ROKEY BOOT CAMP

ROS2 기초-5차시

Apr, 2025

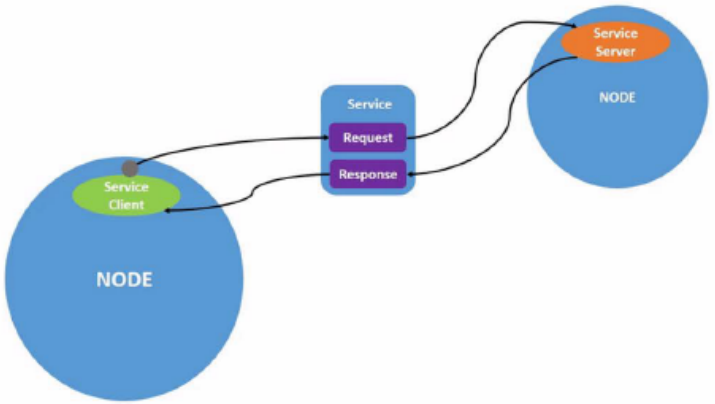
훈련 일정

	오전	오후
1차시	▪ 로봇의 역사 ▪ 컴퓨터 구조(Booting, CPU 작동원리, POST)	▪ 리눅스와 운영체계 ▪ 리눅스 CLI 실습(디렉토리, 계정, 기본 명령어등), Terminator, 커널, 쉘, gedit, bash
	▪ Application 작동원리(마이크로 프로세서, 메모리, 저장장치)	▪ 리눅스 CLI 실습
2차시	▪ 리눅스 CLI 실습	▪ 네트워크와 통신(IPV4, IPV6, 소켓, 노드, Hub, TCP, UDP)
	▪ API, Library, Framework, 프로세스와 Thread ▪ 인터프리터, 컴파일러(소스코드 → Build → 실행파일)	▪ 소켓프로그래밍 실습 ▪ OSI7 Layer, 프로토콜(RS-232C, RS-485, Ethernet, TCP/UDP, IP)
3차시	▪ 센서 기초, IoT와 Embedded	▪ 로봇기초, 좌표계
	▪ 로봇 센서 활용 및 로봇의 구성(기계기구, 전기전자, 소프트웨어)	▪ ROS2 소개 및 활용 ▪ ROS2 설치(ros.org) 및 demo_node
4차시	▪ ROS2 소개 및 활용	▪ ROS2 실습(Talker, Listener)
	▪ ROS2 패키지 설명	▪ ROS2 실습(Turtlesim, teleop_key)
5차시	▪ ROS2 실습(Turtlesim, Teleop_key 여러 개 만들기)	▪ Topic, Service, Action, Parameter, RQT, RQT_Graph 이론 및 실습
	▪ ROS2 실습(Turtlesim, Namespace 여러 개 만들기)	▪ Ros bag and play 실습, my first package build 실습 ▪ Turtlesim subscribing 실습, ROS의 중요한 개발도구(Rviz, GAZEBO 소개)

Message

[Topic]

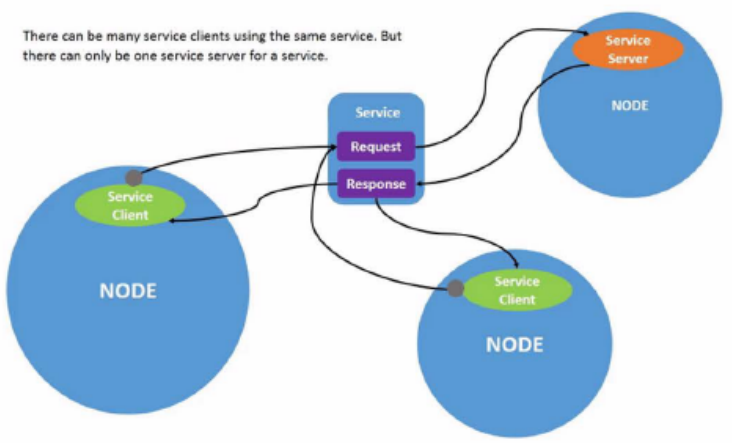
- 비동기식 단방향 메시지 송수신 방식으로, msg 인터페이스 형태의 메시지를 보내는 Publisher와 메시지를 받는 Subscriber 간의 통신
- 이는 1:1 통신을 기본으로 하지만 N:N 통신도 가능하며, ROS 메시지 통신에서 가장 많이 사용
- 비동기성과 연속성을 가지기 때문에 센서 값 전송이나 항시 정보를 주고받아야 하는 부분에서 주로 사용



Message

[Service]

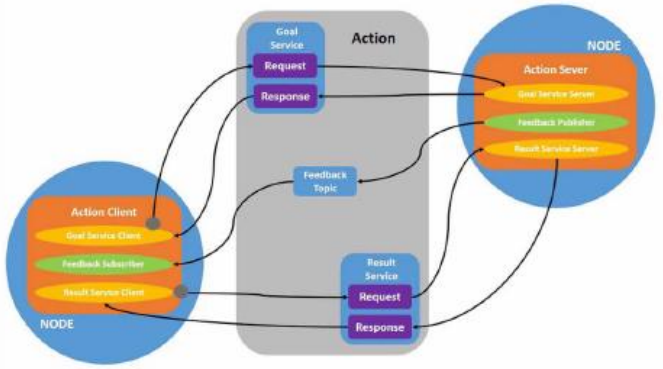
- 동기식 양방향 메시지 송수신 방식으로, 서비스 요청(Request) 하는 쪽을 Server Client, 요청 받은 서비스를 수행한 후 서비스 응답(Response)을 하는 쪽을 Service Server
- 특정 요청을 하는 클라이언트 단과 요청 받은 일을 수행한 후에 결과값을 전달하는 서버 단과의 통신이다. 인터페이스는 srv를 사용
- 동일한 서비스 서버에 대해 복수의 클라이언트를 가질 수 있도록 설계되었지만, 서비스 응답은 서비스 요청이 있었던 서비스 클라이언트에 대해서만 응답하는 형태

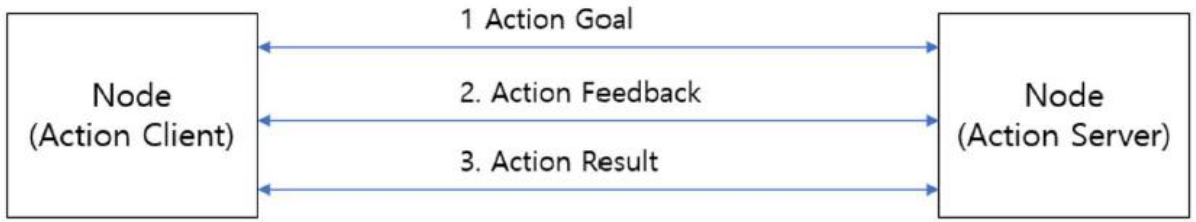


Message

[Action]

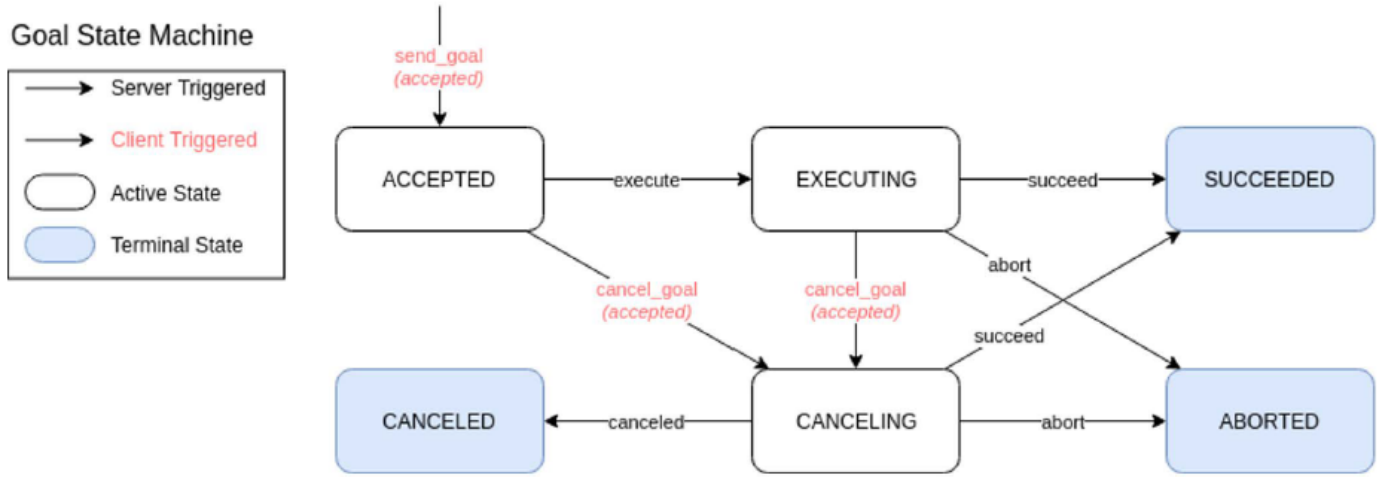
- 비동기식, 동기식 양방향 메시지 송수신 방식
- 액션 목표를 지정하는 Action Client와, 액션 목표(Goal)를 받아 특정 태스크를 수행하면서 중간 결과값을 전송하는 액션 피드백(Feedback), 최종 결과값을 담은 액션 결과(Result)를 전송하는 Action Server 간의 통신





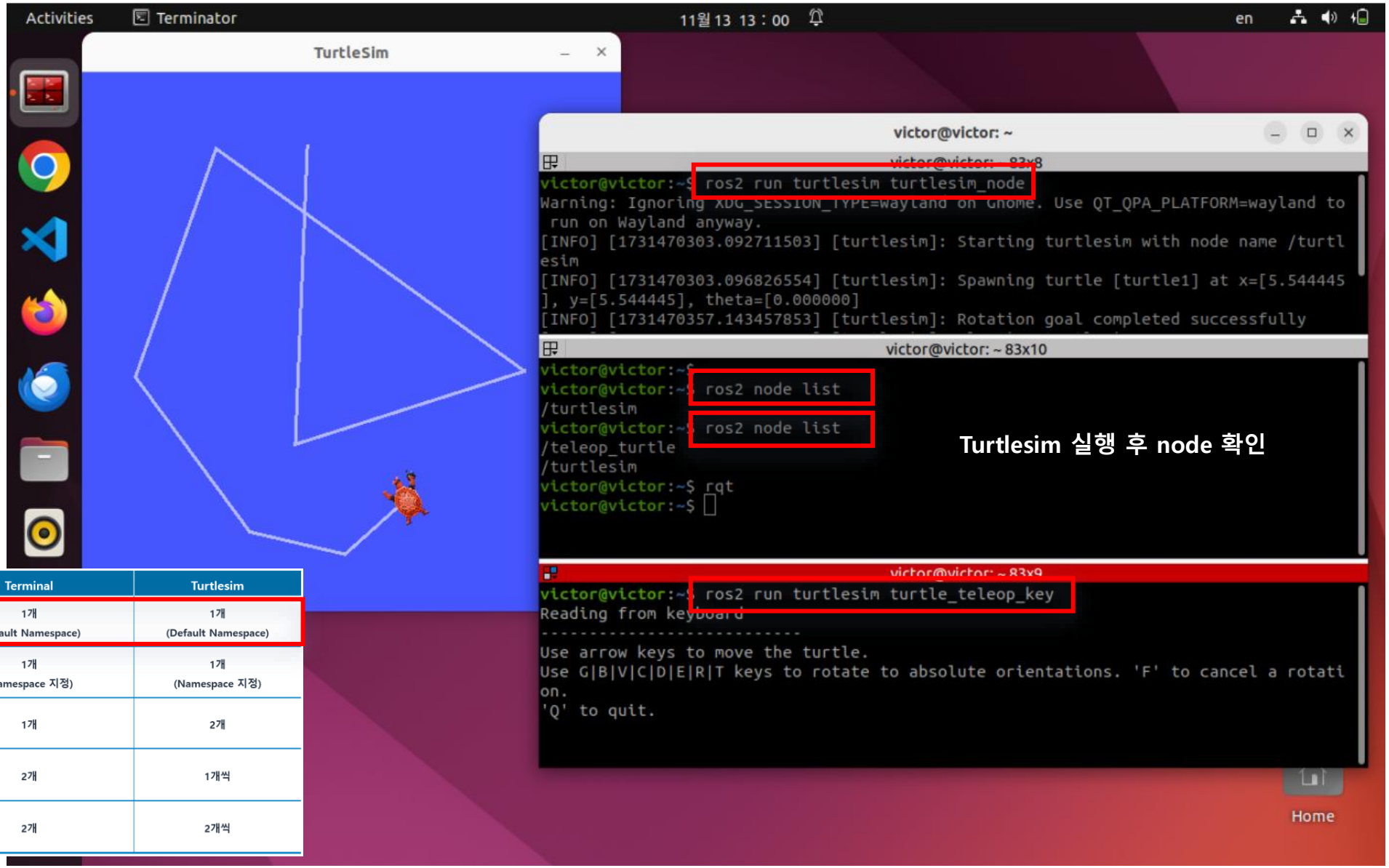
Message

- ROS2에서의 액션은 목표 전달(send_goal), 표 취소(cancel_goal),결과 받기(get_result)를 위한 서비스 통신을 사용하기 위하여 토픽과 서비스 방식을 혼합하여 사용
- 비동기 방식에서 원하는 타이밍에 적절한 액션을 수행하기 위하여 목표상태(goal_state)를 도입, 이는 목표값을 전달한 후 상태 머신을 구동하여 액션의 프로세스를 쫓음
- 즉, 액션 목표 전달 이후 액션의 상태값을 액션 클라이언트에 전달함에 따라, 비동기 및 동기 방식이 혼재된 액션의 처리를 원할하게 해준다.



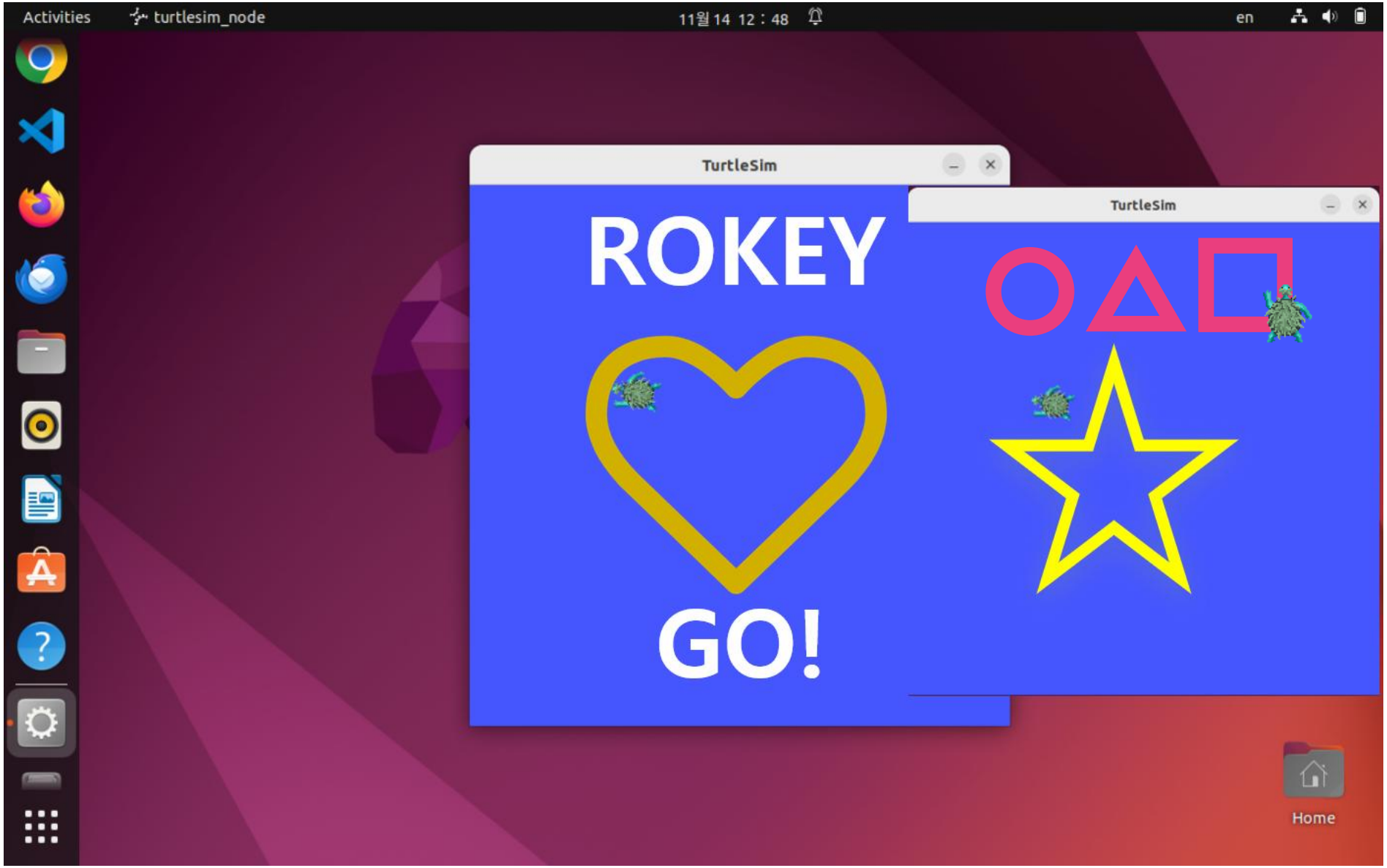
	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

ROS2 실습 - turtlesim 1단계



	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

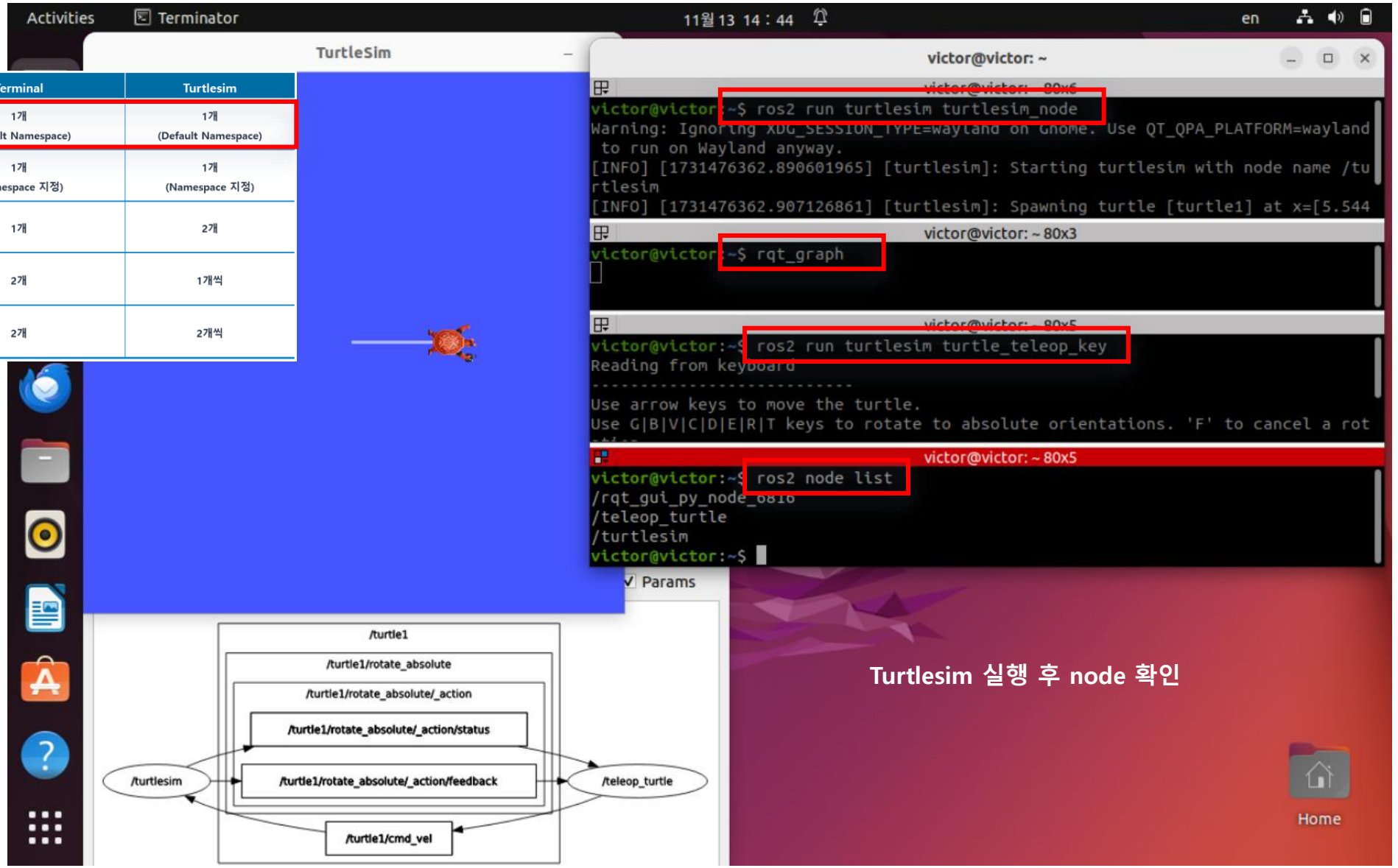
ROS2 실습


Activities
turtlesim_node
11월 14 12 : 48
en
TurtleSim
ROKEY
GO!
TurtleSim
Home

ROS2 실습 - turtlesim

1

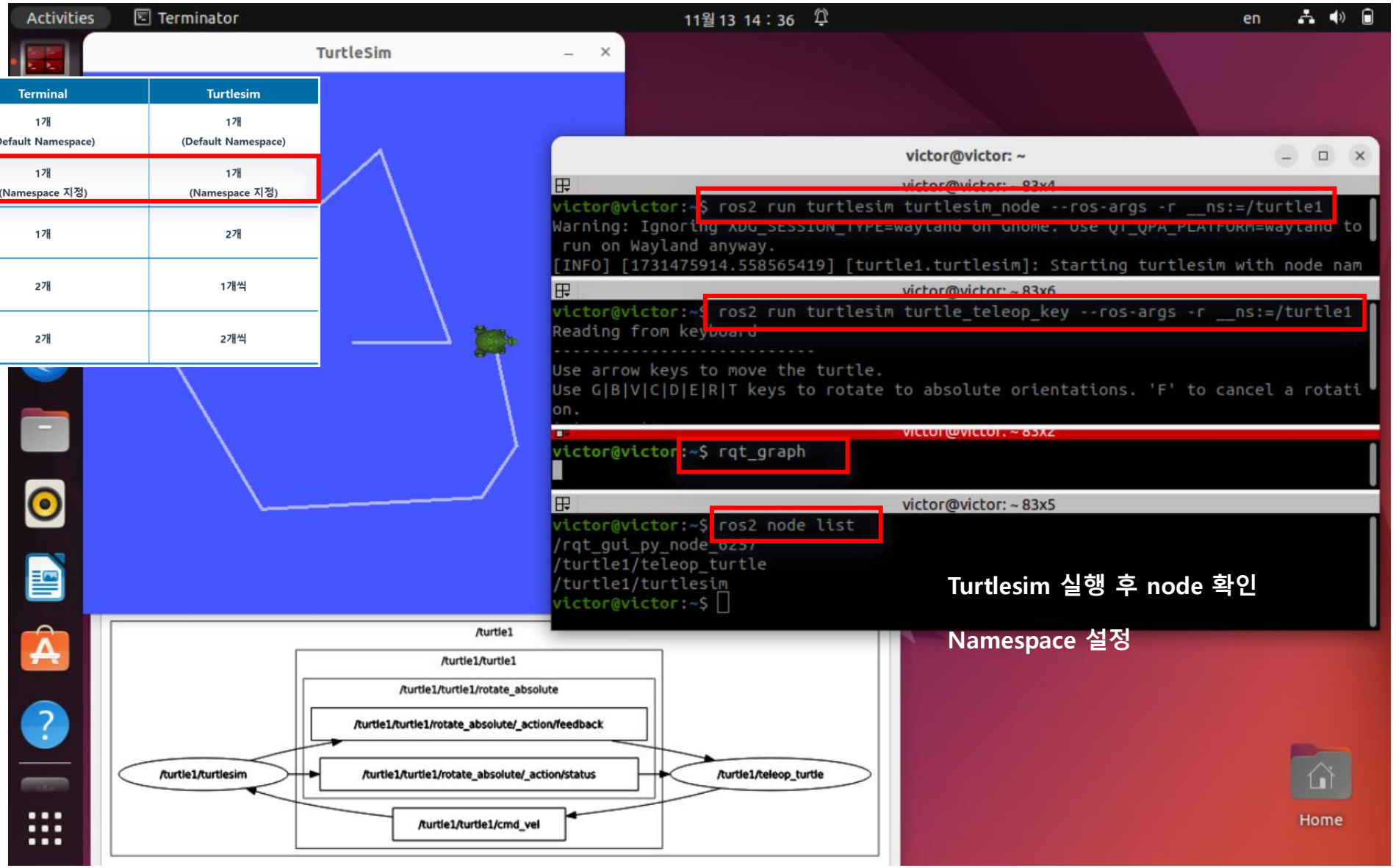
	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩



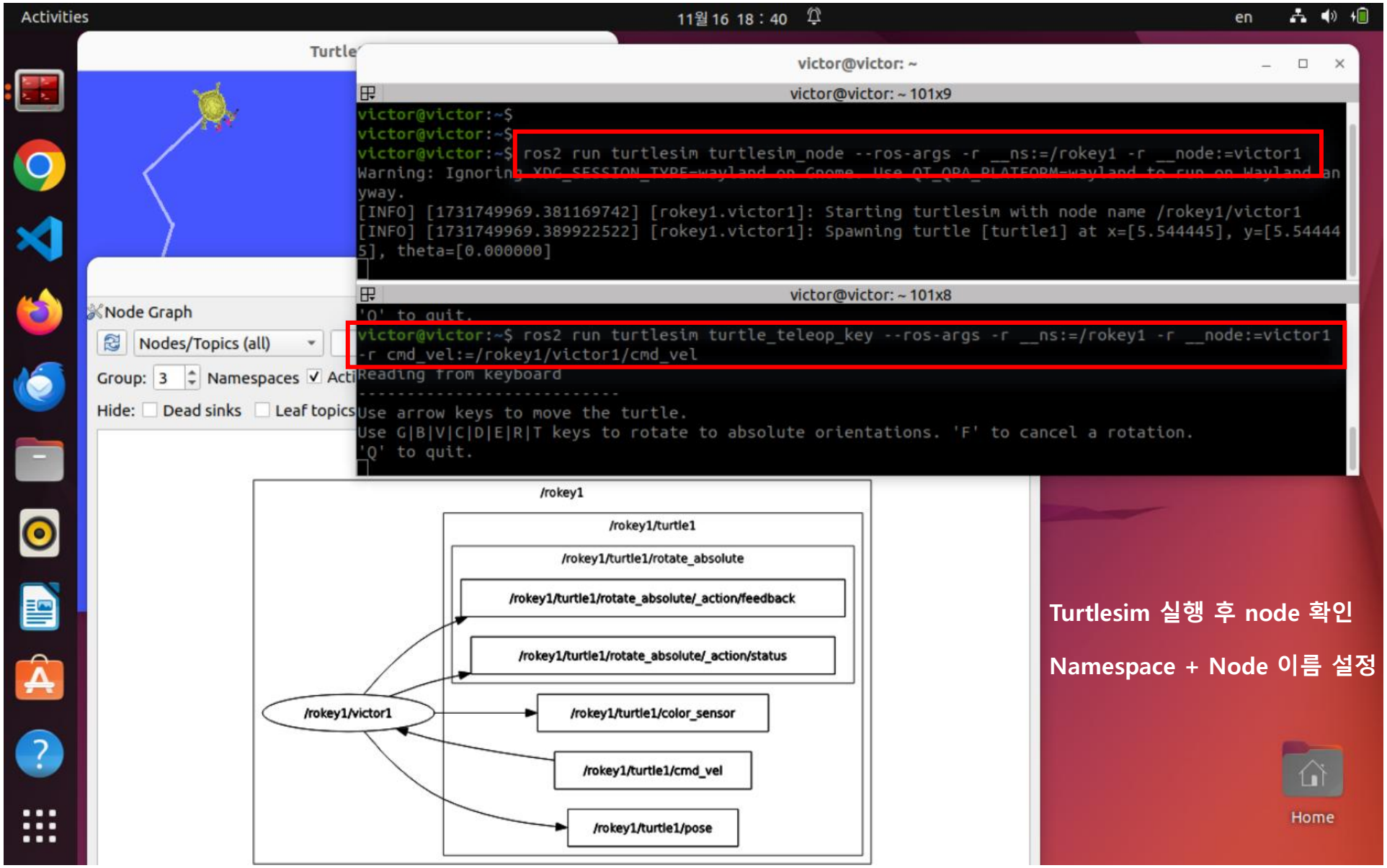
ROS2 실습

2

	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

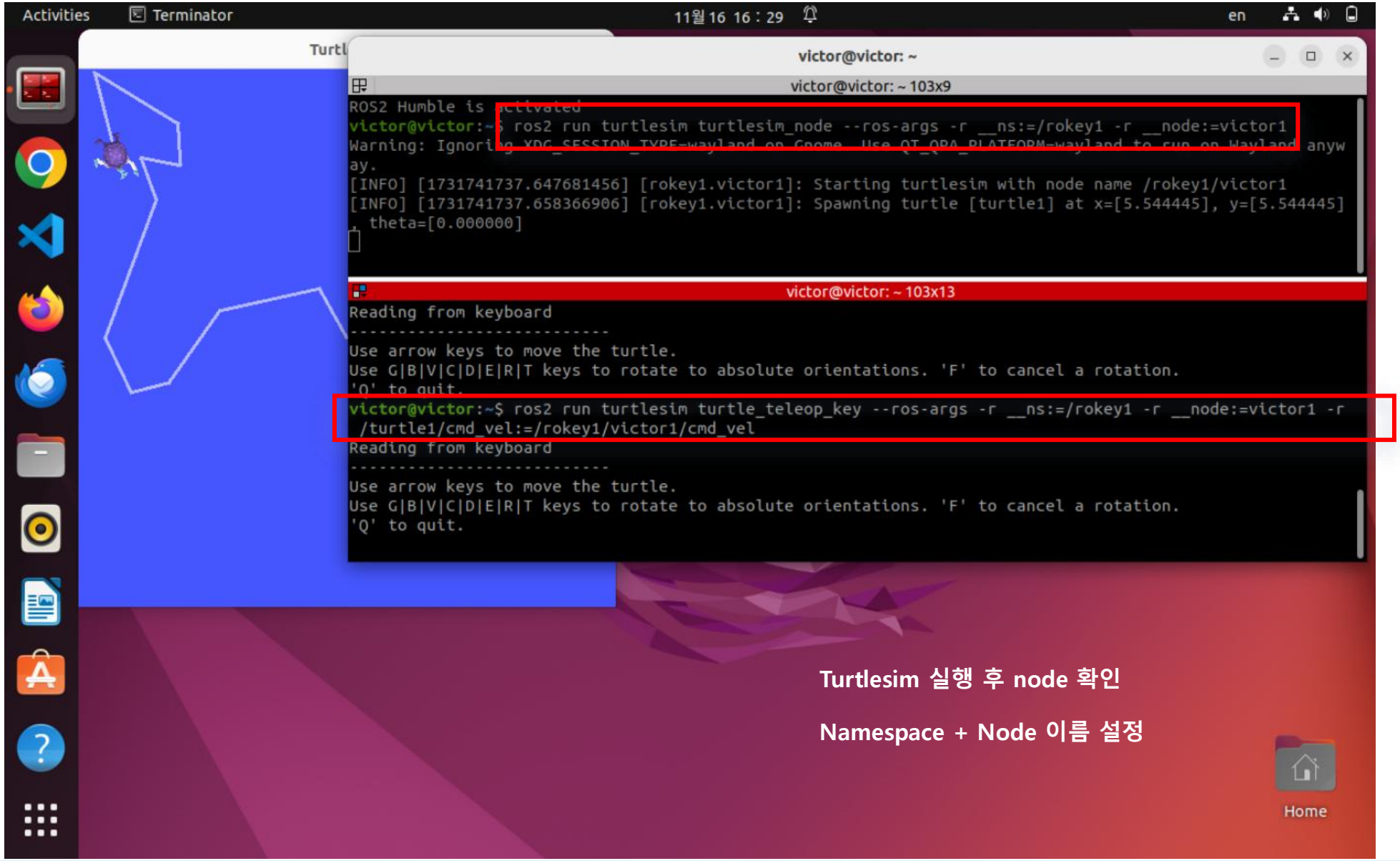


ROS2 실습


Activities
11월 16 18 : 40
en
Turtle
victor@victor: ~
victor@victor: ~ 101x9
victor@victor:~$
victor@victor:~$
victor@victor:~$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/rokey1 -r __node:=victor1
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland an
yway.
[INFO] [1731749969.381169742] [rokey1.victor1]: Starting turtlesim with node name /rokey1/victor1
[INFO] [1731749969.389922522] [rokey1.victor1]: Spawning turtle [turtle1] at x=[5.544445], y=[5.54444
5], theta=[0.000000]
victor@victor: ~ 101x8
'Q' to quit.
victor@victor:~$ ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/rokey1 -r __node:=victor1
-r cmd_vel:=/rokey1/victor1/cmd_vel
Reading from keyboard
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
Node Graph
Nodes/Topics (all)
Group: 3
Namespaces
Hide:
Dead sinks
Leaf topics
/rokey1
/rokey1/turtle1
/rokey1/turtle1/rotate_absolute
/rokey1/turtle1/rotate_absolute/_action/feedback
/rokey1/turtle1/rotate_absolute/_action/status
/rokey1/victor1
/rokey1/turtle1/color_sensor
/rokey1/turtle1/cmd_vel
/rokey1/turtle1/pose
Turtlesim 실행 후 node 확인
Namespace + Node 이름 설정
Home

ROS2 실습

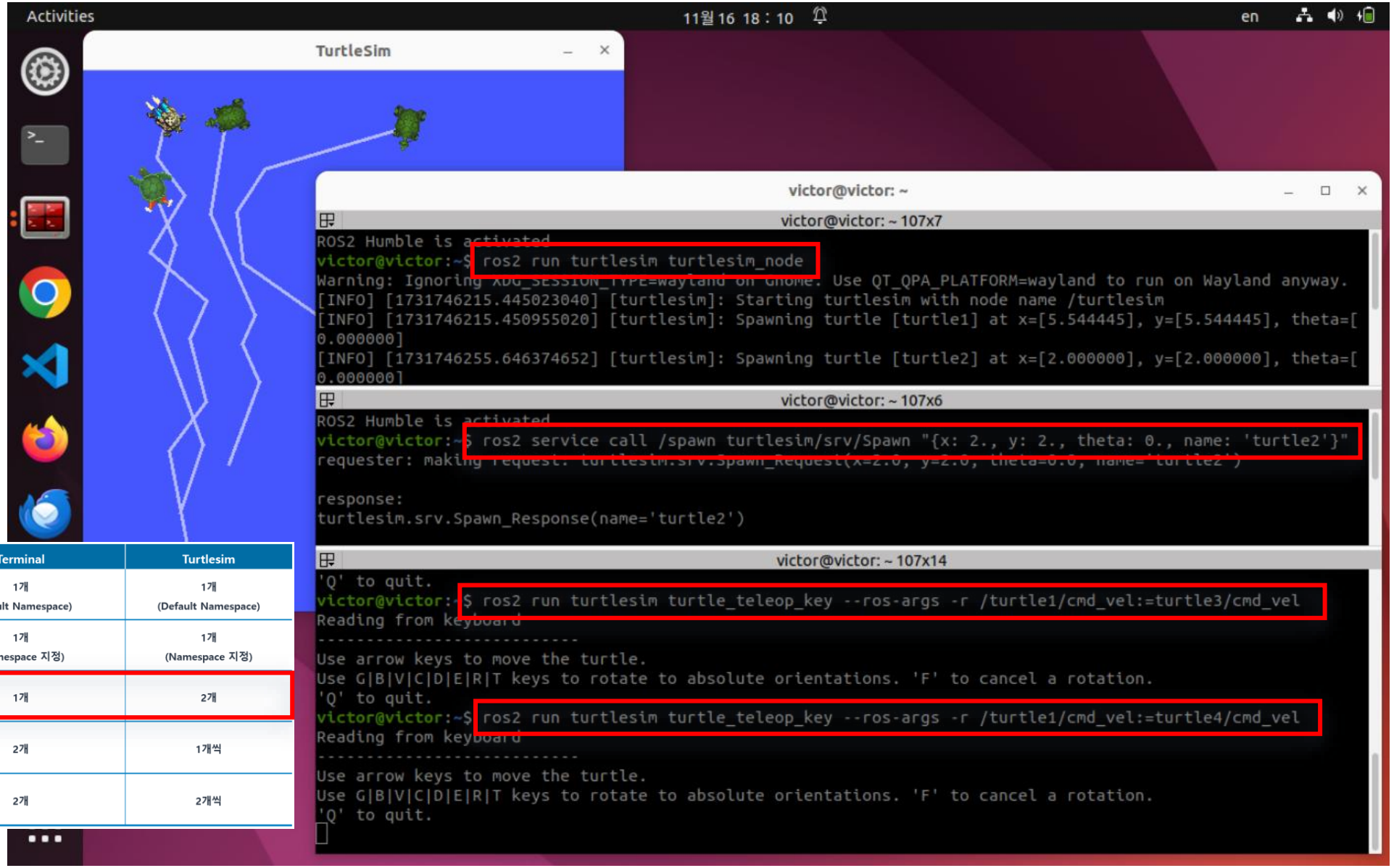
2



Turtlesim 실행 후 node 확인

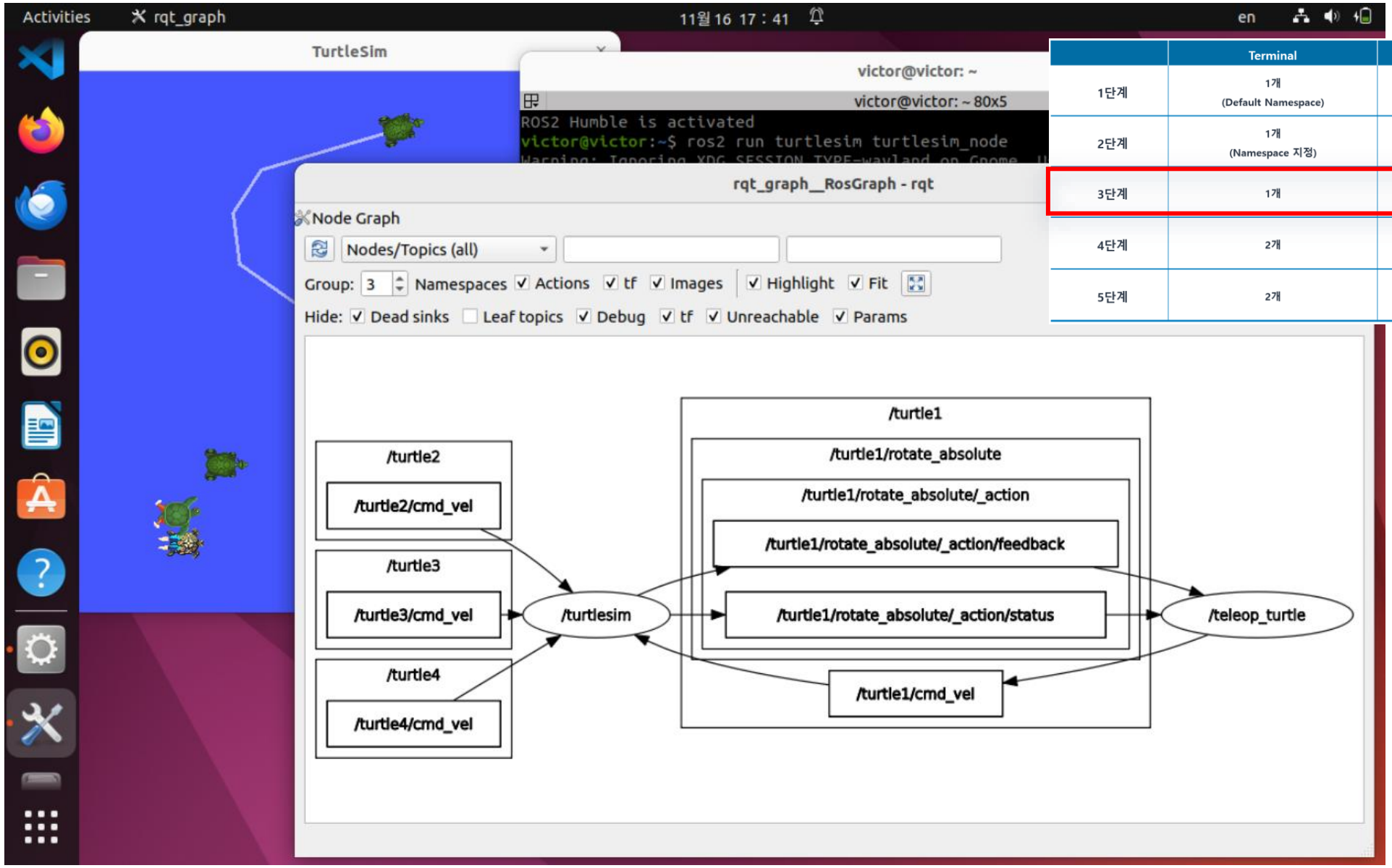
Namespace + Node 이름 설정

ROS2 실습



	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

ROS2 실습

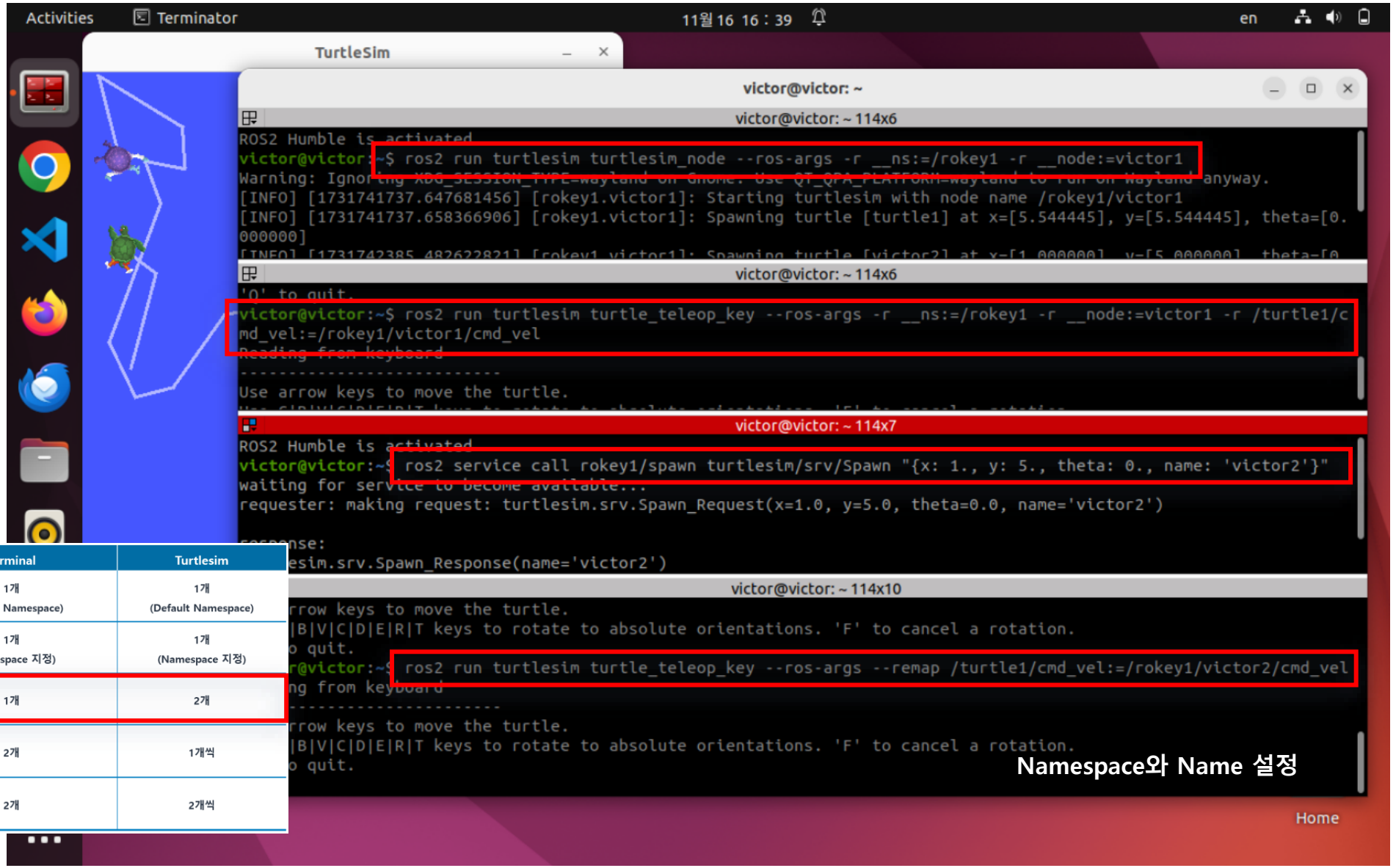


3

	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

ROS2 실습

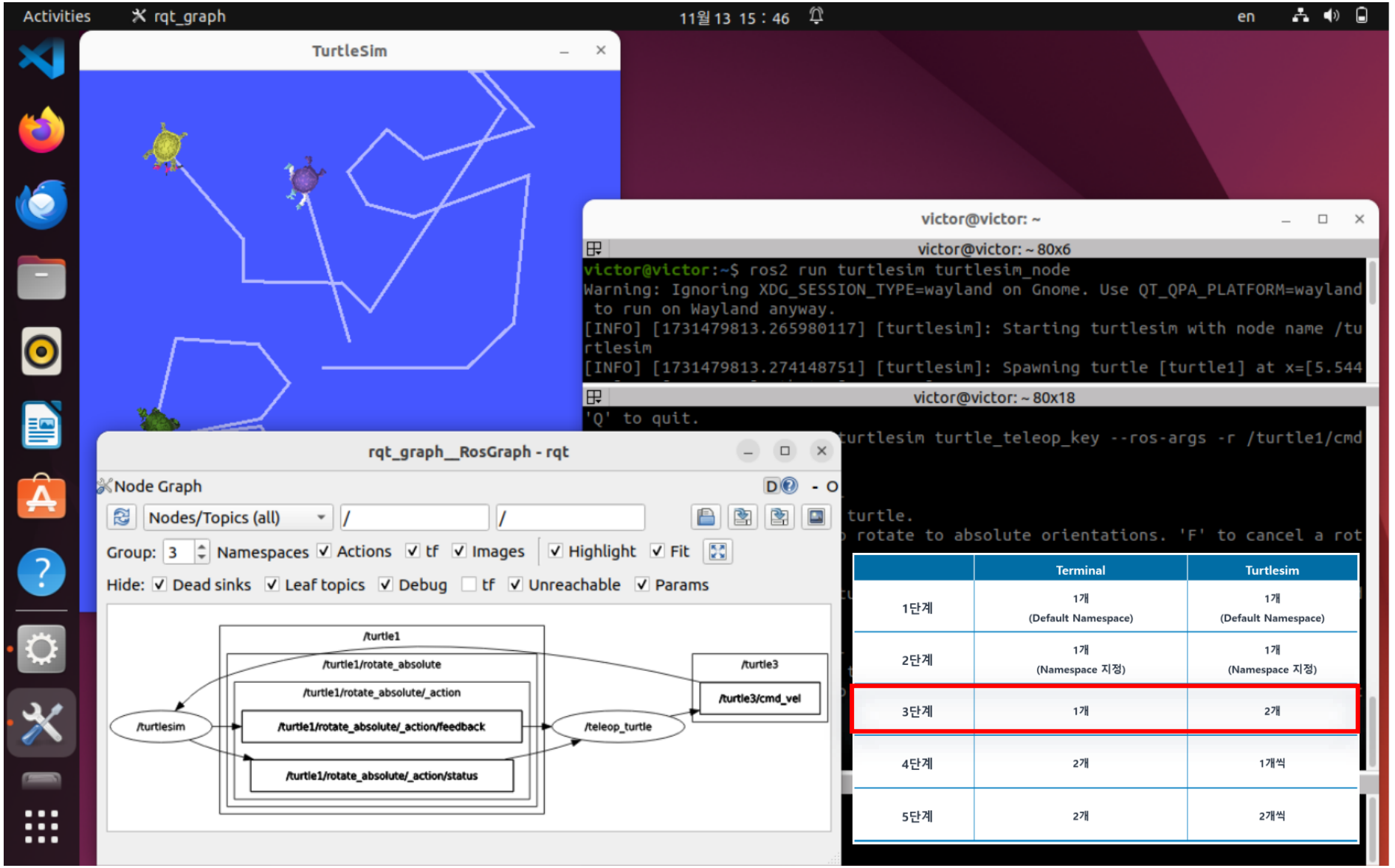
3



	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

Namespace와 Name 설정

ROS2 실습

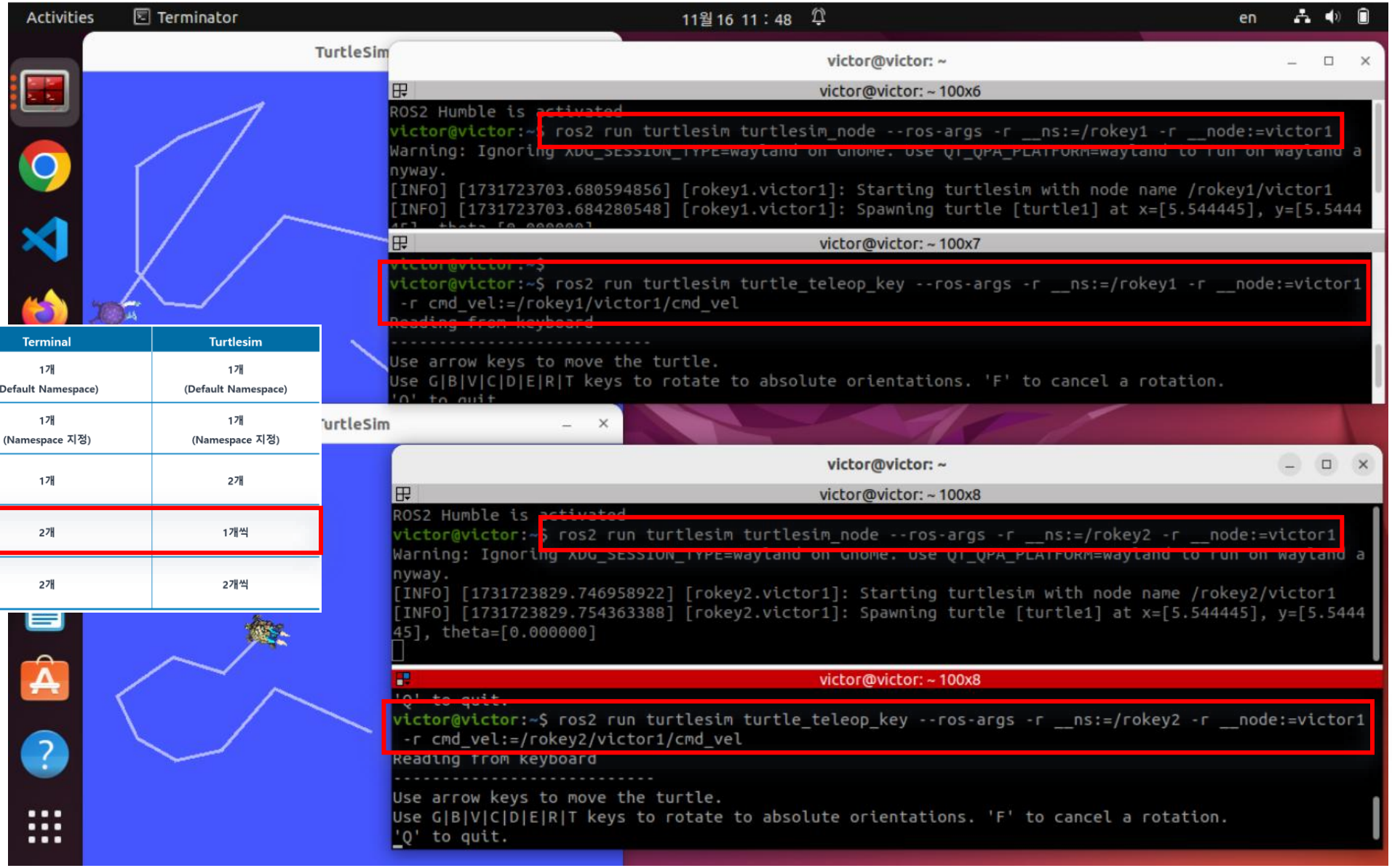


	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

ROS2 실습

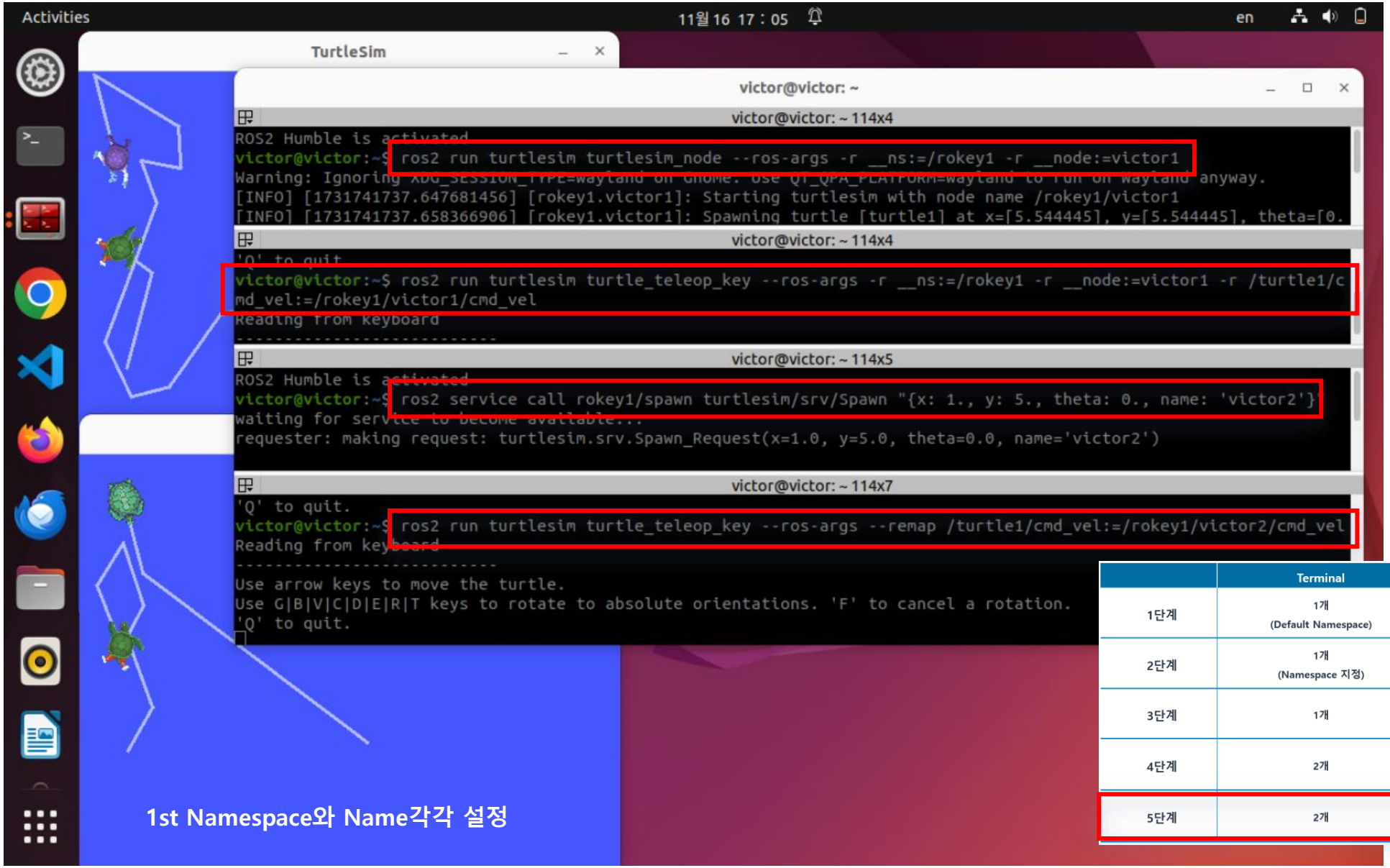
4

	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩



ROS2 실습

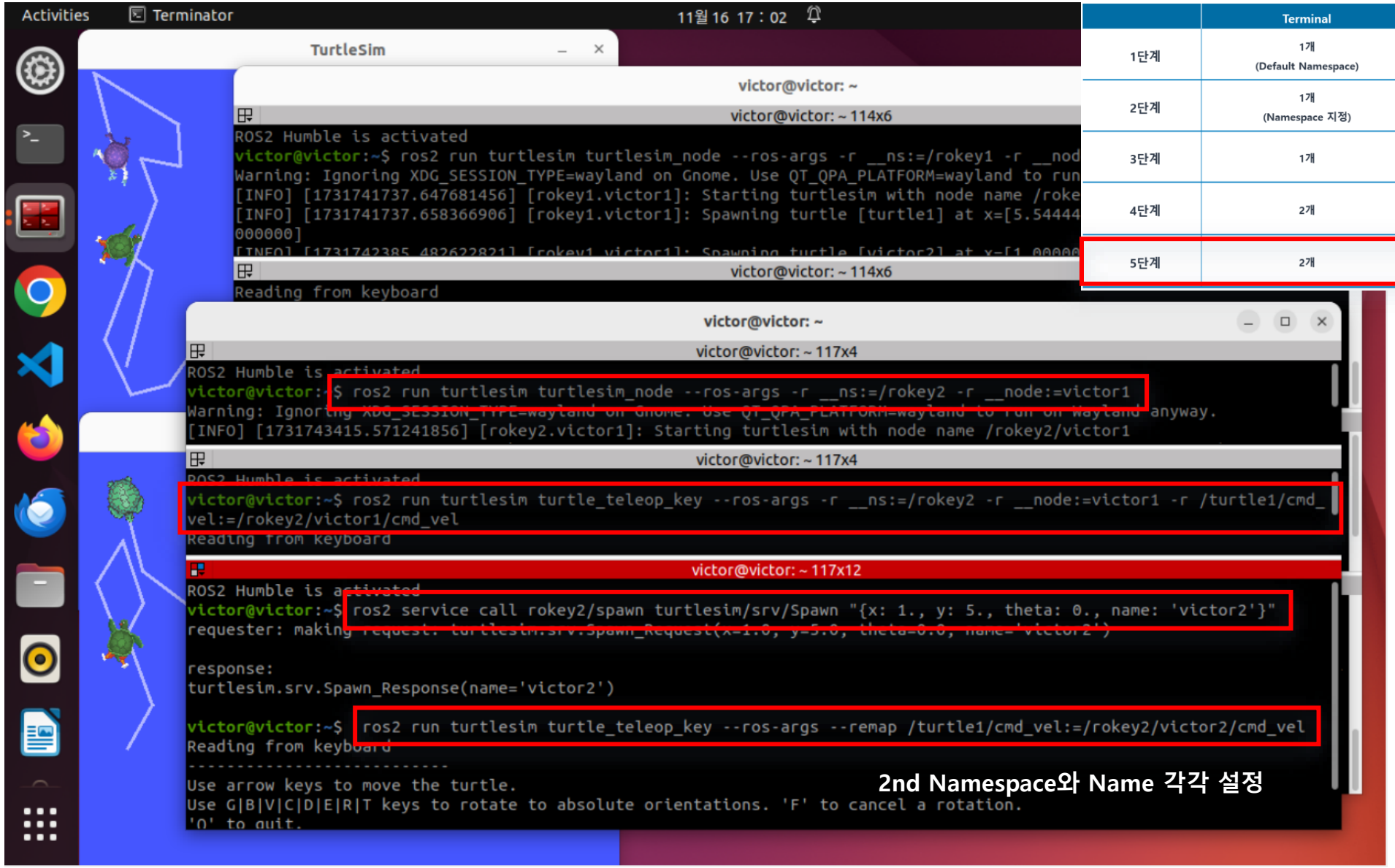
5



1st Namespace와 Name각각 설정

	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

ROS2 실습

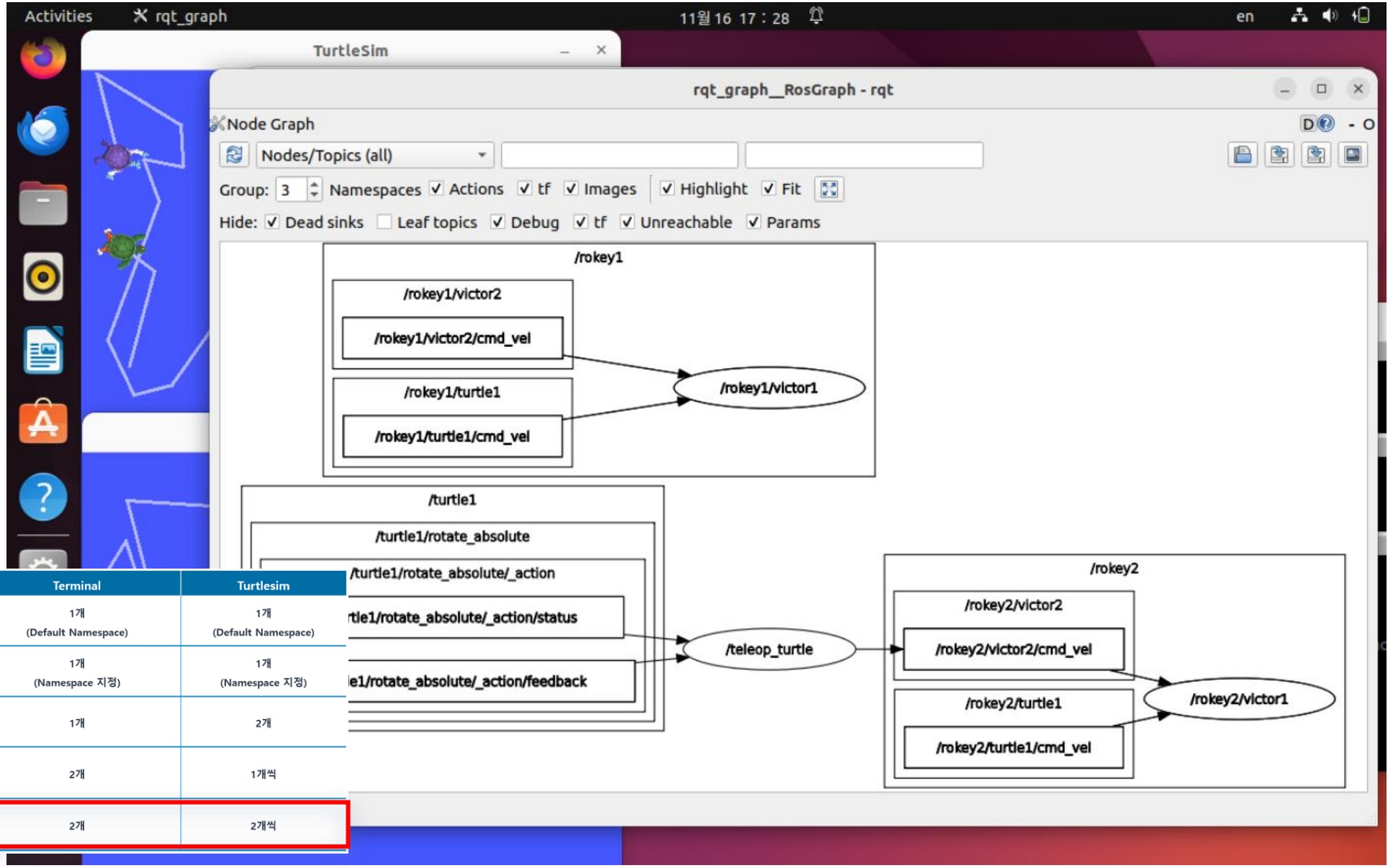


	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

5

ROS2 실습

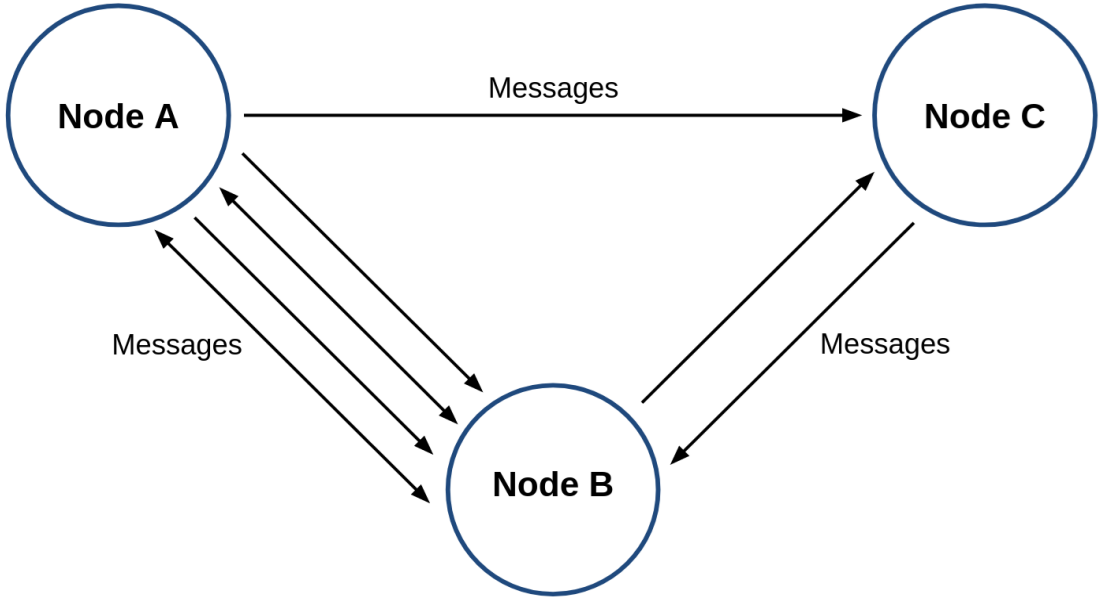
5



	Terminal	Turtlesim
1단계	1개 (Default Namespace)	1개 (Default Namespace)
2단계	1개 (Namespace 지정)	1개 (Namespace 지정)
3단계	1개	2개
4단계	2개	1개씩
5단계	2개	2개씩

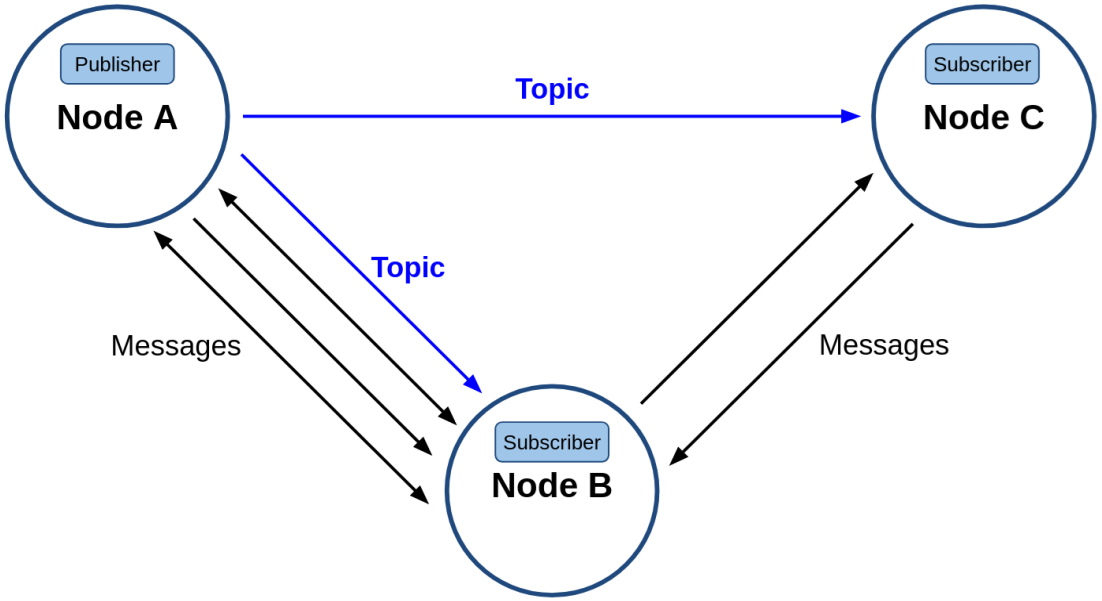
노드(node)

- 아래 그림처럼 Node A, Node B, Node C 각각의 노드들은 서로 유기적으로 Message로 연결
- 수행하고자 하는 태스크가 많아질수록 메시지로 연결되는 노드가 늘어나며 시스템이 확장



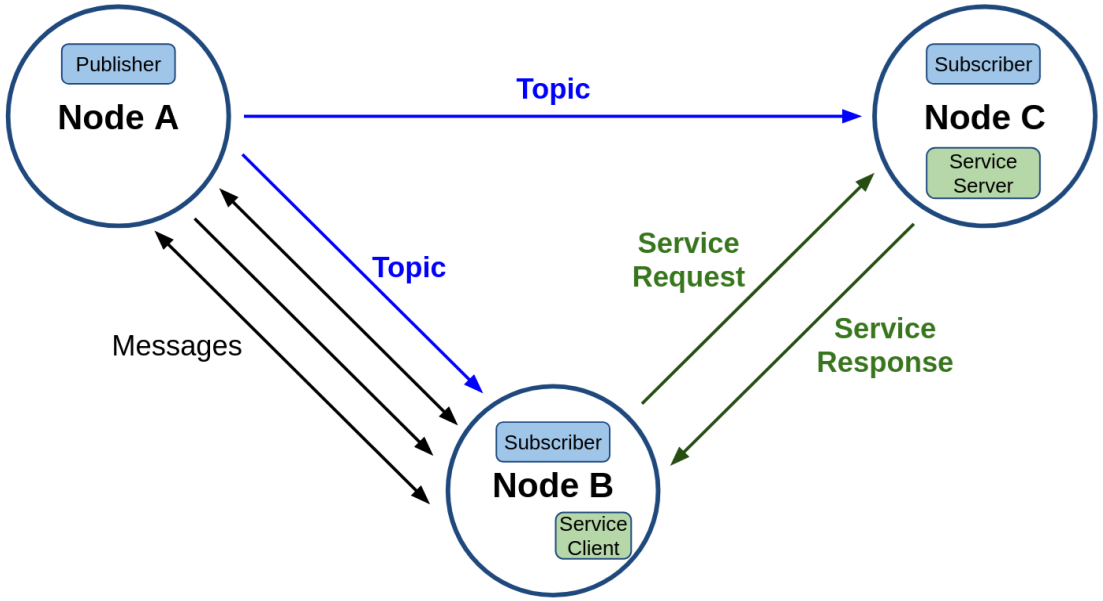
토픽(topic)

- 아래 그림의 `Node A – Node B`, `Node A – Node C`처럼 비동기식 단방향 메시지 송수신 방식
- msg 메시지 형태의 메시지를 발간하는 Publisher
- 메시지를 구독하는 Subscriber 간의 통신
- 이는 1:N, N:1, N:N 통신도 가능
- ROS 메시지 통신에서 가장 널리 사용되는 통신 방법이다.



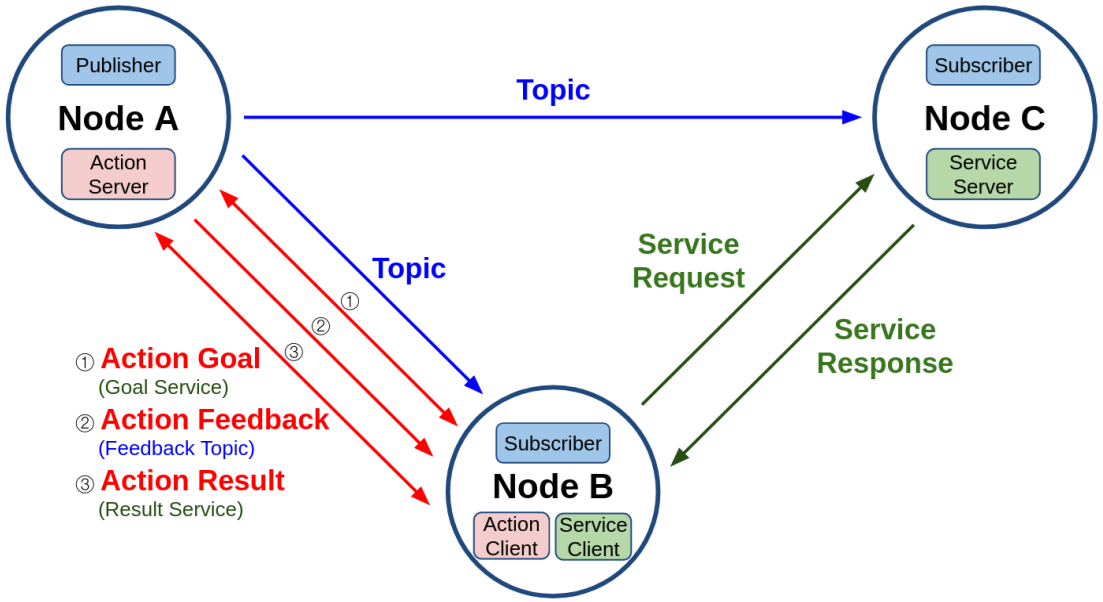
서비스(Service)

- 아래 그림의 `Node B - Node C`처럼 동기식 양방향 메시지 송수신 방식
- 서비스의 요청(Request)을 하는 쪽은 Service client
- 서비스의 응답(Response)을 하는 쪽을 Service server
- 특정 요청을 하는 클라이언트 단과 요청받은 일 수행 후 결과 값을 전달하는 서버 단과의 통신
- 서비스 요청 및 응답(Request/Response) 또한 위에서 언급한 msg 메시지의 변형으로 srv 메시지라고 함.



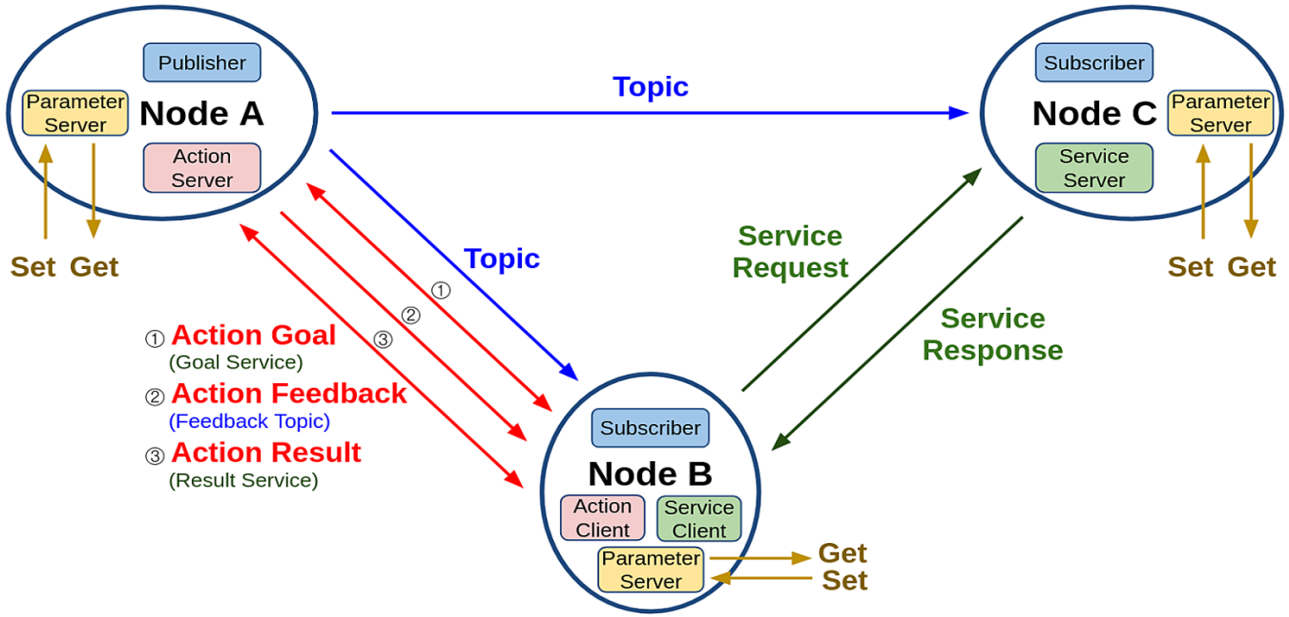
액션(Action)

- 토픽(topic)과 서비스(service)의 혼합
- 액션 목표 및 액션 결과를 전달하는 방식은 서비스와 같고,
- 액션 피드백은 토픽과 같은 메시지 전송 방식
- 액션 목표/피드백/결과(Goal/Feedback/Result) 메시지 또한 위에서 언급한 msg 메시지의 변형으로 action 메시지라고 함.



파라미터(Parameter)

- 각 노드에 파라미터 관련 Parameter server를 실행시켜 외부의 Parameter client 간의 통신으로 파라미터를 변경하는 것으로 서비스와 동일
- 노드 내 매개변수 또는 글로벌 매개변수를 서비스 메시지 통신 방법을 사용하여 노드 내부 또는 외부에서 쉽게 지정(Set) 하거나 변경할 수 있고, 쉽게 가져(Get)와서 사용할 수 있게 하는 점에서 목적이 다름



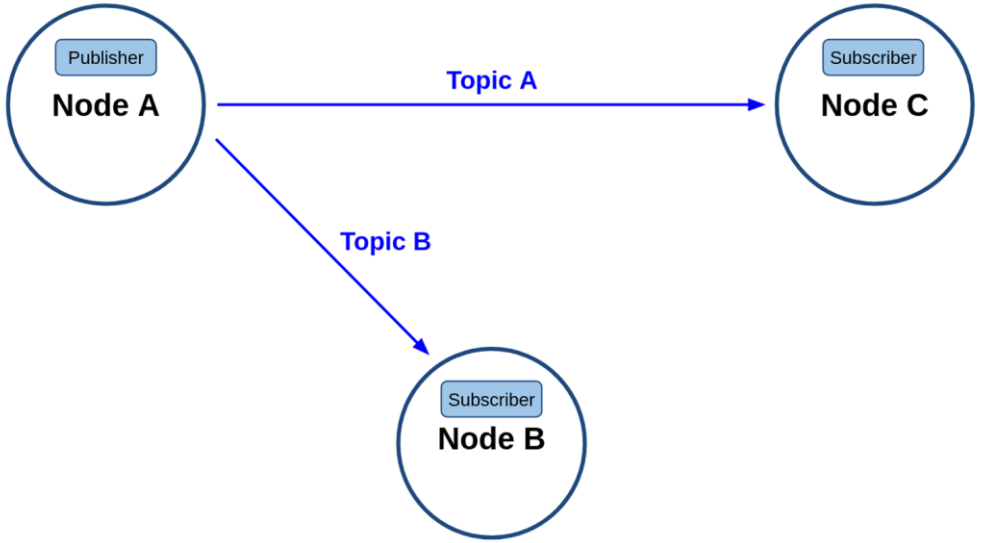
ROS2 실습 – 노드와 메시지 통신(10장)

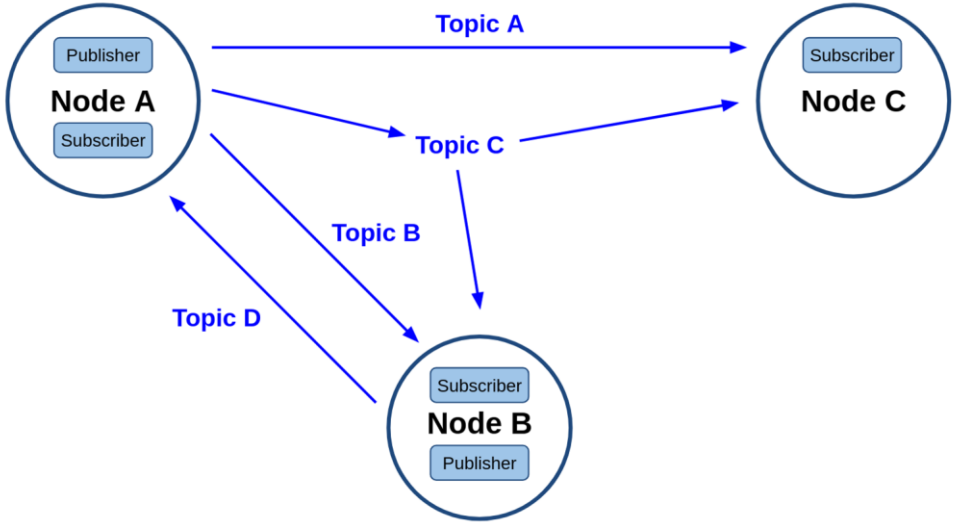
```
$ ros2 node list
/turtlesim
/teleop_turtle

$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose

$ ros2 action list
/turtle1/rotate_absolute


$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
……

$ ros2 node info /teleop_turtle
/turtlesim
  Subscribers:
…
```

```
$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

ROS2 실습 - ROS2 Topic(11장)

- `Node A`처럼 하나의 이상의 토픽을 발행
- `Publisher` 기능과 동시에 토픽(예: Topic D)을 구독하는 `Subscriber` 역할도 동시에 수행
- 자신이 발행한 토픽을 셀프 구독할 수 있게 구성할 수도 있음
- 토픽 기능은 목적에 따라 다양한 방법으로 사용할 수 있으며 이런 유연성으로 다양한 곳에 사용 중
- ROS 프로그래밍시에 70% 이상이 토픽으로 사용됨
- 통신 방식 중에 가장 기본이 되며 가장 널리쓰이는 방법
- 비동기성과 연속성을 가지기에 센서 값 전송 및 항시 정보를 주고 받아야 하는 부분에 주로 사용





ROS2 실습 - Topic

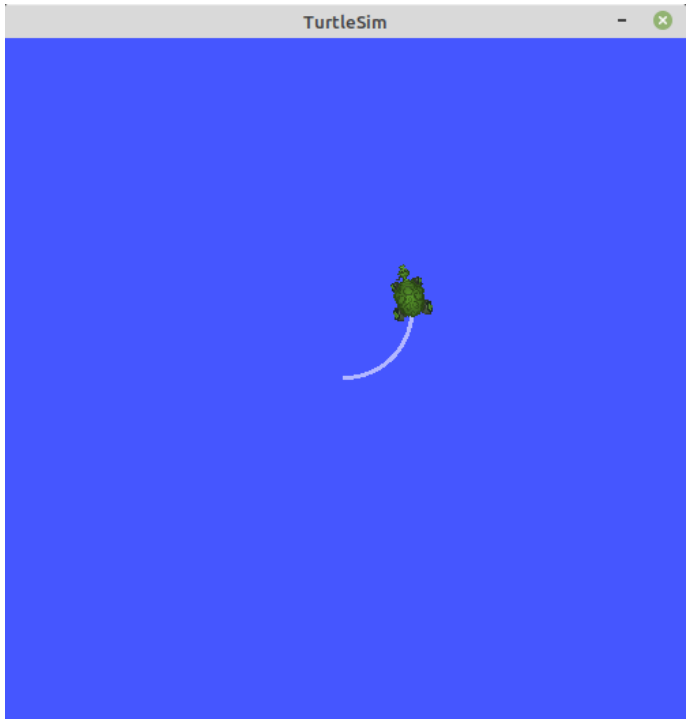
```
$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
```

ROS2 실습 - Topic

```
$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscriber count: 1

$ ros2 topic echo /turtle1/cmd_vel
linear:
x: 1.0
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.0
```

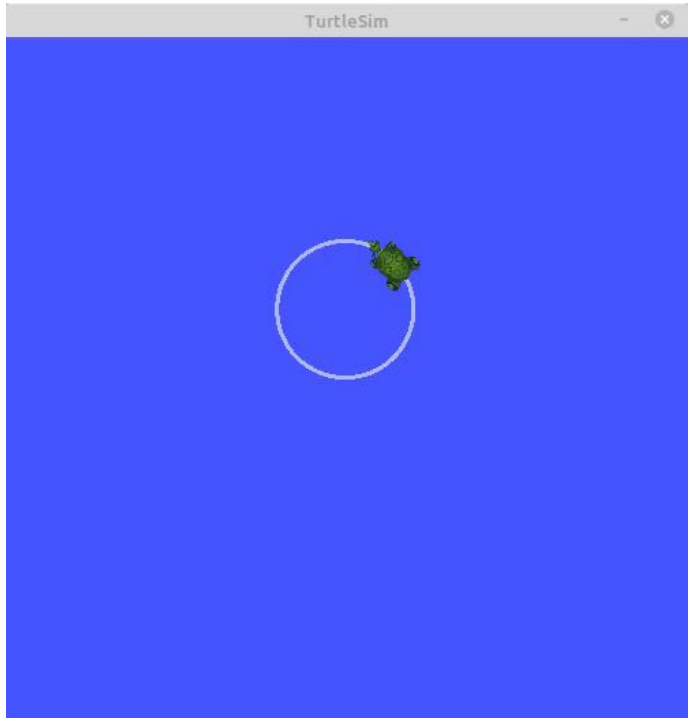
ROS2 실습 – Topic

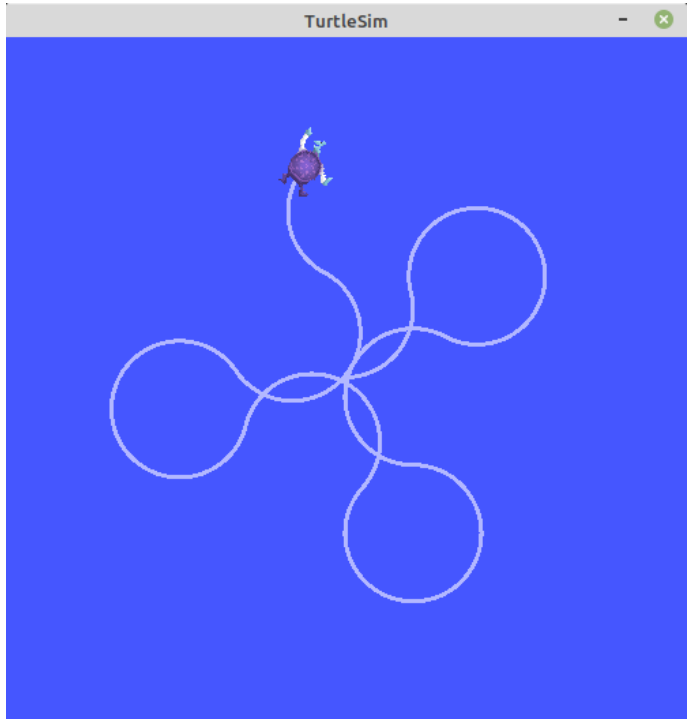
```
$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```



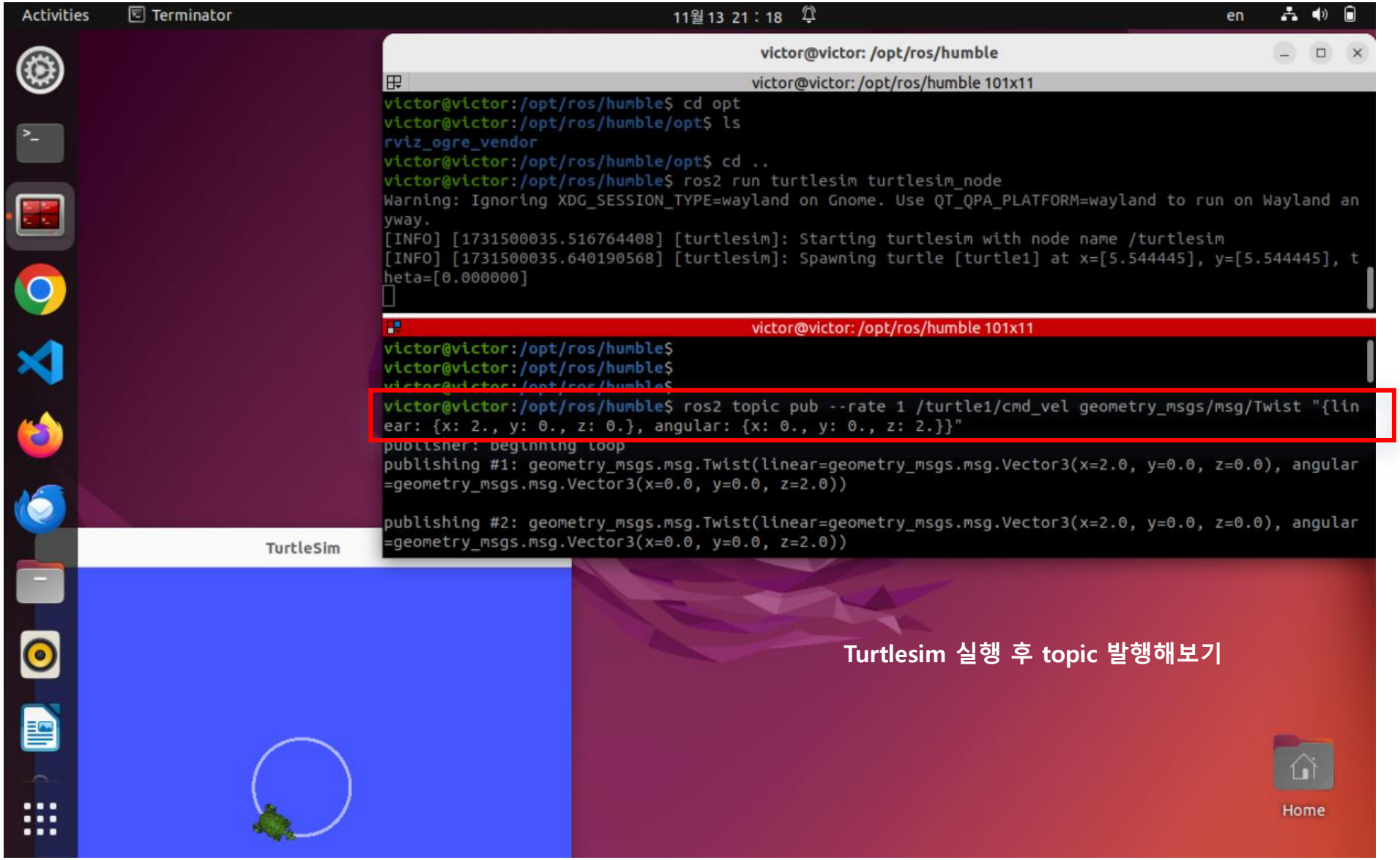
ROS2 실습 - Topic

```
$ 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}}"
```



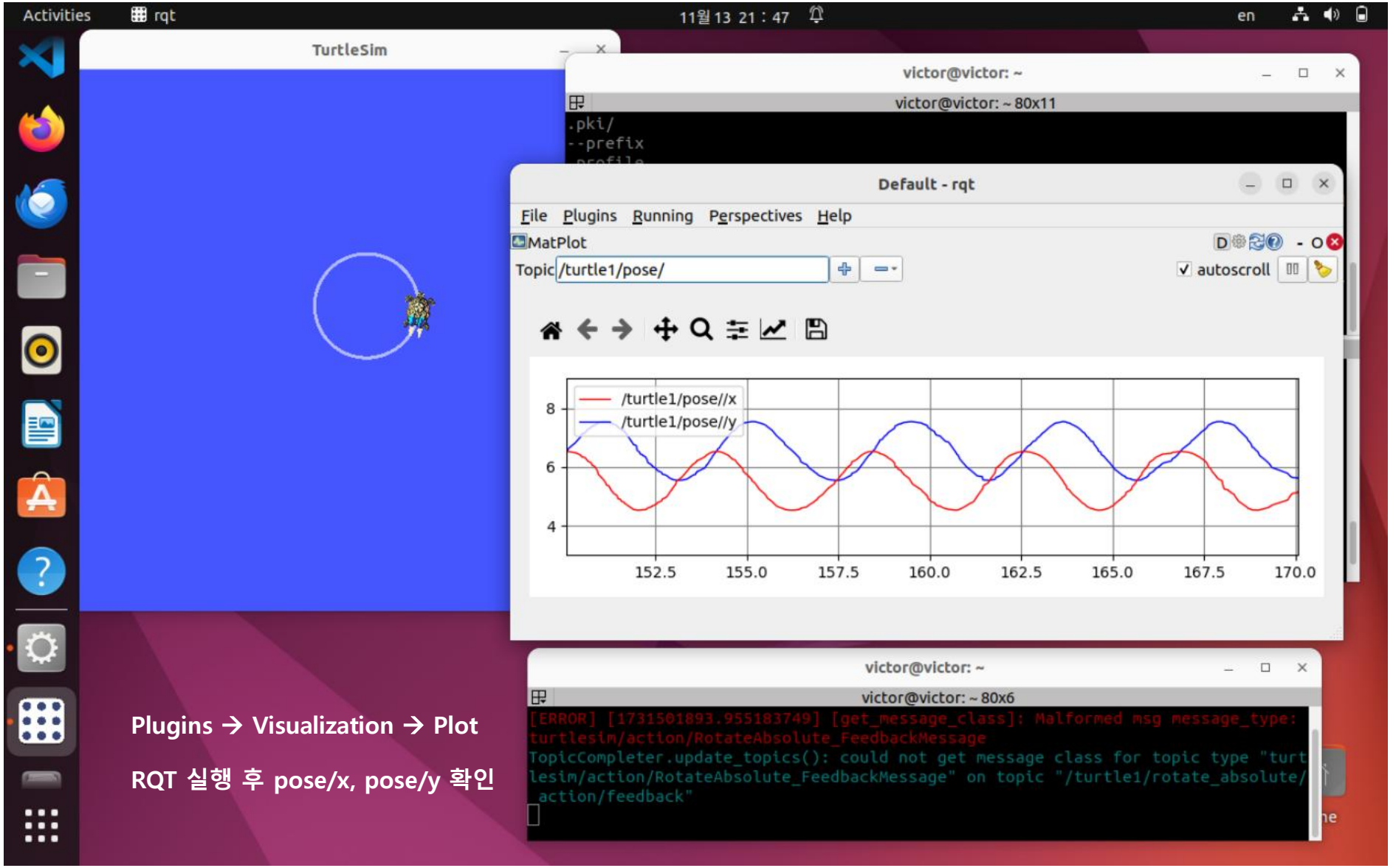


ROS2 실습 - Topic

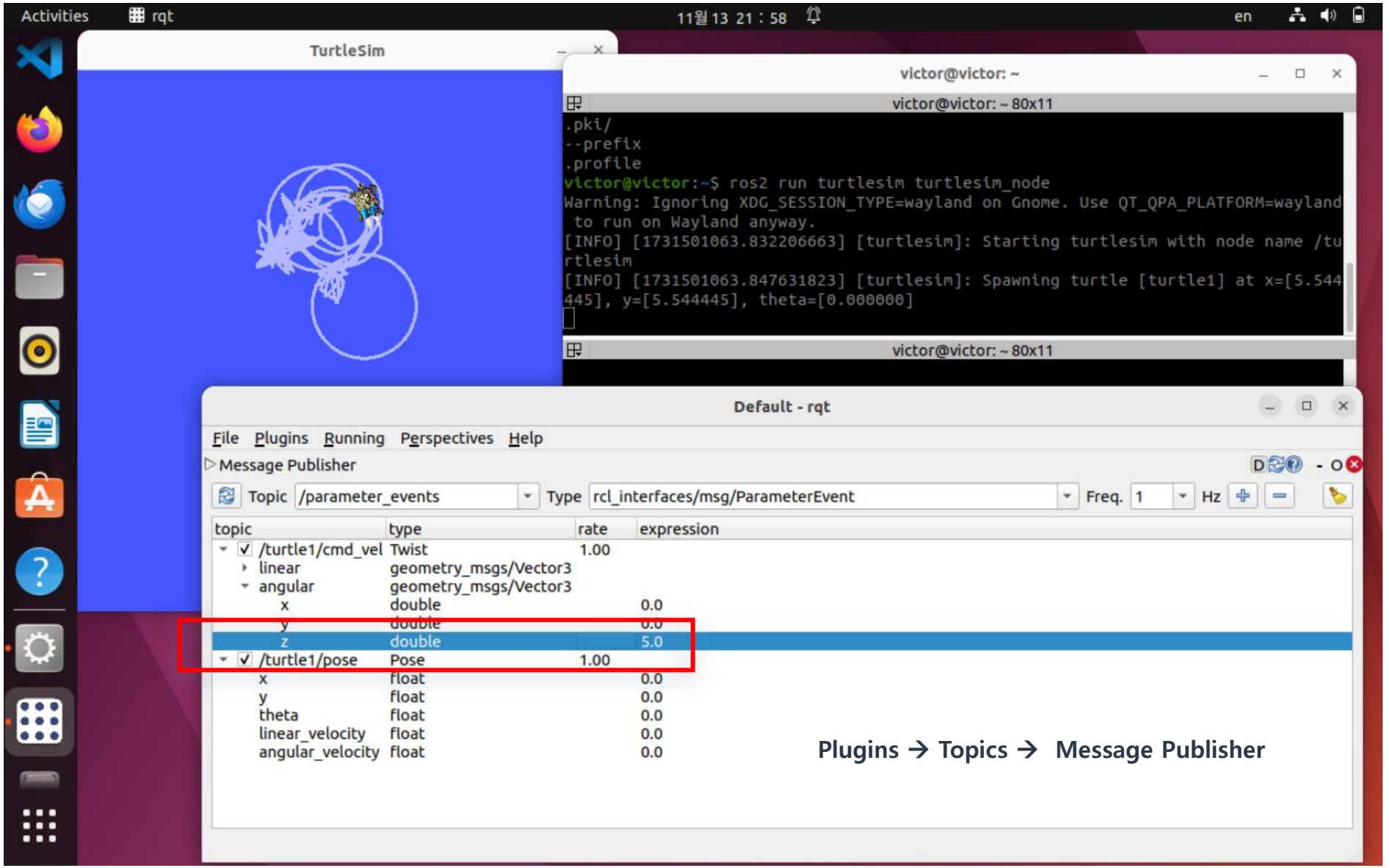


Turtlesim 실행 후 topic 발행해보기

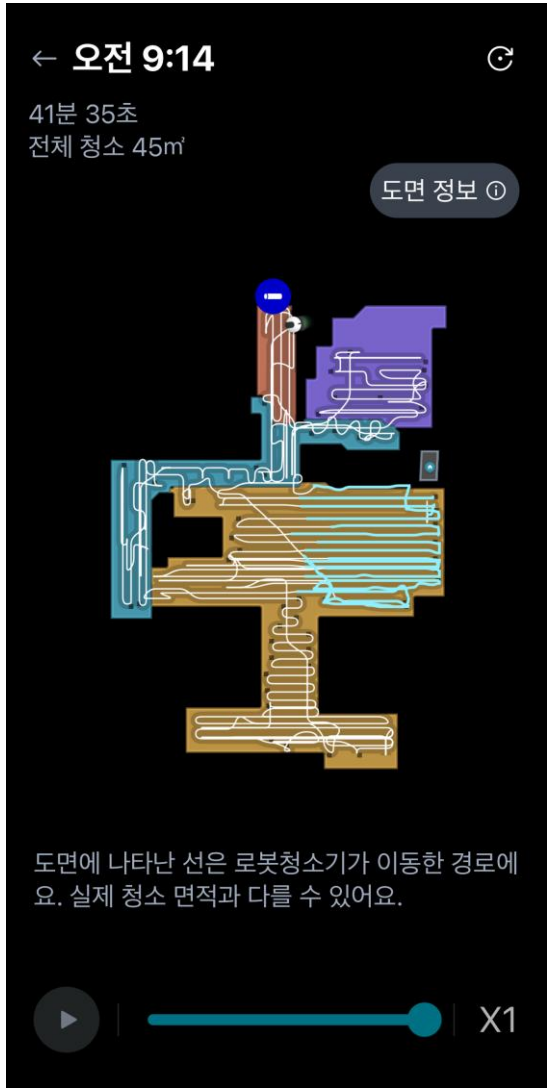
ROS2 실습 – Topic with RQT

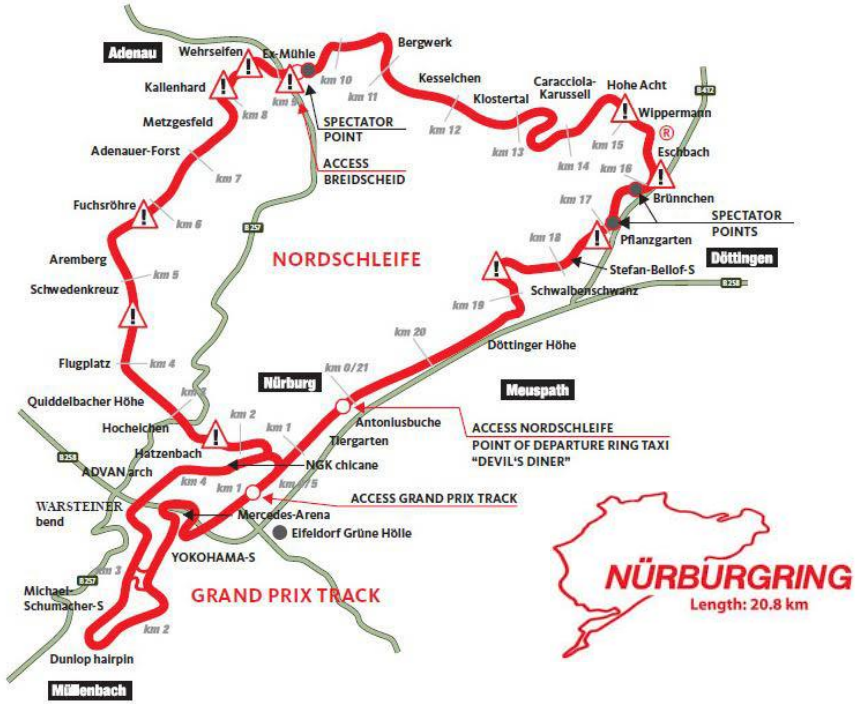


ROS2 실습 – Topic Message with RQT

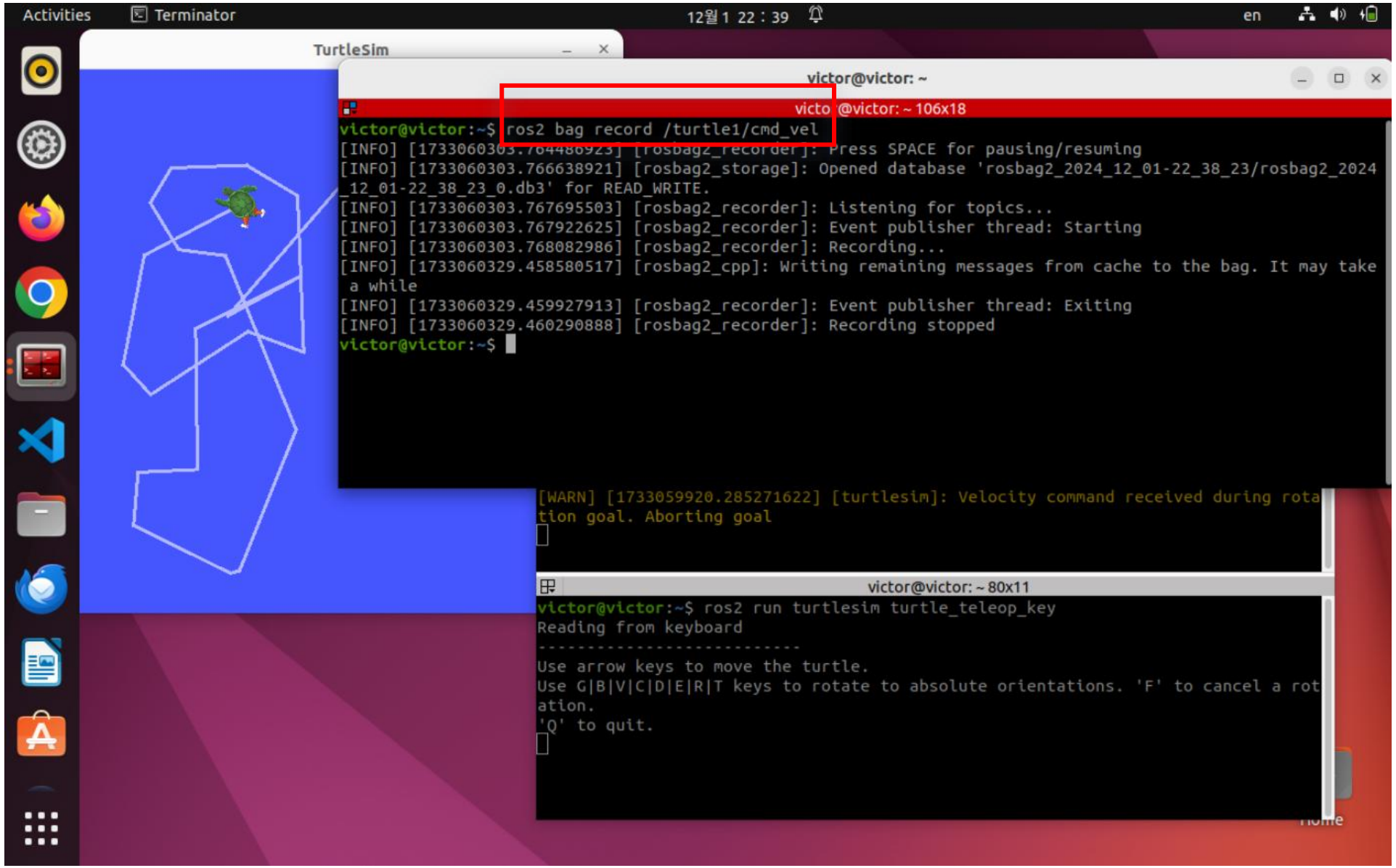


ROS2 실습 – ros bag

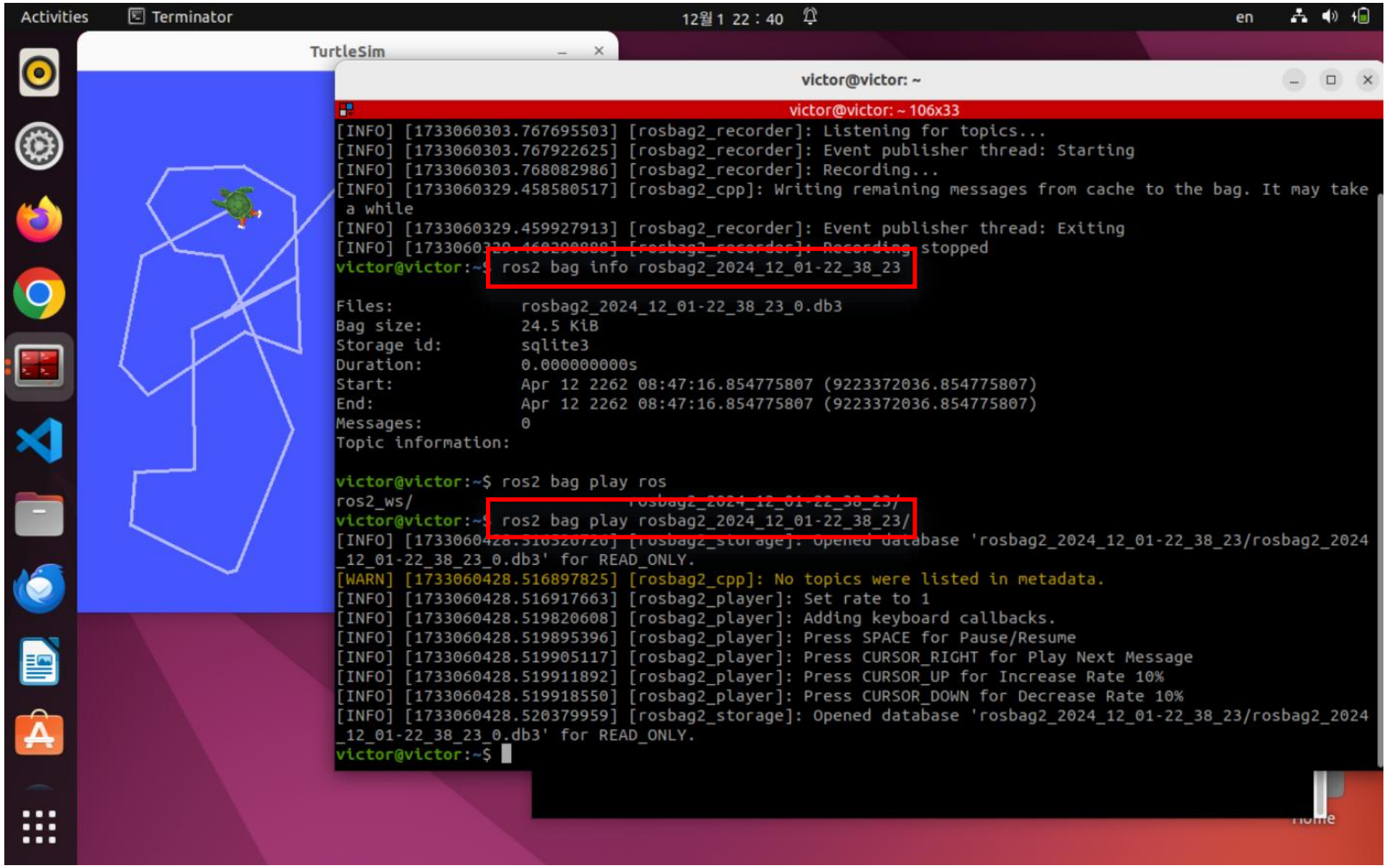




ROS2 실습 – ros2 bag and ros2 play

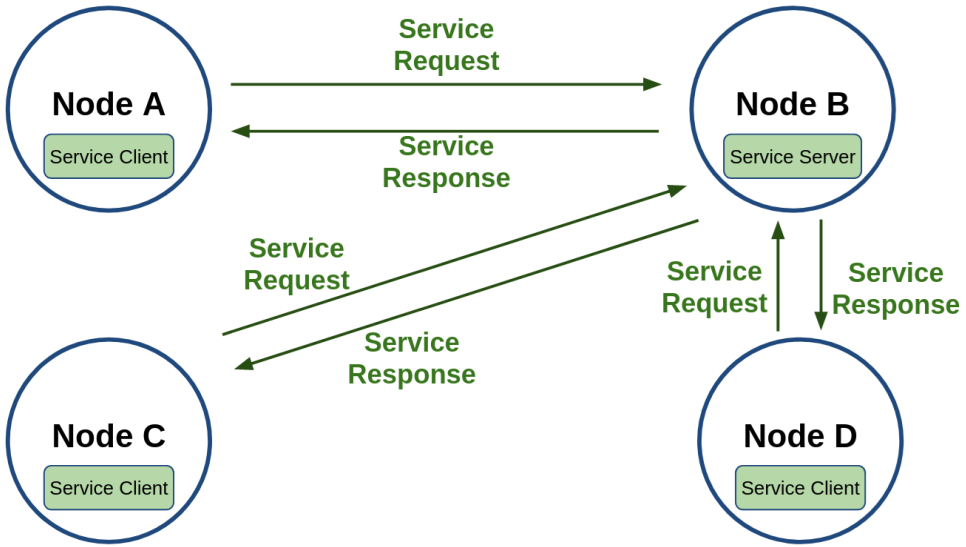


ROS2 실습 – ros2 bag and ros2 play



ROS2 실습 - Service

- 서비스는 다음 그림과 같이 동일 서비스에 대해 복수의 클라이언트를 가질 수 있도록 설계되었다.
- 단, 서비스 응답은 서비스 요청이 있었던 서비스 클라이언트에 대해서만 응답을 하는 형태
- Node C의 Service Client가 Node B의 Service Server에게 서비스 요청을 하였다면 Node B의 Service Server는 요청 받은 서비스를 수행한 후 Node C의 Service Client에게만 서비스 응답



ROS2 실습 - Service

```
$ ros2 run turtlesim turtlesim_node
```

```
$ ros2 service list
/clear
/kill
/reset
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

ROS2 실습 - Service

```
$ ros2 service type /clear
std_srvs/srv/Empty

$ ros2 service type /kill
turtlesim/srv/Kill

$ ros2 service type /spawn
turtlesim/srv/Spawn
```

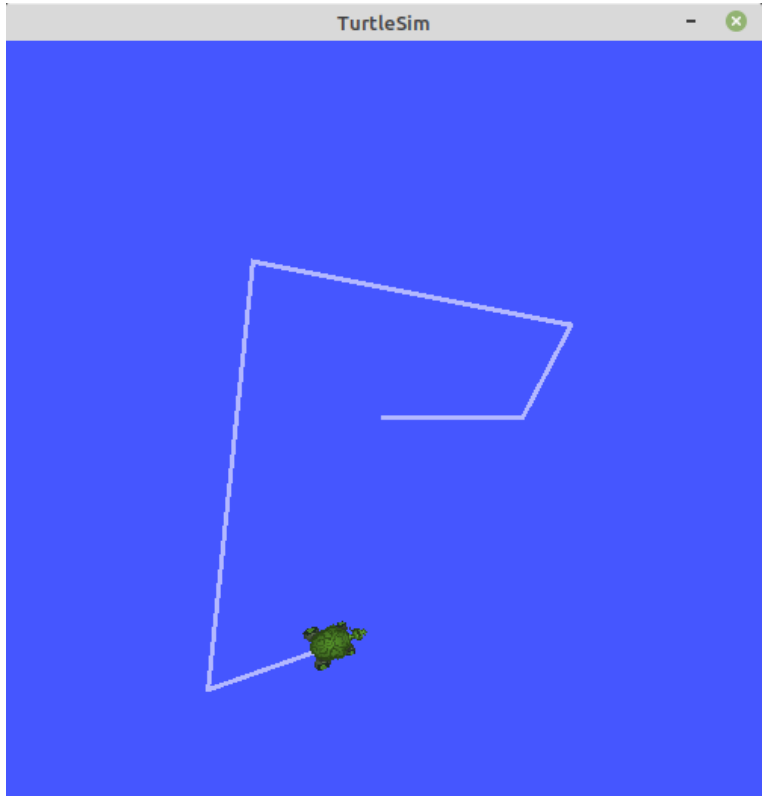
```
$ ros2 service list -t
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/reset [std_srvs/srv/Empty]
/spawn [turtlesim/srv/Spawn]
/turtle1/set_pen [turtlesim/srv/SetPen]
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]
/turtle1/teleport_relative [turtlesim/srv/TeleportRelative]
(생략)
```

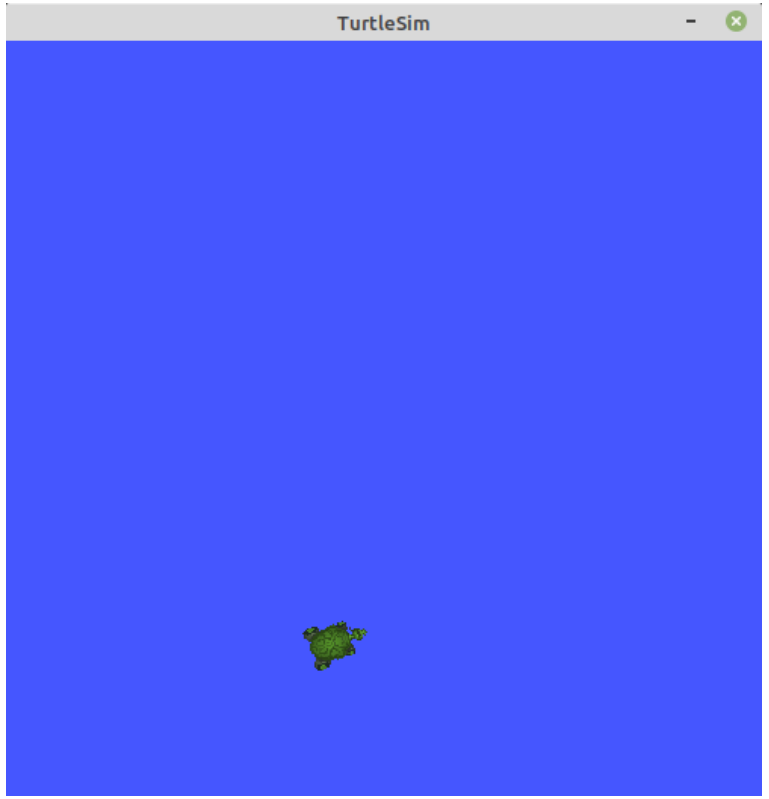
```
$ ros2 service find std_srvs/srv/Empty
/clear
/reset

$ ros2 service find turtlesim/srv/Kill
/kill
```

ROS2 실습 - Service

```
$ ros2 run turtlesim turtle_teleop_key
```

```
$ ros2 service call /clear std_srvs/srv/Empty
requester: making request: std_srvs.srv.Empty_Request()
response:
std_srvs.srv.Empty_Response()
```

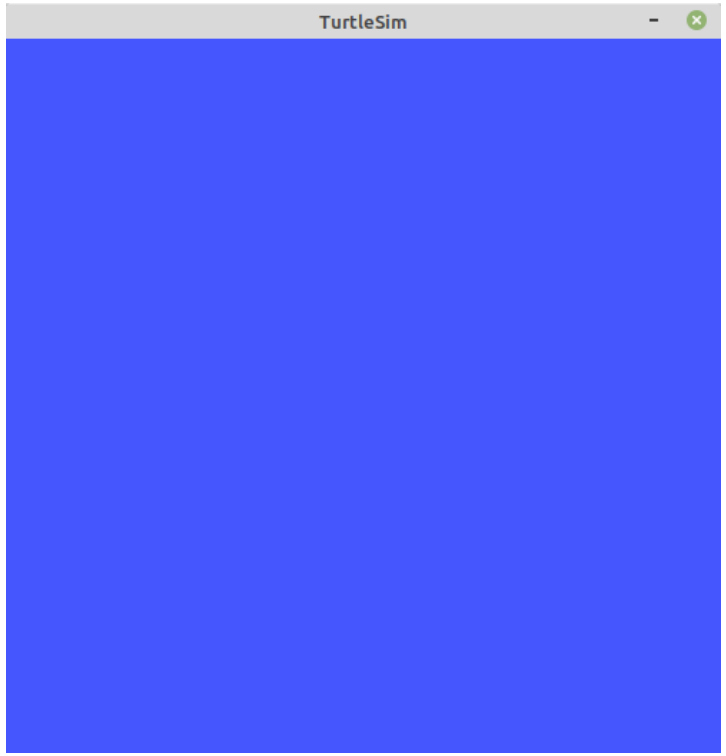


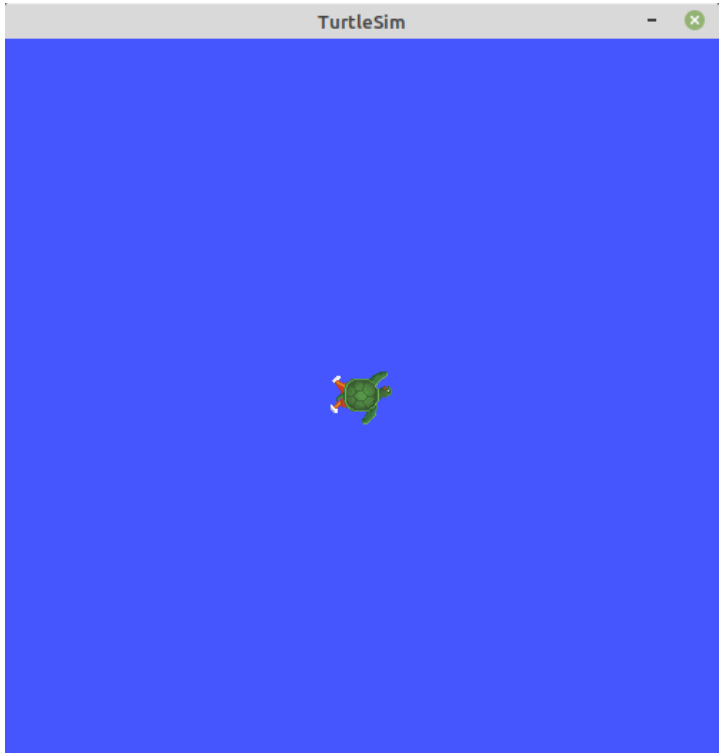


ROS2 실습 - Service

```
$ ros2 service call /kill turtlesim/srv/Kill "name: 'turtle1'"
requester: making request:
turtlesim.srv.Kill_Request(name='turtle1')
response:
turtlesim.srv.Kill_Response()
```

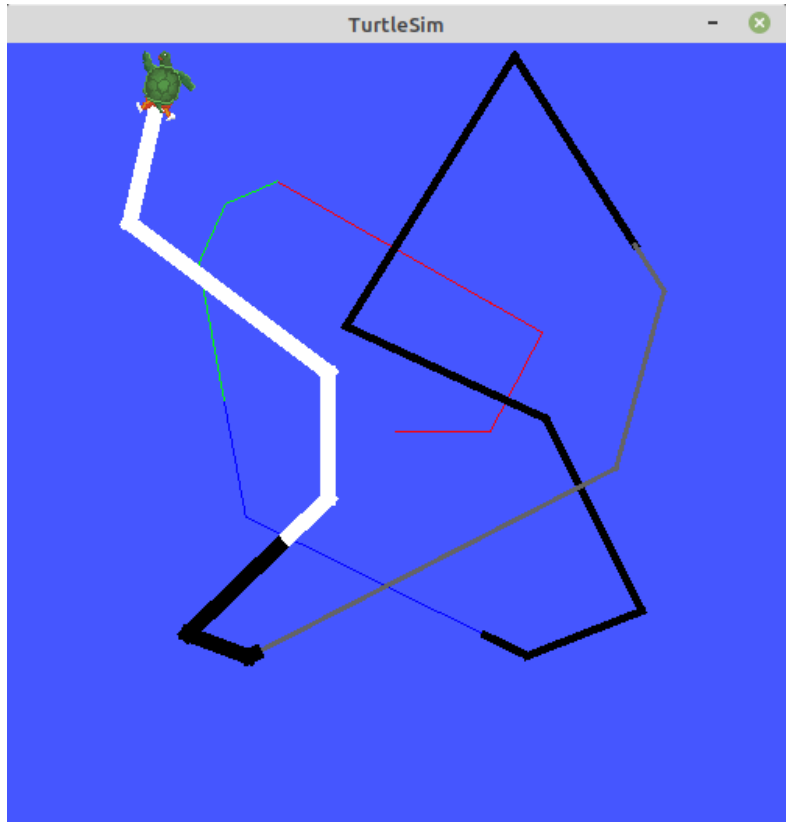
```
$ ros2 service call /reset std_srvs/srv/Empty
requester: making request: std_srvs.srv.Empty_Request()
response:
std_srvs.srv.Empty_Response()
```



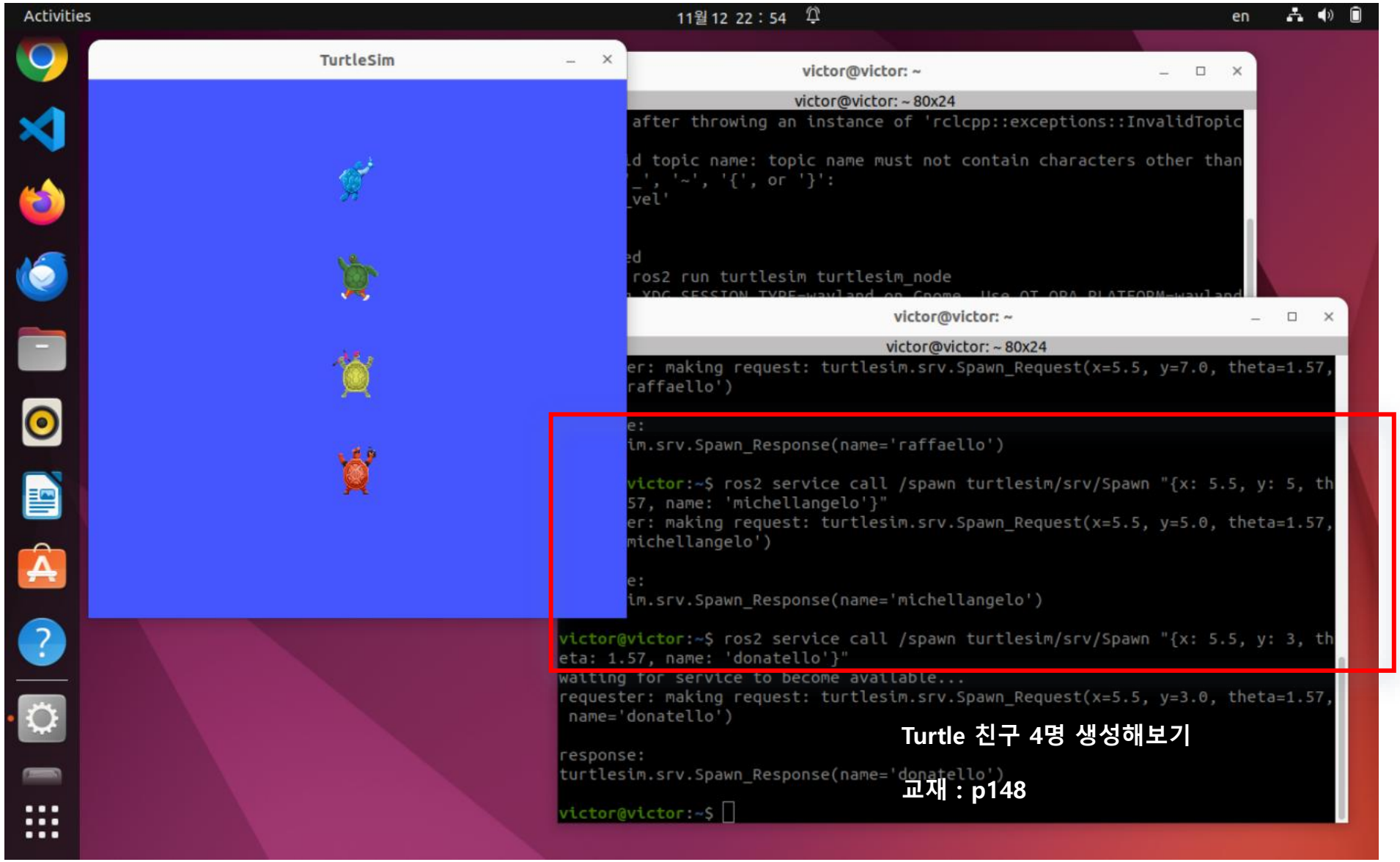


ROS2 실습 - Service

```
$ ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r: 255, g: 255, b: 255, width: 10}"
requester: making request: turtlesim.srv.SetPen_Request(r=255, g=255, b=255, width=10, off=0)
response:
turtlesim.srv.SetPen_Response()
```



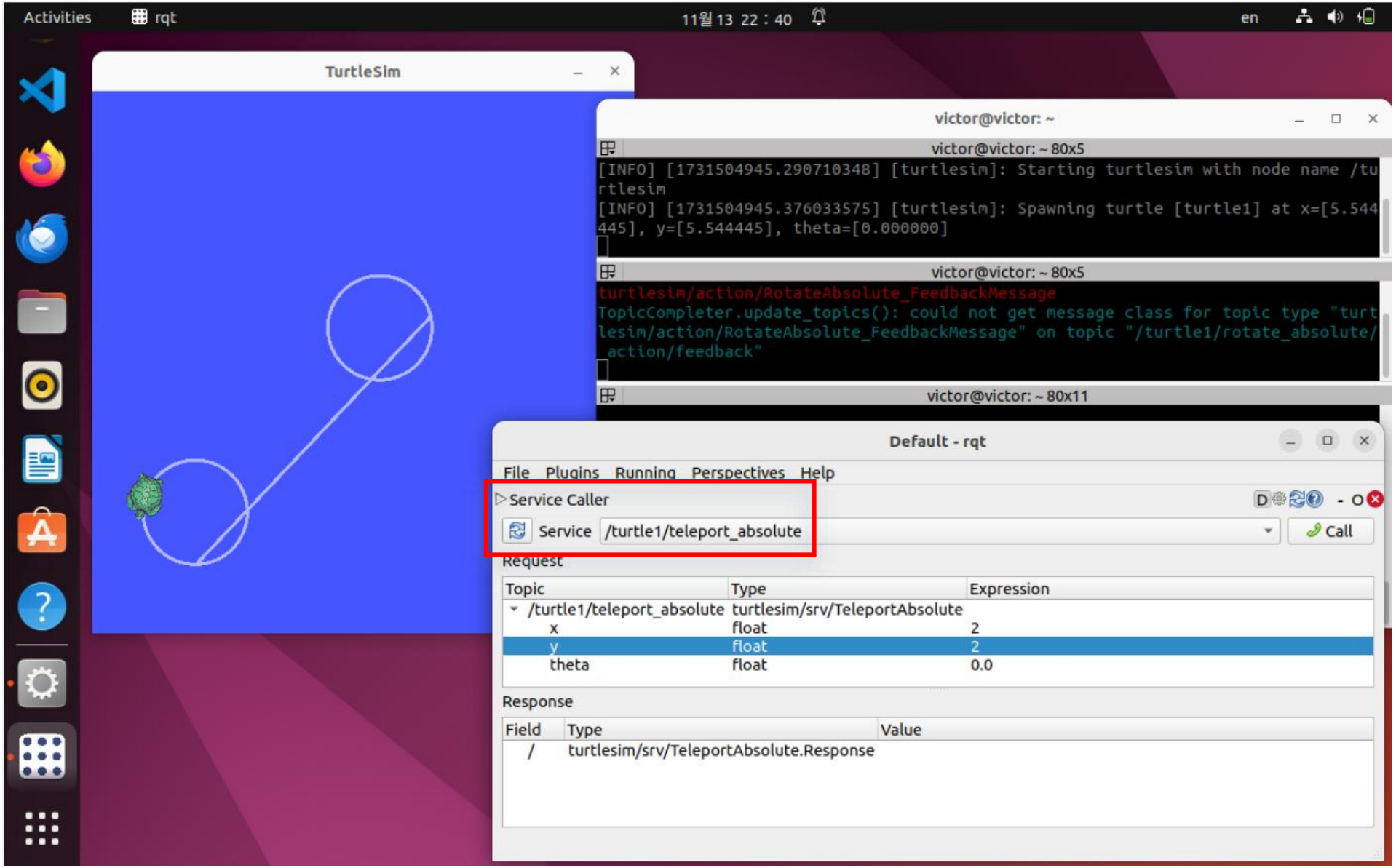
ROS2 실습 - Service



Turtle 친구 4명 생성해보기

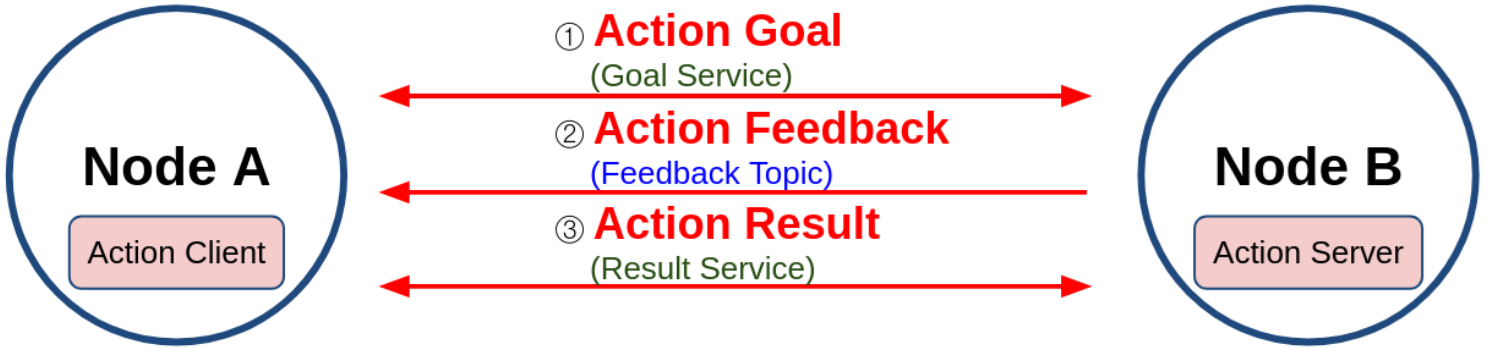
교재 : p148

ROS2 실습 – Service with RQT



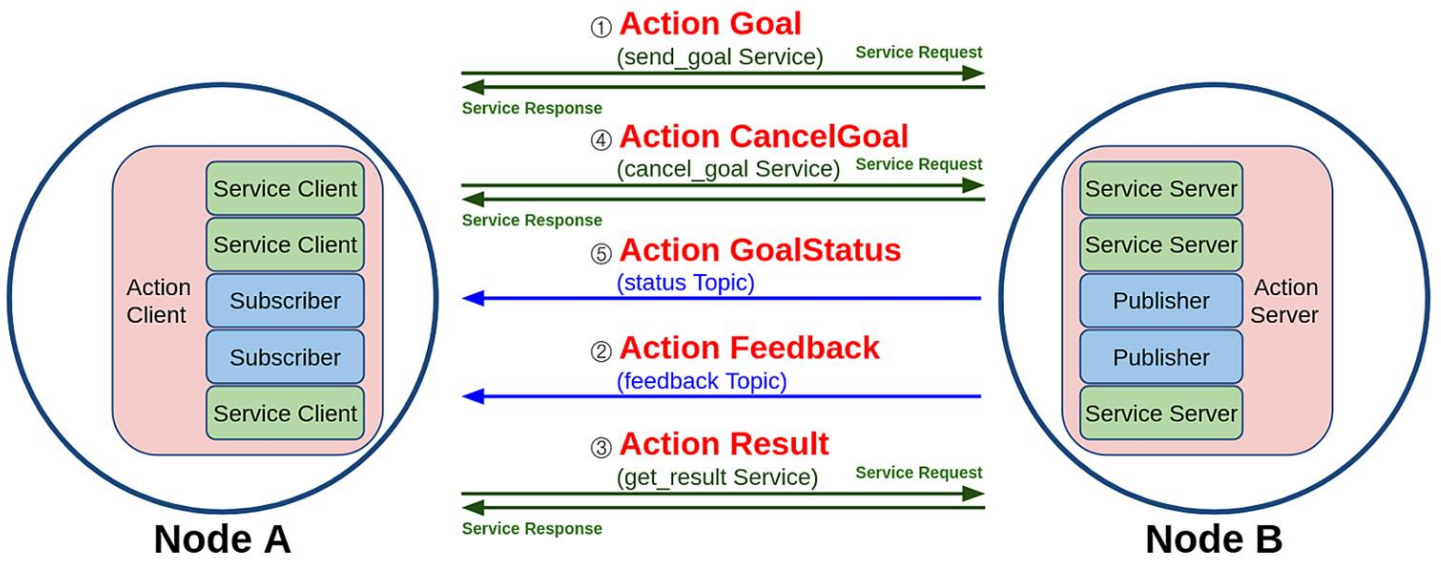
액션(Action)

- 토픽(topic)과 서비스(service)의 혼합
- 액션 목표 및 액션 결과를 전달하는 방식은 서비스와 같고,
- 액션 피드백은 토픽과 같은 메시지 전송 방식
- 액션 목표/피드백/결과(Goal/Feedback/Result) 메시지 또한 위에서 언급한 msg 메시지의 변형으로 action 메시지라고 함.



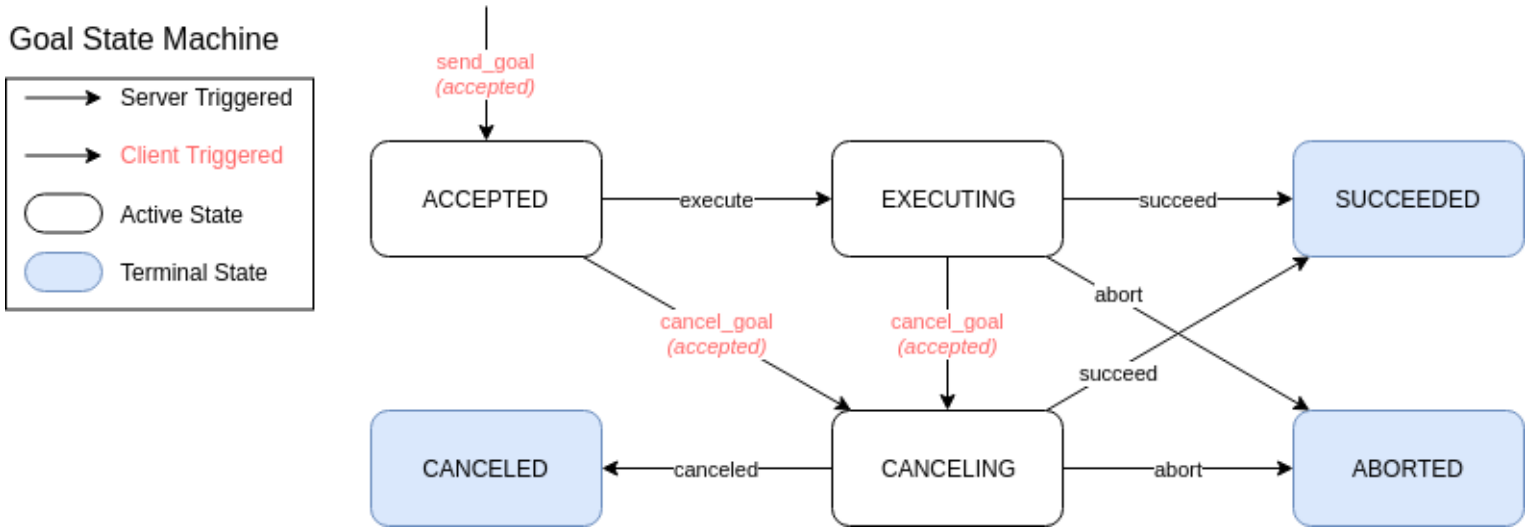
ROS2 실습 - Action

액션의 구현 방식을 더 자세히 살펴보면 다음 그림과 같이 토픽과 서비스의 혼합이라고 볼 수 있는데 ROS 1이 토픽만을 사용하였다면 ROS 2에서는 액션 목표, 액션 결과, 액션 피드백은 토픽과 서비스가 혼합되어 있다.
즉, 다음 그림과 같이 Action Client는 Service Client 3개와 Topic Subscriber 2개로 구성되어있으며, Action Server는 Service Server 3개와 Topic Publisher 2개로 구성된다. 액션 목표/피드백/결과(goal/feedback/result) 데이터는 msg 및 srv 인터페이스의 변형으로 action 인터페이스라고 한다.



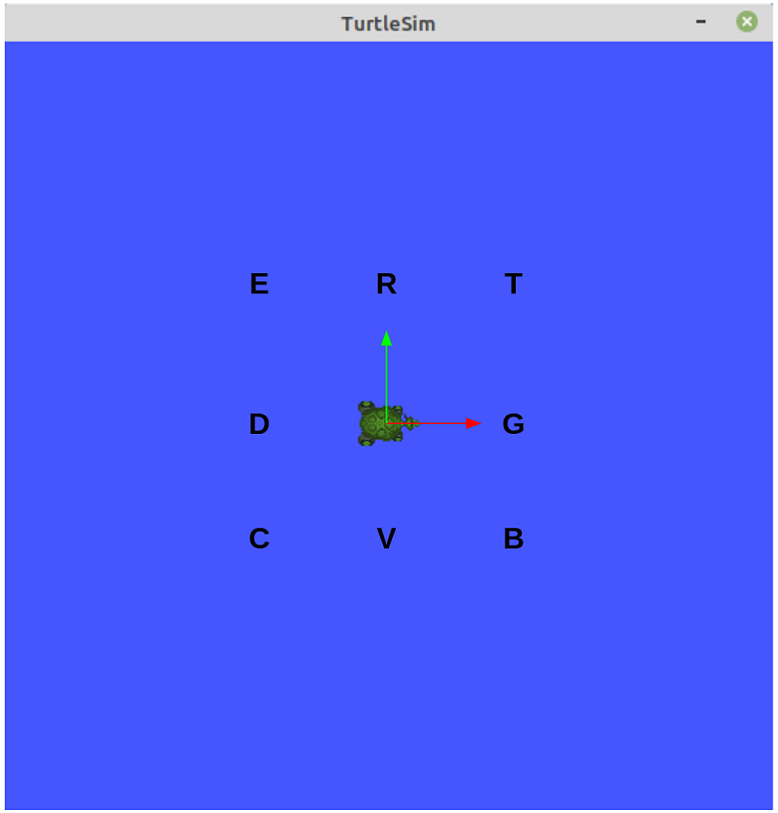
ROS2 실습 - Action

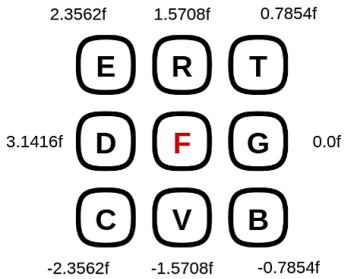
ROS 1에서의 액션은 목표, 피드백, 결과 값을 토픽으로만 주고 받았는데 **ROS 2에서는 토픽과 서비스 방식을 혼합하여 사용**하였다. 그 이유로 토픽으로만 액션을 구성하였을 때 토픽의 특징인 비동기식 방식을 사용하게 되어 ROS 2 액션에서 새롭게 선보이는 **목표 전달(send_goal), 목표 취소 (cancel_goal), 결과 받기 (get_result)를 동기식인 서비스를 사용하기 위해서**이다. 이런 비동기 방식을 이용하다보면 원하는 타이밍에 적절한 액션을 수행하기 어려운데 이를 원활히 구현하기 위하여 **목표 상태(goal_state)**라는 것이 ROS 2에서 새롭게 선보였다. 목표 상태는 **목표 값을 전달 한 후의 상태 머신을 구동하여 액션의 프로스세를 쫓는 것**이다. 여기서 말하는 상태머신은 **Goal State Machine**으로 다음 그림과 같이 액션 목표 전달 이후의 액션의 상태 값을 액션 클라이언트에게 전달할 수 있어서 비동기, 동기 방식이 혼재된 액션의 처리를 원활하게 할 수 있게 되어 있다.

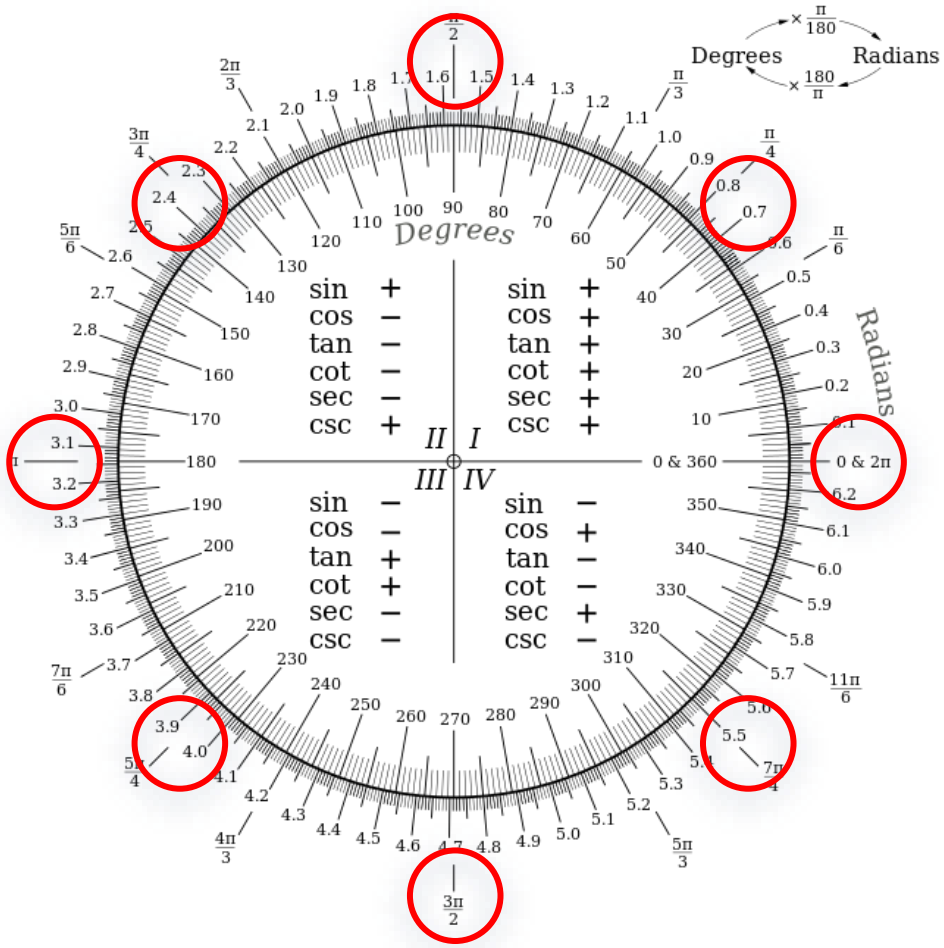


ROS2 실습 - Action

교재 p153



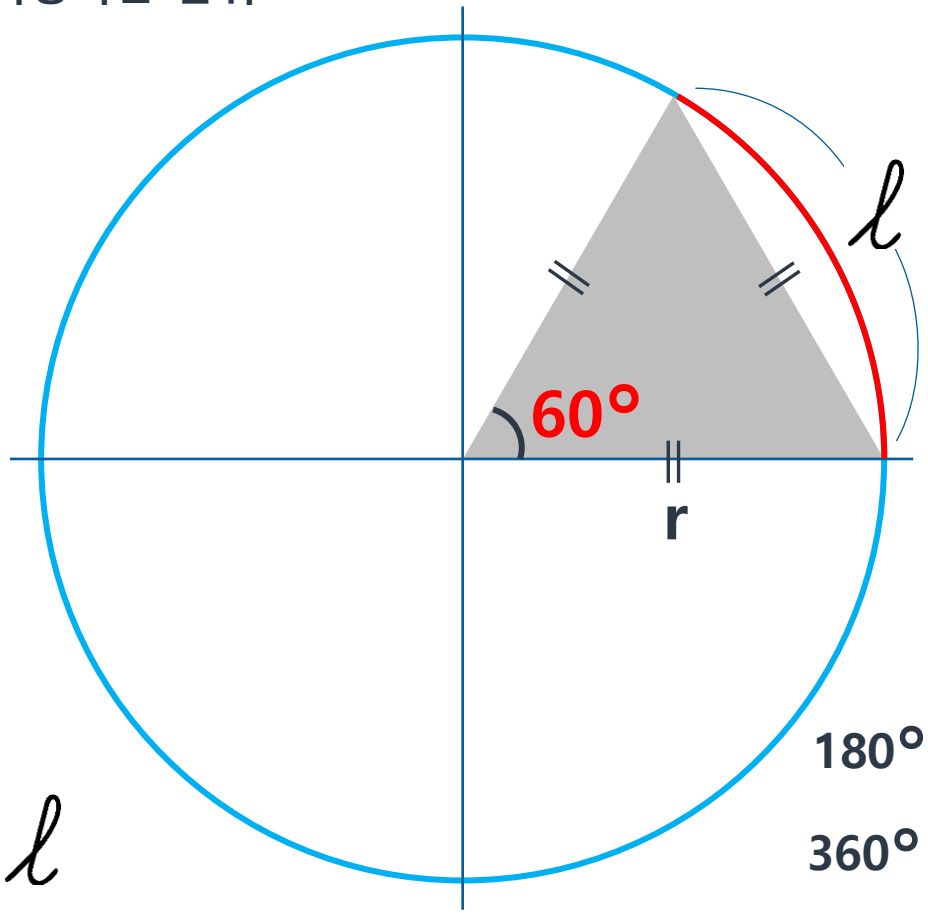




Degree vs Radian

Ros2에서 사용하는 단위

- Kg
- Sec
- Meter
- Radian

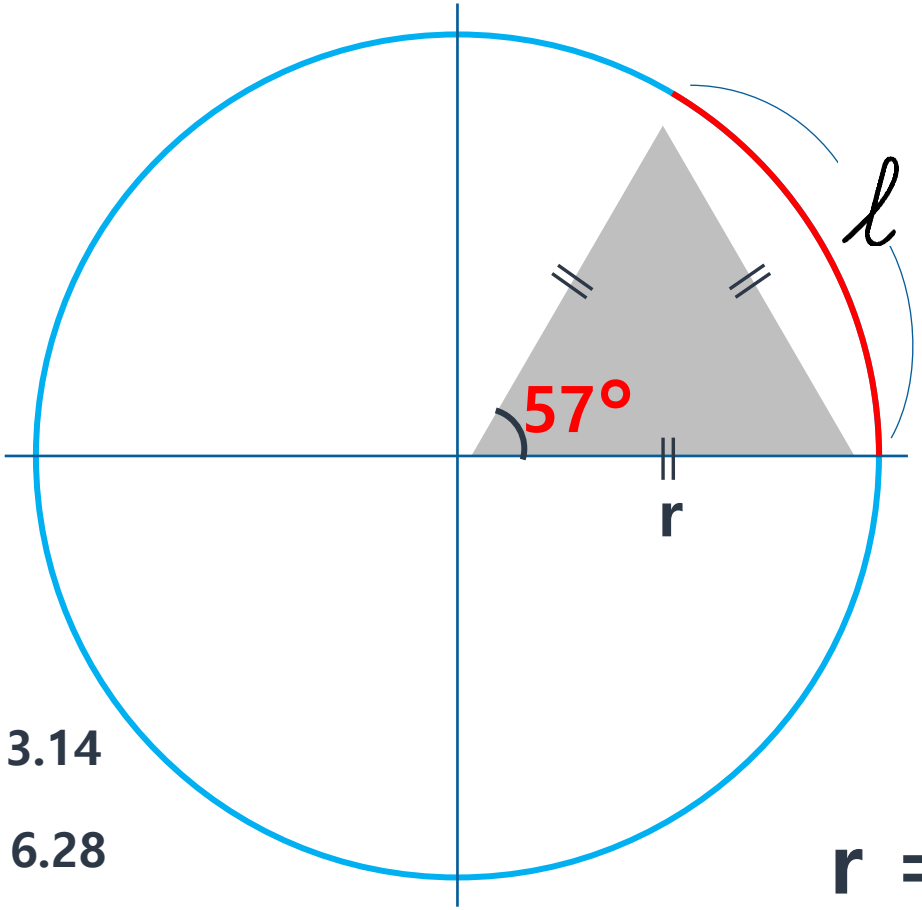


$r < \ell$

$180° = \pi = 3.14$

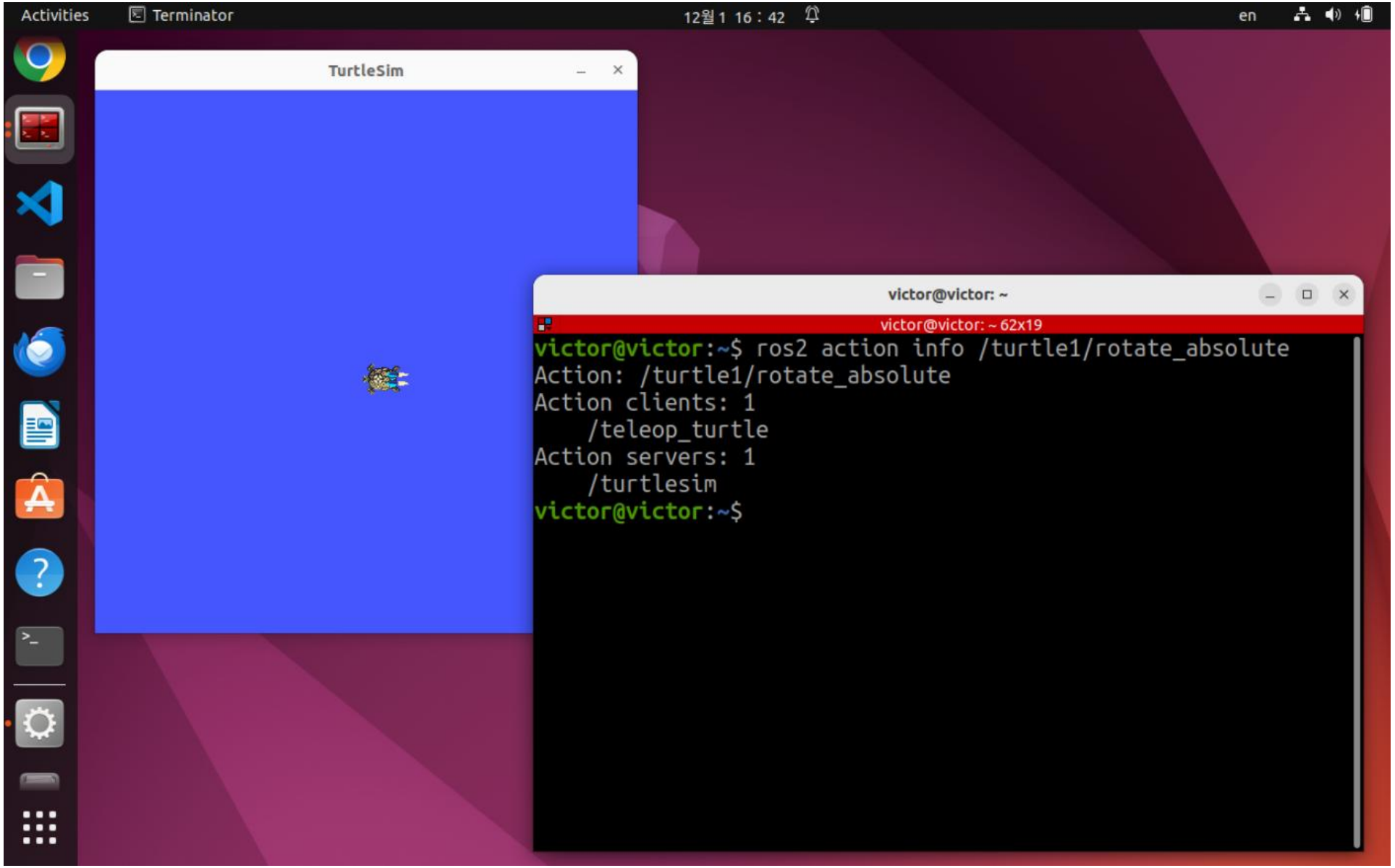
$360° = 2\pi = 6.28$

1 rad ≒ 180°/ 3.14 ≒ 57°

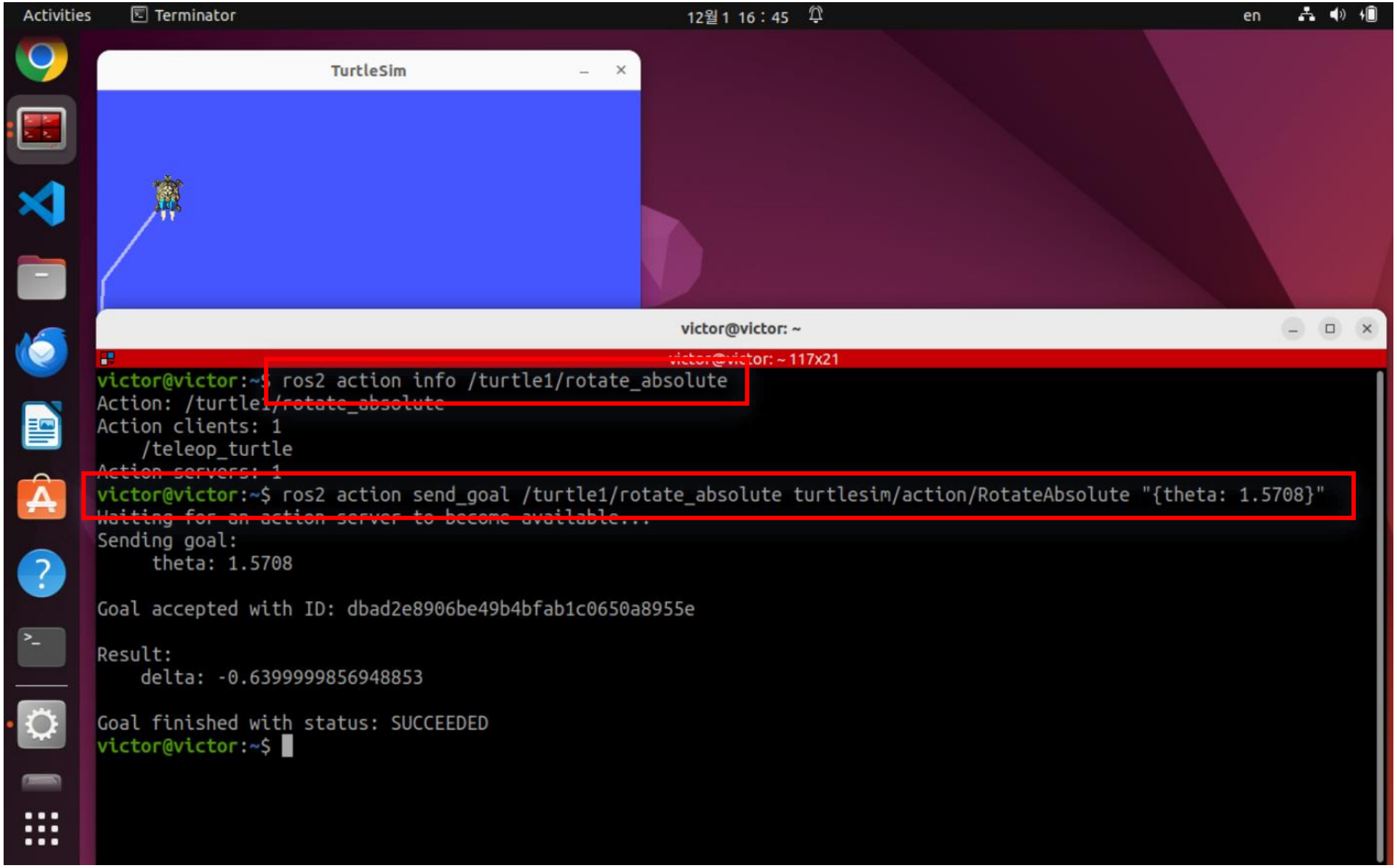


$r = \ell$

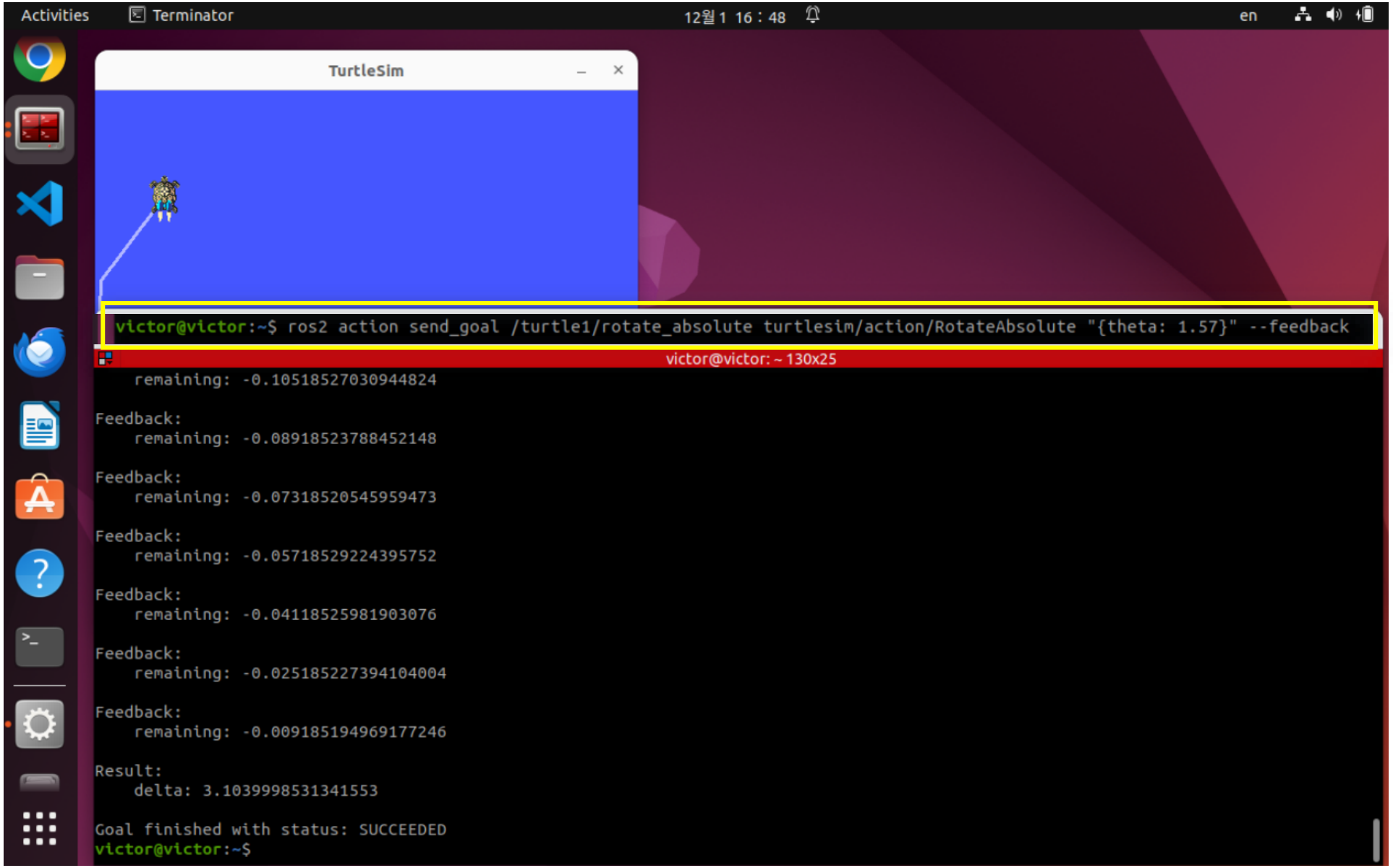
ROS2 실습 - Action



ROS2 실습 - Action



ROS2 실습 - Action



ROS2 실습 - Interface

ROS의 노드 간에 데이터를 주고받을 때에는 **토픽, 서비스, 액션이 사용되는데 이 때 사용되는 데이터의 형태를** [ROS 인터페이스(interface)] 이라고 한다. ROS 인터페이스에는 ROS 2에 새롭게 추가된 **IDL**(interface definition language)과 ROS 1부터 ROS 2까지 널리 사용 중인 **msg**, **srv**, **action** 이 있다. 토픽, 서비스, 액션은 각각 msg, srv, action interface를 사용하고 있으며 정수, 부동 소수점, 불리언과 같은 단순 자료형을 기본으로 하여 메시지 안에 메시지를 품고 있는 간단한 데이터 구조 및 메시지들이 나열된 배열과 같은 구조도 사용할 수 있다.

- Bool
- Byte
- Char
- Float32
- Float64
- Int8
- Int16
- Int32
- Int64
- String
- UInt8
- UInt16
- UInt32
- UInt64

[**단순 자료형**]

- 예) 정수(integer), 부동 소수점(floating point), 불(boolean)
- https://github.com/ros2/common_interfaces/tree/humble/std_msgs

[**메시지 안에 메시지를 품고 있는 간단한 데이터 구조**]

- 예) geometry_msgs/msgs/Twist의 `Vector3 linear`
- https://github.com/ros2/common_interfaces/blob/humble/geometry_msgs/msg/Twist.msg

```
1    # This expresses velocity in free space broken into
2
3    Vector3  linear
4    Vector3  angular
```

[**메시지들이 나열된 배열과 같은 구조**]

- 예) sensor_msgs/msgs/LaserScan 의 `float32[] ranges`
- https://github.com/ros2/common_interfaces/blob/humble/sensor_msgs/msg/LaserScan.msg

```
float64 x
float64 y
float64 z
```

common_interfaces / geometry_msgs / msg / **Vector3.msg**

```
float32 angle_min
float32 angle_max
float32 angle_increment

float32 time_increment


float32 scan_time

float32 range_min
float32 range_max

float32[] ranges

float32[] intensities
```

ROS2 실습 - Interface

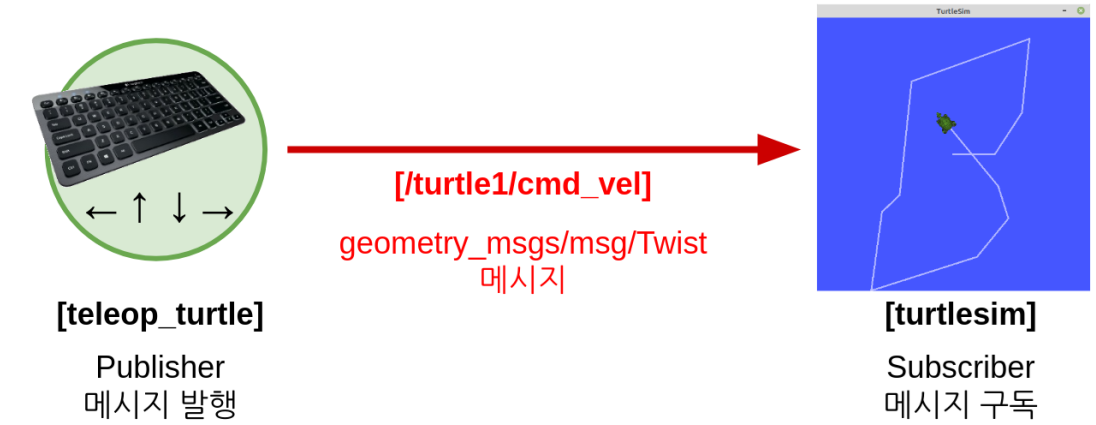
토픽은 고유의 인터페이스를 가지고 있는데 이를 **메시지 인터페이스**라 부르며, 파일로는 **msg 파일**을 가르킨다.
예를 들어, 위 예제에서 /turtle1/cmd_vel 토픽은 geometry_msgs/msgs/Twist 형태이다. 이름이 좀 긴데 풀어서 설명하면 기하학 관련 메시지를 모아둔 geometry_msgs 패키지의 msgs 분류의 Twist 데이터 형태라는 것이다.

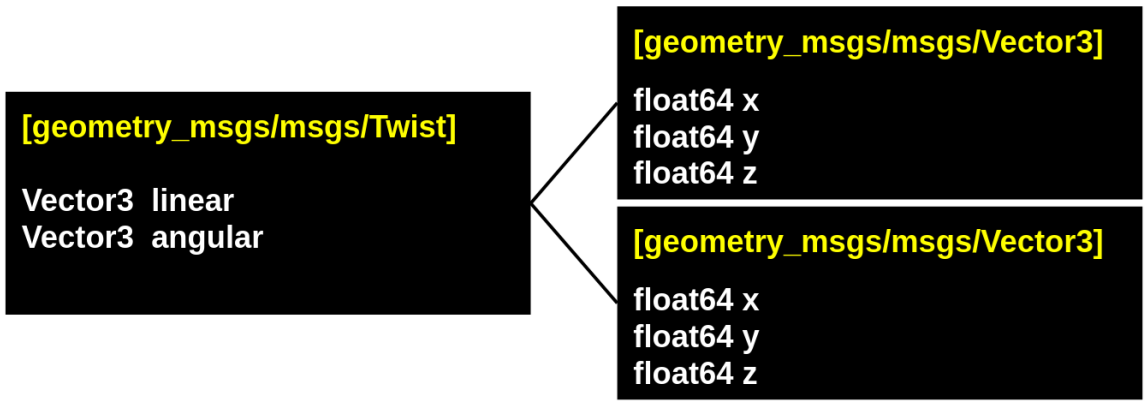
토픽은 고유의 인터페이스를 가지고 있는데 이를 **메시지 인터페이스**라 부르며, 파일로는 **msg 파일**을 가르킨다.

예를 들어, 위 예제에서 /turtle1/cmd_vel 토픽은 geometry_msgs/msgs/Twist 형태이다. 이름이 좀 긴데 풀어서 설명하면 기하학 관련 메시지를 모아둔 geometry_msgs 패키지의 msgs 분류의 Twist 데이터 형태라는 것이다.

Twist 데이터 형태를 자세히 보면 Vector3 linear과 Vector3 angular 이라고 되어 있다. 이는 메시지 안에 메시지를 품고 있는 것으로 Vector3 형태에 linear 이라는 이름의 메시지와 Vector3 형태에 angular 이라는 이름의 메시지, 즉 2개의 메시지가 있다는 것이며 Vector3는 다시 float64 형태에 x, y, z 값이 존재한다.

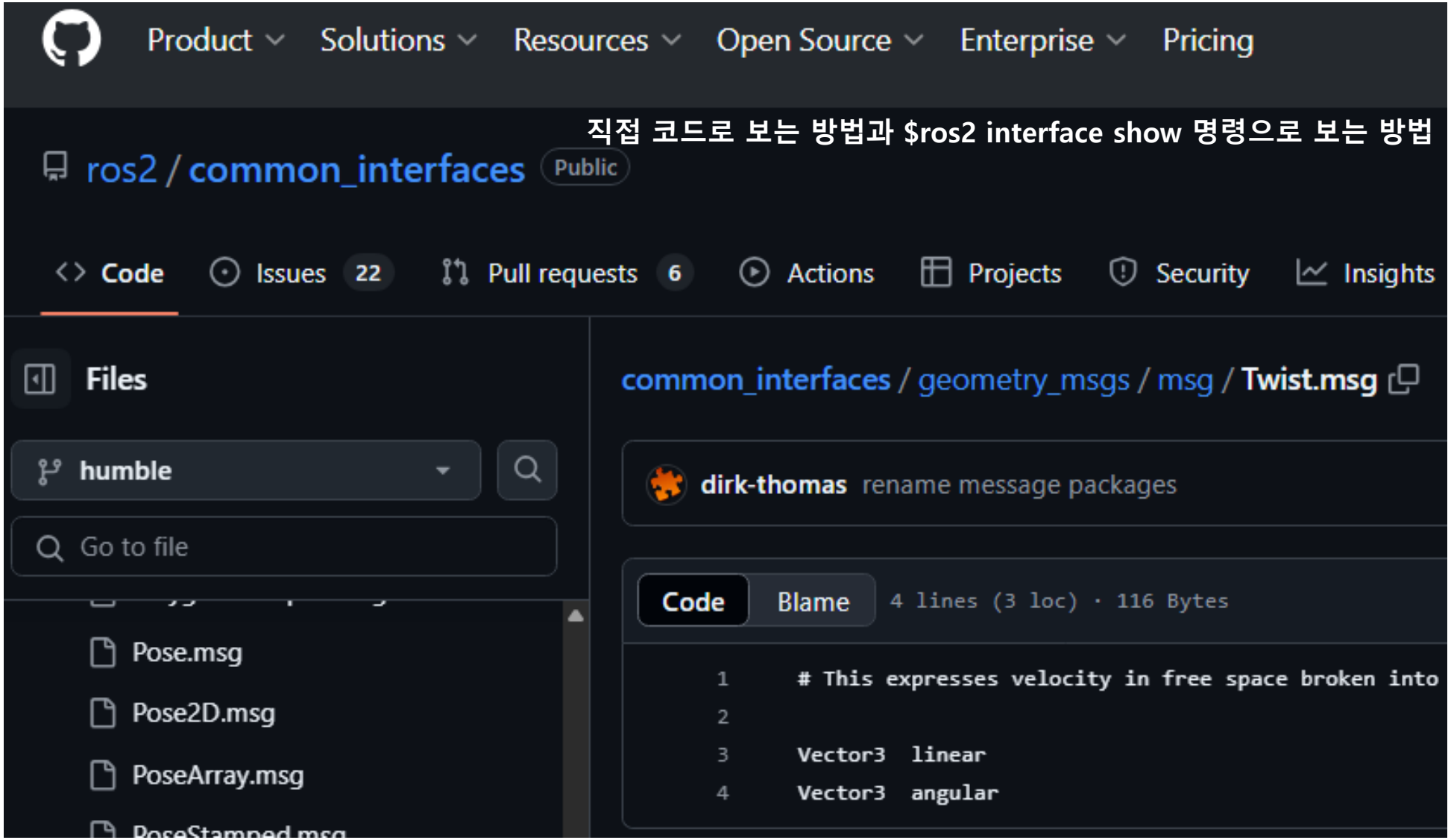
다시 말해 **geometry_msgs/msgs/Twist 메시지 형태는 float64 자료형의 linear.x, linear.y, linear.z, angular.x, angular.y, angular.z 라는 이름의 메시지인 것이다. 이를 통해 병진 속도 3개, 회전 속도 3개를 표현할 수 있게 된다.**



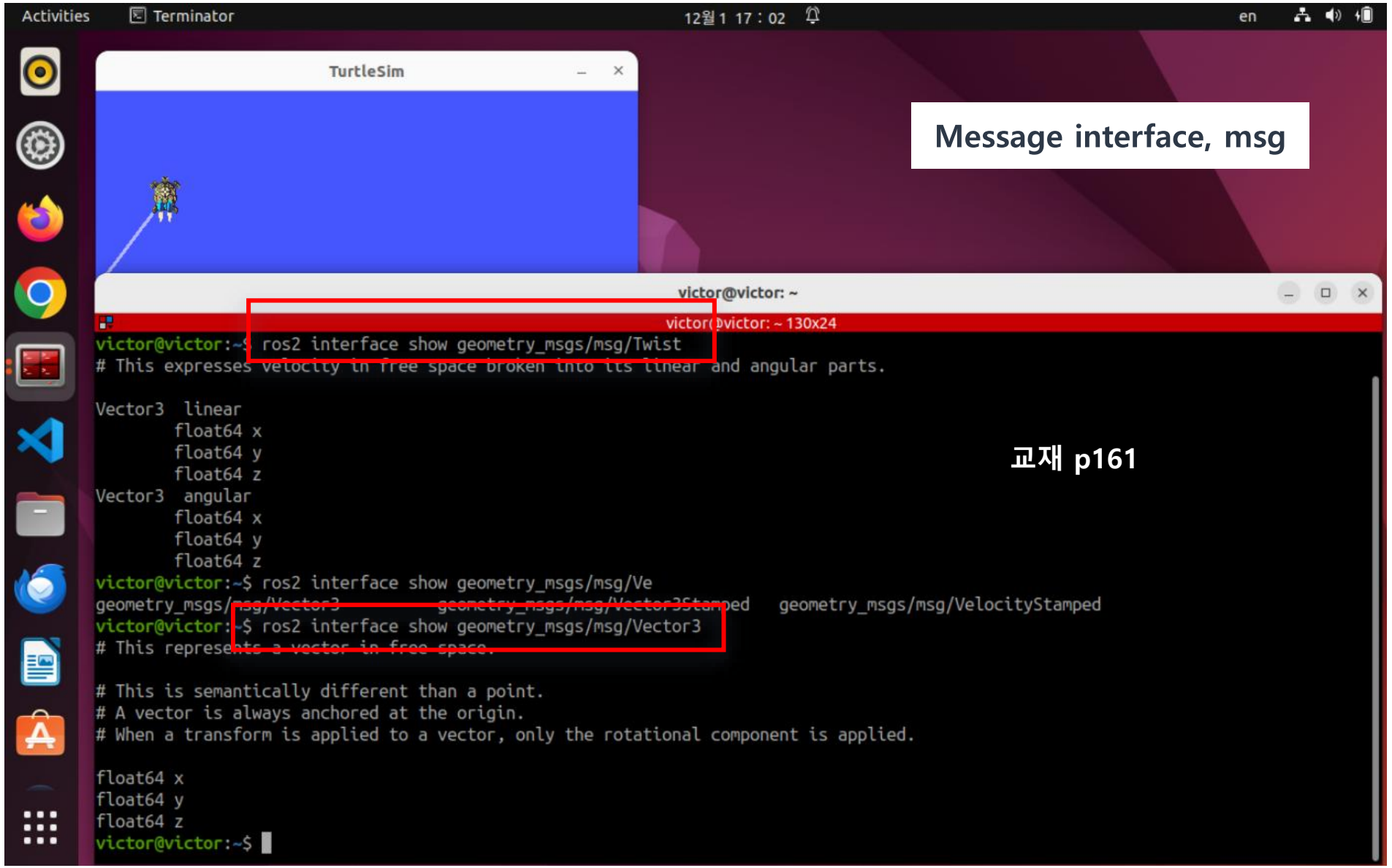


ROS2 실습 - Interface

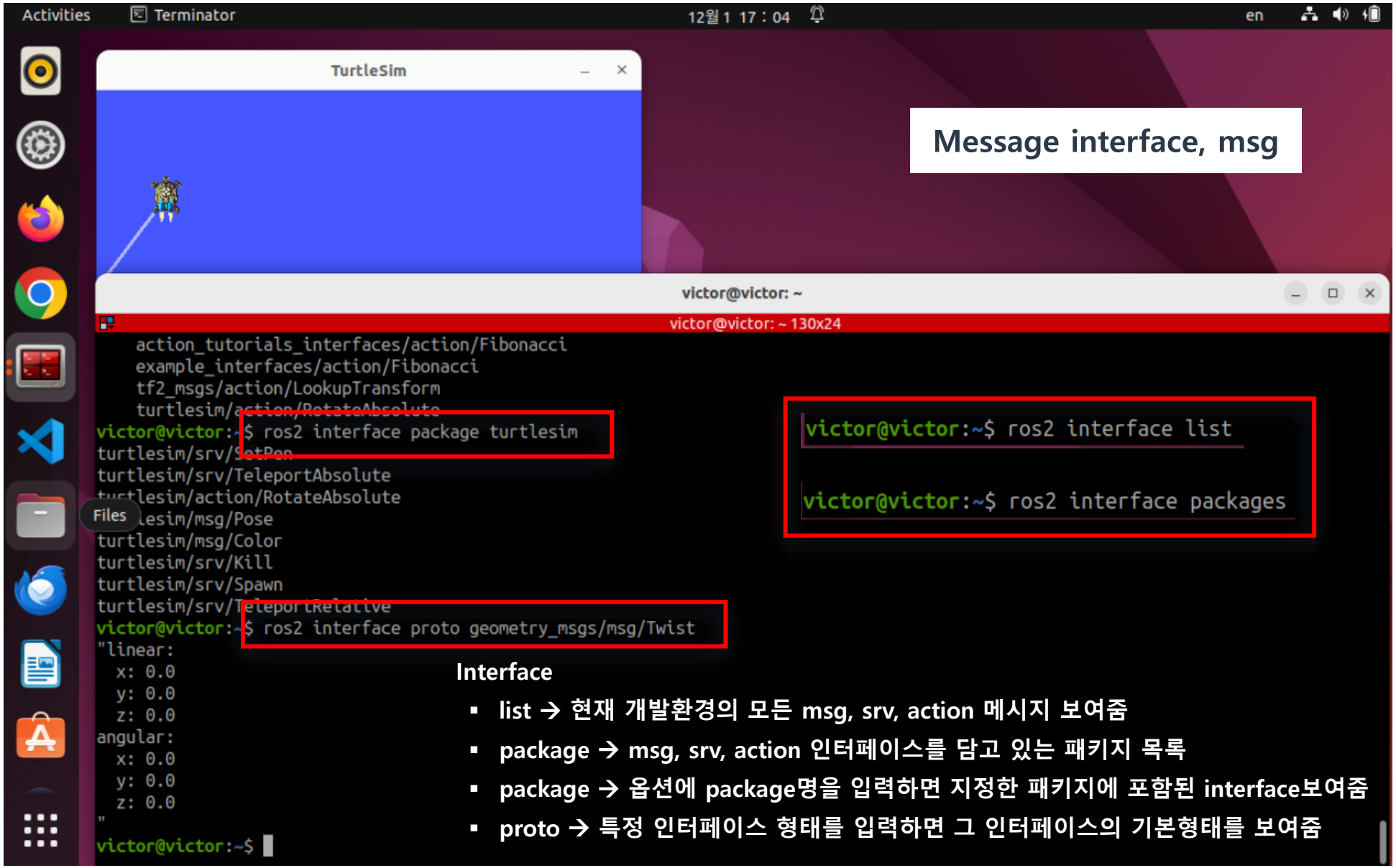
교재 p161



ROS2 실습 - Interface



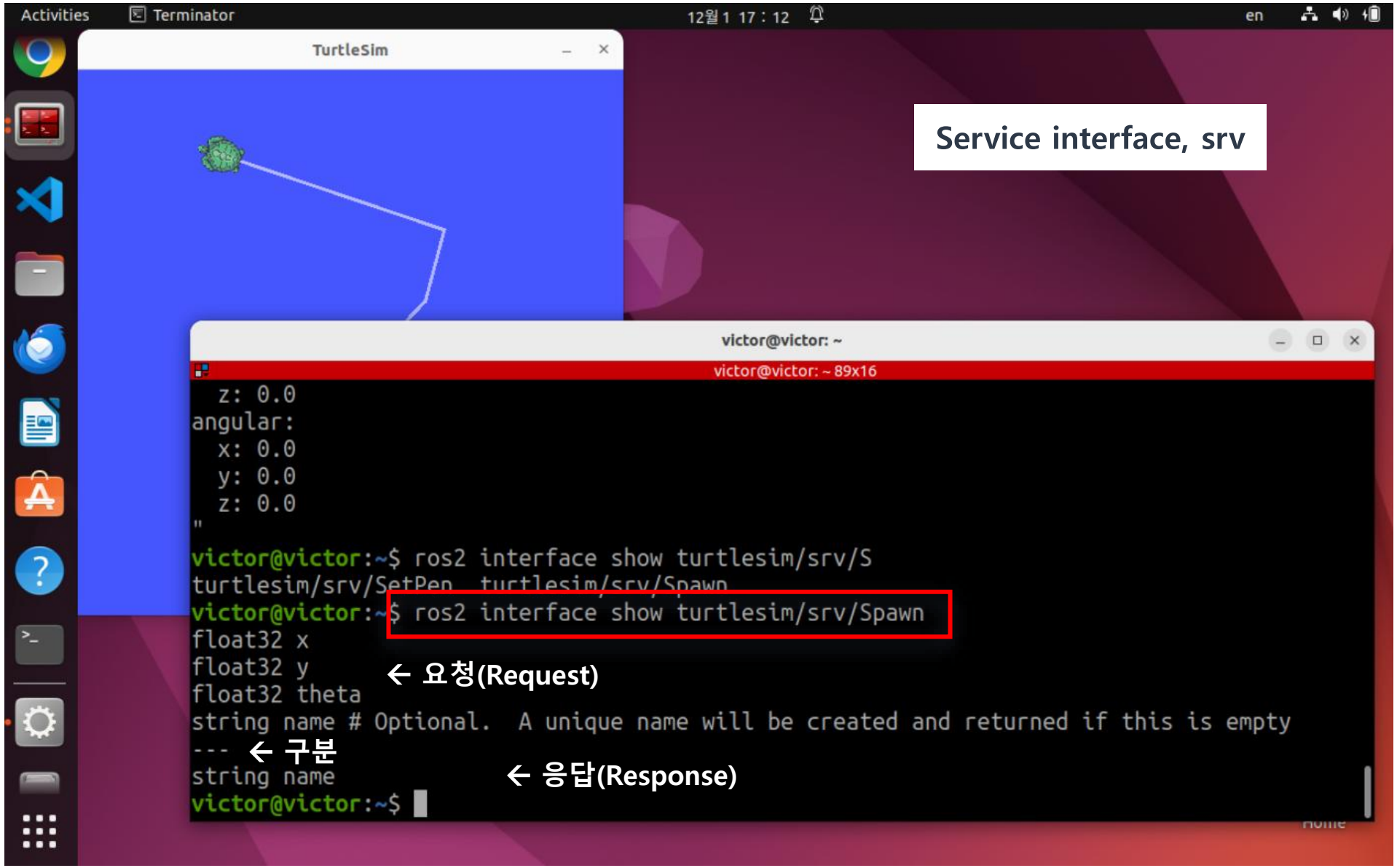
ROS2 실습 - Interface



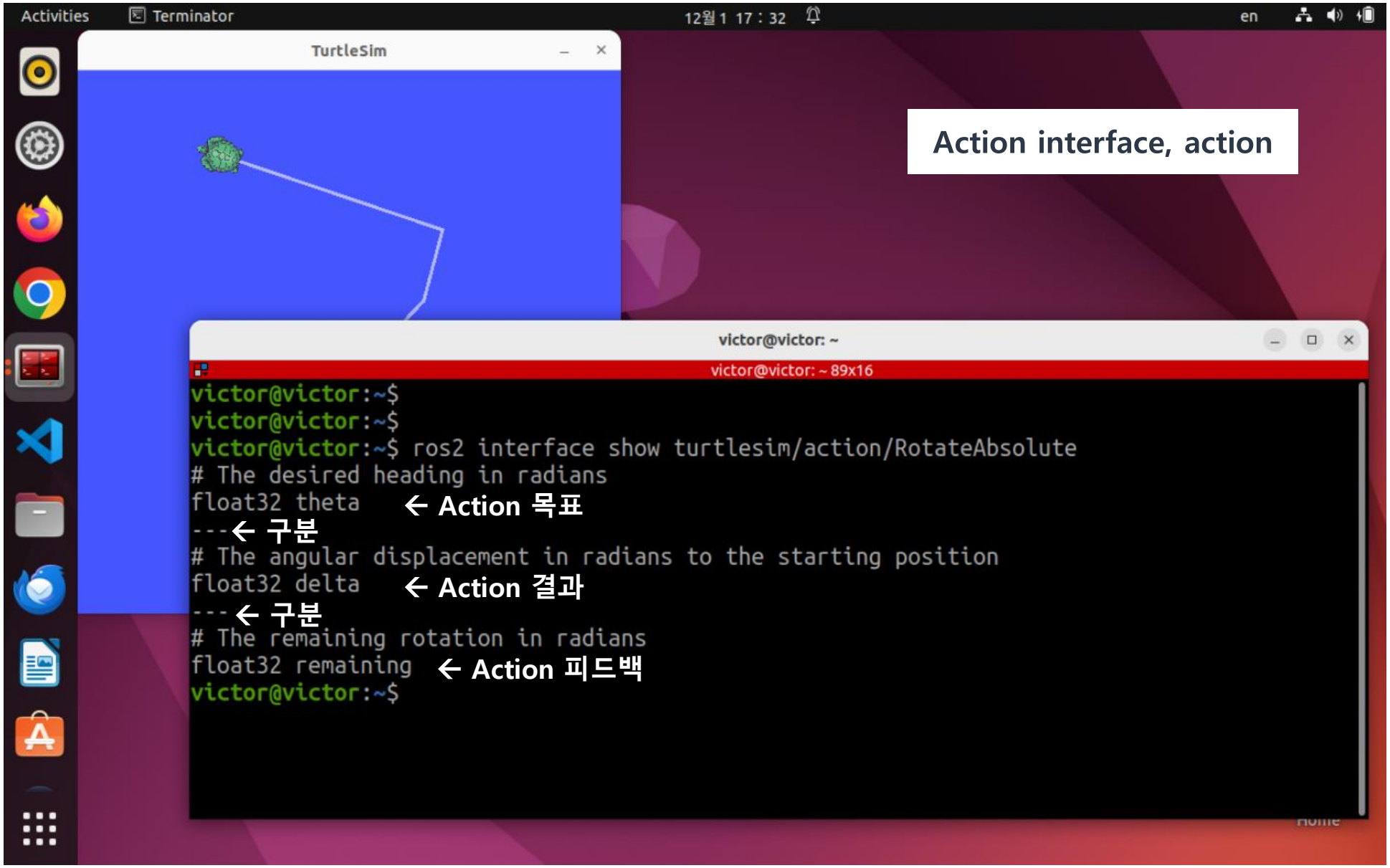
Interface

- list → 현재 개발환경의 모든 msg, srv, action 메시지 보여줌
- package → msg, srv, action 인터페이스를 담고 있는 패키지 목록
- package → 옵션에 package명을 입력하면 지정한 패키지에 포함된 interface보여줌
- proto → 특정 인터페이스 형태를 입력하면 그 인터페이스의 기본형태를 보여줌

ROS2 실습 - Interface



ROS2 실습 - Interface



ROS2 실습 - Interface

```
$ ros2 node info /turtlesim
/turtlesim
(생략)
  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

Node 정보 확인

```
$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
    /teleop_turtle
Action servers: 1
    /turtlesim
```

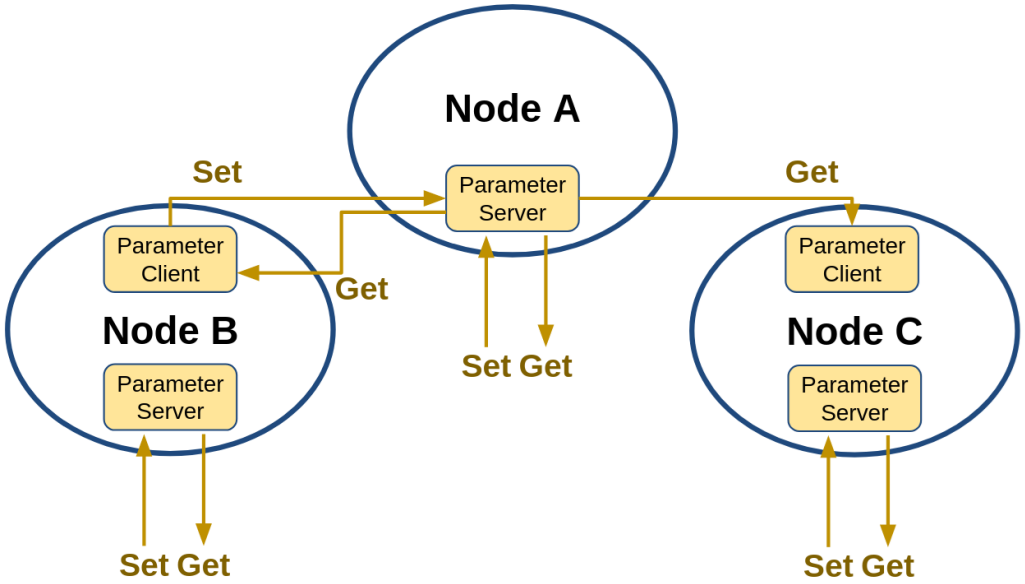
```
$ ros2 node info /teleop_turtle
/teleop_turtle
(생략)
  Action Servers:

  Action Clients:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
```

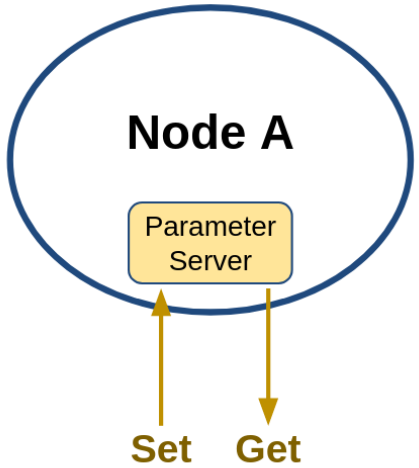
```
$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]
```

Action list

ROS2 실습 - Parameter

파라미터 관련 기능은 **RCL(ROS Client Libraries)의 기본 기능**으로 다음 그림과 같이 모든 노드가 자신만의 **Parameter server**를 가지고 있고, 그림과 같이 각 노드는 **Parameter client**도 포함시킬 수 있어서 자기 자신의 파라미터 및 다른 노드의 파라미터를 읽고 쓸수 있게 된다. 이를 활용하면 각 노드의 다양한 매개변수를 글로벌 매개변수처럼 사용할 수 있게 되어 추가 프로그래밍이나 컴파일 없이 능동적으로 변화 가능한 프로세스를 만들 수 있게 된다. 그리고 각 파라미터는 yaml 파일 형태의 파라미터 설정 파일을 만들어 초기 파라미터 값 설정 및 노드 실행시에 파라미터 설정 파일을 불러와서 사용할 수 있기에 ROS 2 프로그래밍에 매우 유용하게 사용할 수 있다.`






```yaml
name: John Doe
age: 30
address:
  street: 123 Main St
  city: Springfield
  zip: 12345
languages:
  - English
  - Spanish
  - French
```

YAML(YAML Ain't Markup Language 또는 Yet Another Markup Language)은 사람이 읽기 쉬운 데이터 직렬화 형식입니다. 주로 구성 파일, 데이터 전송, 설정 파일 등을 저장하는 데 사용됩니다. YAML은 JSON과 유사하지만, 더 간결하고 가독성이 뛰어난 문법을 제공합니다

ROS2 실습 – Parameter get & set

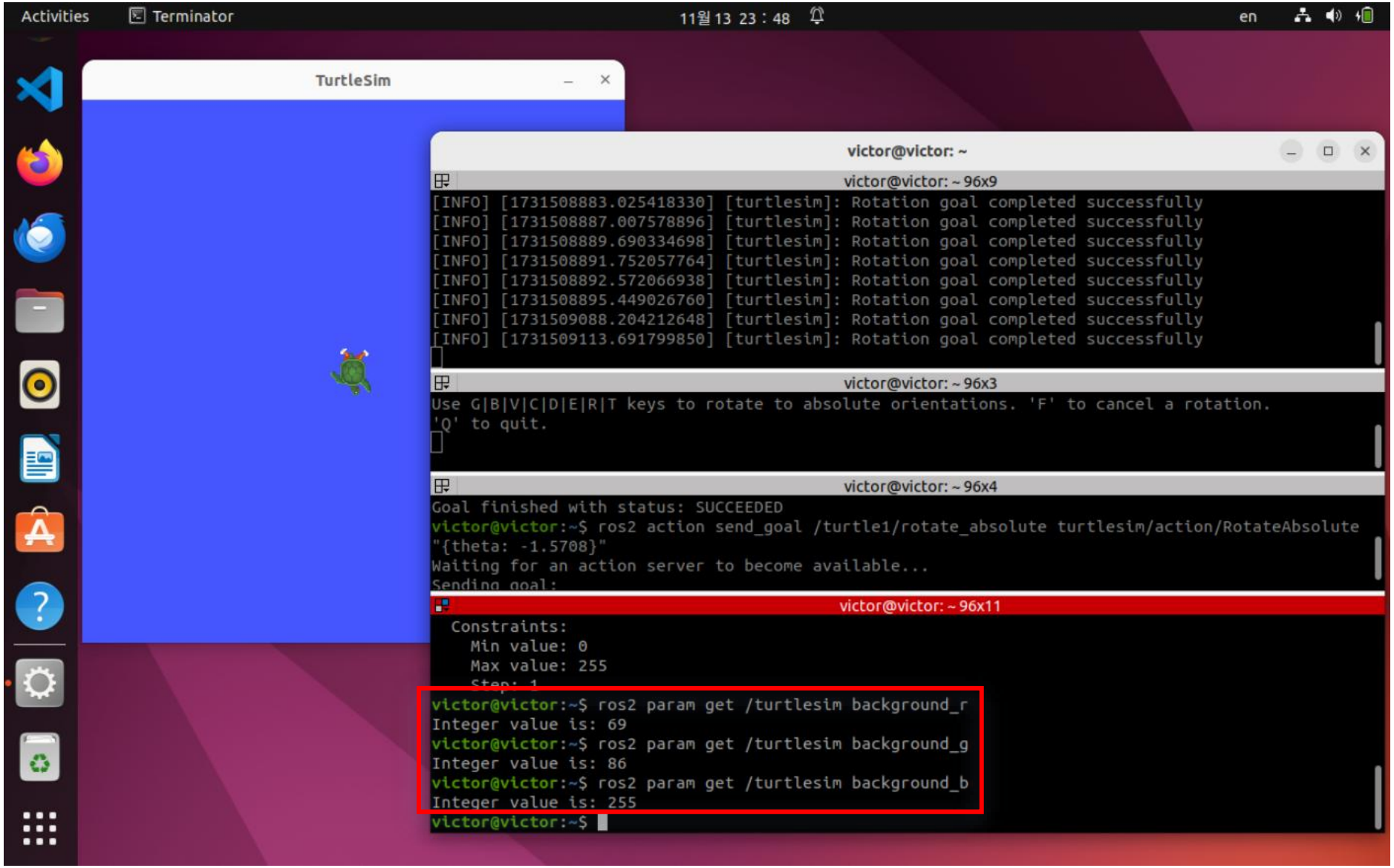
```
$ ros2 run turtlesim turtlesim_node
```

```
$ ros2 run turtlesim turtle_teleop_key
```

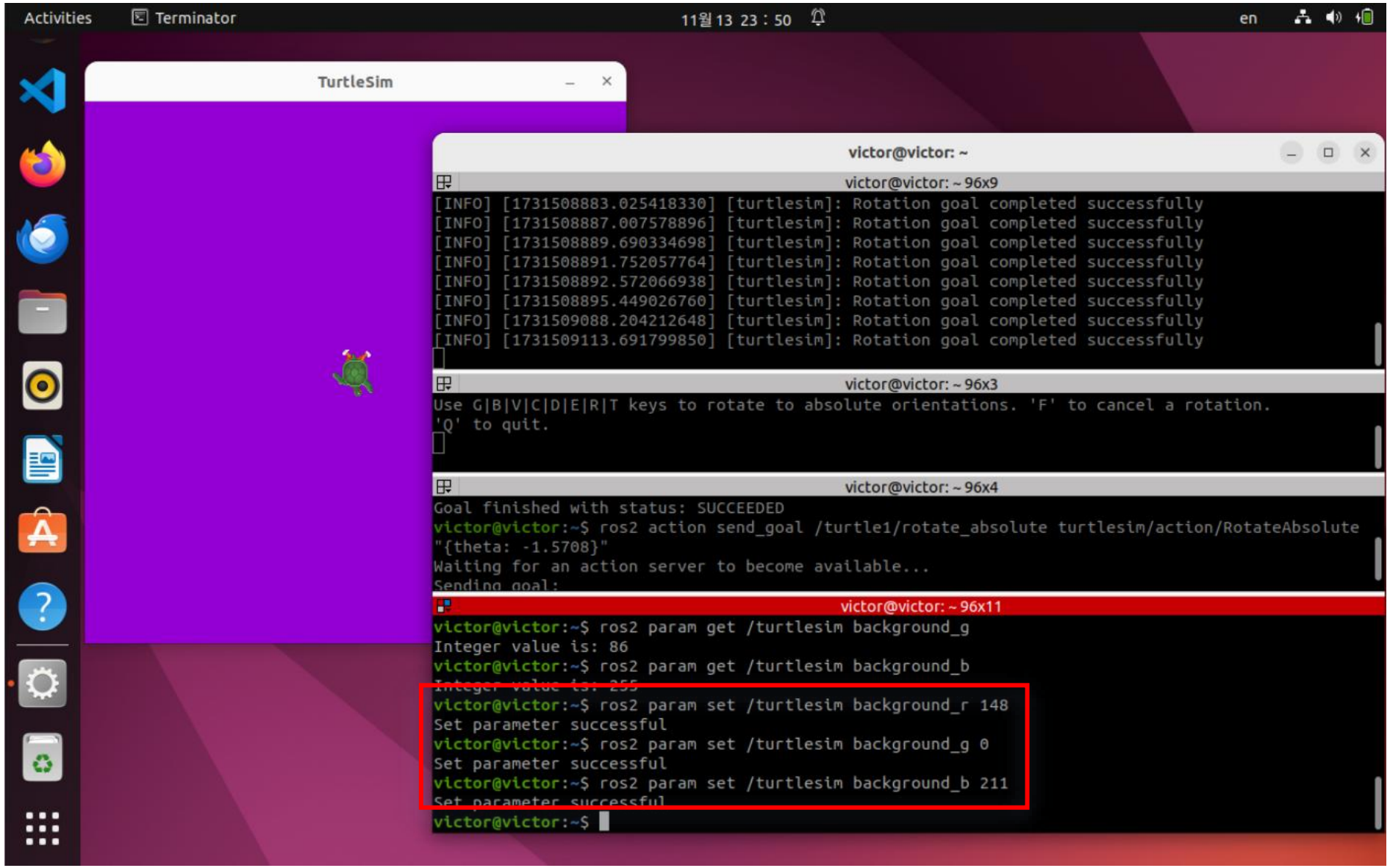
```
$ ros2 param list
/teleop_turtle:
  scale_angular
  scale_linear
  use_sim_time
/turtlesim:
  background_b
  background_g
  background_r
  use_sim_time
```

```
$ ros2 param describe /turtlesim background_b
Parameter name: background_b
  Type: integer
  Description: Blue channel of the background color
  Constraints:
    Min value: 0
    Max value: 255
    Step: 1
```

ROS2 실습 – Parameter get & set



ROS2 실습 – Parameter get & set



ROS2 실습 – Parameter get & set

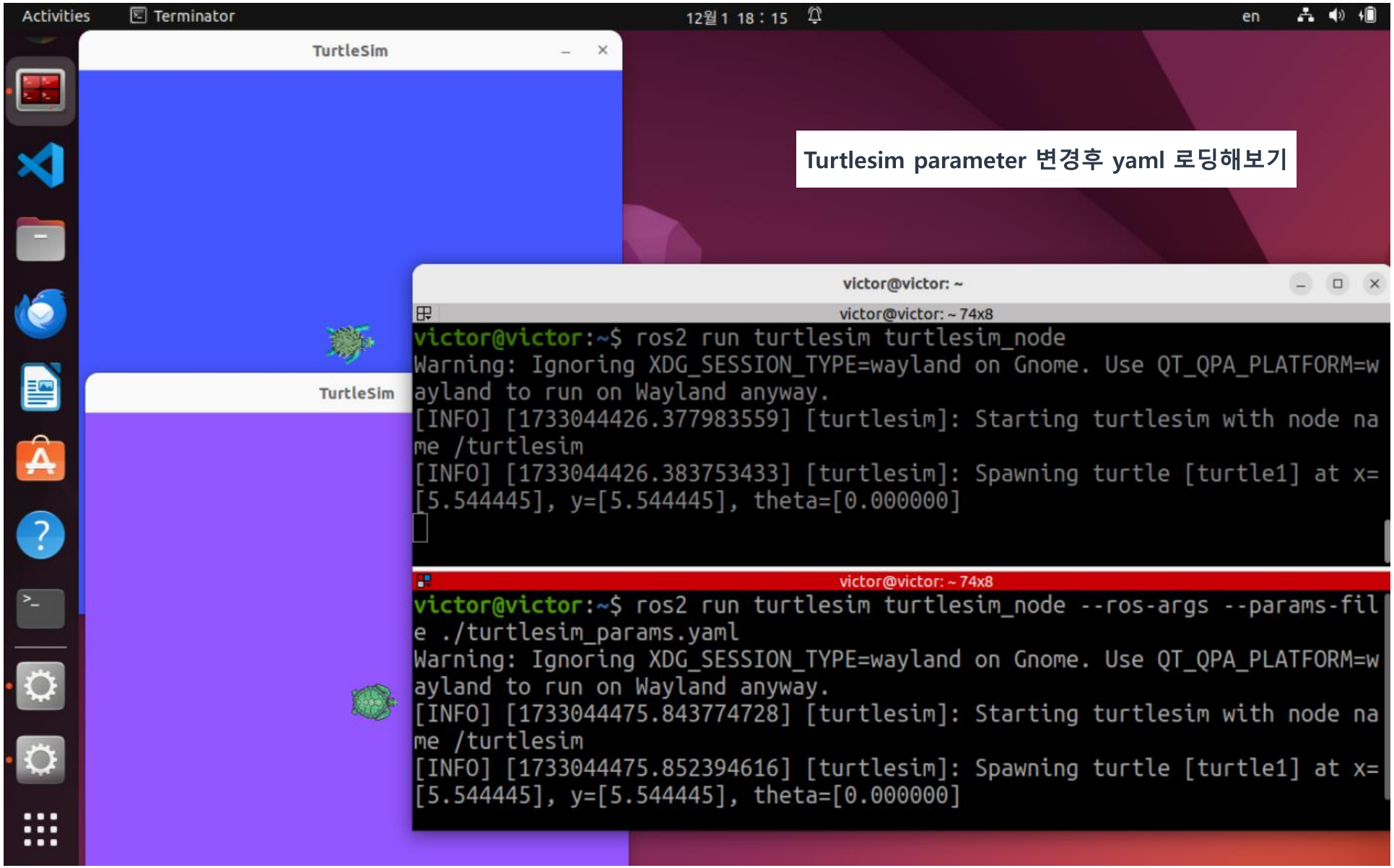
```
victor@victor: ~
victor@victor: ~ 89x16
victor@victor:~$ ros2 param dump /turtlesim
/turtlesim:
  ros__parameters:
    background_b: 255
    background_g: 86
    background_r: 69
    qos_overrides:
      /parameter_events:
        publisher:
          depth: 1000
          durability: volatile
          history: keep_last
          reliability: reliable
    use_sim_time: false

victor@victor:~$
```

```
victor@victor:~$ ros2 param dump /turtlesim > turtlesim_params.yaml
```

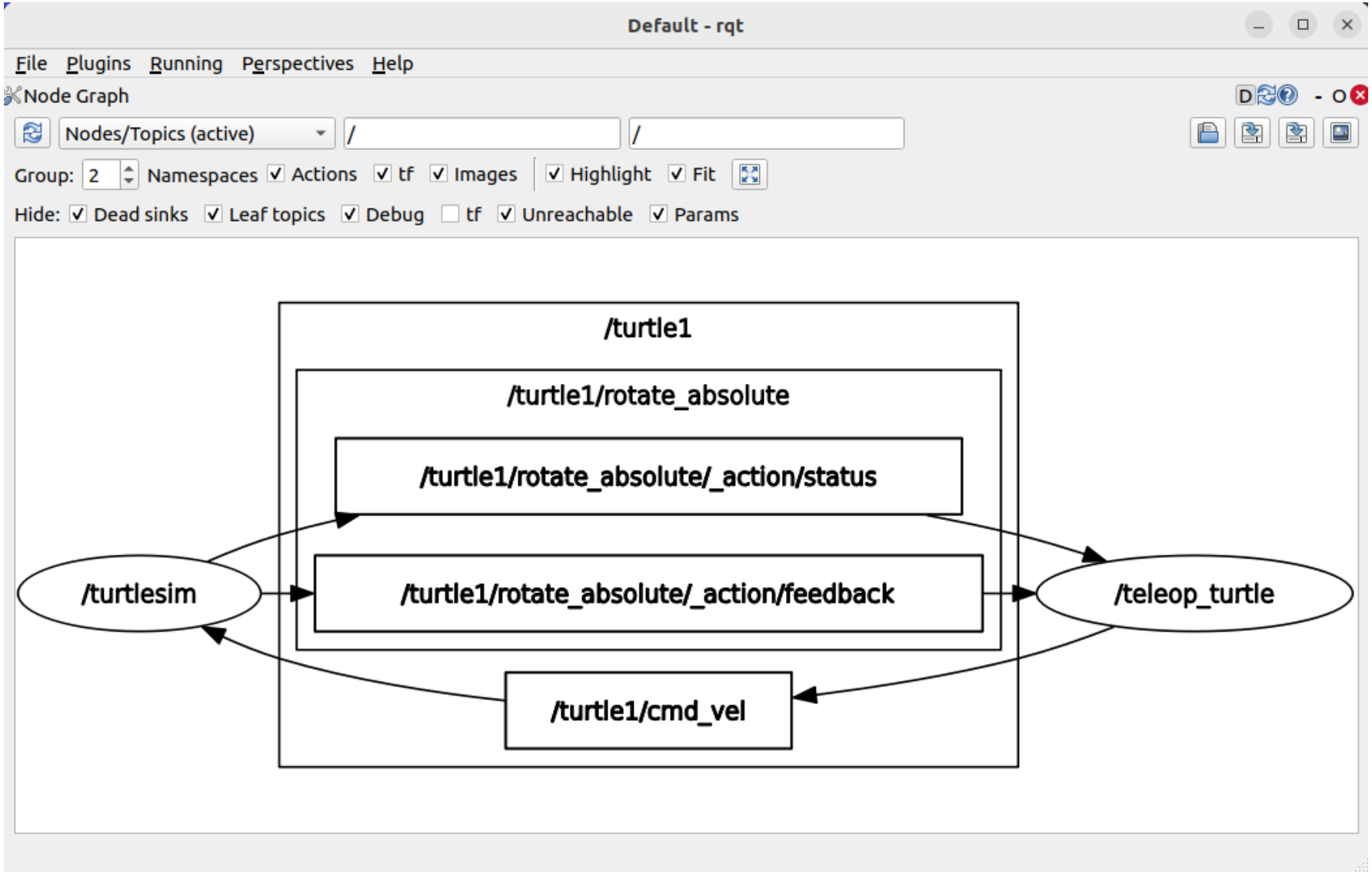
```
victor@victor: ~
victor@victor: ~ 89x21
drwxr-xr-x  2 victor victor      4096  6월   2 17:27 Templates/
-rw-rw-r--  1 victor victor       297 12월   1 18:08 turtlesim_params.yaml
-rw-rw-r--  1 victor victor      1990  6월   2 21:26 Untitled.ipynb
drwxr-xr-x  2 victor victor      4096  6월   2 17:27 Videos/
drwxrwxr-x  4 victor victor      4096  6월   2 21:36 .vscode/
victor@victor:~$ cat turtlesim_params.yaml
/turtlesim:
  ros__parameters:
    background_b: 255
    background_g: 86
    background_r: 69
    qos_overrides:
      /parameter_events:
        publisher:
          depth: 1000
          durability: volatile
          history: keep_last
          reliability: reliable
    use_sim_time: false

victor@victor:~$
```

Turtlesim parameter 변경후 yaml 로딩해보기

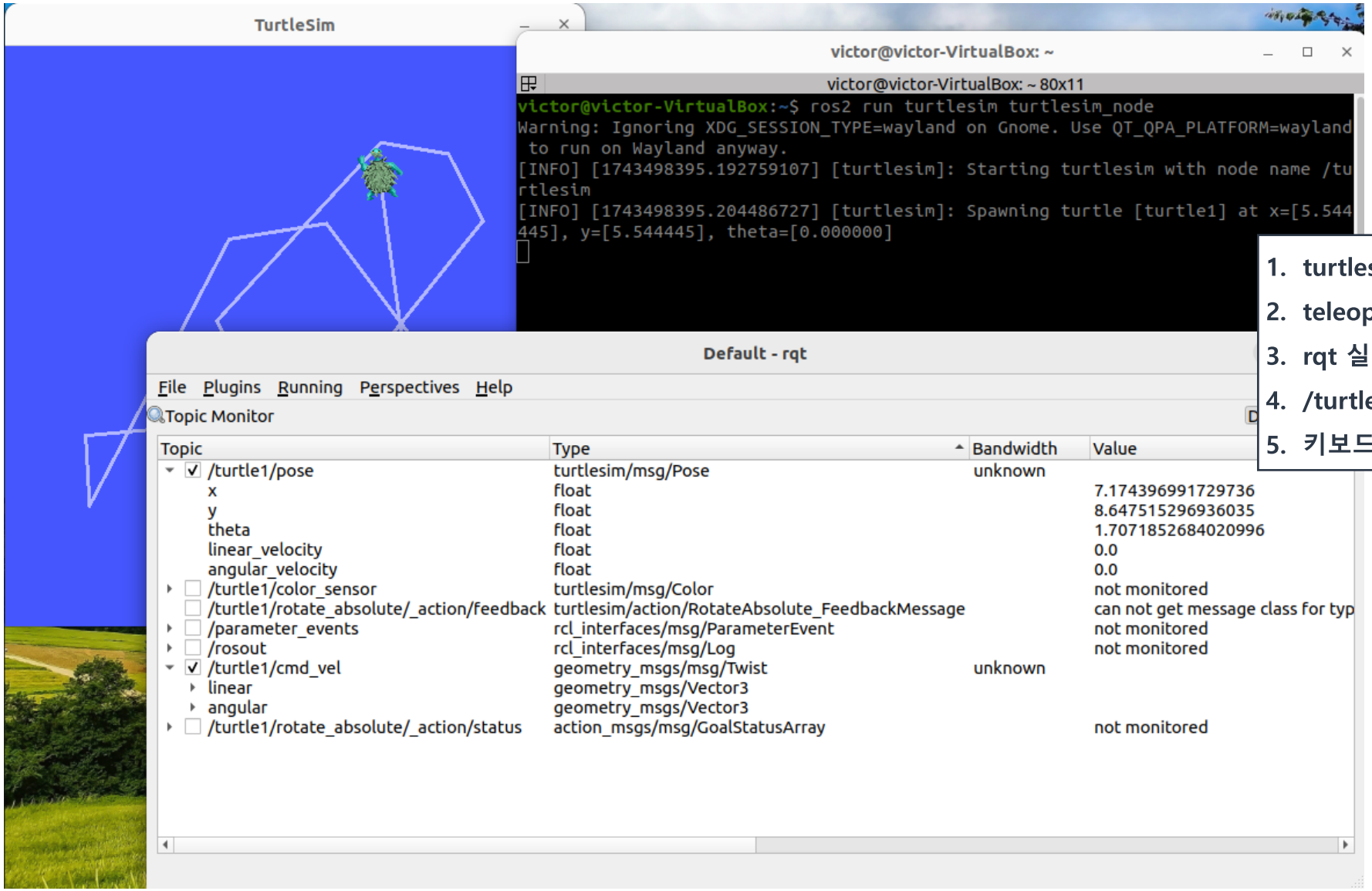
ROS2 실습 – Parameter get & set



ROS2 실습 – RQt(Node Graph)

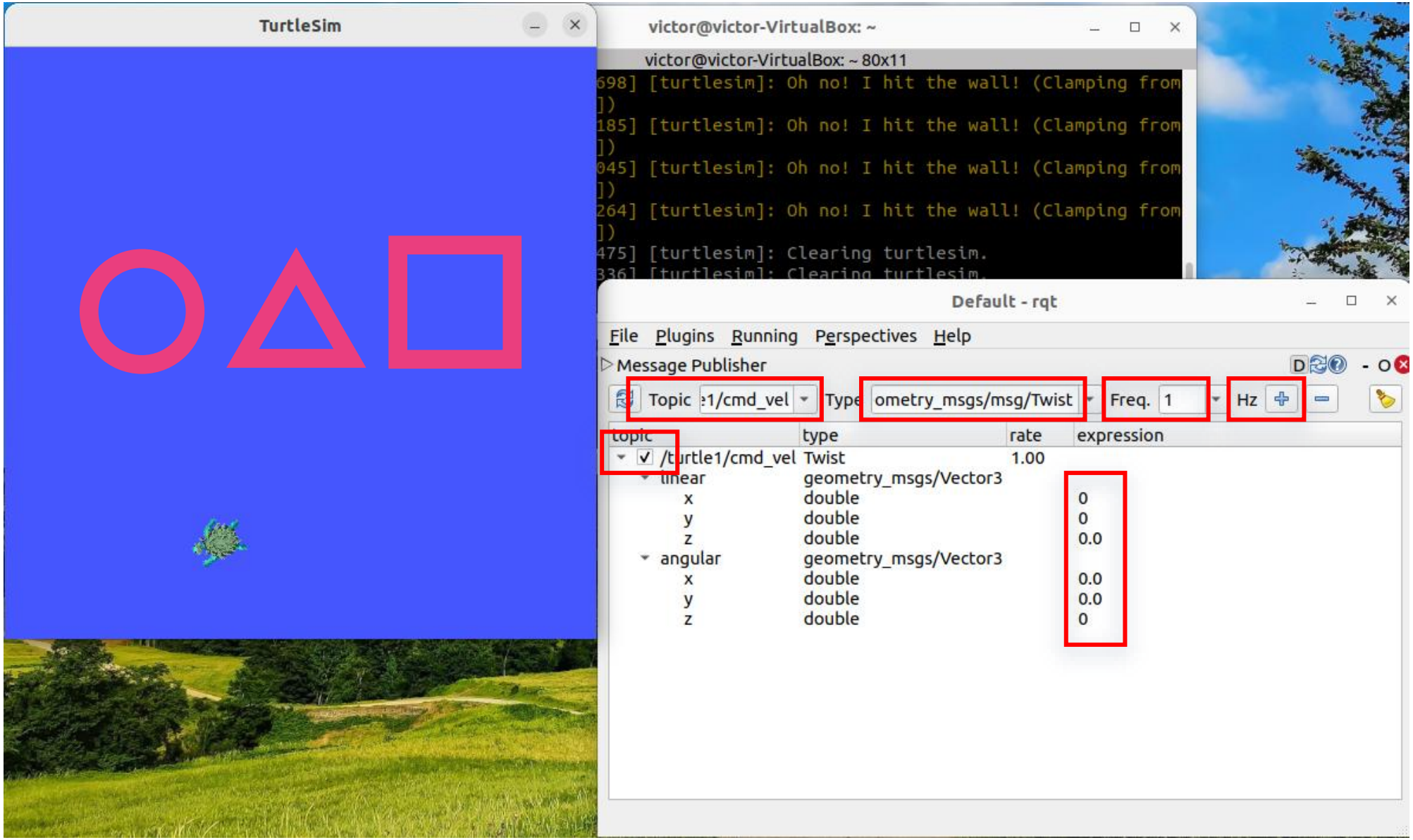


ROS2 실습 – RQt(Topic Monitor)

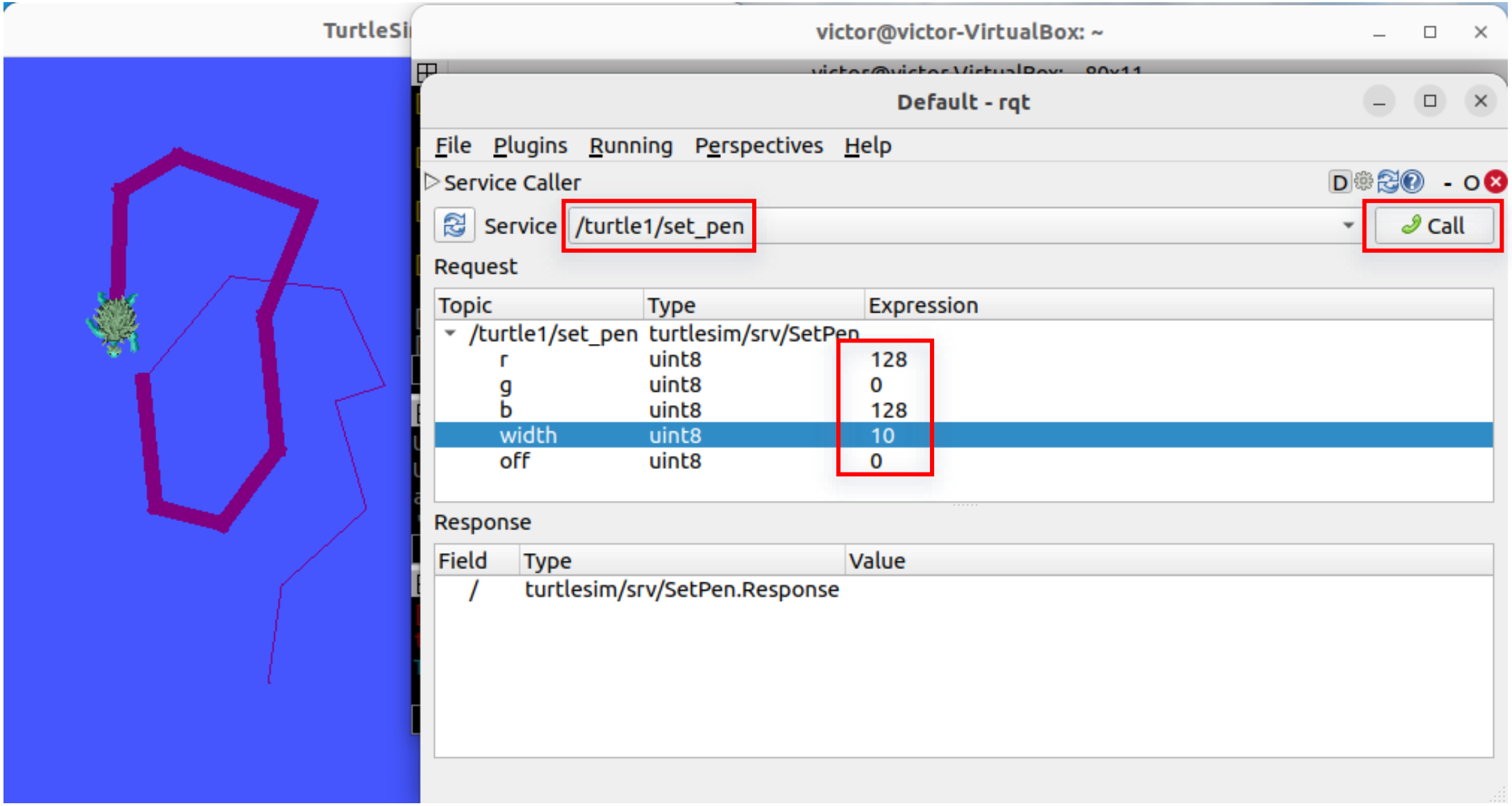


1. turtlesim_node 실행
2. teleop_key 실행
3. rqt 실행
4. /turtle1/pose와 /turtle1/cmd_vel 체크
5. 키보드로 움직이면서 value 바뀌는지 확인

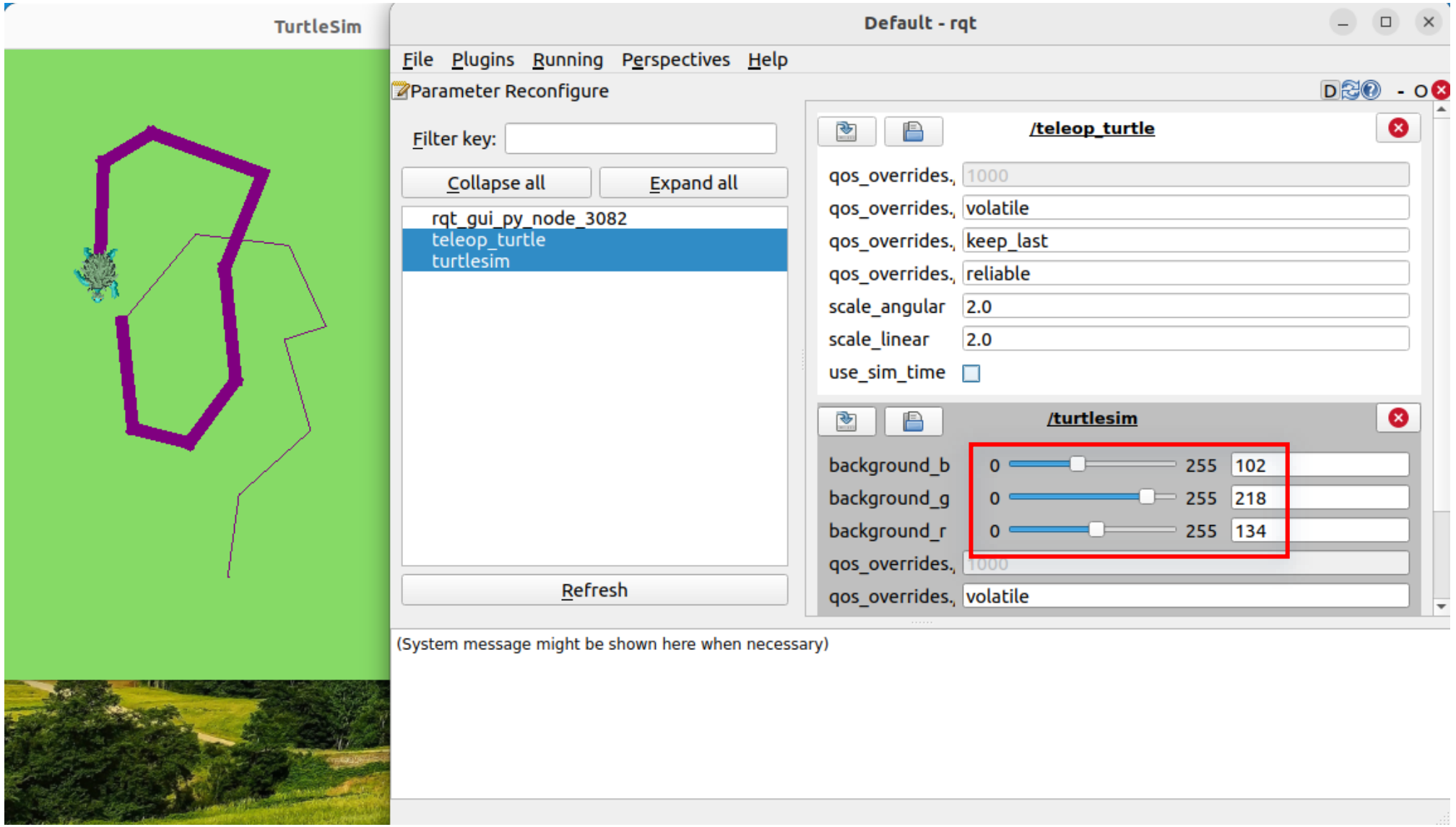
ROS2 실습 – RQt(Message Publisher)



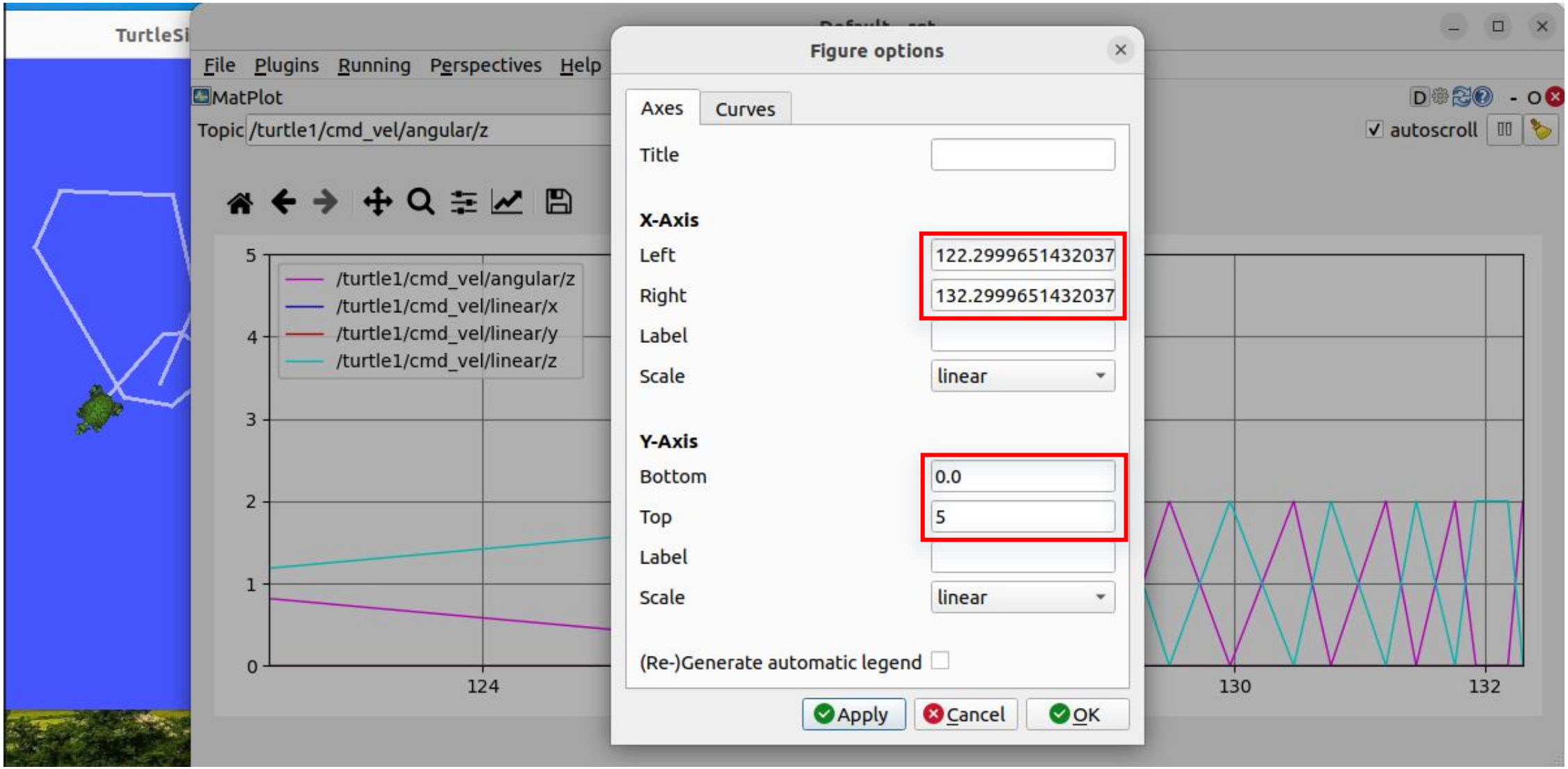
ROS2 실습 – RQt(Service Caller)



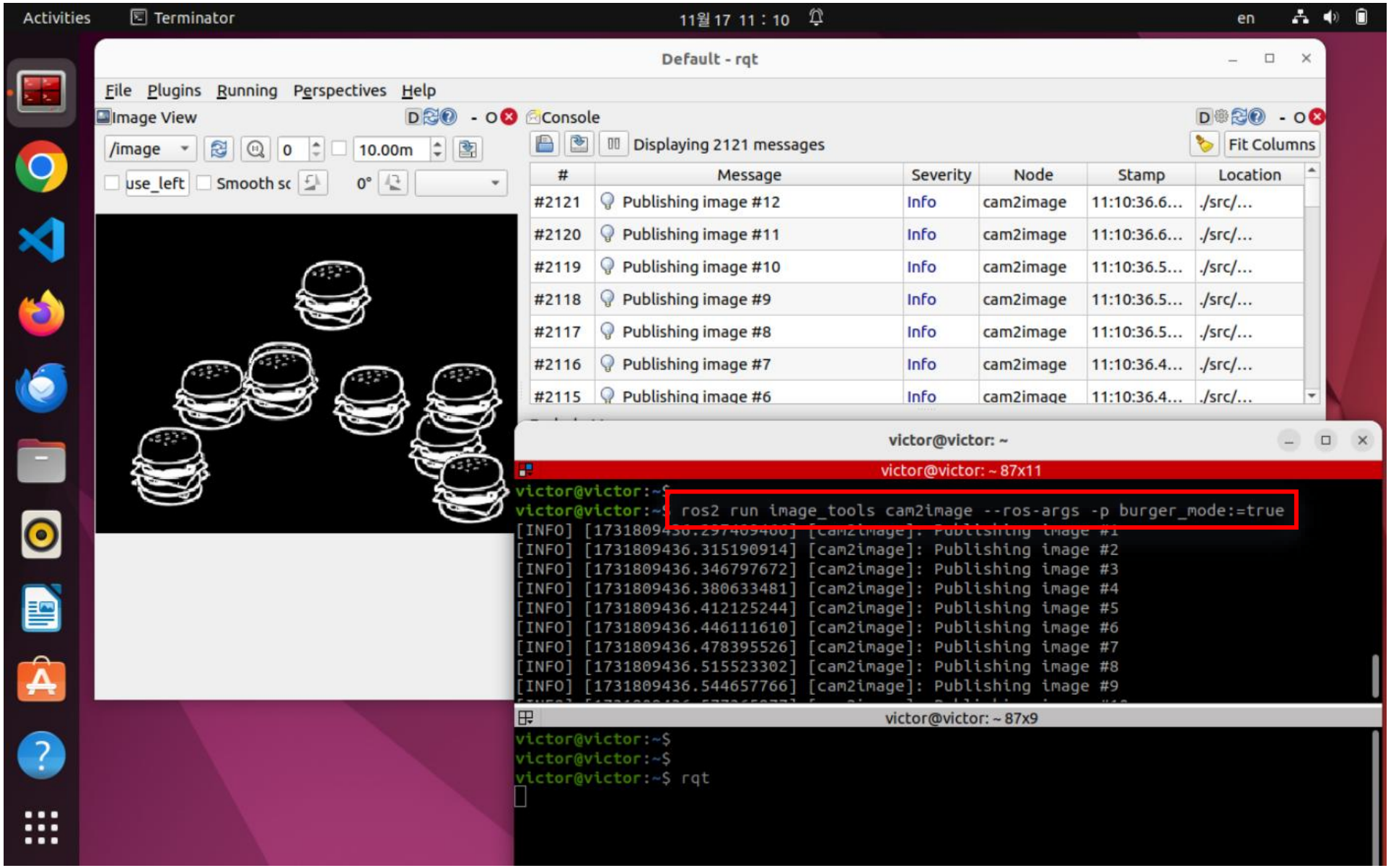
ROS2 실습 – RQt(Parameter Reconfigure)



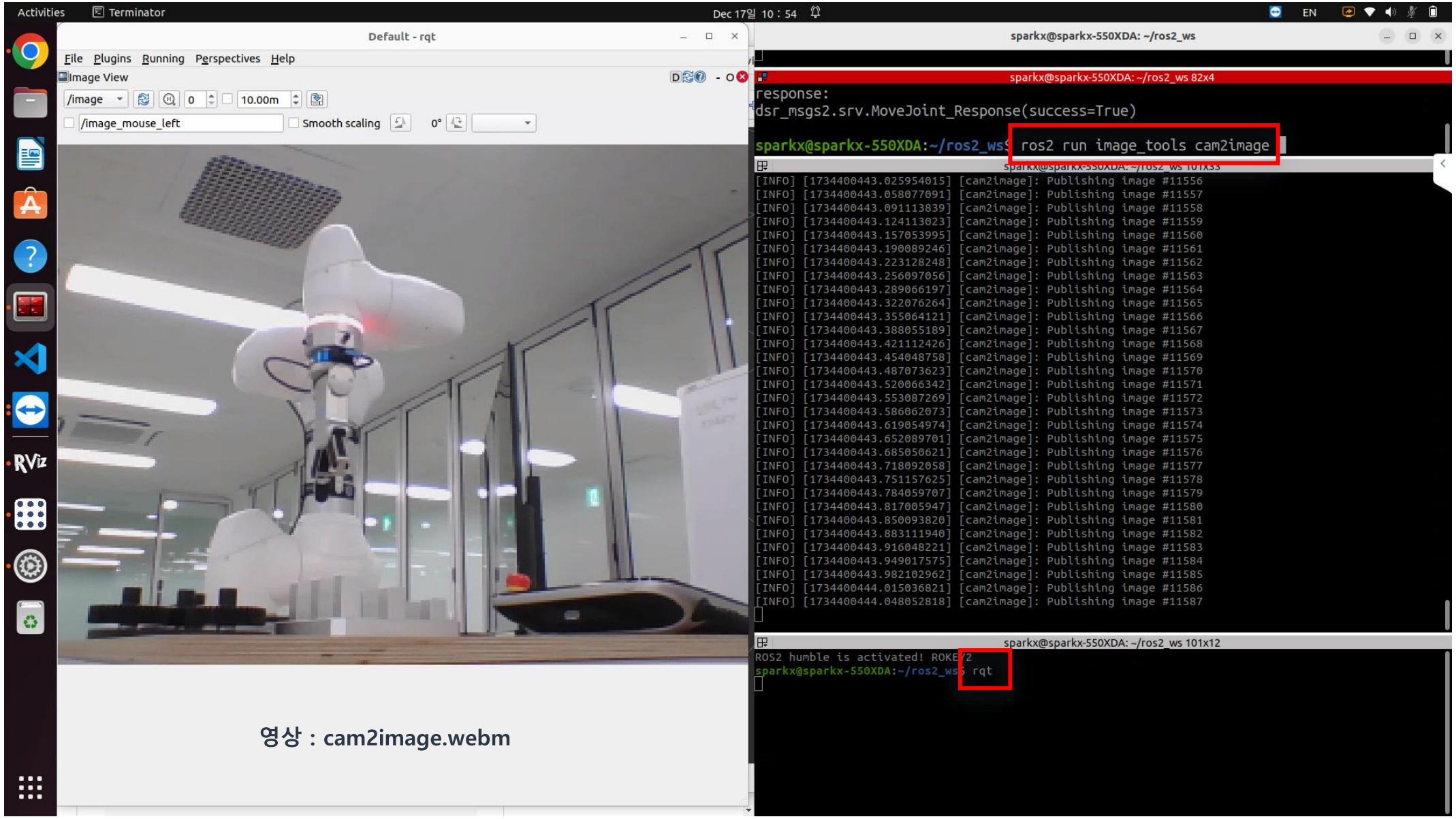
ROS2 실습 – RQt(Plot)



ROS2 실습 – RQt(image & console)

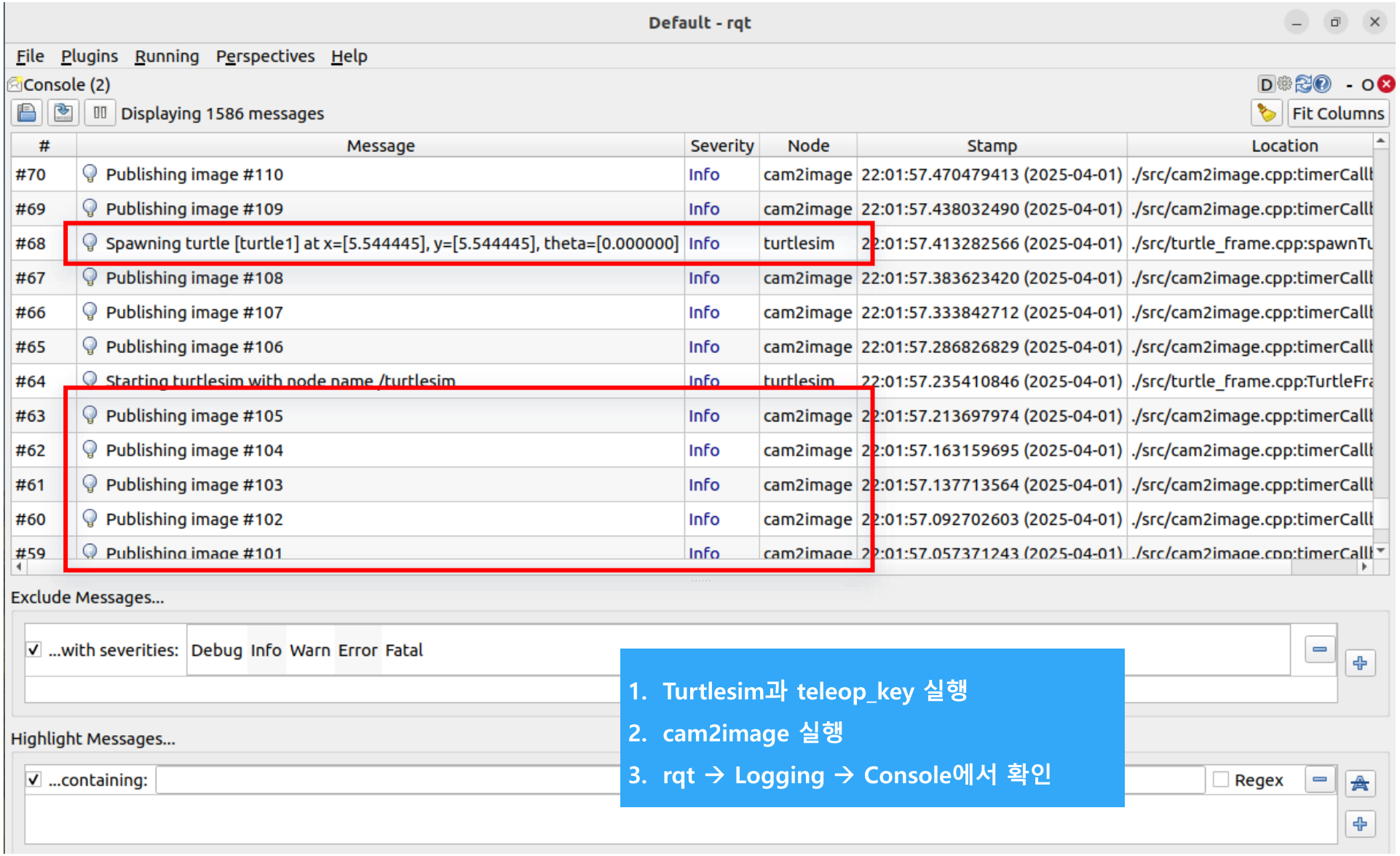


ROS2 실습 – RQt(Image)



영상 : cam2image.webm

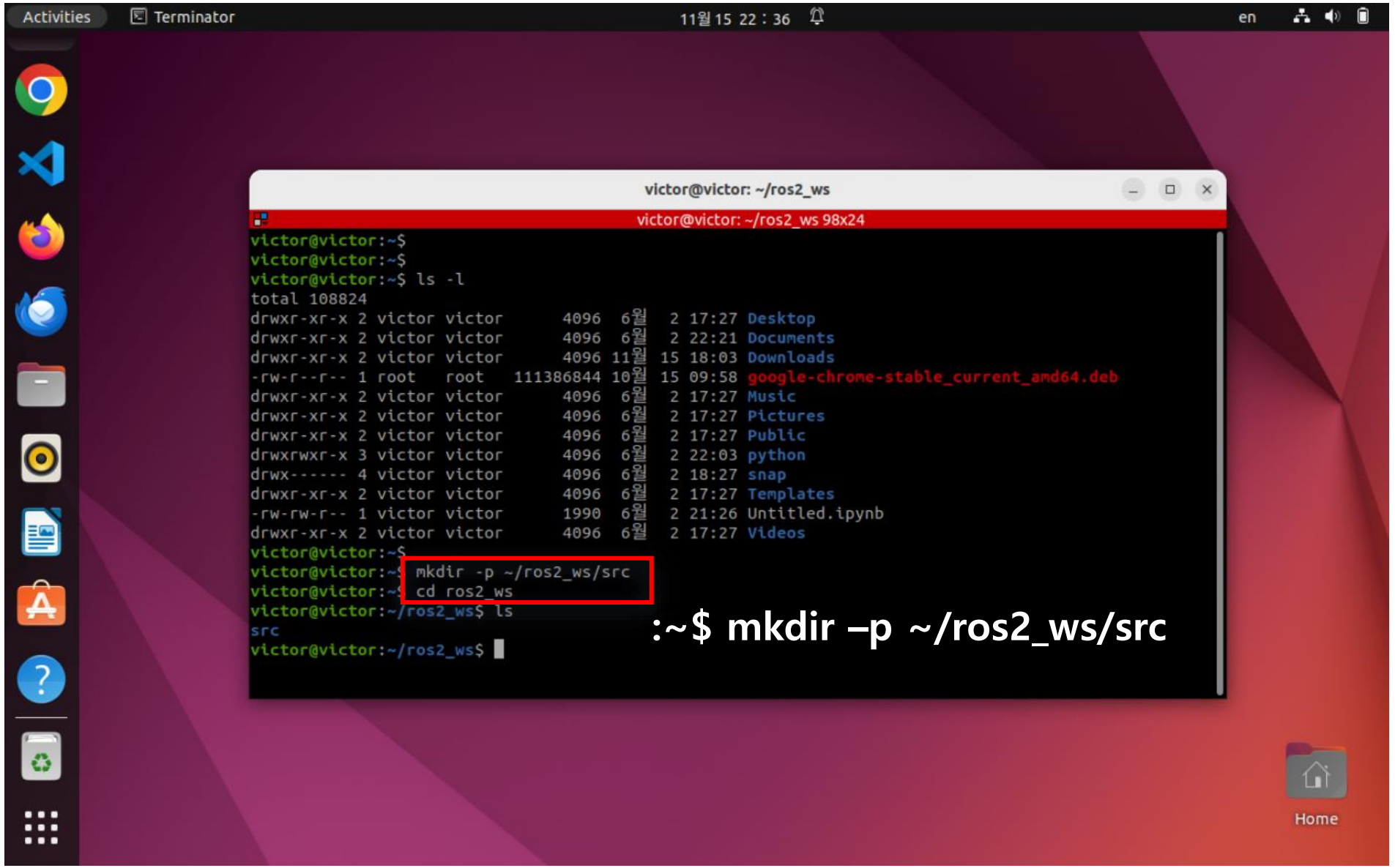
ROS2 실습 – RQt(Console)



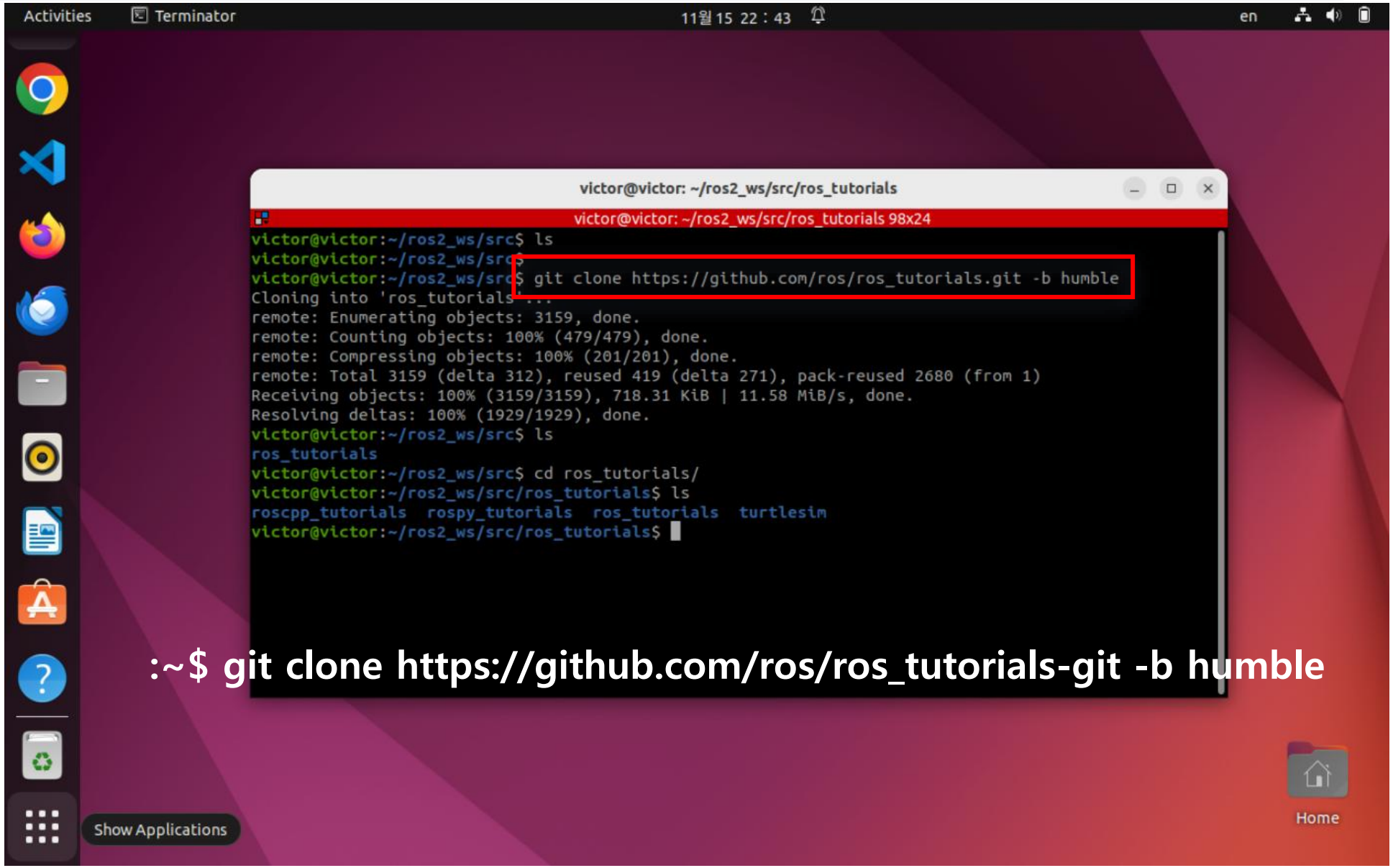
1. Turtlesim과 teleop_key 실행
2. cam2image 실행
3. rqt → Logging → Console에서 확인


HOW TO
Create a ROS2
Package (Python)

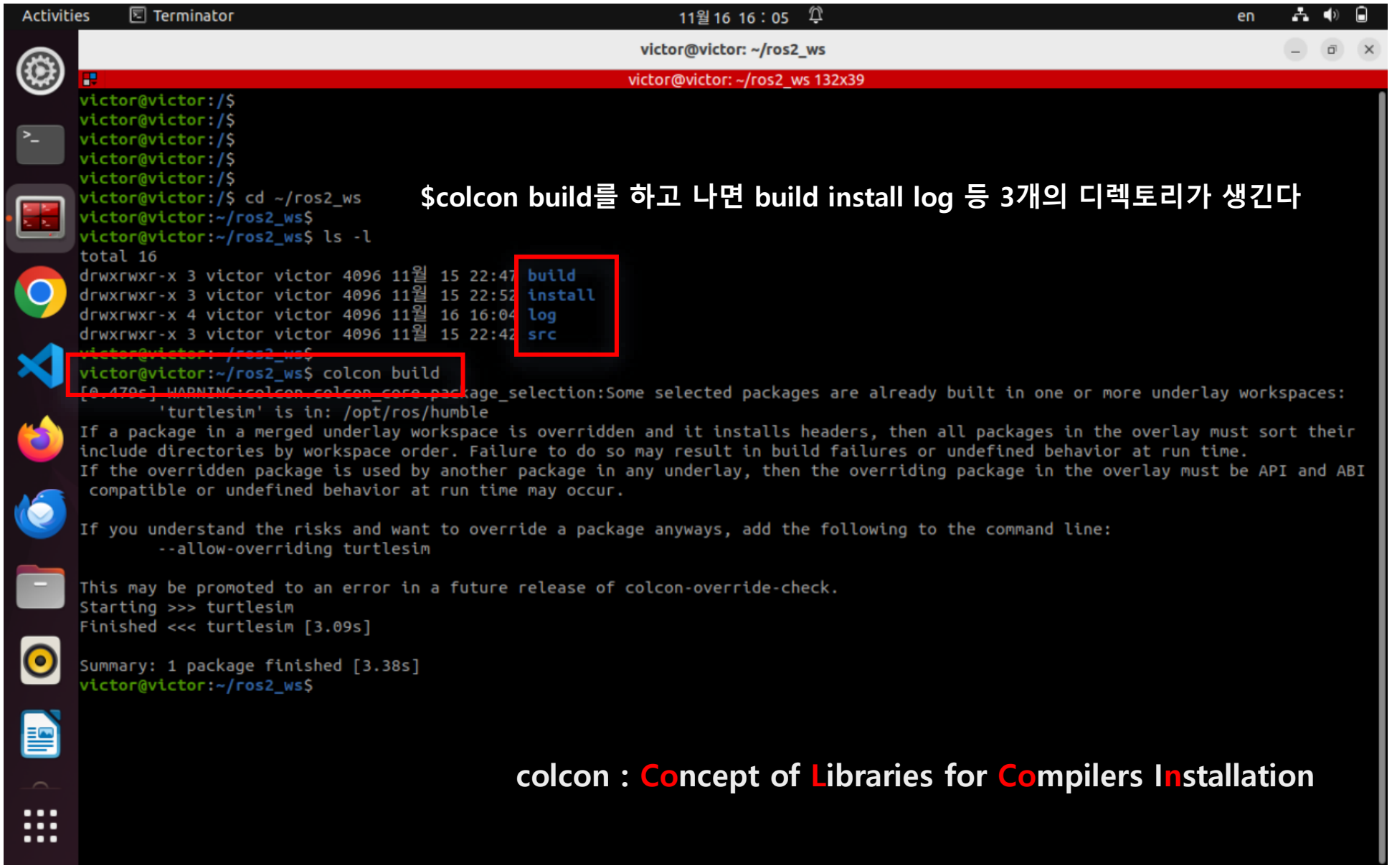
ROS2 실습 - Build



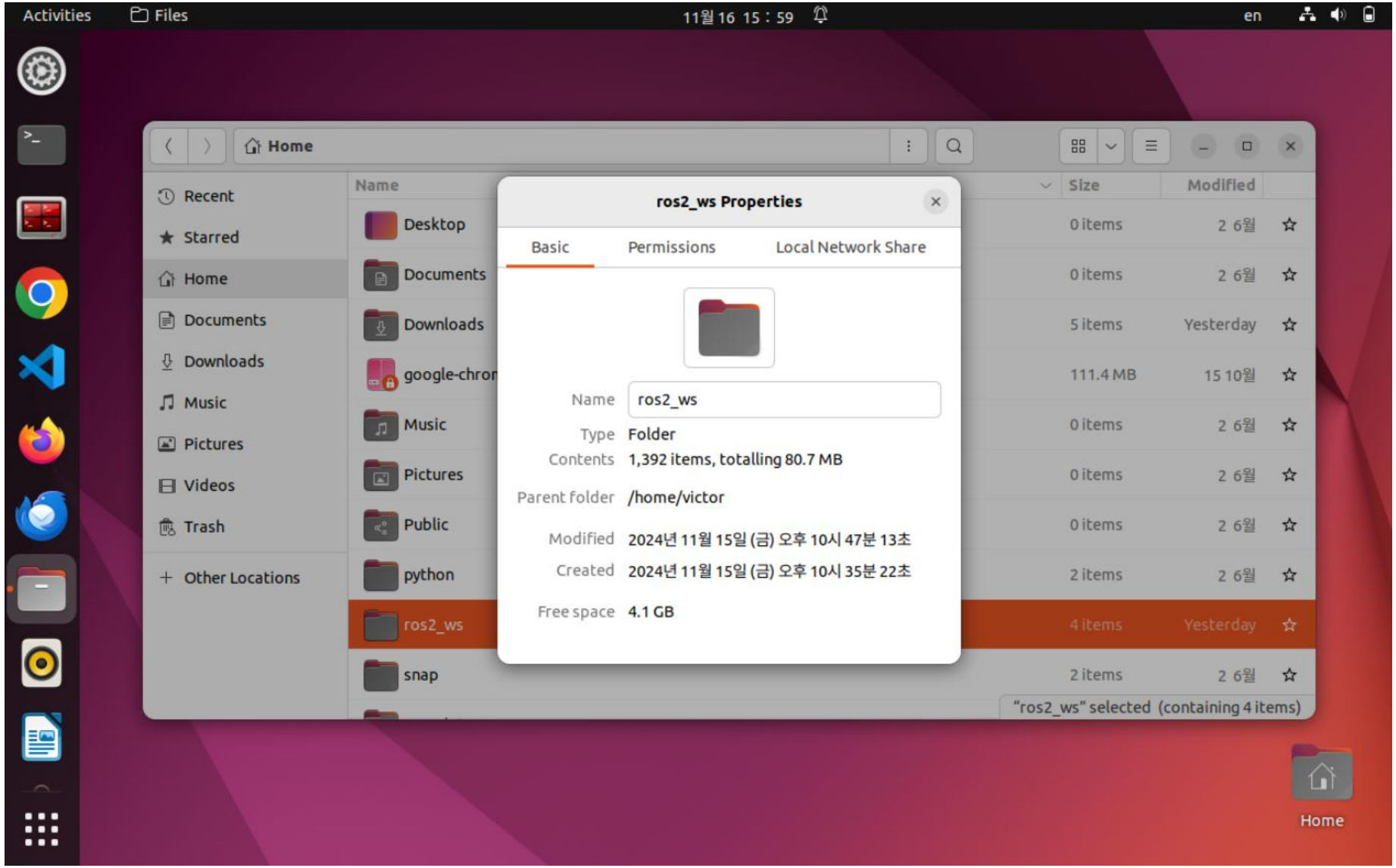
ROS2 실습


Activities
Terminator
11월 15 22 : 43
en
victor@victor: ~/ros2_ws/src/ros_tutorials
victor@victor: ~/ros2_ws/src/ros_tutorials 98x24
victor@victor:~/ros2_ws/src$ ls
victor@victor:~/ros2_ws/src$
victor@victor:~/ros2_ws/src$ git clone https://github.com/ros/ros_tutorials.git -b humble
Cloning into 'ros_tutorials'...
remote: Enumerating objects: 3159, done.
remote: Counting objects: 100% (479/479), done.
remote: Compressing objects: 100% (201/201), done.
remote: Total 3159 (delta 312), reused 419 (delta 271), pack-reused 2680 (from 1)
Receiving objects: 100% (3159/3159), 718.31 KiB | 11.58 MiB/s, done.
Resolving deltas: 100% (1929/1929), done.
victor@victor:~/ros2_ws/src$ ls
ros_tutorials
victor@victor:~/ros2_ws/src$ cd ros_tutorials/
victor@victor:~/ros2_ws/src/ros_tutorials$ ls
roscpp_tutorials rospy_tutorials ros_tutorials turtlesim
victor@victor:~/ros2_ws/src/ros_tutorials$
:~$ git clone https://github.com/ros/ros_tutorials-git -b humble
Show Applications
Home

ROS2 실습


Activities
Terminator
11월 16 16 : 05
en
victor@victor: ~/ros2_ws
victor@victor: ~/ros2_ws 132x39
victor@victor:/$
victor@victor:/$
victor@victor:/$
victor@victor:/$
victor@victor:/$
victor@victor:/$ cd ~/ros2_ws
victor@victor:~/ros2_ws$
victor@victor:~/ros2_ws$ ls -l
total 16
drwxrwxr-x 3 victor victor 4096 11월 15 22:47 build
drwxrwxr-x 3 victor victor 4096 11월 15 22:52 install
drwxrwxr-x 4 victor victor 4096 11월 16 16:04 log
drwxrwxr-x 3 victor victor 4096 11월 15 22:42 src
victor@victor:~/ros2_ws$ colcon build
'turtlesim' is in: /opt/ros/humble
If a package in a merged underlay workspace is overridden and it installs headers, then all packages in the overlay must sort their include directories by workspace order. Failure to do so may result in build failures or undefined behavior at run time.
If the overridden package is used by another package in any underlay, then the overriding package in the overlay must be API and ABI compatible or undefined behavior at run time may occur.
If you understand the risks and want to override a package anyways, add the following to the command line:
--allow-overriding turtlesim
This may be promoted to an error in a future release of colcon-override-check.
Starting >>> turtlesim
Finished <<< turtlesim [3.09s]
Summary: 1 package finished [3.38s]
victor@victor:~/ros2_ws$
$colcon build를 하고 나면 build install log 등 3개의 디렉토리가 생긴다
colcon : Concept of Libraries for Compilers Installation

ROS2 실습

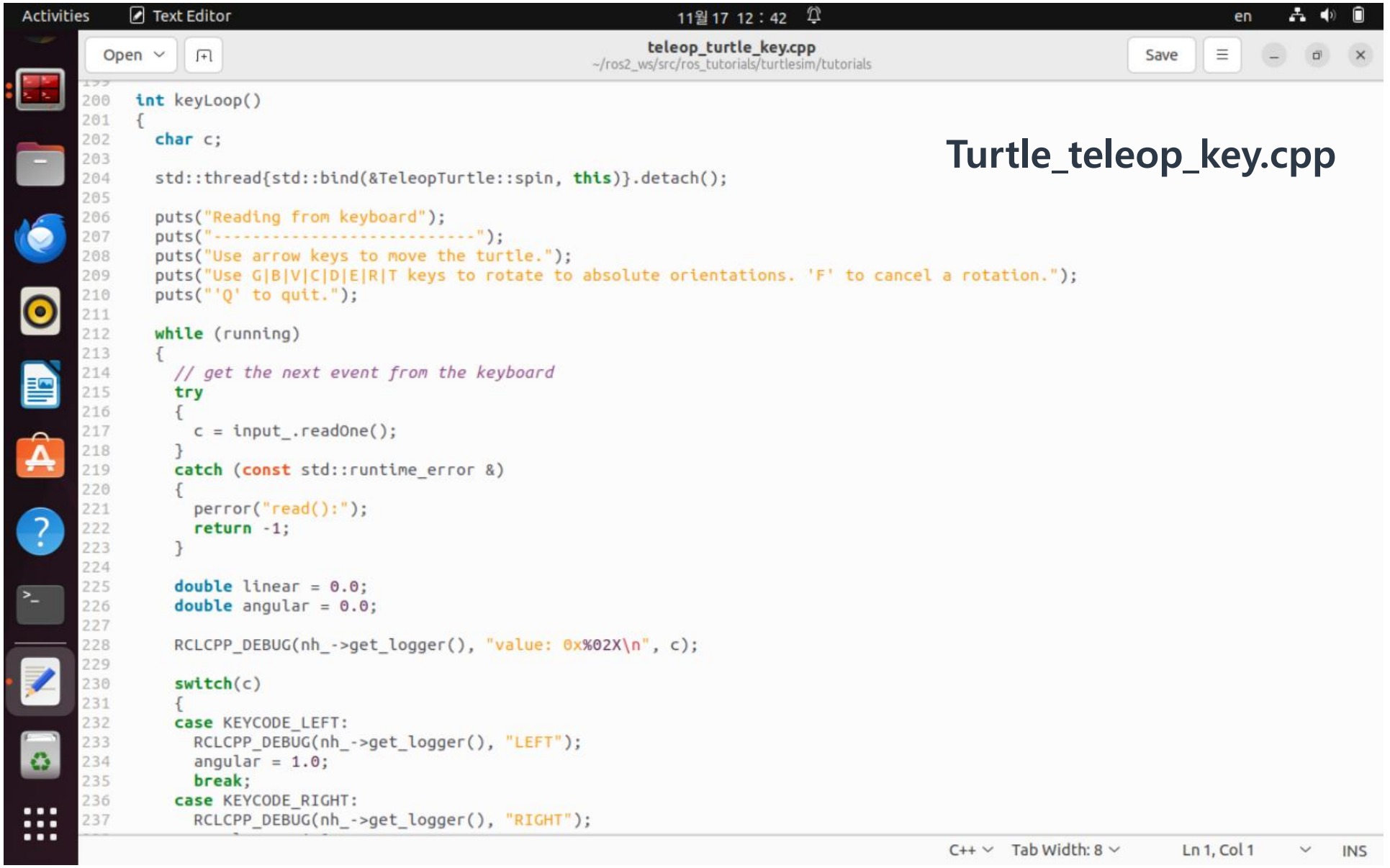

Activities
Files
11월 16 15 : 59
en
Home
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
Name
Size
Modified
Desktop
Documents
Downloads
google-chron
Music
Pictures
Public
python
ros2_ws
snap
0 items
2 6월
0 items
2 6월
5 items
Yesterday
111.4 MB
15 10월
0 items
2 6월
0 items
2 6월
0 items
2 6월
2 items
2 6월
4 items
Yesterday
2 items
2 6월
"ros2_ws" selected (containing 4 items)
ros2_ws Properties
Basic
Permissions
Local Network Share
Name
ros2_ws
Type
Folder
Contents
1,392 items, totalling 80.7 MB
Parent folder
/home/victor
Modified
2024년 11월 15일 (금) 오후 10시 47분 13초
Created
2024년 11월 15일 (금) 오후 10시 35분 22초
Free space
4.1 GB
Home

Turtle_teleop_key.py(for test only)

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import sys
import termios
import tty

# 키보드 입력을 처리하는 클래스
class TurtleTeleopKey(Node):
    def __init__(self):
        super().__init__('turtle_teleop_key')
        self.publisher = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        self.msg = Twist()
        self.print_instructions()

    def print_instructions(self):
        print("Control Your Turtle!")
        print("Use arrow keys to move: ")
        print("↑: Forward, ↓: Backward, →: Turn Right, ←: Turn Left")
        print("Press 'q' to quit.")

    def get_key(self):
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            key = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return key
```

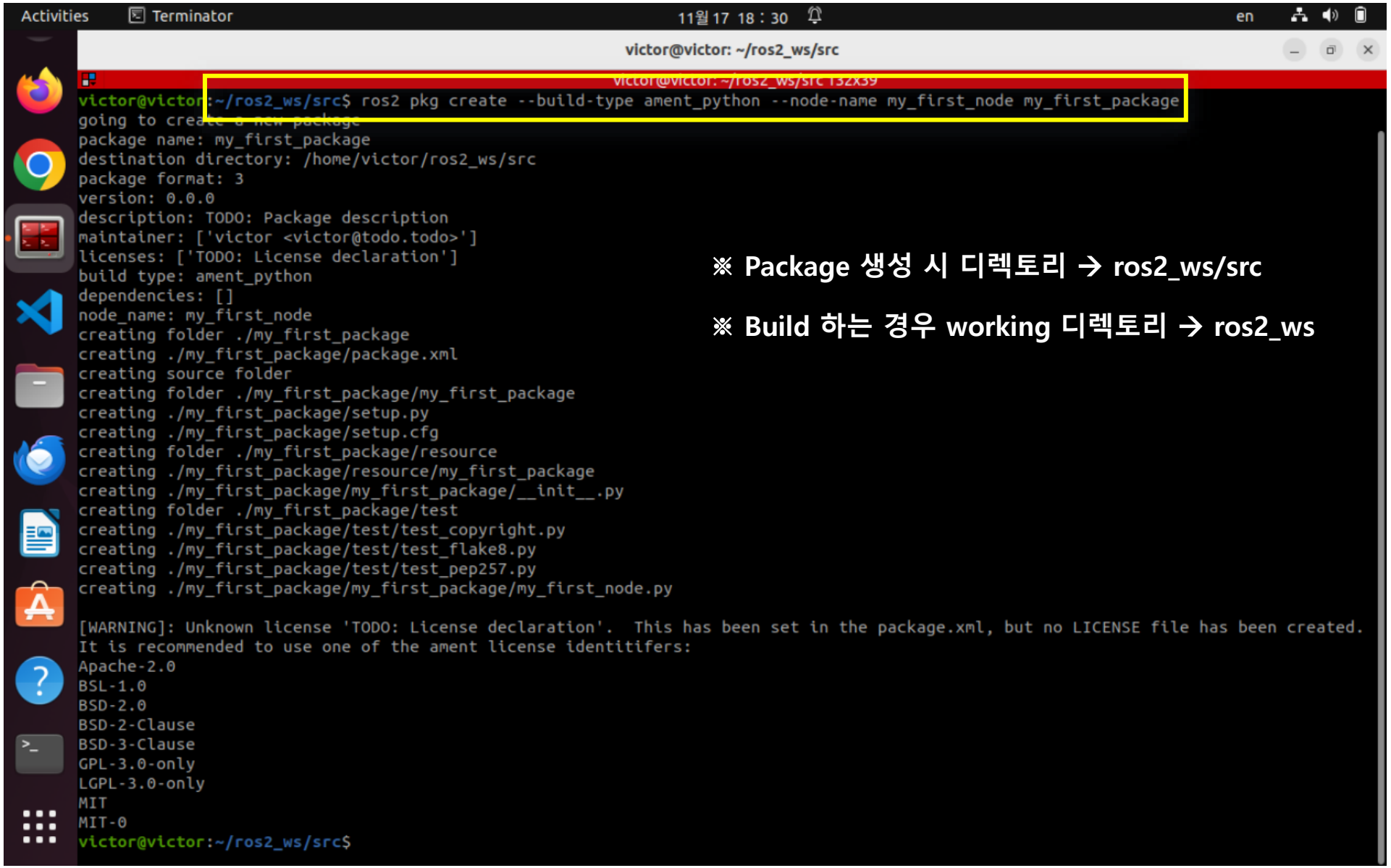
```
    def run(self):
        while True:
            key = self.get_key()
            if key == 'q':  # 종료
                break
            elif key == '₩x1b[A':  # ↑
                self.msg.linear.x = 2.0
                self.msg.angular.z = 0.0
            elif key == '₩x1b[B':  # ↓
                self.msg.linear.x = -2.0
                self.msg.angular.z = 0.0
            elif key == '₩x1b[C':  # →
                self.msg.linear.x = 0.0
                self.msg.angular.z = -2.0
            elif key == '₩x1b[D':  # ←
                self.msg.linear.x = 0.0
                self.msg.angular.z = 2.0
            else:
                self.msg.linear.x = 0.0
                self.msg.angular.z = 0.0

            self.publisher.publish(self.msg)

        print("Exiting...")

def main():
    rclpy.init()
    node = TurtleTeleopKey()
    node.run()
    node.destroy_node()
    rclpy.shutdown()

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

Turtle_teleop_key.cpp




```cpp
int keyLoop()
{
  char c;

  std::thread{std::bind(&TeleopTurtle::spin, this)}.detach();

  puts("Reading from keyboard");
  puts("---------------------------");
  puts("Use arrow keys to move the turtle.");
  puts("Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.");
  puts("'Q' to quit.");

  while (running)
  {
    // get the next event from the keyboard
    try
    {
      c = input_.readOne();
    }
    catch (const std::runtime_error &)
    {
      perror("read():");
      return -1;
    }

    double linear = 0.0;
    double angular = 0.0;

    RCLCPP_DEBUG(nh_->get_logger(), "value: 0x%02X\n", c);

    switch(c)
    {
    case KEYCODE_LEFT:
      RCLCPP_DEBUG(nh_->get_logger(), "LEFT");
      angular = 1.0;
      break;
    case KEYCODE_RIGHT:
      RCLCPP_DEBUG(nh_->get_logger(), "RIGHT");
```

ROS2 실습 – package build - my_first_package



※ Package 생성 시 디렉토리 → ros2_ws/src

※ Build 하는 경우 working 디렉토리 → ros2_ws

1) 전체 패키지를 빌드할 때

```
$ cd ~/robot_ws && colcon build --symlink-install
```

2) 해당 패키지만 빌드할 때

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-select [패키지 이름]
```

- vcstool (버전 컨트롤 시스템 툴)

```
$ wget https://raw.githubusercontent.com/ros2/ros2/humble/ros2.repos
$ vcs import src < ros2.repos
```

- rosdep (의존성 관리 툴)

```
$ sudo rosdep init
$ rosdep update
$ rosdep install --from-paths src -y --ignore-src
```

- bloom (바이너리 패키지 관리 툴)

```
$ bloom-release --ros-distro humble --track humble awesome_pkg
```

패키지 파일 사용 방법 ROS 2 프로그래밍 전에 알아둬야 할 필수 사전 정보

- 패키지 설정 파일 — package.xml(ROS pkg의 필수 구성요소)
- 빌드 설정 파일 — CMakeLists.txt(순수 Python pkg는 CMakeLists.txt 파일 없음)
- 파이썬 패키지 설정 파일 — setup.py(순수한 ROS2 Python pkg에만 사용하는 배포를 위한 설정 파일)
- 파이썬 패키지 환경 설정 파일 — setup.cfg(순수한 ROS2 Python pkg에만 사용하는 배포를 위한 구성 파일)
- RQt 플러그인 설정 파일 — plugin.xml(RQT plugin으로 pkg를 작성할때의 필수 구성요소)
- 패키지 변경로그 파일 — CHANGELOG.rst(pkg업데이트 내역 기술, 개발 이력 추적)
- 라이선스 파일 — LICENSE(pkg코드에 사용된 라이센스 기술)
- 패키지 설명 파일 — README.md(markdown 파일)

ROS2 실습 – 패키지 설정파일(package.xml)

패키지 설정 파일은 ROS 패키지의 필수 구성 요소로서 **패키지의 정보를 기술하는 파일**

기술하는 내용으로는 **패키지 이름, 저작자, 라이선스, 의존성 패키지 등**이 있으며,

XML 형식으로 기술하고 파일명은 `**package.xml**`을 사용

사용되는 빌드 툴, 의존성 패키지들이 모두 기술되기에 빌드 및 패키지 설치, 사용에 있어서 매우 중요한 파일

모든 ROS 패키지의 필수 파일로 각 패키지당 무조건 1개의 패키지 설정 파일 (package.xml)을 포함

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_first_ros_rclcpp_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pyo@robotis.com">pyo</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_first_ros_rclpy_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pyo@robotis.com">pyo</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

ROS2 실습 – 패키지 설정파일(package.xml)

항목	내용
<?xml>	문서 문법을 정의하는 문구로 아래의 내용은 xml 버전 1.0을 따르고 있다는 것을 알린다.
<package>	이 구문부터 맨 끝의 </package>까지가 ROS 패키지 설정 부분이다. 세부 사항으로 format="3" 이라고 패키지 설정 파일의 버전을 기재한다. ROS 2는 3를 사용하면 된다.
<name>	패키지의 이름이다. 패키지를 생성할 때 입력한 패키지 이름이 사용된다. 다른 옵션도 마찬가지지만 이는 사용자가 원할 때 언제든지 변경할 수 있다.
<version>	패키지의 버전이다. 자유롭게 지정할 수 있는데 나중에 패키지를 바이너리 패키지로 공개한다면 버전 관리에 사용되므로 신중할 필요가 있다.
<description>	패키지의 간단한 설명이다. 보통 2~3 문장으로 기술한다.
<maintainer>	패키지 관리자의 이름과 이메일 주소를 기재한다.
<license>	라이선스를 기재한다. Apache 2.0, BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3, Proprietary 등을 기재하면 된다.
<url>	패키지를 설명하는 웹 페이지 또는 버그 관리, 소스 코드 저장소 등의 주소를 기재한다. 이 종류에 따라 type에 website, bugtracker, repository를 대입하면 된다.
<author>	패키지 개발에 참여한 개발자의 이름과 이메일 주소를 적는다. 복수의 개발자가 참여한 경우에는 바로 다음 줄에 <author> 태그를 이용하여 추가로 넣음
<buildtool_depend>	빌드 툴의 의존성을 기술한다.
<build_depend>	패키지를 빌드할 때 필요한 의존 패키지 이름을 적는다.
<exec_depend>	패키지를 실행할 때 필요한 의존 패키지 이름을 적는다.
<test_depend>	패키지를 테스트할 때 필요한 의존 패키지 이름을 적는다.
<export>	위에서 명시하지 않은 확장 태그명을 사용할 때 쓰인다. 빌드 타입을 적는 <build_type>, RViz 플러그인에 사용되는 <rviz>, RQt 플러그인에 사용되는 <rqt_gui>, deprecated되는 패키지일 경우 유저에게 알릴 수 있는 <deprecated> 태그 등이 있다.

ROS2 실습 – 빌드 설정파일(CMakeList.txt)

ROS 2의 빌드 시스템인 ament에서는 **C++ 프로그래밍 언어**를 사용한 패키지나 RQt Plugin의 경우 CMake(Cross Platform Make)를 이용하고 있고 패키지 폴더의 `**CMakeLists.txt**`라는 파일에 **빌드 환경을 기술**하여 사용하고 있다.

이 빌드 설정 파일에 실행 파일 생성, 의존성 패키지 우선 빌드, 링크 생성 등을 설정하게 되어 있다. ROS에서 CMake를 이용하는 이유는 ROS 패키지를 멀티 플랫폼에서 빌드할 수 있게 하기 위함이다. Make가 유닉스 계열만 지원하는 것과 달리, CMake는 유닉스 계열인 리눅스, BSD, OS X뿐만 아니라 윈도우즈 계열도 지원하기 때문이다.

또한 CMakeLists.txt은 Visual Studio, Eclipse, Qt Creator 등 다양한 IDE에서 기본으로 지원하여 쉽게 사용할 수 있다.

```
cmake_minimum_required(VERSION 3.5)
project(my_first_ros_rclcpp_pkg)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # uncomment the line when a copyright and license is not present in all source files
  #set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # uncomment the line when this package is not in a git repo
  #set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

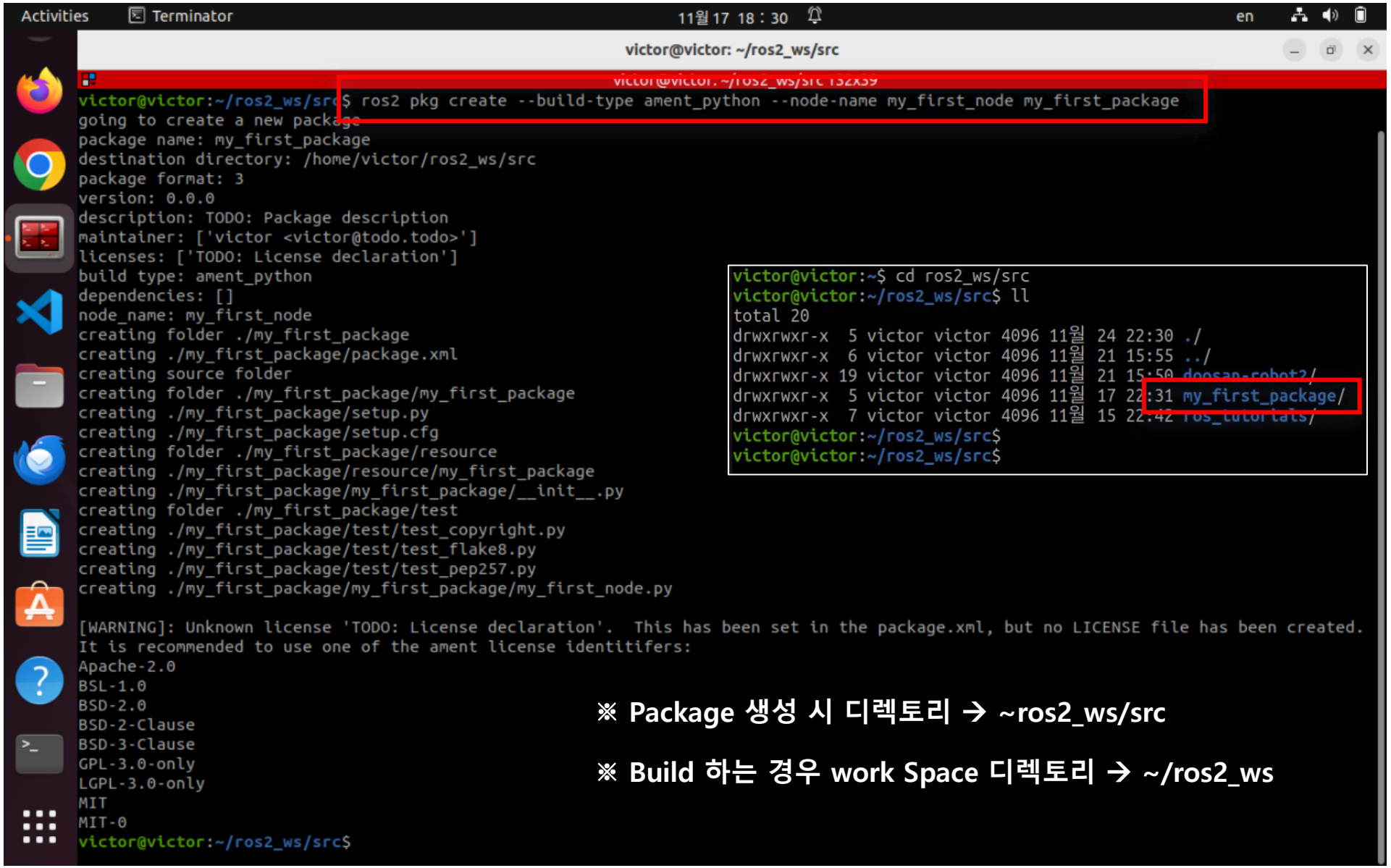
ROS2 실습 – 패키지 설정파일(package.xml)

항목	내용
name	패키지의 이름
Version	패키지의 버전
Packages	의존하는 패키지, 하나씩 나열해도 되지만 `find_packages()`를 기입해주면 자동으로 의존하는 패키지를 찾아준다.
data_files	이 패키지에서 사용되는 파일들을 기입하여 함께 배포한다. `ROS`에서는 주로 `resource` 폴더 내에 있는 `ament_index`를 위한 패키지의 이름의 빈 파일이나 `package.xml`, `*.launch.py`, `*.yaml` 등을 기입한다.
install_requires	의존하는 패키지, 이 패키지를 `pip`을 통해 설치할 때 이곳에 기술된 패키지들을 함께 설치하게 된다. `ROS`에서는 `pip`로 설치하지 않기에 `setuptools`, `launch`만을 기입해준다.
tests_require	테스트에 필요한 패키지, `ROS`에서는 `pytest`를 사용한다.
zip_safe	설치시 zip 파일로 아카이브할지 여부를 설정한다.
author **author_email** **maintainer** **maintainer_email**	저작자, 관리자의 이름과 이메일을 기입한다.
Keywords	이 패키지의 키워드, Python Package Index (PyPI) 배포시 검색하여 이 패키지를 찾을 수 있도록 한다.
Classifiers	PyPI에 등록될 메타 데이터 설정으로 `PyPI` 페이지의 좌측 Meta란에서 확인 가능하다.
Description	패키지 설명을 기입한다.
License	라이선스 종류를 기입한다.
entry_points	플랫폼 별로 콘솔 스크립트를 설치하도록 콘솔 스크립트 이름과 호출 함수를 기입한다.
name	패키지의 이름

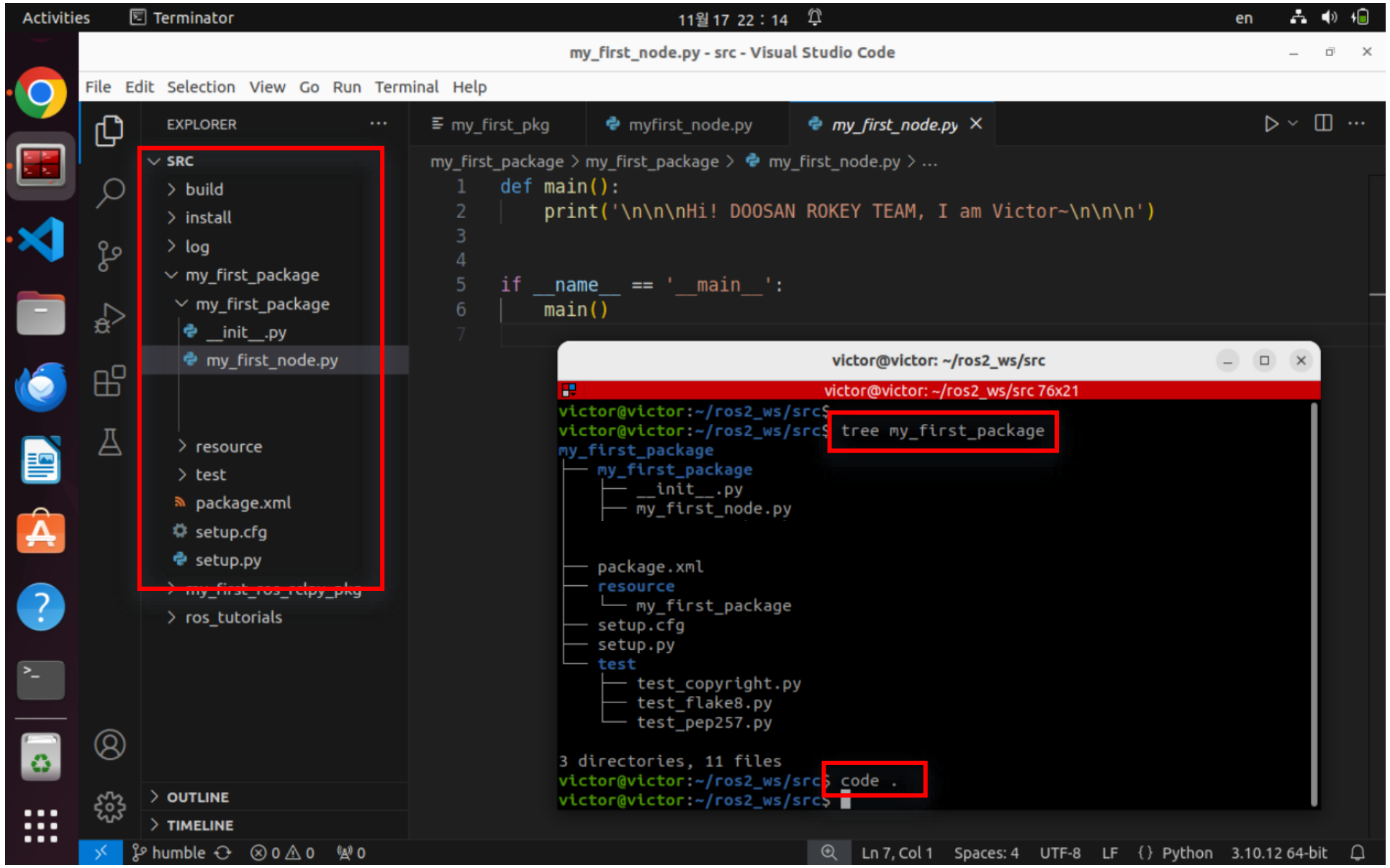
ROS2 실습 – 파이썬 패키지 설정파일(Setup.py)

```
from setuptools import setup

package_name = 'my_first_ros_rclpy_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pyo',
    maintainer_email='victor@datacore.kr,
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

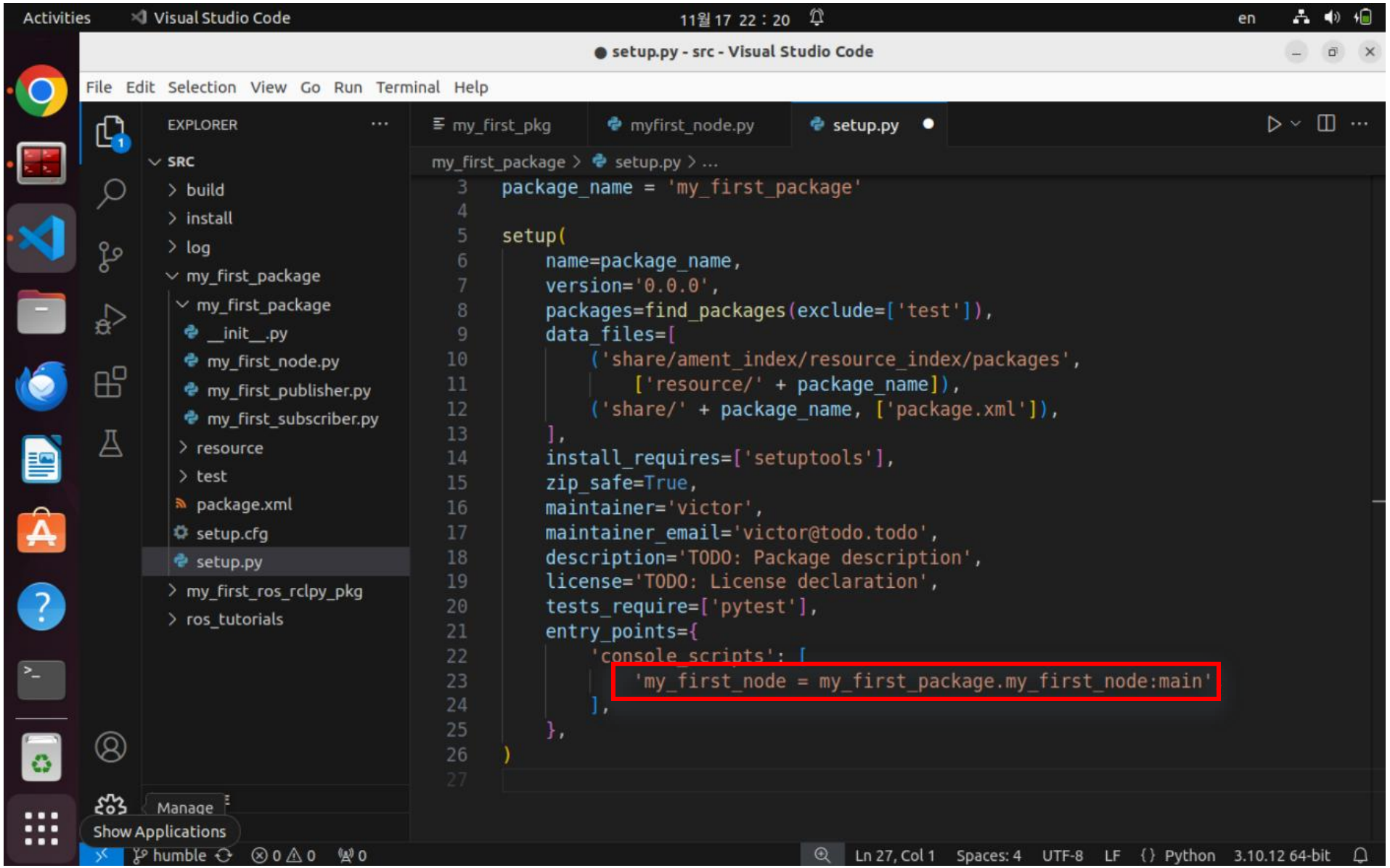
ROS2 실습 – Publisher & Subscriber(my_first_package)



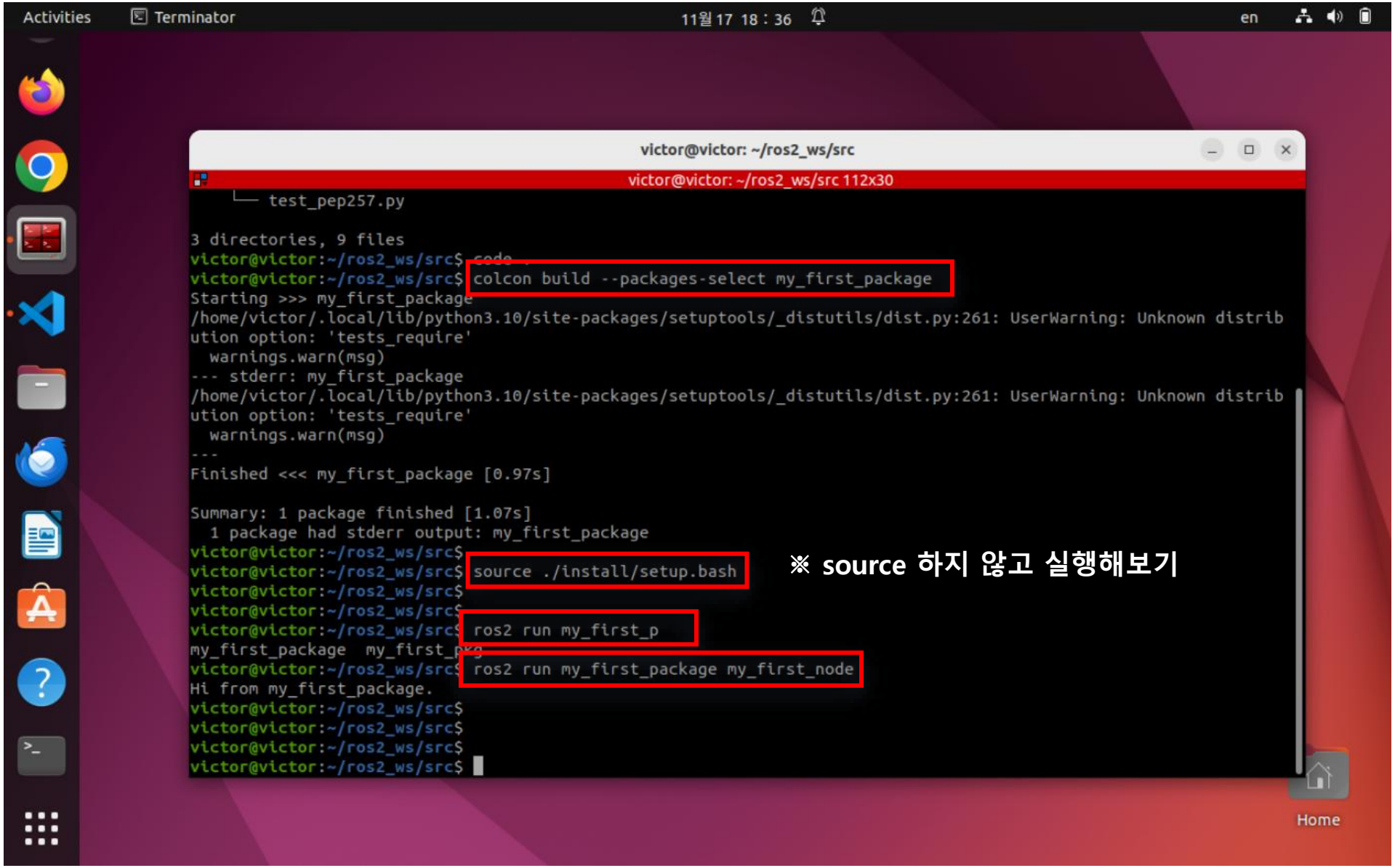
ROS2 실습 – Publisher & Subscriber(my_first_package)



ROS2 실습 – Publisher & Subscriber(my_first_package)



ROS2 실습 – Publisher & Subscriber(my_first_package)

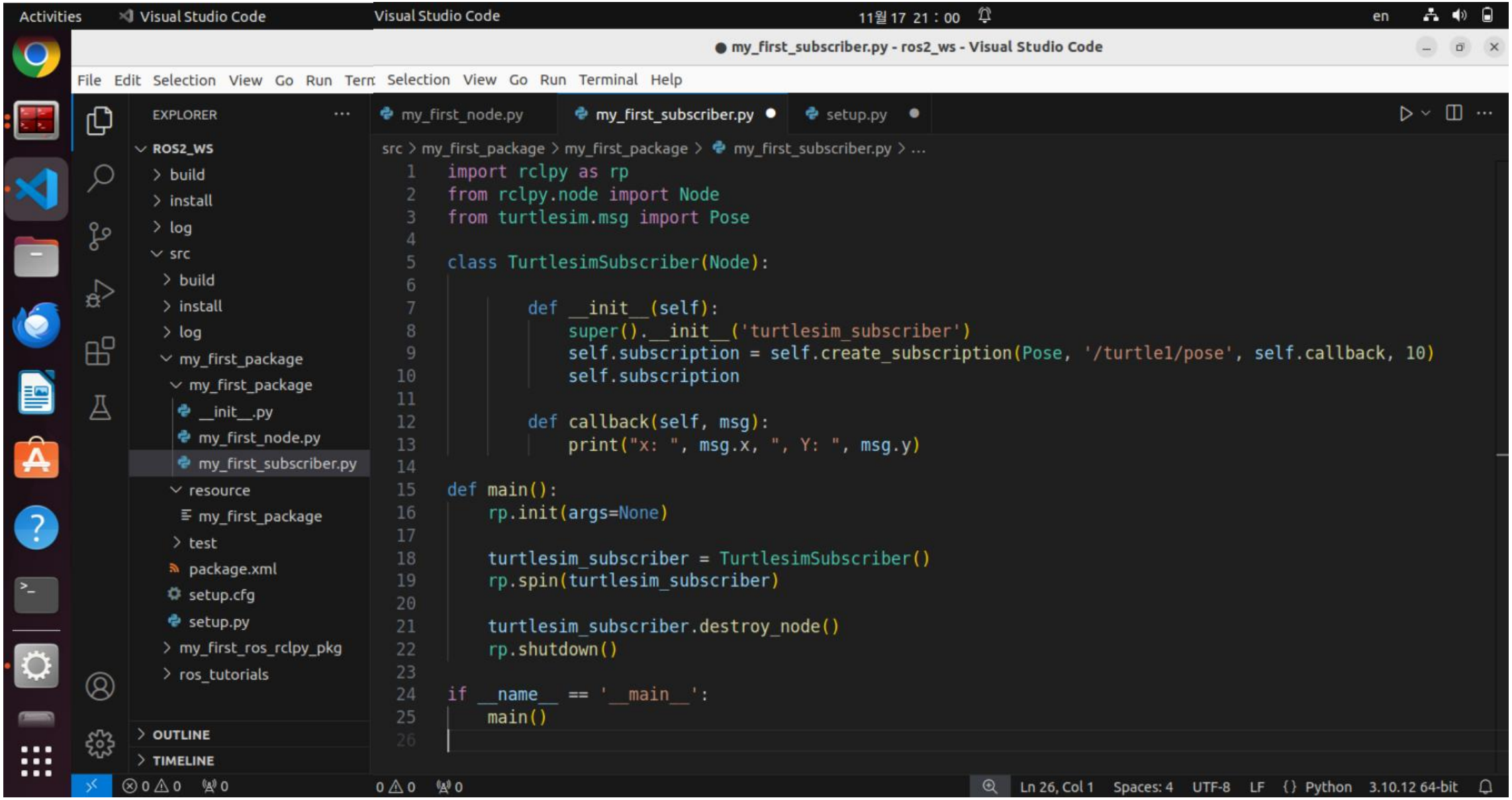


ROS2 실습 – Publisher & Subscriber(my_first_package)

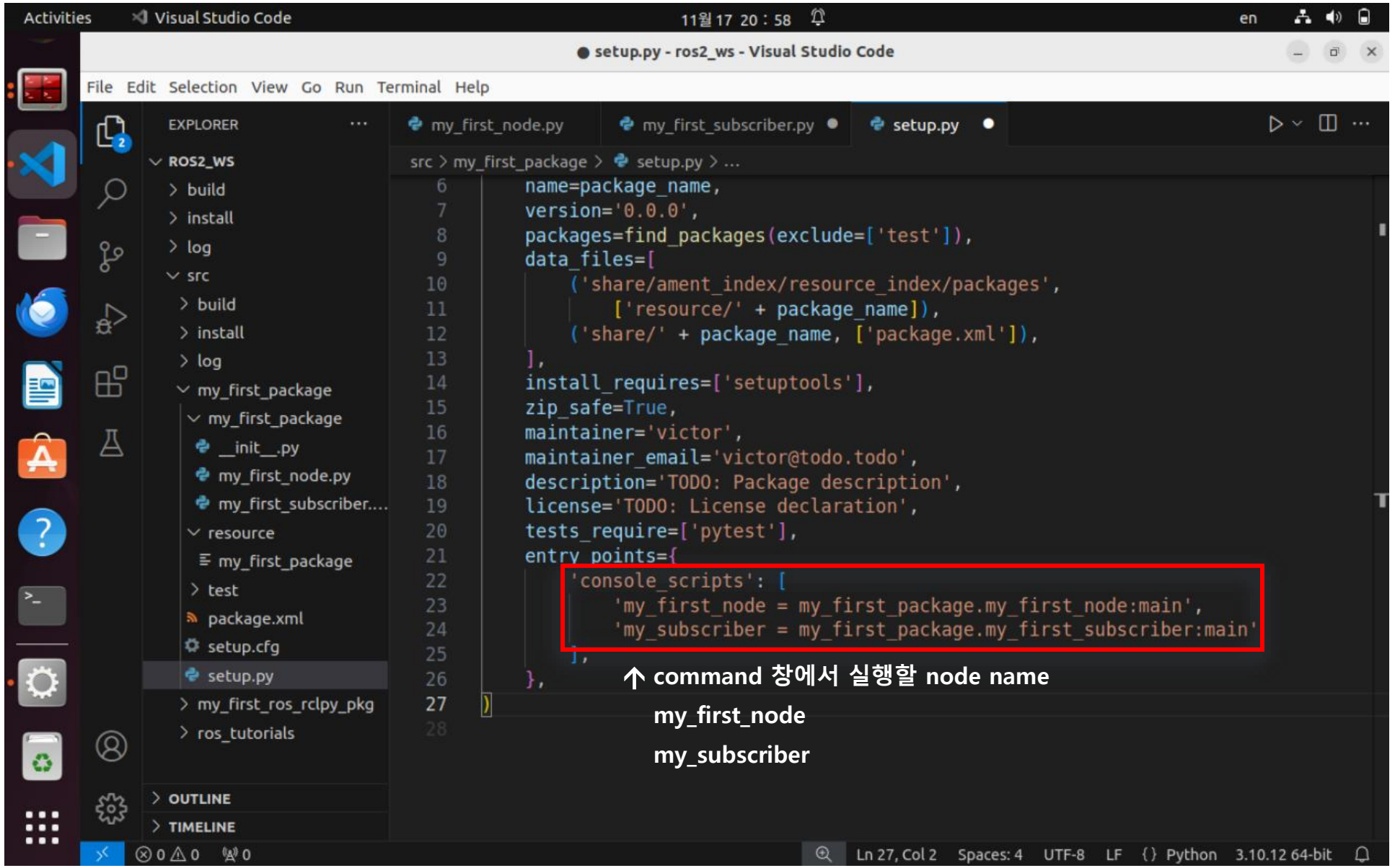
※ .bashrc example

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi


source /opt/ros/humble/setup.bash
alias sb="source ~/.bashrc; echo \"bashrc is reloaded\""
alias humble="source /opt/ros/humble/setup.bash; ros_domain; echo \"ROS2 humble is activated.\""
alias ros_domain="export ROS_DOMAIN_ID=32"
alias ros2_ws="humble; source ~/ros2_ws/install/local_setup.bash; echo \"ros2_ws workspace is activated.\""
```

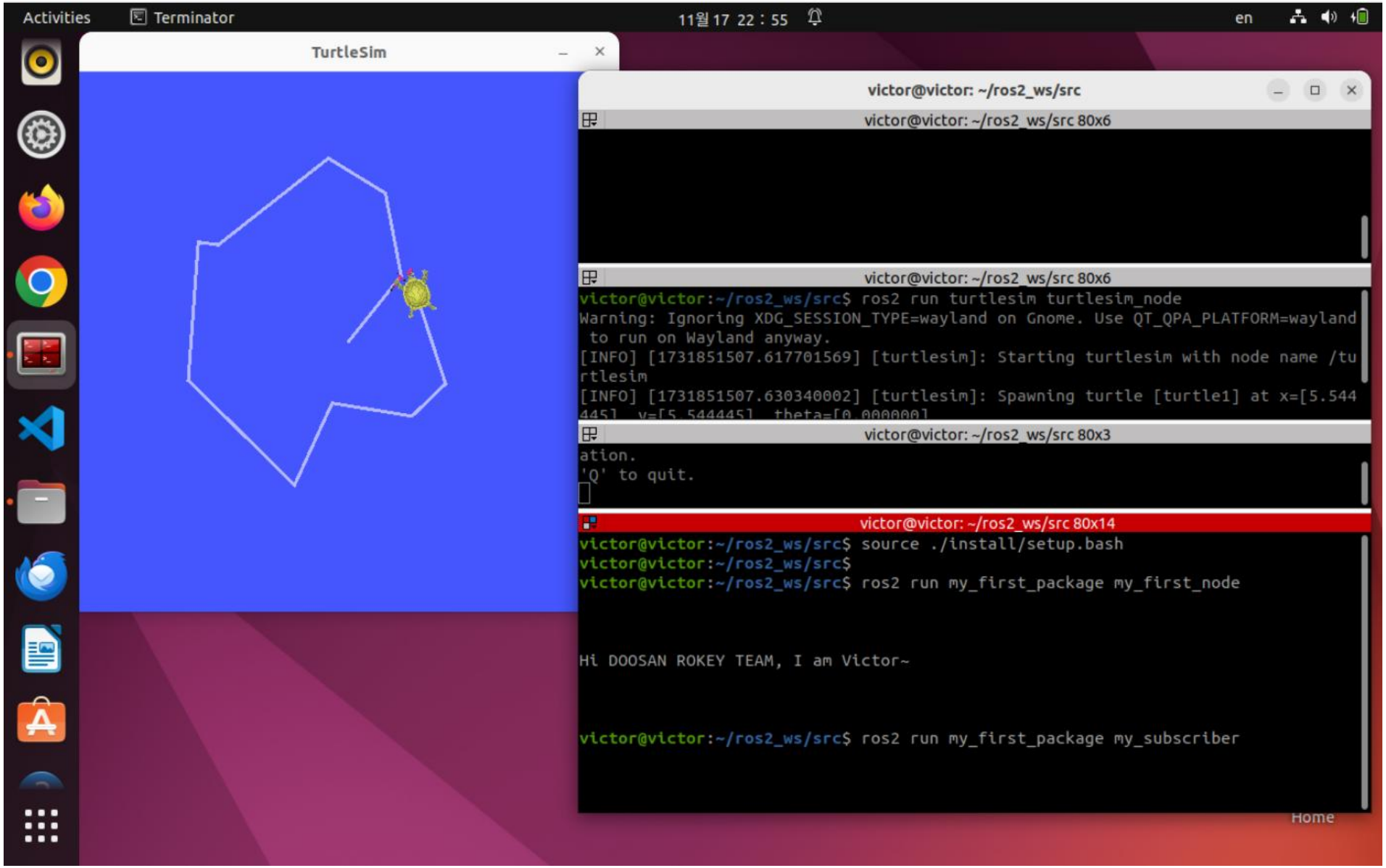
ROS2 실습 – Publisher & Subscriber(my_first_package)



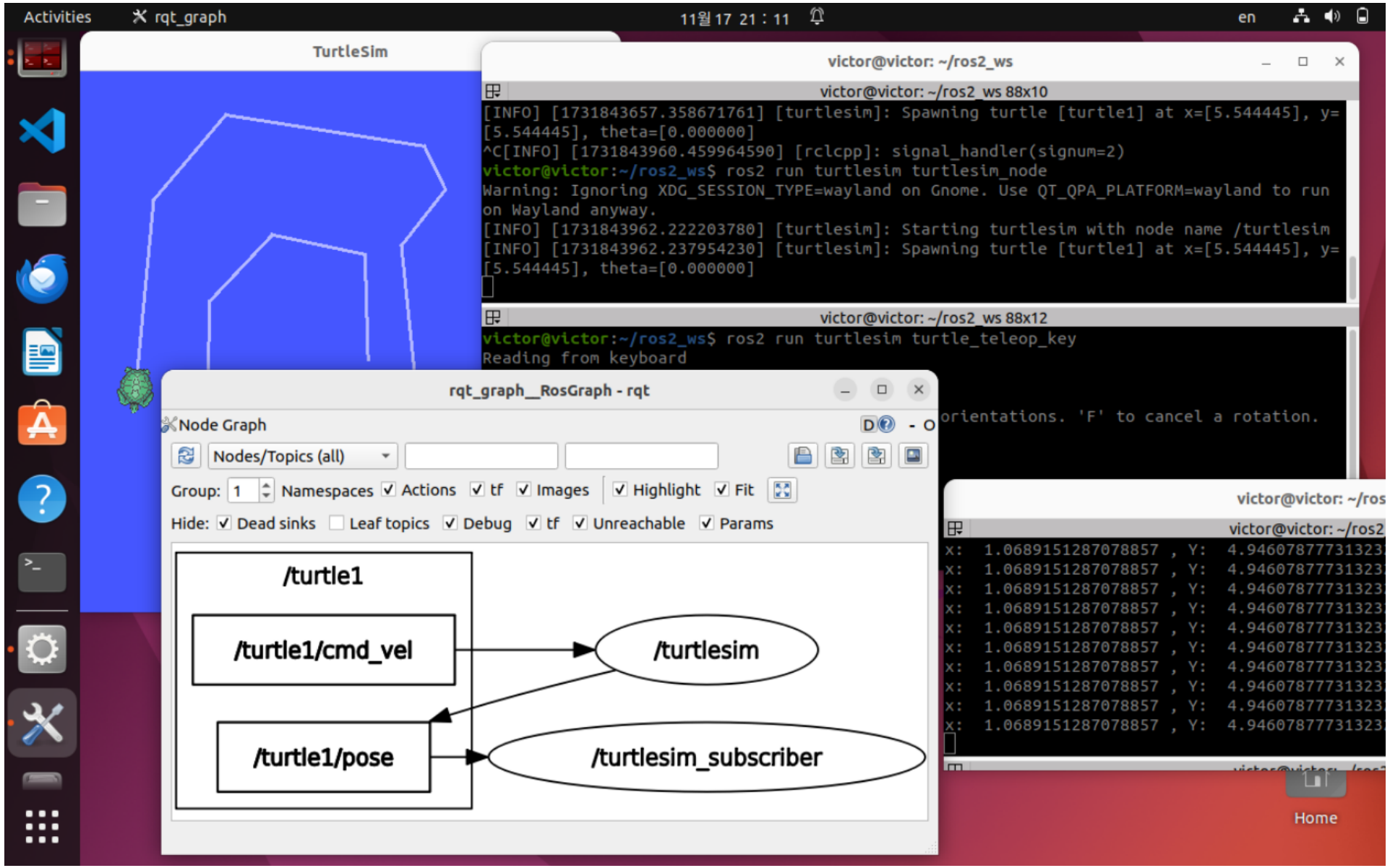
ROS2 실습 – Publisher & Subscriber(my_first_package)



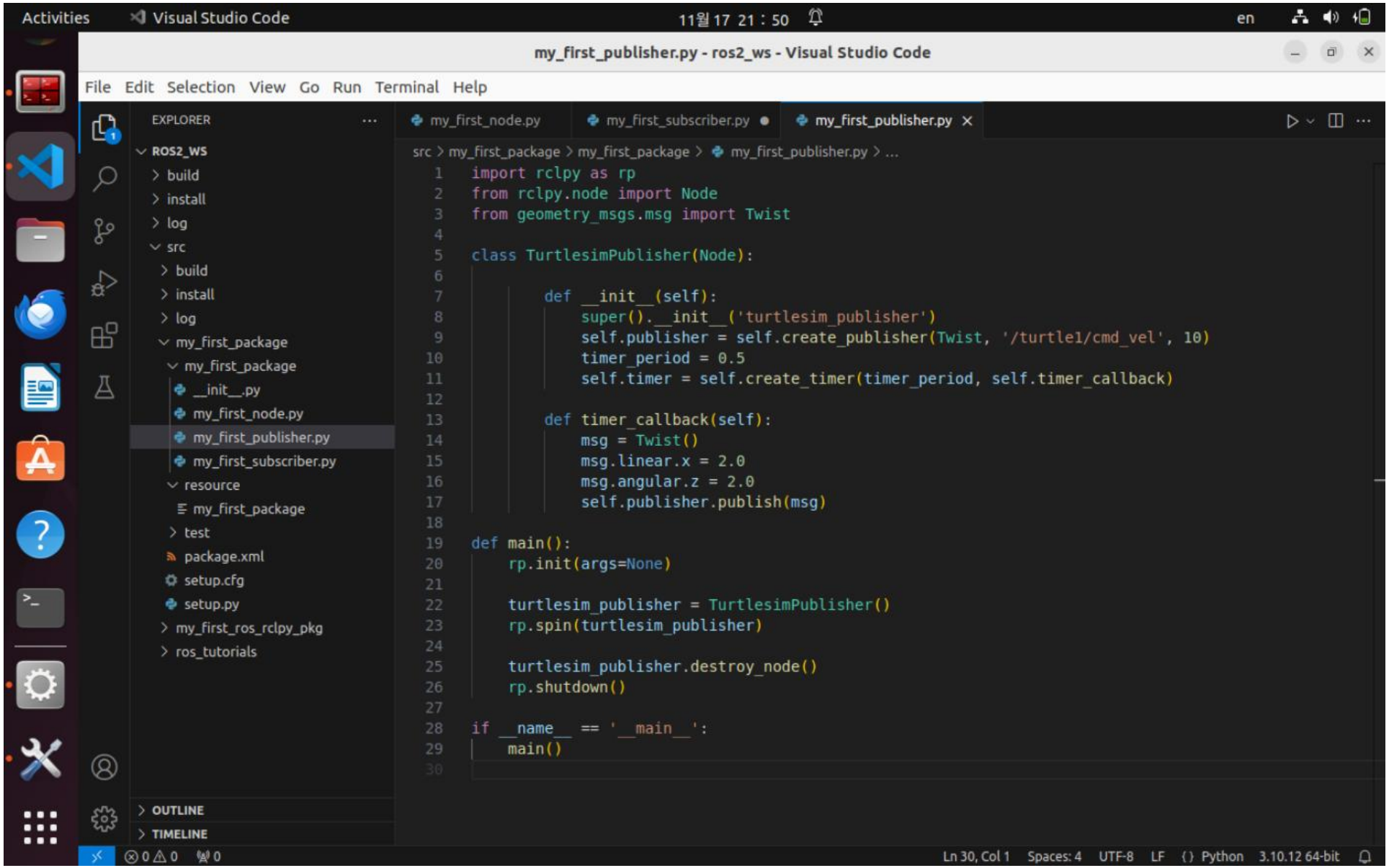
ROS2 실습 – Publisher & Subscriber(my_first_package)



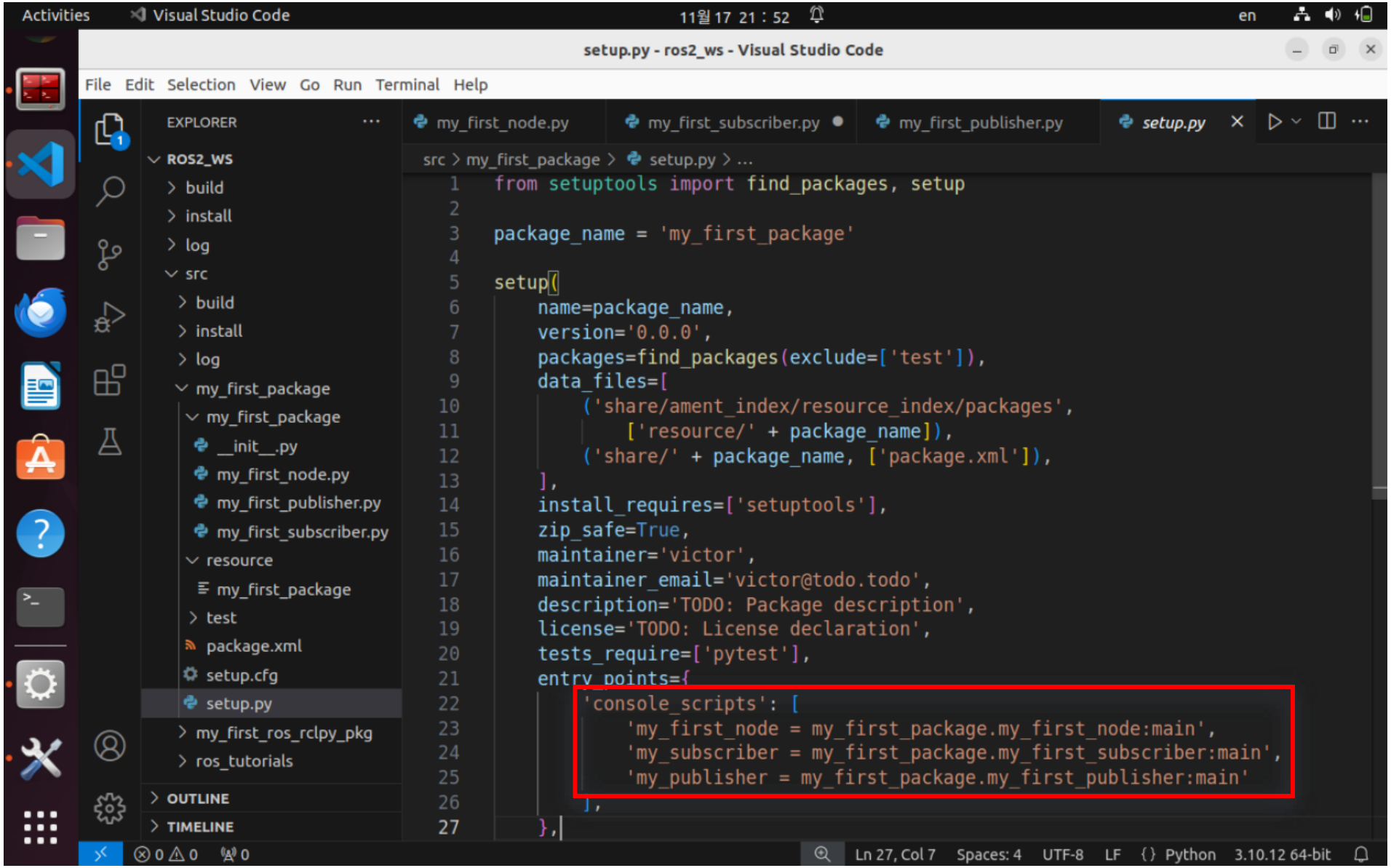
ROS2 실습 – Publisher & Subscriber(my_first_package)



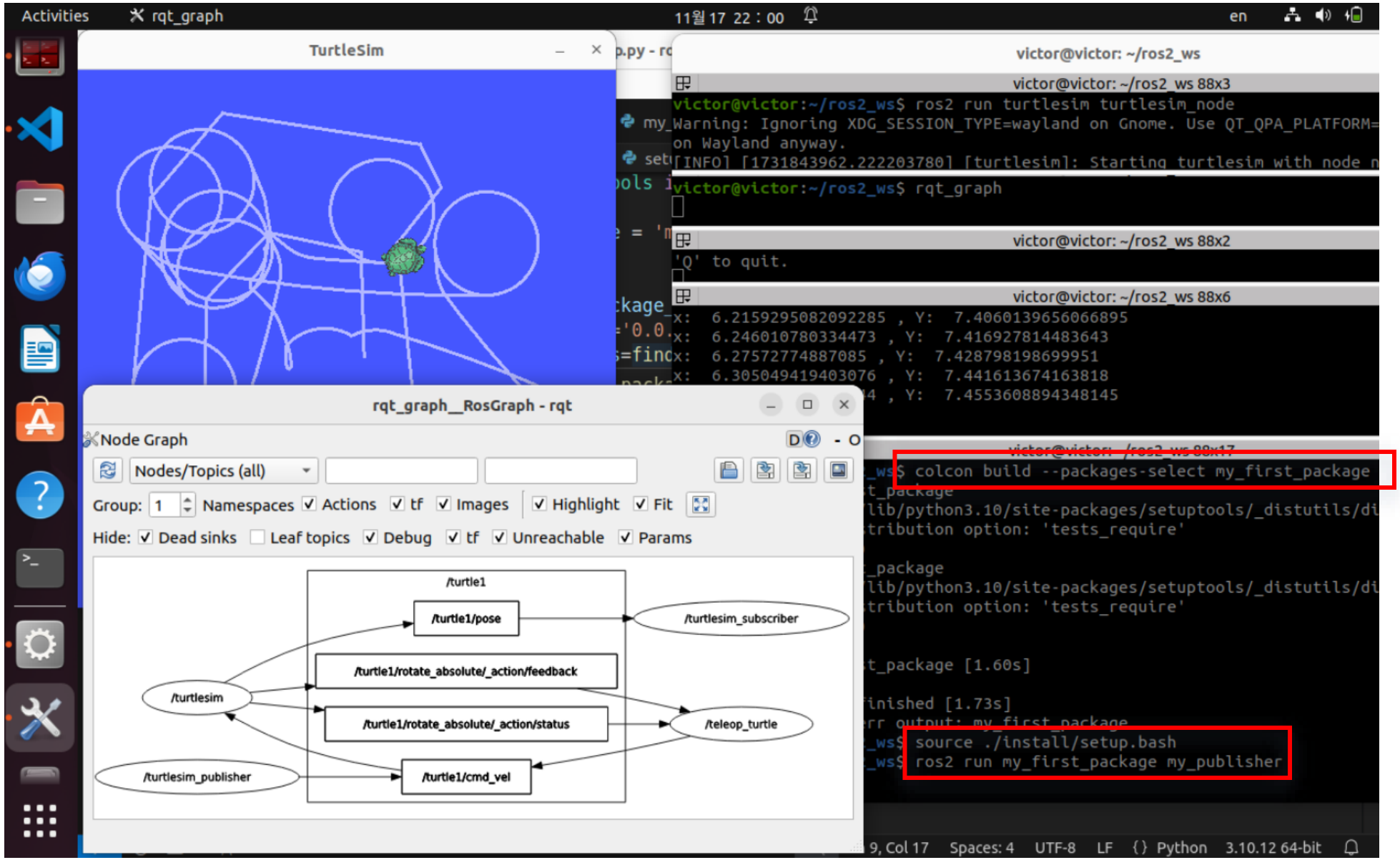
ROS2 실습 – Publisher & Subscriber(my_first_package)



ROS2 실습 – Publisher & Subscriber(my_first_package)



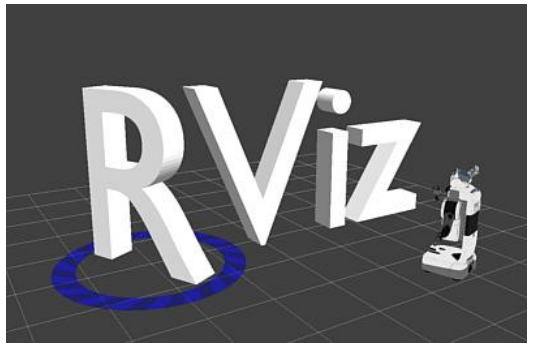
ROS2 실습 – Publisher & Subscriber(my_first_package)



ROS의 중요한 개발도구들

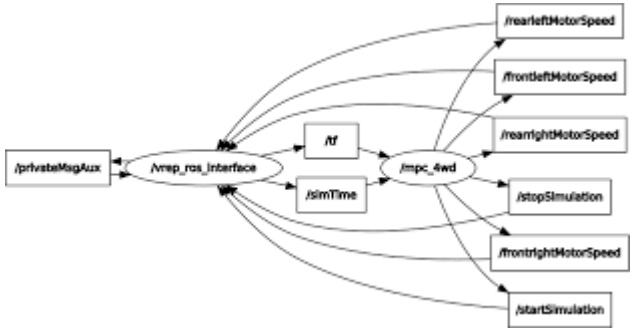
- ROS는 GUI기반이 아닌, 터미널에서 command line 방식으로 동작을 시킬 수 있음
- 여기에 더해서 ROS 사용의 효율을 높이기 위해서 다양한 개발도구를 제공함

rViz



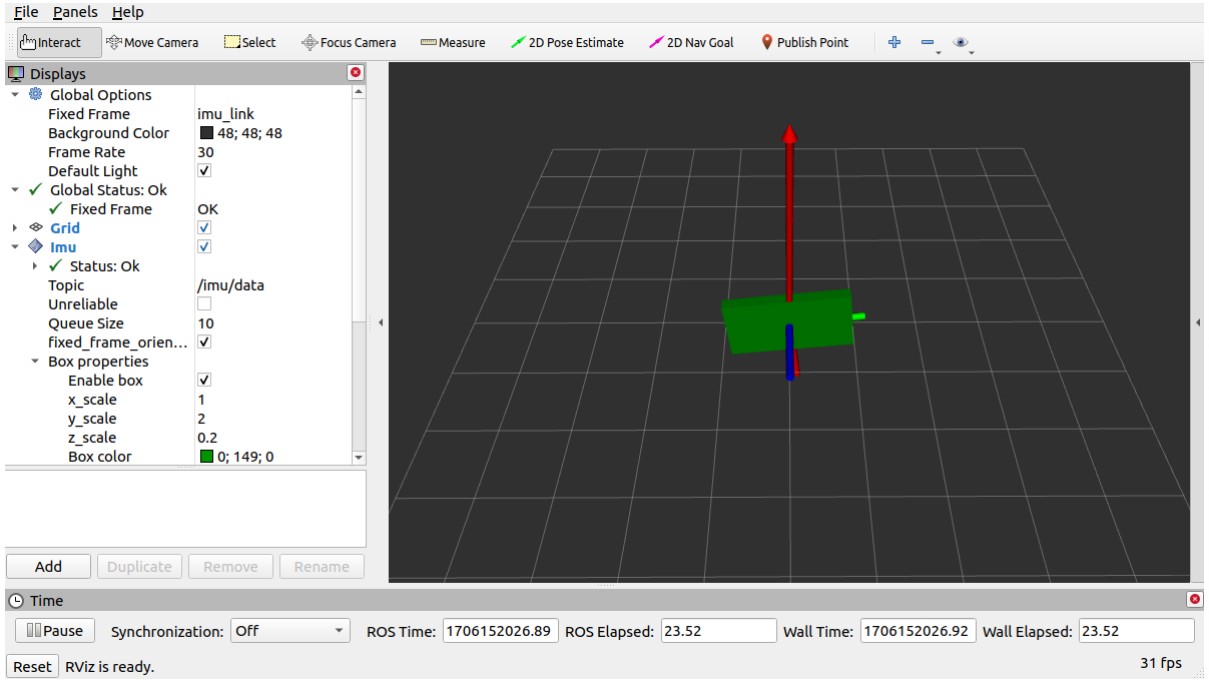
gazebo

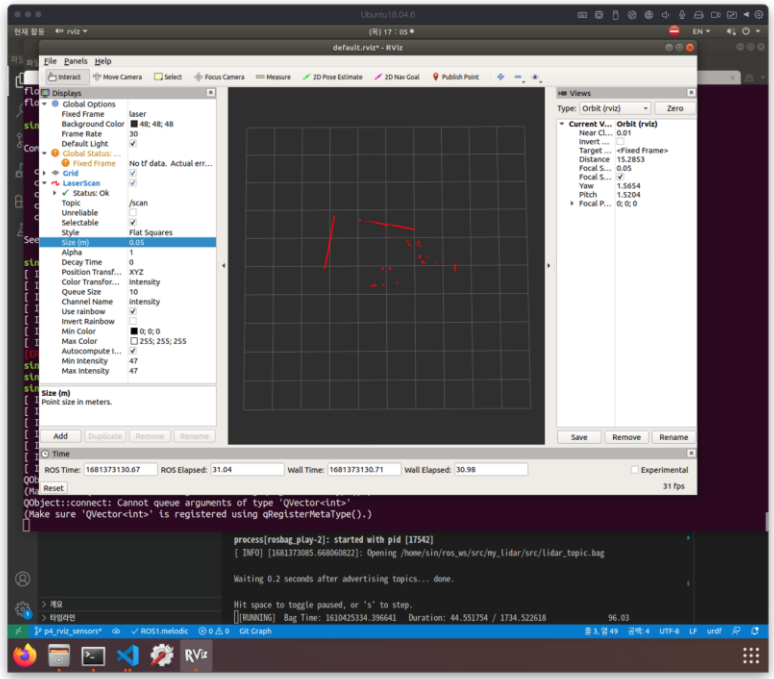
rqt



rViz (ROS Visualization Tool)

- rViz는 ROS에서 얻어지는 데이터를 시각화(visualization)하는 도구임
- IMU 데이터의 시각화
- URDF 파일을 이용한 로봇암 동작의 시각화
- 라이다 센서 데이터 시각화를 통한 SLAM 적용



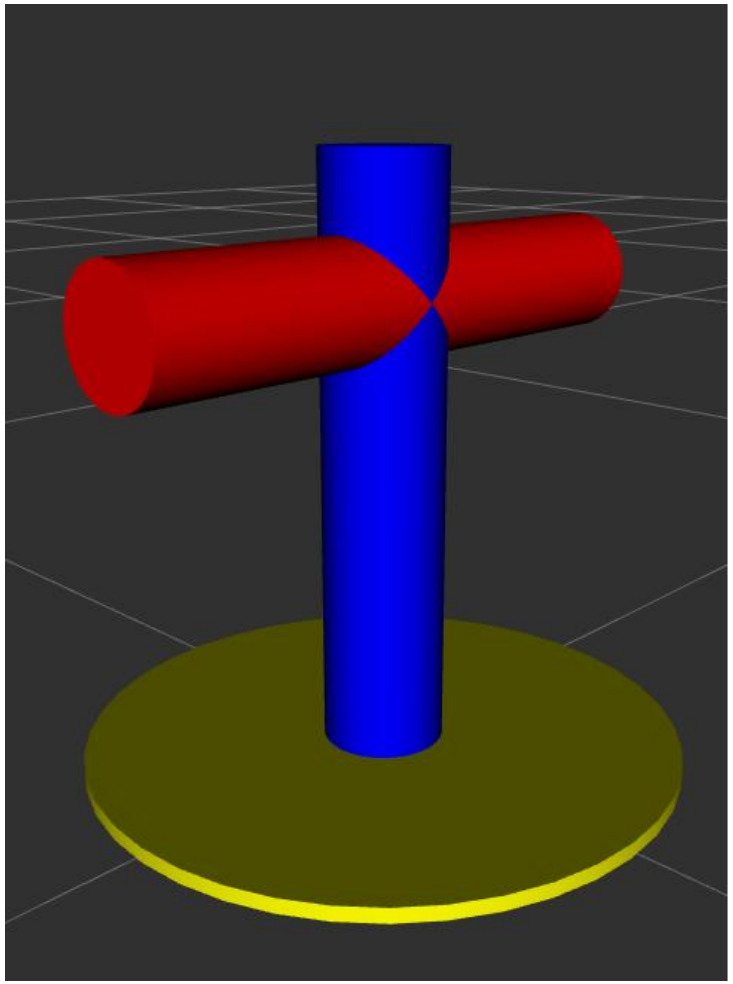


https://velog.io/@y2k4388/rViz%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%84%BC%EC%84%9C-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%8B%9C%EA%B0%81%ED%99%94

Robot URDF 파일을 통한 로봇암 시각화

- rViz의 기능 중 URDF 파일을 읽어서 로봇암의 동작을 시각화하는 기능이 있음
-
- URDF(Universal Robot description Format)이란?
  - URDF는 xml기반의 텍스트 파일로, 로봇의 형태와 동작을 정의한 파일임
  - ROS 초보자가 URDF 파일을 작성하기는 어렵지만,
    어떤 로봇에 URDF 파일이 존재하면,
  - 이 로봇을 rViz를 통해서 가상으로 동작 시켜 볼 수 있음

- URDF 파일로 할 수 있는 것
  - 로봇의 구조를 정의
  - 로봇 동작의 시각화
  - 로봇의 충돌 모델 정의



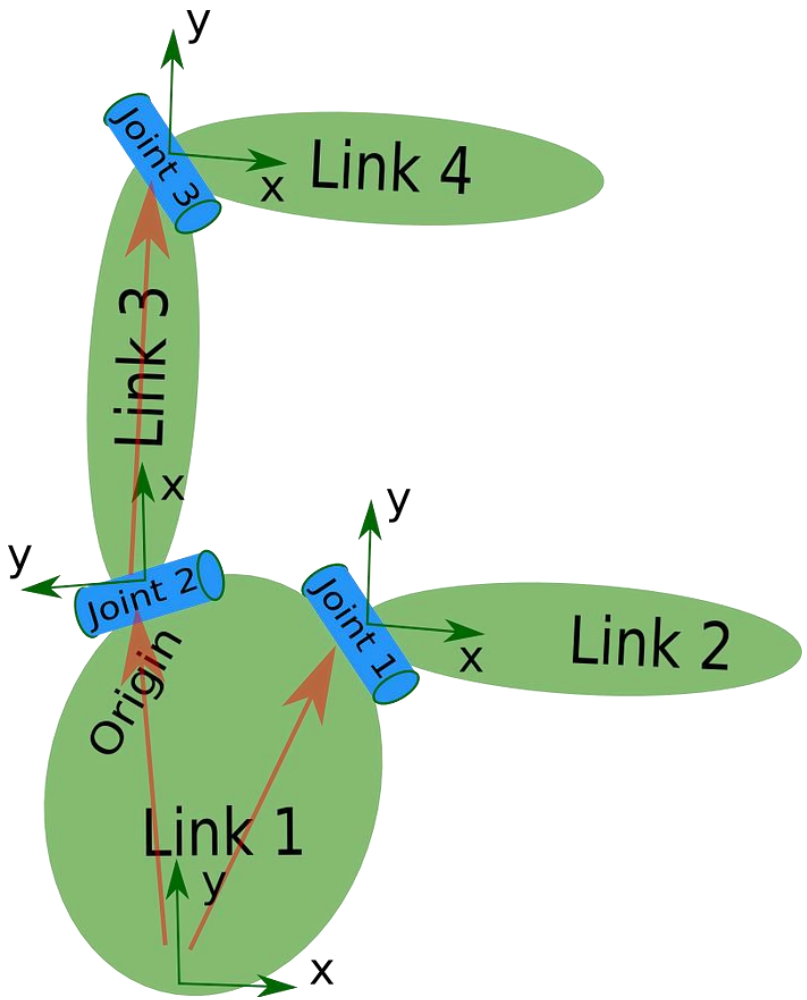
Robot URDF 파일의 구성

- URDF 파일은 다음과 같은 요소로 구성되어 있음
- Link, joint

- Link
  - 로봇을 구성하는 구성요소 중 하나, 3가지 속성이 있음
  - <inertial> 링크의 관성 정보. Link의 관성 중심, 질량, 관성계수 등을 기록함
  - <visual> rViz 같은 시각화 도구에서 로봇을 시가화 할 때 사용되는 속성들을 정의함
  - <collision> 물리적인 충돌 속성 정의, 충돌 모델 정의

- Joint
  - Link 와 link를 연결하는 로봇 구성요소. URDF는 6가지 joint 타입이 있음
  - <origin> 부모 link에서 자식 링크에 변환 정보
  - <parent> 부모 link 이름
  - <child> 자식 link 이름

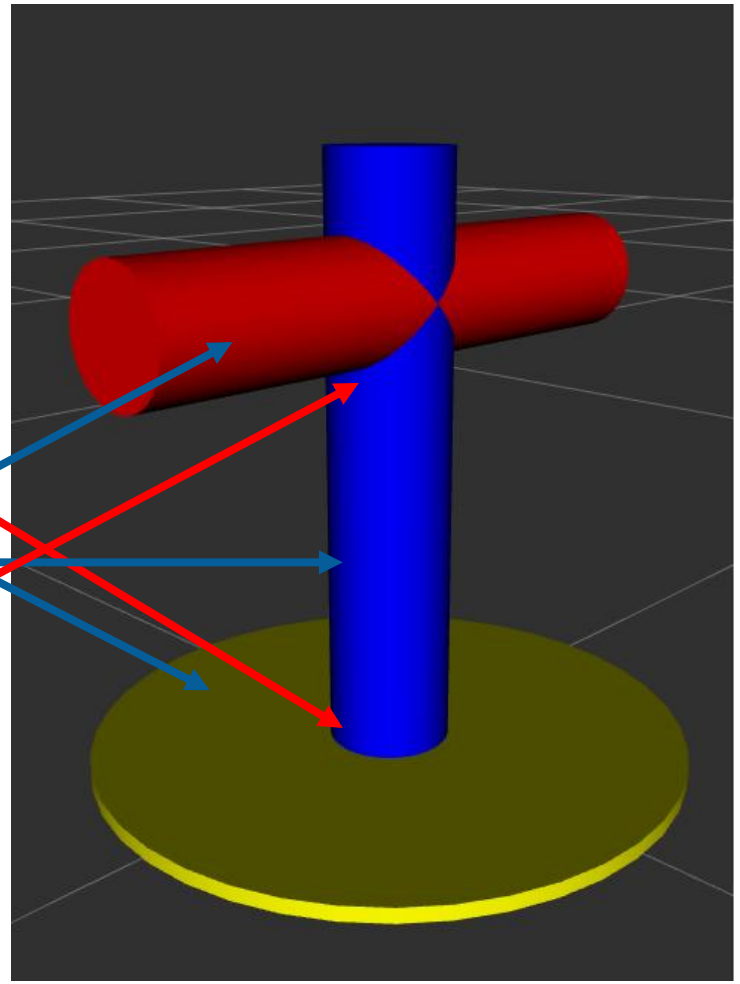
https://wiki.ros.org/urdf/Examples



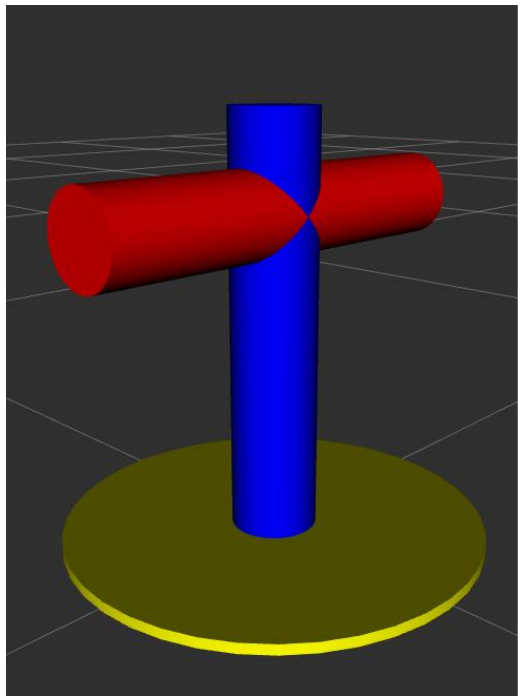
https://medium.com/newworld-kim/ros-urdf-b6979bfa31aa

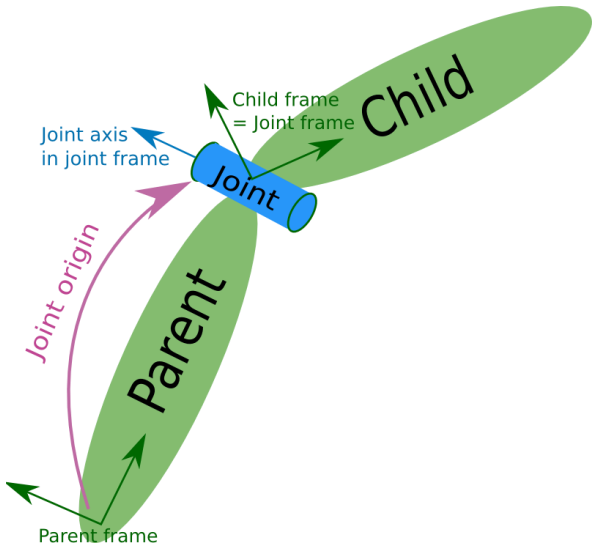
Robot URDF 파일의 예 – pan-tilt 로봇

```
<?xml version="1.0"?>
<robot name="pan_tilt">
      <link name="base_link">
      </link>
      <joint name="pan_joint" type="revolute">
        </joint>
      <link name="pan_link">
      </link>
      <joint name="tilt_joint" type="revolute">
        </joint>
      <link name="tilt_link">
      </link>
</robot>
```



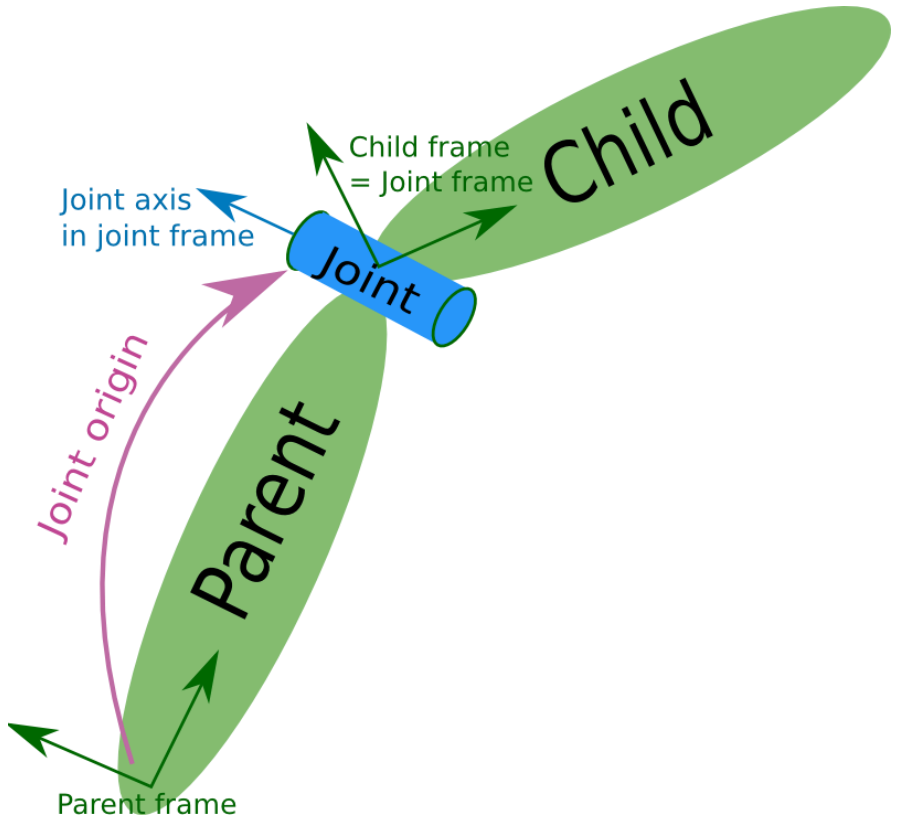
URDF 파일 pan-tilt 로봇 – link - joint 예





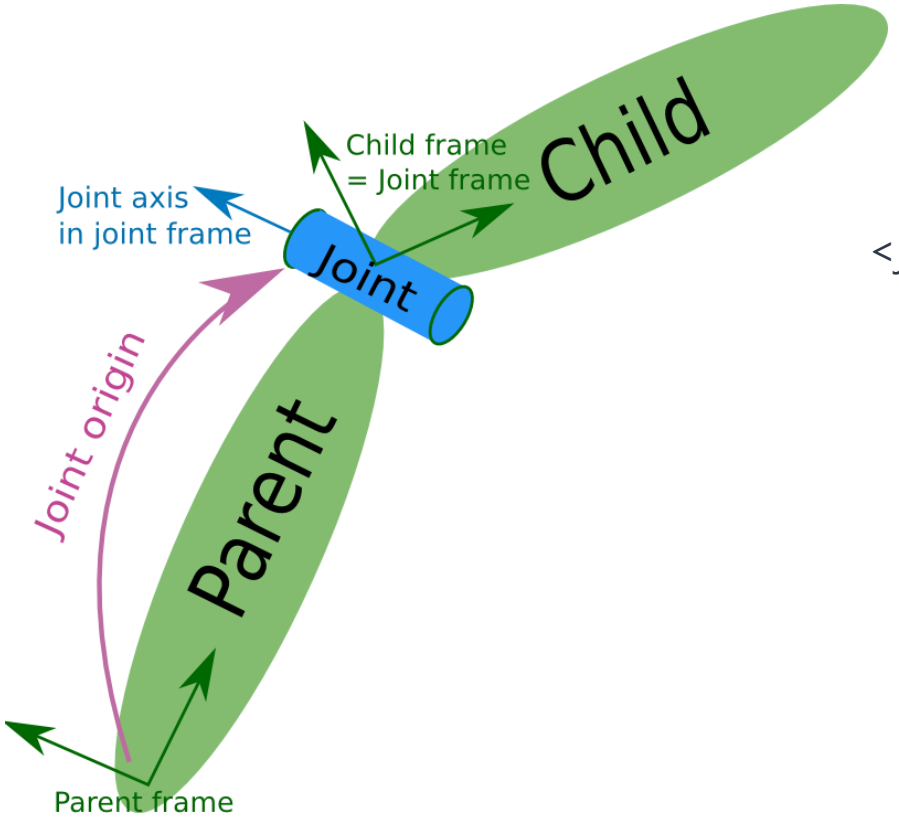

```
<link name="base_link">
   <visual>
    중간 생략
   </visual>
   <collision>
      중간 생략
   </collision>
    중간 생략
</link>
<joint name="pan_joint" type="revolute">
    <parent link="base_link"/>
    <child link="pan_link"/>
   중간 생략
</joint>

<link name="pan_link">
  중간 생략
</link>
```

URDF 파일 pan-tilt 로봇 – link 상세




```
<link name="base_link">
  <visual>
    <geometry>
          <cylinder length="0.01" radius="0.2"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <material name="yellow">
      <color rgba="1 1 0 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
          <cylinder length="0.03" radius="0.2"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </collision>
  <inertial>
          <mass value="1"/>
          <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
  </inertial>
</link>
```

URDF 파일 pan-tilt 로봇 – joint 상세




```
<joint name="pan_joint" type="revolute">
    <parent link="base_link"/>
    <child link="pan_link"/>
    <origin xyz="0 0 0.1"/>
    <axis xyz="0 0 1" />
    <limit effort="300" velocity="0.1" lower="-3.14" upper="3.14"/>
    <dynamics damping="50" friction="1"/>
  </joint>
```

ROKEY BOOT CAMP

수고하셨습니다.