

ROKEY BOOT CAMP

ROS-2 심화학습 강의자료 – tf

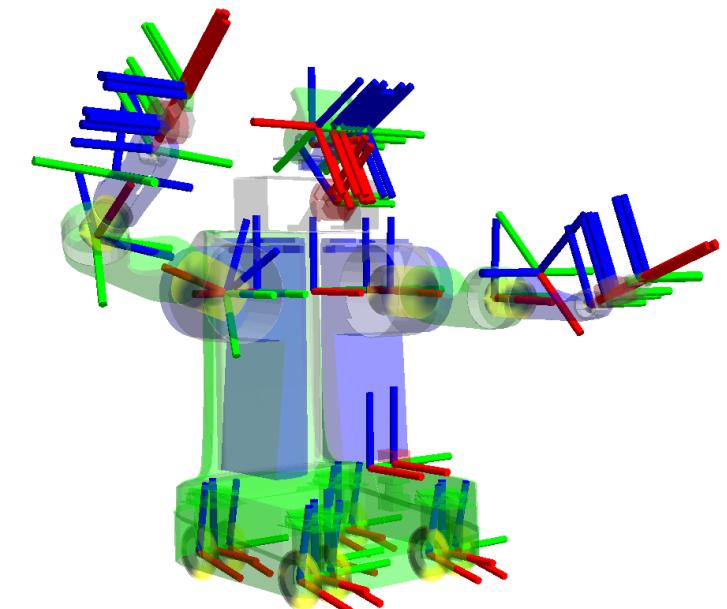
Contents

- Introducing tf2
- Writing a static broadcaster
- Writing a broadcaster

- Adding a fixed frame broadcaster
- Adding a dynamic frame broadcaster

tf2

- tf2: ROS2에서 여러 좌표 프레임 간의 변환과 관계를 추적하고 관리하는 라이브러리
 - 로봇 시스템은 일반적으로 세계 프레임, 기본 프레임, 그리퍼 프레임, 헤드 프레임 등과 같이 시간에 따라 변경되는 많은 3D 좌표 프레임을 가짐
 - tf2는 시간에 따른 이러한 모든 프레임을 추적하고 다음과 같은 질문을 할 수 있도록 함
 - Q1. 5초 전 world frame 대비 head frame은 어디에 있는가?
 - Q2. Gripper에 있는 물체의 포즈는 내 base에 비해 어떤가?
 - Q3. Map frame에서 base frame의 현재 포즈는 어떤가?



Turtlesim 예제

- tf2의 broadcaster와 listener를 통해 로봇의 좌표계가 어떻게 생성되고 활용되는지 이해하기
- view_frames 및 tf2_echo 도구를 사용하여 프레임 간의 관계를 시각적으로 분석하는 방법과, rviz2를 사용하여 프레임을 시각화하기

✓ Turtlesim 예제

1. 데모 패키지와 종속 파일들 설치



```
sudo apt-get install ros-humble-rviz2 ros-humble-turtle-tf2-py  
ros-humble-tf2-ros ros-humble-tf2-tools ros-humble-turtlesim
```

✓ Turtlesim 예제

2. 터미널에서 다음 명령어 실행



```
ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```



✓ Turtlesim 예제

3. 두번째 터미널을 열어 teleopkey를 실행시켜 거북이를 움직이면 거북이가 따라오는 것을 관찰 가능



```
ros2 run turtlesim turtle_teleop_key
```



✓ Turtlesim 예제

1. 이 데모는 tf2 라이브러리를 사용하여 세 개의 좌표 프레임(world frame, turtle1 frame, turtle2 frame)을 만듦
2. 이 튜토리얼은 tf2 브로드캐스터를 사용하여 거북이 좌표 프레임을 게시하고 tf2 리스너를 사용하여 거북이 프레임의 차이를 계산하고 한 거북이를 움직여 다른 거북이를 따라가게 함
3. 지금부터 3가지 방법(view_frame, tf2_echo, rviz)를 통해 이 데모를 만드는데 tf2가 어떻게 사용되었는지를 확인할 것임

✓ Turtlesim 예제 – view_frames

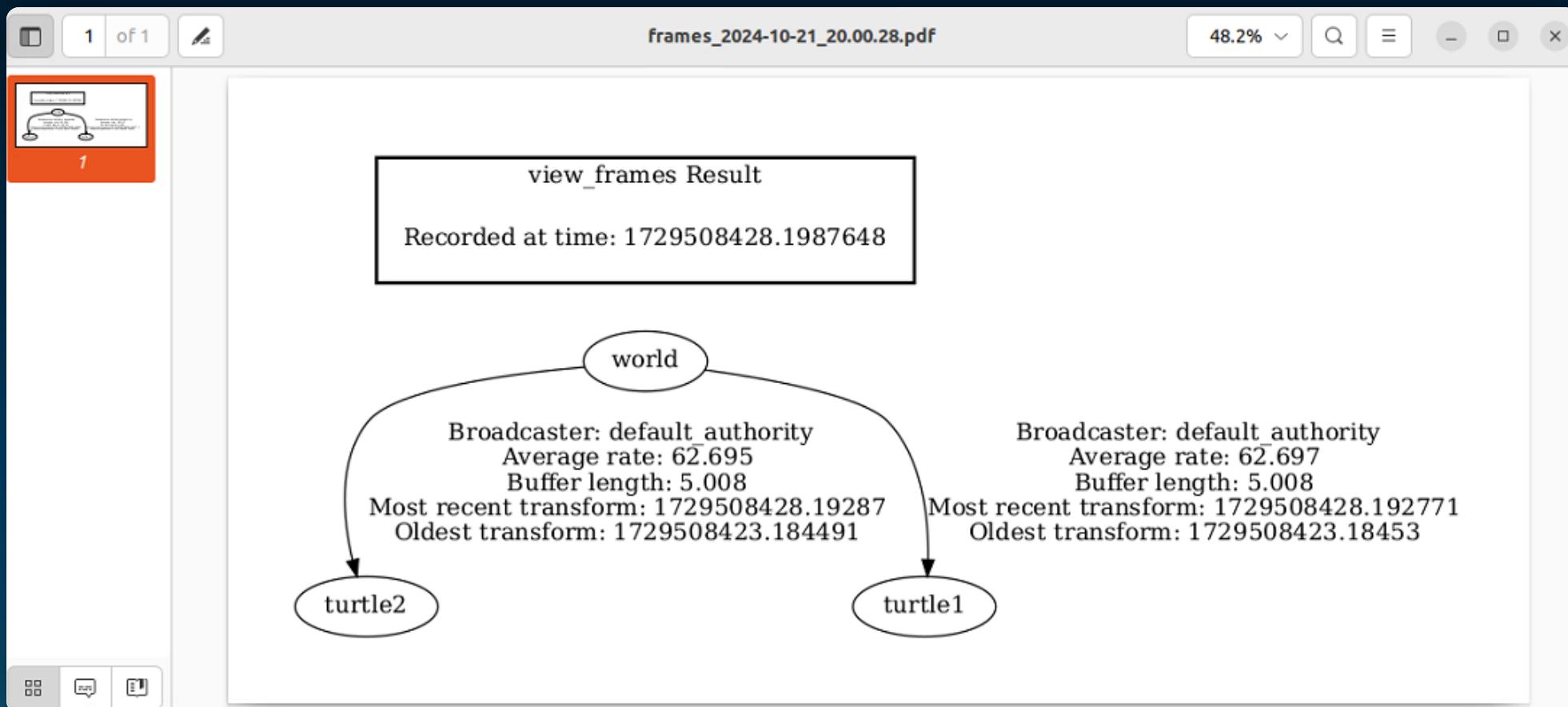
1. view_frames는 ROS를 통해 tf2가 브로드캐스트하는 프레임의 다이어그램을 만듦
2. 다음 명령어를 이용하여 tf2가 브로드캐스트하는 프레임의 다이어그램을 제작



```
ros2 run tf2_tools view_frames
```

✓ Turtlesim 예제 - view_frames

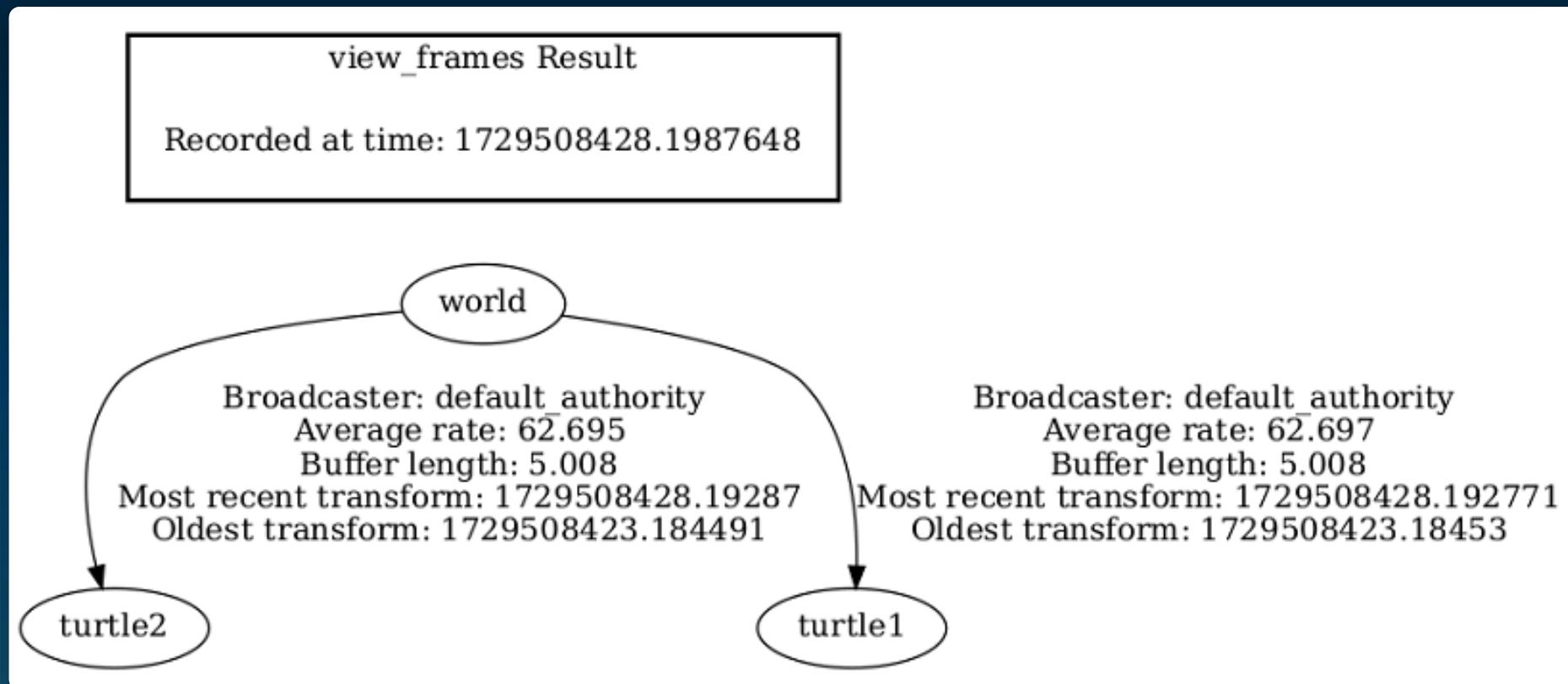
3. 생성된 "frames_2024-10-*****.pdf" 파일 열기



✓ Turtlesim 예제- view_frames

4. 해당 파일을 토해 tf2에서 브로드캐스트하는 세 개의 프레임을 확인 가능

- World frame: turtle1과 turtle2 frame의 부모 frame
- view_frames는 가장 오래되고 가장 최근의 프레임 변환이 수신된 시기와 디버깅 목적으로 tf2 프레임이 tf2에 게시되는 속도에 대한 진단 정보를 보고함



✓ Turtlesim 예제- tf2_echo

1. tf2_echo는 ROS를 통해 브로드캐스트된 두 프레임 간의 변환을 보고함
2. 다음 명령어를 이용하여 tf2_echo 실행



```
ros2 run tf2_ros tf2_echo turtle2 turtle1
```

✓ Turtlesim 예제- tf2_echo

3. 다음과 같은 출력을 통해 tf2_echo 리스너가 ROS2를 통해 broadcasting 된 프레임을 수신할 때 transform이 표시되는 것을 관찰 가능



```
At time 1729511402.418714435
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.482, 0.876]
- Rotation: in RPY (radian) [0.000, 0.000, -1.005]
- Rotation: in RPY (degree) [0.000, 0.000, -57.577]
- Matrix:
  0.536  0.844  0.000  0.000
 -0.844  0.536 -0.000  0.000
 -0.000  0.000  1.000  0.000
  0.000  0.000  0.000  1.000
```

✓ Turtlesim 예제- rviz

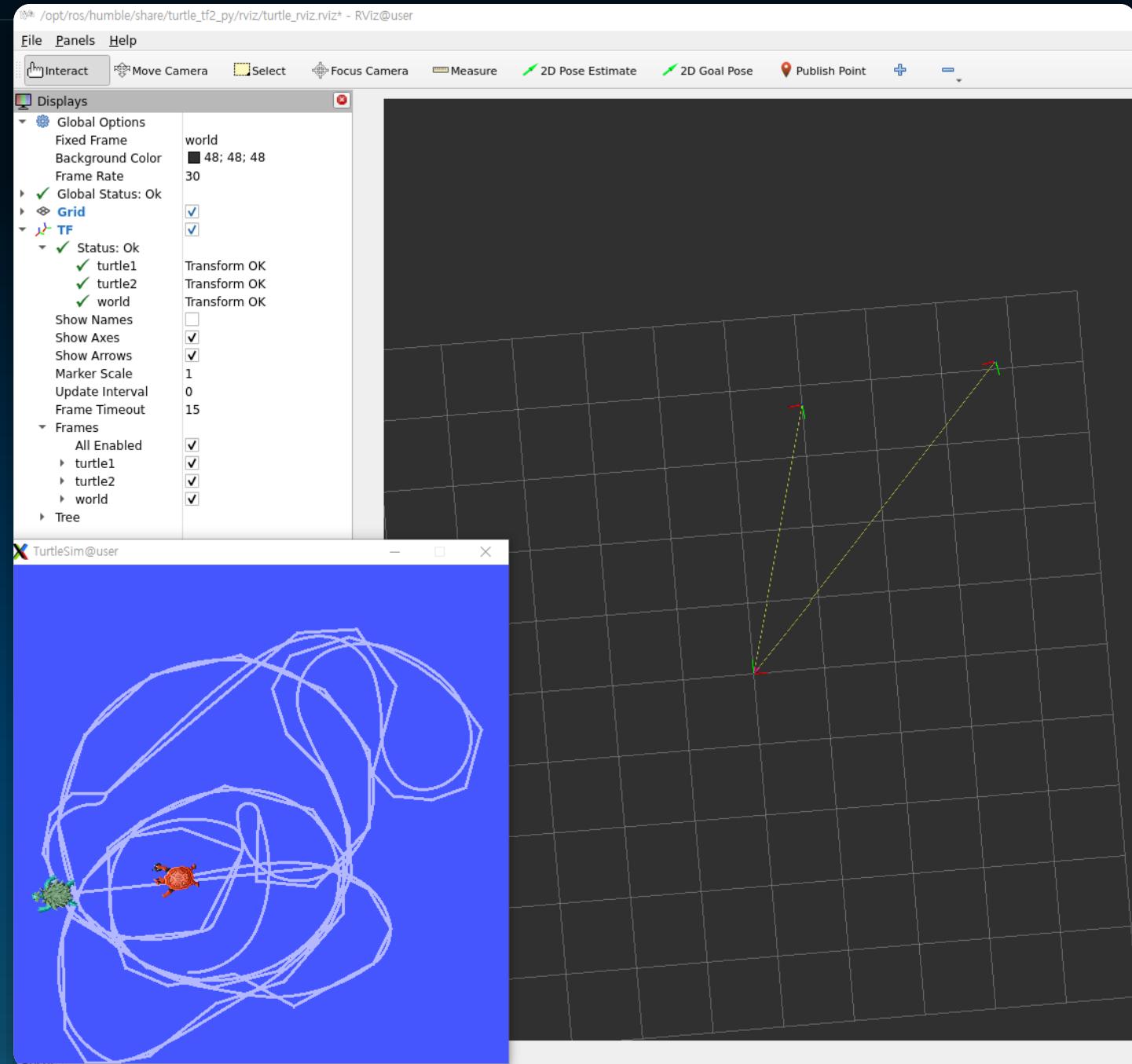
1. Rviz2 역시 tf2 프레임을 검사하는 데 유용한 시각화 도구로 사용 가능
 - -d 옵션을 사용하여 구성 파일로 시작하여 rviz2를 사용하여 거북이 프레임을 관찰 가능
2. 다음 명령어를 통해 rviz2 실행



```
ros2 run rviz2 rviz2 -d $(ros2 pkg prefix --share turtle_tf2_py)/rviz/turtle_rviz.rviz
```

✓ Turtlesim 예제- rviz

3. 사이드 바를 통해 tf2에서 broadcast 된 frame을 관찰 가능하며, 거북이를 움직이면 rviz에서도 frame이 움직이는 것을 관찰 가능



Writing a static broadcaster

- 파이썬으로 정적 브로드캐스터를 작성하기

✓ Writing a static broadcaster

1. 정적 변환(static transforms)은 좌표계 간의 변환이 시간에 따라 변하지 않을 때 유용하게 사용됨
ex) 센서가 로봇에 고정되어 있거나, 두 로봇 부품 간의 상대적 위치가 고정된 경우
2. 이번 섹션은 정적 변환의 기본을 다루는 독립형 튜토리얼로, 두 부분으로 구성됨
 1. 정적 변환을 tf2에 게시하는 코드 작성
 2. tf2_ros에서 명령줄 static_transform_publisher 실행 가능 도구(executable tool)를 사용하는 방법을 설명

✓ Writing a static broadcaster

- 터미널에서 다음 명령어를 이용하여 패키지 생성



```
ros2 pkg create --build-type ament_python --license Apache-2.0 -- learning_tf2_py
```

- src/learning_tf2_py/learning_tf2_py 디렉토리에 static_turtle_tf2_broadcaster.py 파일 생성

✓ Writing a static broadcaster

- 해당 파일에 다음과 같이 코드 작성

```
import math
import sys

from geometry_msgs.msg import TransformStamped

import numpy as np

import rclpy
from rclpy.node import Node

from tf2_ros.static_transform_broadcaster import StaticTransformBroadcaster
```

(이어서 작성)

✓ Writing a static broadcaster

- 해당 파일에 다음과 같이 코드 작성

```
● ● ●  
def quaternion_from_euler(ai, aj, ak):  
    ai /= 2.0  
    aj /= 2.0  
    ak /= 2.0  
    ci = math.cos(ai)  
    si = math.sin(ai)  
    cj = math.cos(aj)  
    sj = math.sin(aj)  
    ck = math.cos(ak)  
    sk = math.sin(ak)  
    cc = ci*ck  
    cs = ci*sk  
    sc = si*ck  
    ss = si*sk  
  
    q = np.empty((4, ))  
    q[0] = cj*sc - sj*cs  
    q[1] = cj*ss + sj*cc  
    q[2] = cj*cs - sj*sc  
    q[3] = cj*cc + sj*ss  
  
    return q
```

✓ Writing a static broadcaster

- 해당 파일에 다음과 같이 코드 작성

```
class StaticFramePublisher(Node):
    """
    Broadcast transforms that never change.

    This example publishes transforms from `world` to a static turtle frame.
    The transforms are only published once at startup, and are constant for all
    time.
    """

    def __init__(self, transformation):
        super().__init__('static_turtle_tf2_broadcaster')

        self.tf_static_broadcaster = StaticTransformBroadcaster(self)

        # Publish static transforms once at startup
        self.make_transforms(transformation)

    def make_transforms(self, transformation):
        t = TransformStamped()

        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'world'
        t.child_frame_id = transformation[1]

        t.transform.translation.x = float(transformation[2])
        t.transform.translation.y = float(transformation[3])
        t.transform.translation.z = float(transformation[4])
        quat = quaternion_from_euler(
            float(transformation[5]), float(transformation[6]),
            float(transformation[7]))
        t.transform.rotation.x = quat[0]
        t.transform.rotation.y = quat[1]
        t.transform.rotation.z = quat[2]
        t.transform.rotation.w = quat[3]

        self.tf_static_broadcaster.sendTransform(t)
```

(이어서 작성)

✓ Writing a static broadcaster

- 해당 파일에 다음과 같이 코드 작성

```
● ● ●

def main():
    logger = rclpy.logging.get_logger('logger')

    # obtain parameters from command line arguments
    if len(sys.argv) != 8:
        logger.info('Invalid number of parameters. Usage: \n'
                    '$ ros2 run turtle_tf2_py static_turtle_tf2_broadcaster'
                    '\nchild_frame_name x y z roll pitch yaw')
        sys.exit(1)

    if sys.argv[1] == 'world':
        logger.info('Your static turtle name cannot be "world"')
        sys.exit(2)

    # pass parameters and initialize node
    rclpy.init()
    node = StaticFramePublisher(sys.argv)
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Writing a static broadcaster – 코드 설명

▪ 필요한 라이브러리 가져오기

- **TransformStamped**

변환 트리에 게시할 메시지에
대한 템플릿을 제공

- **rclpy**

Node 클래스 사용을 지원

- **tf2_ros**

StaticTransformBroadcaster를
제공하며, 정적 변환을 쉽게 게
시할 수 있도록 지원함



```
import math
import sys

from geometry_msgs.msg import TransformStamped

import numpy as np

import rclpy
from rclpy.node import Node

from tf2_ros.static_transform_broadcaster
import StaticTransformBroadcaster
```

- **quaternion_from_euler** 함수:

오일러 각도(roll, pitch, yaw)를 쿼터니언으로 변환

- 일반적으로 로봇이나 시뮬레이션에서는 짐벌락 문제와 계산의 효율성 때문에 오일러 각도 대신 쿼터니언을 사용함
- 공식은 다음과 같음

$$\begin{aligned} \mathbf{q}_{IB} &= \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\phi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \sin(\phi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\phi/2) \cos(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \cos(\theta/2) \sin(\psi/2) - \sin(\phi/2) \sin(\theta/2) \cos(\psi/2) \end{bmatrix} \end{aligned}$$



```
def quaternion_from_euler(ai, aj, ak):
    ai = 2.0
    aj = 2.0
    ak = 2.0
    ci = math.cos(ai)
    si = math.sin(ai)
    cj = math.cos(aj)
    sj = math.sin(aj)
    ck = math.cos(ak)
    sk = math.sin(ak)
    cc = ci*ck
    cs = ci*sk
    sc = si*ck
    ss = si*sk

    q = np.empty((4, ))
    q[0] = cj*sc - sj*cs
    q[1] = cj*ss + sj*cc
    q[2] = cj*cs - sj*sc
    q[3] = cj*cc + sj*ss

    return q
```

■ StaticFramePublisher 클래스 생성자

- static_turtle_tf2_broadcaster라는 이름으로 노드를 초기화
- StaticTransformBroadcaster가 생성되어 시작 시 정적 변환을 하나 보냄



```
class StaticFramePublisher(Node):
    def __init__(self, transformation):
        super().__init__('static_turtle_tf2_broadcaster')

        self.tf_static_broadcaster = StaticTransformBroadcaster(self)

        # Publish static transforms once at startup
        self.make_transforms(transformation)
```

- `make_transforms` 함수: 전달된 변환 정보(프레임 이름, 위치, 오일러 각도)를 기반으로 `TransformStamped` 메시지를 생성하고, 이를 정적 브로드캐스터를 통해 전송
 - `self.get_clock().now()` 게시되는 변환에 타임스탬프를 제공해야 하며 현재 시간으로 스탬프를 찍음
 - 만들어지고 있는 링크의 부모와 자식 프레임의 이름을 설정
 - 6D 포즈(이동 및 회전)값을 채움
 - 마지막으로 `sendTransform()` 함수를 사용하여 정적 변환을 브로드캐스트

```
● ● ●

def make_transforms(self, transformation):
    t = TransformStamped()

    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'world'
    t.child_frame_id = transformation[1]

    ...

    t.transform.translation.x = float(transformation[2])
    t.transform.translation.y = float(transformation[3])
    t.transform.translation.z = float(transformation[4])
    quat = quaternion_from_euler(
        float(transformation[5]), float(transformation[6]), float(transformation[7]))
    t.transform.rotation.x = quat[0]
    t.transform.rotation.y = quat[1]
    t.transform.rotation.z = quat[2]
    t.transform.rotation.w = quat[3]

    self.tf_static_broadcaster.sendTransform(t)
```

- **main** 함수: ROS2 시스템을 초기화하고 **StaticFramePublisher** 노드 실행

- `sys.argv`를 통해 인자(프레임 이름, 위치, 오일러 각도)를 가져옴
- `rclpy.init()`으로 ROS2를 초기화하고, `StaticFramePublisher` 객체를 생성하여 인자로 받은 변환 정보 전달
- `rclpy.spin(node)`를 통해 노드를 계속 실행하며 변환 유지

```
def main():
    logger = rclpy.logging.get_logger('logger')

    # obtain parameters from command line arguments
    if len(sys.argv) != 8:
        logger.info('Invalid number of parameters. Usage: \n'
                    '$ ros2 run turtle_tf2_py static_turtle_tf2_broadcaster'
                    'child_frame_name x y z roll pitch yaw')
        sys.exit(1)

    if sys.argv[1] == 'world':
        logger.info('Your static turtle name cannot be "world"')
        sys.exit(2)

    # pass parameters and initialize node
    rclpy.init()
    node = StaticFramePublisher(sys.argv)
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Writing a static broadcaster – Update package.xml

1. src/learning_tf2_py 디렉토리에 있는 package.xml 파일에 <description>, <maintainer>, <license> 태그 채우기



```
<description>Learning tf2 with rclpy</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

✓ Writing a static broadcaster – Update package.xml

2. 다음과 같이 종속성 선언하기



```
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>python3-numpy</exec_depend>
<exec_depend>rclpy</exec_depend>
<exec_depend>tf2_ros_py</exec_depend>
<exec_depend>turtlesim</exec_depend>
```

- **geometry_msgs** 로봇의 위치, 속도, 변환 등의 기하학적 메시지 타입을 정의하는 ROS 패키지
- **python3-numpy** 파이썬에서 수학적 계산과 배열 연산을 처리하는 라이브러리
- **rclpy** ROS2의 파이썬 클라이언트 라이브러리
- **tf2_ros_py** ROS2에서 프레임 간의 변환을 관리하고 브로드캐스팅하기 위한 라이브러리
- **turtlesim** ROS2에서 기본적인 시뮬레이션 환경을 제공하는 패키지

✓ Writing a static broadcaster – setup.py

- ros2 run 명령으로 노드를 실행하기 위하여 `setup.py`(`src/learning_tf2_py` 디렉터리에 있음)에 `entry_points`를 추가



```
entry_points={  
    'console_scripts': [  
        'static_turtle_tf2_broadcaster = learning_tf2_py.static_turtle_tf2_broadcaster:main'  
    ],  
},
```

✓ Writing a static broadcaster – Build

1. 빌드 전 누락된 종속성 확인하기



```
rosdep install -i --from-path src --rosdistro humble -y
```

2. 작업 공간에서 새 패키지를 빌드



```
colcon build --packages-select learning_tf2_py
```

3. 작업 공간의 루트로 이동한 후 설정 파일 가져오기



```
. install/setup.bash
```

✓ Writing a static broadcaster – Run

- **static_turtle_tf2_broadcaster 노드 실행**
 - my_turtle이 지상 1m 높이에 떠있도록 설정함

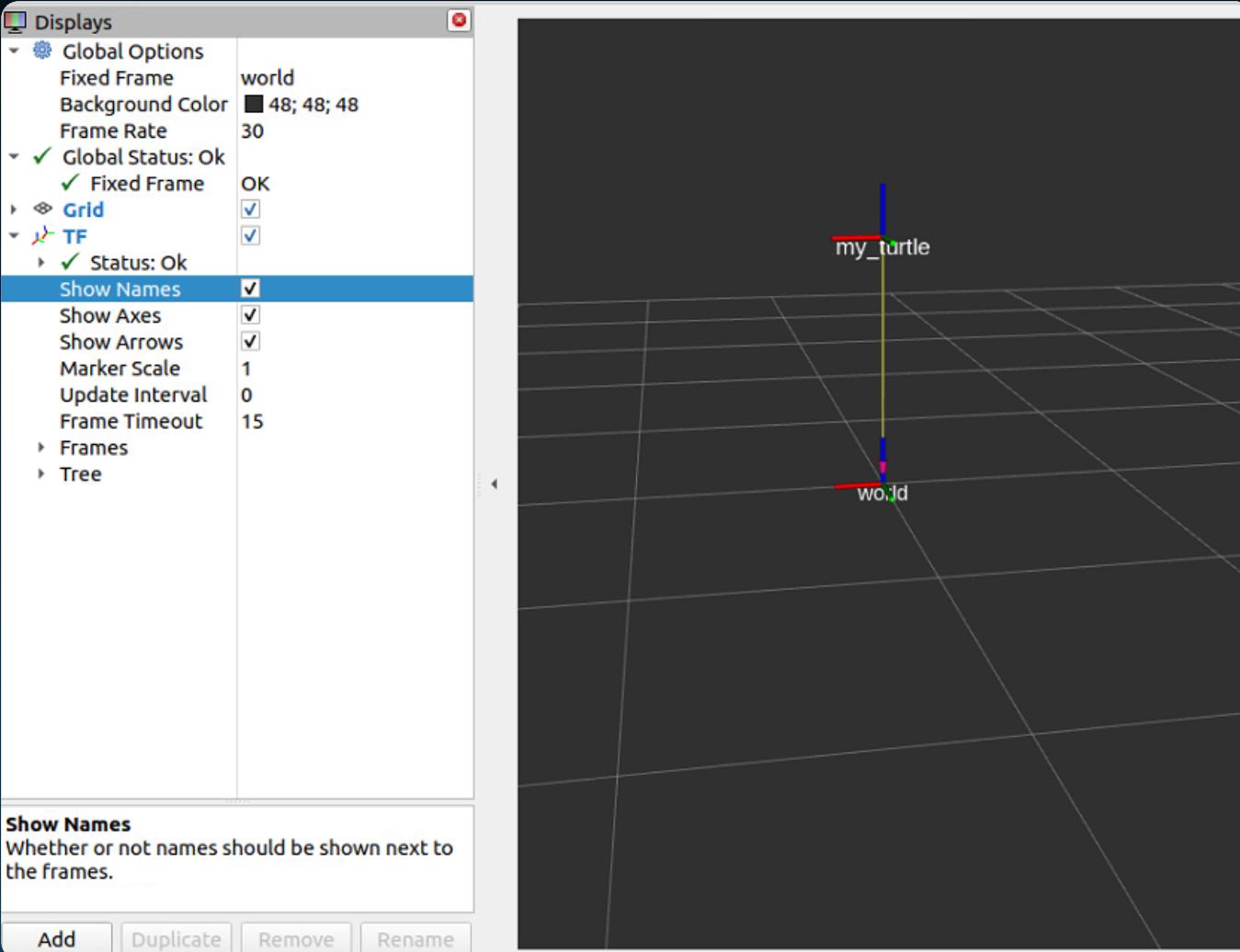


```
ros2 run learning_tf2_py static_turtle_tf2_broadcaster my_turtle 0 0 1 0 0 0
```

✓ Writing a static broadcaster – Run

▪ static_turtle_tf2_broadcaster 노드 실행

- rviz를 열어 Fixed Frame을 world로 바꾼 후 TF를 추가한후 Show Names를 체크하면 아래와 같이 my_turtle이 1m 높이에 떠있는 것을 관찰 가능



✓ Writing a static broadcaster – Run

▪ static_turtle_tf2_broadcaster 노드 실행

- tf_static 토픽을 에코하여 정적 변환이 게시되었는지 확인 가능



```
ros2 topic echo /tf_static
```

```
transforms:  
- header:  
  stamp:  
    sec: 1729566292  
    nanosec: 603306826  
    frame_id: world  
    child_frame_id: my_turtle  
  transform:  
    translation:  
      x: 0.0  
      y: 0.0  
      z: 1.0  
    rotation:  
      x: 0.0  
      y: 0.0  
      z: 0.0  
      w: 1.0  
---
```

✓ Writing a static broadcaster

1. 본 섹션에서는 **StaticTransformBroadcaster**를 사용하여 정적 변환을 게시하는 방법을 제시하는 것을 목표로 함
2. 다음 명령은 미터 단위의 x/y/z 오프셋과 **라디안 단위**의 roll/pitch/yaw를 사용하여 tf2에 정적 좌표 변환을 게시함 (x, y, z, yaw, pitch, roll에 적절한 상수값을 배치할 것)



```
ros2 run tf2_ros static_transform_publisher --x x --y y --z z --yaw yaw --pitch pitch  
--roll roll --frame-id frame_id --child-frame-id child_frame_id
```

3. 다음 명령은 미터 단위의 x/y/z 오프셋과 **쿼터니언으로** roll/pitch/yaw를 사용하여 tf2에 정적 좌표 변환을 게시함



```
ros2 run tf2_ros static_transform_publisher --x x --y y --z z --qx qx --qy qy --qz qz  
--qw qw --frame-id frame_id --child-frame-id child_frame_id
```

✓ Writing a static broadcaster

1. static_transform_publisher는 수동 사용을 위한 명령줄 도구로 설계되었으며, 정적 변환을 설정하기 위한 실행 파일 내에서 사용하도록 설계됨
2. 예를 들어 다음과 같이 사용 가능



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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='tf2_ros',
            executable='static_transform_publisher',
            arguments = ['--x', '0', '--y', '0', '--z', '1', '--yaw', '0', '--pitch', '0', '--roll', '0', '--frame-id', 'world', '--child-frame-id', 'mystaticturtle']
        ),
    ])
```

Writing a broadcaster

- 로봇의 상태를 tf2에 broadcast하는 방법

✓ Writing a broadcaster (1)

1. src/learning_tf2_py/learning_tf2_py에 turtle_tf2_broadcaster.py파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●  
import math  
  
from geometry_msgs.msg import TransformStamped  
  
import numpy as np  
  
import rclpy  
from rclpy.node import Node  
  
from tf2_ros import TransformBroadcaster  
  
from turtlesim.msg import Pose
```

```
● ● ●  
def quaternion_from_euler(ai, aj, ak):  
    ai /= 2.0  
    aj /= 2.0  
    ak /= 2.0  
    ci = math.cos(ai)  
    si = math.sin(ai)  
    cj = math.cos(aj)  
    sj = math.sin(aj)  
    ck = math.cos(ak)  
    sk = math.sin(ak)  
    cc = ci*ck  
    cs = ci*sk  
    sc = si*ck  
    ss = si*sk  
  
    q = np.empty((4, ))  
    q[0] = cj*sc - sj*cs  
    q[1] = cj*ss + sj*cc  
    q[2] = cj*cs - sj*sc  
    q[3] = cj*cc + sj*ss  
  
    return q
```

✓ Writing a broadcaster (2)

1. src/learning_tf2_py/learning_tf2_py에 turtle_tf2_broadcaster.py 파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성



```

class FramePublisher(Node):
    def __init__(self):
        super().__init__('turtle_tf2_frame_publisher')

        # Declare and acquire `turtlename` parameter
        self.turtlename = self.declare_parameter(
            'turtlename', 'turtle').get_parameter_value().string_value

        # Initialize the transform broadcaster
        self.tf_broadcaster = TransformBroadcaster(self)

        # Subscribe to a turtle{1}{2}/pose topic and call handle_turtle_pose
        # callback function on each message
        self.subscription = self.create_subscription(
            Pose,
            f'/{self.turtlename}/pose',
            self.handle_turtle_pose,
            1)
        self.subscription # prevent unused variable warning
...

```



```

def handle_turtle_pose(self, msg):
    t = TransformStamped()

    # Read message content and assign it to
    # corresponding tf variables
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'world'
    t.child_frame_id = self.turtlename

    # Turtle only exists in 2D, thus we get x and y translation
    # coordinates from the message and set the z coordinate to 0
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0

    # For the same reason, turtle can only rotate around one axis
    # and this why we set rotation in x and y to 0 and obtain
    # rotation in z axis from the message
    q = quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    # Send the transformation
    self.tf_broadcaster.sendTransform(t)

```

✓ Writing a broadcaster (3)

1. src/learning_tf2_py/learning_tf2_py에 turtle_tf2_broadcaster.py파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●

def main():
    rclpy.init()
    node = FramePublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

rclpy.shutdown()
```

✓ Writing a broadcaster – 코드 설명

■ 필요한 라이브러리 가져오기

- TransformStamped: 변환 트리에 게시할 메시지에 대한 템플릿을 제공
- rclpy: Node 클래스 사용을 지원
- tf2_ros: StaticTransformBroadcaster를 제공하며, 정적 변환을 쉽게 게시할 수 있도록 지원함
- turtlesim.msg.Pose: Turtlesim에서 사용되는 메시지 타입으로, 터틀의 위치(x, y)와 방향(θ)을 나타냄

```
● ● ●

import math

from geometry_msgs.msg import TransformStamped

import numpy as np

import rclpy
from rclpy.node import Node

from tf2_ros import TransformBroadcaster

from turtlesim.msg import Pose
```

✓ Writing a broadcaster – 코드 설명(이전과 동일)

- **quaternion_from_euler** 함수:

오일러 각도(roll, pitch, yaw)를 쿼터니언으로 변환

- 일반적으로 로봇이나 시뮬레이션에서는 짐벌락 문제와 계산의 효율성 때문에 오일러 각도 대신 쿼터니언을 사용함

$$\begin{aligned} \mathbf{q}_{IB} &= \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\phi/2) \cos(\theta/2) \cos(\psi/2) + \sin(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \sin(\phi/2) \cos(\theta/2) \cos(\psi/2) - \cos(\phi/2) \sin(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \sin(\theta/2) \cos(\psi/2) + \sin(\phi/2) \cos(\theta/2) \sin(\psi/2) \\ \cos(\phi/2) \cos(\theta/2) \sin(\psi/2) - \sin(\phi/2) \sin(\theta/2) \cos(\psi/2) \end{bmatrix} \end{aligned}$$



```
def quaternion_from_euler(ai, aj, ak):
    ai = 2.0
    aj = 2.0
    ak = 2.0
    ci = math.cos(ai)
    si = math.sin(ai)
    cj = math.cos(aj)
    sj = math.sin(aj)
    ck = math.cos(ak)
    sk = math.sin(ak)
    cc = ci*ck
    cs = ci*sk
    sc = si*ck
    ss = si*sk

    q = np.empty((4, ))
    q[0] = cj*sc - sj*cs
    q[1] = cj*ss + sj*cc
    q[2] = cj*cs - sj*sc
    q[3] = cj*cc + sj*ss

    return q
```

✓ Writing a broadcaster – 코드 설명



```
class FramePublisher(Node):  
  
    def __init__(self):  
        super().__init__('turtle_tf2_frame_publisher')  
  
        # Declare and acquire `turtlename` parameter  
        self.turtlename = self.declare_parameter(  
            'turtlename', 'turtle').get_parameter_value().string_value  
  
        # Initialize the transform broadcaster  
        self.tf_broadcaster = TransformBroadcaster(self)  
  
        # Subscribe to a turtle{1}{2}/pose topic and call handle_turtle_pose  
        # callback function on each message  
        self.subscription = self.create_subscription(  
            Pose,  
            f'/{self.turtlename}/pose',  
            self.handle_turtle_pose,  
            1)  
        self.subscription # prevent unused variable warning  
  
    ...
```

■ 생성자

1. 노드를 초기화

Node 클래스를 상속받아 ROS2 노드로 동작하도록 함

2. 파라미터를 선언

거북이 이름(예: turtle1 또는 turtle2)을 지정하는 단일 매개변수 turtlename 을 정의

✓ Writing a broadcaster – 코드 설명



```
class FramePublisher(Node):  
  
    def __init__(self):  
        super().__init__('turtle_tf2_frame_publisher')  
  
        # Declare and acquire `turtlename` parameter  
        self.turtlename = self.declare_parameter(  
            'turtlename', 'turtle').get_parameter_value().string_value  
  
        # Initialize the transform broadcaster  
        self.tf_broadcaster = TransformBroadcaster(self)  
  
        # Subscribe to a turtle{1}{2}/pose topic and call handle_turtle_pose  
        # callback function on each message  
        self.subscription = self.create_subscription(  
            Pose,  
            f'/{self.turtlename}/pose',  
            self.handle_turtle_pose,  
            1)  
        self.subscription # prevent unused variable warning  
  
    ...
```

■ 생성자

3. Transform 브로드캐스터 초기화

TransformBroadcaster 객체를 초기화하여 터틀의 위치와 회전 정보를 브로드캐스트할 준비를 진행

4. 토픽 구독

/turtle1/pose 또는 /turtle2/pose 토픽을 구독

✓ Writing a broadcaster – 코드 설명

```
def handle_turtle_pose(self, msg):
    t = TransformStamped()

    # Read message content and assign it to
    # corresponding tf variables
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'world'
    t.child_frame_id = self.turtlename

    # Turtle only exists in 2D, thus we get x and y translation
    # coordinates from the message and set the z coordinate to 0
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0

    # For the same reason, turtle can only rotate around one axis
    # and this why we set rotation in x and y to 0 and obtain
    # rotation in z axis from the message
    q = quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    # Send the transformation
    self.tf_broadcaster.sendTransform(t)
```

- **Handle_turtle_pose 함수:**
터틀의 포즈(Pose) 메시지를 받아
변환 정보를 브로드캐스트

1. TransformStamped 생성

TransformStamped 객체 t를 생성
하여 메시지 데이터를 담음

2. 시간 정보 설정

t.header.stamp에 현재 시간을 기
록

✓ Writing a broadcaster – 코드 설명

```
def handle_turtle_pose(self, msg):
    t = TransformStamped()

    # Read message content and assign it to
    # corresponding tf variables
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'world'
    t.child_frame_id = self.turtlename

    # Turtle only exists in 2D, thus we get x and y translation
    # coordinates from the message and set the z coordinate to 0
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0

    # For the same reason, turtle can only rotate around one axis
    # and this why we set rotation in x and y to 0 and obtain
    # rotation in z axis from the message
    q = quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    # Send the transformation
    self.tf_broadcaster.sendTransform(t)
```

- **Handle_turtle_pose 함수:**
터틀의 포즈(Pose) 메시지를 받아 변환 정보를 브로드캐스트

3. 프레임 설정

t.header.frame_id는 'world', 자식 프레임(t.child_frame_id)은 터틀 이름(self.turtlename)으로 설정됨

4. 위치 설정

터틀의 위치(msg.x, msg.y)를 변환 객체의 translation.x, translation.y에 설정하고, 2D에서만 동작하므로 translation.z는 0으로 설정함

✓ Writing a broadcaster – 코드 설명

```
def handle_turtle_pose(self, msg):
    t = TransformStamped()

    # Read message content and assign it to
    # corresponding tf variables
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'world'
    t.child_frame_id = self.turtlename

    # Turtle only exists in 2D, thus we get x and y translation
    # coordinates from the message and set the z coordinate to 0
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0

    # For the same reason, turtle can only rotate around one axis
    # and this why we set rotation in x and y to 0 and obtain
    # rotation in z axis from the message
    q = quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    # Send the transformation
    self.tf_broadcaster.sendTransform(t)
```

- **Handle_turtle_pose 함수:**
터틀의 포즈(Pose) 메시지를 받아 변환 정보를 브로드캐스트

5. 회전 설정

터틀은 z축을 기준으로만 회전하므로, quaternion_from_euler 함수에 msg.theta를 넣어 쿼터니언을 생성하고, 이를 변환 객체의 회전 값에 설정함

6. 변환 브로드캐스트

self.tf_broadcaster.sendTransform(t)
를 통해 변환을 브로드캐스트

✓ Writing a broadcaster – 코드 설명

▪ main 함수: ROS2 시스템을 초기화하고, FramePublisher 노드를 실행

1. **ROS2 초기화** rclpy.init()으로 ROS2 시스템을 초기화
2. **노드 실행** FramePublisher 노드를 생성하고, rclpy.spin(node)로 노드가 중단될 때 까지 계속 실행되도록 만듦

```
● ● ●  
def main():  
    rclpy.init()  
    node = FramePublisher()  
    try:  
        rclpy.spin(node)  
    except KeyboardInterrupt:  
        pass  
  
rclpy.shutdown()
```

✓ Writing a broadcaster – launch file

1. src/learning_tf2_py에 launch폴더를 생성 후 해당 폴더에 turtle_tf2_demo.launch.py라는 Launch 파일 생성
2. 생성된 파일에 다음과 같이 코드 작성

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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='learning_tf2_py',
            executable='turtle_tf2_broadcaster',
            name='broadcaster1',
            parameters=[
                {'turtlename': 'turtle1'}
            ]
        ),
    ])
```

✓ Writing a broadcaster – launch file

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

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='learning_tf2_py',
            executable='turtle_tf2_broadcaster',
            name='broadcaster1',
            parameters=[
                {'turtlename': 'turtle1'}
            ]
        ),
    ])
```

- **launch와 launch_ros 패키지에서 필요한 모듈 가져오기**
 - launch실행할 여러 노드를 목록으로 정의할 수 있음
 - launch_ros: ROS2에서 노드를 실행하기 위한 Node 클래스를 불러옴
- **generate_launch_description() 함수: 런치 파일을 생성하는 함수로, LaunchDescription 객체를 반환하며, 여기에 정의된 노드들이 동시에 실행됨**
 - Node1: turtlesim 패키지를 사용하여, turtlesim_node를 실행하는 sim이라는 노드 생성
 - Node2: TF2(좌표 변환) 관련 학습을 위한 learning_tf2_py 패키지를 사용하여, turtle_tf2_broadcaster를 실행하는 broadcaster1노드 생성

✓ Writing a broadcaster – Update package.xml

- [src/learning_tf2_py](#) 디렉토리로에 있는 package.xml 파일에 다음과 같이 종속성 선언하기



```
<exec_depend>launch</exec_depend>
<exec_depend>launch_ros</exec_depend>
```

- launch: ROS2에서 노드를 실행하고 관리하기 위한 런치 파일을 사용할 때 필요한 종속성
- launch_ros: ROS2에서 노드를 런치 파일을 통해 실행할 때 ROS2에서 제공하는 고유한 기능(노드 관리, 토픽과 서비스 통신 등)을 지원하는 패키지

✓ Writing a broadcaster – setup.py

1. ros2 run 명령으로 노드를 실행하기 위하여 setup.py(src/learning_tf2_py 디렉터리에 있음)에 entry_points를 추가

```
● ● ●  
entry_points={  
    "console_scripts": [  
        'static_turtle_tf2_broadcaster = learning_tf2_py.static_turtle_tf2_broadcaster:main',  
        'turtle_tf2_broadcaster = learning_tf2_py.turtle_tf2_broadcaster:main',  
    ],  
},
```

✓ Writing a broadcaster – setup.py

2. launch폴더의 실행 파일이 설치되도록 data_files를 아래와 같이 수정

```
● ● ●  
data_files=[  
    ("share/ament_index/resource_index/packages", ["resource/" + package_name]),  
    ("share/" + package_name, ["package.xml"]),  
    (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*'))),  
],
```

3. 파일 최상단에 다음 코드를 추가하여 필요한 라이브러리 가져오기

```
● ● ●  
import os  
from glob import glob
```

✓ Writing a broadcaster – Build

1. 빌드 전 누락된 종속성 확인하기



```
rosdep install -i --from-path src --rosdistro humble -y
```

2. 작업 공간에서 새 패키지를 빌드



```
colcon build --packages-select learning_tf2_py
```

3. 작업 공간의 루트로 이동한 후 설정 파일 가져오기



```
. install/setup.bash
```

✓ Writing a broadcaster – Run

1. turtlesim 시뮬레이션 노드와 turtle_tf2_broadcaster 노드를 시작하는 실행 파일을 실행



```
ros2 launch learning_tf2_py turtle_tf2_demo.launch.py
```

2. 두번째 터미널에서 turtle_teleop_key 실행



```
ros2 run turtlesim turtle_teleop_key
```

3. 마지막으로 세번째 터미널에서 tf2_echo 도구를 사용하여 거북이 포즈가 실제로 tf2에 브로드캐스트 되는지 확인



```
ros2 run tf2_ros tf2_echo world turtle1
```

Writing a listener

- tf2를 사용하여 프레임 변환에 접근하기

✓ Writing a listener

1. src/learning_tf2_py/learning_tf2_py에 turtle_tf2_listener.py파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●  
import math  
  
from geometry_msgs.msg import Twist  
  
import rclpy  
from rclpy.node import Node  
  
from tf2_ros import TransformException  
from tf2_ros.buffer import Buffer  
from tf2_ros.transform_listener import TransformListener  
  
from turtlesim.srv import Spawn
```

(이어서 작성)

✓ Writing a listener

(0|어서 작성)

```
class FrameListener(Node):

    def __init__(self):
        super().__init__('turtle_tf2_frame_listener')

        # Declare and acquire `target_frame` parameter
        self.target_frame = self.declare_parameter(
            'target_frame', 'turtle1').get_parameter_value().string_value

        self.tf_buffer = Buffer()
        self.tf_listener = TransformListener(self.tf_buffer, self)

        # Create a client to spawn a turtle
        self.spawner = self.create_client(Spawn, 'spawn')
        # Boolean values to store the information
        # if the service for spawning turtle is available
        self.turtle_spawning_service_ready = False
        # if the turtle was successfully spawned
        self.turtle_spawned = False

        # Create turtle2 velocity publisher
        self.publisher = self.create_publisher(Twist, 'turtle2/cmd_vel', 1)

        # Call on_timer function every second
        self.timer = self.create_timer(1.0, self.on_timer)
```

✓ Writing a listener
(0|어서 작성)

```
def on_timer(self):
    # Store frame names in variables that will be used to
    # compute transformations
    from_frame_rel = self.target_frame
    to_frame_rel = 'turtle2'

    if self.turtle_spawning_service_ready:
        if self.turtle_spawned:
            # Look up for the transformation between target_frame and turtle2 frames
            # and send velocity commands for turtle2 to reach target_frame
            try:
                t = self.tf_buffer.lookup_transform(
                    to_frame_rel,
                    from_frame_rel,
                    rclpy.time.Time())
            except TransformException as ex:
                self.get_logger().info(
                    f'Could not transform {to_frame_rel} to {from_frame_rel}: {ex}')
            return

            msg = Twist()
            scale_rotation_rate = 1.0
            msg.angular.z = scale_rotation_rate * math.atan2(
                t.transform.translation.y,
                t.transform.translation.x)

            scale_forward_speed = 0.5
            msg.linear.x = scale_forward_speed * math.sqrt(
                t.transform.translation.x ** 2 +
                t.transform.translation.y ** 2)

            self.publisher.publish(msg)
```

✓ Writing a listener (0|어서 작성)

```
else:  
    if self.result.done():  
        self.get_logger().info(  
            f'Successfully spawned {self.result.result().name}')  
        self.turtle_spawned = True  
    else:  
        self.get_logger().info('Spawn is not finished')  
else:  
    if self.spawner.service_is_ready():  
        # Initialize request with turtle name and coordinates  
        # Note that x, y and theta are defined as floats in turtlesim/srv/Spawn  
        request = Spawn.Request()  
        request.name = 'turtle2'  
        request.x = float(4)  
        request.y = float(2)  
        request.theta = float(0)  
        # Call request  
        self.result = self.spawner.call_async(request)  
        self.turtle_spawning_service_ready = True  
    else:  
        # Check if the service is ready  
        self.get_logger().info('Service is not ready')
```

✓ Writing a listener

(0|어서 작성)

```
def main():
    rclpy.init()
    node = FrameListener()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Writing a listener – 코드 설명

```
import math

from geometry_msgs.msg import Twist

import rclpy
from rclpy.node import Node

from tf2_ros import TransformException
from tf2_ros.buffer import Buffer
from tf2_ros.transform_listener import TransformListener

from turtlesim.srv import Spawn
```

1. 필요한 라이브러리 가져오기

- **geometry_msgs.msg.Twist**: 터틀봇의 속도 명령을 나타내는 메시지 형식
- **rclpy**: ROS2 파이썬 클라이언트를 사용하기 위해 불러오는 기본 패키지
- **rclpy.node.Node**: 모든 ROS2 노드가 상속받아야 하는 기본 클래스
- **tf2_ros.TransformException**: 변환 실패 시 발생하는 예외를 처리하는 클래스
- **tf2_ros.Buffer**: 변환 데이터를 저장하는 버퍼
- **tf2_ros.TransformListener**: 변환 정보를 실시간으로 수신하는 리스너
- **turtlesim.srv.Spawn**: 터틀심 시뮬레이션에서 새 거북이를 생성하는 서비스를 위한 메시지 형식

✓ Writing a listener – 코드 설명

▪ 생성자

```
def __init__(self):
    super().__init__('turtle_tf2_frame_listener')

    self.target_frame = self.declare_parameter(
        'target_frame', 'turtle1').get_parameter_value().string_value

    self.tf_buffer = Buffer()
    self.tf_listener = TransformListener(self.tf_buffer, self)

    self.spawner = self.create_client(Spawn, 'spawn')
    self.turtle_spawning_service_ready = False
    self.turtle_spawned = False

    self.publisher = self.create_publisher(Twist, 'turtle2/cmd_vel', 1)
    self.timer = self.create_timer(1.0, self.on_timer)
```

1. Node 상속

FrameListener 클래스는 ROS2 노드로 동작하며, 'turtle_tf2_frame_listener'라는 이름으로 초기화

2. target_frame 파라미터

target_frame이라는 파라미터를 선언하고 기본값으로 'turtle1'을 설정합니다. 이 파라미터는 이동할 목표 프레임(거북이)을 지정

✓ Writing a listener – 코드 설명

▪ 생성자

```
def __init__(self):
    super().__init__('turtle_tf2_frame_listener')

    self.target_frame = self.declare_parameter(
        'target_frame', 'turtle1').get_parameter_value().string_value

    self.tf_buffer = Buffer()
    self.tf_listener = TransformListener(self.tf_buffer, self)

    self.spawner = self.create_client(Spawn, 'spawn')
    self.turtle_spawning_service_ready = False
    self.turtle_spawned = False

    self.publisher = self.create_publisher(Twist, 'turtle2/cmd_vel', 1)
    self.timer = self.create_timer(1.0, self.on_timer)
```

3. Buffer와 TransformListener

tf_buffer는 변환 데이터를 저장하고, TransformListener는 변환 정보를 지속적으로 수신하며, turtle1과 turtle2 사이의 위치 변환 처리

4. spawner

'spawn' 서비스를 호출해 새로운 거북이('turtle2')를 생성하는 클라이언트를 만들어 터틀이 생성되었는지 성공 여부를 판단 후 변수 (turtle_spawning_service_ready, turtle_spawned)를 업데이트

✓ Writing a listener – 코드 설명

■ 생성자

```
def __init__(self):
    super().__init__('turtle_tf2_frame_listener')

    self.target_frame = self.declare_parameter(
        'target_frame', 'turtle1').get_parameter_value().string_value

    self.tf_buffer = Buffer()
    self.tf_listener = TransformListener(self.tf_buffer, self)

    self.spawner = self.create_client(Spawn, 'spawn')
    self.turtle_spawning_service_ready = False
    self.turtle_spawned = False

    self.publisher = self.create_publisher(Twist, 'turtle2/cmd_vel', 1)
    self.timer = self.create_timer(1.0, self.on_timer)
```

5. publisher

publisher는 'turtle2/cmd_vel' 토픽에 Twist 메시지를 퍼블리시하여 turtle2의 속도를 제어

6. timer

create_timer는 1초마다 on_timer 메서드를 호출하는 타이머를 생성

✓ Writing a listener – 코드 설명



```
def on_timer(self):
    # Store frame names in variables that will be used to
    # compute transformations
    from_frame_rel = self.target_frame
    to_frame_rel = 'turtle2'

    ...

```

- **on_timer** 함수
1. 프레임 정의

from_frame_rel은 목표 프레임(turtle1), to_frame_rel은 turtle2로, turtle2가 목표 프레임인 turtle1을 따라가도록 변환 정보를 계산

```
...  
  
    if self.turtle_spawning_service_ready:  
        if self.turtle_spawned:  
            # Look up for the transformation between  
            target_frame and turtle2 frames  
            # and send velocity commands for turtle2 to  
            reach target_frame  
            try:  
                t = self.tf_buffer.lookup_transform(  
                    to_frame_rel,  
                    from_frame_rel,  
                    rclpy.time.Time())  
            except TransformException as ex:  
                self.get_logger().info(  
                    f'Could not transform {to_frame_rel}  
                    to {from_frame_rel}: {ex}')  
            return  
  
...  

```

■ on_timer 함수

2. 터틀 스폰 및 변환 계산 로직

터틀 스폰이 완료되었다면, `lookup_transform`을 통해 `turtle1`에서 `turtle2`로의 변환 정보를 조회하여 `turtle2`가 `turtle1`을 따라가는 데 필요한 속도와 방향을 계산하고, 변환 정보가 없으면 `TransformException` 예외를 처리하여 로그를 기록

```
...  
  
else:  
    if self.spawner.service_is_ready():  
        # Initialize request with turtle name and coordinates  
        # Note that x, y and theta are defined as floats in turtlesim/srv/Spawn  
        request = Spawn.Request()  
        request.name = 'turtle2'  
        request.x = float(4)  
        request.y = float(2)  
        request.theta = float(0)  
        # Call request  
        self.result = self.spawner.call_async(request)  
        self.turtle_spawning_service_ready = True  
    else:  
        # Check if the service is ready  
        self.get_logger().info('Service is not ready')
```

▪ **on_timer** 함수

3. 스폰 서비스 완료 여부 확인

스폰이 터틀이 성공적으로 생성되었음을 기록

4. 스폰 요청

service_is_ready() 메서드를 통해 스폰 서비스가 준비되었는지 확인한 후, 터틀2를 특정 위치에 스폰하도록 비동기 요청을 보냄

✓ Writing a listener – 코드 설명

▪ main 함수

rclpy.init()을 통해 ROS2를 초기화하고, FrameListener 노드를 생성한 후 rclpy.spin()을 통해 노드를 지속적으로 실행

```
● ● ●  
def main( ):  
    rclpy.init()  
    node = FrameListener()  
    try:  
        rclpy.spin(node)  
    except KeyboardInterrupt:  
        pass  
  
    rclpy.shutdown()
```

✓ Writing a listener – setup.py

- ros2 run 명령으로 노드를 실행하기 위하여 `setup.py`(src/learning_tf2_py 디렉터리에 있음)에 `entry_points`를 추가



```
entry_points={  
    "console_scripts": [  
        "static_turtle_tf2_broadcaster = learning_tf2_py.static_turtle_tf2_broadcaster:main",  
        "turtle_tf2_broadcaster = learning_tf2_py.turtle_tf2_broadcaster:main",  
        'turtle_tf2_listener = learning_tf2_py.turtle_tf2_listener:main',  
  
    ],  
},
```

✓ Writing a listener– launch file

- `src/learning_tf2_py/launch` 폴더에 있는 `turtle_tf2_demo.launch.py` 파일을 아래와 같이 수정

```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration

from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='learning_tf2_py',
            executable='turtle_tf2_broadcaster',
            name='broadcaster1',
            parameters=[
                {'turtlename': 'turtle1'}
            ]
        ),
    ])
```

```
        DeclareLaunchArgument(
            'target_frame', default_value='turtle1',
            description='Target frame name.'
        ),
        Node(
            package='learning_tf2_py',
            executable='turtle_tf2_broadcaster',
            name='broadcaster2',
            parameters=[
                {'turtlename': 'turtle2'}
            ]
        ),
        Node(
            package='learning_tf2_py',
            executable='turtle_tf2_listener',
            name='listener',
            parameters=[
                {'target_frame': LaunchConfiguration('target_frame')}
            ]
        ),
    ])
```

✓ Writing a listener– launch file

- `src/learning_tf2_py/launch` 폴더에 있는 `turtle_tf2_demo.launch.py` 파일을 아래와 같이 수정

```
DeclareLaunchArgument(  
    'target_frame', default_value='turtle1',  
    description='Target frame name.'  
,  
    Node(  
        package='learning_tf2_py',  
        executable='turtle_tf2_broadcaster',  
        name='broadcaster2',  
        parameters=[  
            {'turtlename': 'turtle2'}  
        ]  
,  
        Node(  
            package='learning_tf2_py',  
            executable='turtle_tf2_listener',  
            name='listener',  
            parameters=[  
                {'target_frame': LaunchConfiguration('target_frame')}  
            ]  
,  
        ))
```

1. **DeclareLaunchArgument**

런치 파일 실행 시 target_frame이라는 인자를 받아 사용하며 기본값은 turtle1임

2. **Broadcaster2 노드**

learning_tf2_py 패키지를 사용하여 turtle2의 좌표 정보를 브로드캐스트하는 노드

3. **Listener 노드**

learning_tf2_py 패키지의 TF2 리스너 노드를 실행하는 역할

✓ Writing a listener – Build

- 빌드 전 누락된 종속성 확인하기



```
rosdep install -i --from-path src --rosdistro humble -y
```

- 작업 공간에서 새 패키지를 빌드



```
colcon build --packages-select learning_tf2_py
```

- 작업 공간의 루트로 이동한 후 설정 파일 가져오기



```
. install/setup.bash
```

✓ Writing a listener – Run

1. 다음 명령어로 turtle_demo 실행(잠시 기다리면 두마리의 거북이를 확인 가능)



```
ros2 launch learning_tf2_py turtle_tf2_demo.launch.py
```

2. 다음 명령어로 turlte_teleop_key를 실행



```
ros2 run turtlesim turtle_teleop_key
```

3. teleop_key를 이용하여 turtle을 움직일 때마다 다른 한 거북이가 따라오는 것을 확인 가능

Adding a fixed frame broadcaster

- tf2에 새로운 프레임을 추가하기

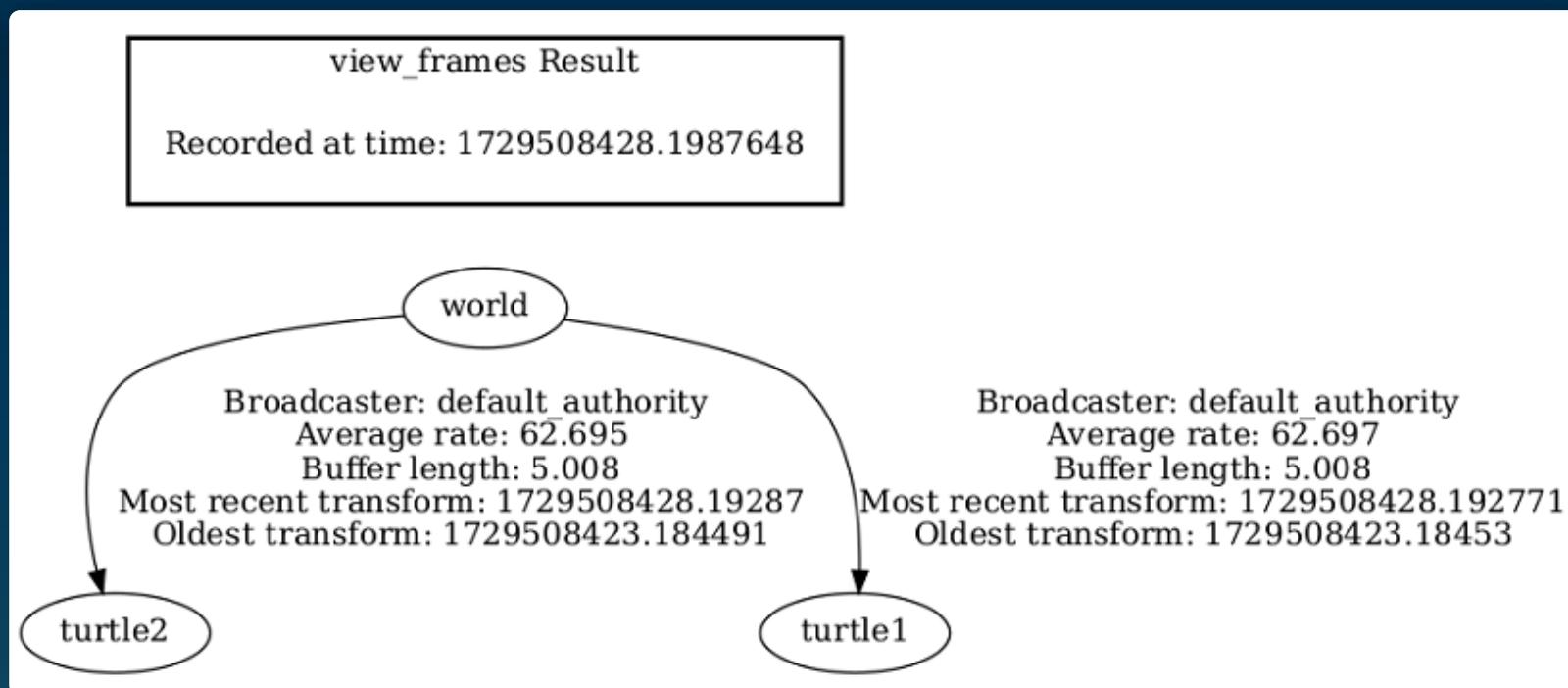
✓ Adding a fixed frame broadcaster

1. 본 섹션에서는 변환 트리에 추가 고정 및 동적 프레임을 추가하는 방법을 다룸
2. tf2에 프레임을 추가하는 것은 tf2 브로드캐스터를 만드는 것과 유사하지만 몇가지 추가 기능이 있음
3. 본 섹션에서는 이 추가 기능에 관하여 다룸

✓ Adding a fixed frame broadcaster

▪ tf2 tree

1. tf2는 프레임의 트리 구조를 구축하므로 프레임 구조에서 닫힌 루프를 허용하지 않음
2. 즉, 프레임은 하나의 부모만 가지지만 여러 자식을 가지는 것이 가능
3. 현재 tf2 트리에는 world, turtle1 및 turtle2의 세 프레임이 존재
4. tf2에 새 프레임을 추가하려면 기존 프레임 세 개 중 하나가 부모 프레임이어야 하며 새 프레임은 자식 프레임이 되어야 함



✓ Adding a fixed frame broadcaster

1. src/learning_tf2_py/learning_tf2_py
에 fixed_frame_tf2_broadcaster.py
파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
from geometry_msgs.msg import TransformStamped
import rclpy
from rclpy.node import Node
from tf2_ros import TransformBroadcaster
class FixedFrameBroadcaster(Node):
    def __init__(self):
        super().__init__('fixed_frame_tf2_broadcaster')
        self.tf_broadcaster = TransformBroadcaster(self)
        self.timer = self.create_timer(0.1, self.broadcast_timer_callback)

    def broadcast_timer_callback(self):
        t = TransformStamped()

        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'turtle1'
        t.child_frame_id = 'carrot1'
        t.transform.translation.x = 0.0
        t.transform.translation.y = 2.0
        t.transform.translation.z = 0.0
        t.transform.rotation.x = 0.0
        t.transform.rotation.y = 0.0
        t.transform.rotation.z = 0.0
        t.transform.rotation.w = 1.0

        self.tf_broadcaster.sendTransform(t)
```

(이어서 작성)

✓ Adding a fixed frame broadcaster

1. src/learning_tf2_py/learning_tf2_py에 fixed_frame_tf2_broadcaster.py파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●

def main( ):
    rclpy.init()
    node = FixedFrameBroadcaster( )
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Adding a fixed frame broadcaster – 코드 설명

■ 필요한 라이브러리 가져오기

- tf2_ros.TransformBroadcaster: TF2를 통해 좌표 프레임 간의 변환을 브로드캐스트하는 ROS2 클래스

```
● ● ●  
from geometry_msgs.msg import TransformStamped  
  
import rclpy  
from rclpy.node import Node  
  
from tf2_ros import TransformBroadcaster
```

✓ Adding a fixed frame broadcaster – 코드 설명

▪ 생성자

1. Node를 초기화하면서 노드 이름을 'fixed_frame_tf2_broadcaster'로 설정
2. TransformBroadcaster 객체를 생성해, 이 노드에서 변환을 브로드캐스트를 준비
3. 0.1초마다 broadcast_timer_callback 함수를 호출하는 타이머를 설정

```
● ● ●  
class FixedFrameBroadcaster(Node):  
  
    def __init__(self):  
        super().__init__('fixed_frame_tf2_broadcaster')  
        self.tf_broadcaster = TransformBroadcaster(self)  
        self.timer = self.create_timer(0.1, self.broadcast_timer_callback)
```

✓ Adding a fixed frame broadcaster – 코드 설명

- **broadcast_timer_callback** 함수: 고정된 좌표 변환 정보를 생성하여 TF2로 브로드캐스트하는 역할
 - carrot1 프레임은 turtle1 프레임을 기준으로 y 축으로 2미터 오프셋되어 있음



```
def broadcast_timer_callback(self):
    t = TransformStamped()      # 변환 정보를 담을 TransformStamped 메시지 객체 생성

    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'turtle1'  # 'turtle1': 부모 프레임을 'turtle1'로 설정
    t.child_frame_id = 'carrot1'  # 자식 프레임을 'carrot1'로 설정

    # 변환의 위치 정보를 설정
    t.transform.translation.x = 0.0
    t.transform.translation.y = 2.0
    t.transform.translation.z = 0.0

    # 회전 정보를 설정합니다
    t.transform.rotation.x = 0.0
    t.transform.rotation.y = 0.0
    t.transform.rotation.z = 0.0
    t.transform.rotation.w = 1.0

    self.tf_broadcaster.sendTransform(t)  # 설정된 변환 정보를 TF2로 브로드캐스트
```

✓ Adding a fixed frame broadcaster – 코드 설명

▪ main 함수

1. rclpy.init()을 통해 ROS2를 초기화
2. FixedFrameBroadcaster 노드를 생성하여 TF2 브로드캐스트 기능을 실행
3. rclpy.spin()을 통해 노드를 지속적으로 실행

```
● ● ●

def main( ):
    rclpy.init()
    node = FixedFrameBroadcaster()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Adding a fixed frame broadcaster – setup.py

- ros2 run 명령으로 노드를 실행하기 위하여 `setup.py`(`src/learning_tf2_py` 디렉터리에 있음)에 `entry_points`를 추가

```
● ● ●  
entry_points={  
    "console_scripts": [  
        "static_turtle_tf2_broadcaster = learning_tf2_py.static_turtle_tf2_broadcaster:main",  
        "turtle_tf2_broadcaster = learning_tf2_py.turtle_tf2_broadcaster:main",  
        "turtle_tf2_listener = learning_tf2_py.turtle_tf2_listener:main",  
        "fixed_frame_tf2_broadcaster = learning_tf2_py.fixed_frame_tf2_broadcaster:main",  
    ],  
},
```

✓ Adding a fixed frame broadcaster – launch file

1. src/learning_tf2_py에 launch폴더에 turtle_tf2_fixed_frame_demo.launch.py라는 Launch파일 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

from launch_ros.actions import Node
```

(이어서 작성)

✓ Adding a fixed frame broadcaster – launch file

- src/learning_tf2_py에 launch폴더에 turtle_tf2_fixed_frame_demo.launch.py라는 Launch파일 생성



```
def generate_launch_description():
    demo_nodes = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
            get_package_share_directory('learning_tf2_py'), 'launch'),
            '/turtle_tf2_demo.launch.py')]),
    )

    return LaunchDescription([
        demo_nodes,
        Node(
            package='learning_tf2_py',
            executable='fixed_frame_tf2_broadcaster',
            name='fixed_broadcaster',
        ),
    ])

```

- fixed_frame_tf2_broadcaster 노드를 사용하여 고정 carrot1 프레임을 turtlesim world에 추가

✓ Adding a fixed frame broadcaster – Build

- 빌드 전 누락된 종속성 확인하기



```
rosdep install -i --from-path src --rosdistro humble -y
```

- 작업 공간에서 새 패키지를 빌드



```
colcon build --packages-select learning_tf2_py
```

- 작업 공간의 루트로 이동한 후 설정 파일 가져오기



```
. install/setup.bash
```

✓ Adding a fixed frame broadcaster – Run

1. 다음 명령어를 이용하여 broadcaster demo 실행

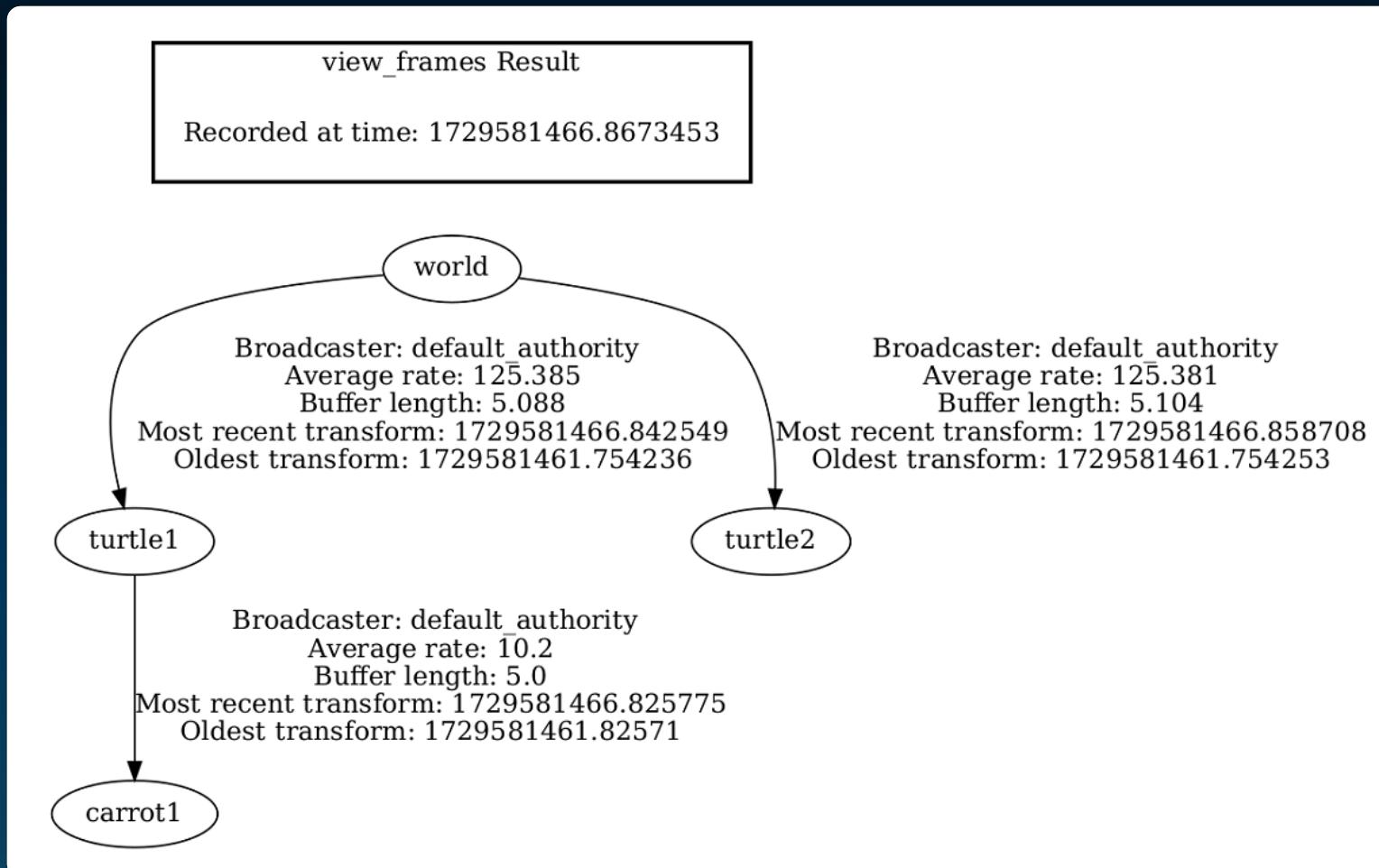
```
ros2 launch learning_tf2_py turtle_tf2_fixed_frame_demo.launch.py
```

2. 다음 명령어를 이용하여 tf2가 브로드캐스트하는 프레임의 다이어그램을 확인

```
ros2 run tf2_tools view_frames
```

✓ Adding a fixed frame broadcaster – Run

3. 다이어그램에 carrot1 프레임이 추가된 것을 관찰 가능



✓ Adding a fixed frame broadcaster – Run

4. 거북이의 동작 변경을 위해서는 프레임을 바꿔줄 필요가 있음
(여전히 두번째 거북이는 첫번째 거북이 바로 뒤를 따라다님)
5. 이를 위해 다음과 같이 명령어에 인자 추가

```
ros2 launch learning_tf2_py turtle_tf2_fixed_frame_demo.launch.py target_frame:=carrot1
```

6. 새로운 터미널에서 `turtle_teleop_key`를 실행하여 거북이 움직이기

```
ros2 run turtlesim turtle_teleop_key
```

✓ Adding a fixed frame broadcaster – Run

7. 다음과 같이 두번째 거북이가 첫번째 거북이 대신 carrot뒤를 따라다니는 것을 확인 가능(carrot은 carrot1 프레임은 turtle1 프레임을 기준으로 y 축으로 2미터 오프셋 되어 있음)



Adding a dynamic frame broadcaster

- tf2에 새로운 프레임을 추가하기

✓ Adding a dynamic frame broadcaster

1. carrot1 프레임을 변경하여 시간이 지남에 따라 turtle1 프레임과 관련하여
변경되도록 전환함

✓ Adding a dynamic frame broadcaster

1. src/learning_tf2_py/learning_tf2_py에 dynamic_frame_tf2_broadcaster파일을 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●  
import math  
  
from geometry_msgs.msg import TransformStamped  
  
import rclpy  
from rclpy.node import Node  
  
from tf2_ros import TransformBroadcaster
```

(이어서 작성)

✓ Adding a dynamic frame broadcaster

```
class DynamicFrameBroadcaster(Node):

    def __init__(self):
        super().__init__('dynamic_frame_tf2_broadcaster')
        self.tf_broadcaster = TransformBroadcaster(self)
        self.timer = self.create_timer(0.1, self.broadcast_timer_callback)

    def broadcast_timer_callback(self):
        seconds, _ = self.get_clock().now().seconds_nanoseconds()
        x = seconds * math.pi

        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'turtle1'
        t.child_frame_id = 'carrot1'
        t.transform.translation.x = 10 * math.sin(x)
        t.transform.translation.y = 10 * math.cos(x)
        t.transform.translation.z = 0.0
        t.transform.rotation.x = 0.0
        t.transform.rotation.y = 0.0
        t.transform.rotation.z = 0.0
        t.transform.rotation.w = 1.0

        self.tf_broadcaster.sendTransform(t)
```

(0|어서 작성)

✓ Adding a dynamic frame broadcaster

```
● ● ●  
def main():  
    rclpy.init()  
    node = DynamicFrameBroadcaster()  
    try:  
        rclpy.spin(node)  
    except KeyboardInterrupt:  
        pass  
  
    rclpy.shutdown()
```

✓ Adding a dynamic frame broadcaster – 코드 설명

▪ 필요한 라이브러리 가져오기

- tf2_ros.TransformBroadcaster: TF2를 통해 좌표 프레임 간의 변환을 브로드캐스트하는 ROS2 클래스

```
● ● ●  
import math  
  
from geometry_msgs.msg import TransformStamped  
  
import rclpy  
from rclpy.node import Node  
  
from tf2_ros import TransformBroadcaster
```

✓ Adding a dynamic frame broadcaster – 코드 설명

▪ 생성자

1. Node를 초기화하면서 노드 이름을 'dynamic_frame_tf2_broadcaster'로 설정
2. TransformBroadcaster 객체를 생성해, 이 노드에서 변환을 브로드캐스트를 준비
3. 0.1초마다 broadcast_timer_callback 함수를 호출하는 타이머를 설정

```
class DynamicFrameBroadcaster(Node):  
  
    def __init__(self):  
        super().__init__('dynamic_frame_tf2_broadcaster')  
        self.tf_broadcaster = TransformBroadcaster(self)  
        self.timer = self.create_timer(0.1, self.broadcast_timer_callback)
```

✓ Adding a dynamic frame broadcaster – 코드 설명

- **broadcast_timer_callback** 함수: 고정된 좌표 변환 정보를 생성하여 TF2로 브로드캐스트하는 역할

- x와 y 오프셋의 고정된 정의 대신, 현재 시간에 sin()과 cos() 함수를 사용하여 carrot1의 오프셋이 지속적으로 변경되도록 함

```
def broadcast_timer_callback(self):
    seconds, _ = self.get_clock().now().seconds.nanoseconds()
    x = seconds * math.pi

    t = TransformStamped() # 변환 정보를 담을 TransformStamped 메시지 객체 생성
    t.header.stamp = self.get_clock().now().to_msg()
    t.header.frame_id = 'turtle1' # 'turtle1': 부모 프레임을 'turtle1'로 설정
    t.child_frame_id = 'carrot1' # 자식 프레임을 'carrot1'로 설정

    # 변환의 위치 정보를 설정
    t.transform.translation.x = 10 * math.sin(x)
    t.transform.translation.y = 10 * math.cos(x)
    t.transform.translation.z = 0.0
    # 회전 정보를 설정합니다
    t.transform.rotation.x = 0.0
    t.transform.rotation.y = 0.0
    t.transform.rotation.z = 0.0
    t.transform.rotation.w = 1.0

    self.tf_broadcaster.sendTransform(t) # 설정된 변환 정보를 TF2로 브로드캐스트
```

✓ Adding a dynamic frame broadcaster – 코드 설명

▪ main 함수

1. rclpy.init()을 통해 ROS2를 초기화
2. DynamicFrameBroadcaster 노드를 생성하여 TF2 브로드캐스트 기능을 실행
3. rclpy.spin()을 통해 노드를 지속적으로 실행

```
● ● ●

def main():
    rclpy.init()
    node = DynamicFrameBroadcaster()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    rclpy.shutdown()
```

✓ Adding a dynamic frame broadcaster – setup.py

- ros2 run 명령으로 노드를 실행하기 위하여 **setup.py(src/learning_tf2_py 디렉터리에 있음)**에 **entry_points**를 추가

```
entry_points={  
    "console_scripts": [  
        "static_turtle_tf2_broadcaster = learning_tf2_py.static_turtle_tf2_broadcaster:main",  
        "turtle_tf2_broadcaster = learning_tf2_py.turtle_tf2_broadcaster:main",  
        "turtle_tf2_listener = learning_tf2_py.turtle_tf2_listener:main",  
        "fixed_frame_tf2_broadcaster = learning_tf2_py.fixed_frame_tf2_broadcaster:main",  
        "dynamic_frame_tf2_broadcaster = learning_tf2_py.dynamic_frame_tf2_broadcaster:main",  
    ],  
},
```

✓ Adding a dynamic frame broadcaster – launch file

1. launch폴더에 turtle_tf2_dynamic_frame_demo.launch.py라는 Launch파일 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

from launch_ros.actions import Node
```

✓ Adding a dynamic frame broadcaster – launch file

1. launch폴더에 turtle_tf2_dynamic_frame_demo.launch.py라는 Launch파일 생성
2. 생성된 파일에 다음과 같이 코드 작성

```
● ● ●

def generate_launch_description():
    demo_nodes = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            [
                os.path.join(get_package_share_directory("learning_tf2_py"), "launch"),
                "/turtle_tf2_demo.launch.py",
            ]
        ),
        launch_arguments={"target_frame": "carrot1"}.items(),
    )

    return LaunchDescription(
        [
            demo_nodes,
            Node(
                package="learning_tf2_py",
                executable="dynamic_frame_tf2_broadcaster",
                name="dynamic_broadcaster",
            ),
        ],
    )
```

✓ Adding a dynamic frame broadcaster – Build

- 빌드 전 누락된 종속성 확인하기



```
rosdep install -i --from-path src --rosdistro humble -y
```

- 작업 공간에서 새 패키지를 빌드



```
colcon build --packages-select learning_tf2_py
```

- 작업 공간의 루트로 이동한 후 설정 파일 가져오기



```
. install/setup.bash
```

✓ Adding a dynamic frame broadcaster – Run

1. 다음 명령어를 이용하여 broadcaster demo 실행

```
ros2 launch learning_tf2_py turtle_tf2_dynamic_frame_demo.launch.py
```

2. 실행 결과 아래와 같이 두번째 거북이가 랜덤하게 움직이는 것을 관찰 가능(움직임은 매번 달라짐)

