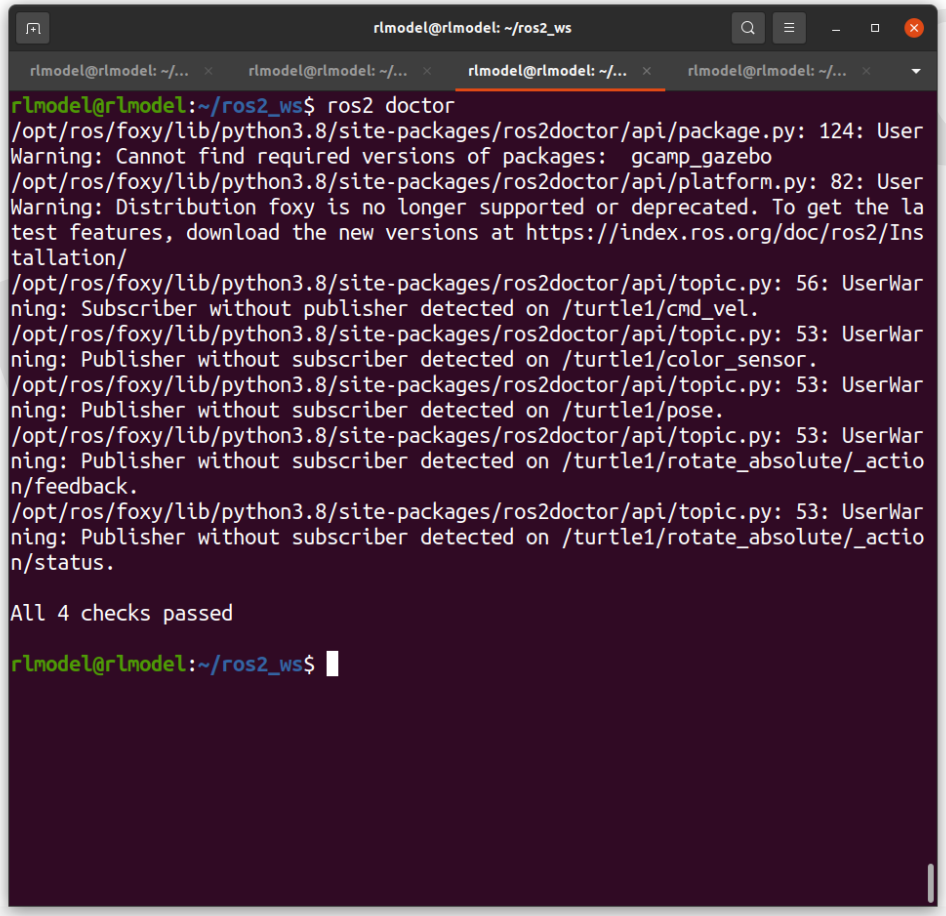


turtlesim

doctor 명령어

ros2 status check

$ ros2 doctor



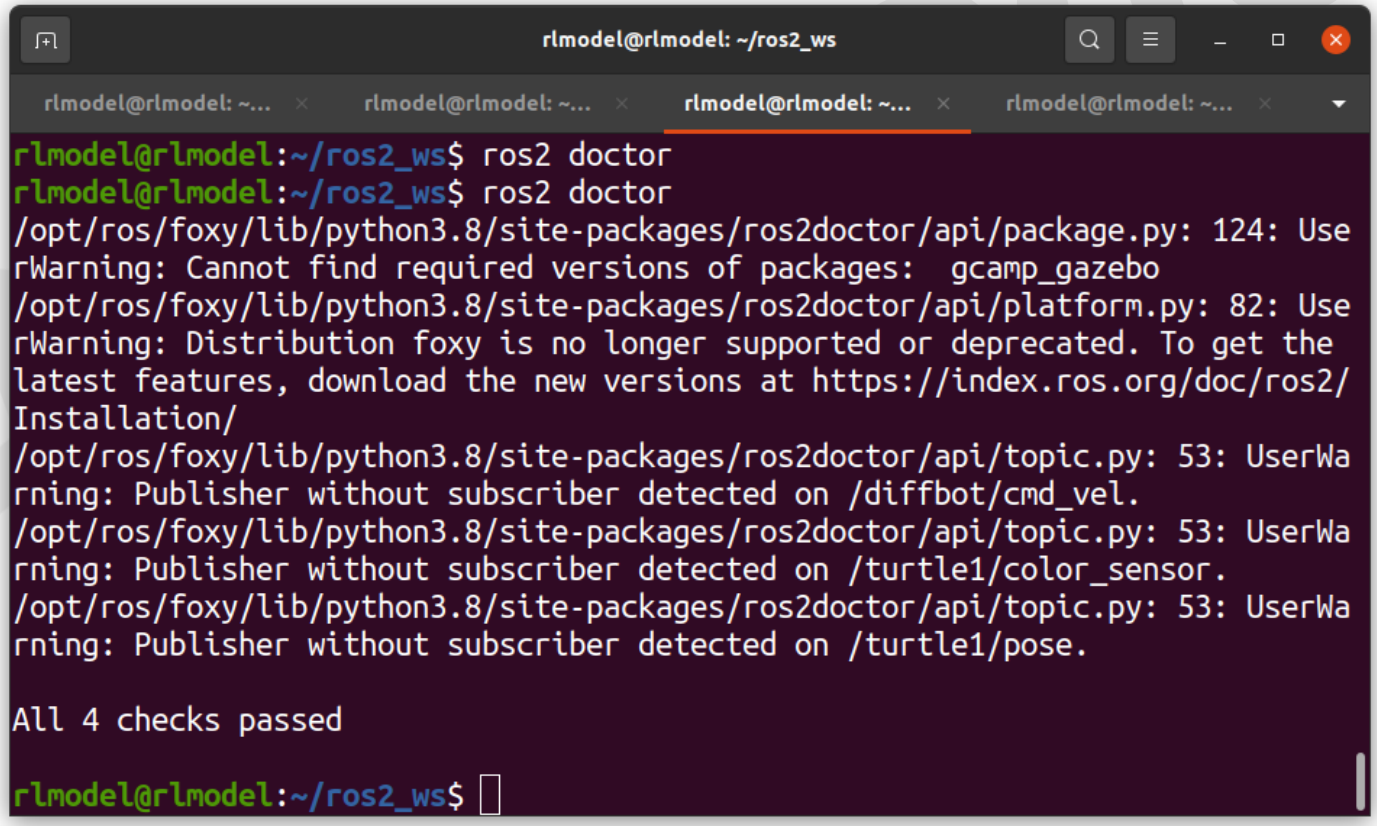
turtlesim

doctor 명령어

ros2 status check

$ ros2 run turtlesim turtle_teleop_key

$ ros2 doctor



turtlesim

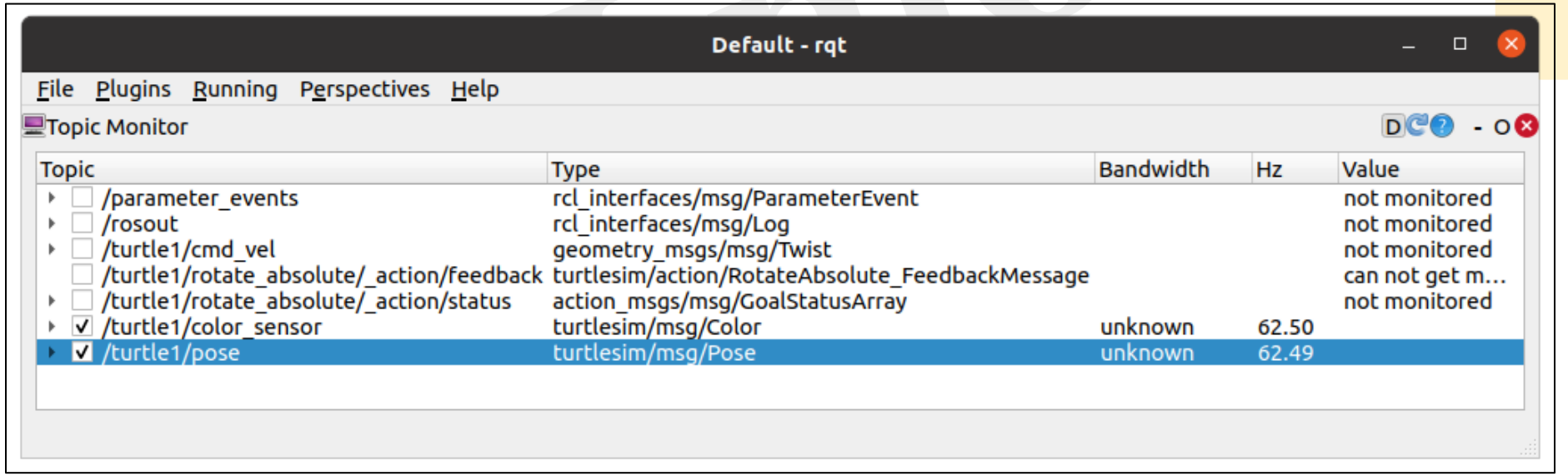
doctor 명령어

ros2 status check

$ rqt

> topic subscribe

- turtle1/colcor_sensor
- turtle1/pose

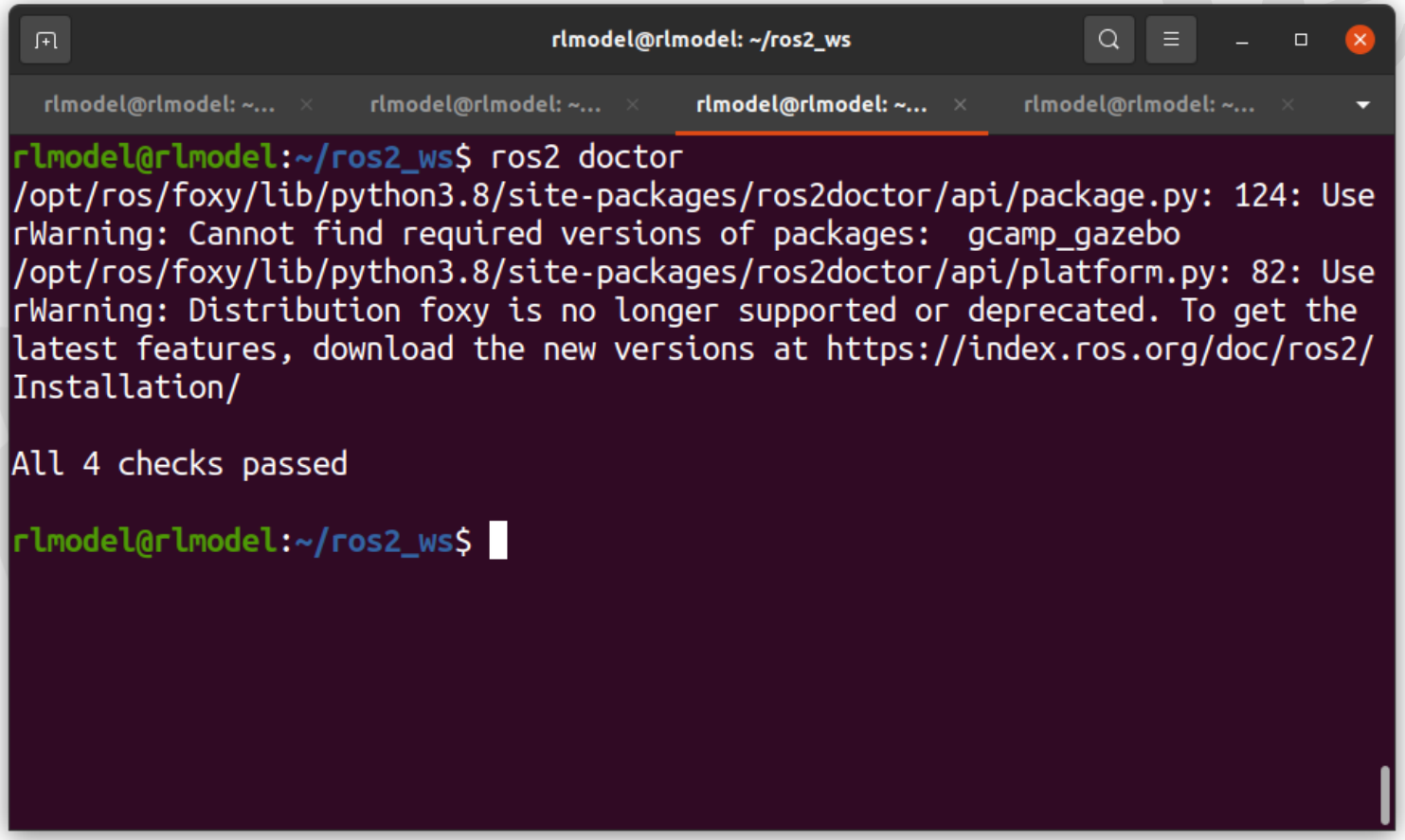


turtlesim

doctor 명령어

ros2 status check

$ ros2 doctor

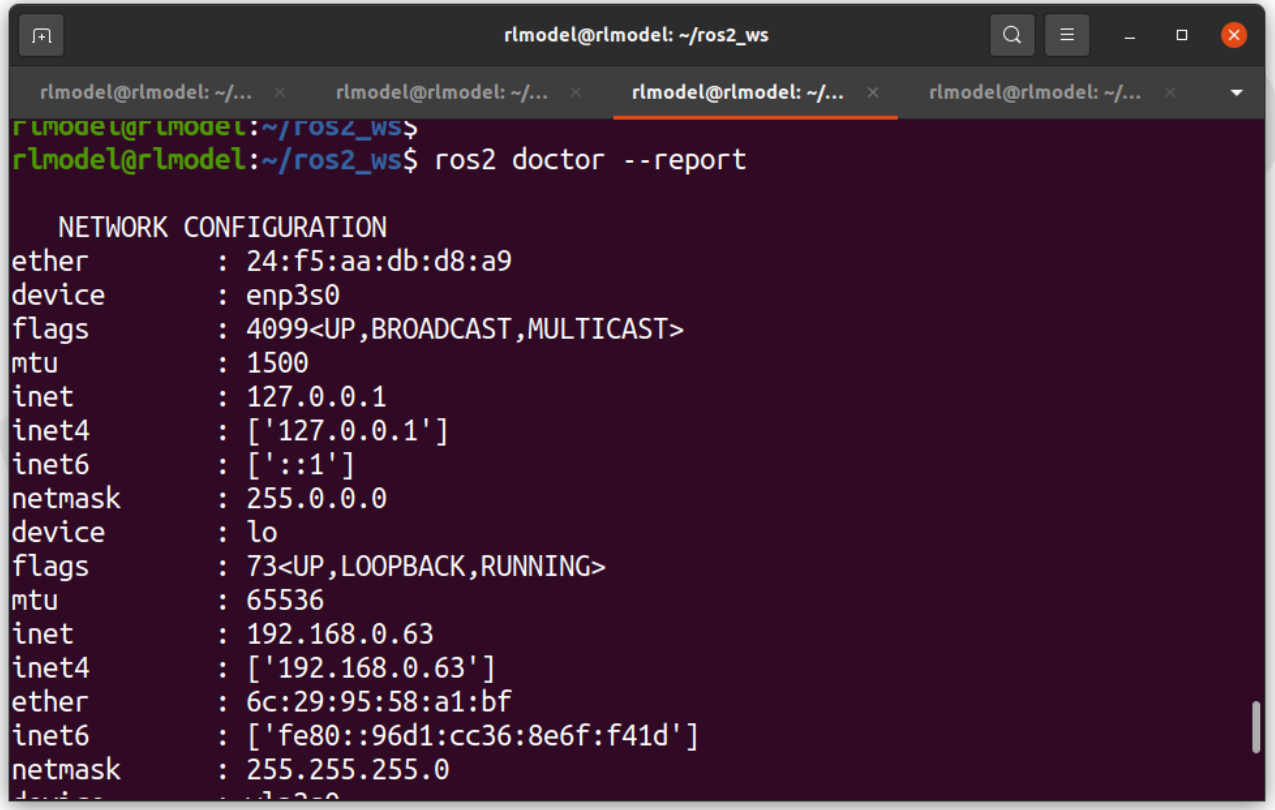


turtlesim

doctor 명령어

ros2 status check

$ ros2 doctor --report



NETWORK CONFIGURATION
...

PLATFORM INFORMATION
...

RMW MIDDLEWARE
...

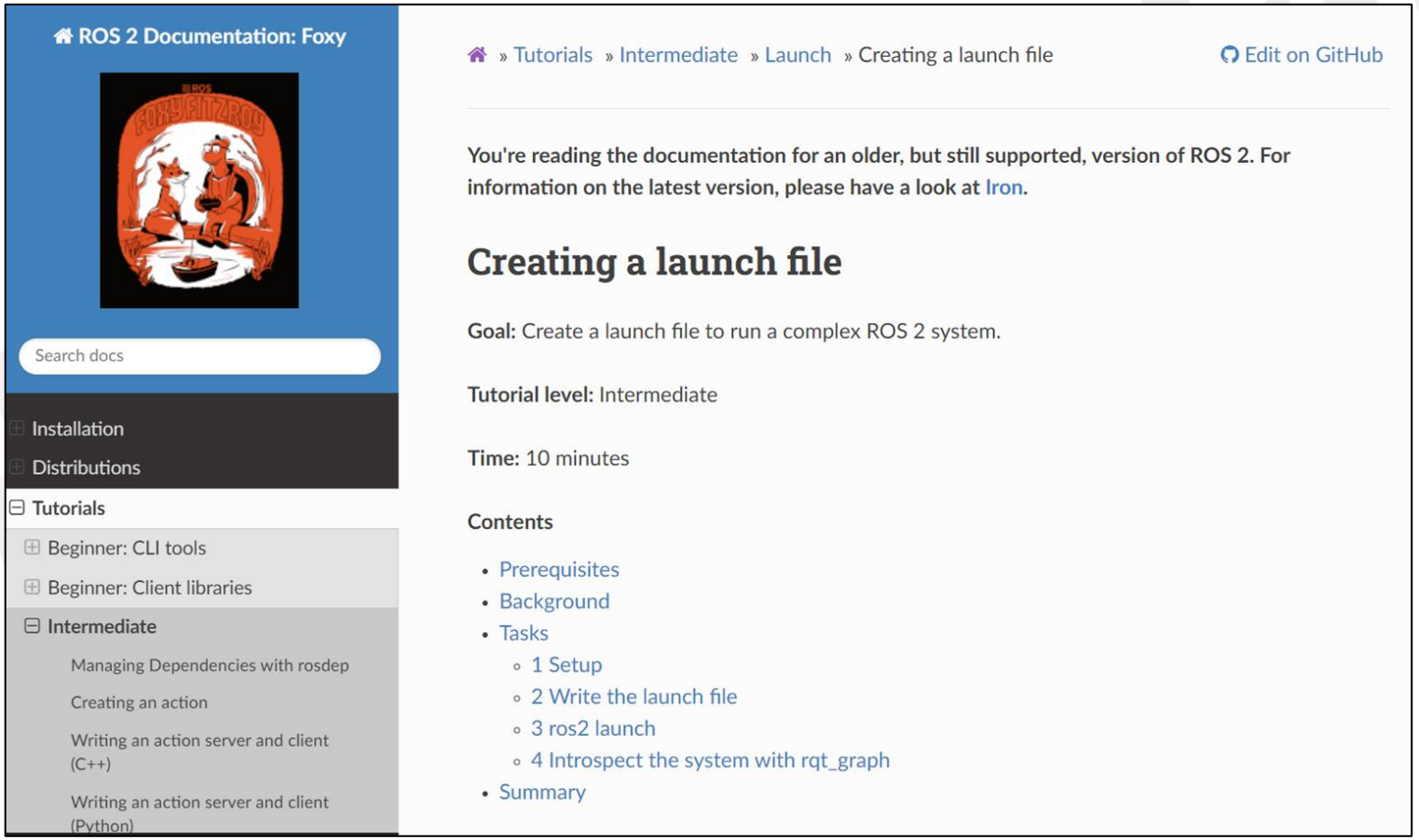
ROS 2 INFORMATION
...

TOPIC LIST

turtlesim

launch file 실습

link: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html



turtlesim

launch file 생성

$ mkdir launch
$ gedit first_turtlesim_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
    Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
    ),
            ]
    )
```

turtlesim

launch file 생성

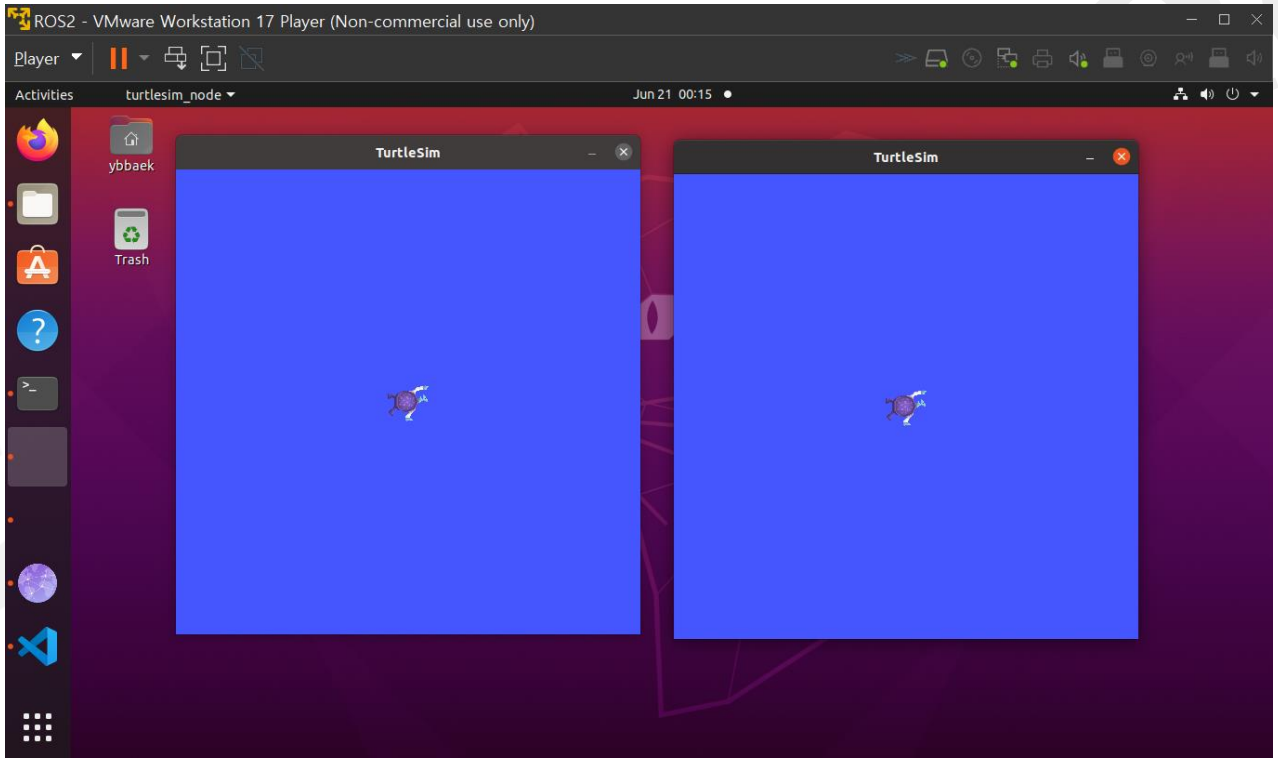
$ gedit second_turtlesim_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

turtlesim

launch file 실행

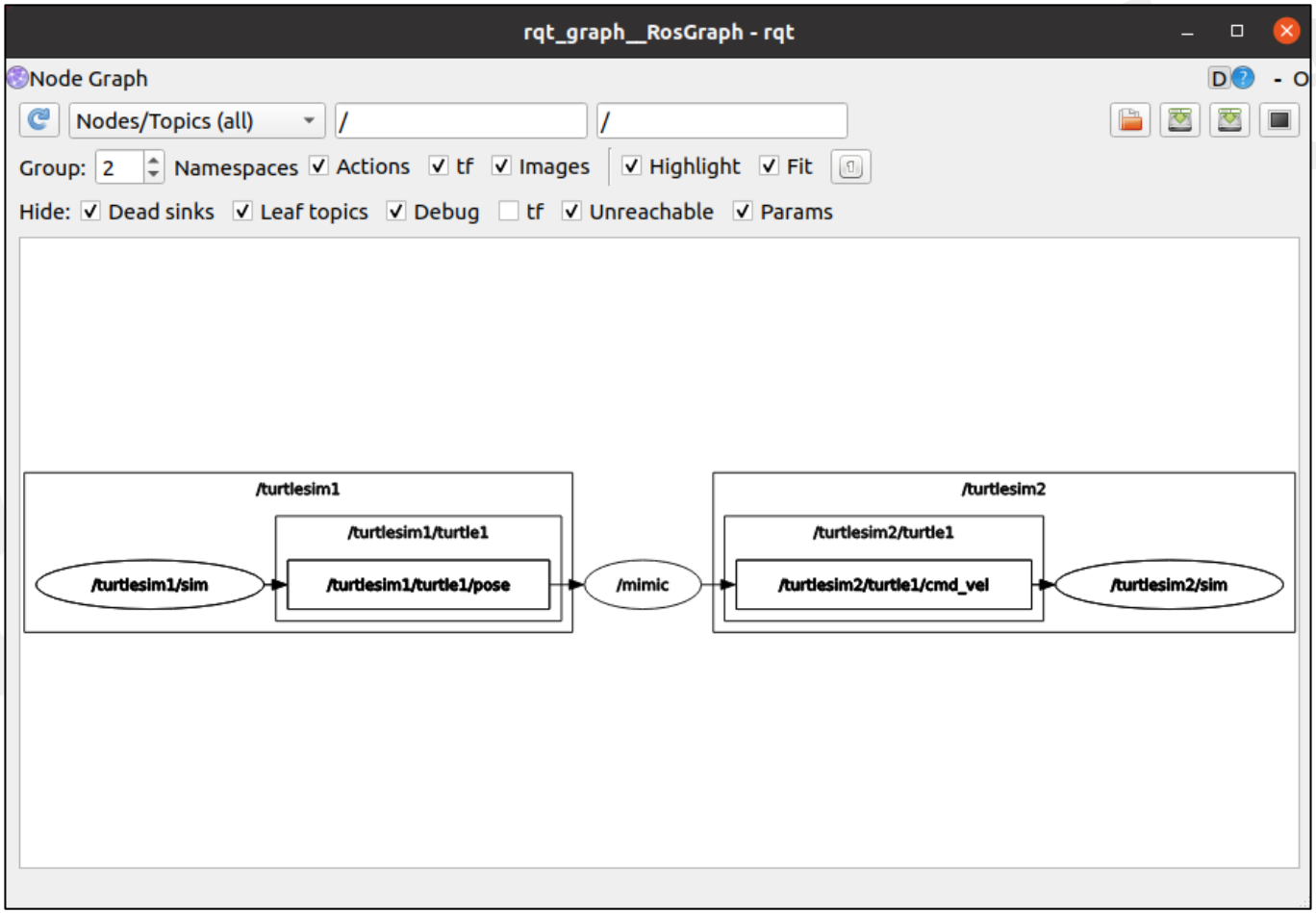
```
$ mkdir launch
$ cd launch
$ ros2 launch first_turtlesim_launch.py
```



turtlesim

launch file 실행

rqt_graph 확인

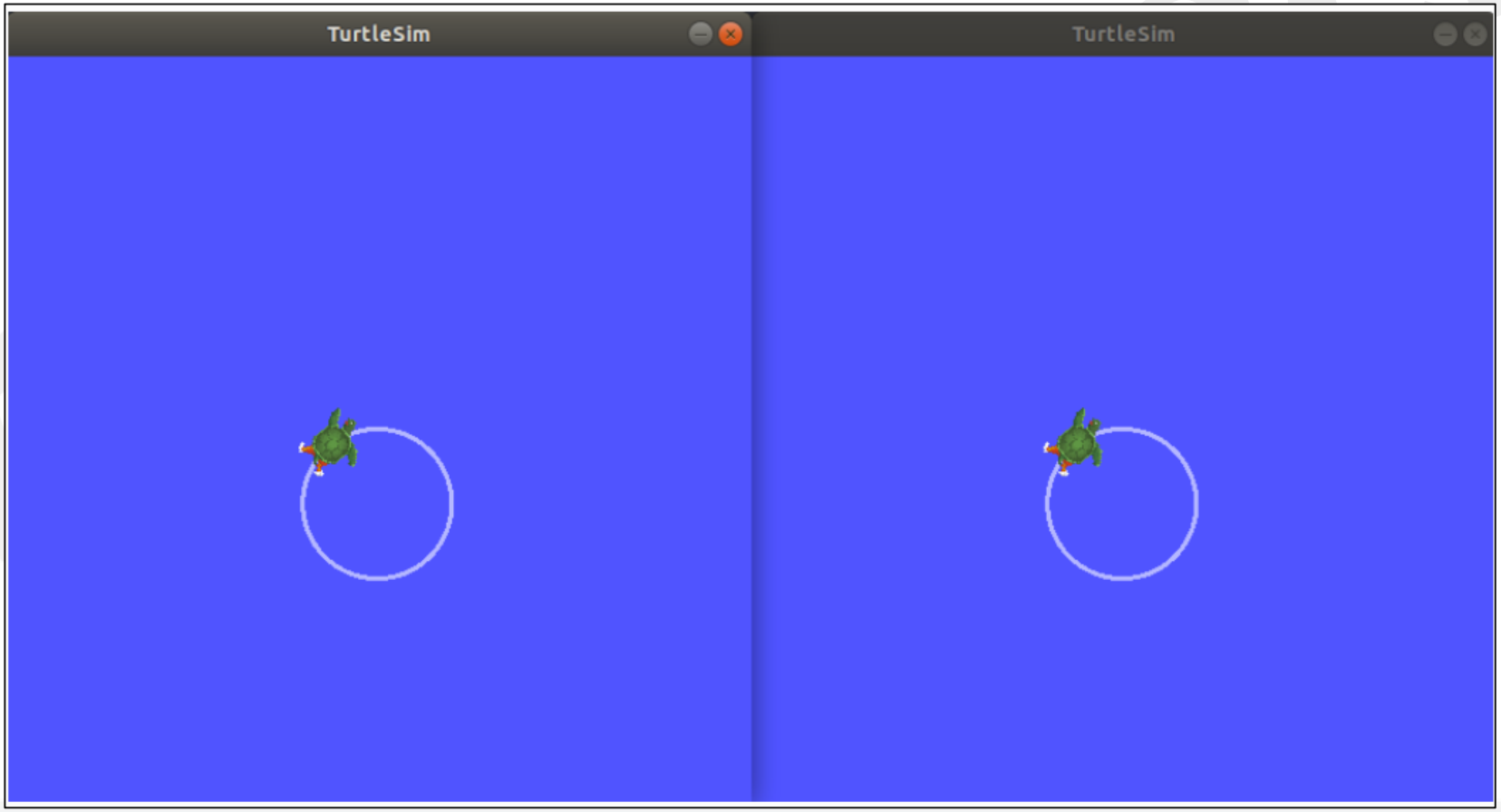


turtlesim

launch file 실행

연동 확인

> ros2 topic pub -r 1 /turtlesim1/turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.8}}"



turtlesim

default logger level

levels: Fatal, Error, Warn, Info, Debug

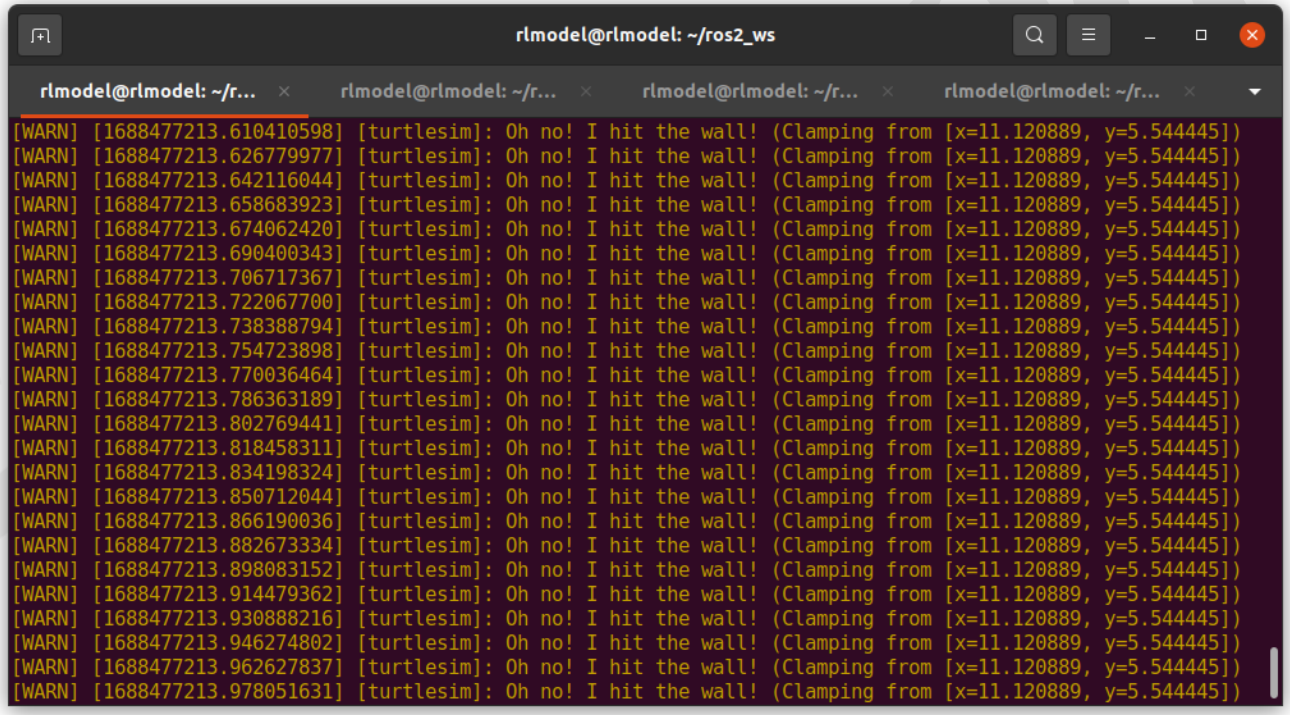
- `Fatal` messages indicate the system is going to terminate to try to protect itself from detriment.
- `Error` messages indicate significant issues that won't necessarily damage the system, but are preventing it from functioning properly.
- `Warn` messages indicate unexpected activity or non-ideal results that might represent a deeper issue, but don't harm functionality outright.
- `Info` messages indicate event and status updates that serve as a visual verification that the system is running as expected.
- `Debug` messages detail the entire step-by-step process of the system execution.

turtlesim

default logger level

levels: Fatal, Error, Warn, Info, Debug

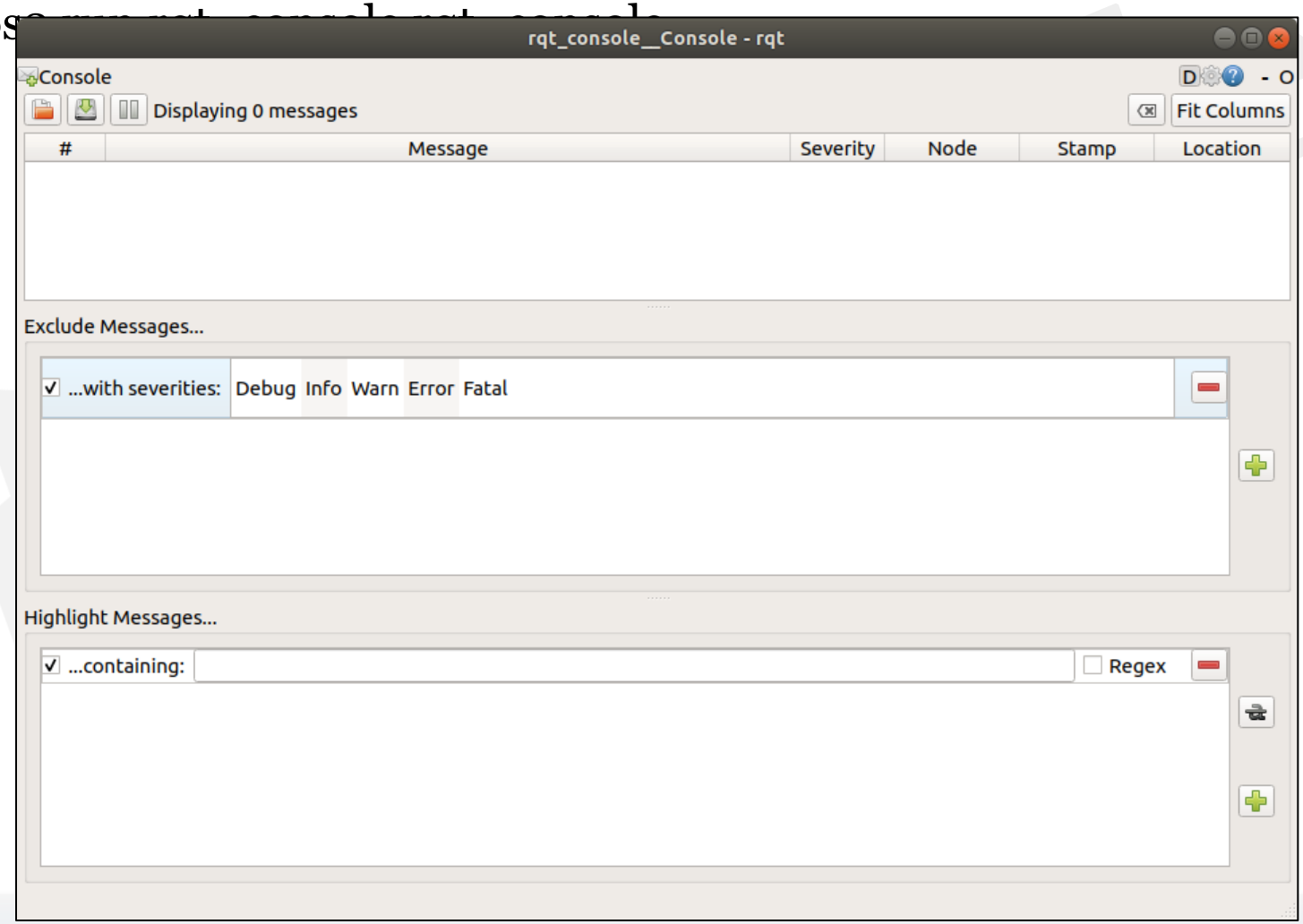
- `ros2 run turtlesim turtlesim_node --ros-args --log-level WARN`



turtlesim

rqt_console

$ ros

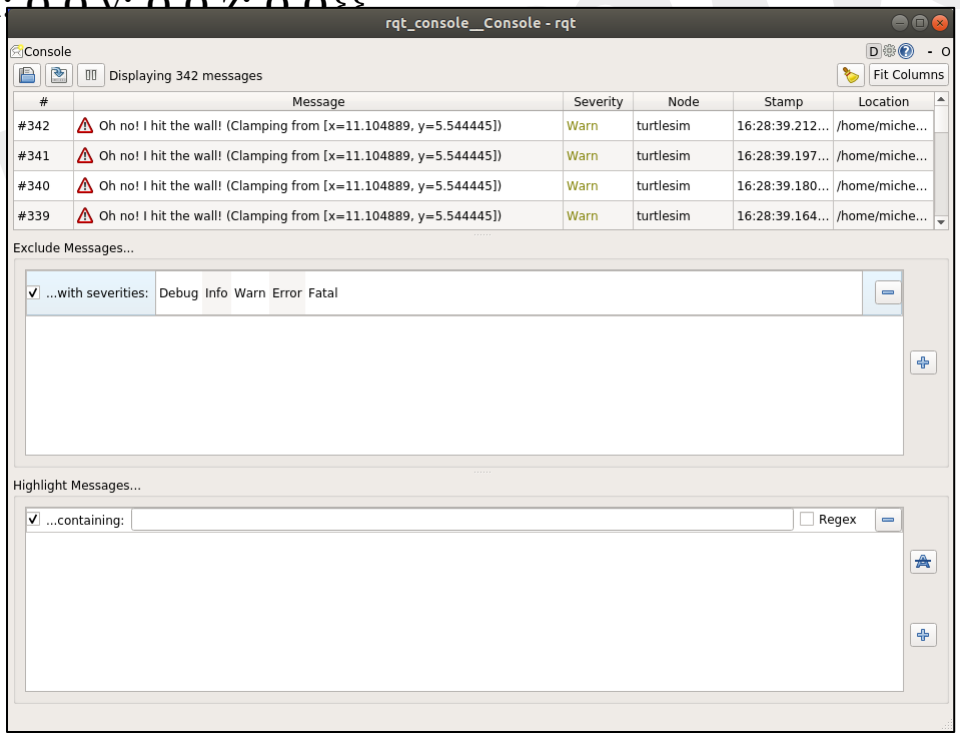


turtlesim

rqt_console

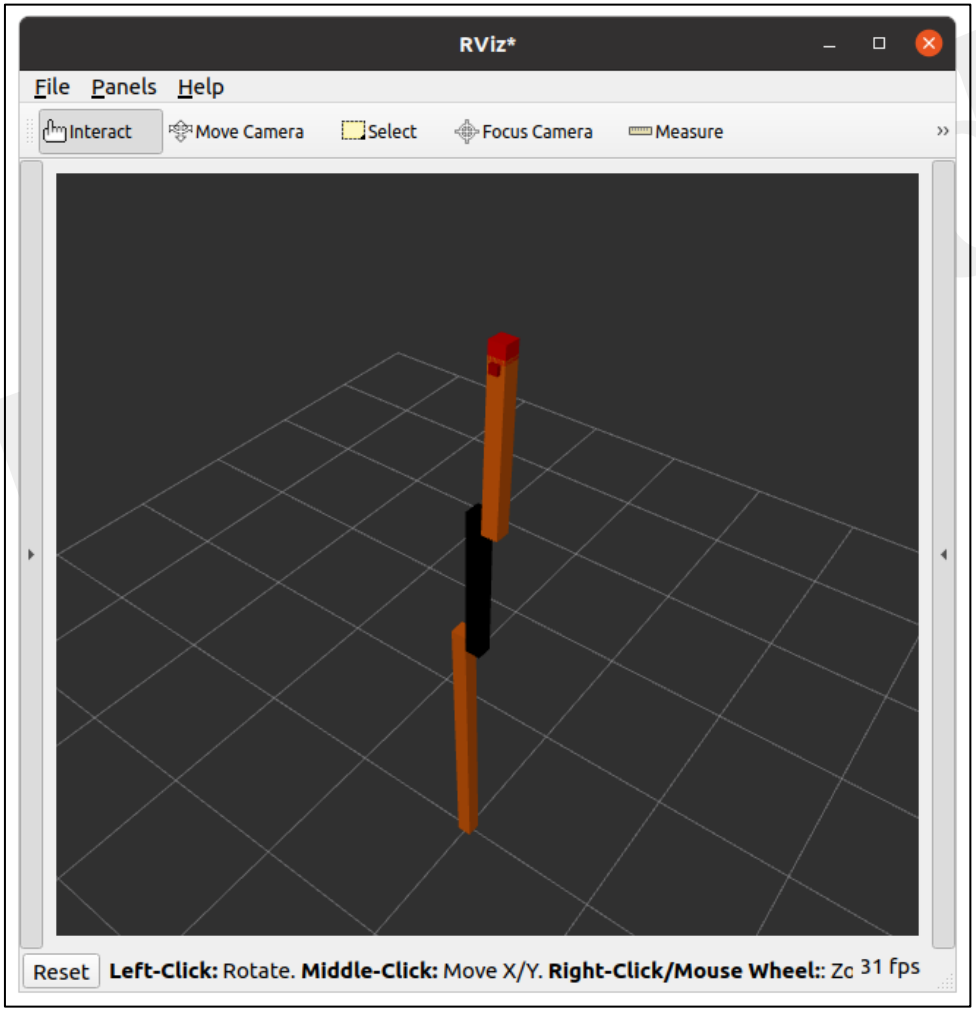
ros2 run rqt_console rqt_console

$ ros2 run turtlesim turtlesim_node
$ ros2 topic pub -r 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0 y: 0.0 z: 0.0}}"



dummy robot

default executable pkg

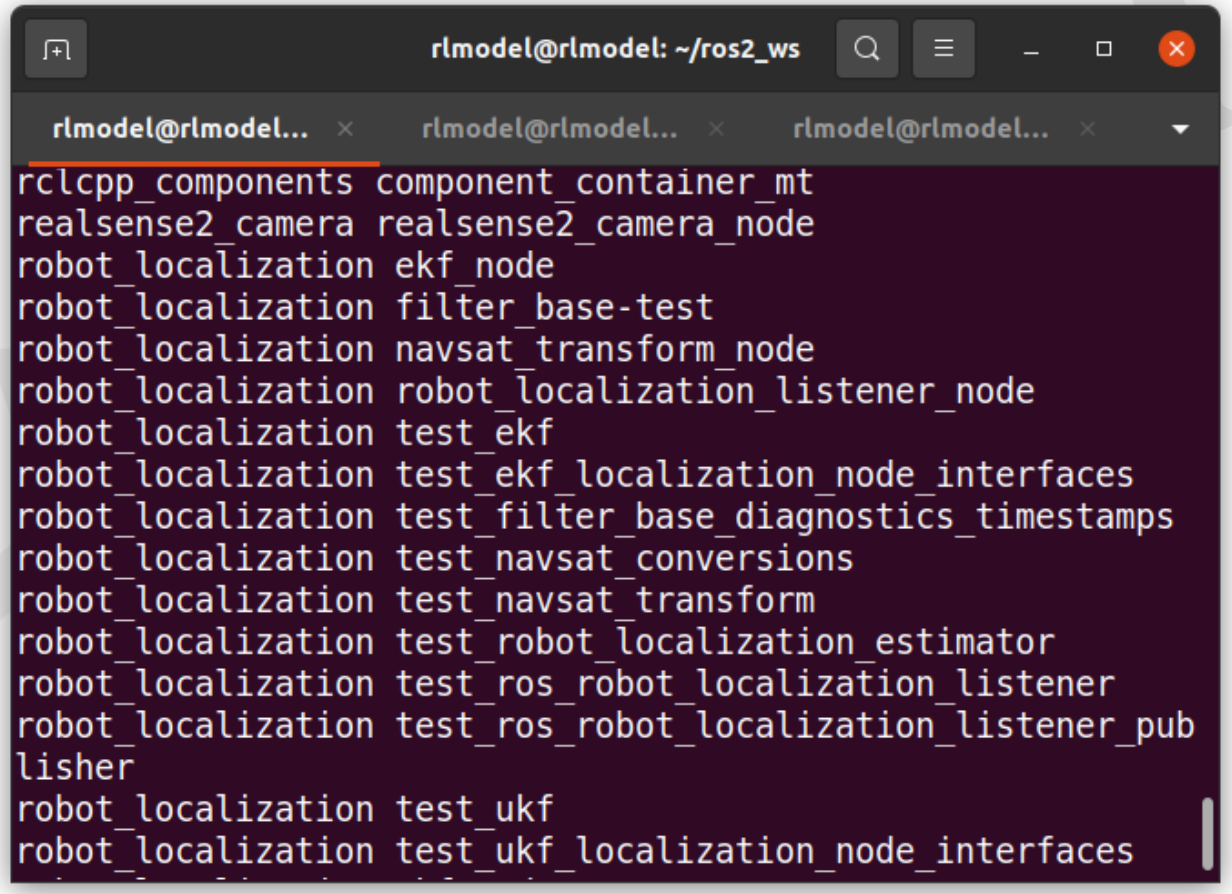


dummy robot

dummy simulation robot 실행

package check

$ ros2 pkg executables



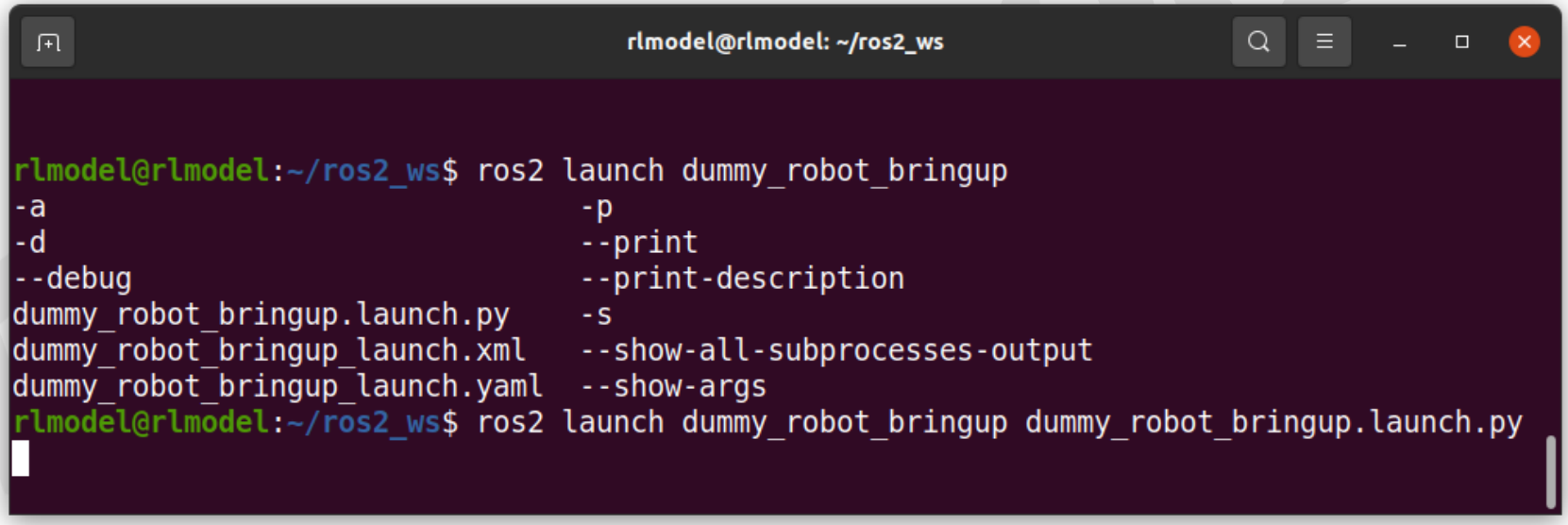
dummy robot

dummy simulation robot 실행

>run dummy_robot

$ source ~/ros2_ws/install/setup.bash

$ ros2 launch dummy_robot_bringup dummy_robot_bringup.launch.py

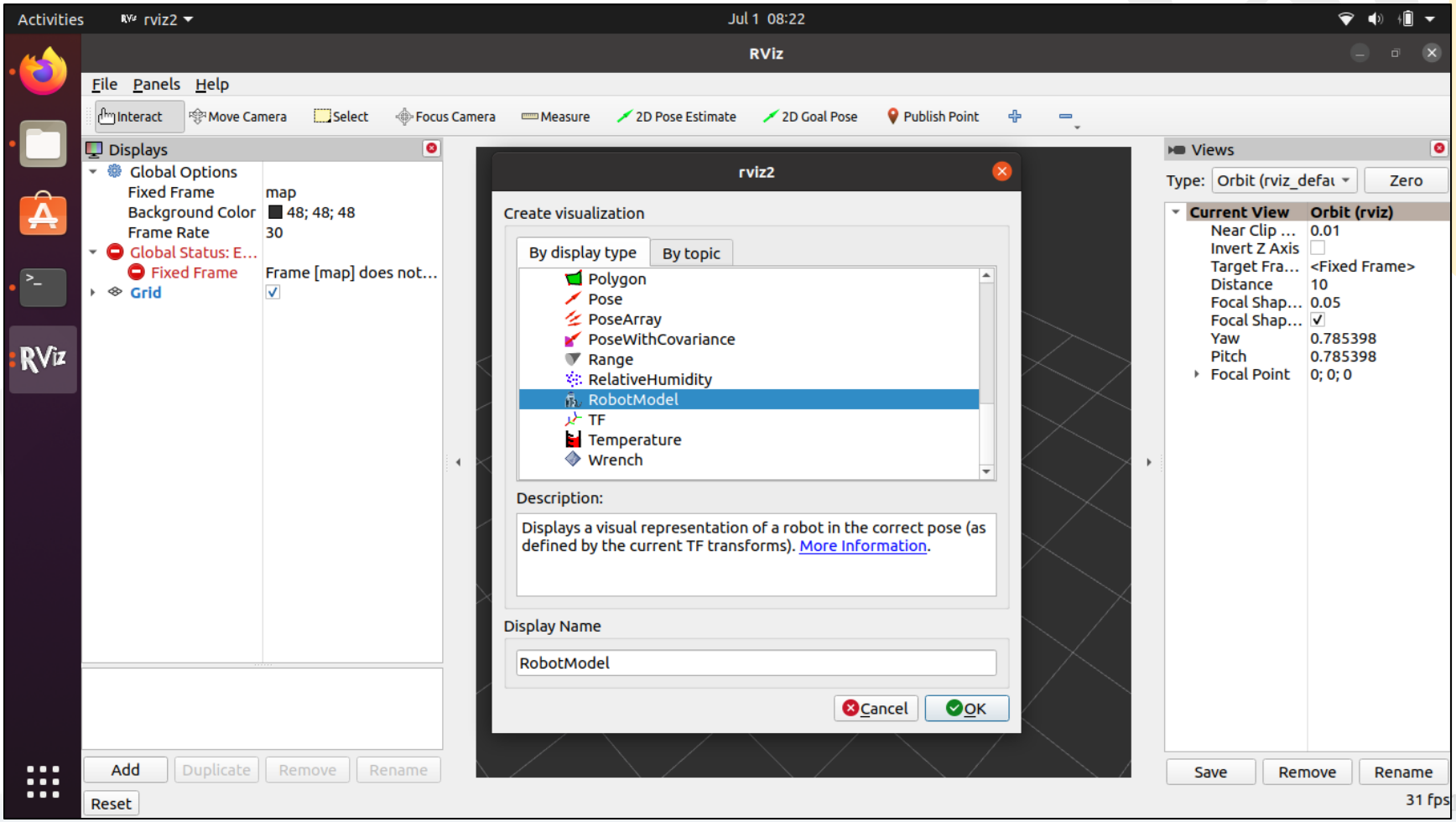


dummy robot

dummy simulation robot 실행

>run rviz

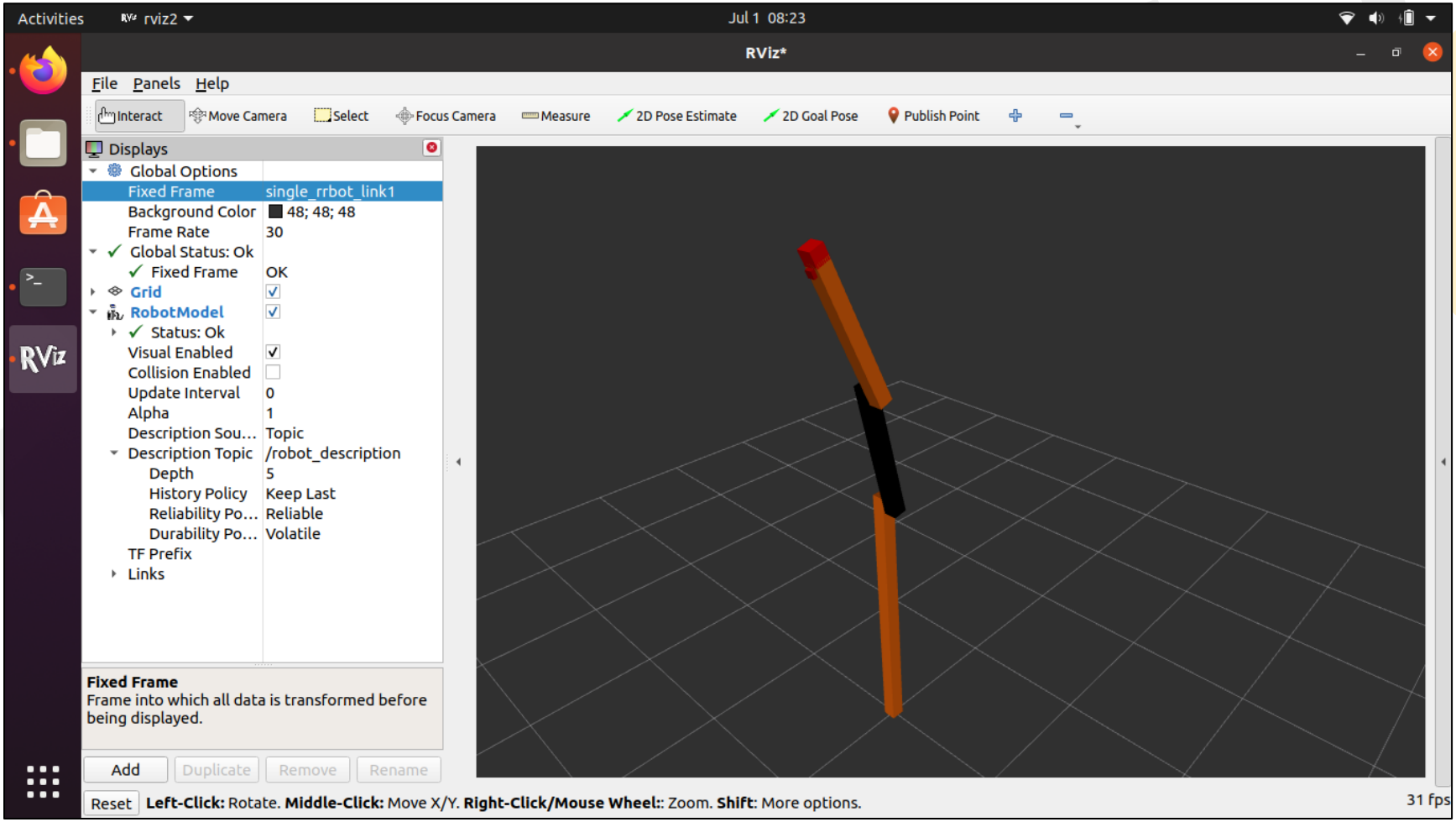
>add - RobotModel



dummy robot

dummy simulation robot 실행

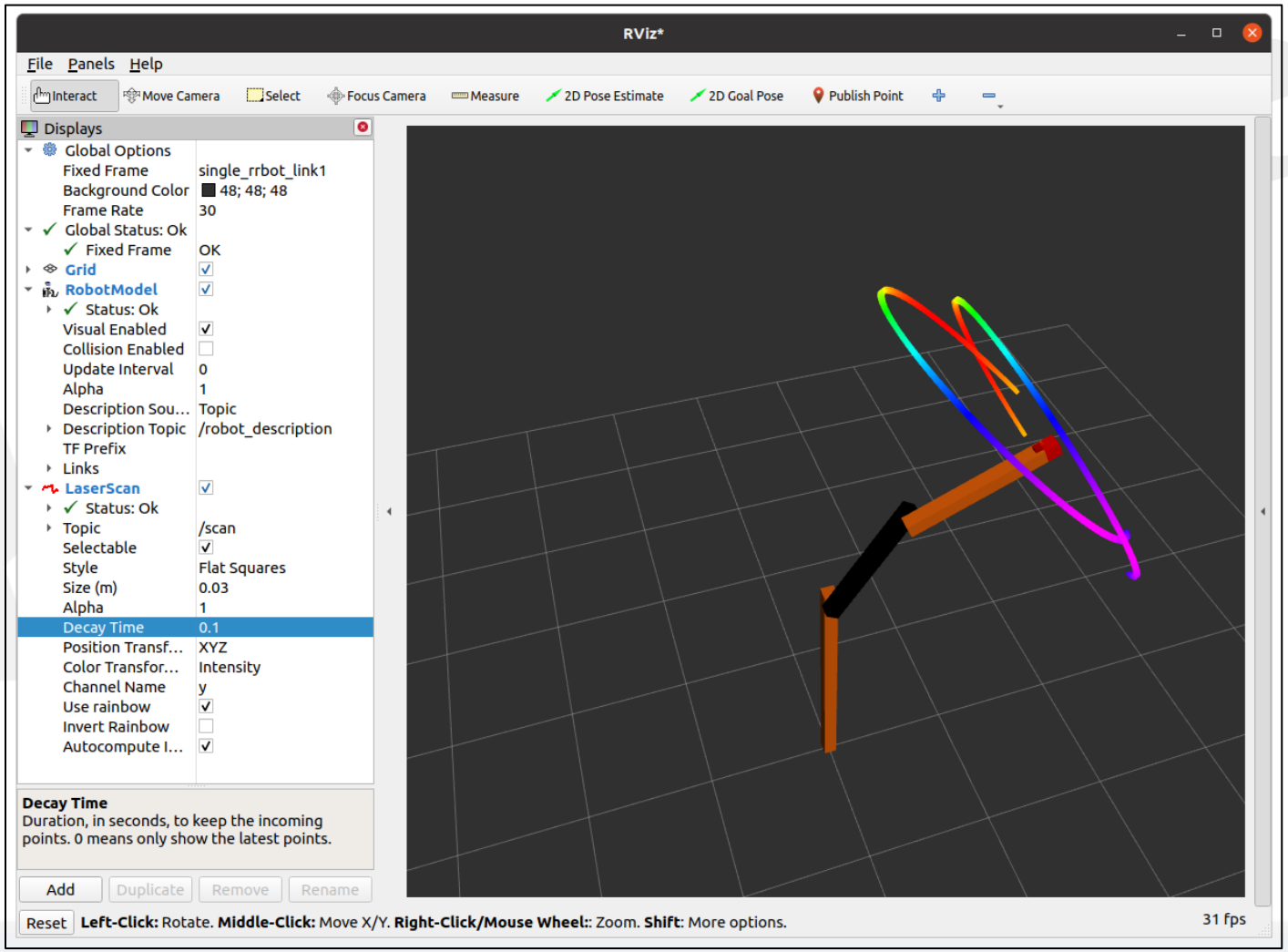
>check TF, Frame id



dummy robot

dummy simulation robot 실행

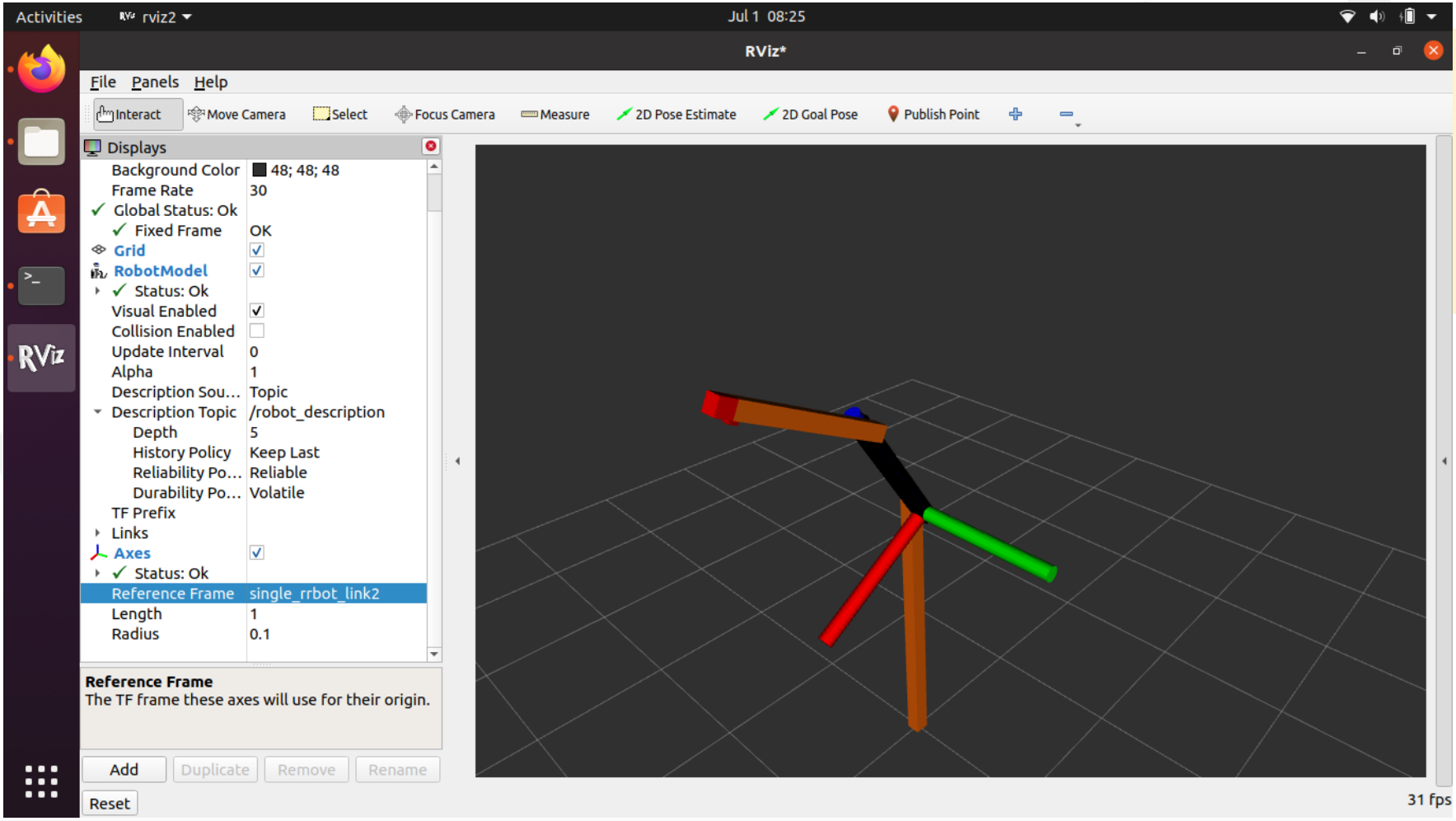
>check /scan topic



dummy robot

dummy simulation robot 실행

>check TF, Frame id



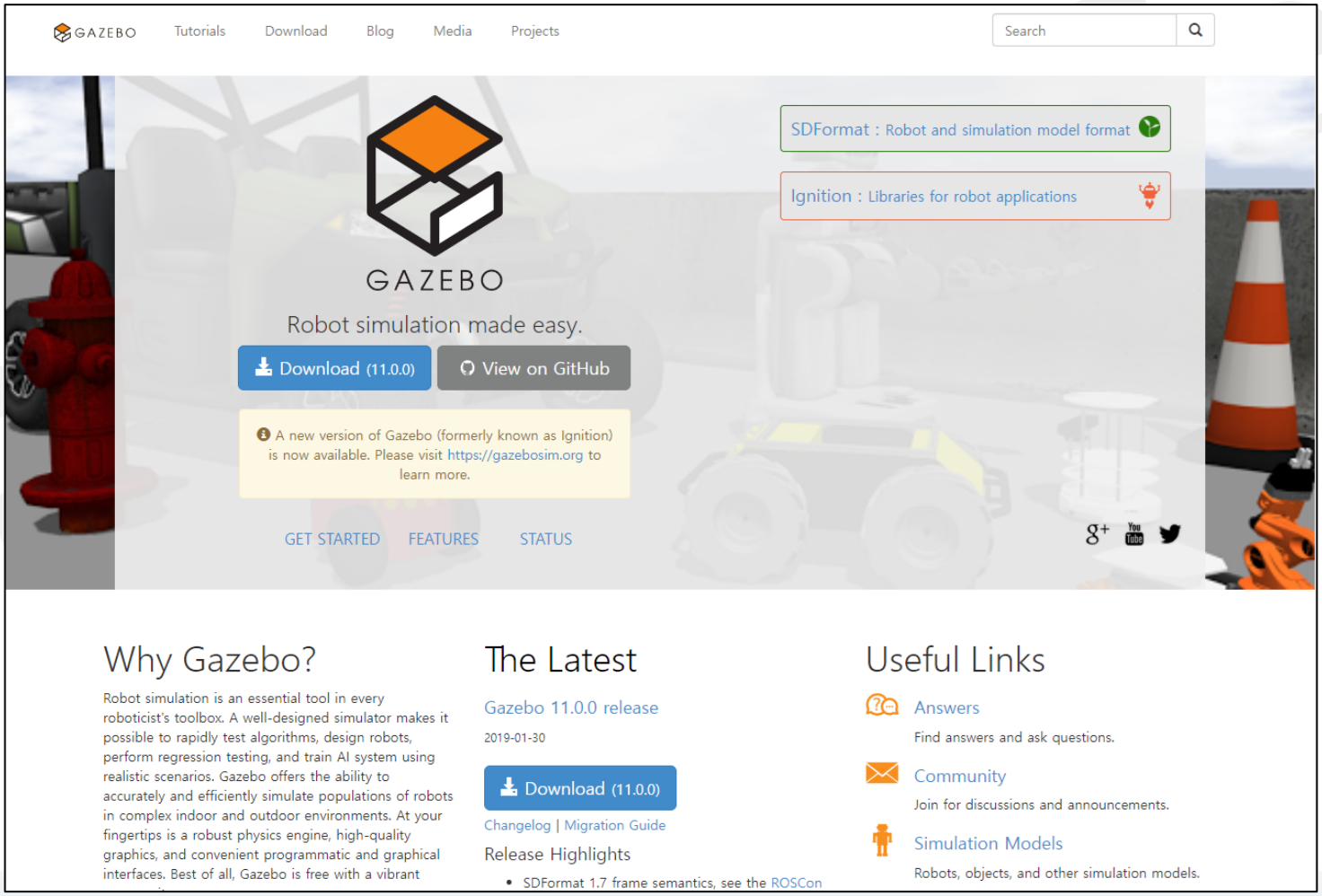
Gazebo

3D simulation

Gazebo 구성

공식문서 (old)

link: https://classic.gazebosim.org/



Gazebo

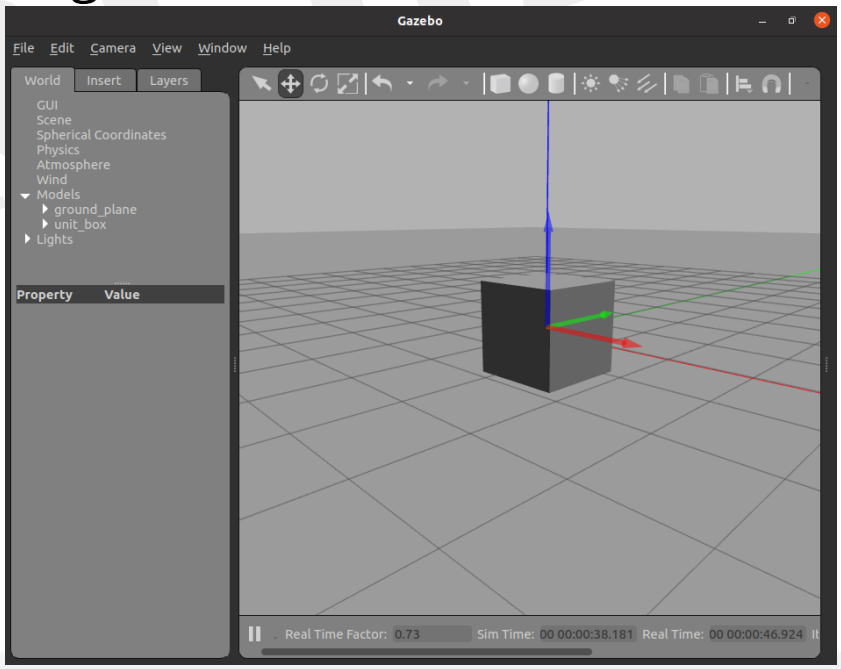
gazebo 소개

개요: Gazebo는 ROS와 별개의 프로그램, 오픈소스 가상 시뮬레이션 툴.

URDF, SDF, Xacro 등의 XML 문서로 로봇을 가상 시뮬레이션에 삽입한 후, 로봇과 사용자가 작성한 프로그램이 메시지를 주고받으며 동작하는 모습을 확인할 수 있음.

설치: sudo apt install gazebo11

실행: $ gazebo

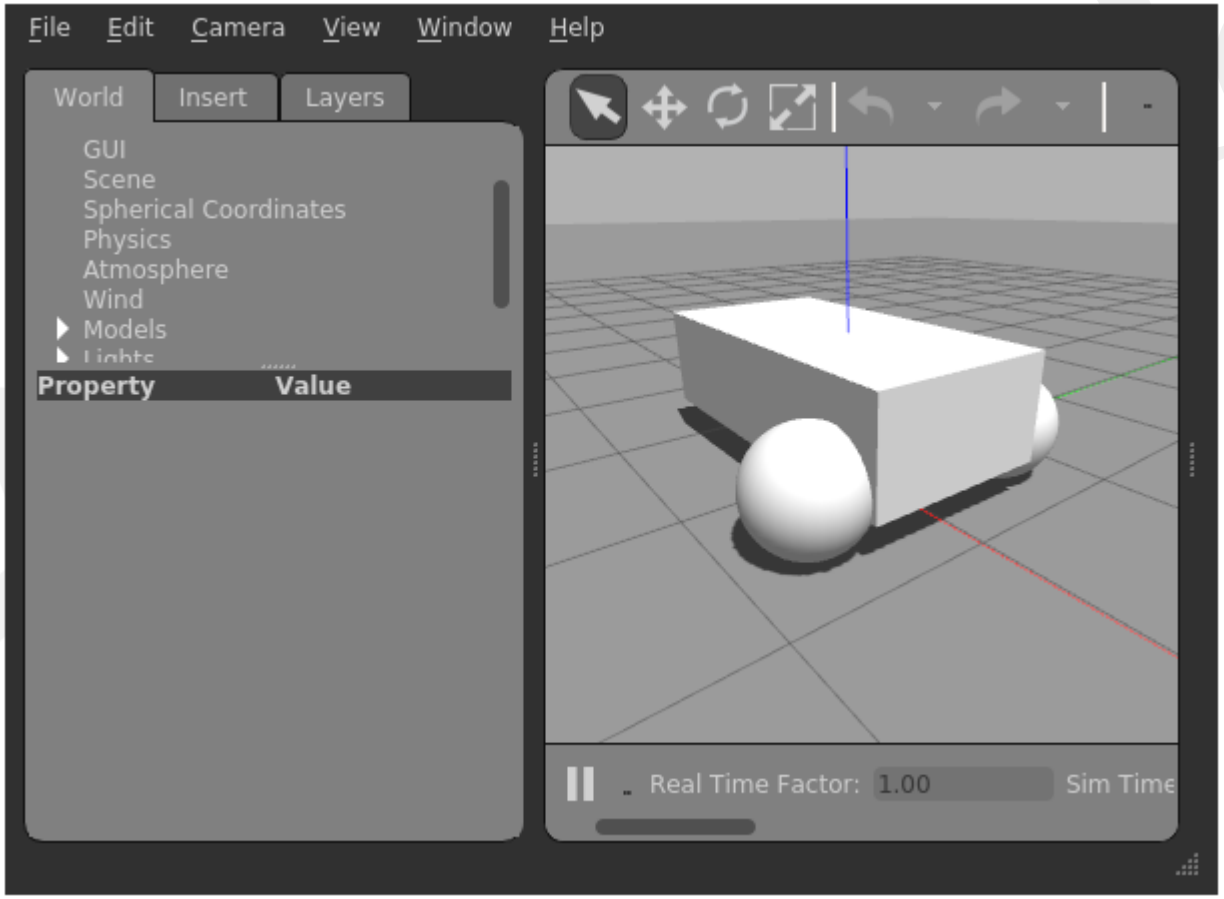


Gazebo 구성

실행

gazebo --verbose /opt/ros/foxy/share/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world

ros2 topic pub /demo/cmd_demo geometry_msgs/Twist '{linear: {x: 1.0}}' -1



Gazebo 구성

설치

```
$ sudo apt-get update
$ sudo apt-get install gazebo11
$ sudo apt-get install libgazebo11-dev
$ gazebo --version
```

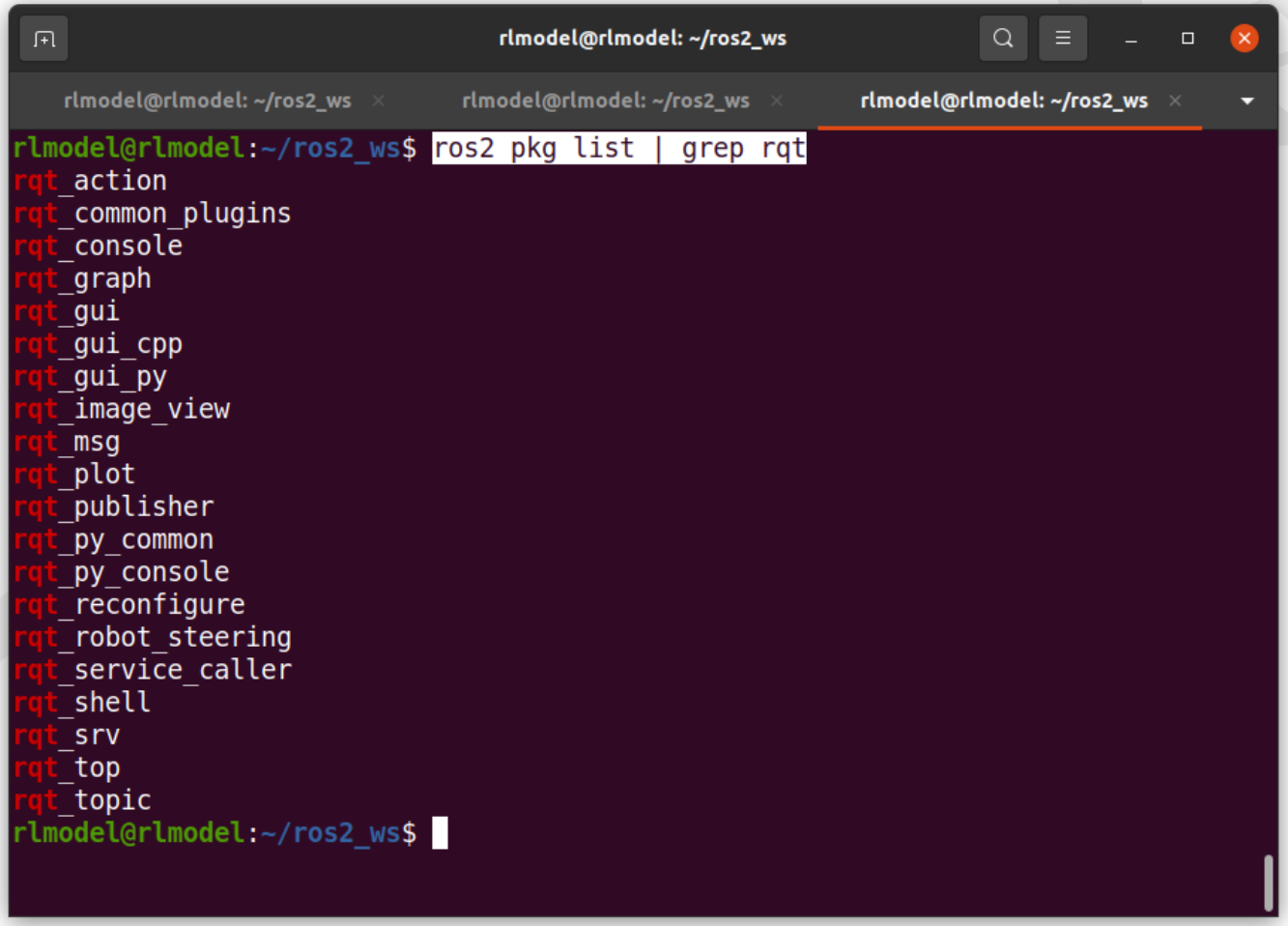
—----------

```
$ sudo apt install ros-foxy-gazebo-ros-pkgs
$ sudo apt install ros-foxy-ros-core ros-foxy-geometry2
```

Gazebo play

rqt_robot_control

check pkg: ros2 pkg list | grep rqt



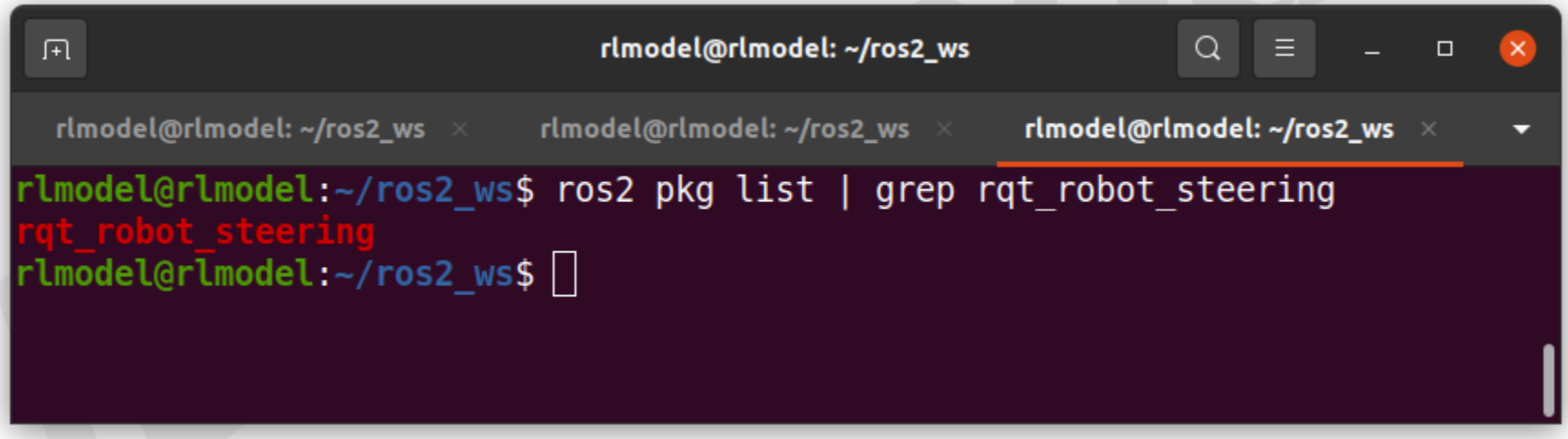
Gazebo play

rqt_robot_control

check pkg: rqt_robot_steering

install: sudo apt-get install ros-kinetic-rqt-robot-steering (x)

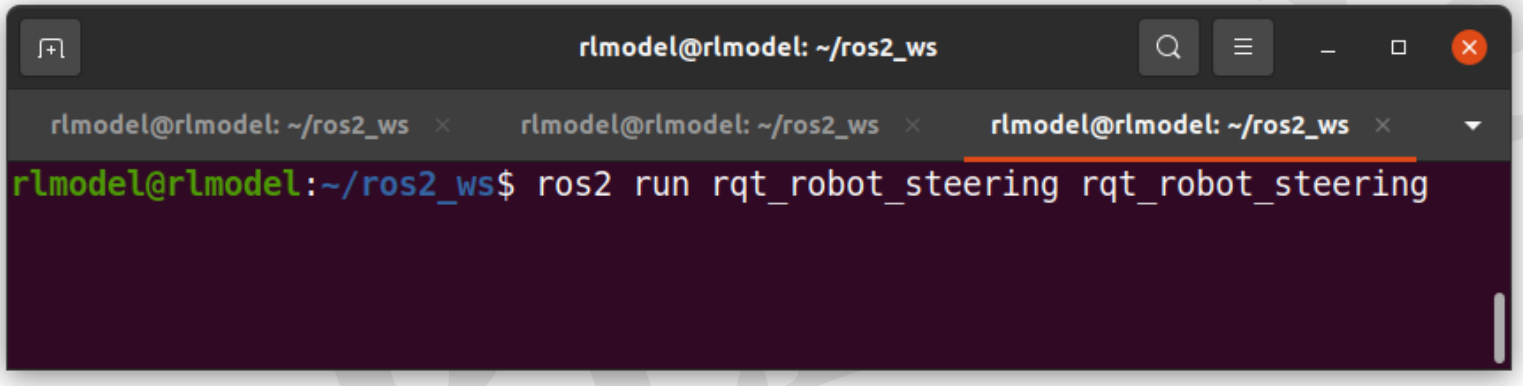
install: sudo apt-get install ros-foxy-rqt-robot-steering (o)

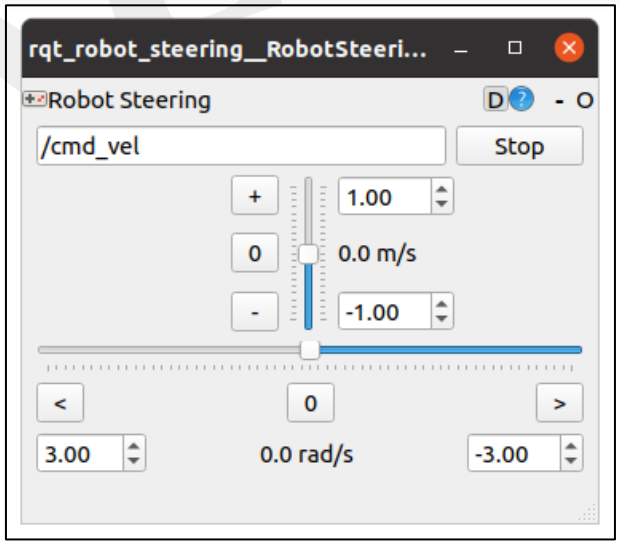


Gazebo play

rqt_robot_control

run: rqt_robot_steering

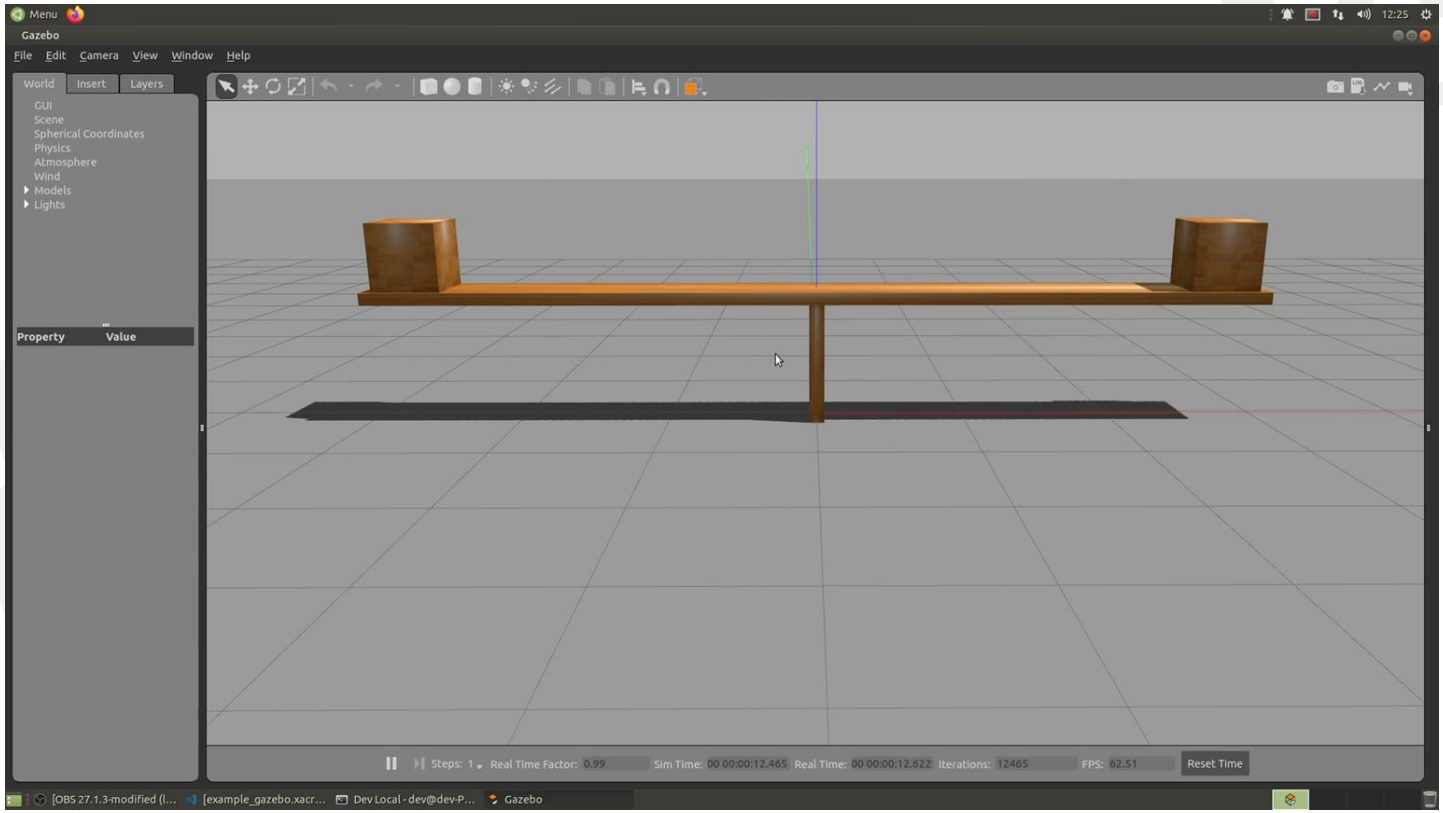




Gazebo 예제

Basic 예제

$ gazebo /usr/share/gazebo-11/worlds/seesaw.world



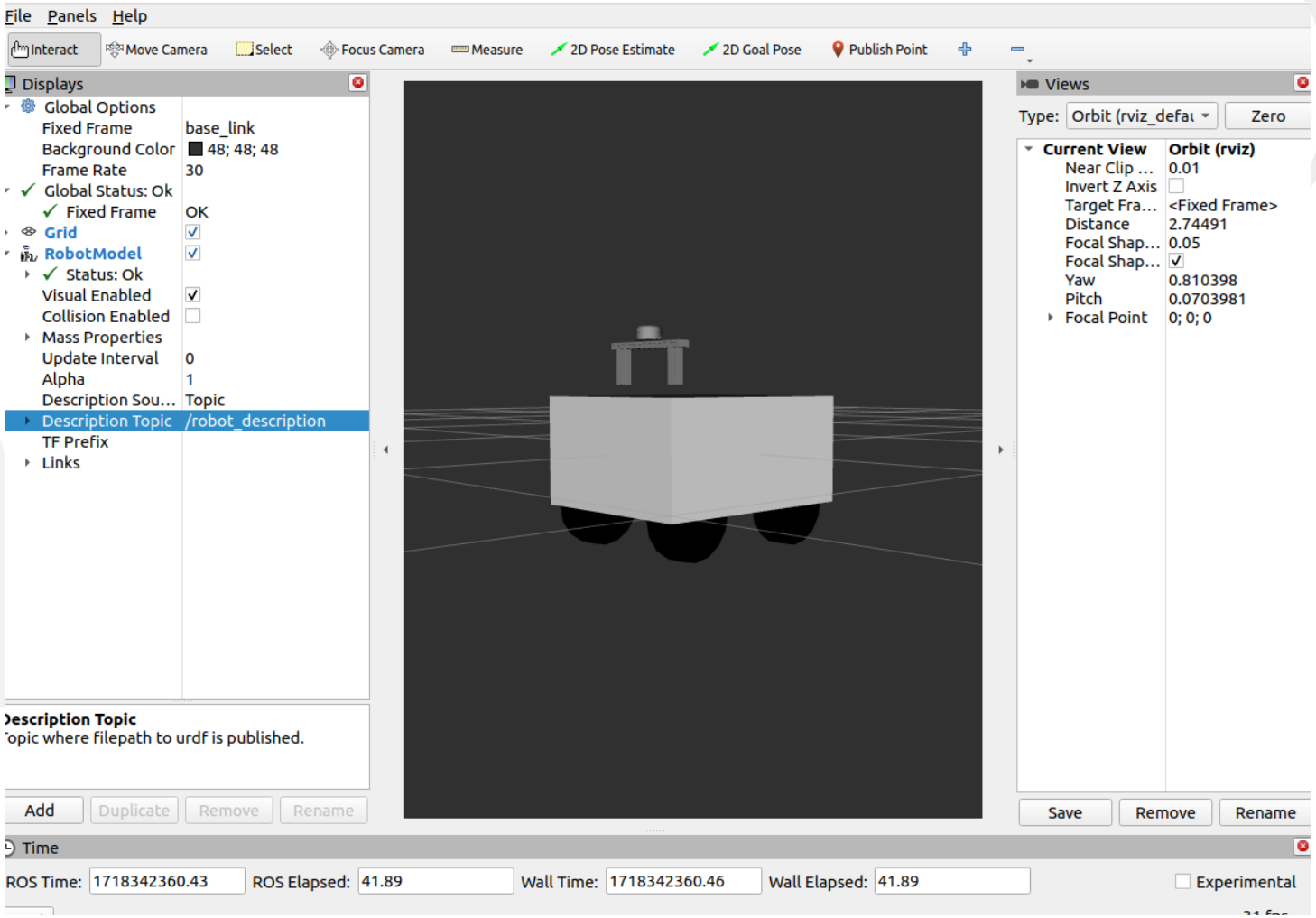
Gazebo 예제

URDF 예제

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="urdf_tutorial">

    <!-- COLOR -->
    <material name="white">
        <color rgba="1 1 1 1" />
    </material>

    <material name="blue">
        <color rgba="0 0 1 1"/>
    </material>

    <material name="black">
        <color rgba="0 0 0 1"/>
    </material>



    <link name="base_link">
        <visual>
            <origin xyz="0 0 0"/>
            <geometry>
                <box size="1.04 0.85 0.45"/>
            </geometry>
            <material name="white"/>
        </visual>
        <collision>
            <origin xyz="0 0 0"/>
            <geometry>
                <box size="1.04 0.85 0.45"/>
            </geometry>
        </collision>
    </link>

    <!-- LEFT WHEEL LINK -->

    <link name="left_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.06" length="0.06"/>
            </geometry>
            <material name="black"/>
        </visual>
        <collision>
            <geometry>
                <cylinder radius="0.06" length="0.06"/>
            </geometry>
        </collision>
```

Gazebo 예제

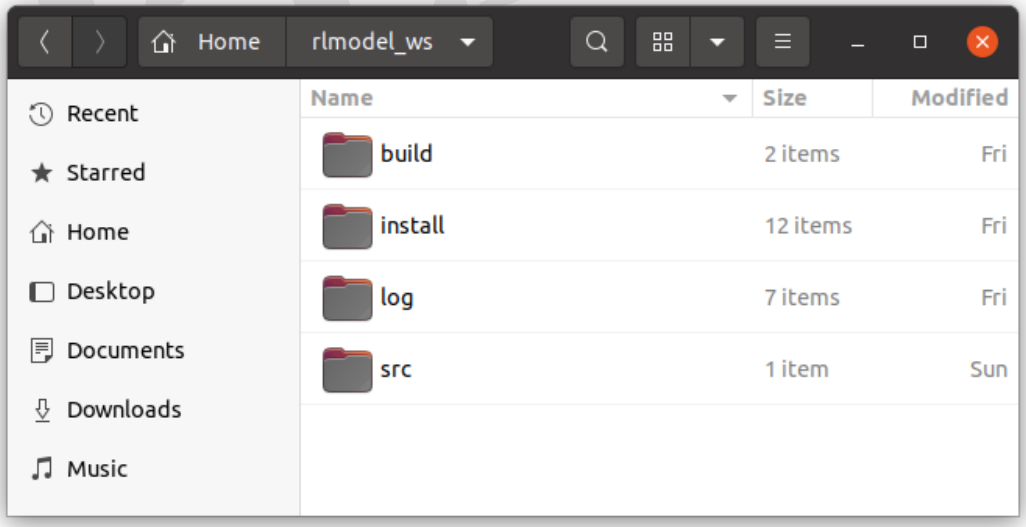
URDF 예제



ROS2 실습

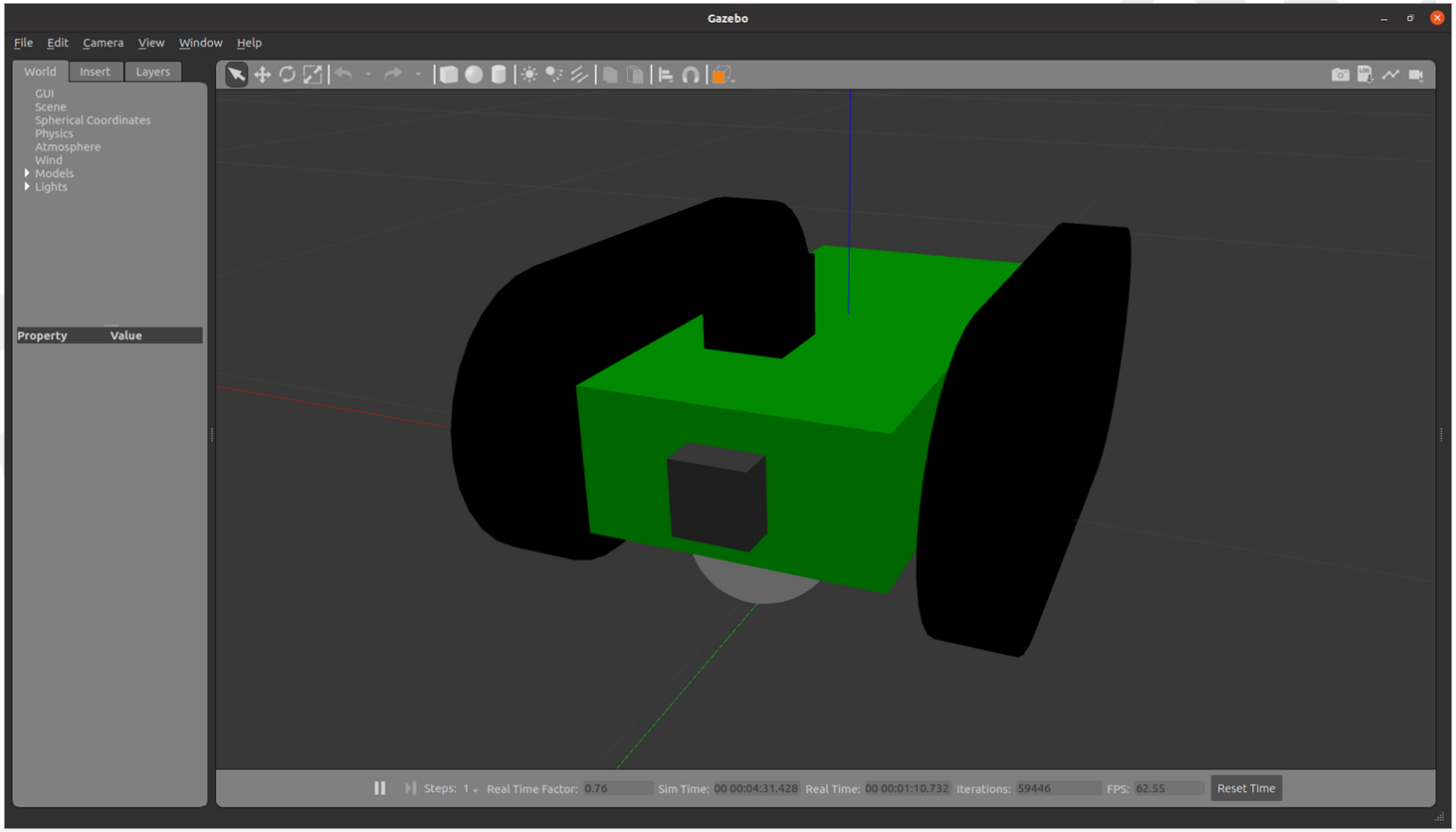
workspace 생성

- $ mkdir -p ~/ros2_ws/src
- $ sudo apt install python3-rosdep
- $ sudo rosdep init
- $ rosdep update
- $ rosdep install -i --from-path src --rosdistro foxy -y
- $ colcon build
- $. install/local_setup.bash



SKI_bot

gazebo maze world example

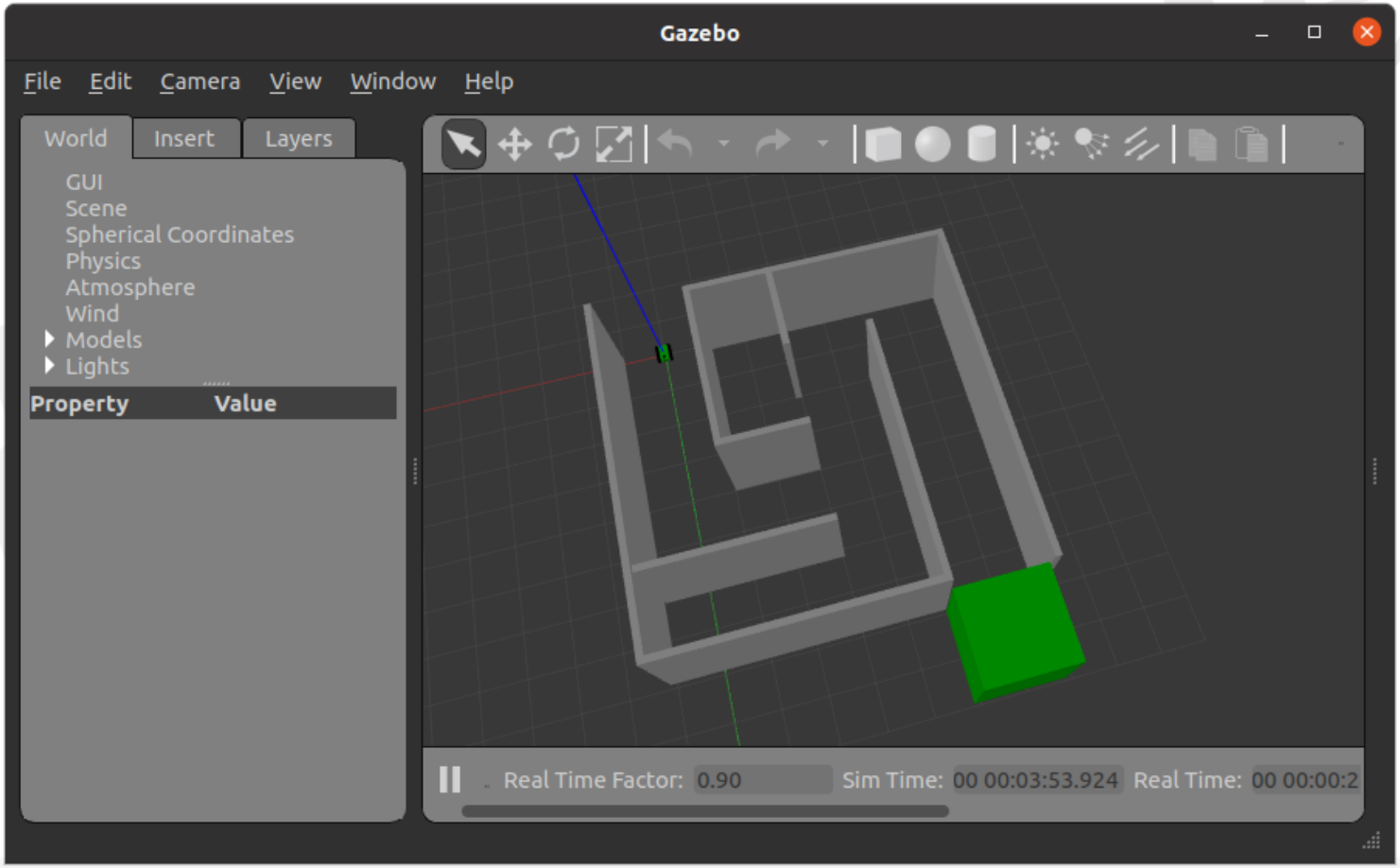


Clone

ski_bot

github: https://github.com/RLmodel/gcamp_gazebo

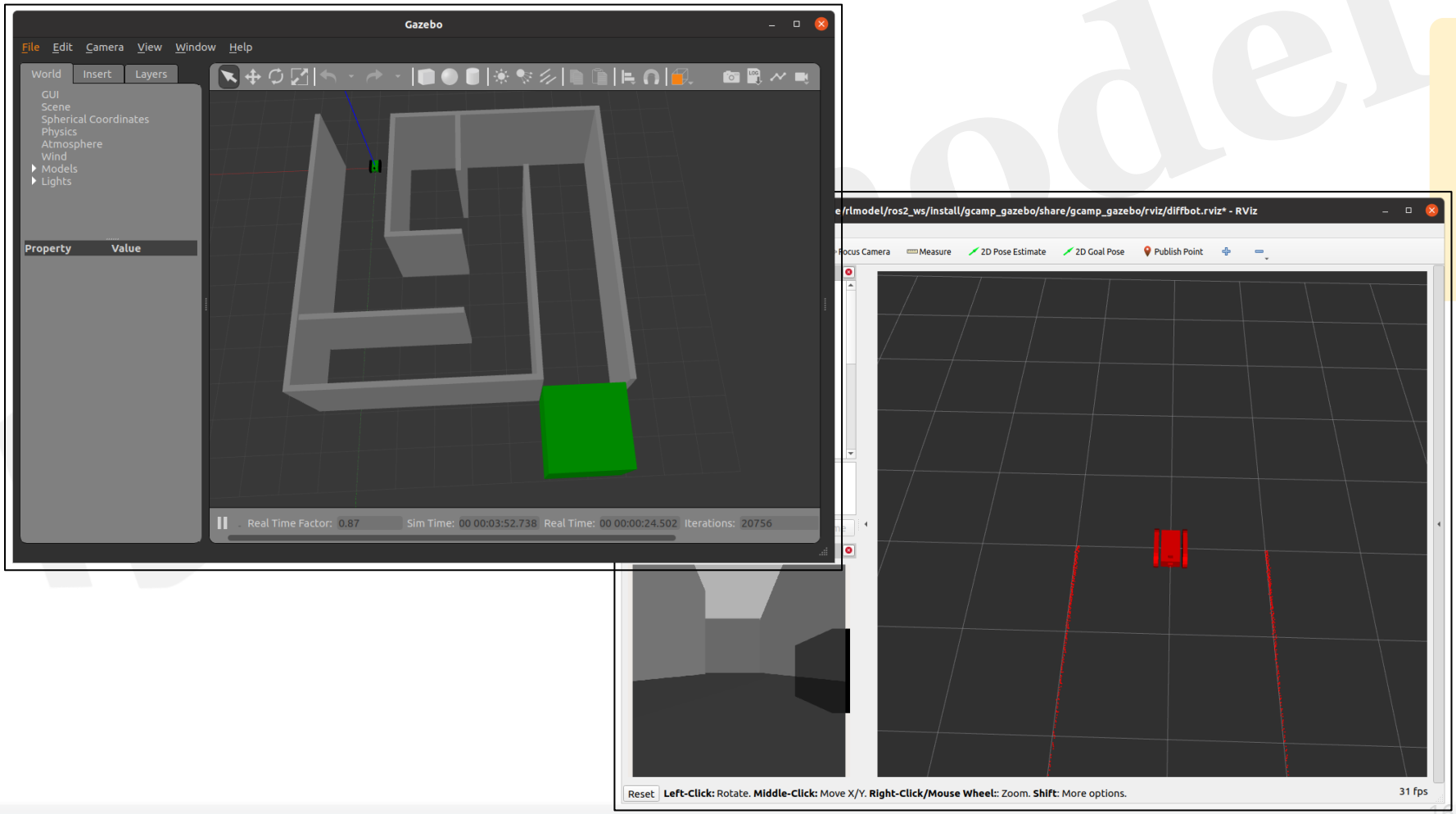
World: maze_world



Gazebo example

UGV robot - skidbot run

$ ros2 launch gcamp_gazebo maze_world.launch.py

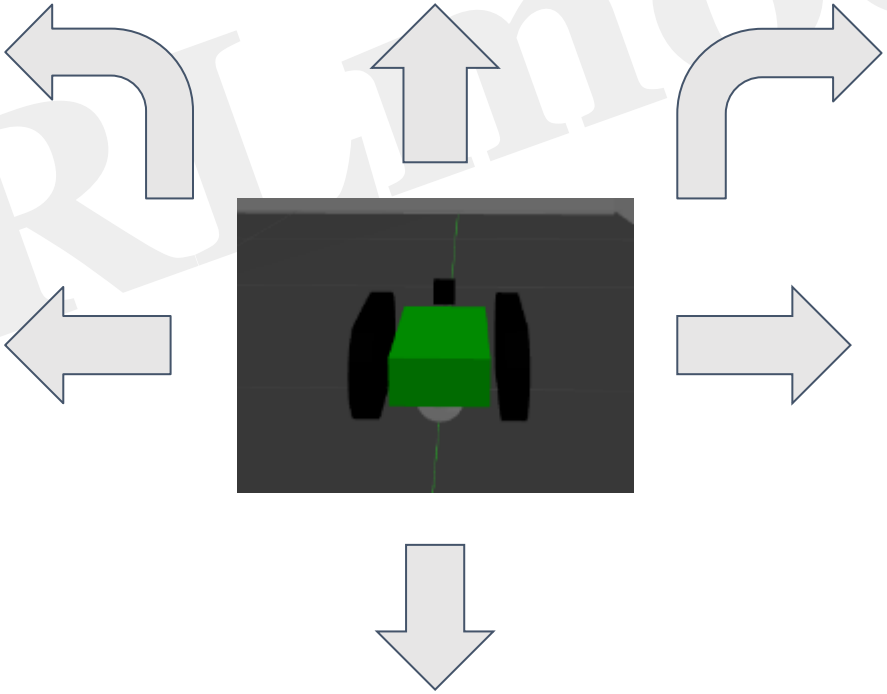


Gazebo example

UGV robot - skidbot control

exercise 1: move robot, check topic, view lidar data

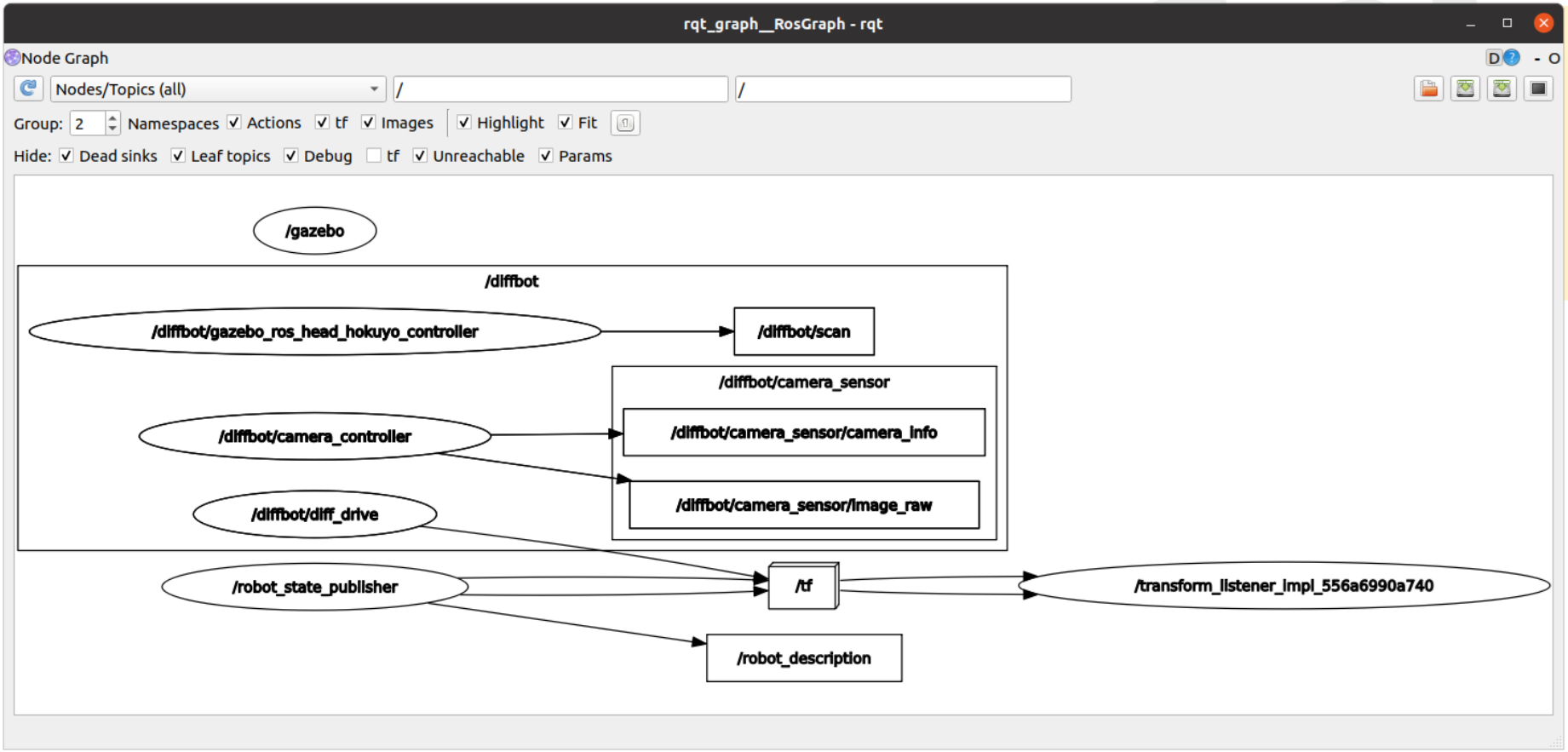
exercise 2: escape maze, replay using bag file



Gazebo example

UGV robot - skidbot control

view rqt_graph



Gazebo example

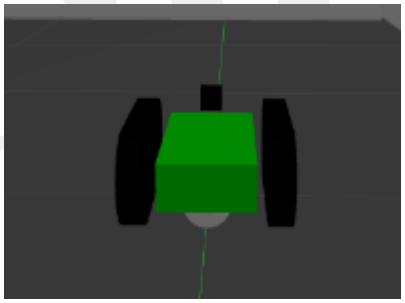
UGV robot - skidbot control

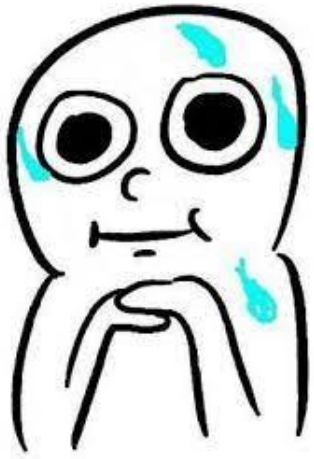
run teleop node

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard

> rqt

> rqt_robot_steering

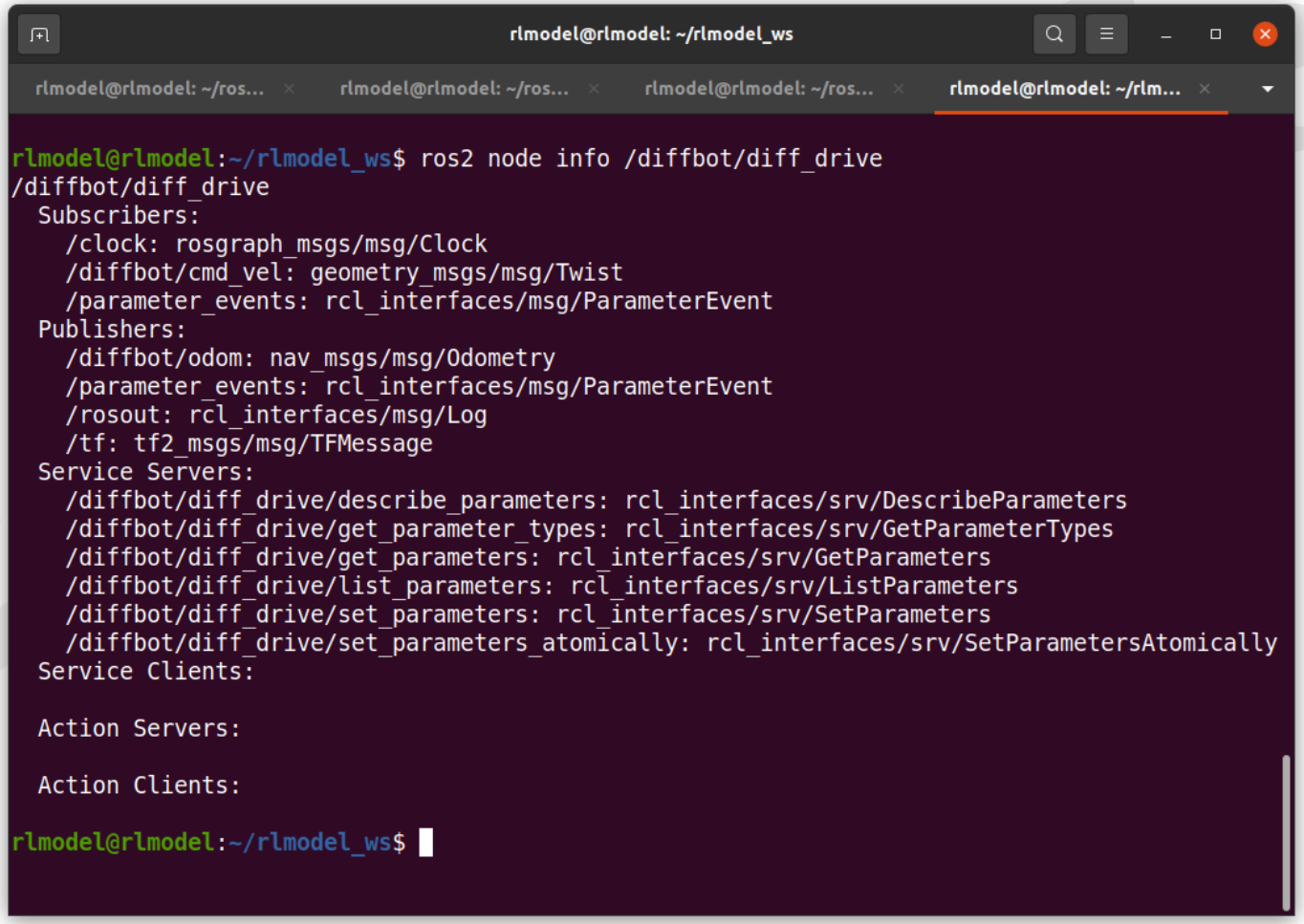




Gazebo example

UGV robot - skidbot control

check node info



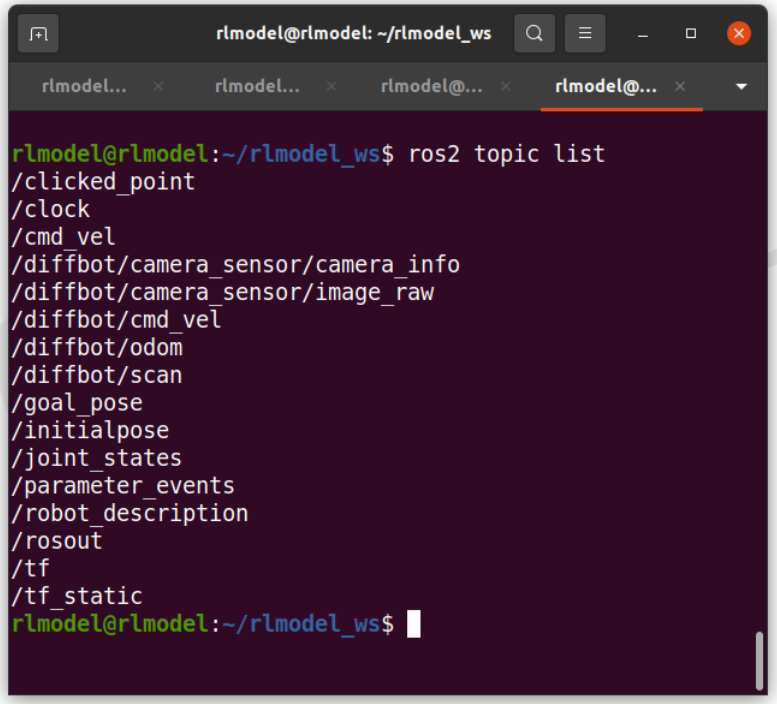
Gazebo example

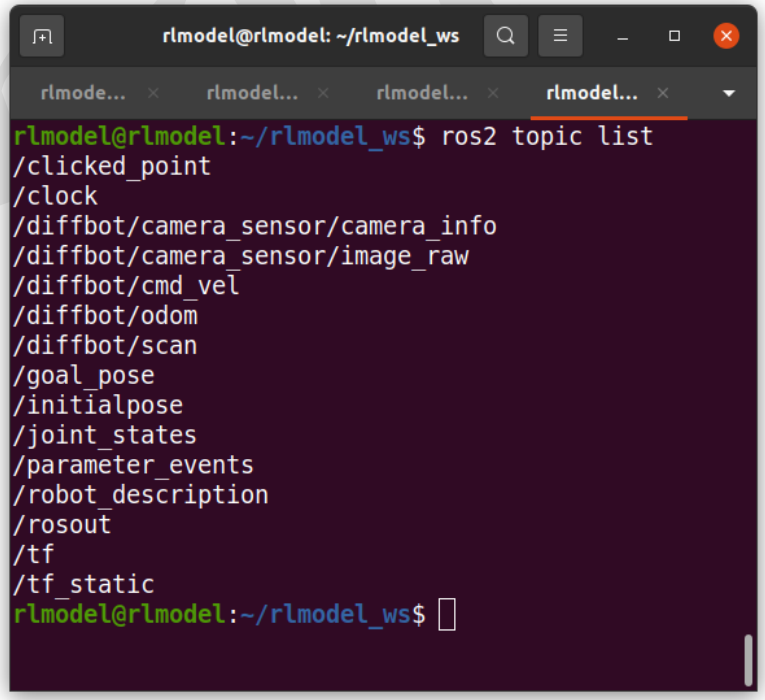
UGV robot - skidbot control

run teleop

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard

$ ros2 topic list

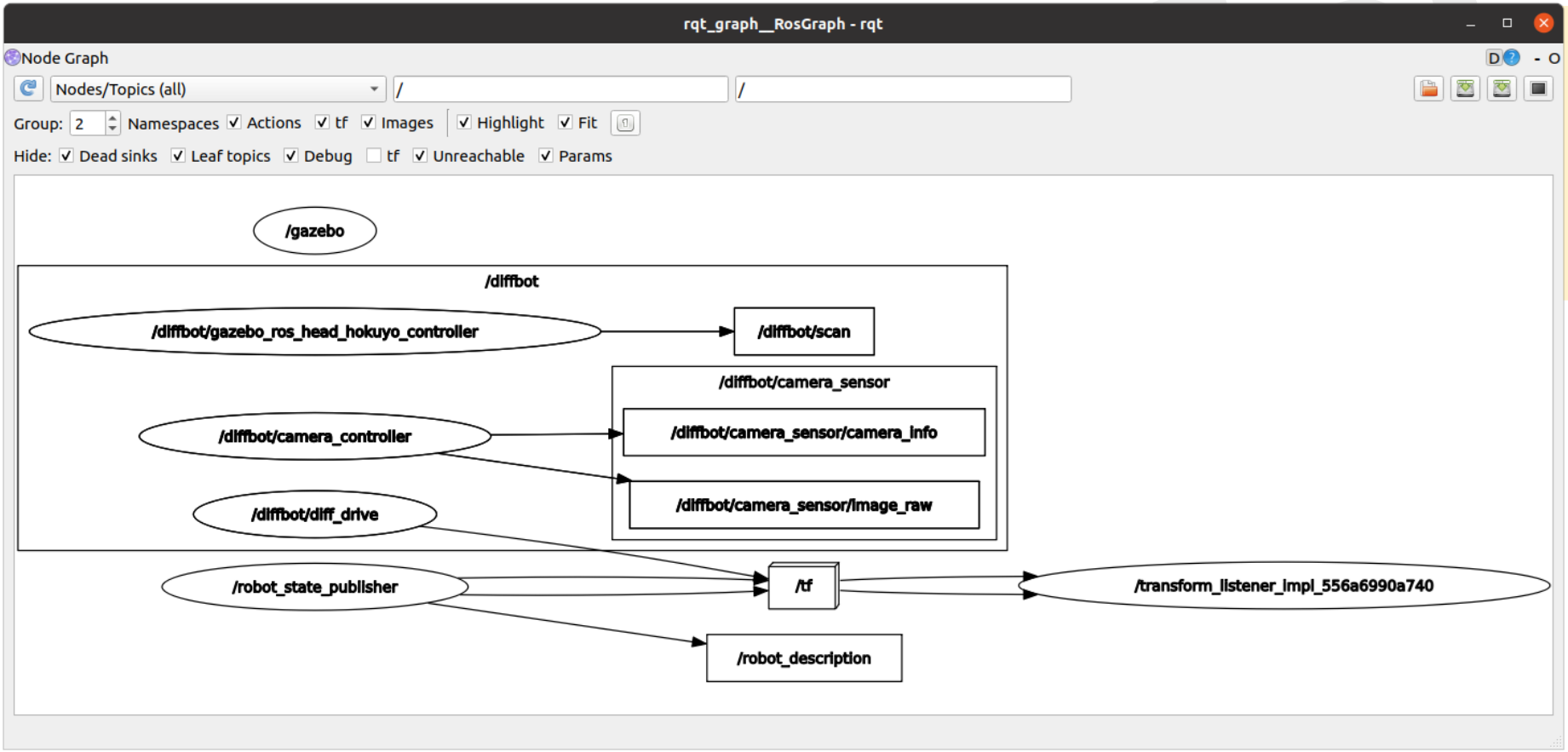




Gazebo example

UGV robot - skidbot control

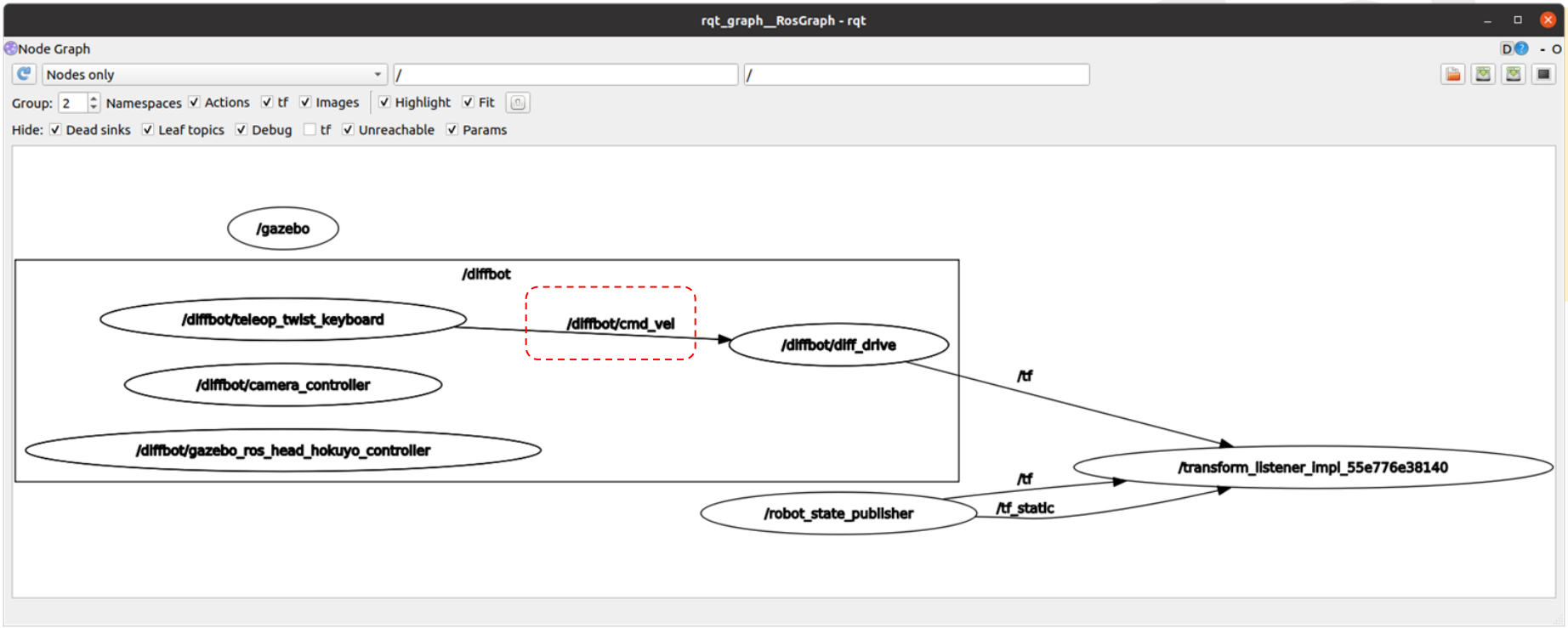
run -> /diffbot/cmd_vel



Gazebo example

UGV robot - skidbot control

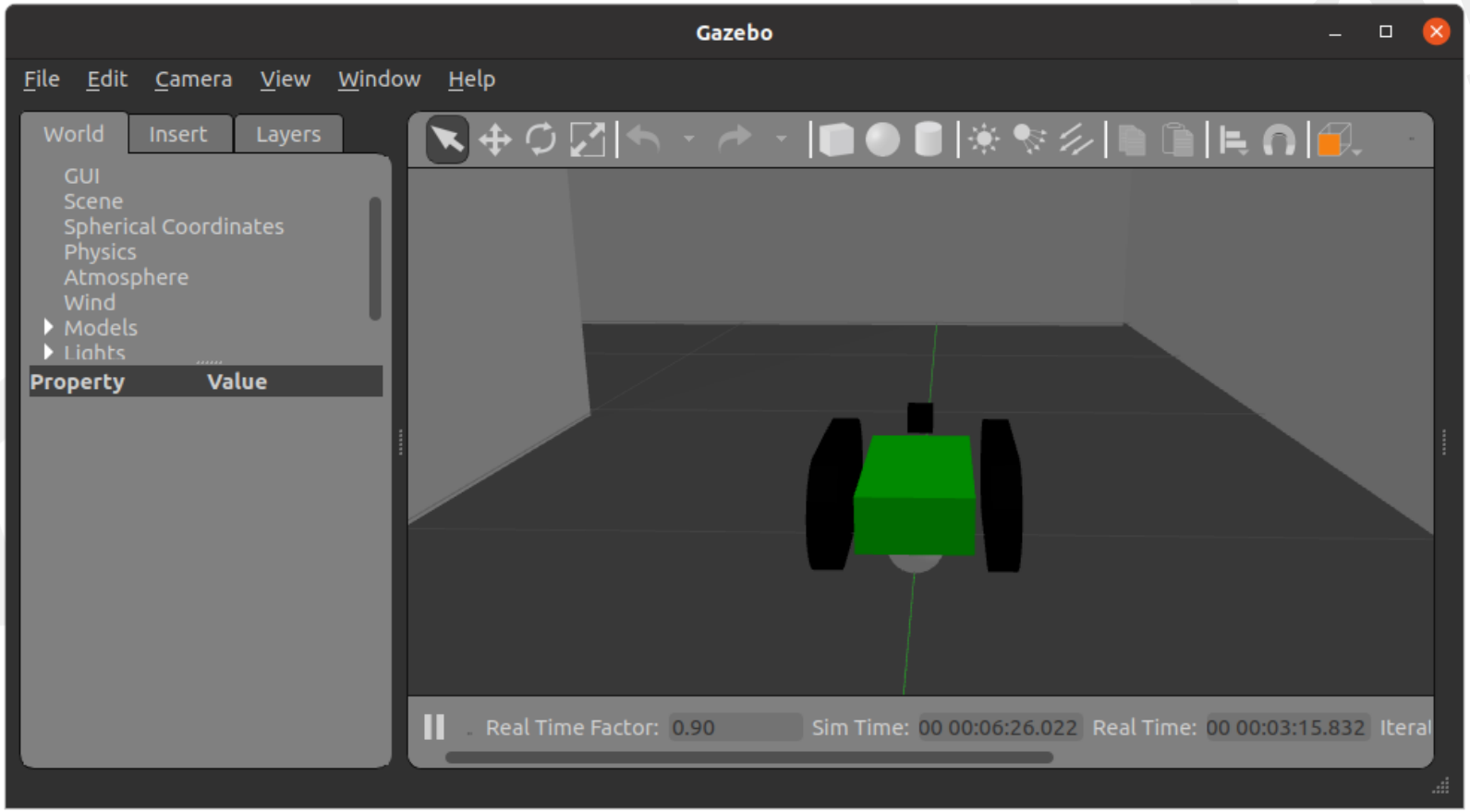
run -> /diffbot/cmd_vel



Gazebo example

UGV robot - skidbot run

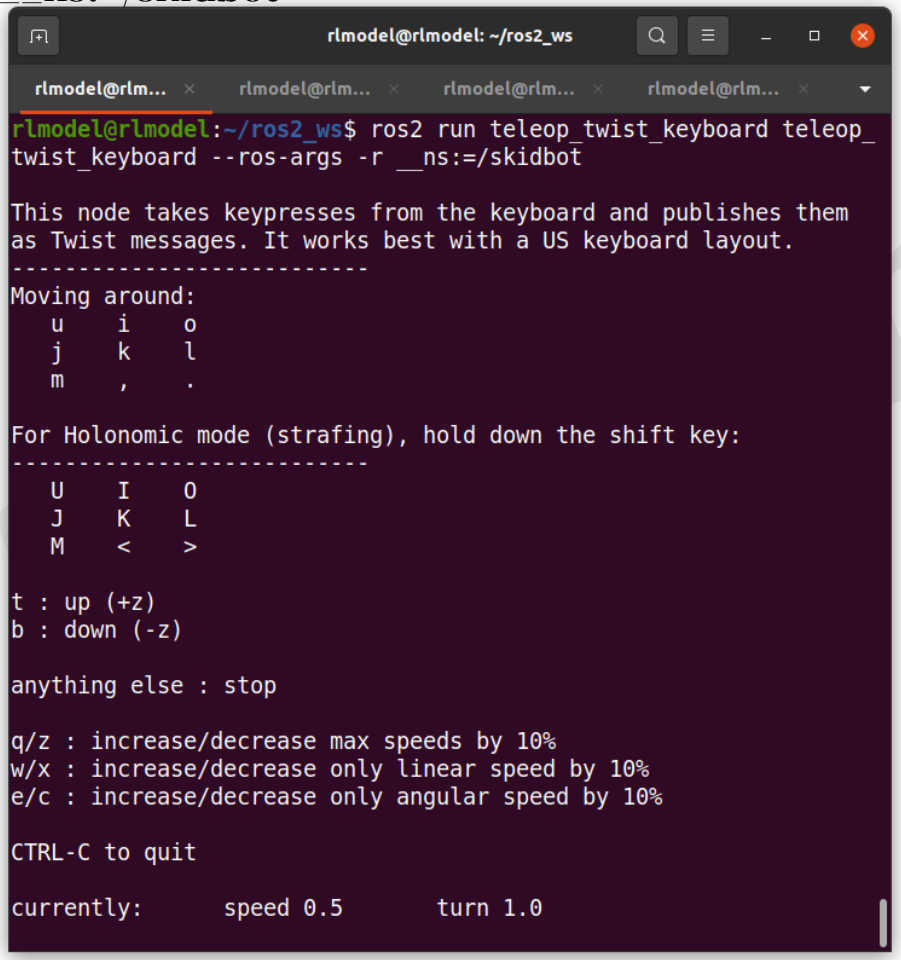
```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r __ns:=/diffbot
```

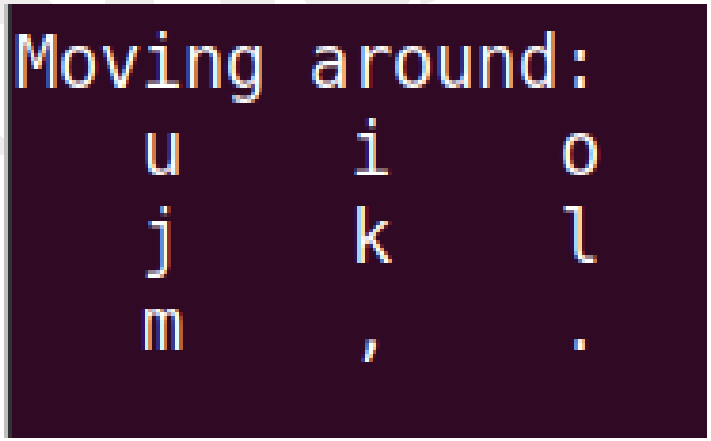


Gazebo example

UGV robot - skid bot control

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r __ns:=/skidbot

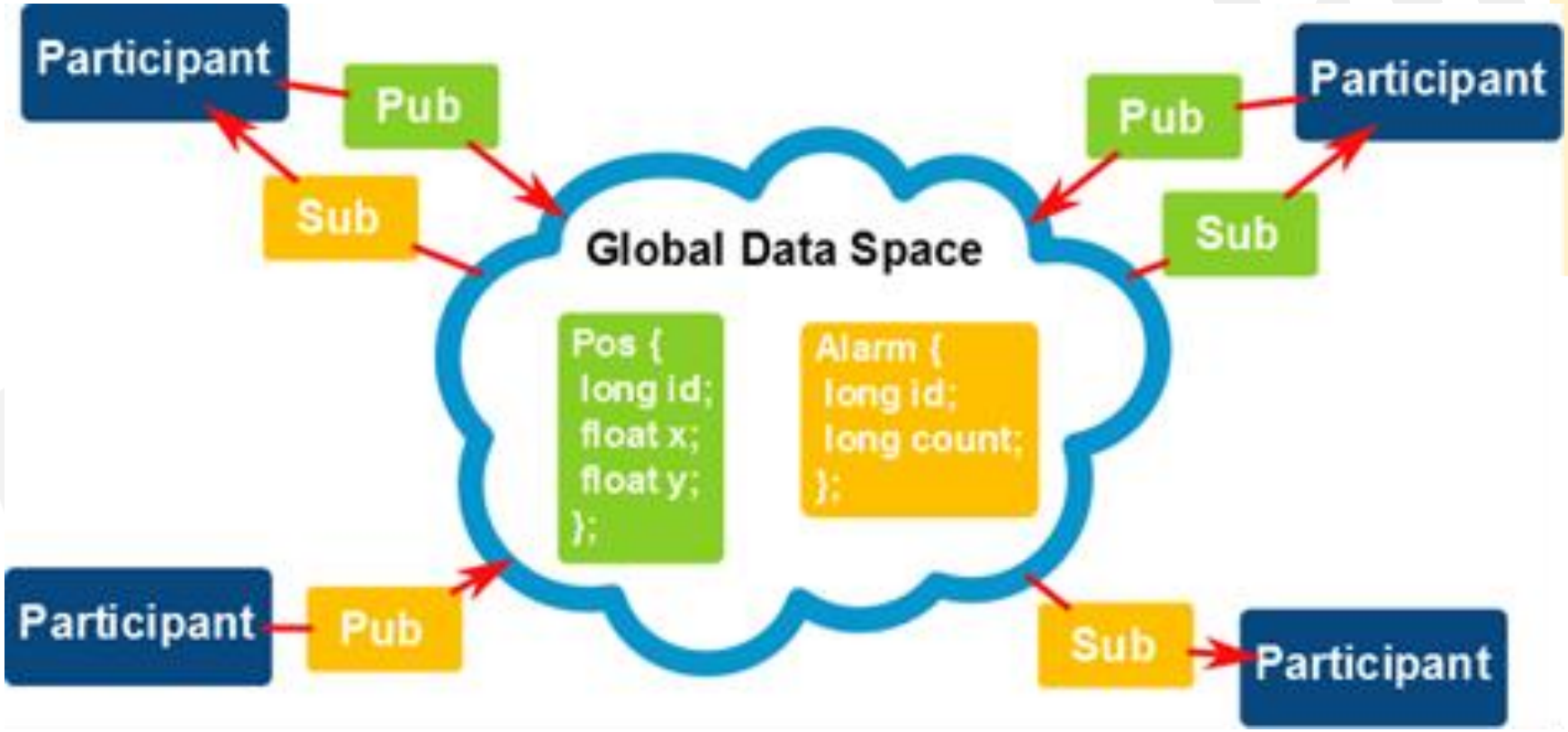




DDS config

Domain ID

```
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r __ns:=/skidbot
```

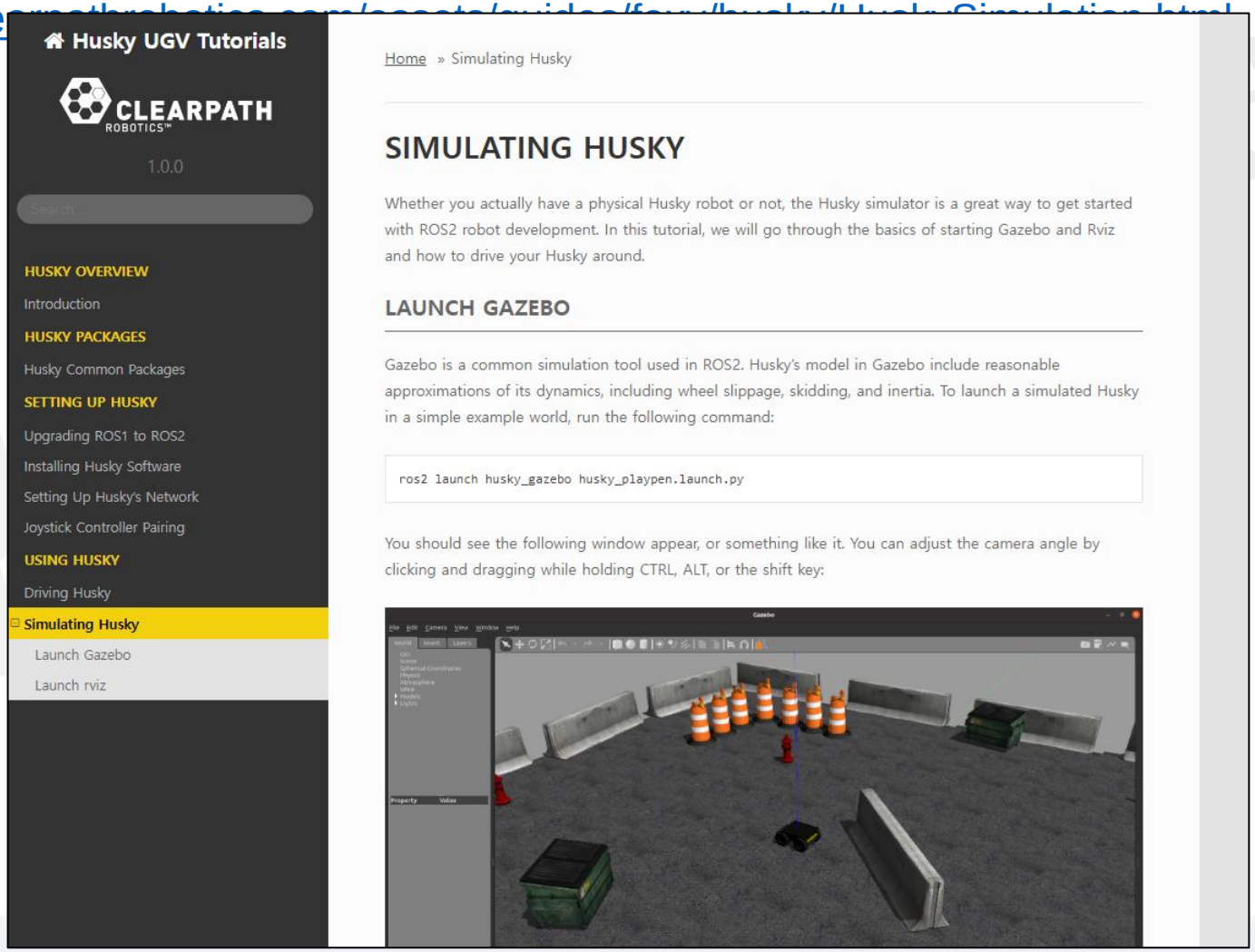


Gazebo

Clearpath Husky

Link:

https://clearpathrobotics.com/assets/guides/foxy/husky/HuskySimulation.html

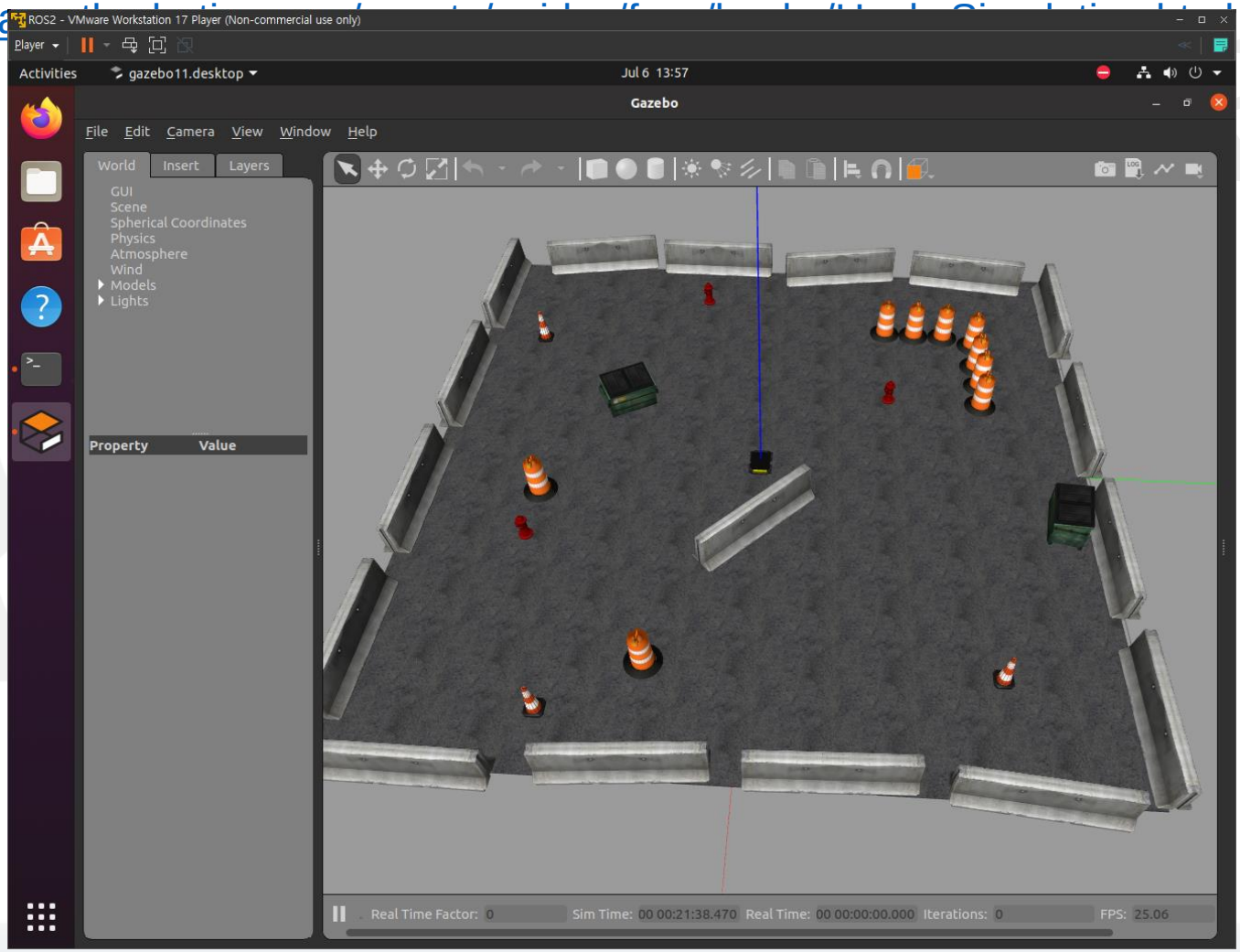


Gazebo

Clearpath Husky

Link:

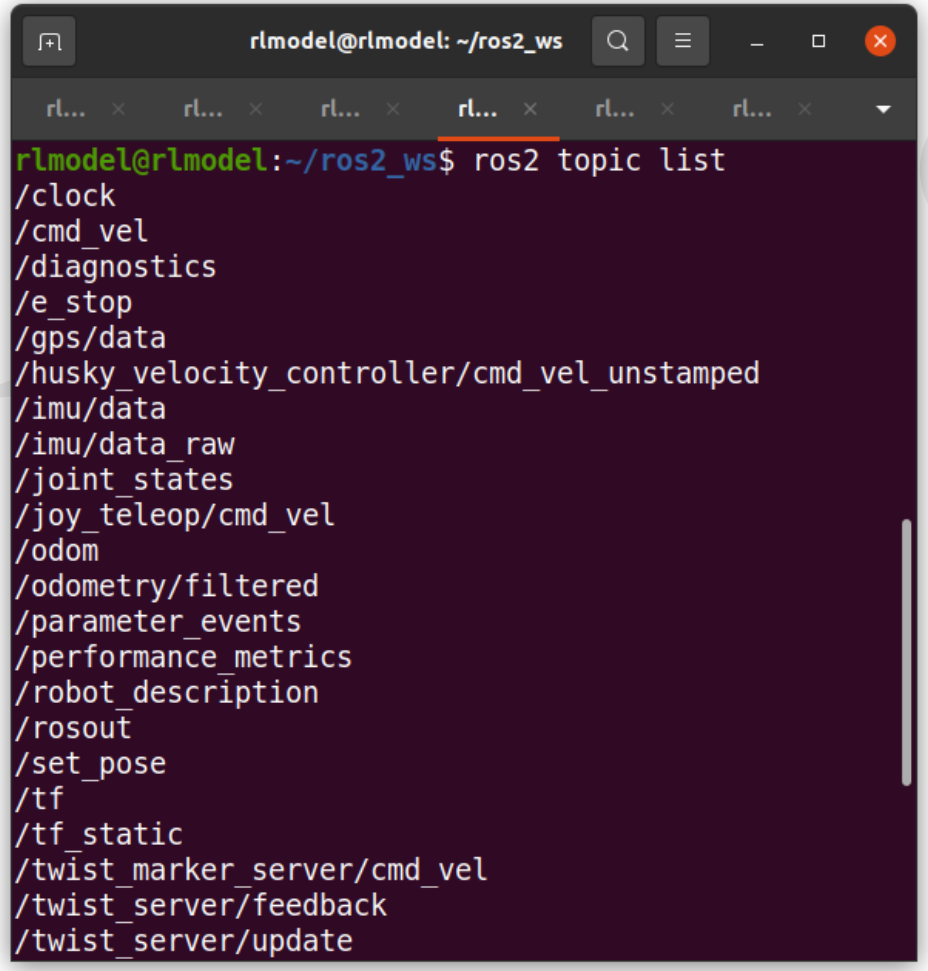
https://clea



Gazebo

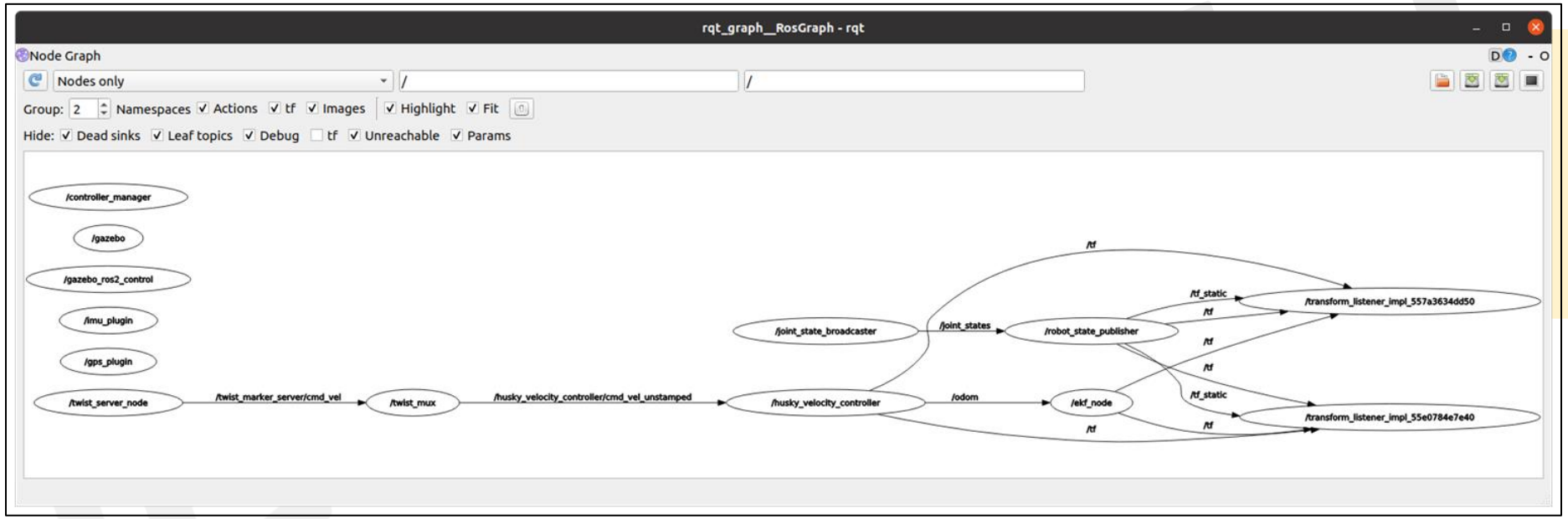
Clearpath Husky

check topics: IMU, GPS, etc



Clearpath Husky

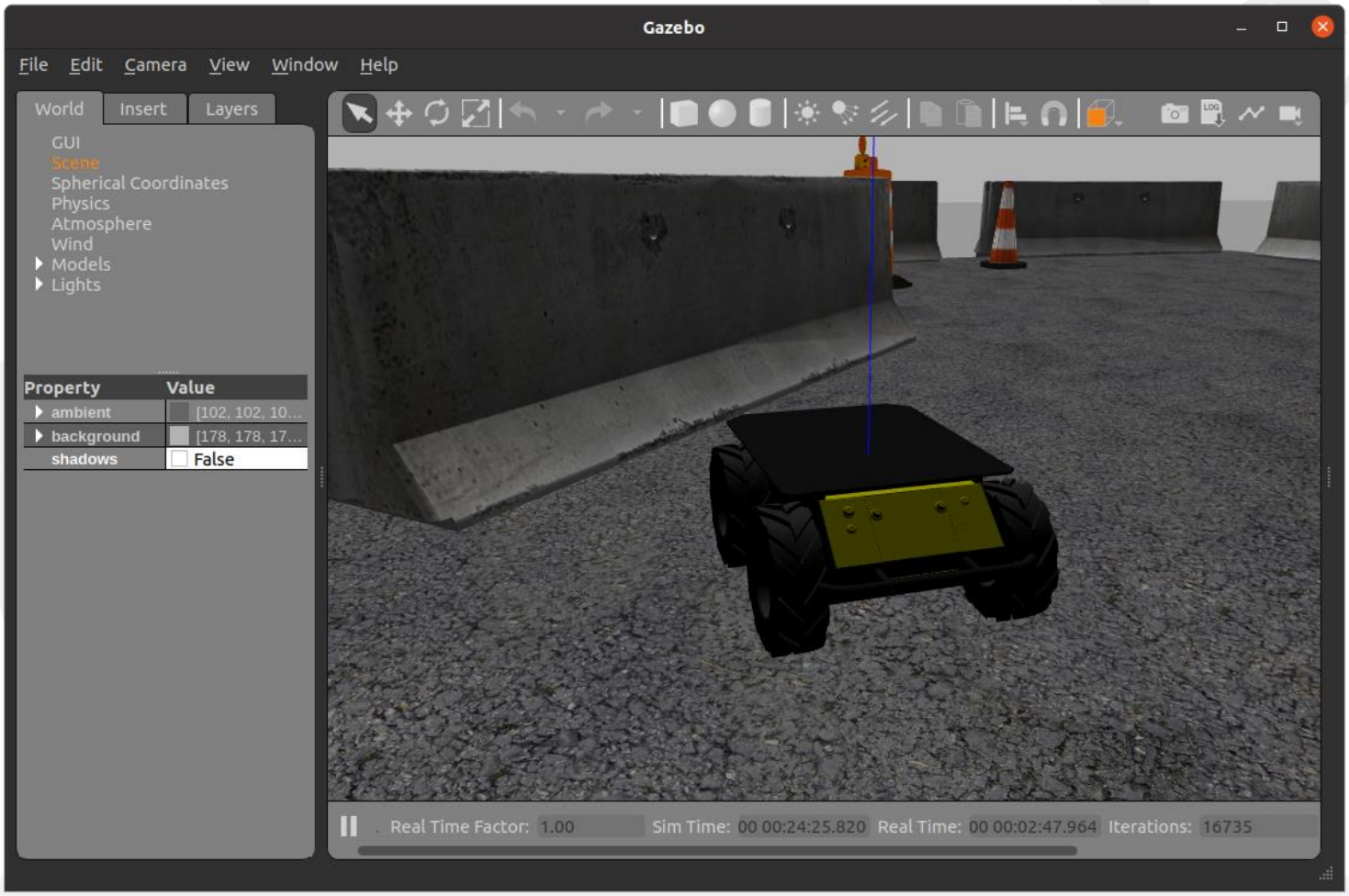
rqt_graph: IMU, GPS, ekf, .. etc



Gazebo

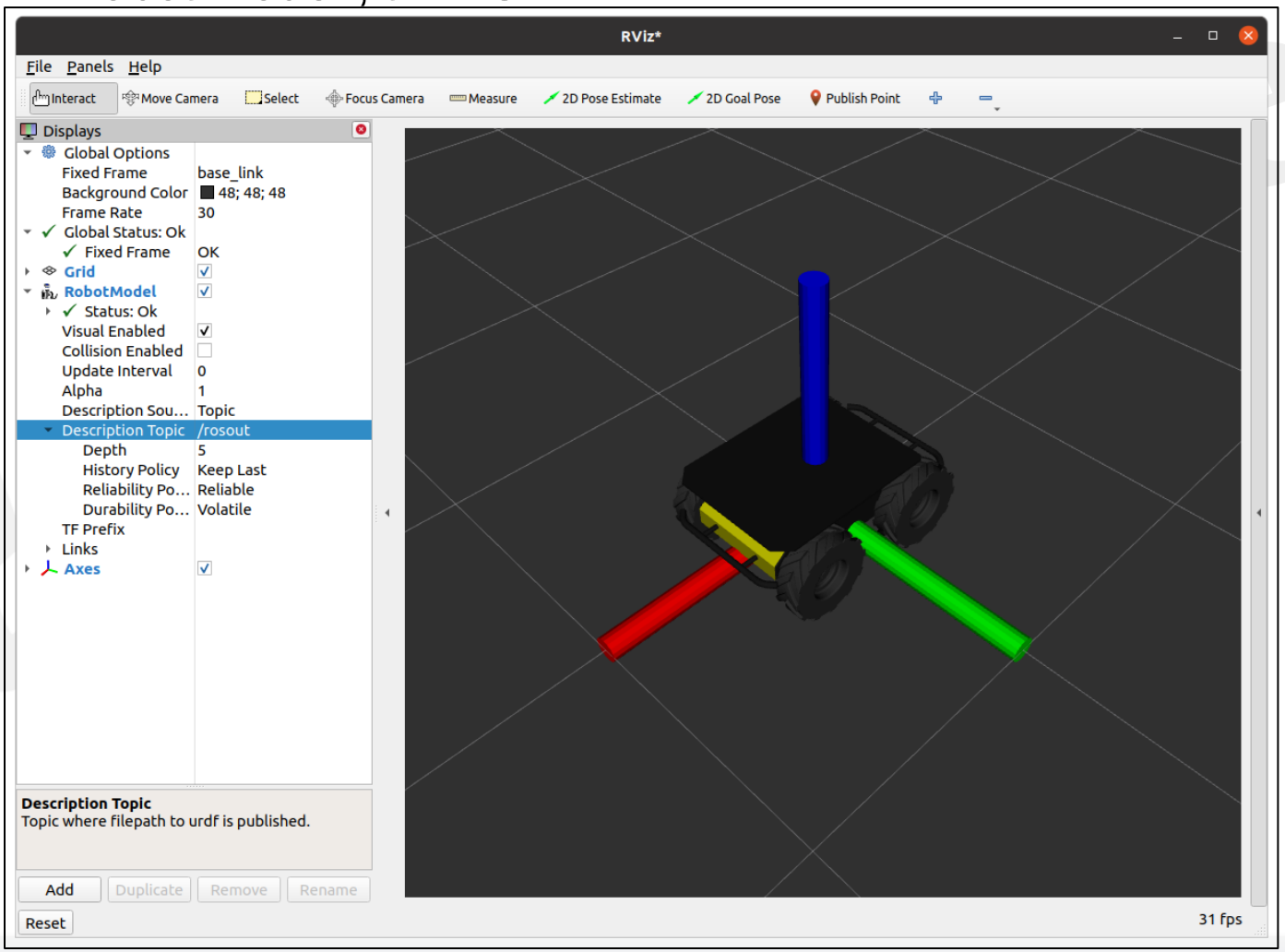
Clearpath Husky

cmd: ros2 topic pub --once /husky_velocity_controller/cmd_vel_unstamped geometry_msgs/msg/Twist '{linear: {x: -0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'



Clearpath Husky

rviz: Robot model , tf Axis



Gazebo

Clearpath Husky

Link: https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

EXTERNAL DIMENSIONS
990 x 670 x 390 mm
(39 x 26.4 x 14.6 in)

INTERNAL DIMENSIONS
296 x 411 x 155 mm
(11.7 x 16.2 x 6.1 in)

WEIGHT
50 kg
(110 lbs)

MAX PAYLOAD
75 kg
(165 lbs)

Gazebo

Clearpath Husky

GPS Waypoint navigation

Link 1: https://www.youtube.com/watch?v=1Ys1_akUD-s

Gazebo

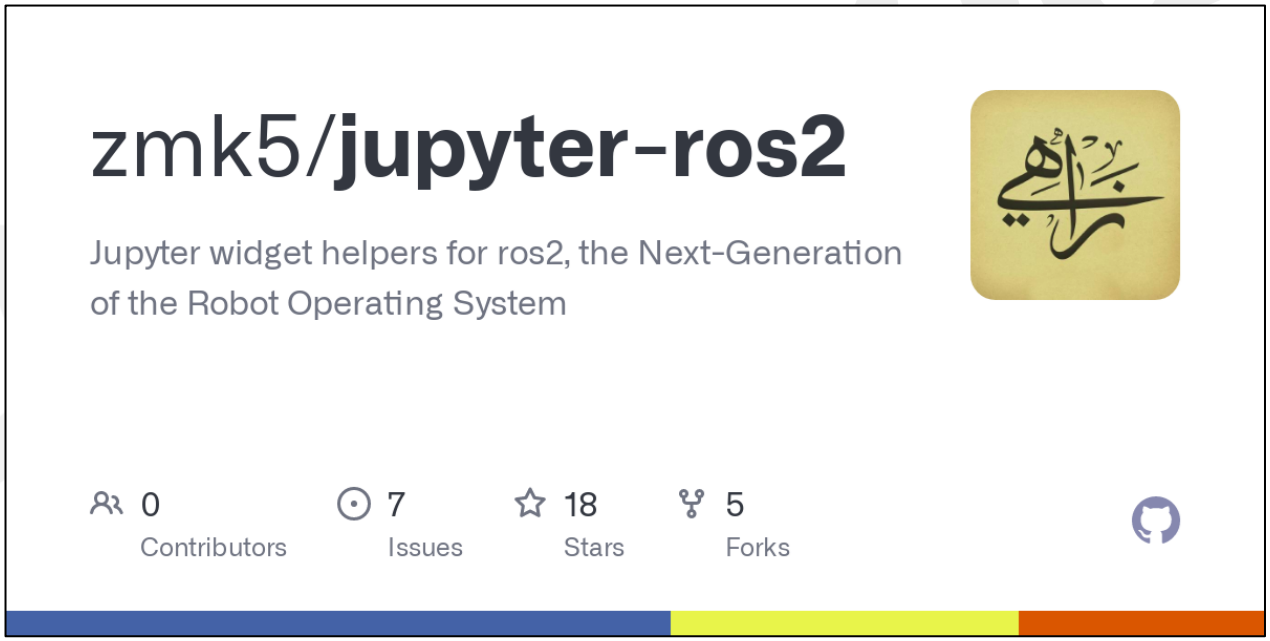
Clearpath Husky

GPS Waypoint navigation

Link 2: https://www.youtube.com/watch?v=hP6NiGzOLmI

프로그램 실습

ROS2 Jupyter notebook



ROS2 Python 실습

Jupyter notebook

설치:

>sudo apt install python3-pip

>pip3 install --upgrade pip

>pip3 install jupyter ipywidegets pyyaml bqplot

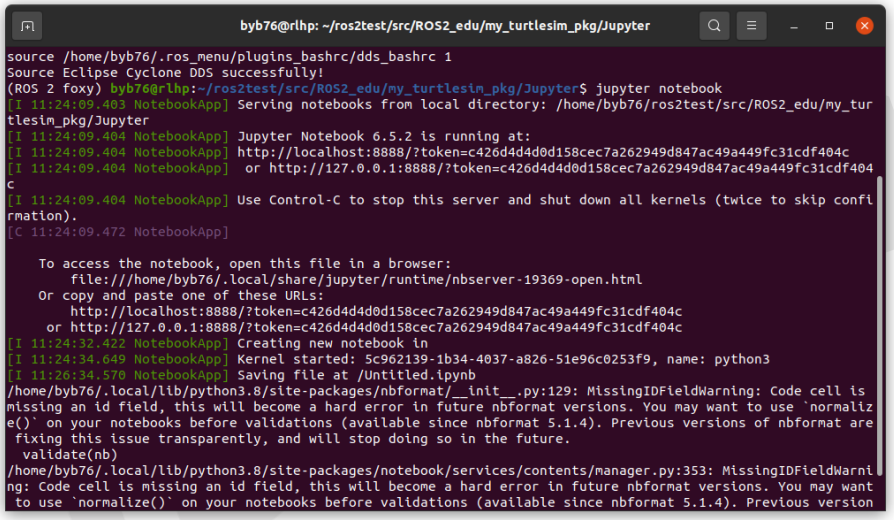
>sudo apt install jupyter-core

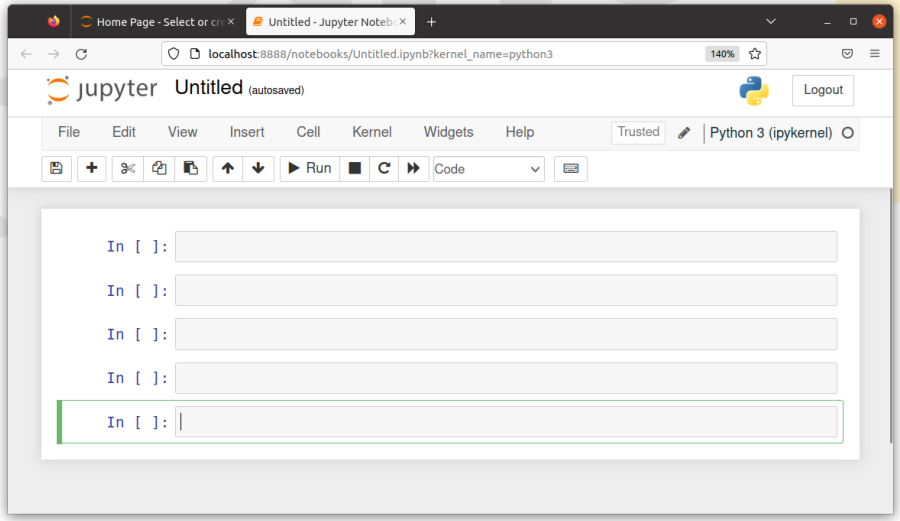
ROS2 Python 실습

Jupyter notebook 실행

>jupyter notebook

>mark down 언어지원





ROS2 Python 실습

mark down 테스트

>기본문법 예시

>See Appendix

```
# title
## *title2
### **title3
* item1
* item2
1. test1
2. test2
> comment
'''
code block
'''
```

title

*title2

**title3

- item1
- item2

1. test1
2. test2

> comment ''' code block '''

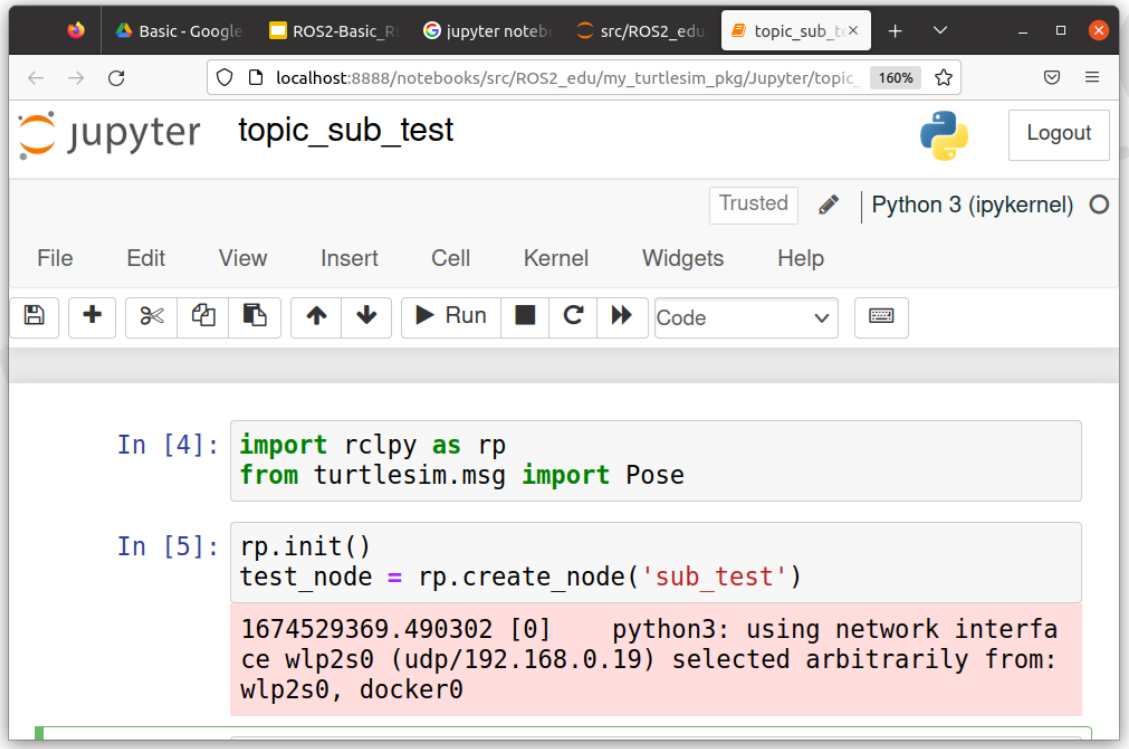
ROS2 Python 실습

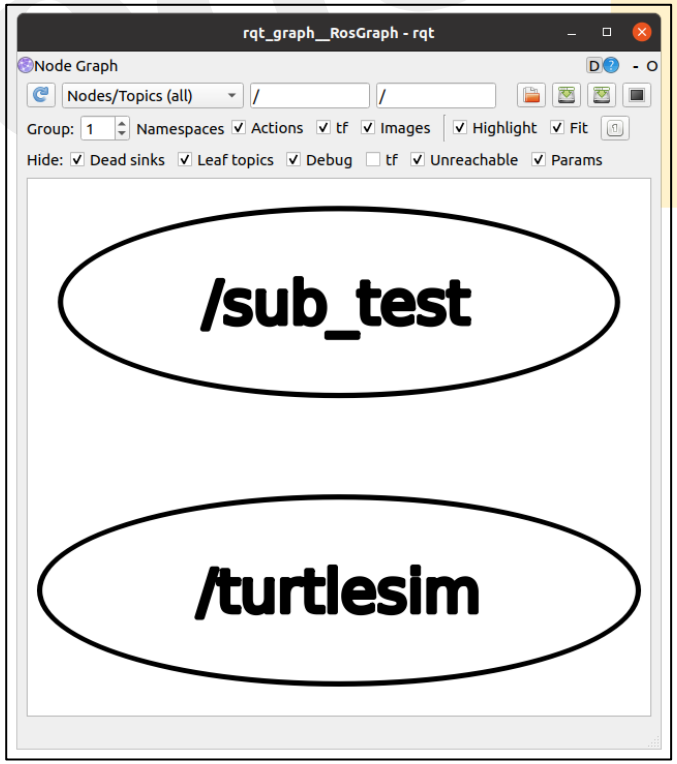
topic 수신

준비: $Source /opt/ros/foxy/setup.bash

순서: python code 작성 > 노드 생성

확인: $ros2 node list



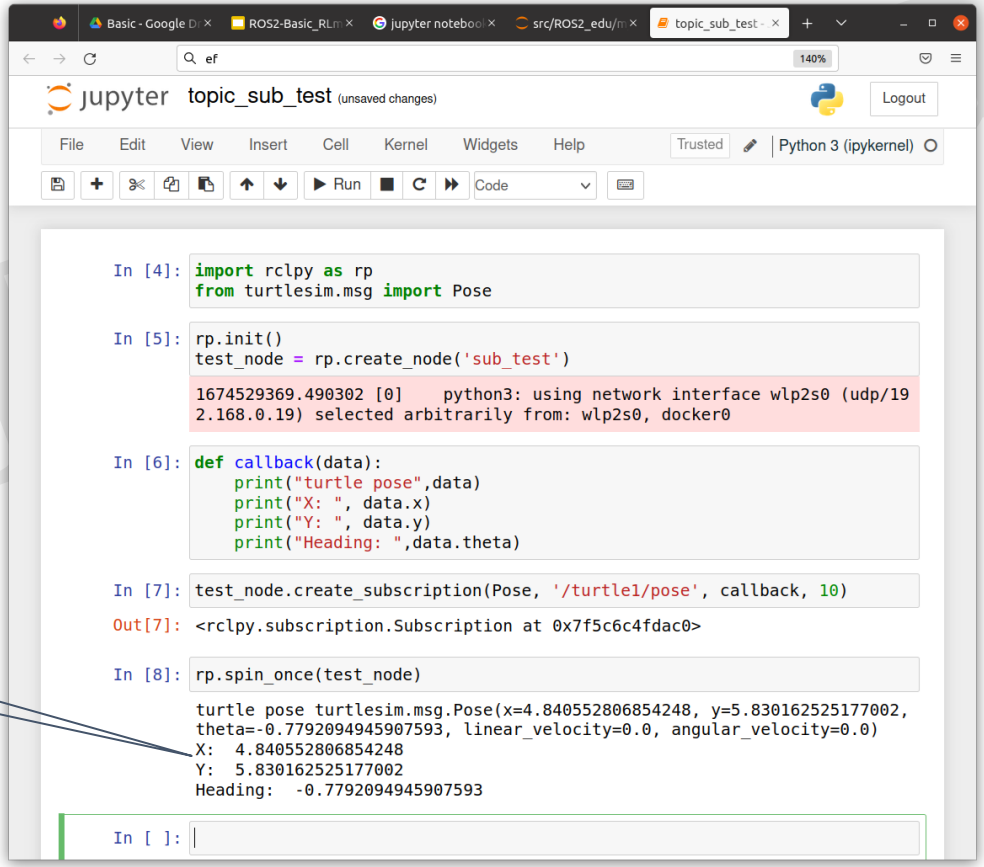


ROS2 Python 실습

topic 코드

>토픽 수신 (subscription)

rp.spin_once(test_node)



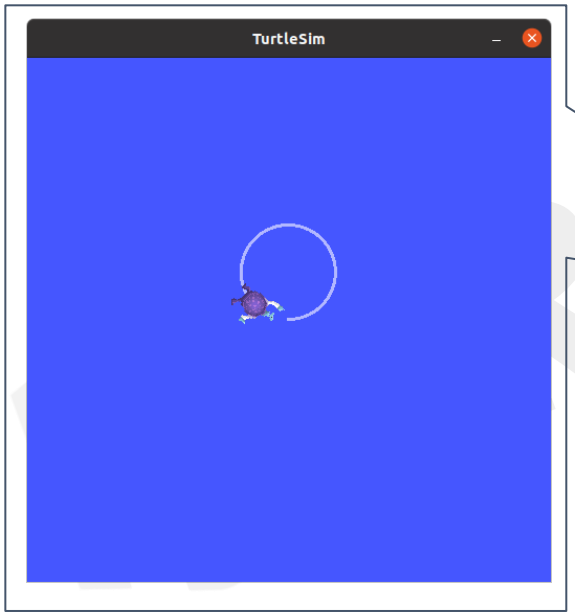
수신된 토픽 내용 확인

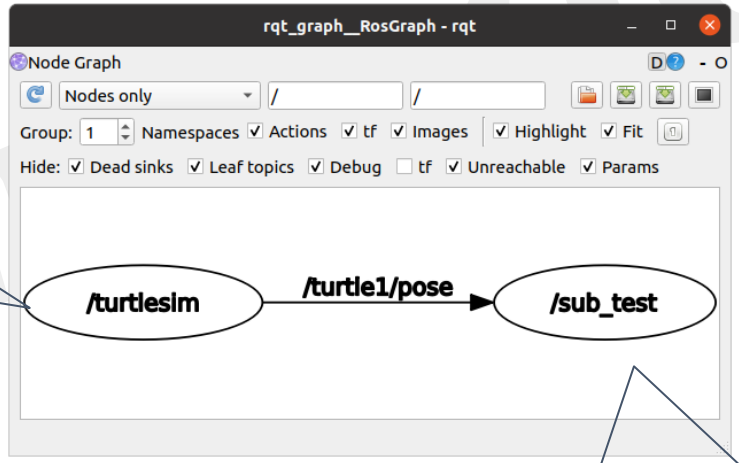
ROS2 Python 실습

topic 코드

>토픽 구독

rp.spin_once(test_node)



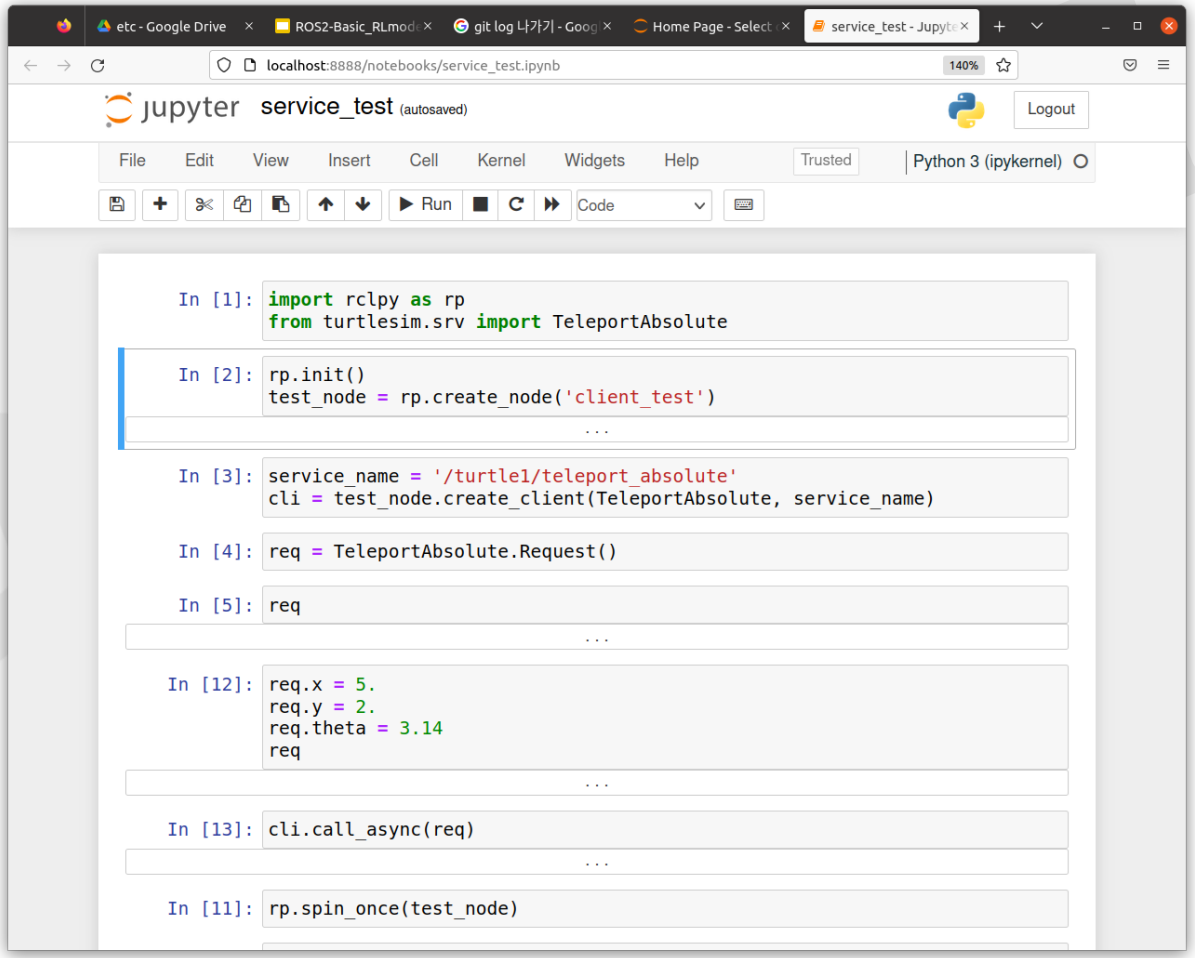



```
In [8]: rp.spin_once(test_node)

        turtle pose turtlesim.msg.Pose(x=4.840552806854248, y=5.830162525177002,
        theta=-0.7792094945907593, linear_velocity=0.0, angular_velocity=0.0)
        X:  4.840552806854248
        Y:  5.830162525177002
        Heading:  -0.7792094945907593
```

ROS2 Python 실습

service 코드

순서: node 생성 > client 생성 > client call >



프로그램 실습

ROS2 Python

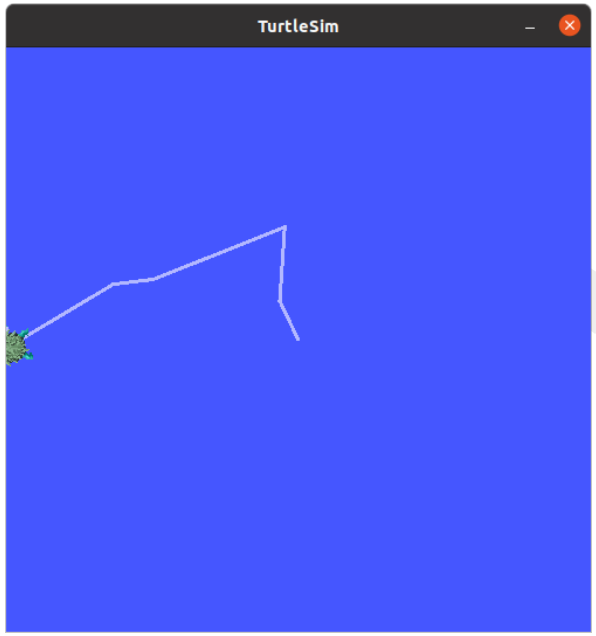
ROS2 실습

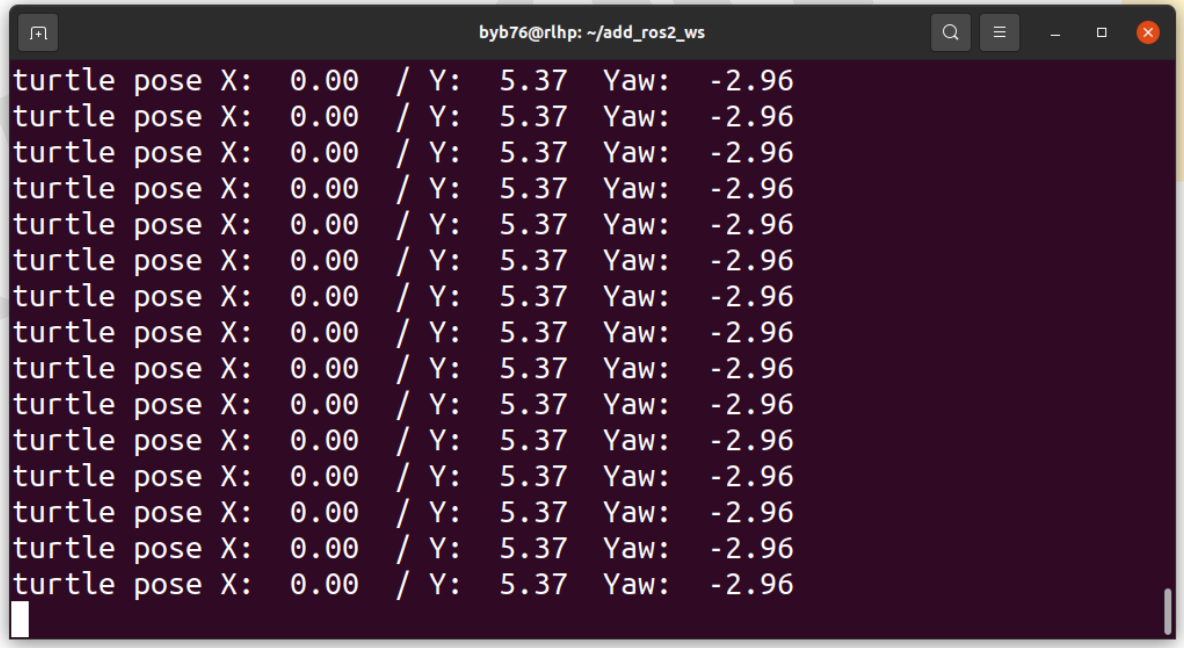
workspace 생성

생성: mkdir -p ~/ros2_ws/src

워크페이스 이름: ros2_ws

link: https://github.com/yunbum/ROS2_edu.git

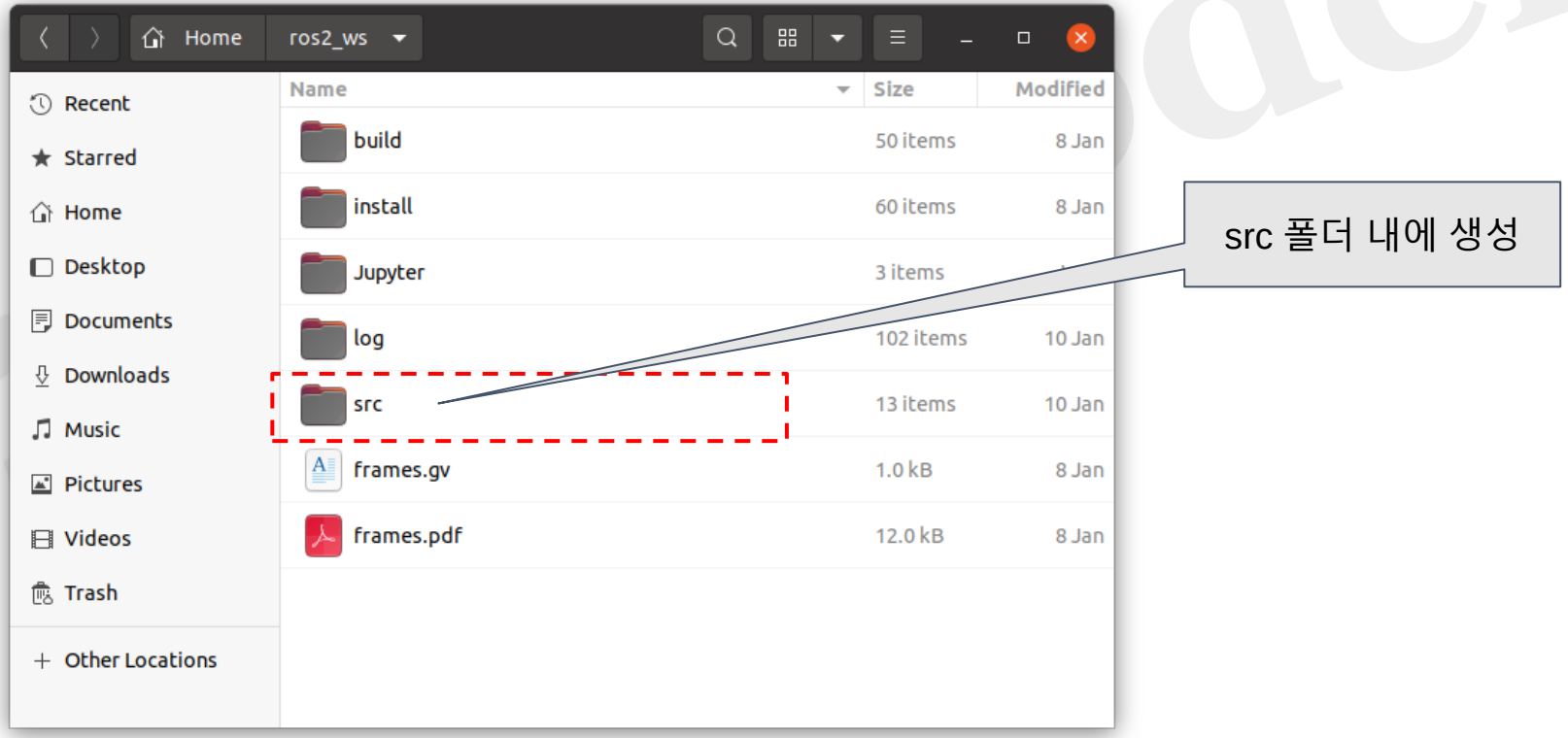




ROS2 실습

Package 생성

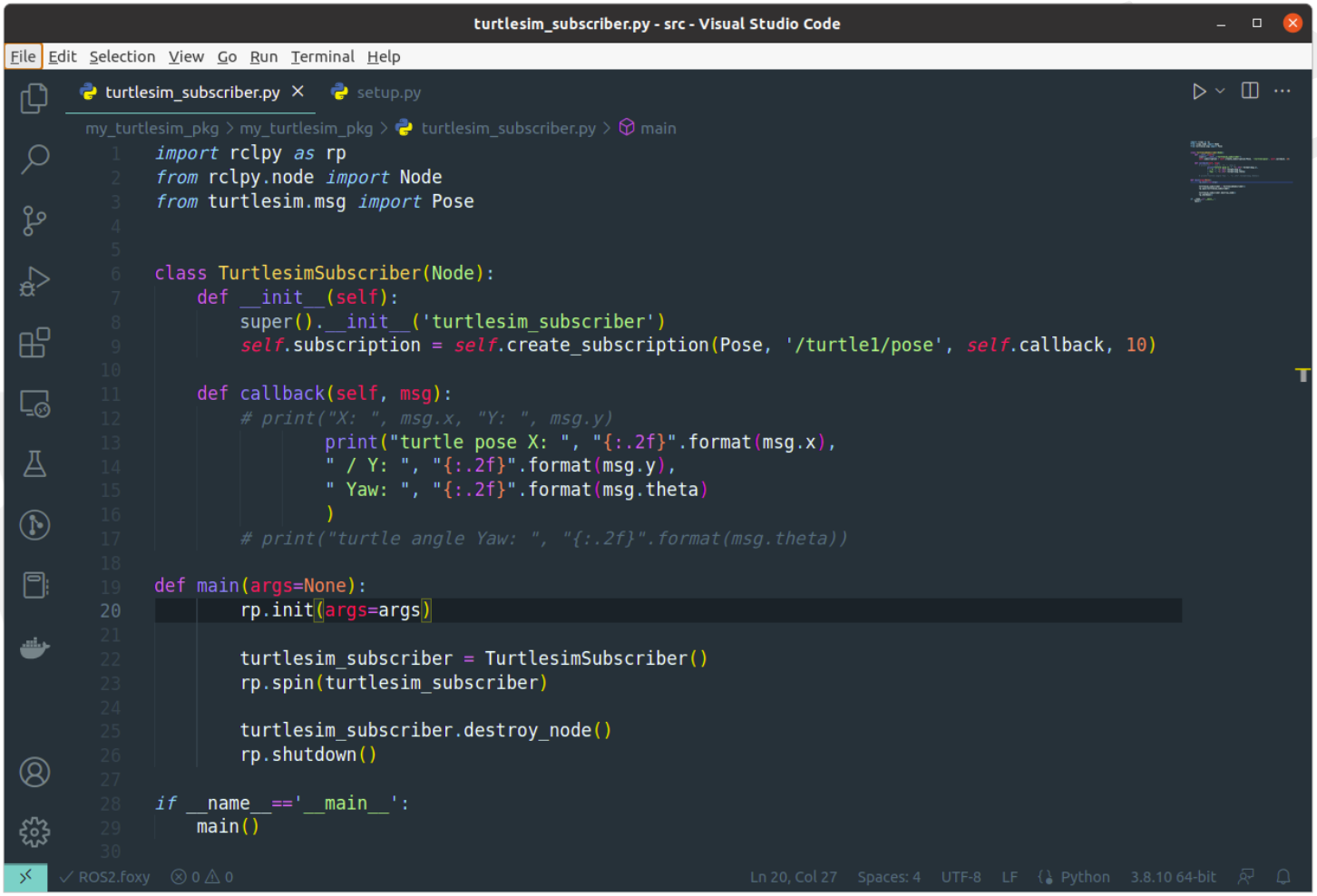
- cd ~/ros2_ws/src
- ros2 pkg create --build-type ament_python --node-name turtlesim_node my_turtlesim_pkg



ROS2 실습

Python code 생성

- turtlesim_subscriber.py



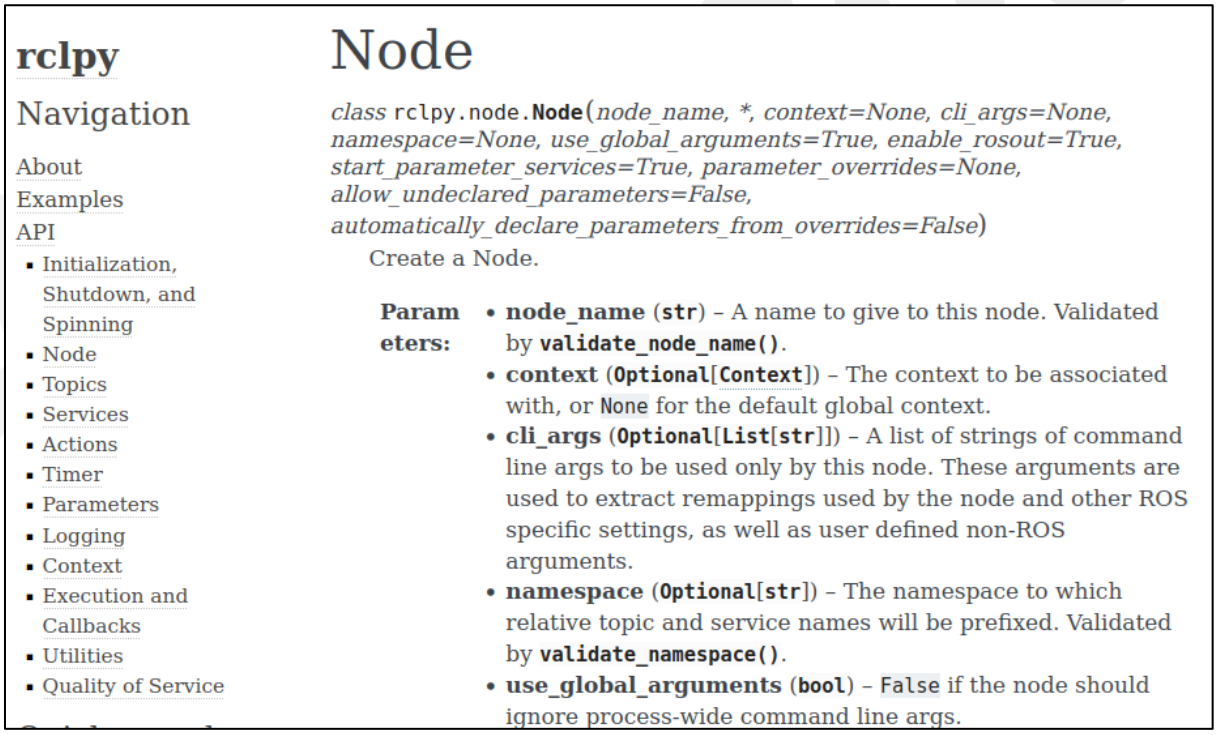
ROS2 실습

Python code 생성

- turtlesim_subscriber.py

Node class 정보

link: https://docs.ros2.org/latest/api/rclpy/api/node.html



rclpy

Navigation

About
Examples
API
- Initialization, Shutdown, and Spinning
- Node
- Topics
- Services
- Actions
- Timer
- Parameters
- Logging
- Context
- Execution and Callbacks
- Utilities
- Quality of Service

Node

class `rclpy.node.`**`Node`**(*node_name, *, context=None, cli_args=None, namespace=None, use_global_arguments=True, enable_rosout=True, start_parameter_services=True, parameter_overrides=None, allow_undeclared_parameters=False, automatically_declare_parameters_from_overrides=False*)

Create a Node.

Parameters:

- **node_name** (**str**) – A name to give to this node. Validated by **validate_node_name()**.
- **context** (**Optional**[**Context**]) – The context to be associated with, or `None` for the default global context.
- **cli_args** (**Optional**[**List**[**str**]]) – A list of strings of command line args to be used only by this node. These arguments are used to extract remappings used by the node and other ROS specific settings, as well as user defined non-ROS arguments.
- **namespace** (**Optional**[**str**]) – The namespace to which relative topic and service names will be prefixed. Validated by **validate_namespace()**.
- **use_global_arguments** (**bool**) – `False` if the node should ignore process-wide command line args.

ROS2 실습

dependency 확인

```
# cd if you're still in the ``src`` directory with the ``ros_tutorials`` clone
```

- cd ..
- rosdep install -i --from-path src --rosdistro foxy -y

colcon build 실행

- colcon build

—----------

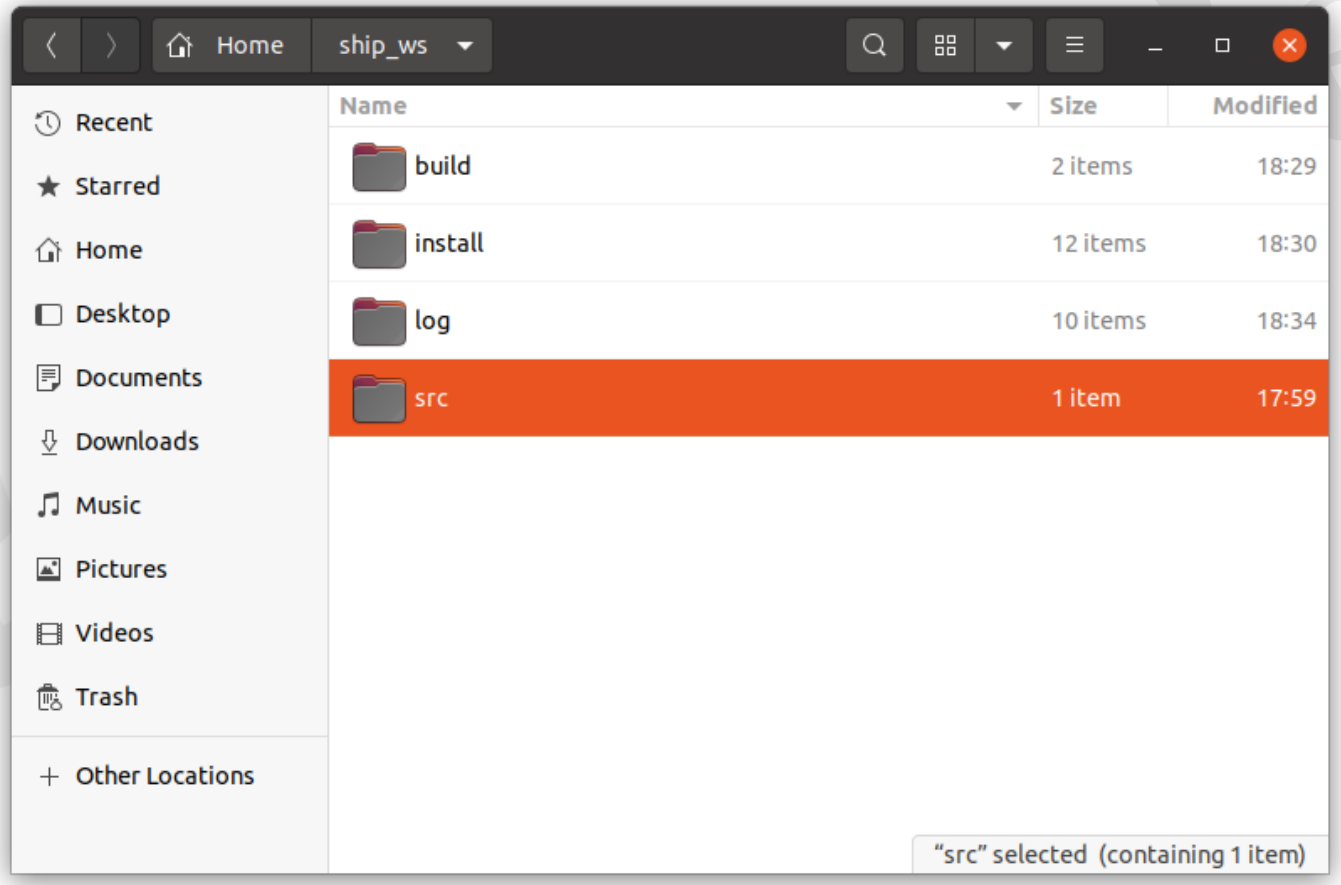
<결과>

- build, install, log 폴더 생성

ROS2 실습

build 결과확인

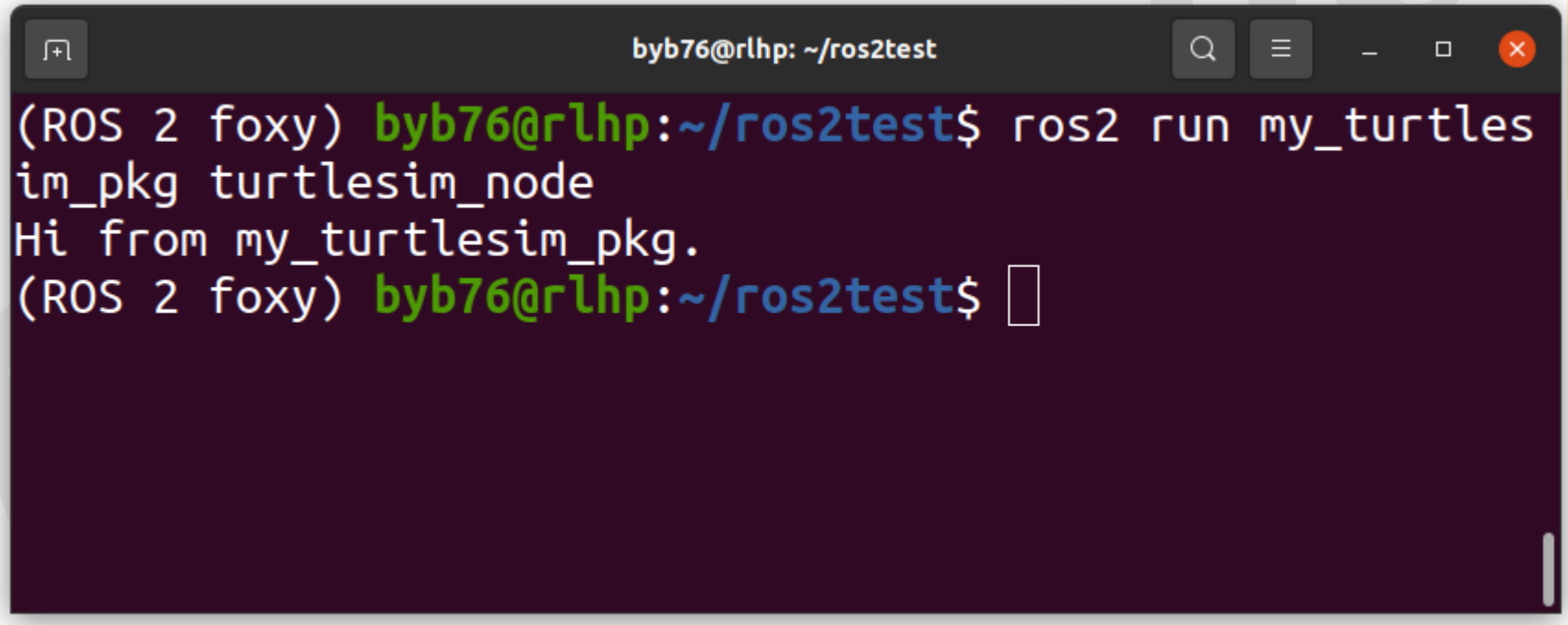
- build, install, log 파일생성



ROS2 실습

package 실행

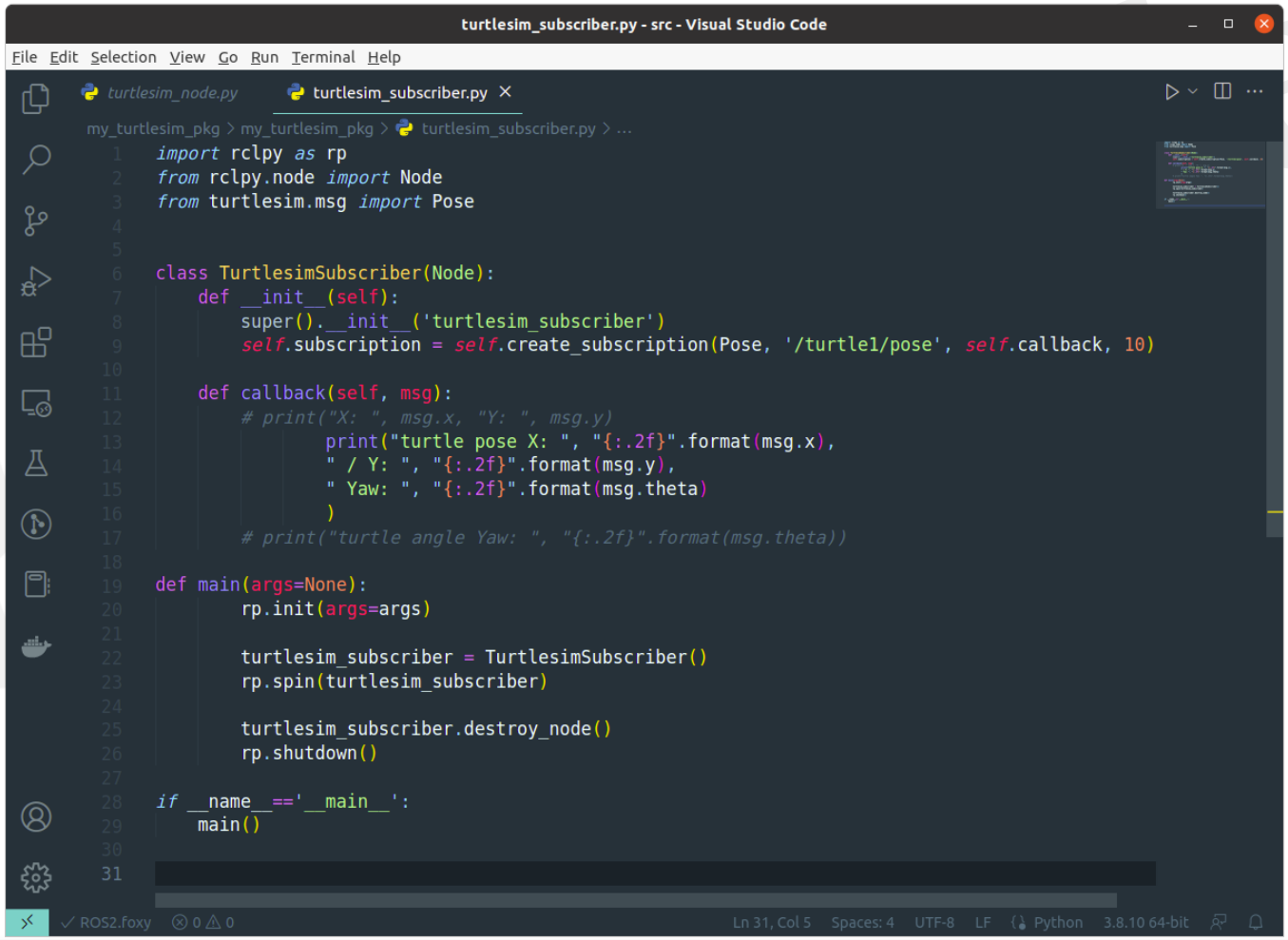
- . install/localsetup.bash
- ros2 run my_turtlesim_pkg turtle_subscriber



ROS2 실습

turtlesim topic subscriber 노드 생성

파이선 파일 생성: turtlesim_subscriber.py



ROS2 실습

turtlesim topic subscriber 노드 생성

setup.py 업데이트: entrypoint 추가 (, 주의)

Open | setup.py ~/ros2test/src/my_turtlesim_pkg | Save

my_turtlesim_subscriber.py | setup.py

```
1 from setuptools import setup
2
3 package_name = 'my_turtlesim_pkg'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=[package_name],
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11             ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='byb76',
17     maintainer_email='tge1375@naver.com',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'turtlesim_node = my_turtlesim_pkg.turtlesim_node:main',
24             'turtle_subscriber = turtlesim_pkg.my_turtlesim_subscriber:main'
25         ],
26     },
27 )
```

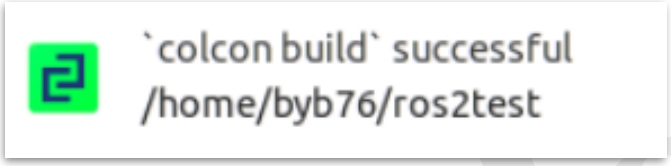
Python | Tab Width: 8 | Ln 24, Col 67 | INS

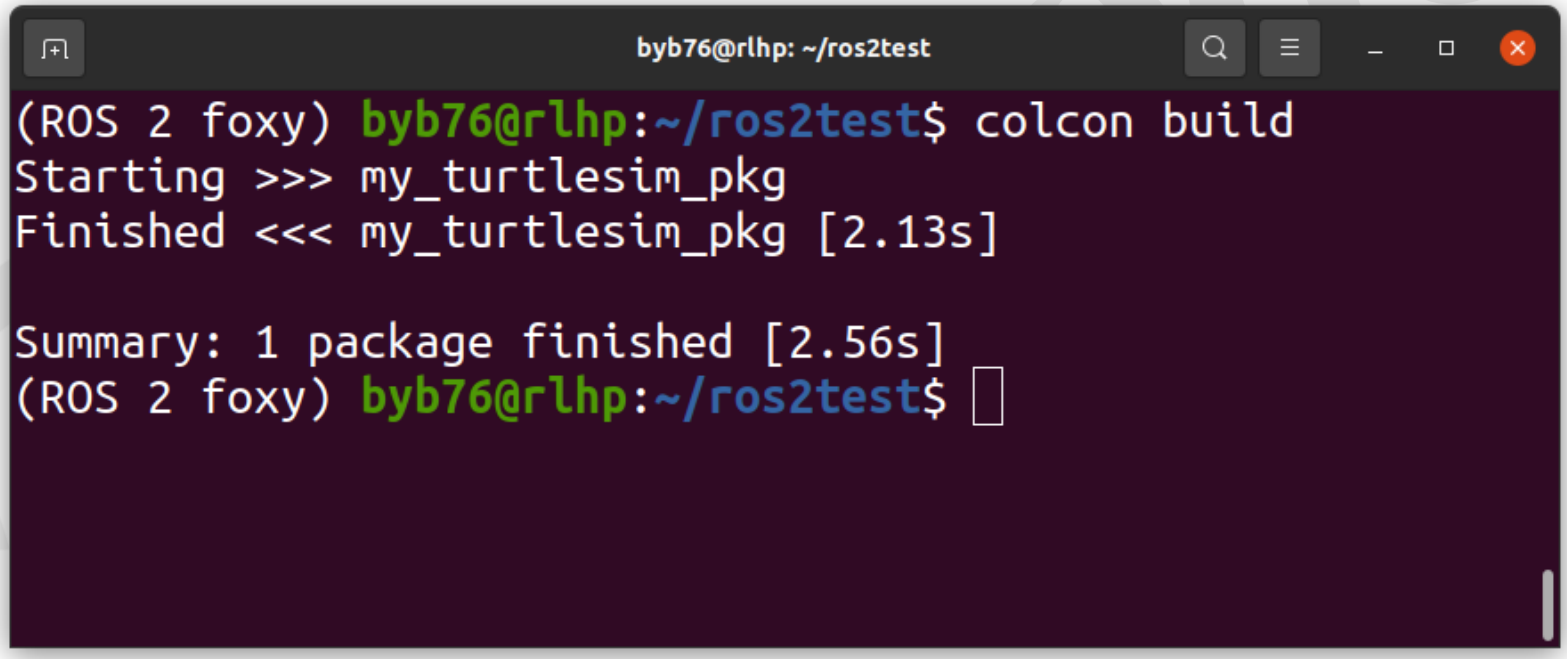
ROS2 실습

package build

$colcon build

$. install/localsetup.bash



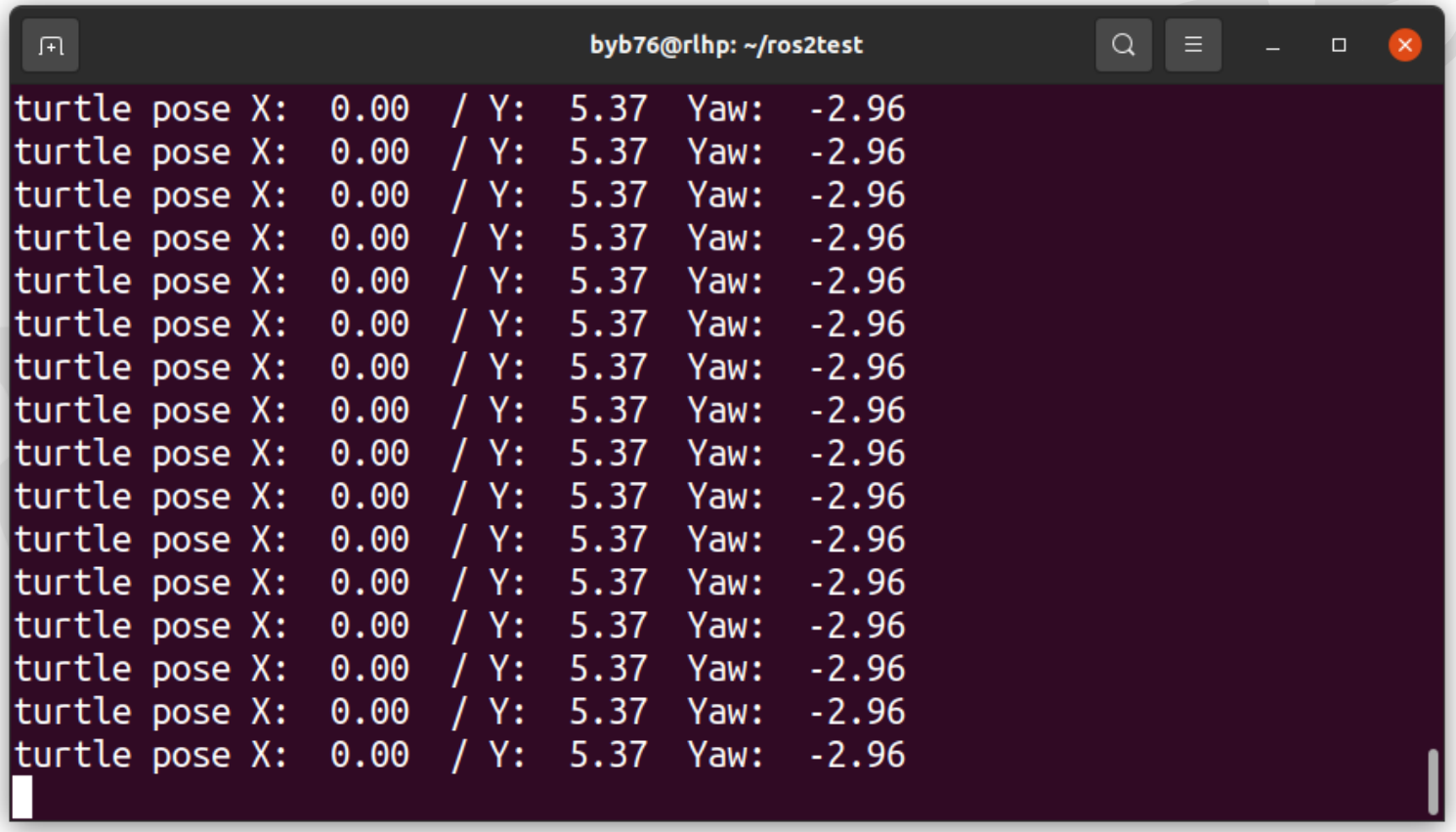


ROS2 실습

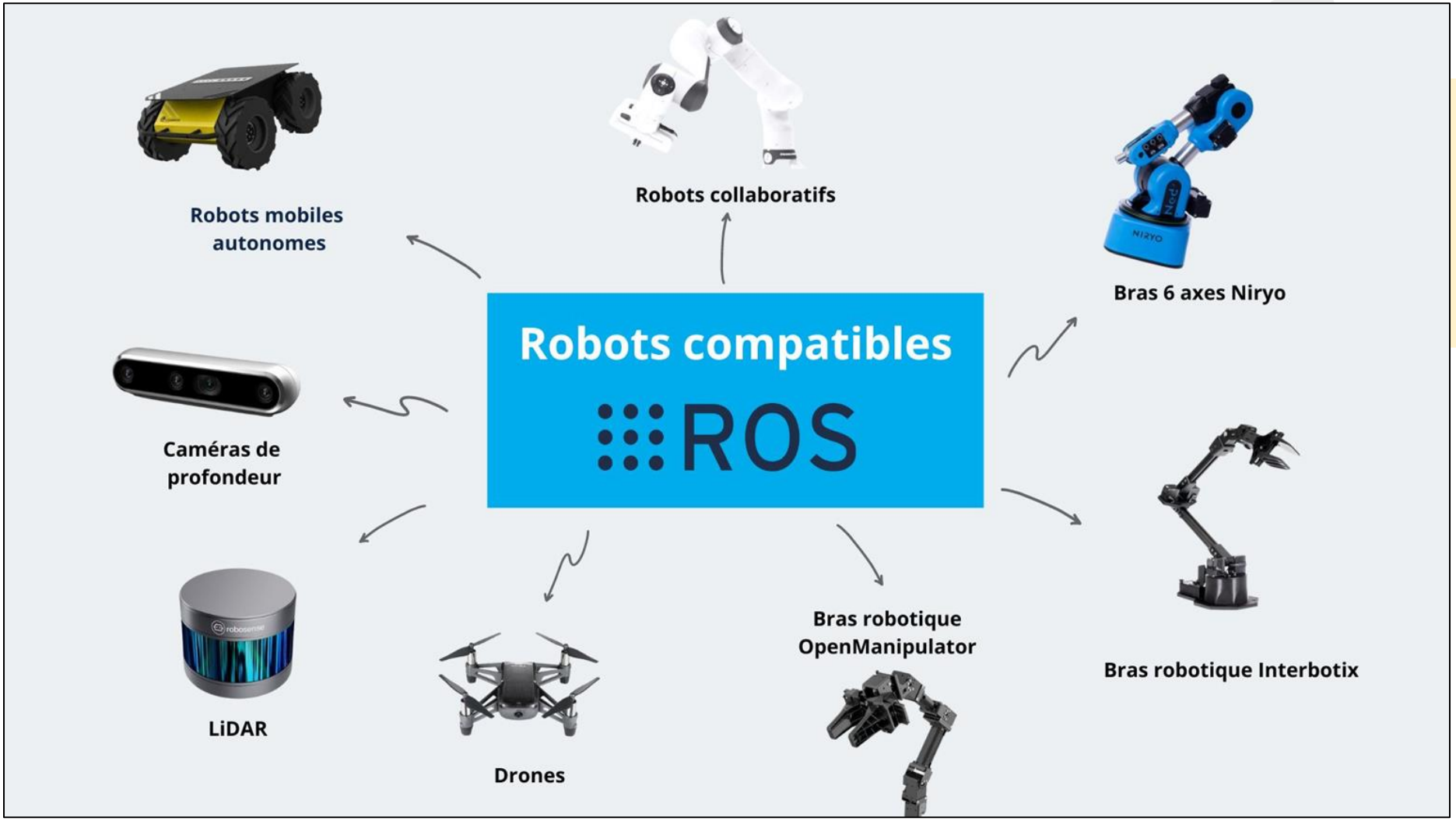
package 실행

$ros2 run my_turtlesim_pkg turtle_subscriber

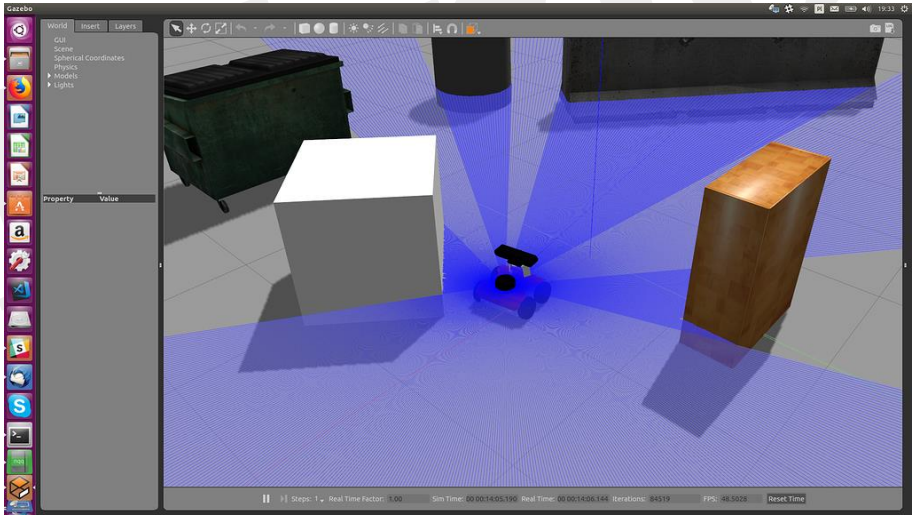
>위 이름은 setup.py entrypoint 이름과 일치



ROS2 HW 실습


Robots mobiles autonomes
Robots collaboratifs
Bras 6 axes Niryo
Robots compatibles
ROS
Caméras de profondeur
LiDAR
Drones
Bras robotique OpenManipulator
Bras robotique Interbotix

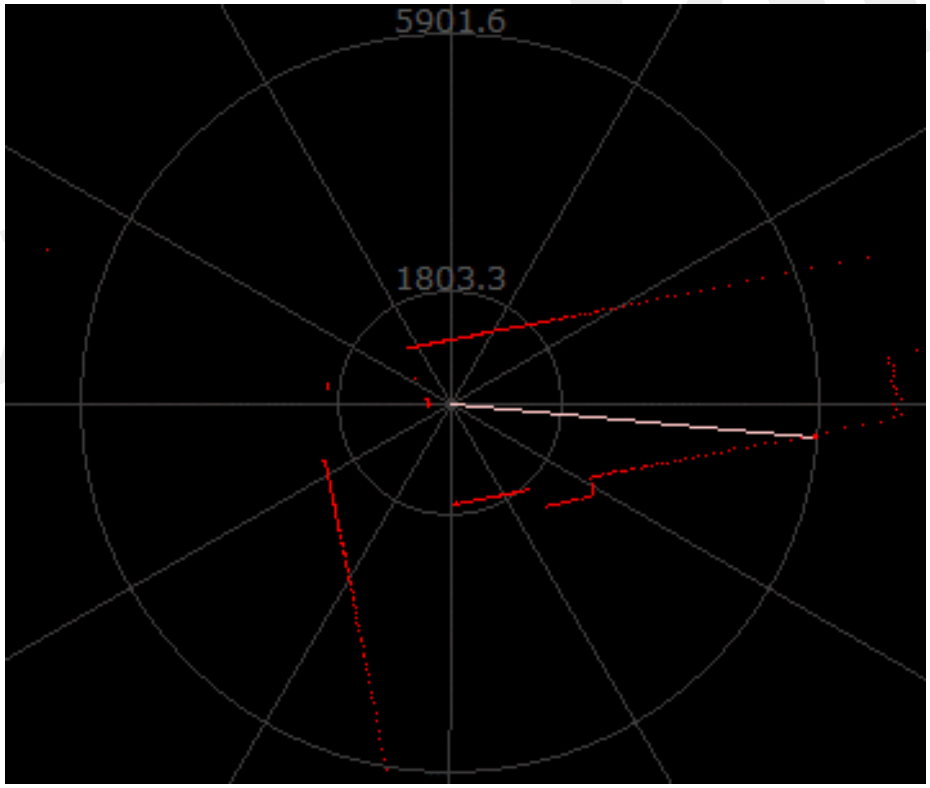
Lidar sensor
실습



ROS2 HW

Lidar

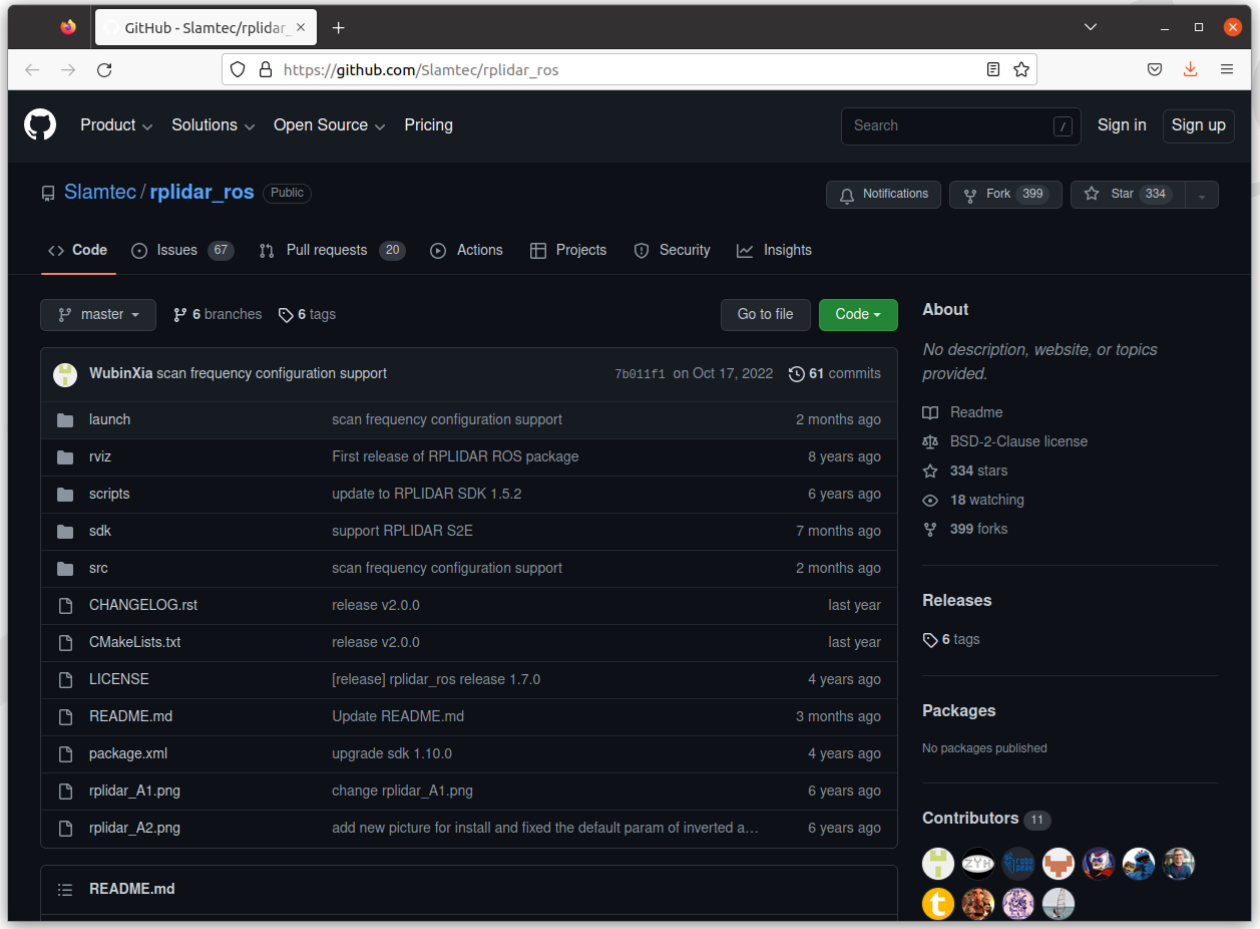
demo - rplidar a2



ROS2 HW

RPLidar

link: https://github.com/Slamtec/rplidar_ros

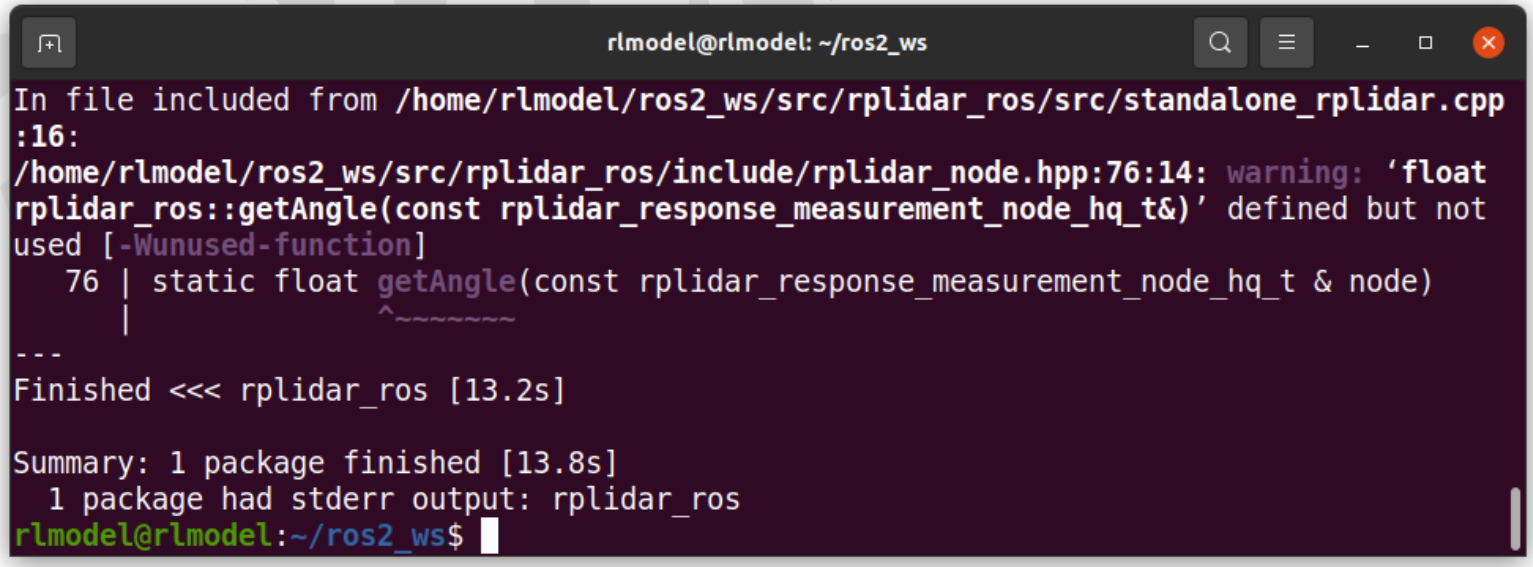


Lidar

Demo - rplidar a2

link: https://github.com/Slamtec/rplidar_ros

$ git clone -b ros2 https://github.com/Slamtec/rplidar_ros.git
$ colcon build --package-select rplidar_ros
$. intall/setup.bash
$ ros2 launch rpliar_ros view_rplidar.launch.py



Lidar

Demo - chmod

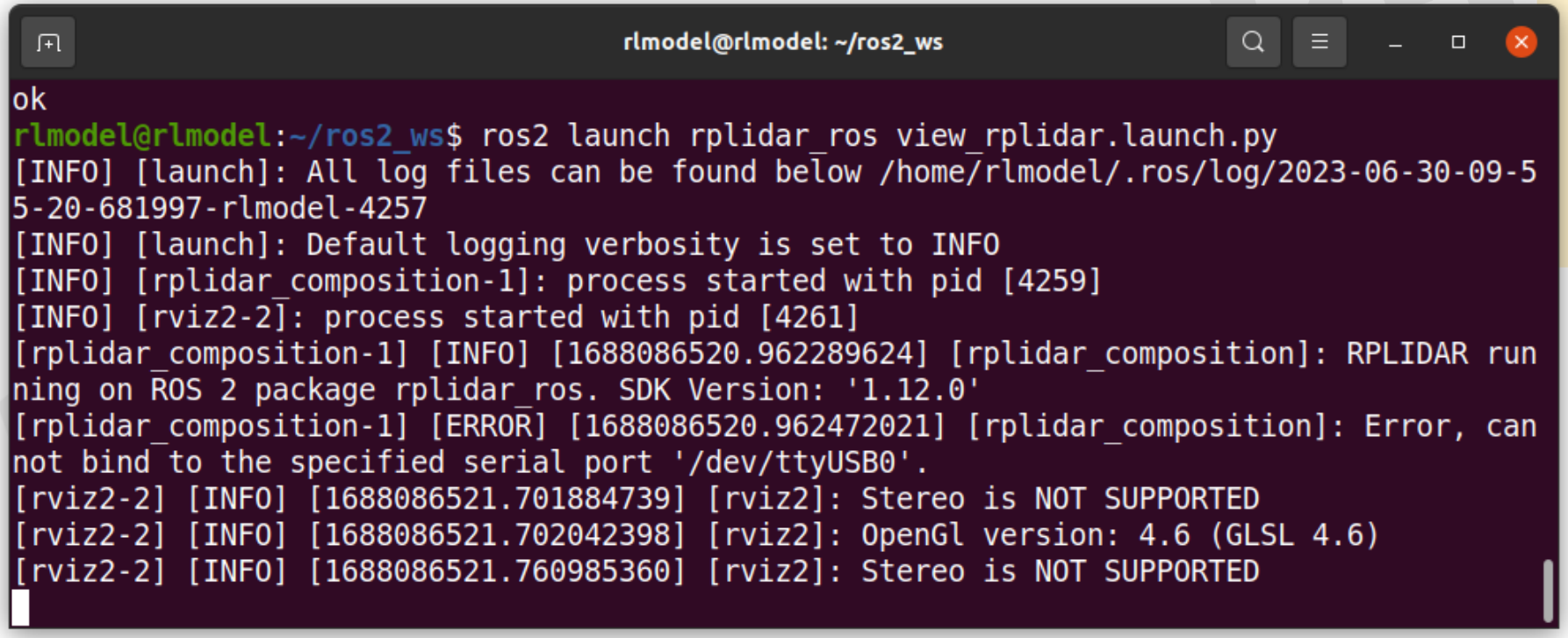
$ ls /dev/ttyUSB*

$ sudo chmod 777 /dev/ttyUSB0 (USB0, USB1...)

Lidar

Demo - rplidar a2

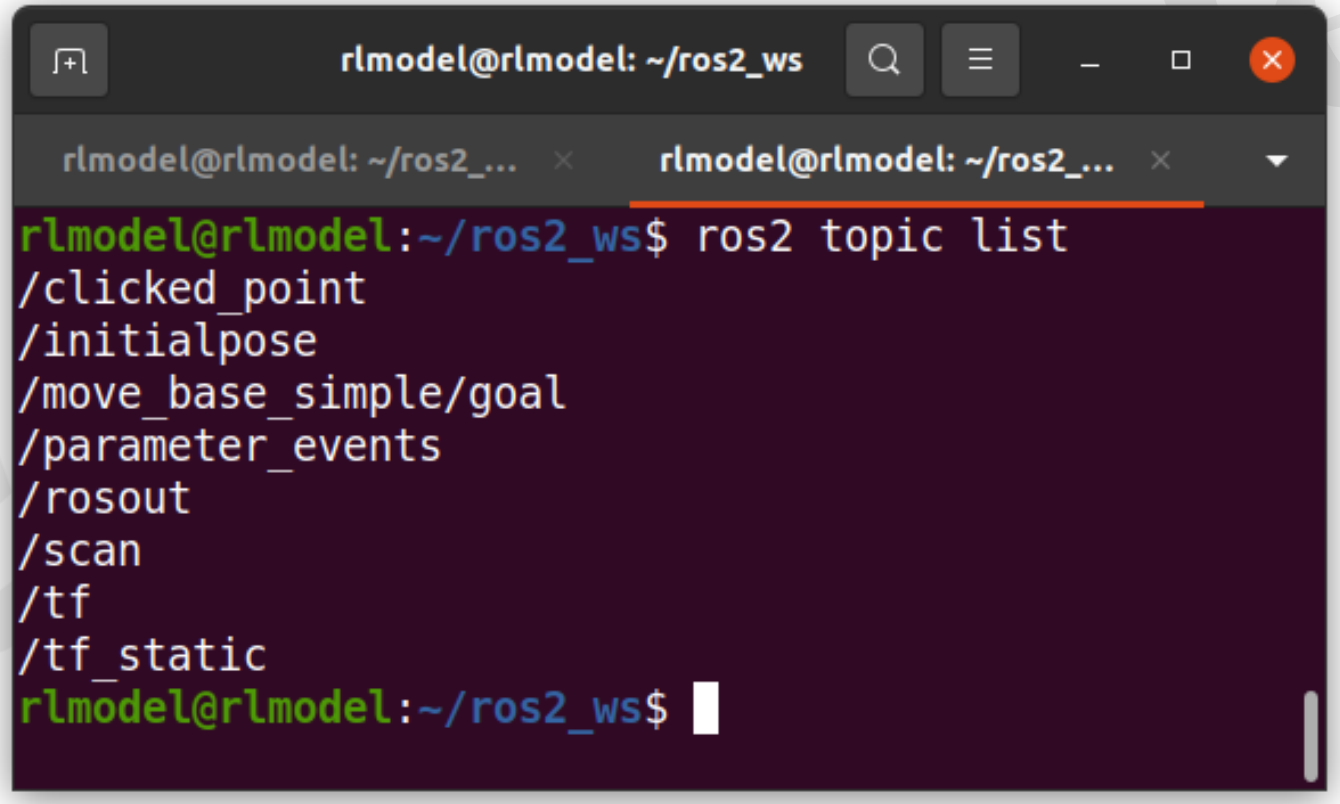
$ros2 launch rpliar_ros view_rplidar.launch.py



Lidar

Demo - rplidar a2

$ros2 launch rpliar_ros view_rplidar.launch.py

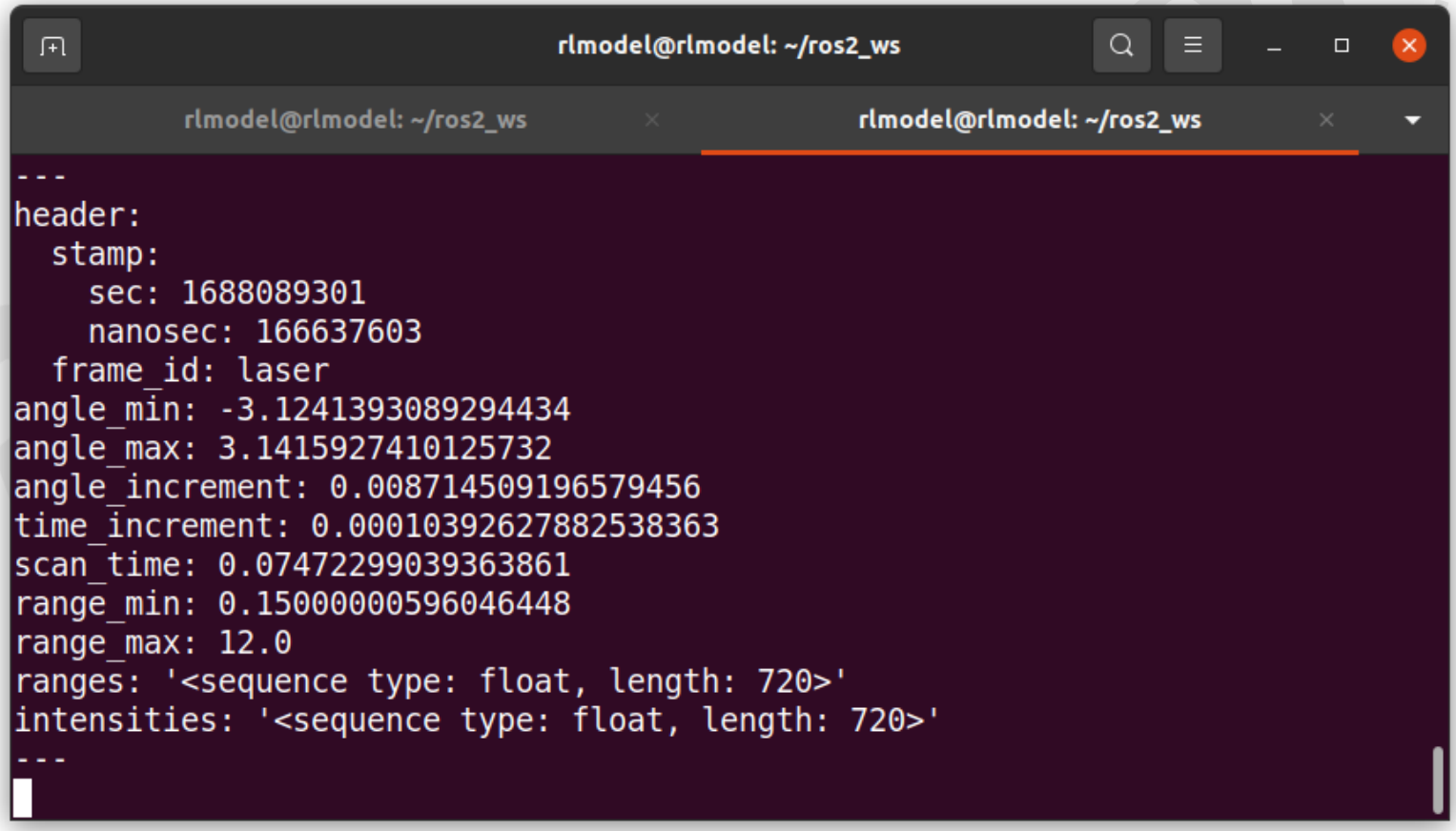


Lidar 2D

Slamtec RPLidar A2

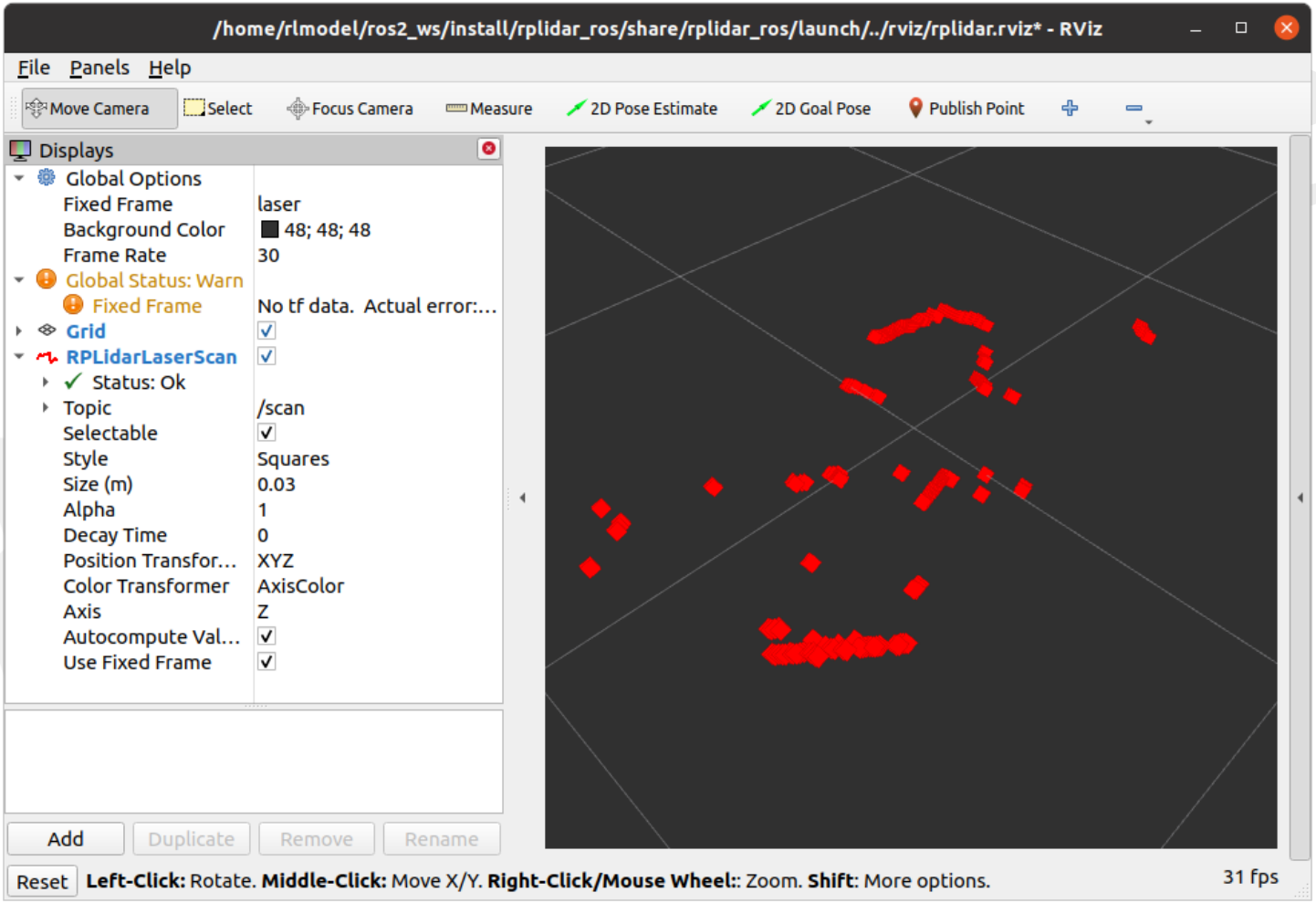
$ ros2 topic echo /scan --no-arr

check frame_id



Lidar 2D

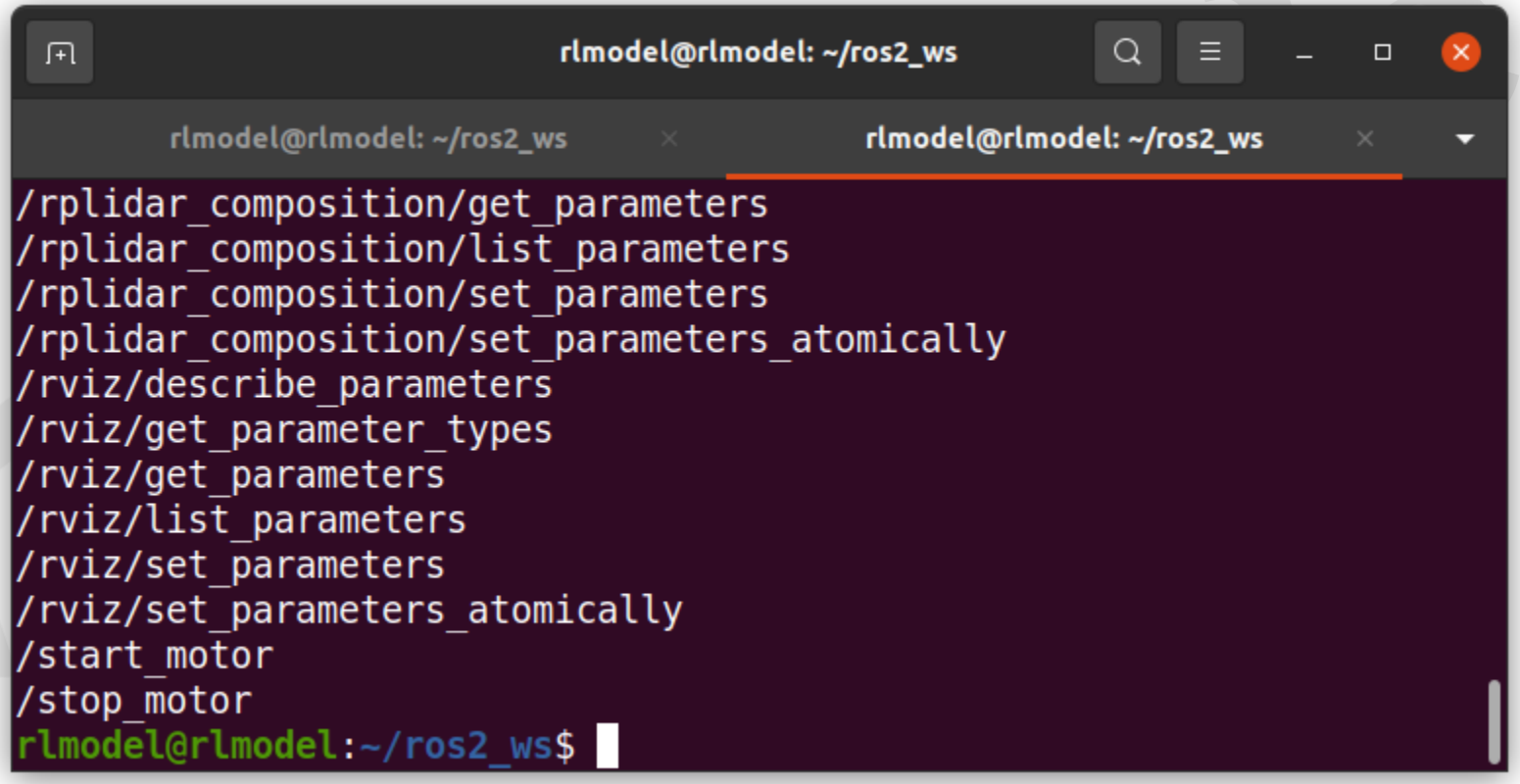
Demo - rplidar a2



Lidar

Demo - rplidar service

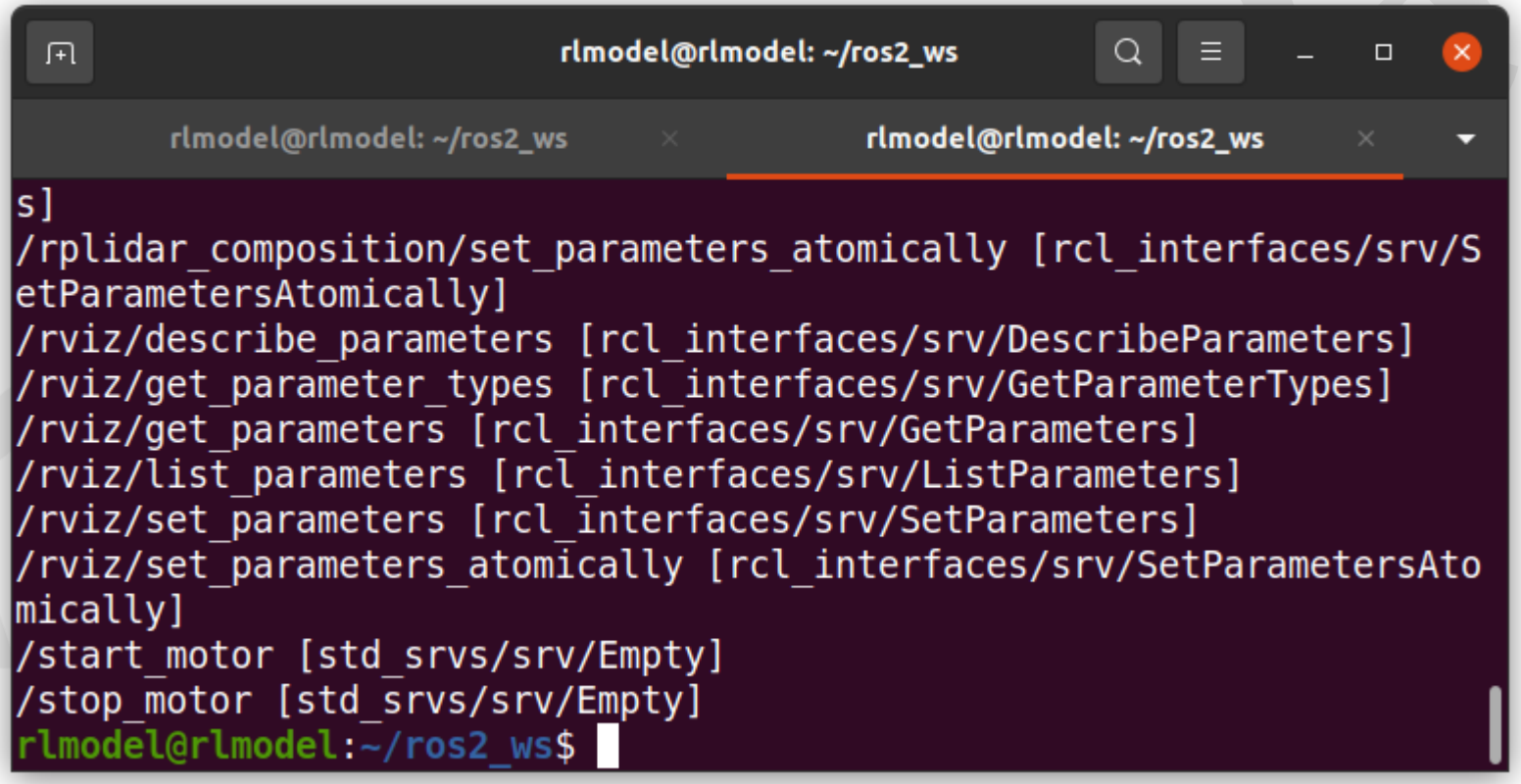
check service: $ ros2 service list



Lidar

Demo - rplidar service

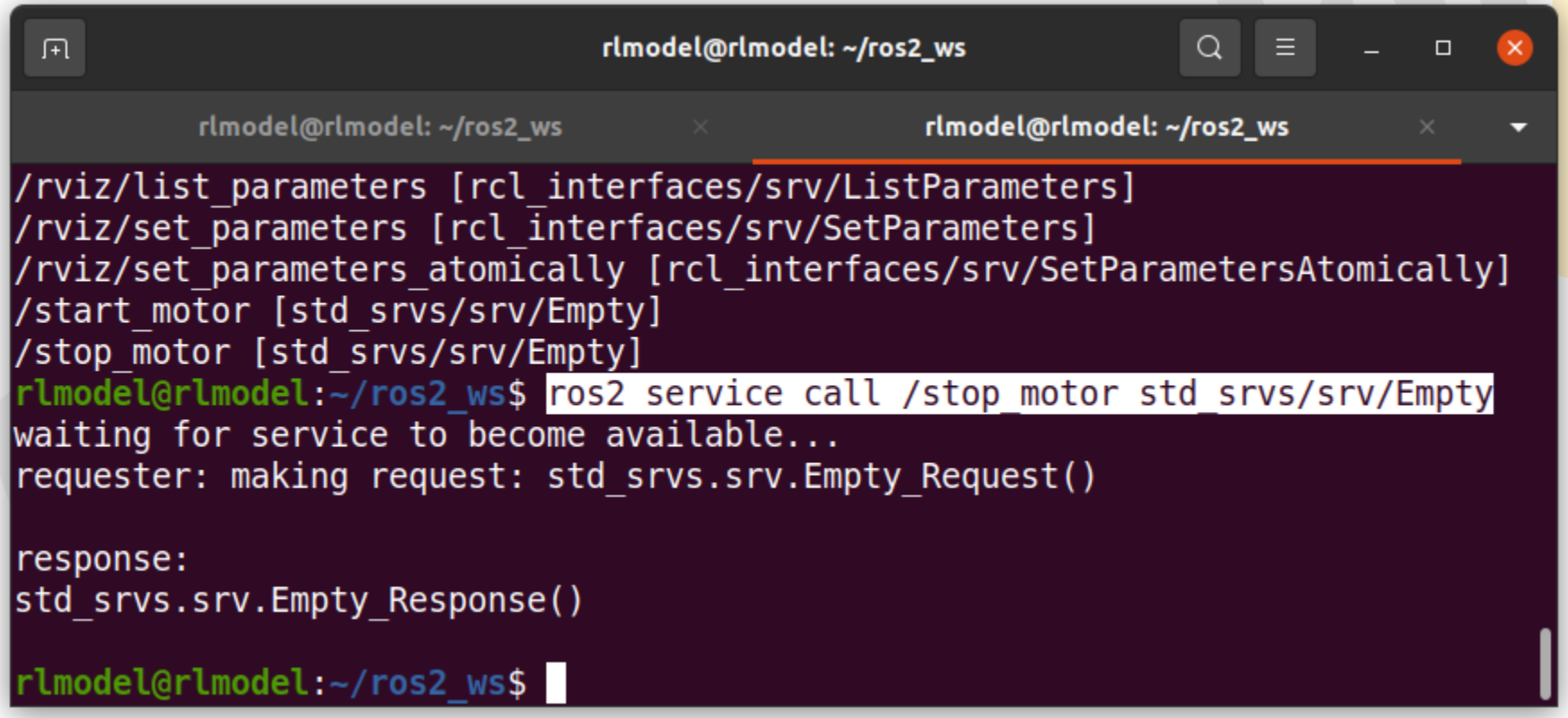
check service: $ ros2 service list -t



Lidar

Demo - rplidar service stop_motor

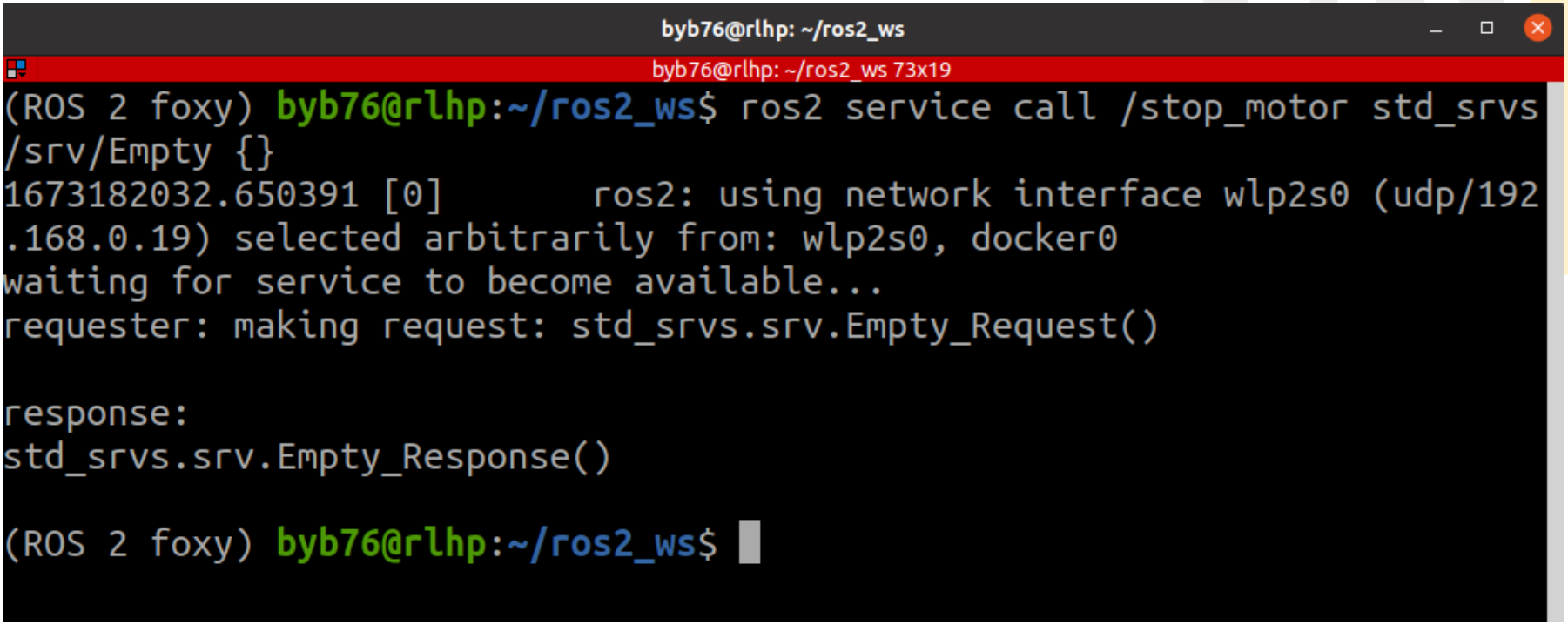
$ ros2 service call /stop_motor std_srvs/srv/Empty



Lidar

Demo - rplidar service stop motor

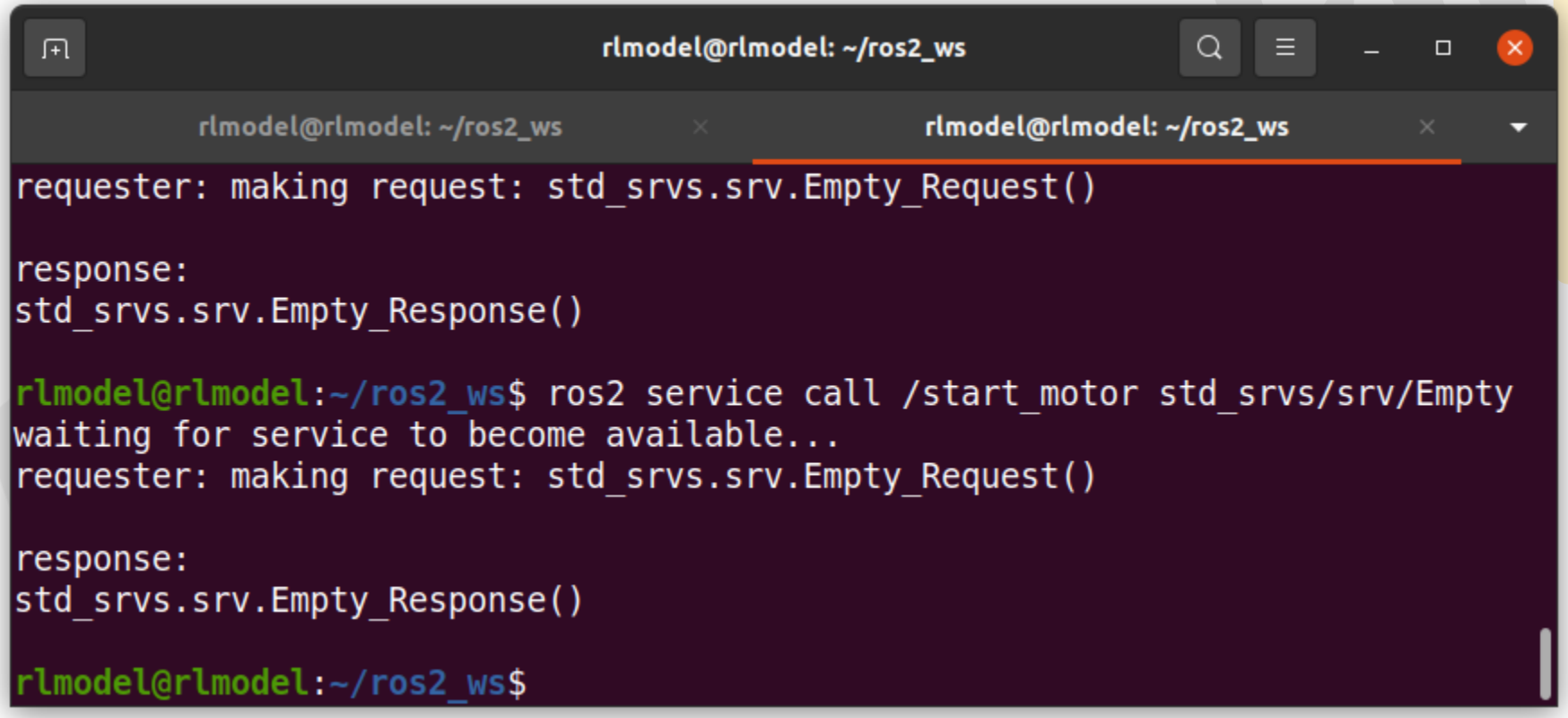
$ ros2 service call /stop_motor std_srvs/srv/Empty {}



Lidar

Demo - rplidar service start_motor

$ ros2 service call /stop_motor std_srvs/srv/Empty

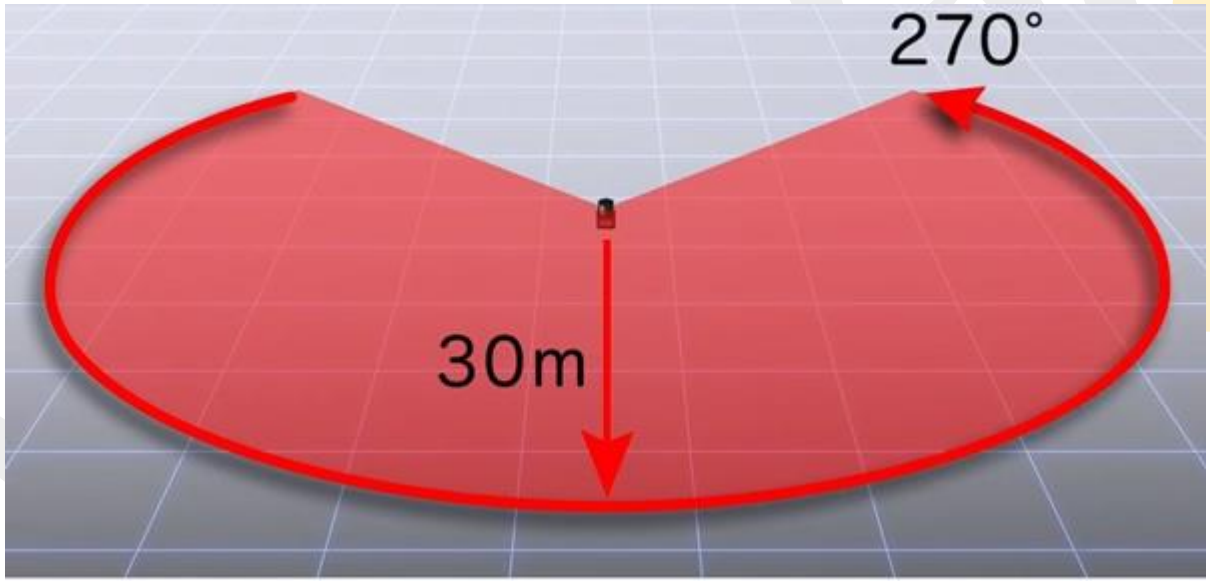


Lidar 2D

scan topic

laser scan data

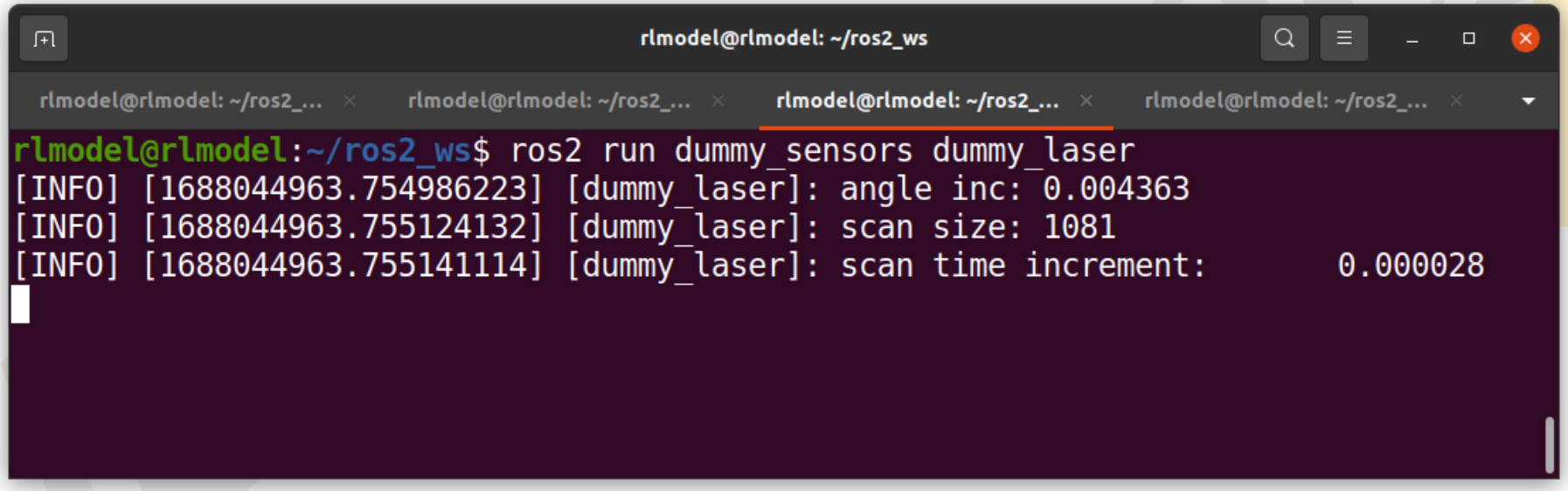
>simulation example



Lidar 2D

scan topic

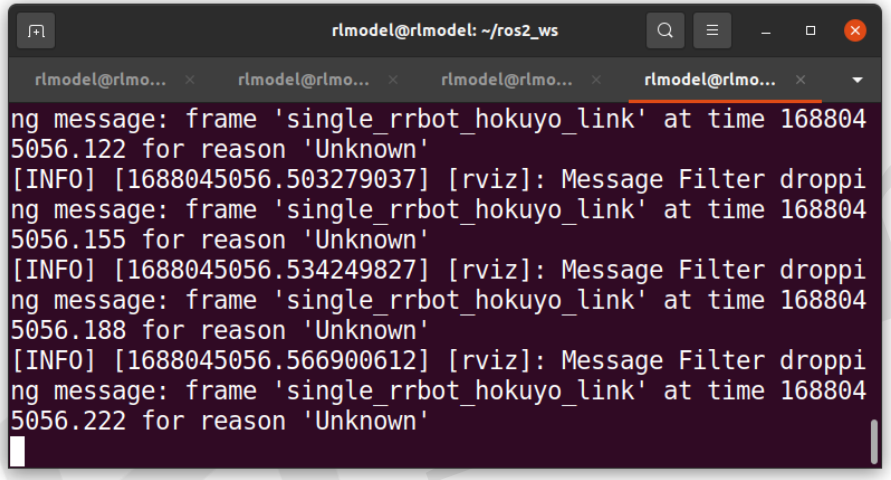
dummy_sensor package

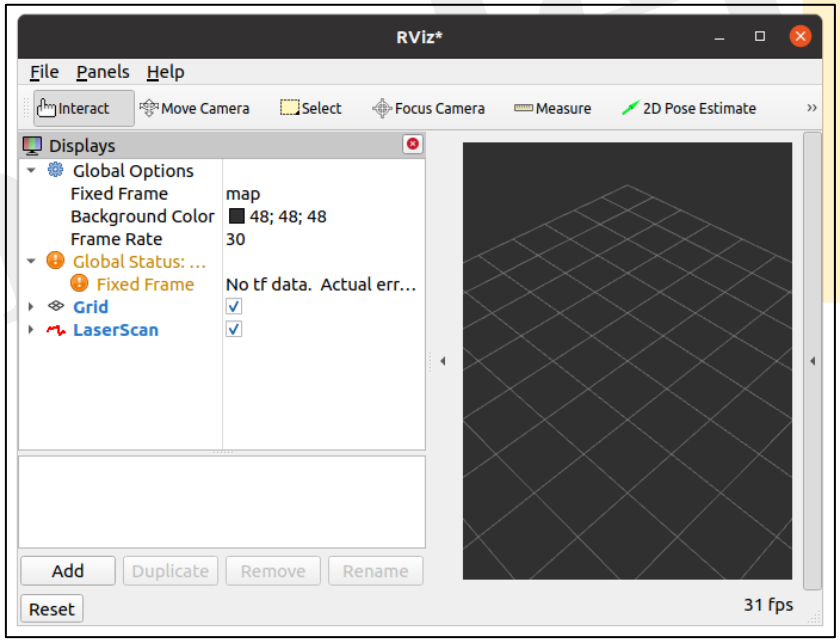


Lidar 2D

scan topic

dummy_sensor package - rviz
frame: single_rrbot_hokuyo_link

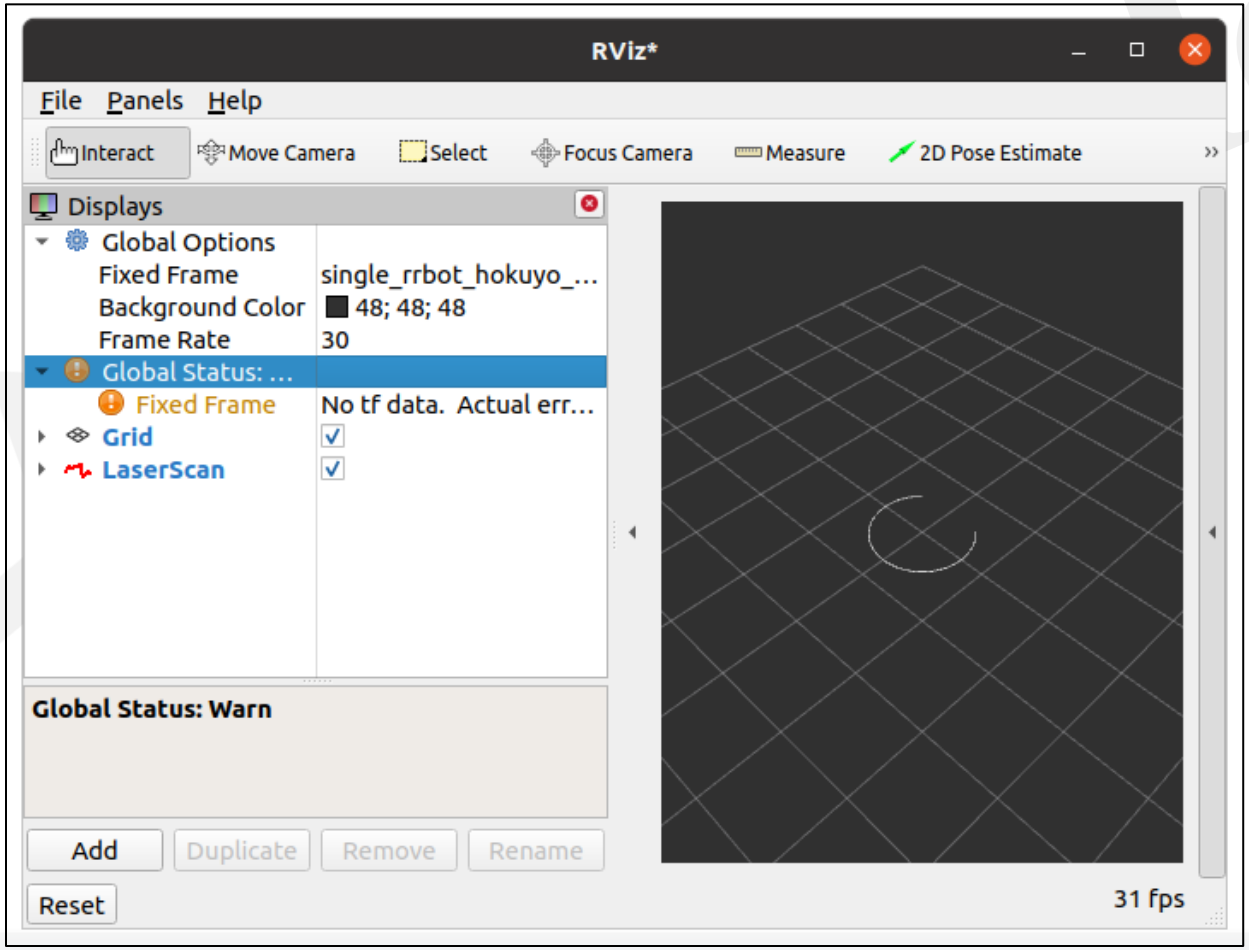




Lidar 2D

scan topic

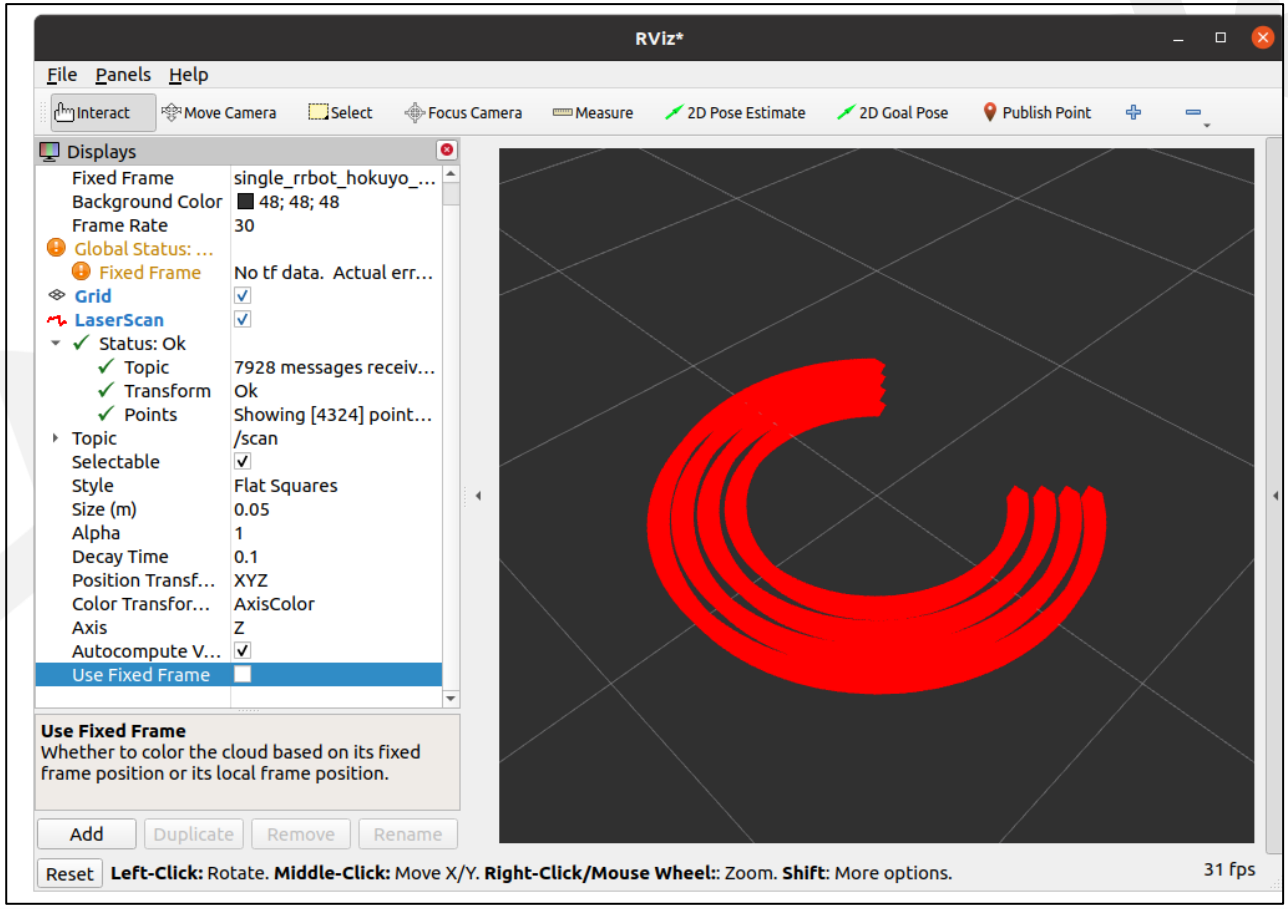
dummy_sensor package - rviz
frame: single_rrbot_hokuyo_link



Lidar 2D

scan topic

dummy_sensor package - rviz
frame: single_rrbot_hokuyo_link

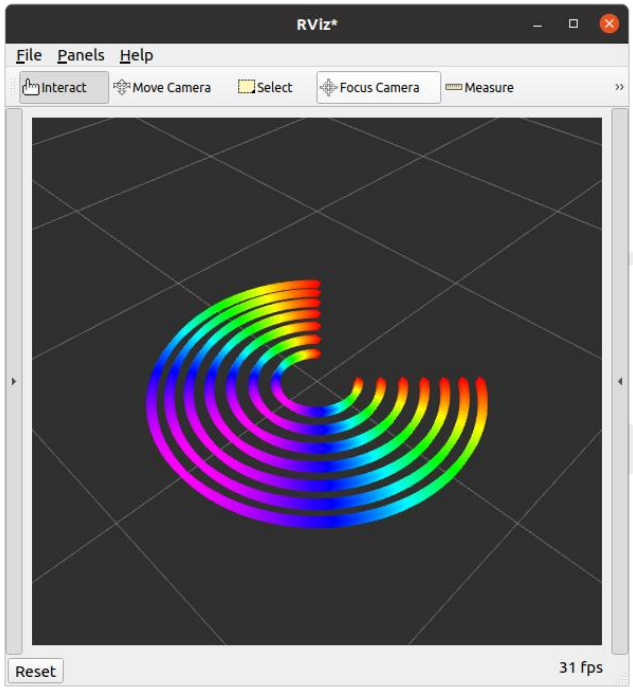


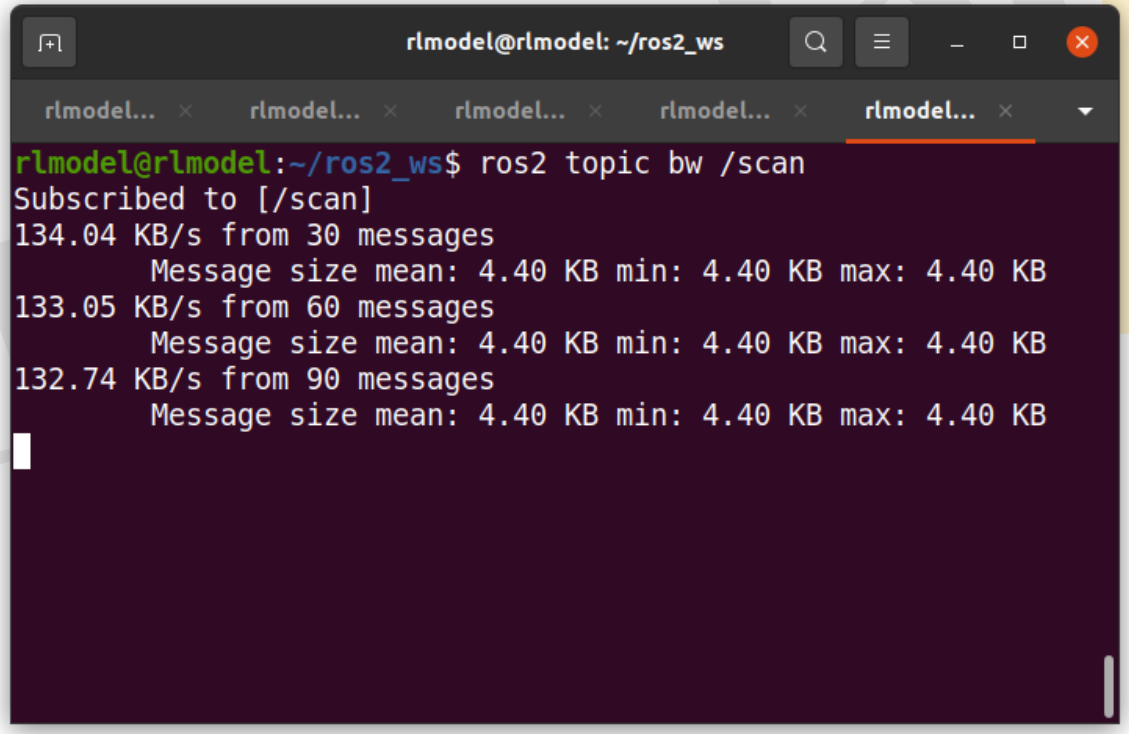
Lidar 2D

topic bandwidth

지정된 토픽 메시지의 송수신되는 토픽의 초당 대역폭

$ ros2 topic bw /scan



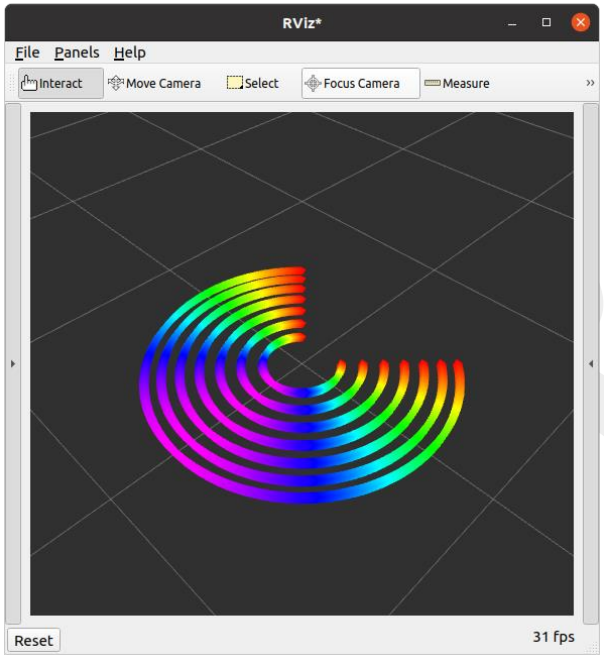


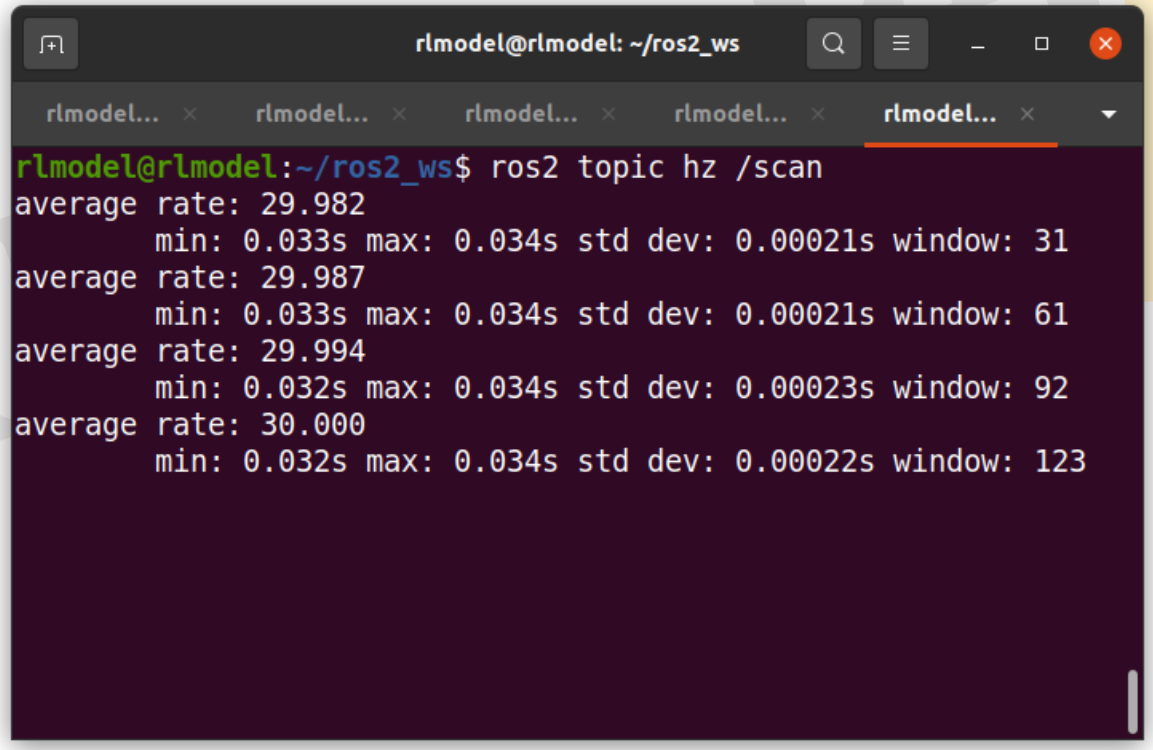
Lidar 2D

topic hertz

토픽의 전송 주기

$ ros2 topic hz /scan



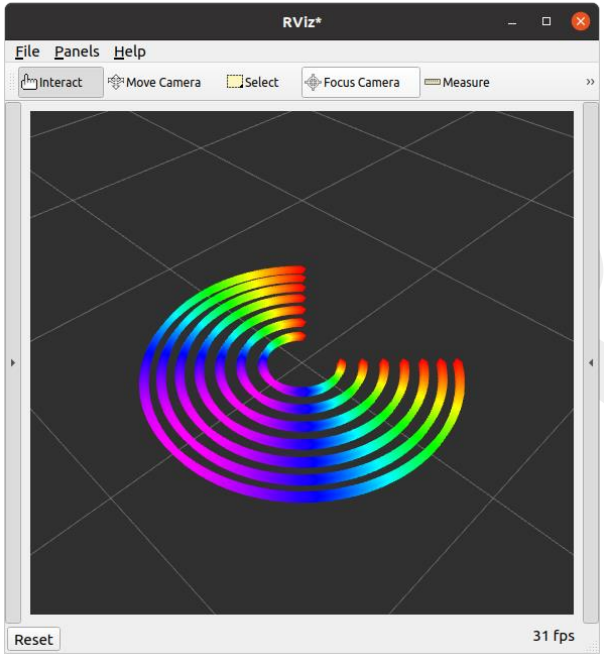


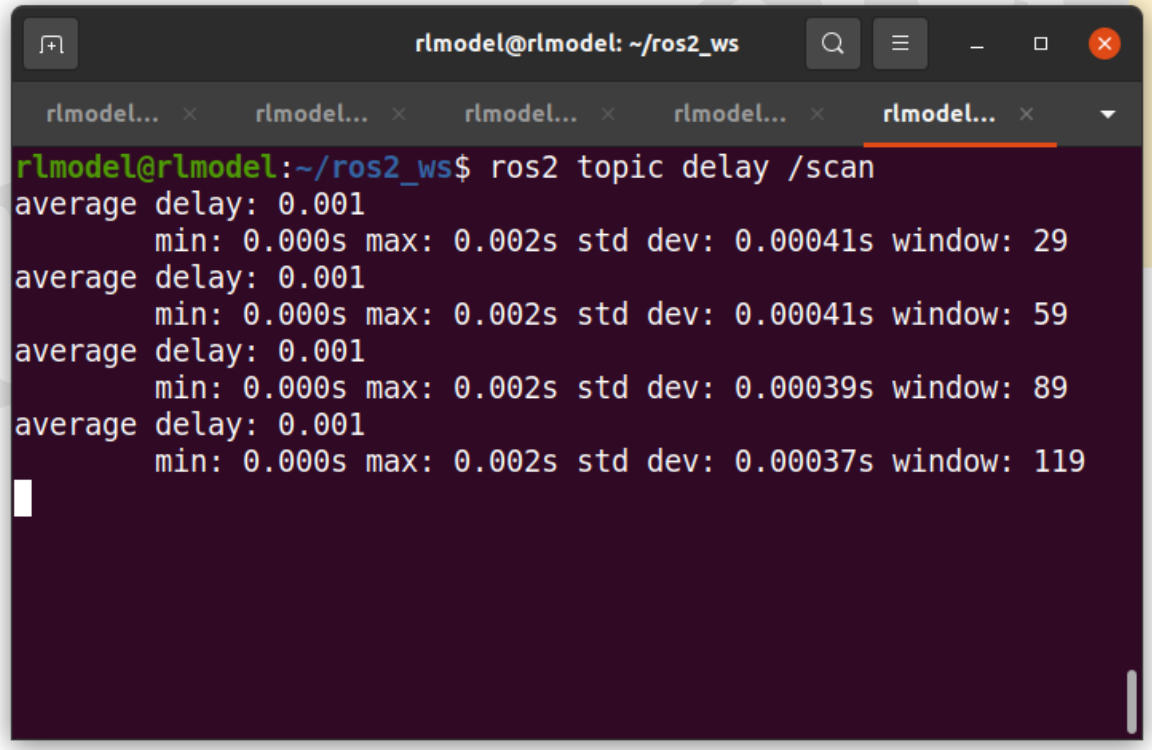
Lidar 2D

topic delay

메시지 발행 시간과 구독 시간의 차를 계산하여 지연 시간을 확인

$ ros2 topic delay /scan



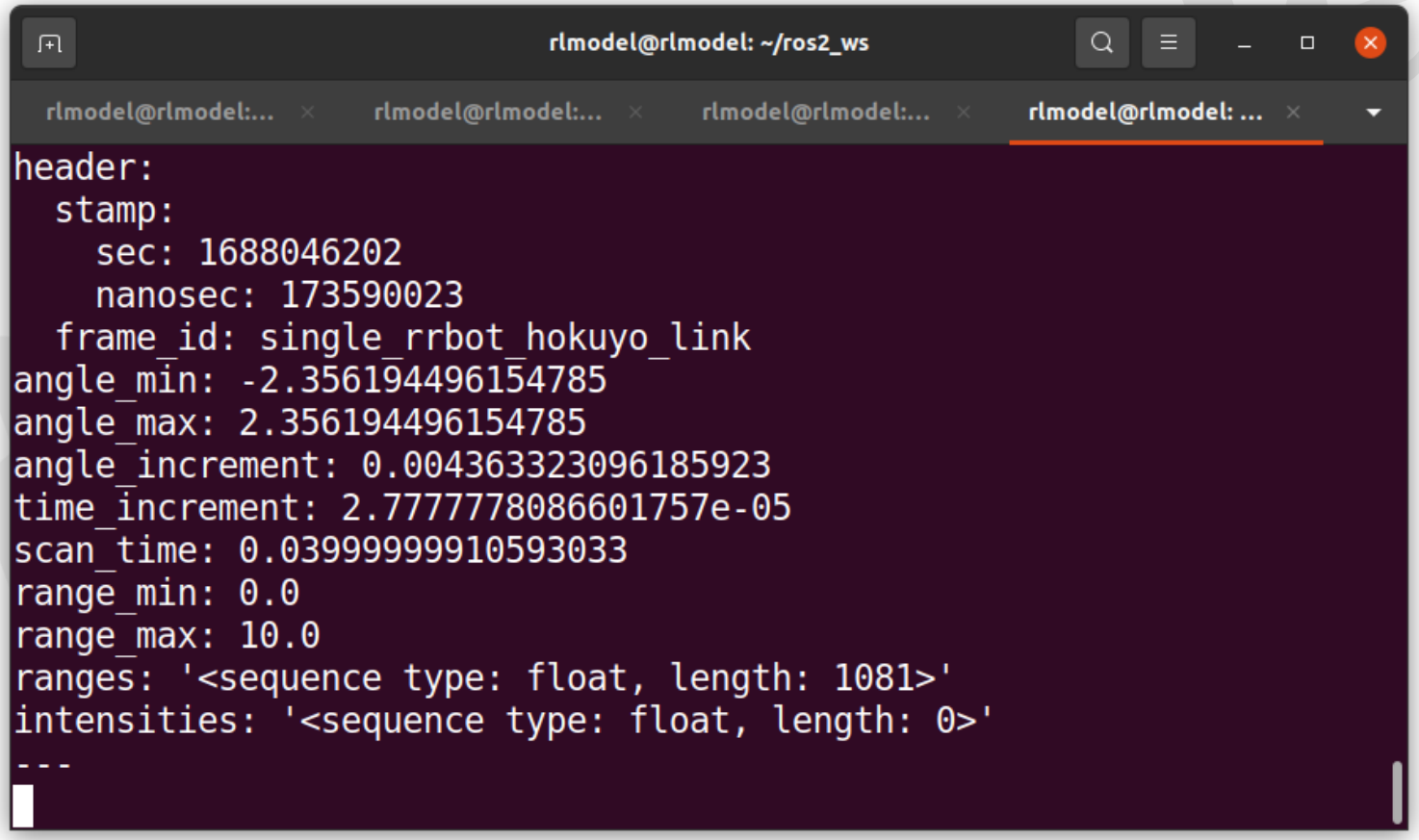


Lidar 2D

scan topic

fram ID check

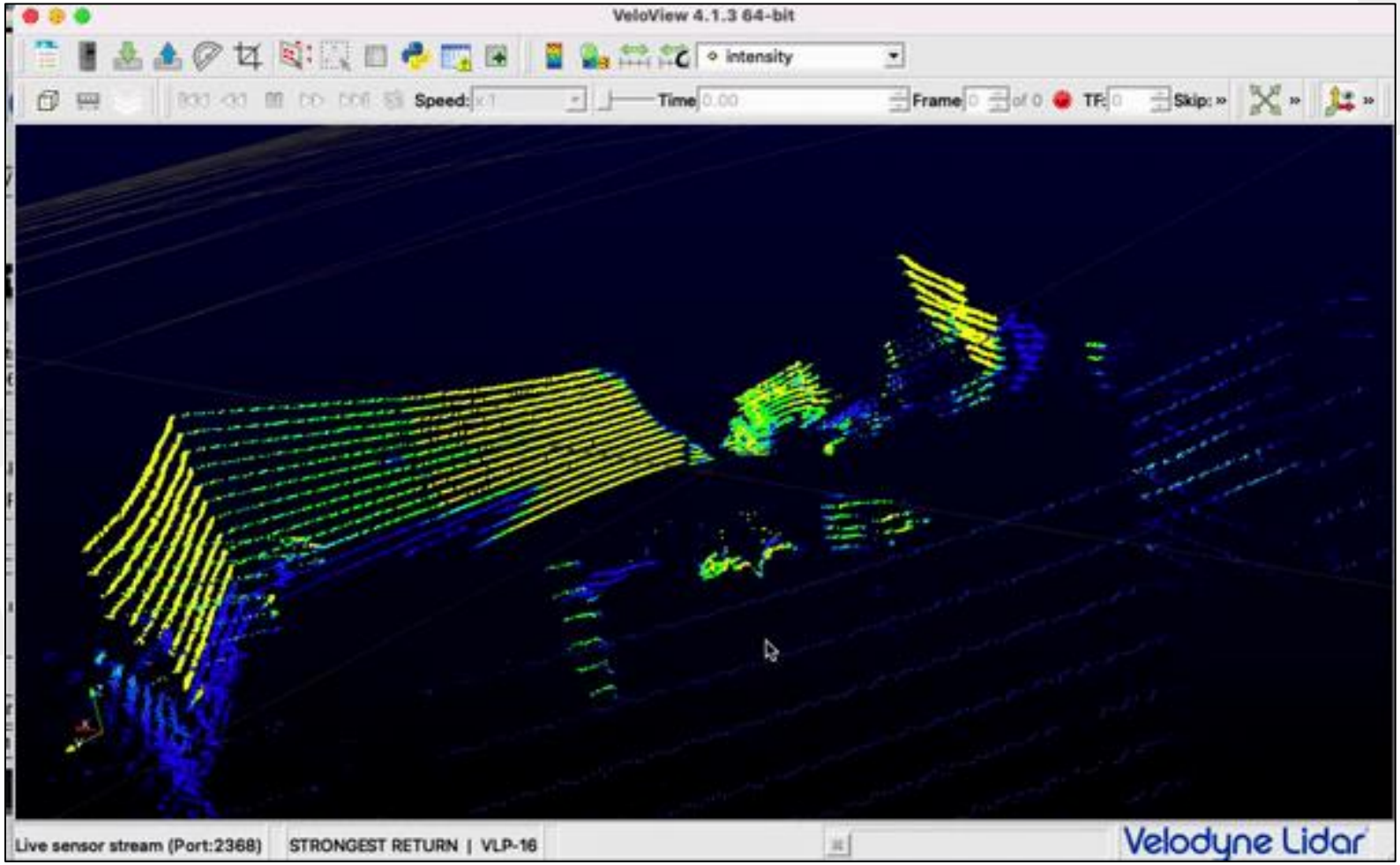
$ ros2 topic echo /scan --no-arr



Lidar 3D

scan topic

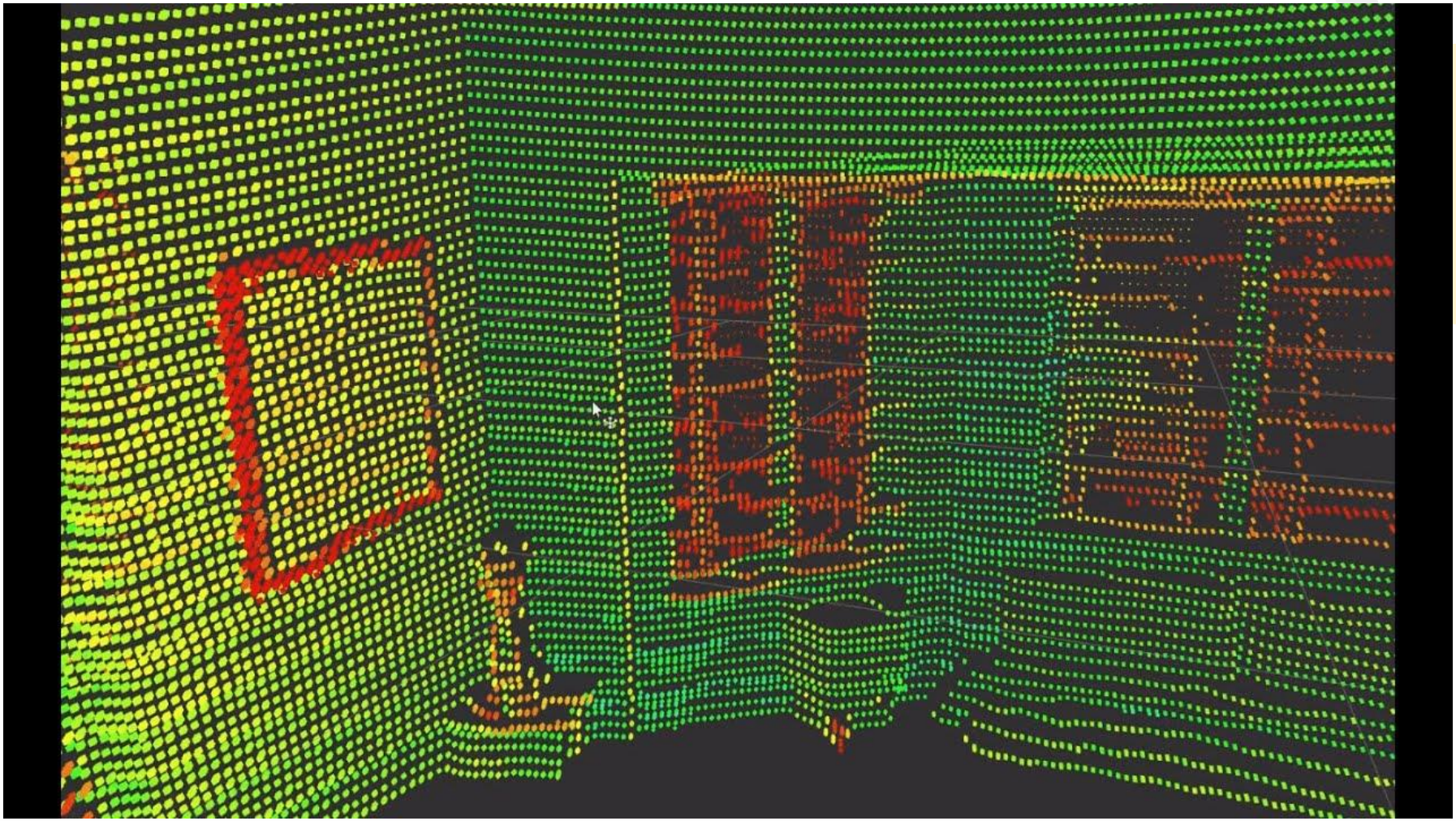
velodyne veloview



Lidar 3D

scan topic

rviz display

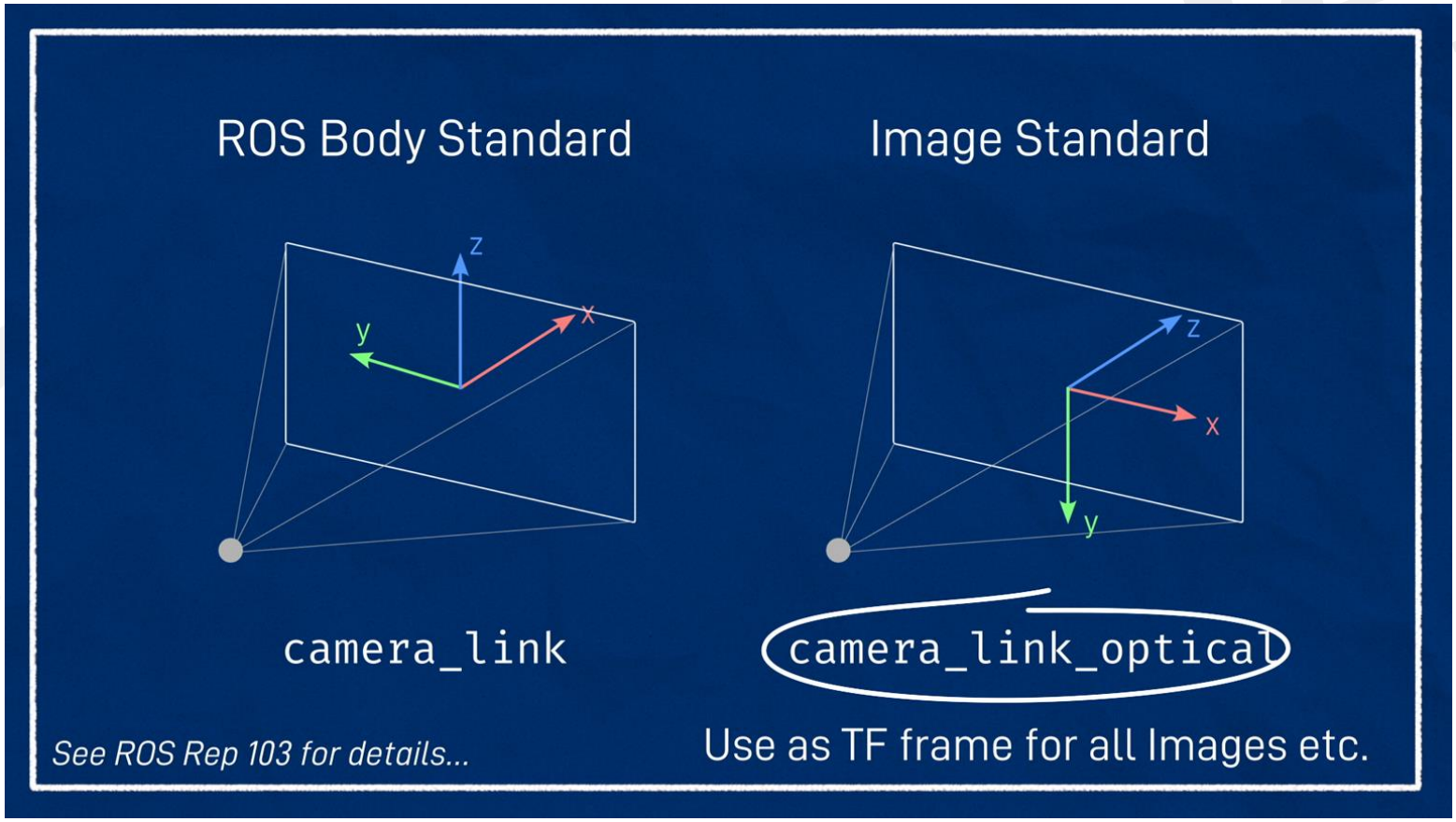


Camera
실습

ROS2 HW

CAMERA

demo - webcam mage

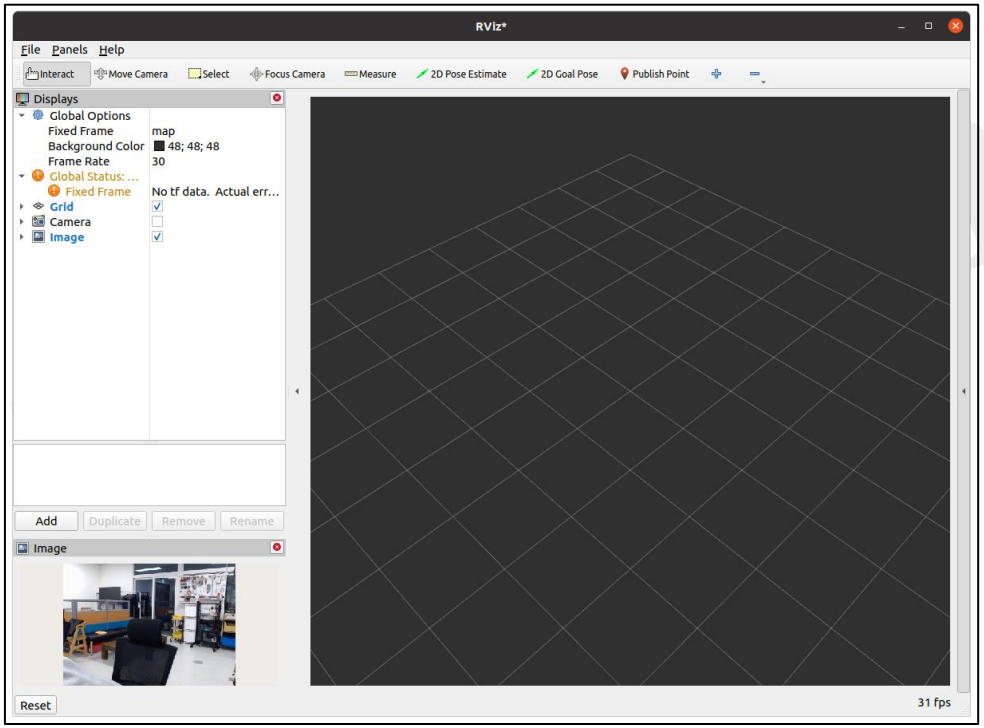


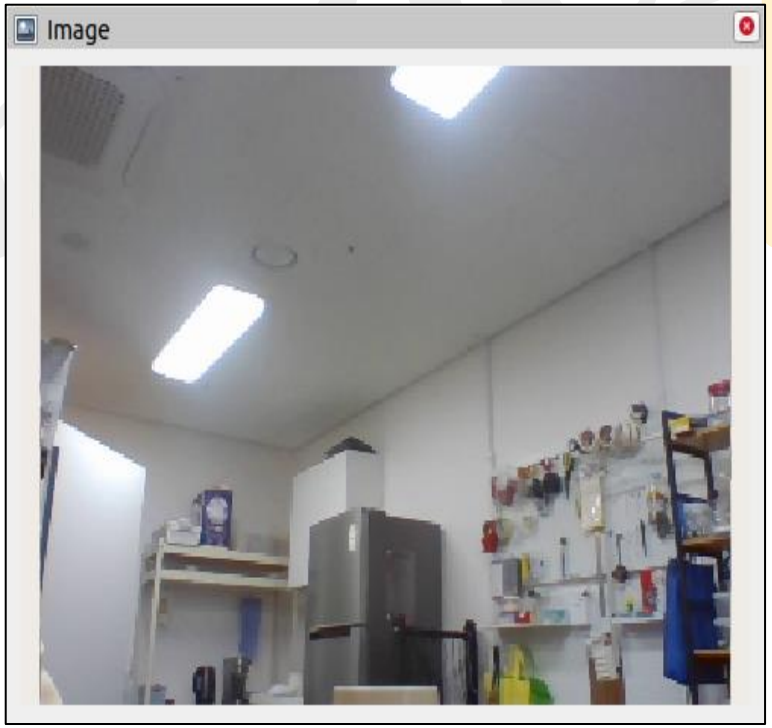
ROS2 HW

CAMERA

명령어: $ ros2 run image_tools cam2image

확인: rviz2 -> add -> By topic -> /image/Image



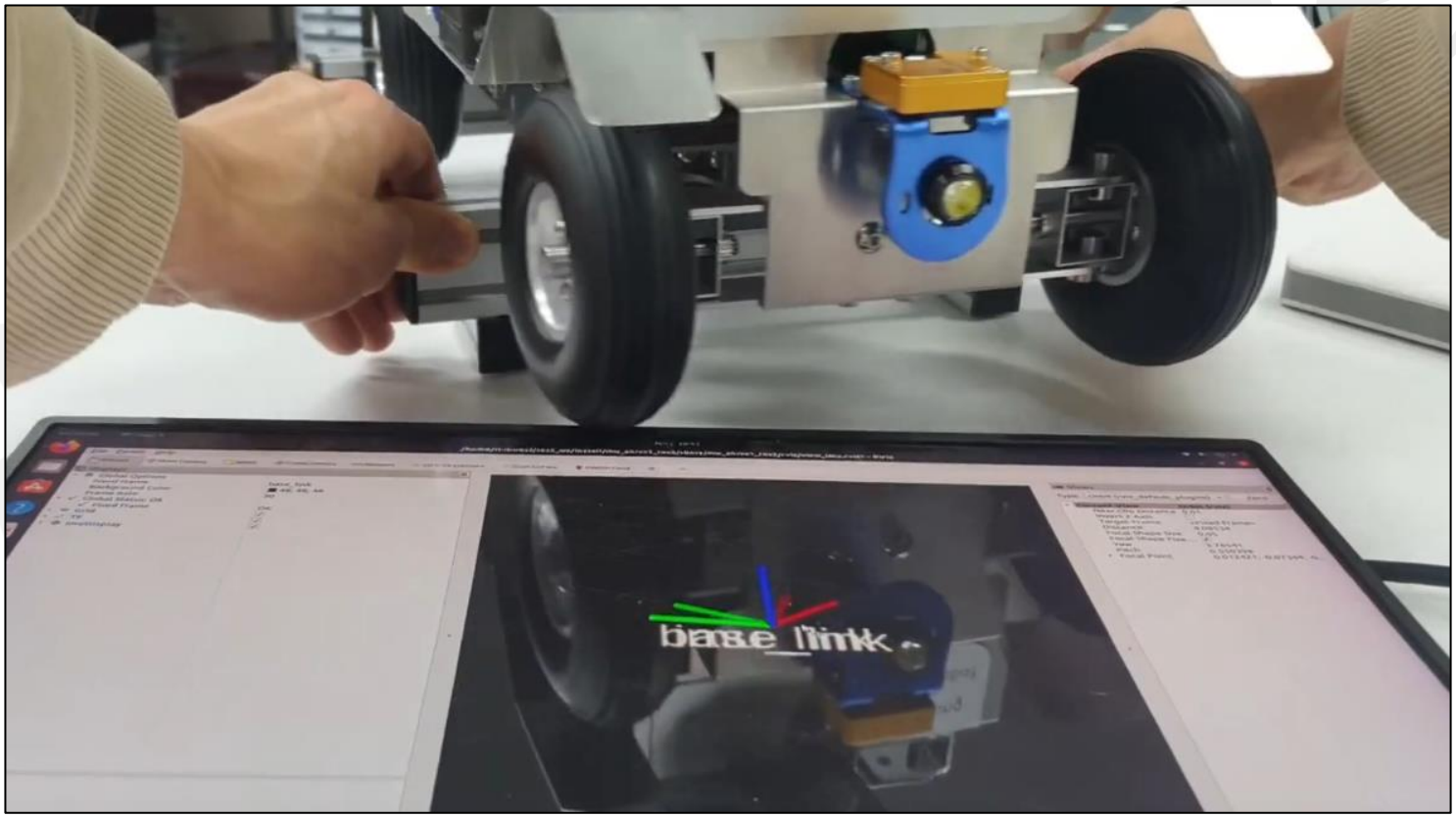


ROS2 HW

IMU

demo: MW-AHRSv1

> *AHRS*(Attitude Heading Reference System)

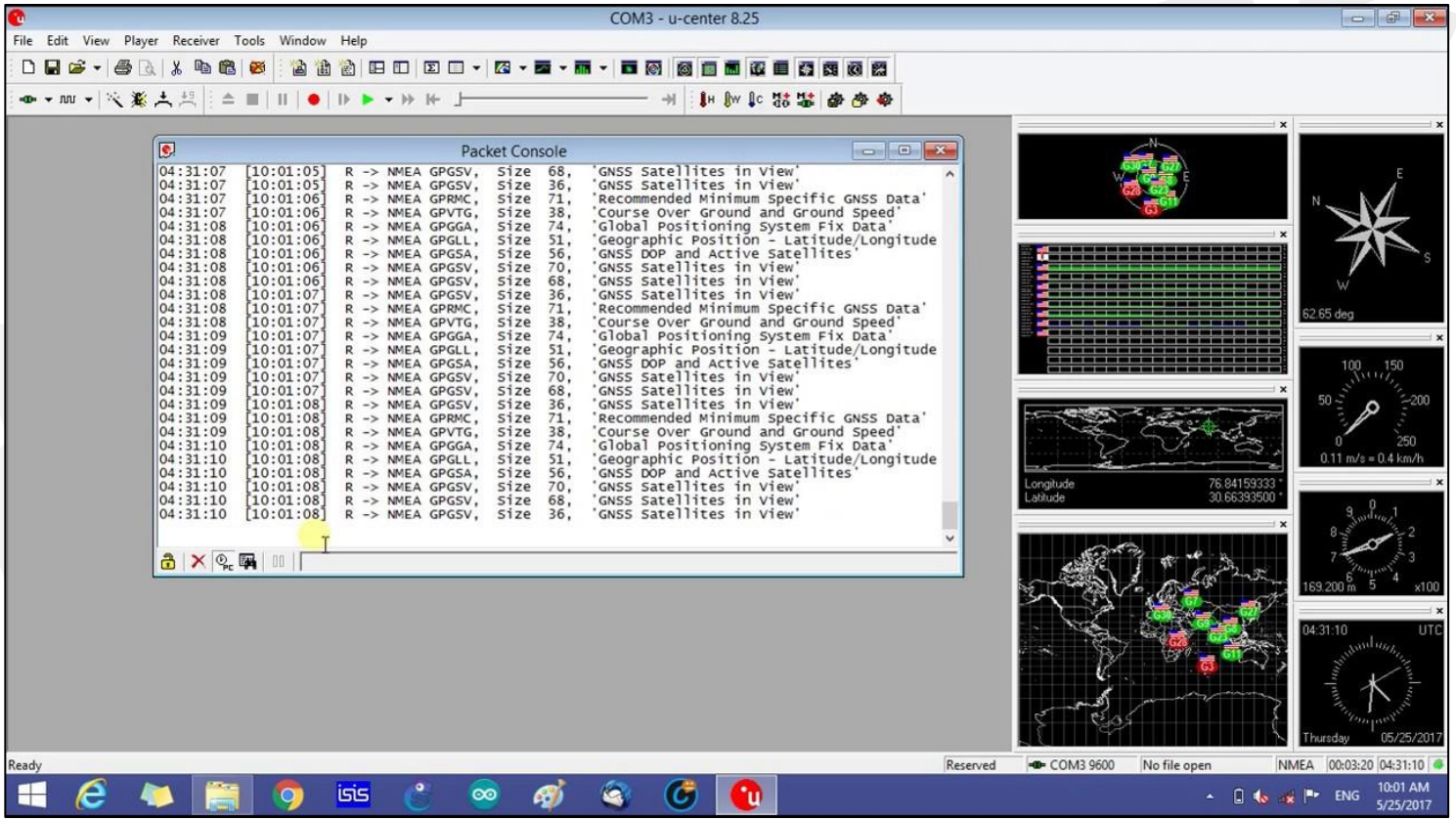


ROS2 HW

GPS

demo - Ublox F9P, RTK, Ntrip client

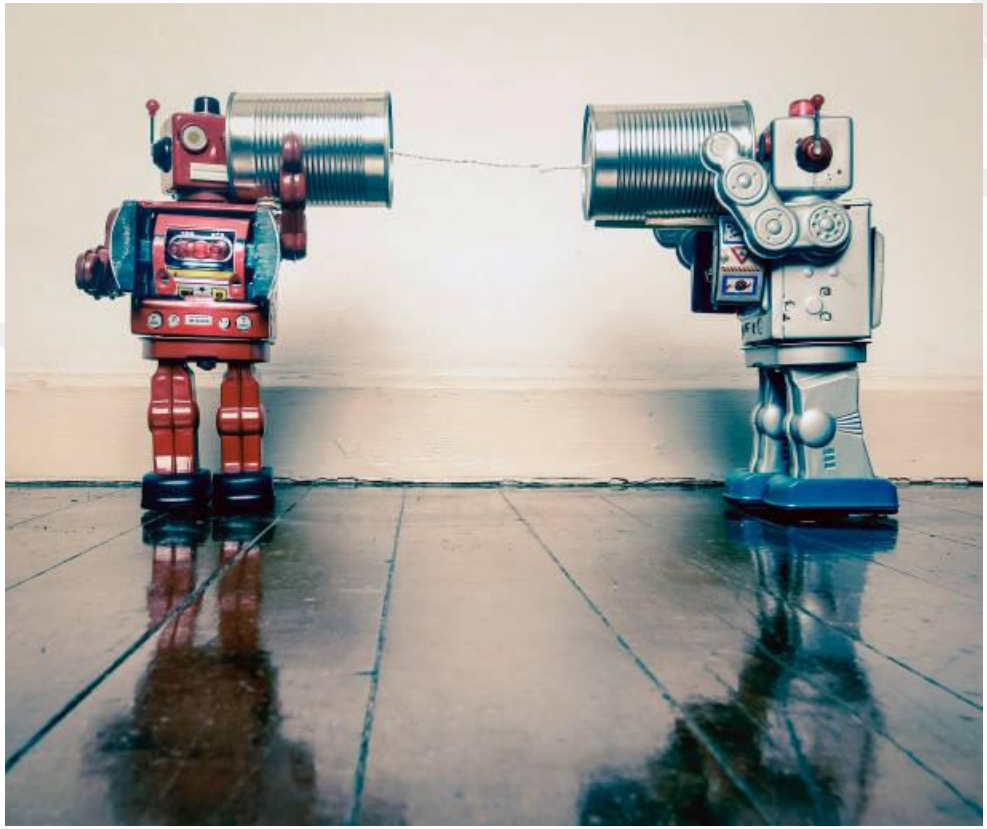
sw: u-center



ROS2 HW

CAN (Controller Area Network)

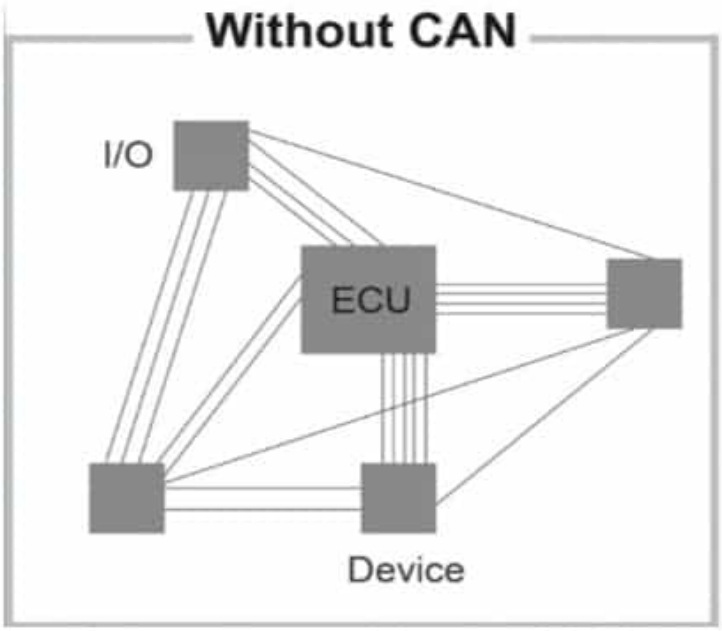
demo - can communication

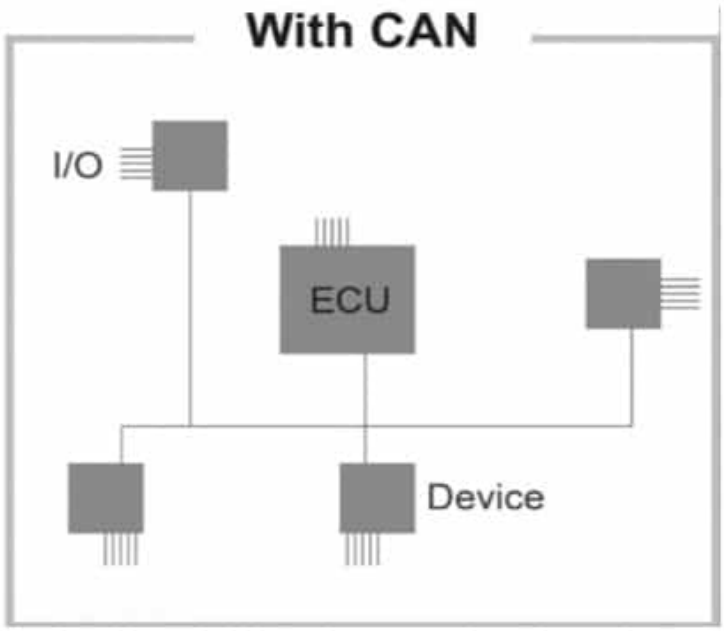


ROS2 HW

CAN

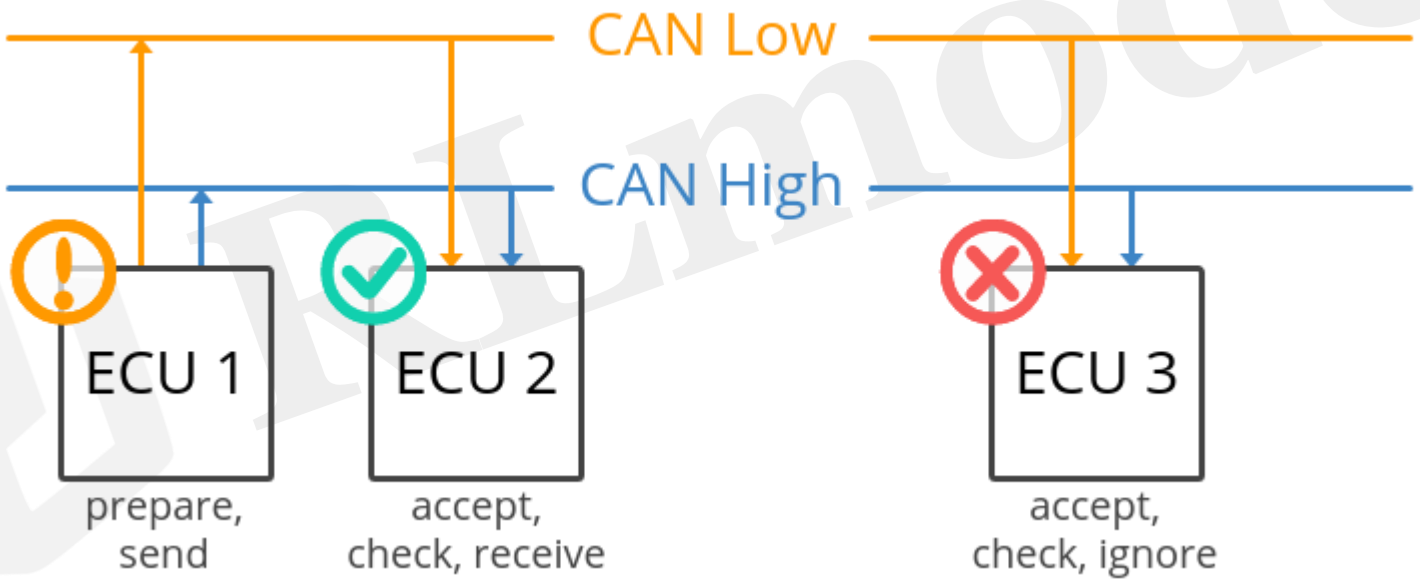
demo - can communication





CAN

demo - can communication



CAN command

CAN device check (USB)

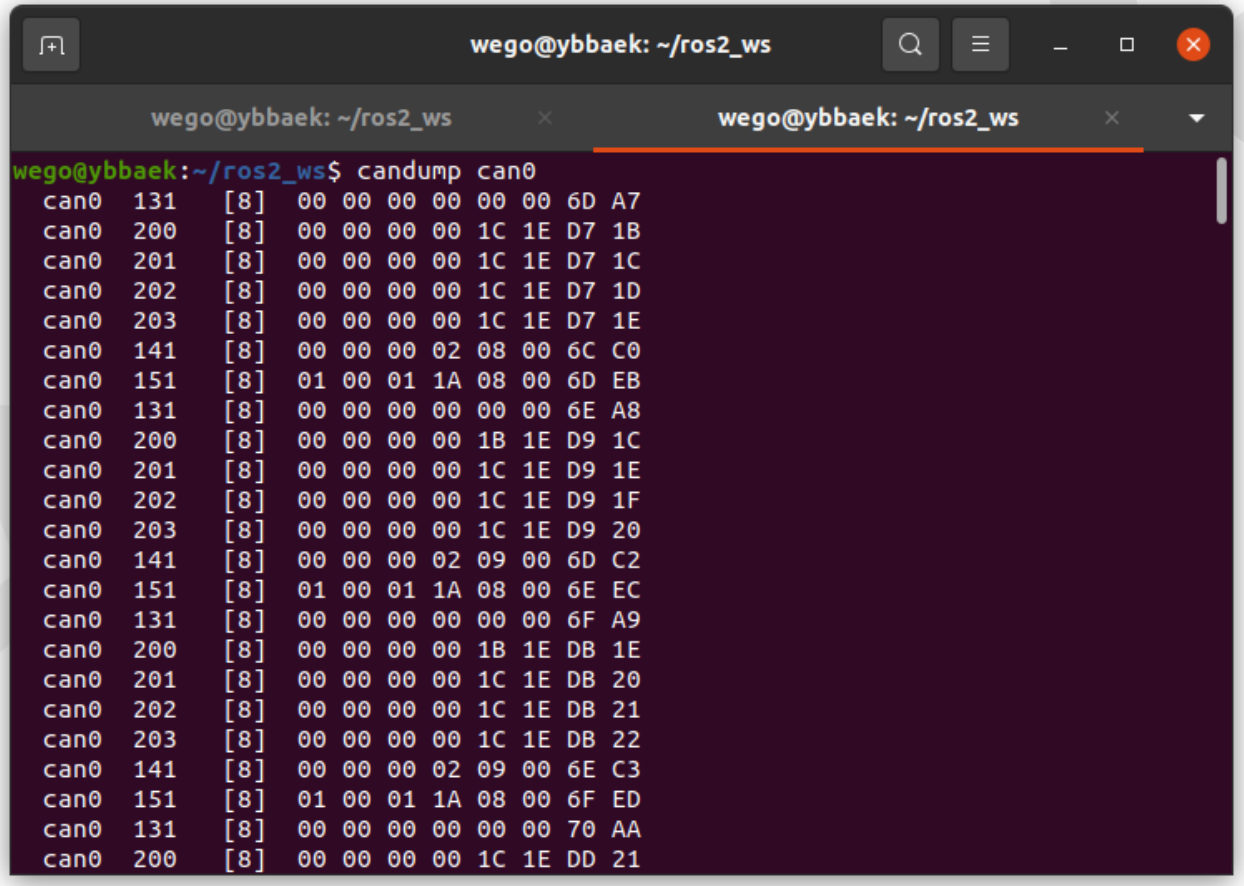
>lsusb

```
rlmodel@rlmodel: ~/rlmodel
rlmodel@rlmodel: ~/rlmodel
rlmodel@rlmodel: ~/rlmodel

rlmodel@rlmodel:~/rlmodel$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 8087:0a2b Intel Corp.
Bus 001 Device 002: ID 046d:c534 Logitech, Inc. Unifying Receiver
Bus 001 Device 011: ID 0c72:000c PEAK System PCAN-USB
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
rlmodel@rlmodel:~/rlmodel$
```

CAN command

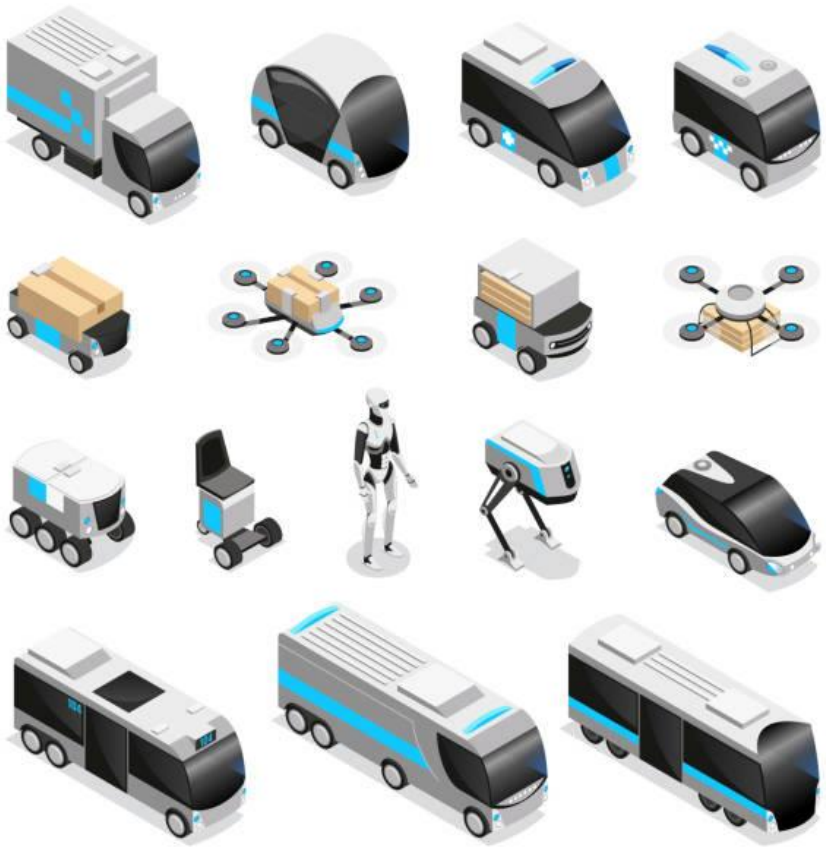
Ubuntu cmd

data check: $ candump cano



Appendix

실제 응용사례



ref

https://ros2-industrial-workshop.readthedocs.io/en/latest/index.html

https://youtu.be/qHr82IPJ-18

yaml 파일

정의

Yaml의 뜻은 'Yaml Ain't Markup Language' 라는 뜻으로 'YAML은 마크업 언어가 아니다'라는 재귀적 이름

```yaml
# This is a comment
title: This is some YAML
publisher: ConvertSimple Books
pages: 250
chapters: 21
time_to_read: 12 hours
descriptors:
  - fun
  - entertaining
  - exciting
contributors:
  author: Mark Templeton
  editor: Cindy Johnson
```

yaml 파일

비교/특징

- 웹에서 데이터 통신을 위해 JSON을 많이 사용하고, reference를 정의할때 복잡한 object 구조를 표현 하기 위해 YAML이 적합

Xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <apiVersion>v1</apiVersion>
  <kind>Pod</kind>
  <metadata>
    <name>hello-pod</name>
    <labels>
      <app>hello</app>
    </labels>
  </metadata>
  <spec>
    <containers>
      <name>hello-container</name>
      <image>tmkube/hello</image>
      <ports>
        <containerPort>8000</containerPort>
      </ports>
    </containers>
  </spec>
</root>
```

Json

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "hello-pod",
    "labels": {
      "app": "hello"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "hello-container",
        "image": "tmkube/hello",
        "ports": [
          {
            "containerPort": 8000
          }
        ]
      }
    ]
  }
}
```

Yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-pod
  labels:
    app: hello
spec:
  containers:
  - name: hello-container
    image: tmkube/hello
    ports:
    - containerPort: 8000
```

yaml 파일

문법/

- yaml 파일의 자료형은 스칼라(Scalar), 시퀀스(Sequence), 맵핑(Mapping)으로 구성
- 스칼라(Scalar) 자료형은 문자열(String)이나 숫자(Number)입니다. 문자열은 큰 따온표("")나 작은 따온(')로 구분하기도 합니다.
- 맵핑(Mapping) 자료형은 해시(Hash) 또는 딕셔너리(Dictionary) 형식입니다. 콜론(:)으로 키(key)와 값(value)를 구분하며 콜론 뒤에는 공백을 반드시 넣어야합니다. 중괄호 { }를 통해서도 작성가능합니다. 노드는 2개의 공백이나 4개의 공백으로 구분됩니다.
- 시퀀스(Sequence) 자료형은 배열(array) 또는 리스트(list) 형식입니다. 대시(-)를 통해 표시하며, 대괄호 [ ] 를 통해서도 작성가능합니다.

Linux 기본 명령어


LINUX
COMMANDS
PASSWD
CAT
NETSTAT
CAT
CHMOD
SUDO
LS
NSLOOKUP
MKDIR
SSH
MV

Ubuntu 기본명령어

- ls - 현재 위치의 파일 목록 조회
- cd - 디렉토리 이동
- touch - 0바이트 파일 생성, 파일의 날짜와 시간을 수정
- mkdir - 디렉토리 생성
- cp - 파일 복사
- mv - 파일 이동
- rm - 파일 삭제
- cat - 파일의 내용을 화면에 출력, 리다이렉션 기호('>')를 사용하여 새로운 파일 생성
- redirection - 화면의 출력 결과를 파일로 저장
- alias - 자주 사용하는 명령어들을 별명으로 정의하여 쉽게 사용할 수 있도록 설정

Ubuntu 기본명령어

1. ls (List segments) : 현재 위치의 파일 목록 조회

- ls -l : 파일의 상세정보
- ls -a : 숨김 파일 표시
- ls -t : 파일들을 생성시간순(제일 최신 것부터)으로 표시
- ls -rt : 파일들을 생성시간순(제일 오래된 것부터)으로 표시
- ls -f : 파일 표시 시 마지막 유형에 나타내는 파일명을 끝에 표시
  ('/' : 디렉터리, '*' : 실행파일, '@' : 링크 등등,,,)

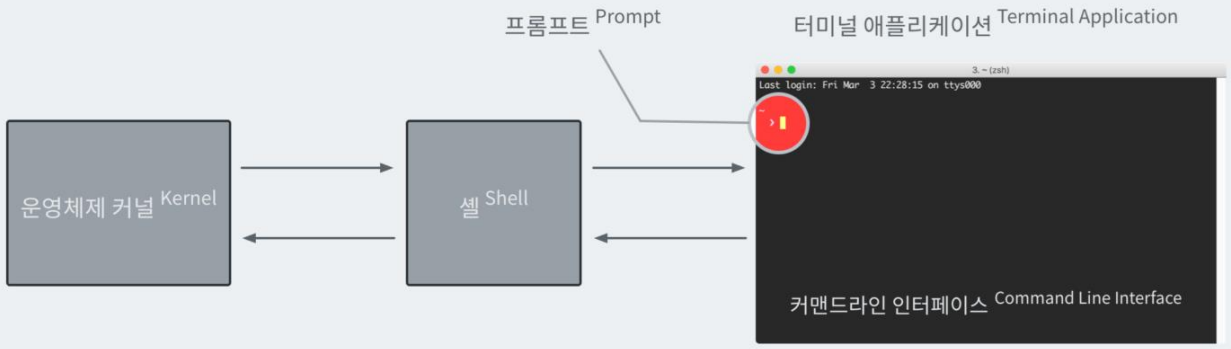
2. cd (Change directory) :디렉터리 이동

- cd [디렉터리 경로] : 이동하려는 디렉터리로 이동 (경로 입력 시 '[', ']'부분은 빼고 입력!)
- cd ~ : 홈 디렉터리로 이동
- cd / : 최상위 디렉터리로 이동
- cd . : 현재 디렉터리
- cd .. : 상위 디렉터리로 이동
- cd - : 이전 경로로 이동

터미널(Terminal)

리눅스 터미널 프로그램

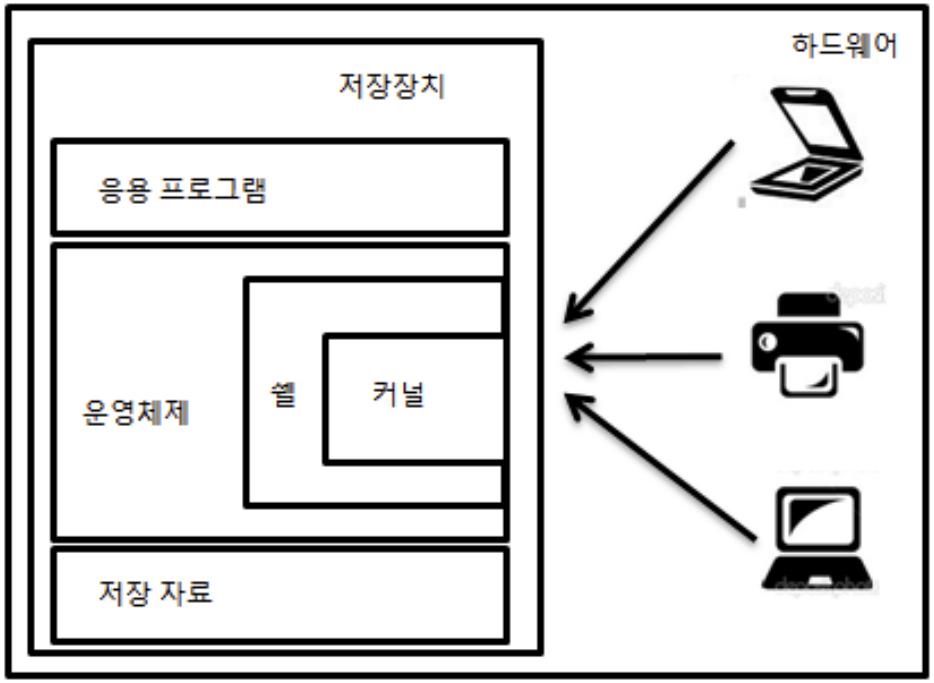
- 터미널은 GUI에서 실행되는 이전 물리적 터미널의 소프트웨어 표현이다.
- 사용자가 명령을 입력하고 텍스트를 인쇄할 수 있는 인터페이스를 제공한다.
- Linux 서버에 SSH로 연결할 때 로컬 컴퓨터에서 실행하고 명령을 입력하는 프로그램은 터미널이다.



셸(shell)이란

1. 기본기능

- 셸(shell)은 터미널 창에 입력한 명령을 해석하여 운영 체제에서 사용자가 원하는 작업을 이해할 수 있도록 하는 프로그램

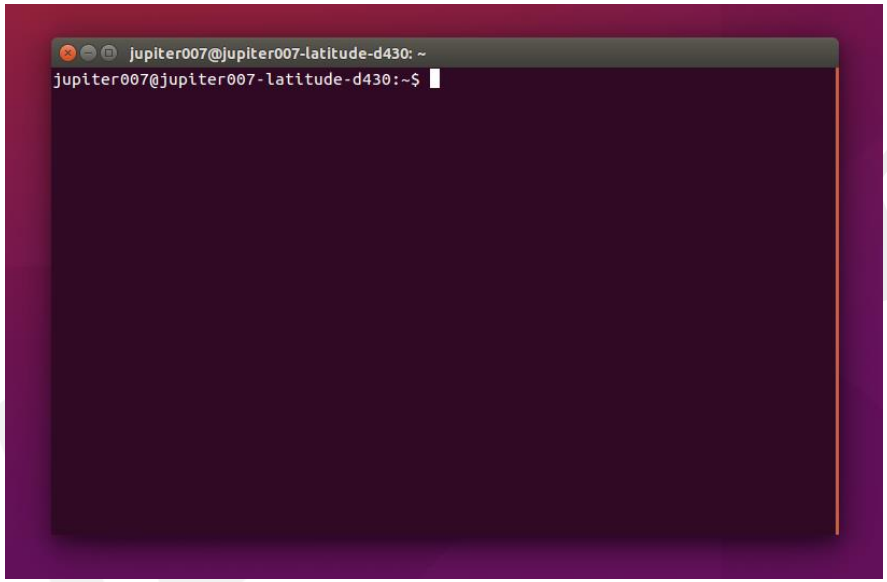


리눅스

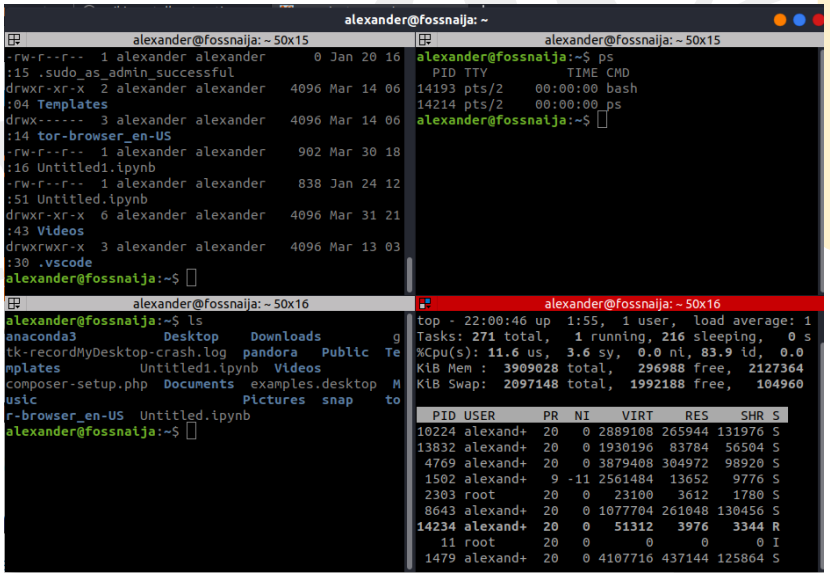
터미널(Terminal)

기본/Terminator

실행: Alt + Ctrl + T



기본 터미널



Terminator

터미널(Terminal)

상세설정

셸(shell)이란

1. 상세기능

- 사용자와 커널 사이에서 명령을 해석해 전달 프로그램
- 쉘은 자체 내에 프로그래밍 기능이 있어서 프로그램을 작성할 수 있음. 따라서 쉘 프로그래밍 기능을 이용하면 여러 명령을 사용해 반복적으로 수행하는 작업을 하나의 프로그램으로 제작 할 수 있습니다. 쉘 프로그래밍을 쉘 스크립트라고 함
- 사용자 환경 설정의 기능 - 초기화 파일 기능을 이용해 사용자의 환경을 설정할 수 있습니다.

bash

설명: 현재 리눅스의 표준 셸

- ○ 우분투(Ubuntu) 기본 셸 > bash
- ◯ bash 는 리눅스 뿐만 아니라 GNU 운영체제, 맥 OS 등 다양한 운영체제에서 사용 중입니다.

bash 의 특징

- Alias 기능 (명령어 단축 기능)
- History 기능 (키보드 화살표)
- 연산기능
- Job Control 기능
- 자동 이름완성 기능 (tab)
- 프롬프트 제어 기능
- 명령어 편집기능 등

CLI 란 (Command Line Interface)

1. 개념

커맨드라인 인터페이스(Command-line Interface, CLI)란 사용자가 텍스트로 명령어를 입력하고 다시 텍스트로 결과를 화면에 출력해주는 인터페이스를 가진 컴퓨팅 인터페이스를 의미

오로지 문자열로만 이루어진 인터페이스를 의미

bash 셸 환경변수

변수	내용	변수	내용
HOME	현재 사용자의 홈 디렉터리	PATH	실행 파일을 찾는 디렉터리 경로
LANG	기본 지원되는 언어	PWD	사용자의 현재 작업 디렉터리
TERM	로그인 터미널 타입	SHELL	사용자의 로그인 셸
USER	사용자의 이름	DISPLAY	X윈도에서 프로그램 실행 시 출력되는 창
COLUMNS	현재 터미널의 컬럼 수	LINES	현재 터미널 라인 수
PS1	1차 명령 프롬프트 변수	PS2	2차 프롬프트 변수
BASH	bash 셸의 경로	BASH_VERSION	bash 버전
HISTFILE	히스토리 파일의 절대 경로	HISTSIZE	히스토리 파일에 저장되는 명령어(줄)의 개수
HISTFILESIZE	히스토리 파일의 크기	HOSTNAME	시스템의 호스트명
MAIL	도착한 메일이 저장되는 경로	LOGNAME	로그인 이름
TMOUT	사용자가 로그인 한 후 일정 시간 동안 작업을 하지 않을 경우에 로그아웃시키는 시간으로 단위는 초		
UID	사용자의 UID	OSTYPE	운영체제 타입

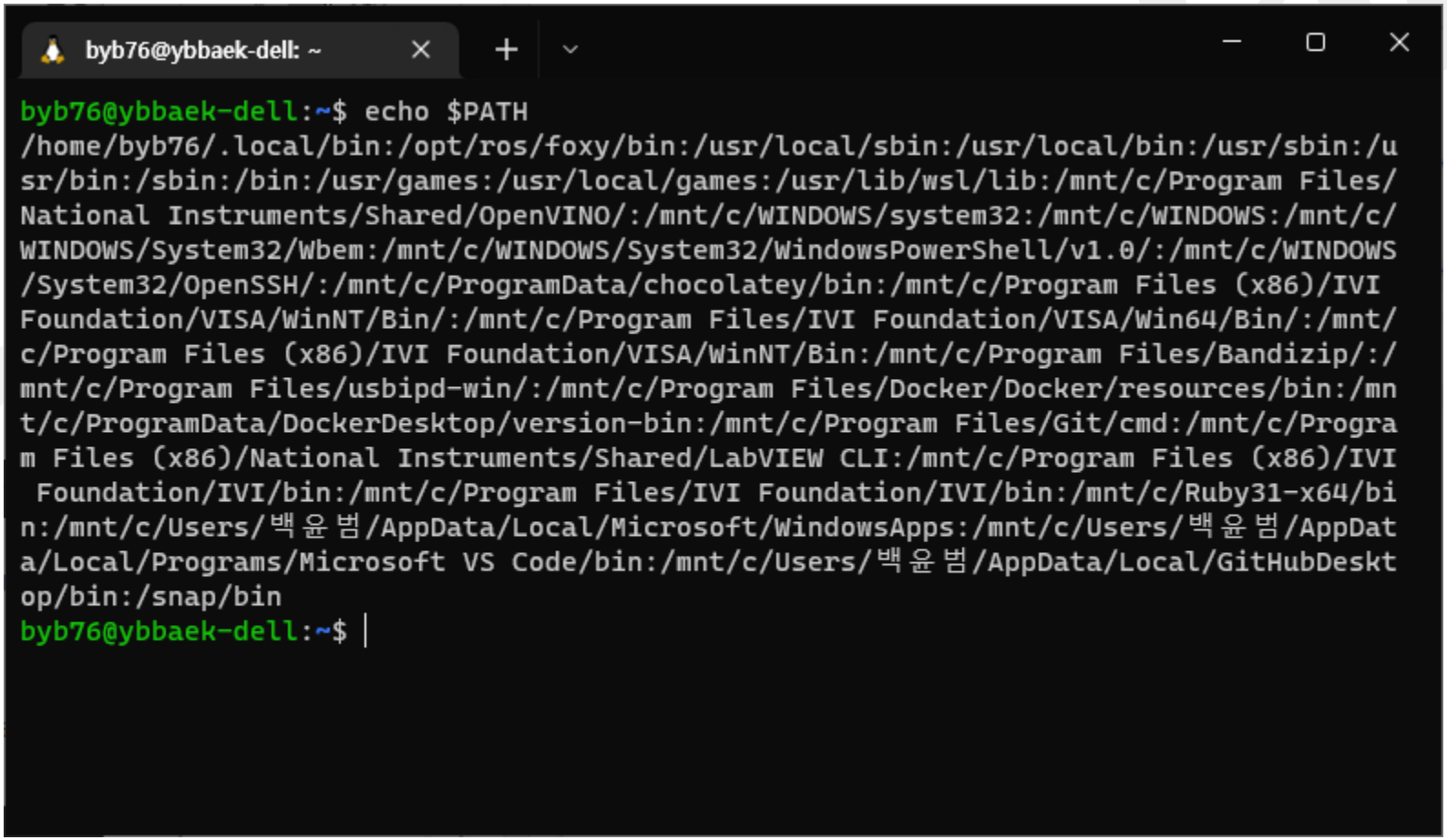
bash 셸 환경변수

/etc/profile
시스템 전역 쉘 변수
시스템 사용자 전체에 적용할 쉘 환경 변수를 담고 있으며 보통

경로 환경변수(PATH),
사용자 환경변수(USER),
로그인 사용자(USERNAME),
사용자 메일박스(MAIL),
호스트 이름(HOSTNAME),
히스토리(HISTORY)

bash 셸 환경변수

- path



리눅스

셸 확인방법

- echo $SHELL

```
byb76@ybbaek-dell:~/ros2_foxy$ echo $SHELL
/bin/bash
byb76@ybbaek-dell:~/ros2_foxy$
```

BASH

THE BOURNE-AGAIN SHELL

bashrc

- Bash Bourn Again Shell widely used shell

  >Next 5 configuration file

  /etc/profile
  /etc/bashrc
  ~/.bash_profile
  ~/.bashrc
  ~/.bash_logout

  Global configuration: /etc/profile, etc/bashrc
  Local configuration: ~/.bashrc, ~/.bash_profile

  rc mean: run command, run configuration

bashrc

- .bashrc
- 일반적으로 전역적 파일은 /etc 디렉토리에 위치
  ~/.bashrc > 별칭(alias) 과 bash가 수행될 때 실행되는 함수를 제어하는 지역적인 시스템 설정과 관련된 파일이다.
  숨김파일은 .으로 시작

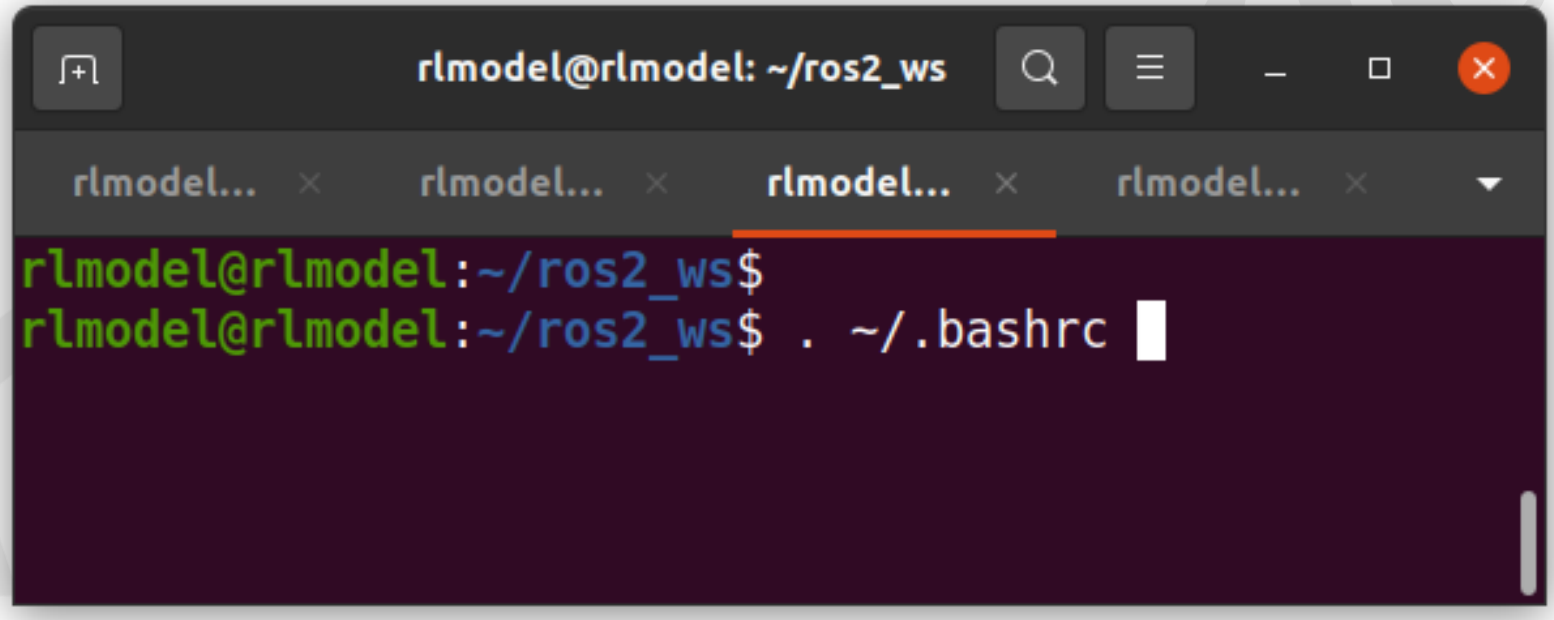
  /etc/profile
  >은 환경변수와 bash 가 수행될 때 실행되는 프로그램을 제어하는 전역적인 시스템 설정과 관련된 파일

리눅스

bashrc

.bashrc

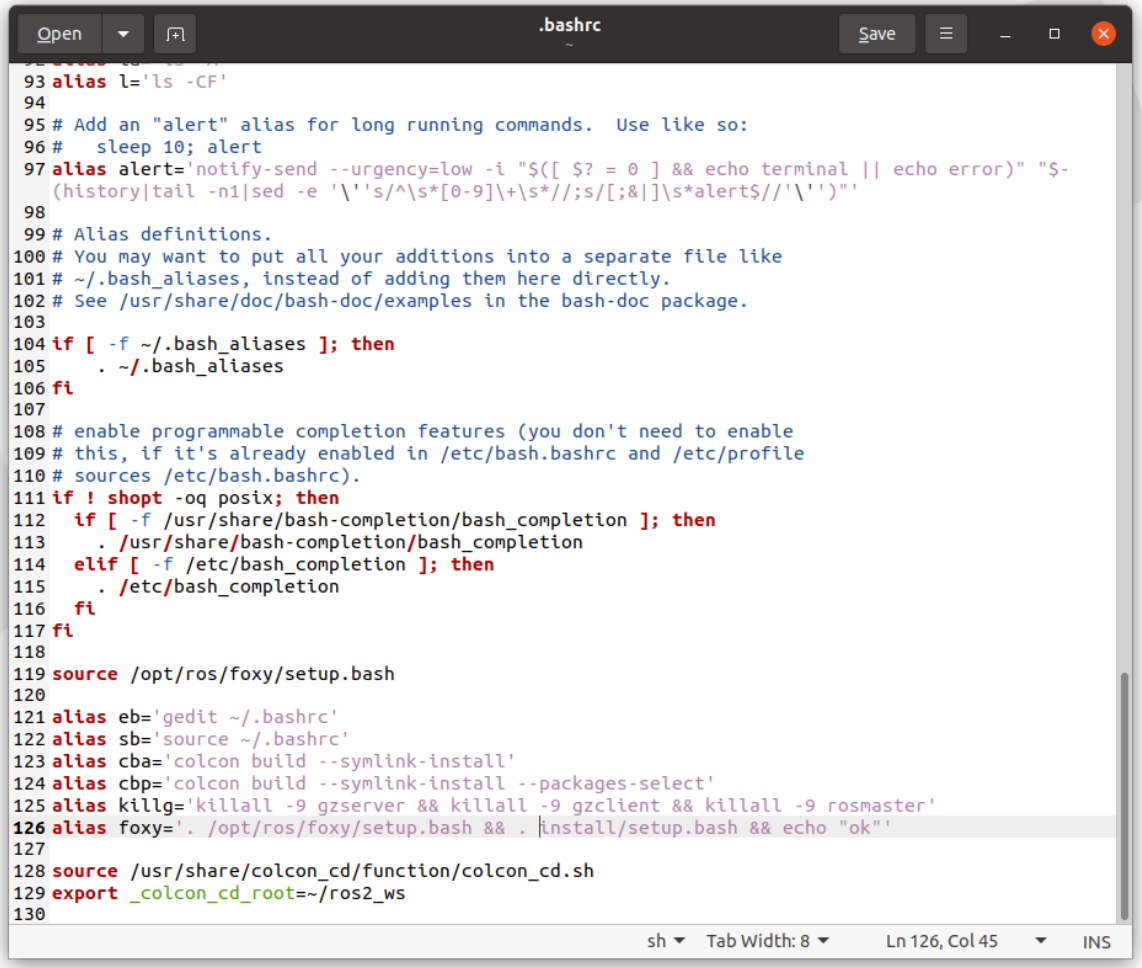
>Run: . ~/.bashrc



리눅스

bashrc

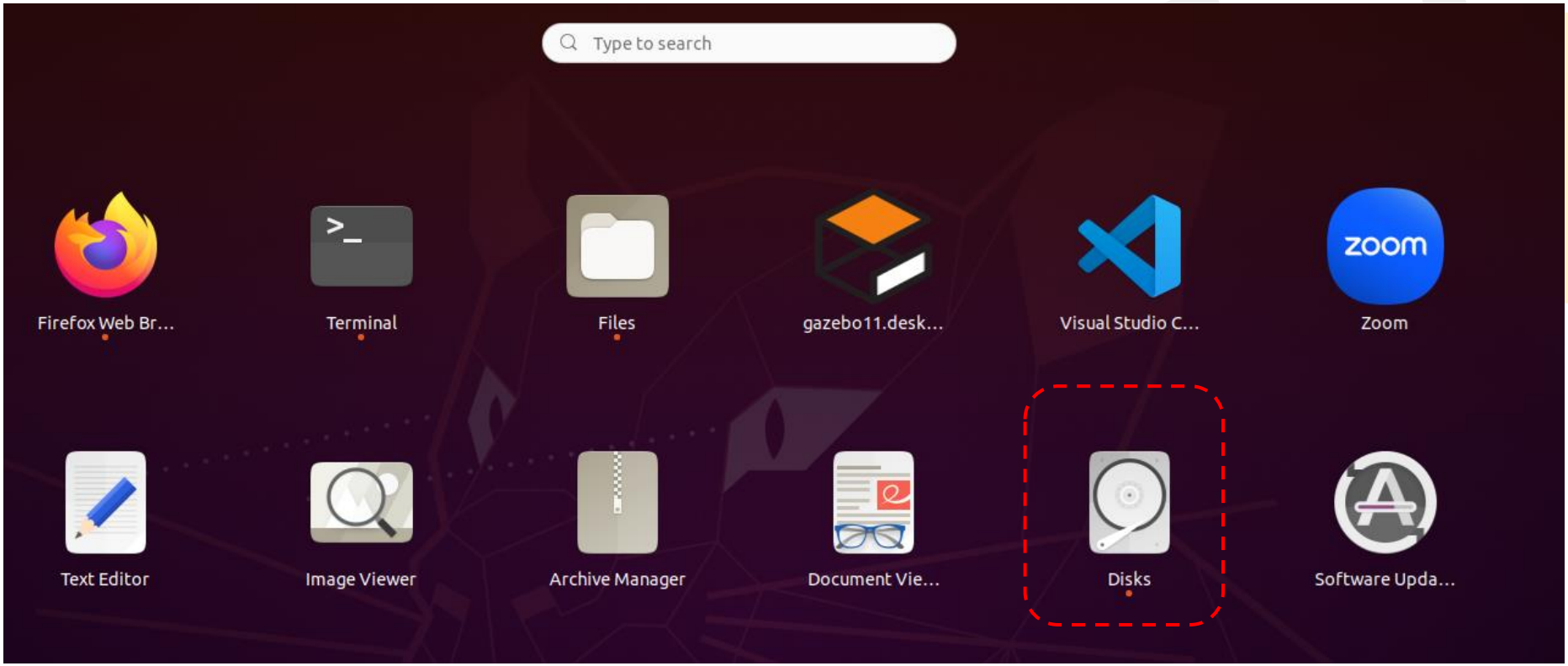
> alias: short key



리눅스

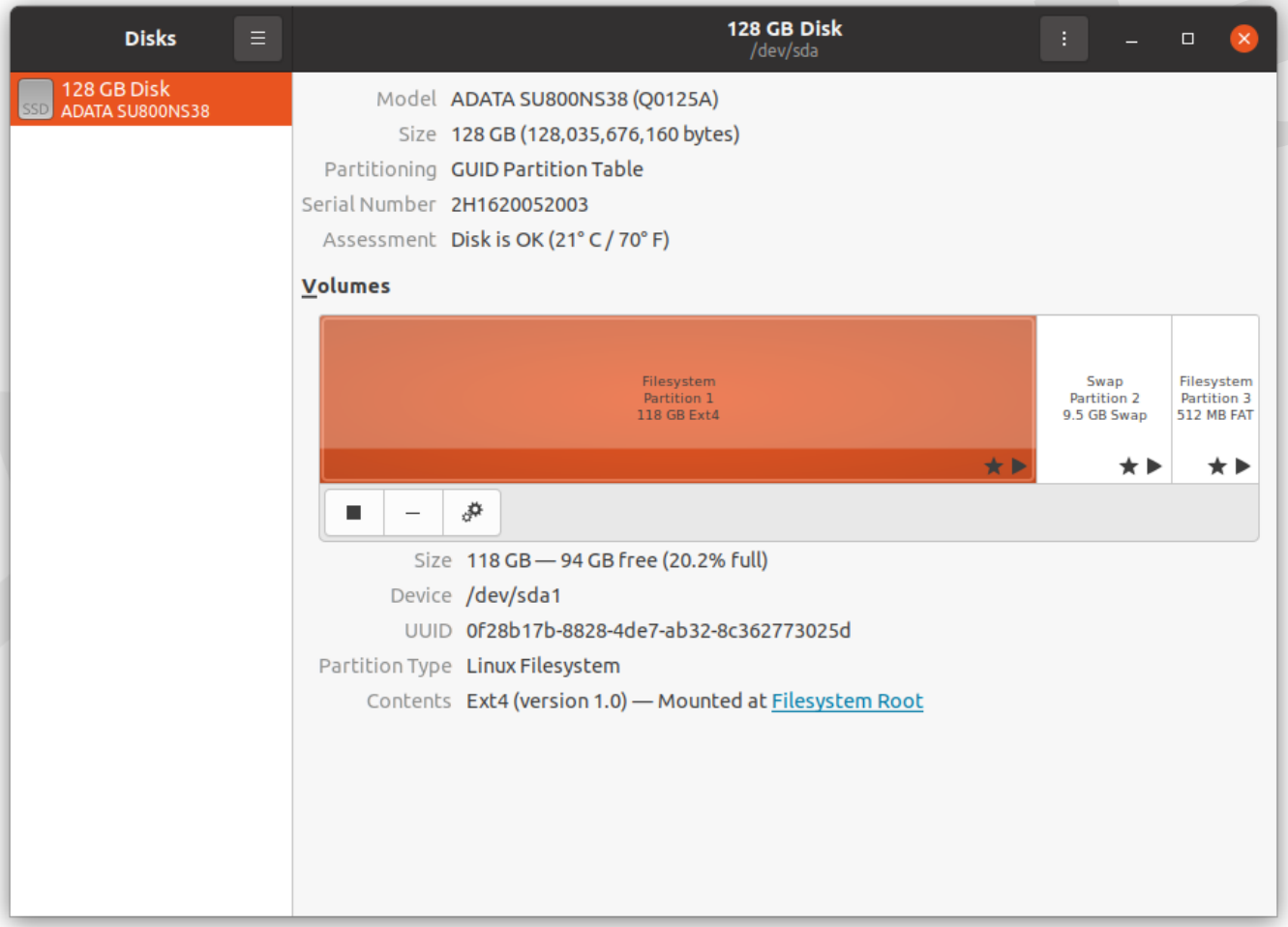
HDD disk usage

> DISK



HDD disk usage

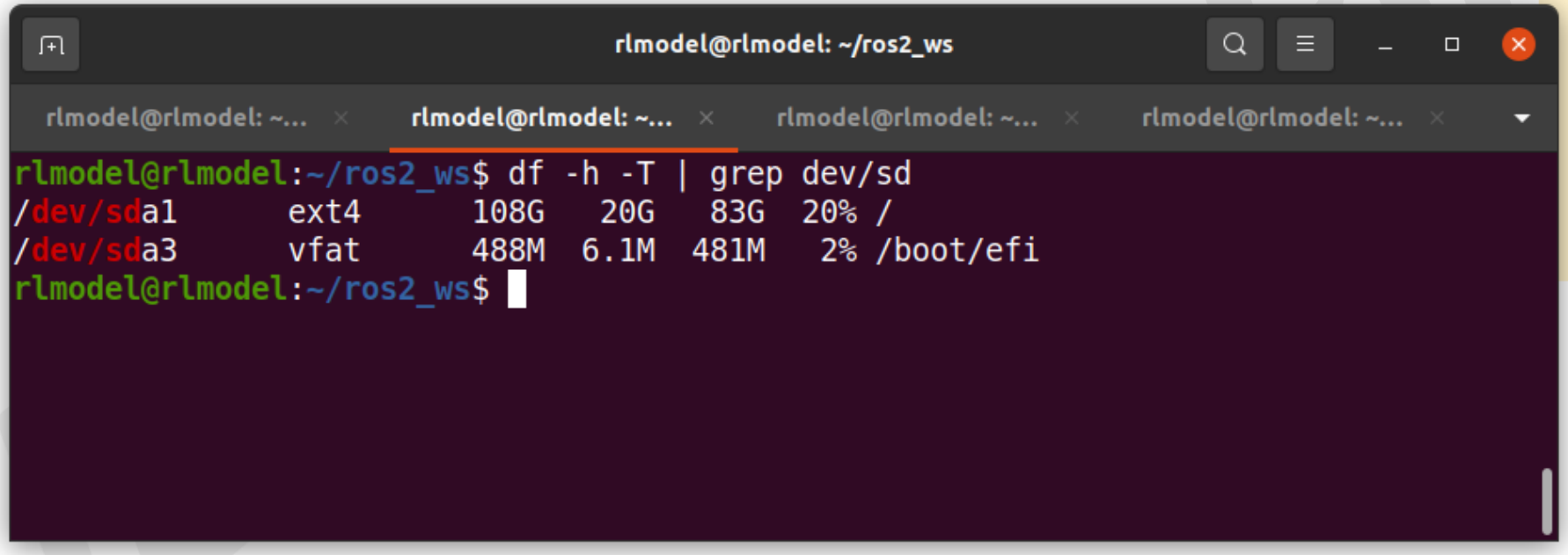
> DISK



리눅스

HDD disk usage

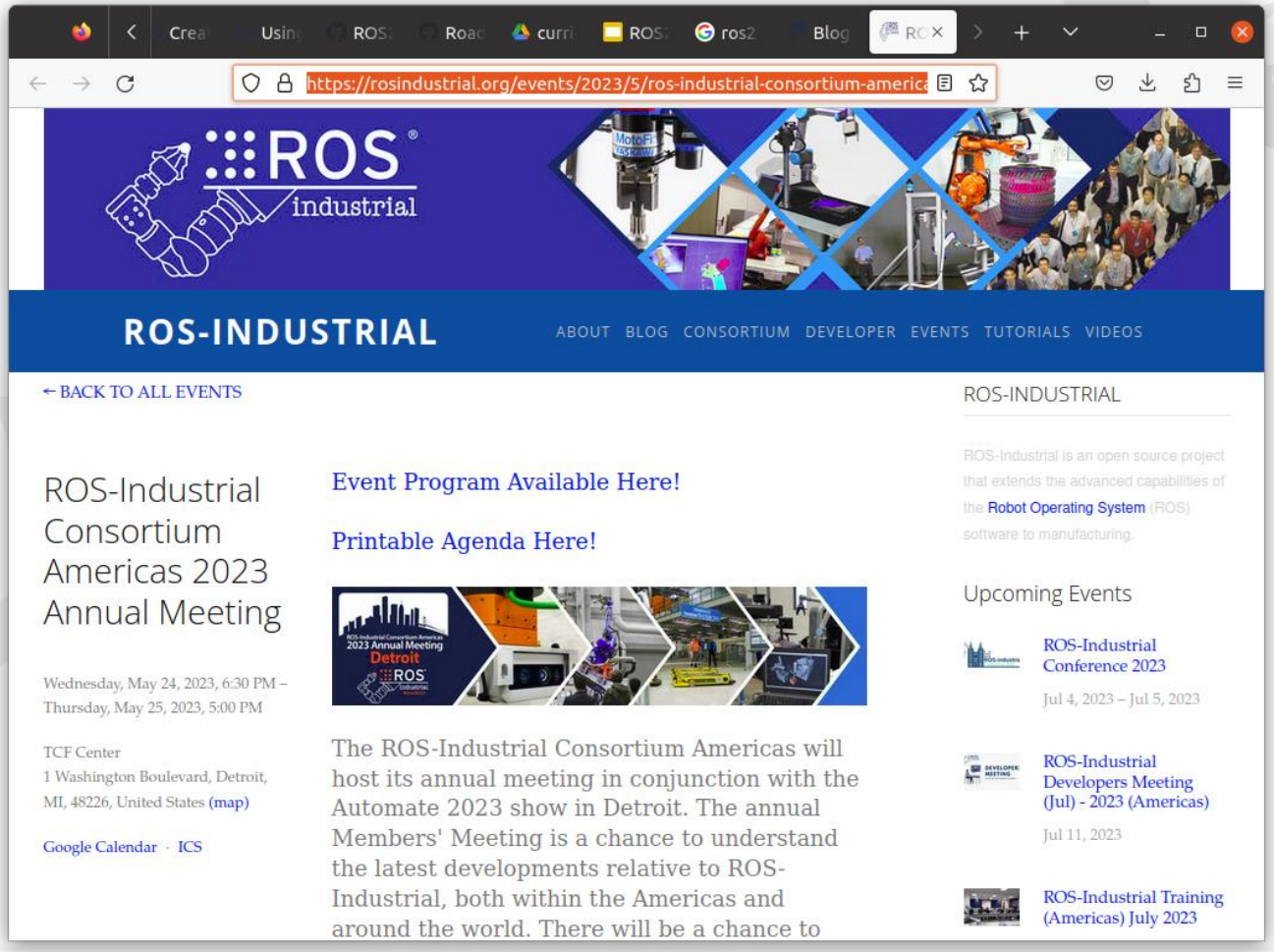
>DISK: df -h -T | grep dev/sd



ROS2 info

ROS2 Industrial

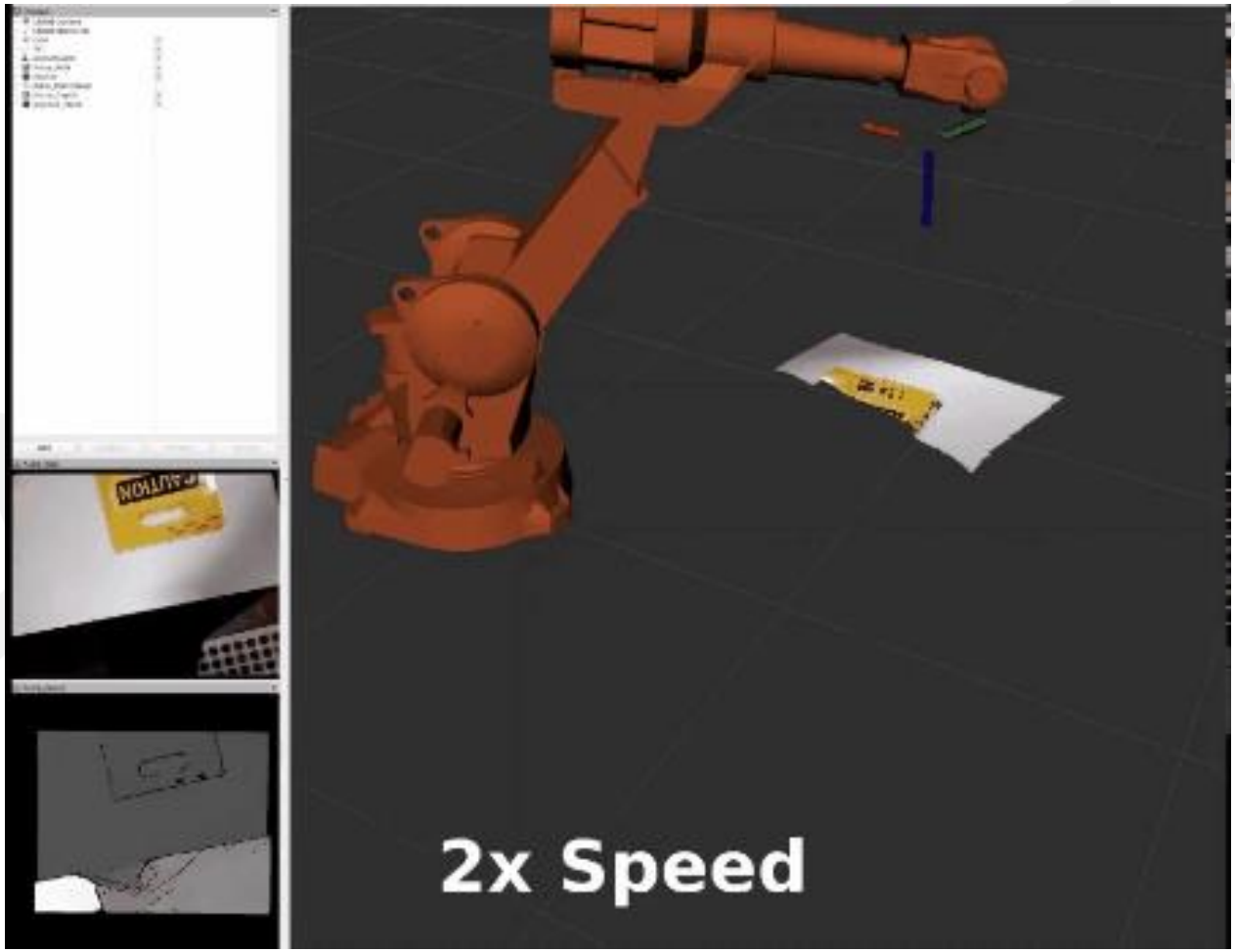
https://rosindustrial.org/events/2023/5/ros-industrial-consortium-americas-2023-annual-meeting



ROS2 info

ROS2 Industrial

https://rosindustrial.org/events/2023/5/ros-industrial-consortium-americas-2023-annual-meeting



DDS 특징

RTPS

1. RTPS(Real Time Publish Subscribe)란

DDS의 표준 wire-protocol로 선정.

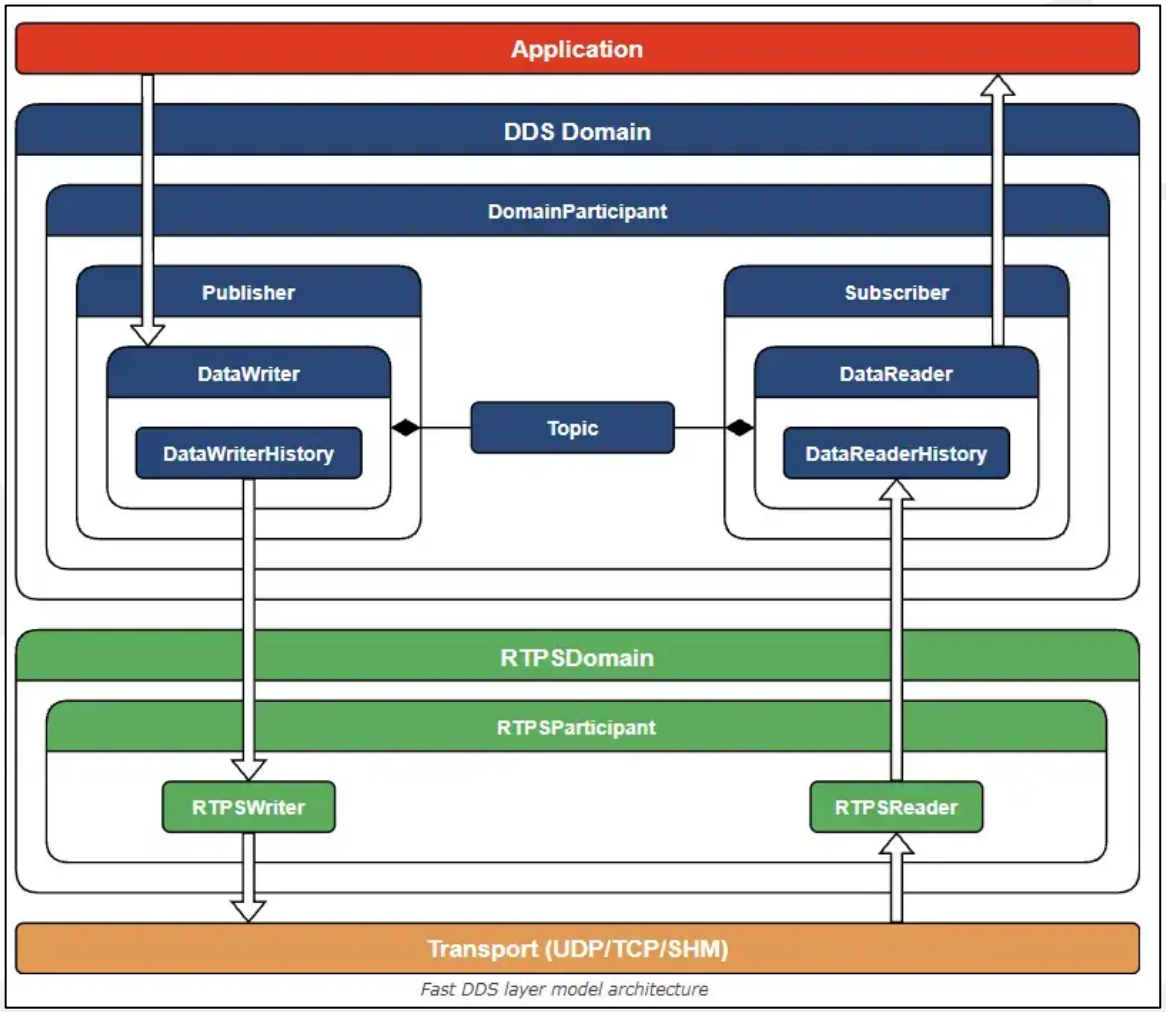
산업자동화 protocol. (산업용 디바이스에 많이 사용되고 있는 검증된 방식)

QoS 설정: best-effort와 reliable publish subscribe 통신 선택 가능.
Fault tolerance: single points failure에도 네트워크 구성 가능.
Extensibility: 프로토콜을 새로운 서비스로 확장시킬 수 있음.
Plug-and-play: 새로운 application이나 service가 자동으로 discover되고 연결 됨.
Scalability: 매우 큰 네트워크로 시스템을 확장할 수 있음.
Type-safety: programming 에러가 remote 노드들을 손상시키지 못하게 함.

DDS 특징

RTPS 구조 (Realtime Publisher Subscribe)

Block Diagram

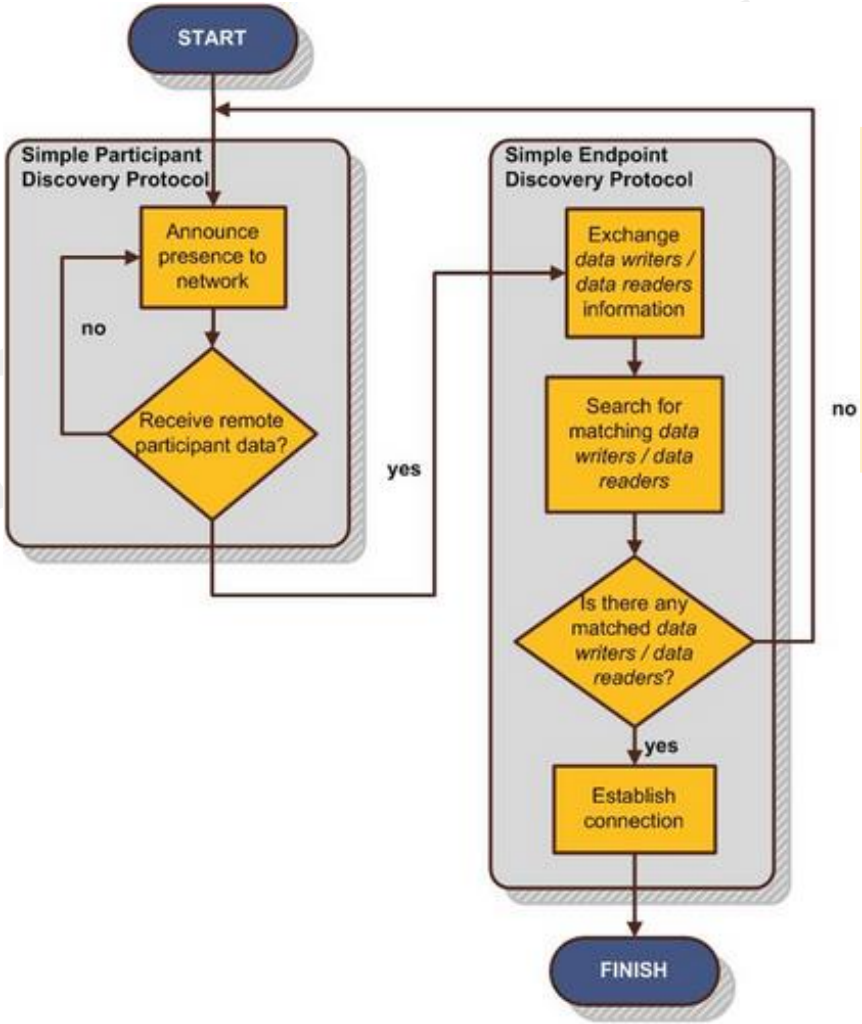


Fast DDS layer model architecture

DDS 특징

PDP, EDP

- Auto discovery
- PDP (Participant Discovery Protocol)
- EDP (Endpoint Discovery Protocol)



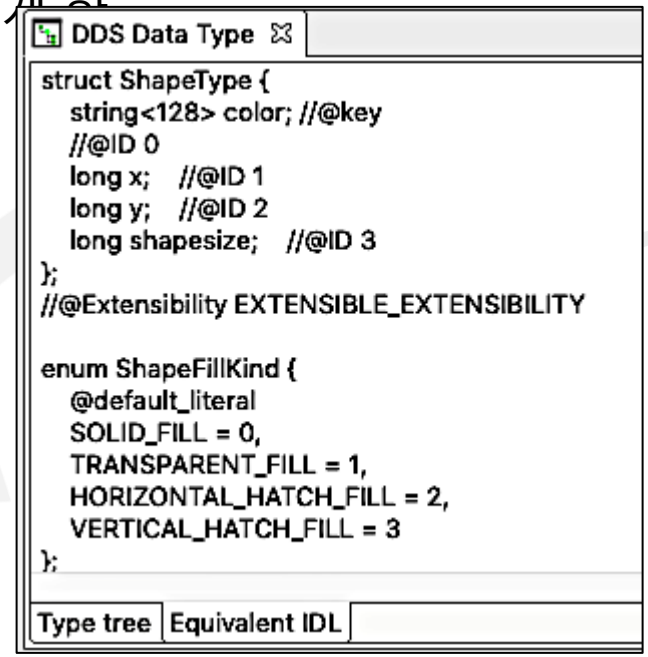
DDS 특징

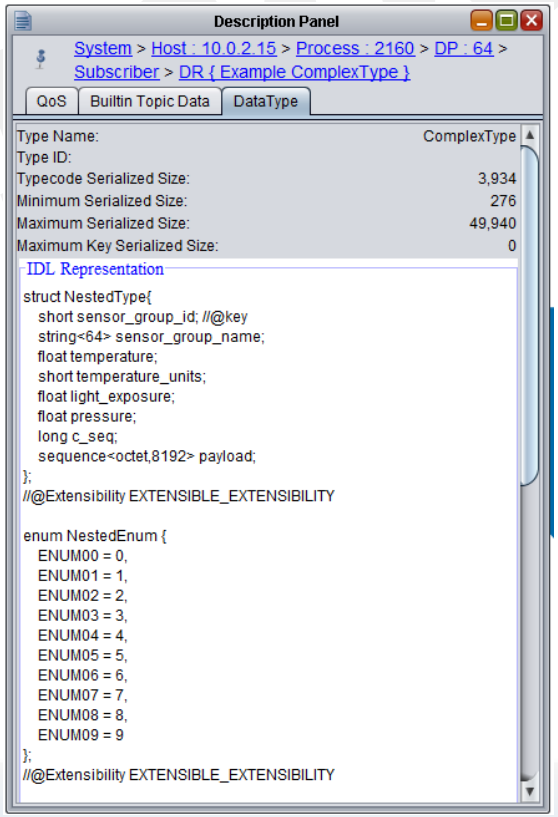
IDL Interface Definition Language

1. 설명

소프트웨어 컴포넌트의 인터페이스를 묘사하기 위한 명세 언어.

IDL은 어느 한 언어에 국한되지 않는 언어중립적인 방법으로 인터페이스를 표현함으로써, 같은 언어를 사용하지 않는 소프트웨어 컴포넌트 사이의 통신을 가능하게 함

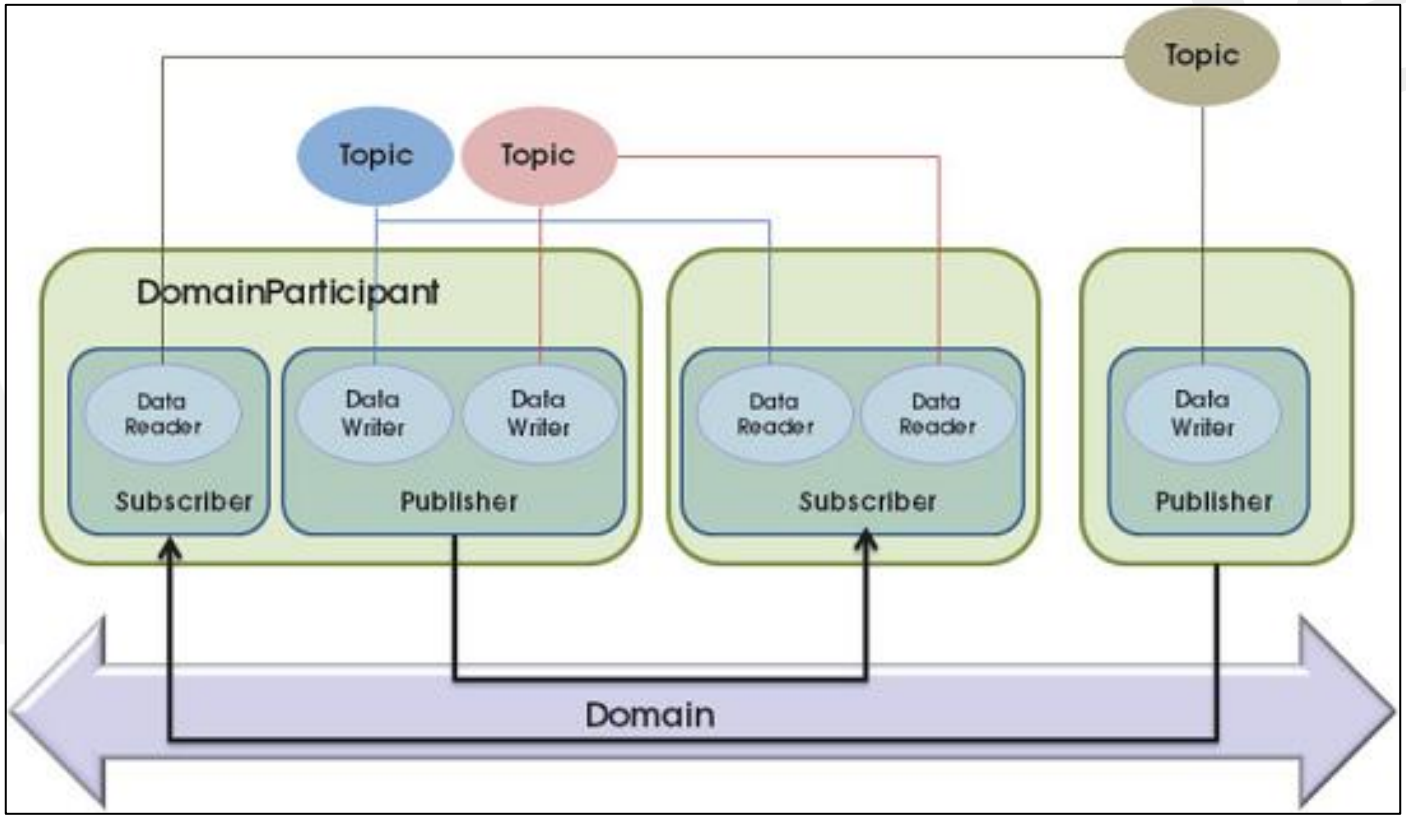




DDS 특징

Domain ID

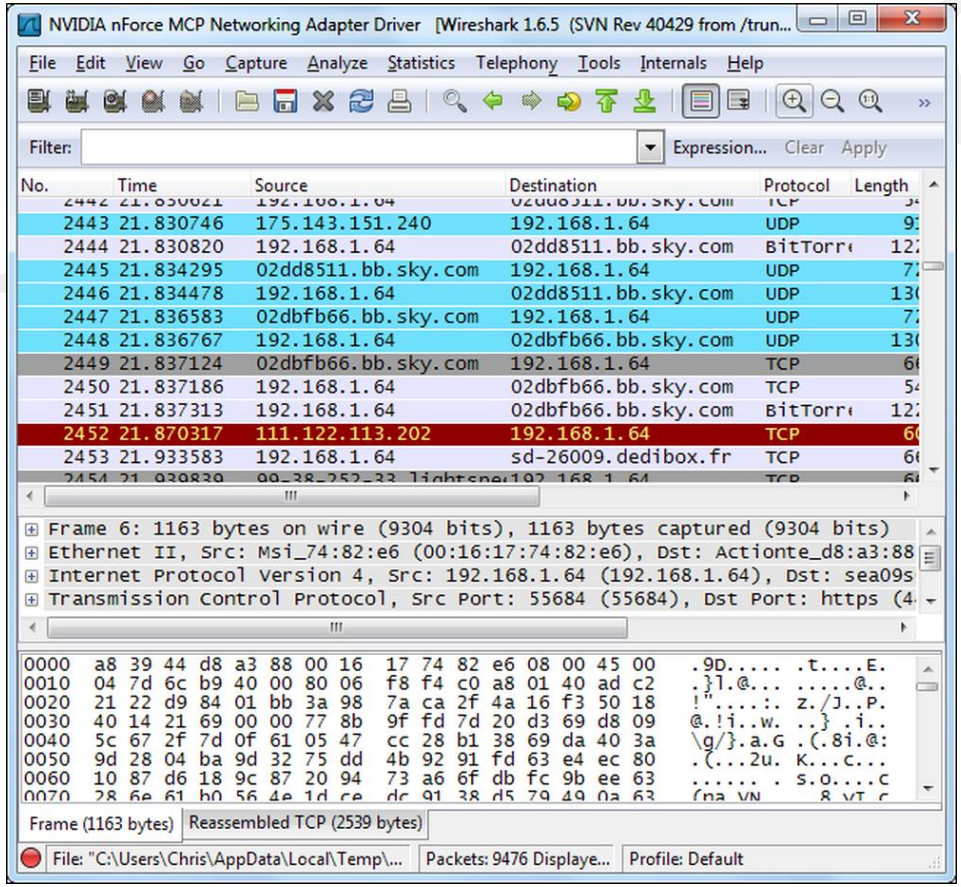
1. 자동으로 연결할 대상을 찾아주는 Auto-Discovery 기능



Wireshark

특징/장점

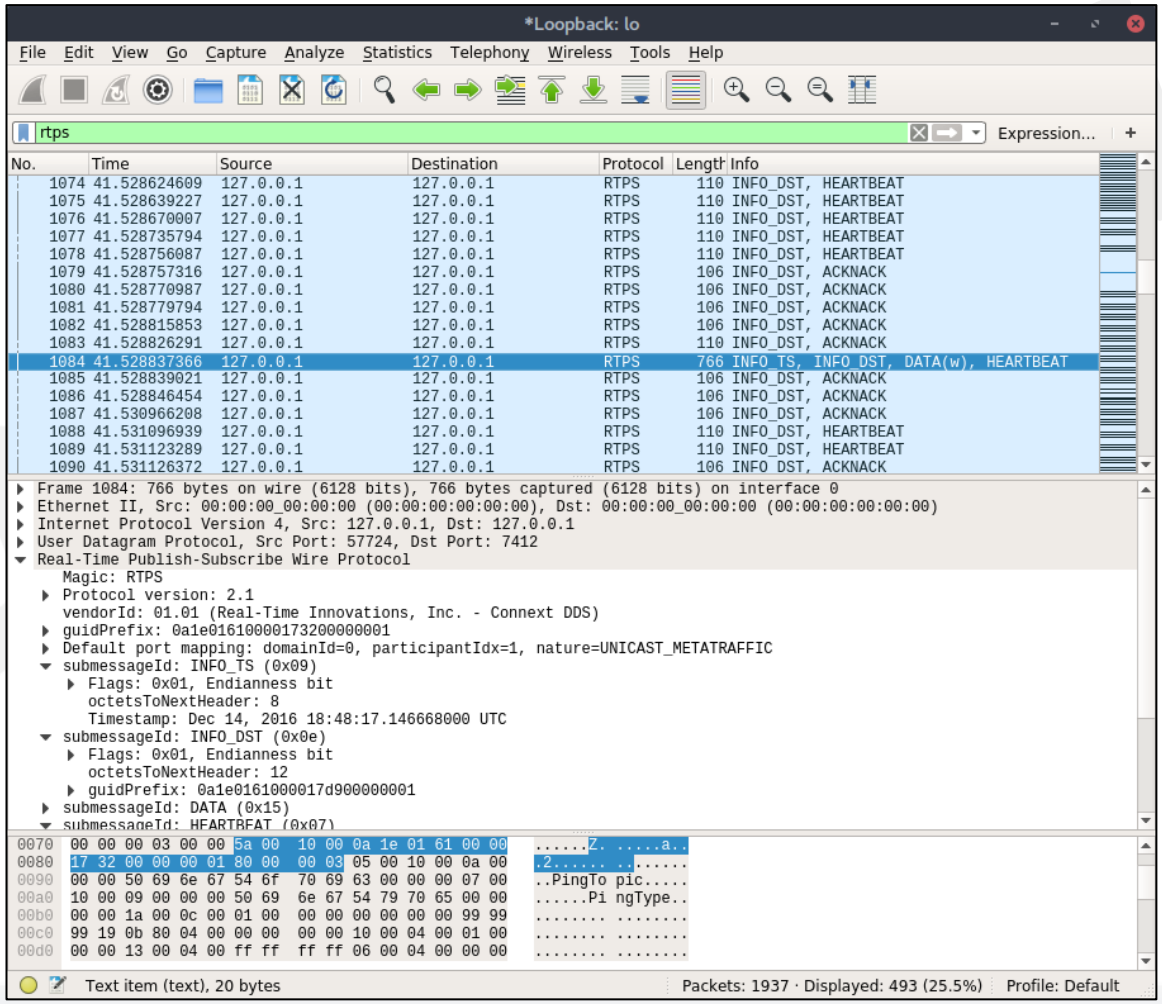
기능: 자유 및 오픈 소스 패킷 분석 프로그램이다.
네트워크의 문제, 분석, 소프트웨어 및 통신 프로토콜 개발, 교육에 쓰임



Wireshark

RTPS protocol

Data packet check



VMware

HW settings

Host / VMware: Disconnect, Connect