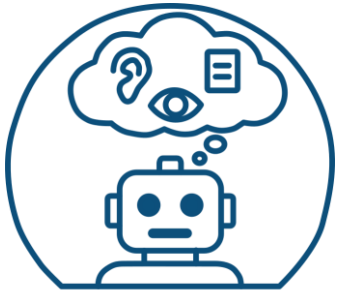
ROS2: Launch files, Parameters, and TF2

운영체제의 실제

안인규 (Inkyu An)



Creating a launch file

- This lecture uses the rqt_graph and turtlesim packages:
  - turtlesim 패키지의 세 개 노드로 이루어진 시스템을 실행
  - 목표는 두 개의 turtlesim 창을 실행하고, 한 거북이가 다른 거북이의 움직임을 따라 하도록 하는 것입니다.

- ROS2 launch files can be written in XML, YAML, and Python
- In this lecture, we will learn how to create the Python format launch file

Creating a launch file

- Create a new directory to store our launch files:

```
$ mkdir launch
```

- Write the launch file:
  - Put together a ROS 2 launch file using the turtlesim package and its executables

```
$ vim launch/turtlesim_mimic_launch.py
```

Creating a launch file

- Write the launch file:

```
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'),
            ]
        )
    ])
```

Creating a launch file

- Write the launch file:

```
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'),
            ]
        )
    ])
```

Launch the two turtlesim windows

The mimic node with the remaps

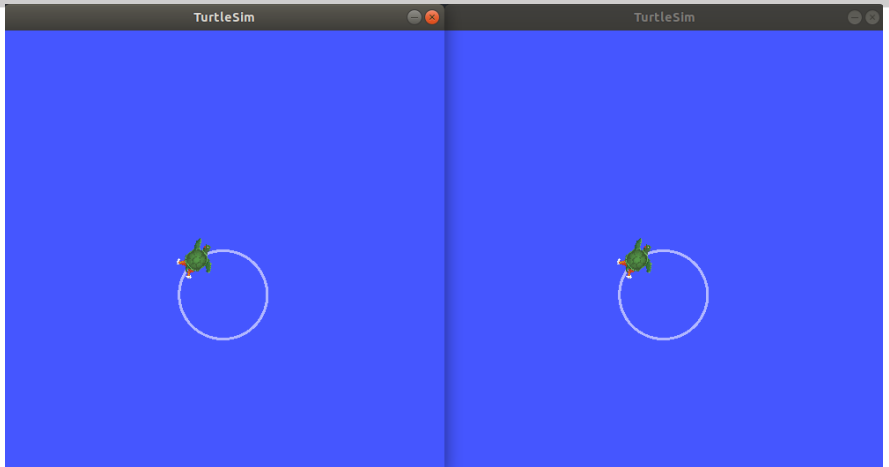
Creating a launch file

- Launch file created above:

```
$ cd launch
$ ros2 launch turtlesim_mimic_launch.py
```

- Then, run the ros2 topic pub command on the /turtlesim1/turtle1/cmd_vel topic to get the first turtle moving:

```
$ 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}}"
```



Creating a launch file

- It is possible to launch a launch file directly:

```
$ ros2 launch py_pubsub turtlesim_mimic_launch.py
```

To do that, we have to modify the "setup.py" file & re-build the package!

```
import os
from glob import glob
from setuptools import setup

...

setup(
    # Other parameters ...
    data_files=[
        # ... Other data files
        # Include all launch files.
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*')))
    ]
)
```

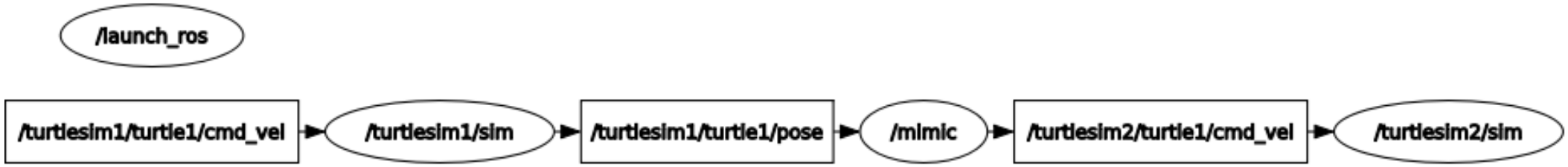
- Then, build it again:

```
$ cd ~/ros2_ws; colcon build
```

Creating a launch file

- get a better idea of the relationship between the nodes in your launch file.

```
$ ros2 run rqt_graph rqt_graph
```



Using Parameters in a Class

- When making our nodes, we will sometimes need to add parameters that can be set from the launch file
- How to create the parameters in a Python class, and how to set them in a launch file?

```python
import rclpy
import rclpy.node

class MinimalParam(rclpy.node.Node):
    def __init__(self):
        super().__init__('minimal_param_node')

        self.declare_parameter('my_parameter', 'world')

        self.timer = self.create_timer(1, self.timer_callback)

    def timer_callback(self):
        my_param = self.get_parameter('my_parameter').get_parameter_value().string_value

        self.get_logger().info('Hello %s!' % my_param)

        my_new_param = rclpy.parameter.Parameter(
            'my_parameter',
            rclpy.Parameter.Type.STRING,
            'world'
        )
        all_new_parameters = [my_new_param]
        self.set_parameters(all_new_parameters)

def main():
    rclpy.init()
    node = MinimalParam()
    rclpy.spin(node)

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

self.get_parameter('my_parameter').value 도 가능

Using Parameters in a Class

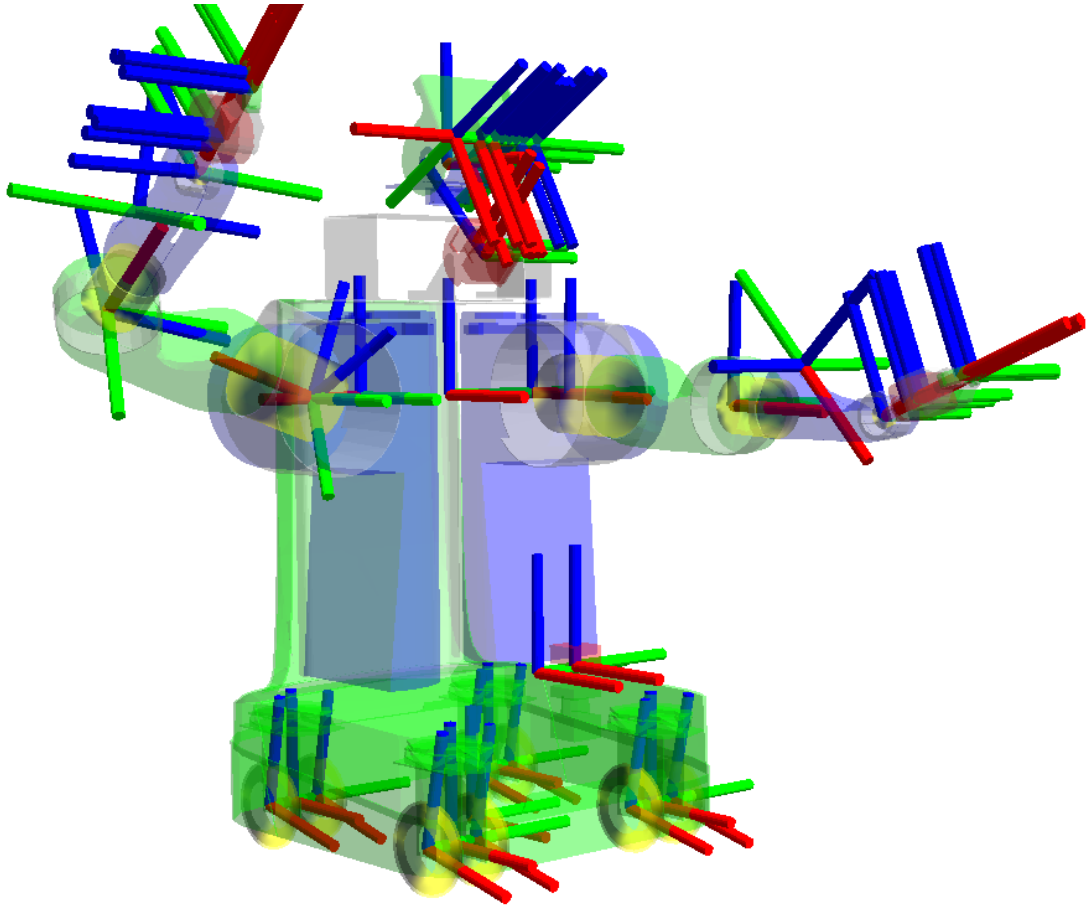
- When making our nodes, we will sometimes need to add parameters that can be set from the launch file
- How to create the parameters in a Python class, and how to set them in a launch file?
- Is it possible to control the timer period of the talker node in the py_pubsub package using a parameter?

```
mair@mair-laptop:~/ros2_ws$ ros2 run py_pubsub talker
[INFO] [1759147131.308747623] [minimal_publisher]: Publishing: "0"
[INFO] [1759147131.799712015] [minimal_publisher]: Publishing: "1"
[INFO] [1759147132.299277349] [minimal_publisher]: Publishing: "2"
[INFO] [1759147132.800260152] [minimal_publisher]: Publishing: "3"
[INFO] [1759147133.300008012] [minimal_publisher]: Publishing: "4"
[INFO] [1759147133.799871795] [minimal_publisher]: Publishing: "5"
[INFO] [1759147134.299919008] [minimal_publisher]: Publishing: "6"
[INFO] [1759147134.799904955] [minimal_publisher]: Publishing: "7"
[INFO] [1759147135.299915458] [minimal_publisher]: Publishing: "8"
[INFO] [1759147135.798477091] [minimal_publisher]: Publishing: "9"
[INFO] [1759147136.300218053] [minimal_publisher]: Publishing: "10"
[INFO] [1759147136.421515516] [minimal_publisher]: Updated timer period to 5.000s
[INFO] [1759147141.422144373] [minimal_publisher]: Publishing: "11"
```

→ **os_practice_lec8.zip 확인!**

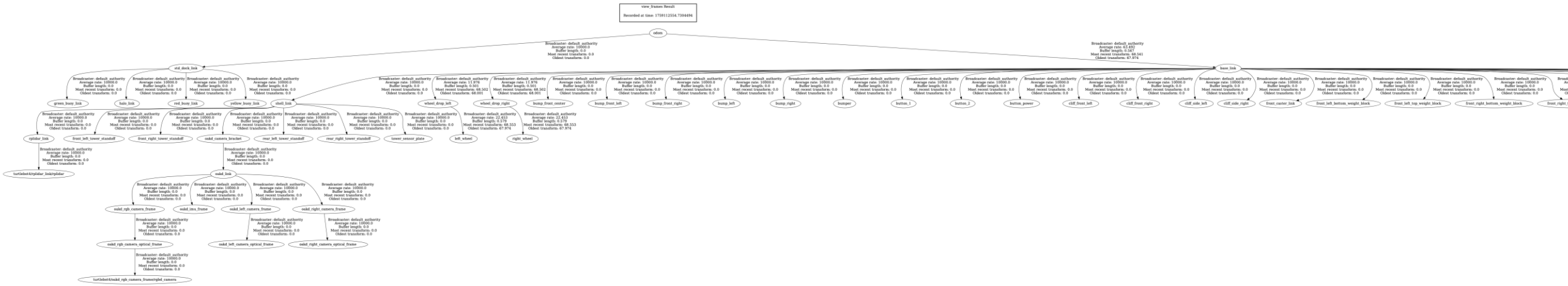
TF2

- A robot is made up of many parts and sensors (a body, wheels, arms, cameras, and LiDAR)
- Each of these has its own unique coordinate frame
  - "an object is located 1 meter in front of the camera,"
  - "the camera is mounted 20 cm above the center of the body."
  - "So how far is the object from the robot's body center?"
- We needs to understand the **geometric relationships (translations and rotations)** between the coordinate frames.

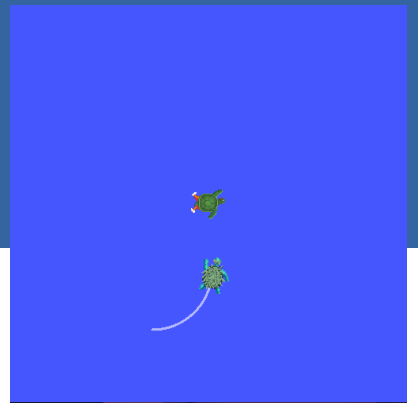


TF2

- **tf2** is the standard library that manages and computes these complex coordinate frame relationships in real time
- It organizes all coordinate frame information of the robot into a single tree structure, enabling easy transformation of points or data from any coordinate frame to another



TF2



- Running the demo:

```
$ sudo apt-get install ros-humble-rviz2 ros-humble-turtle-tf2-py ros-humble-tf2-ros
ros-humble-tf2-tools ros-humble-turtlesim
$ ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```

- In the second terminal window, type the following command:

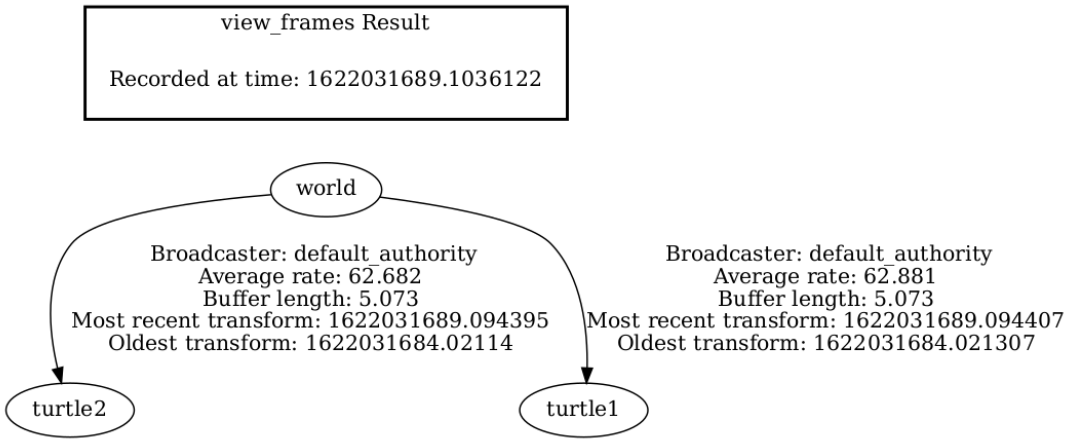
```
$ ros2 run turtlesim turtle_teleop_key
```

  - We can drive the turtle using the keyboard arrows keys
  - We can also see that one turtle continuously moves to follow the turtle you are driving around

TF2

- What is happening?
  - This demo is using tf2 library to create three coordinate frames: a world frame, a turtle1 frame, and a turtle2 frame
  - The tf2 broadcaster publishs the turtle coordinate frame and a tf2 listener to compute the difference in the turtle frames and move one turtle to follow the other
- Use tf2 tools to look at what tf2 is doing behind the scenes:

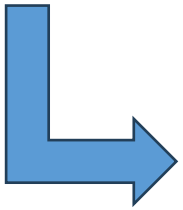
```
$ ros2 run tf2_tools view_frames
```



TF2

- Use tf2_echo
  - tf2_echo reports the transform between any two frames broadcast over ROS:

```
$ ros2 run tf2_ros tf2_echo [source_fram] [target_gram]
```



```
$ ros2 run tf2_ros tf2_echo turtle2 turtle1
At time 1683385337.850619099
- Translation: [2.157, 0.901, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.172, 0.985]
- Rotation: in RPY (radian) [0.000, -0.000, 0.345]
- Rotation: in RPY (degree) [0.000, -0.000, 19.760]
- Matrix:
  0.941 -0.338  0.000  2.157
  0.338  0.941  0.000  0.901
  0.000  0.000  1.000  0.000
  0.000  0.000  0.000  1.000
At time 1683385338.841997774
- Translation: [1.256, 0.216, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.016, 1.000]
- Rotation: in RPY (radian) [0.000, 0.000, -0.032]
- Rotation: in RPY (degree) [0.000, 0.000, -1.839]
- Matrix:
  0.999  0.032  0.000  1.256
 -0.032  0.999 -0.000  0.216
 -0.000  0.000  1.000  0.000
  0.000  0.000  0.000  1.000
```