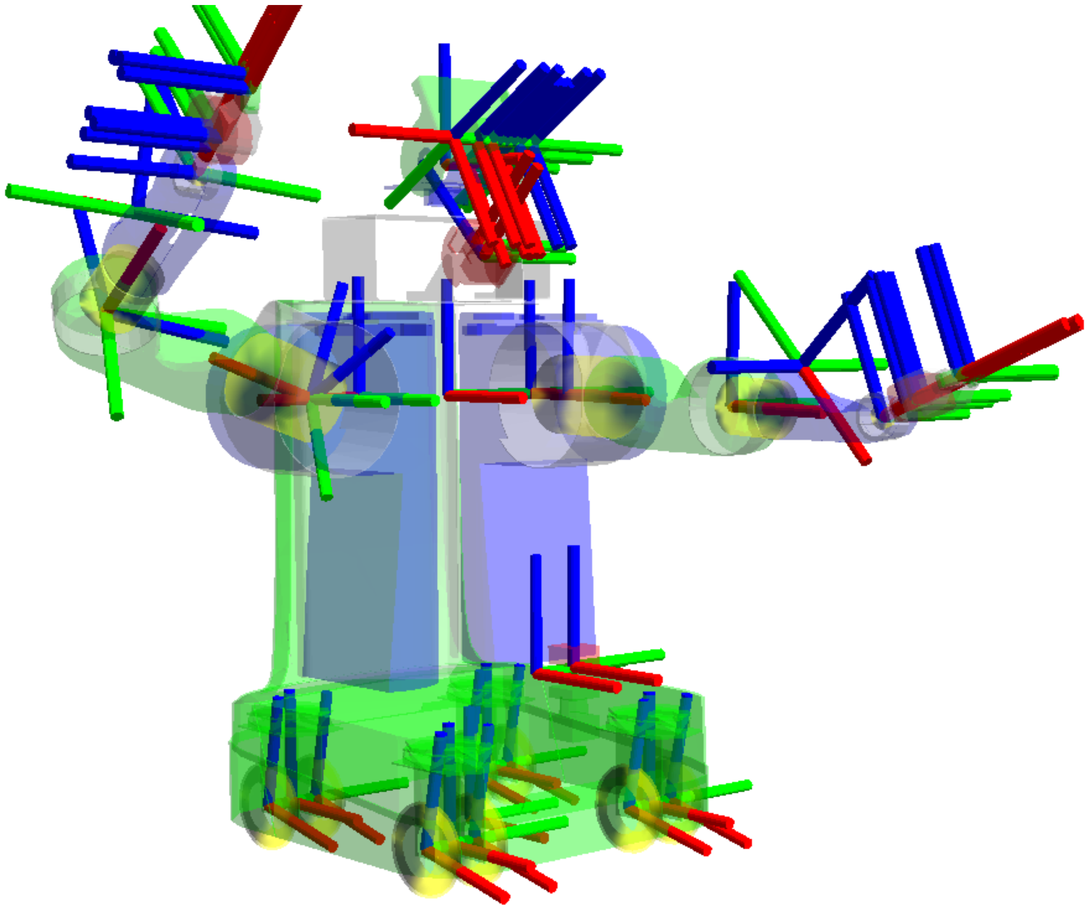
ROS2: TF2

운영체제의 실제
안인규 (Inkyu An)

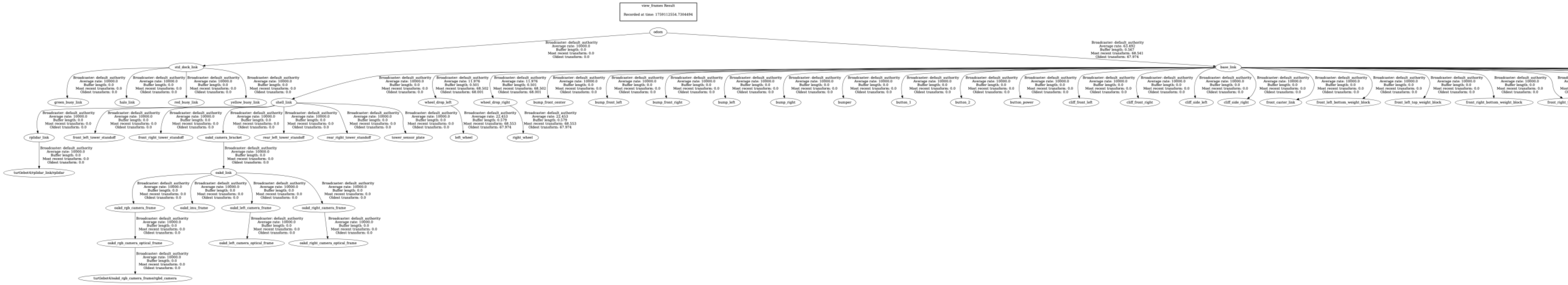
TF2

- A robot is made up of many parts and sensors (a body, wheels, arms, cameras, and LiDAR)
- <u>Each of these has its own unique coordinate frame</u>
  - "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.

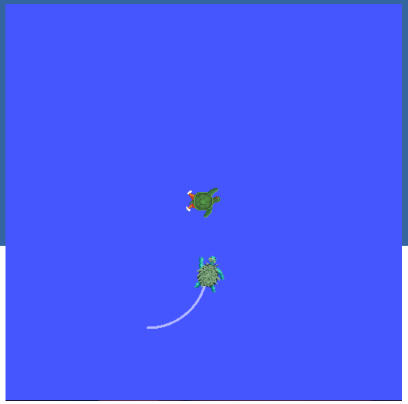


TF2

- **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:

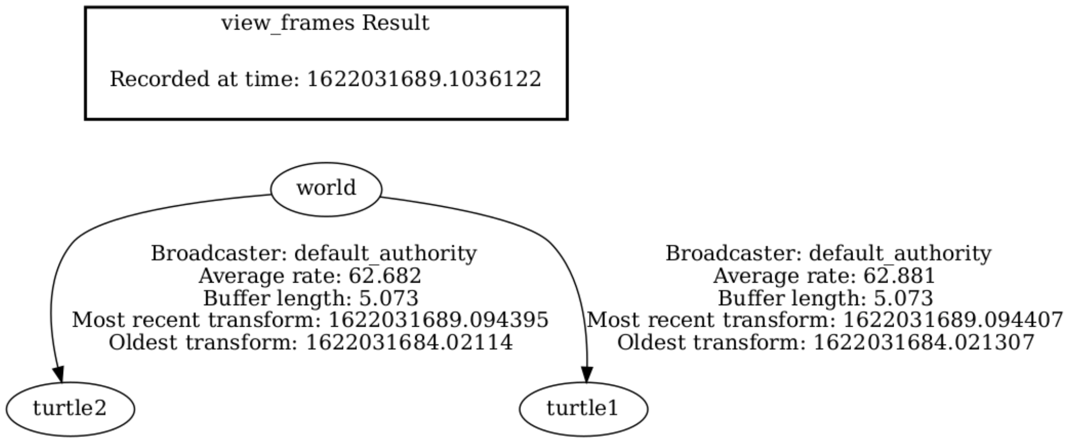
```
$ ros2 run turtlesim turtle_teleop_key
```

  - 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 publishes 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
```

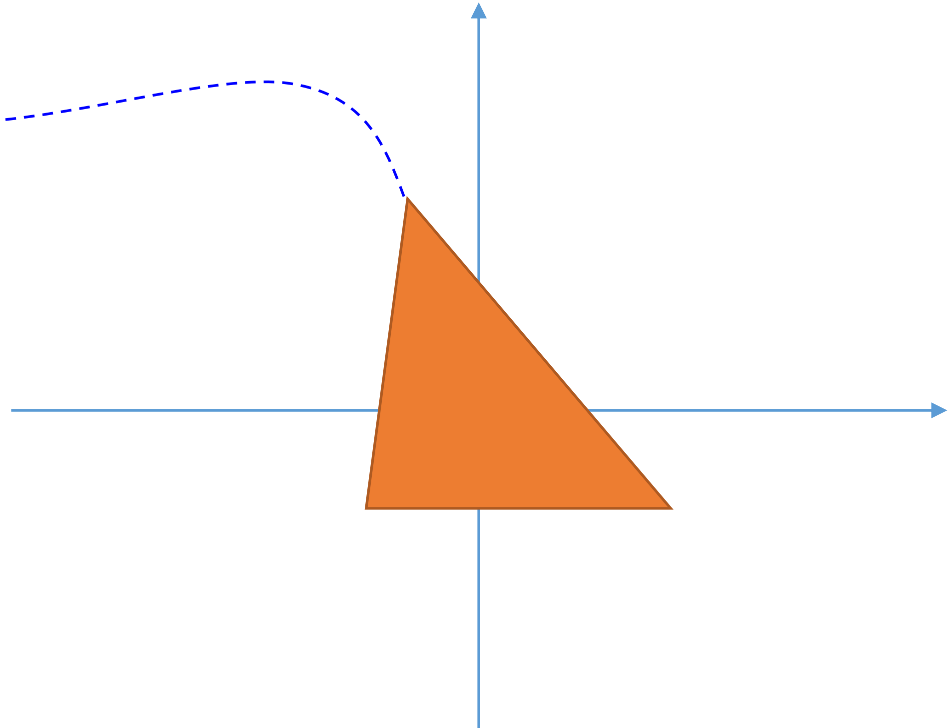
TF2

- What is the Matrix (=Transformation Matrix)

```
- 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
```

- $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$, transform matrix
  - R: 3×3 rotation matrix
  - $t = \left[a_x, a_y, a_z\right]^T$, translation vector

- Given a point $p = [x_1, y_1, z_1, 1]^T$,
  - $p' = T \cdot p$



TF2

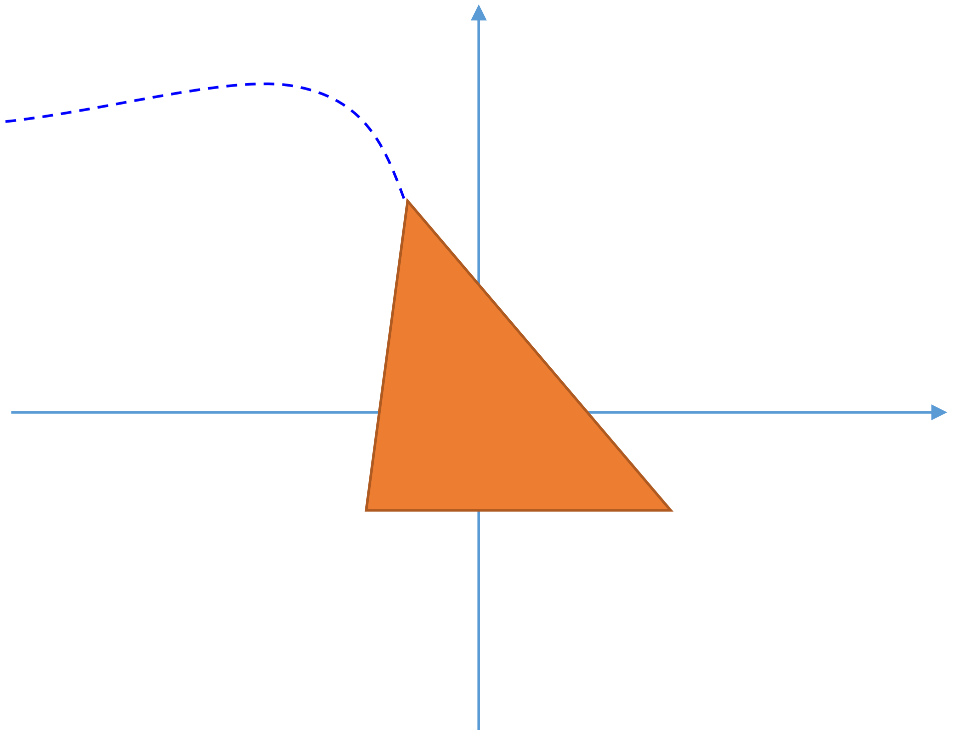
- What is the Matrix (=Transformation Matrix)

```
- 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
```

- $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$, transform matrix
  - R: 3×3 rotation matrix
  - $t = [a_x, a_y, a_z]^T$, translation vector

- Given a point $p = [x_1, y_1, z_1, 1]^T$,
  - $\boxed{p' = T \cdot p}$

$$p' = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ 1 \end{bmatrix} = \begin{bmatrix} R \cdot p_1 + t \\ 1 \end{bmatrix}$$



TF2

- What is the Matrix (=Transformation Matrix)

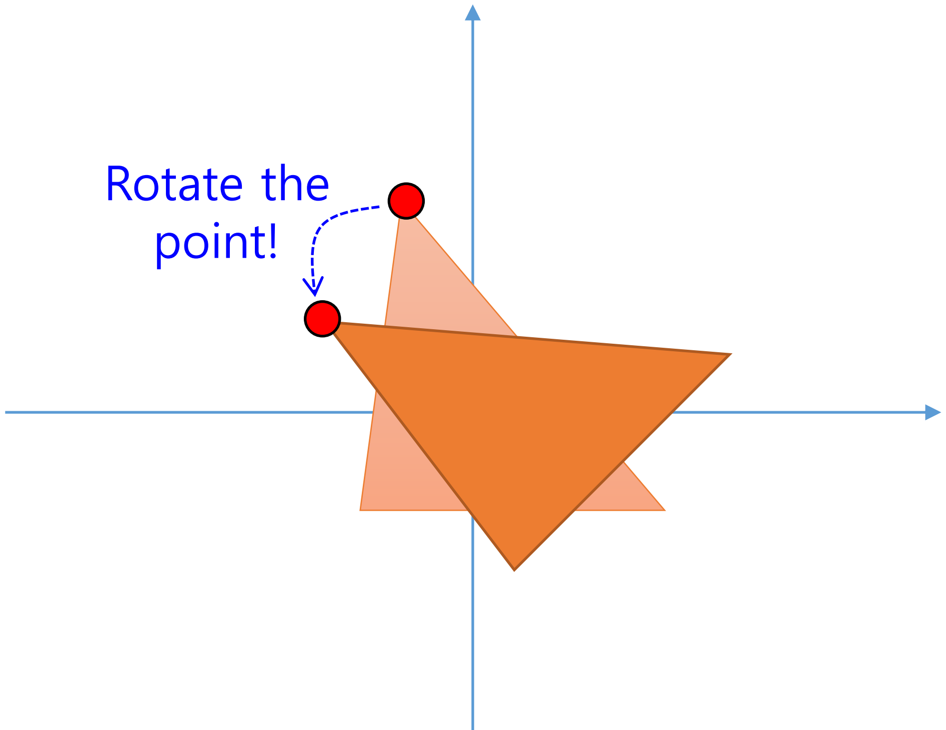
```
- Matrix:
   0.999  0.032  0.000  1.256
  -0.032  0.999 -0.000  0.216
  -0.000  0.000  1.000  0.000
   0.000  0.000  0.000  1.000
```

- $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$, transform matrix
  - R: 3×3 rotation matrix
  - $t = \left[a_x, a_y, a_z\right]^T$, translation vector

- Given a point $p = [x_1, y_1, z_1, 1]^T$,
  - $\boxed{p' = T \cdot p}$

Apply the rotation matrix

$$p' = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \boxed{R \cdot p_1} + t \\ 1 \end{bmatrix}$$



TF2

- What is the Matrix (=Transformation Matrix)

```
- 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
```

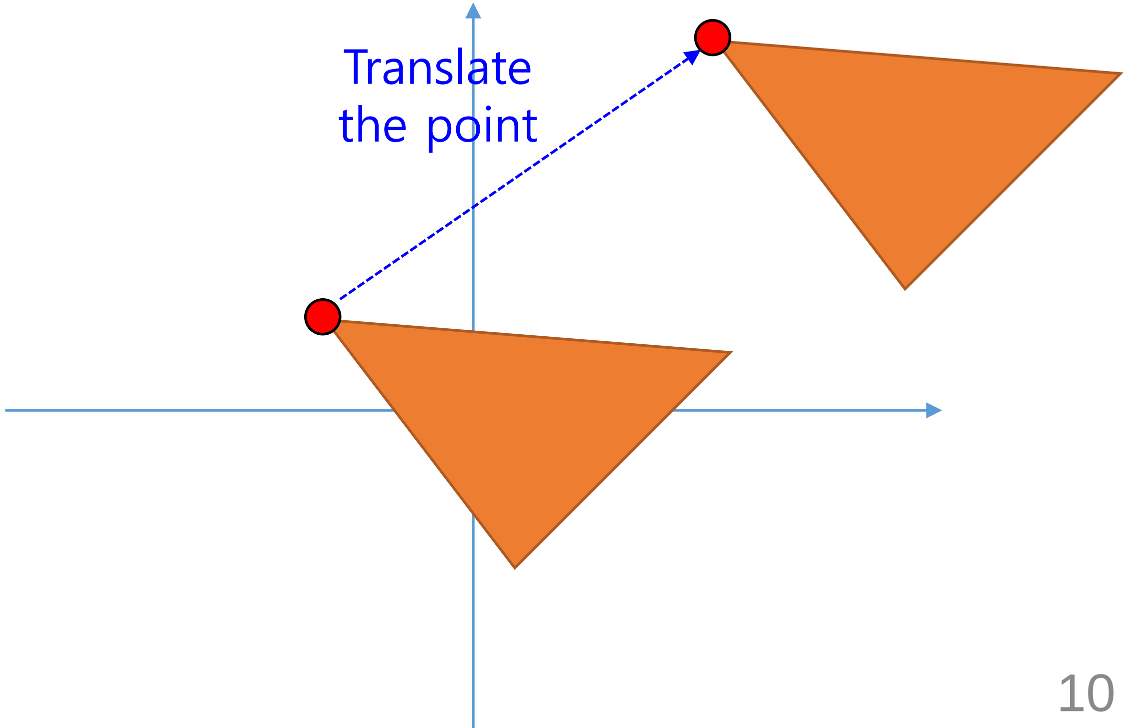
- $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$, transform matrix
  - R: 3×3 rotation matrix
  - $t = \left[a_x, a_y, a_z\right]^T$, translation vector

- Given a point $p = [x_1, y_1, z_1, 1]^T$,
  - $\boxed{p' = T \cdot p}$

Apply the rotation matrix

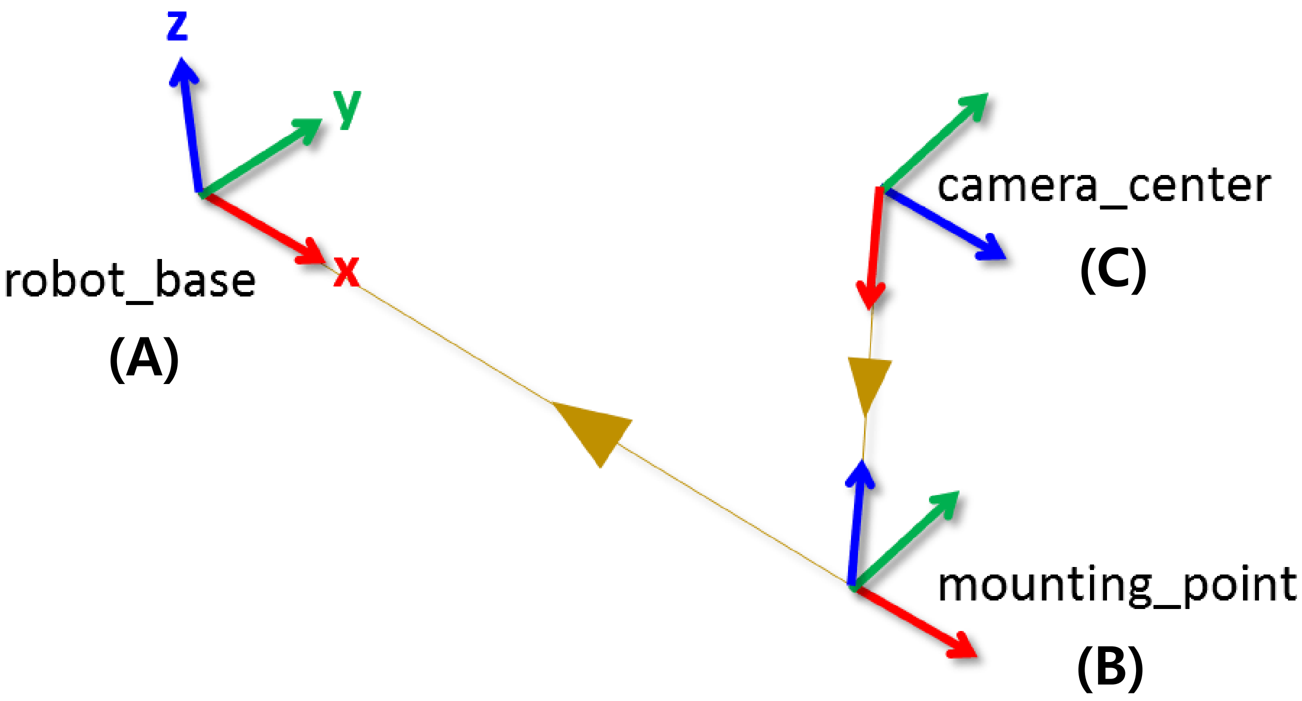
$$p' = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \boxed{R \cdot p_1} + \boxed{t} \\ 1 \end{bmatrix}$$

Apply the translation vector



TF2

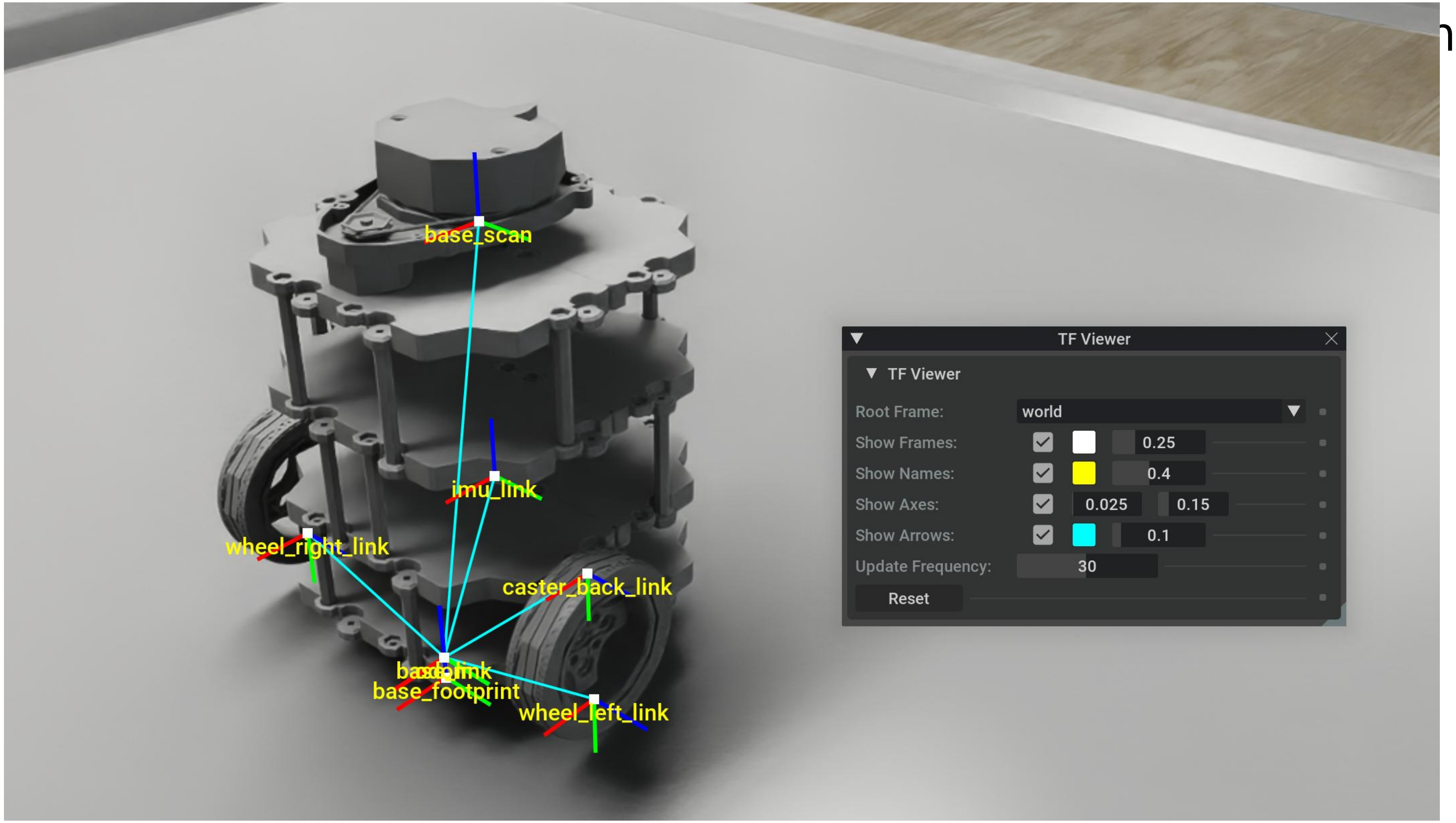
- What is the Matrix (=Transformation Matrix)
  - In TF2, a transformation matrix is used for coordinate transformation
    - **Chain of transformations**
      - $T_{C \to A} = T_{C \to B} \cdot T_{B \to A}$



ROS2에서는,
x: 로봇이 바라보는 방향
y: 로봇의 좌측
z: 로봇의 위측

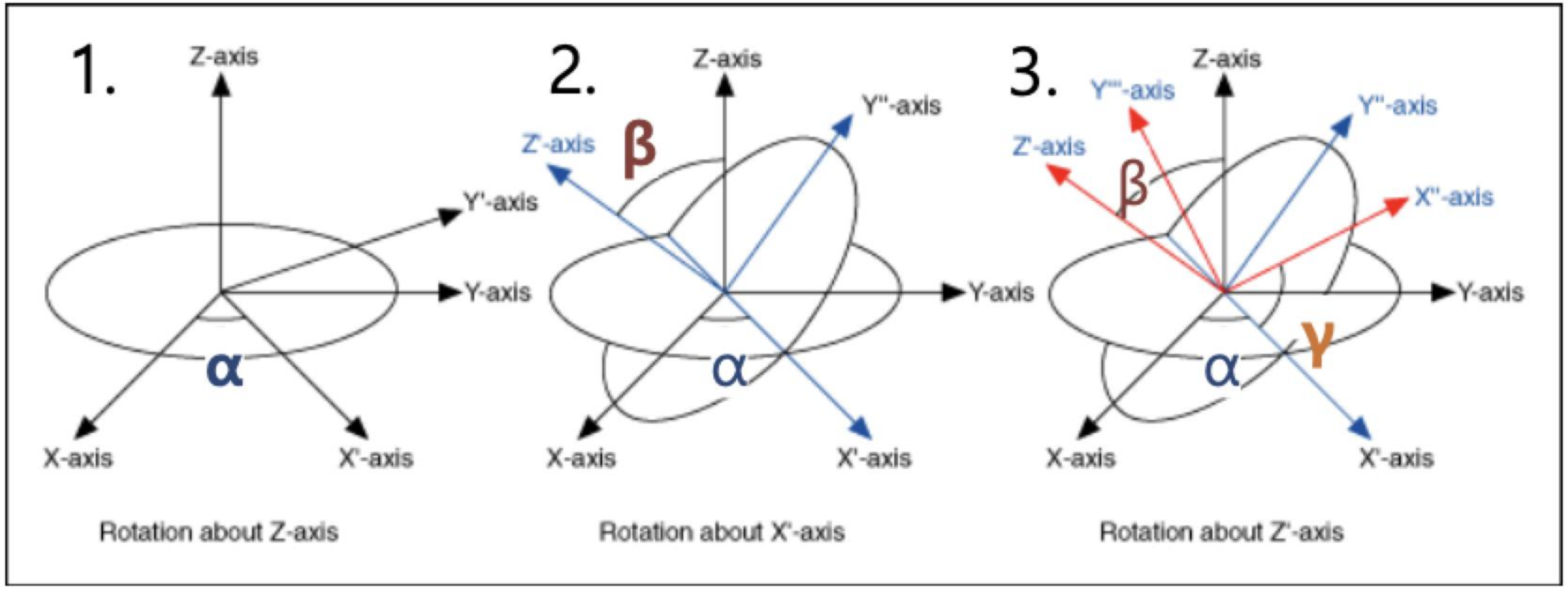
TF2

- What is the Matrix (=Transformation Matrix)
  - I nation



TF2

- How to compute the rotation matrix

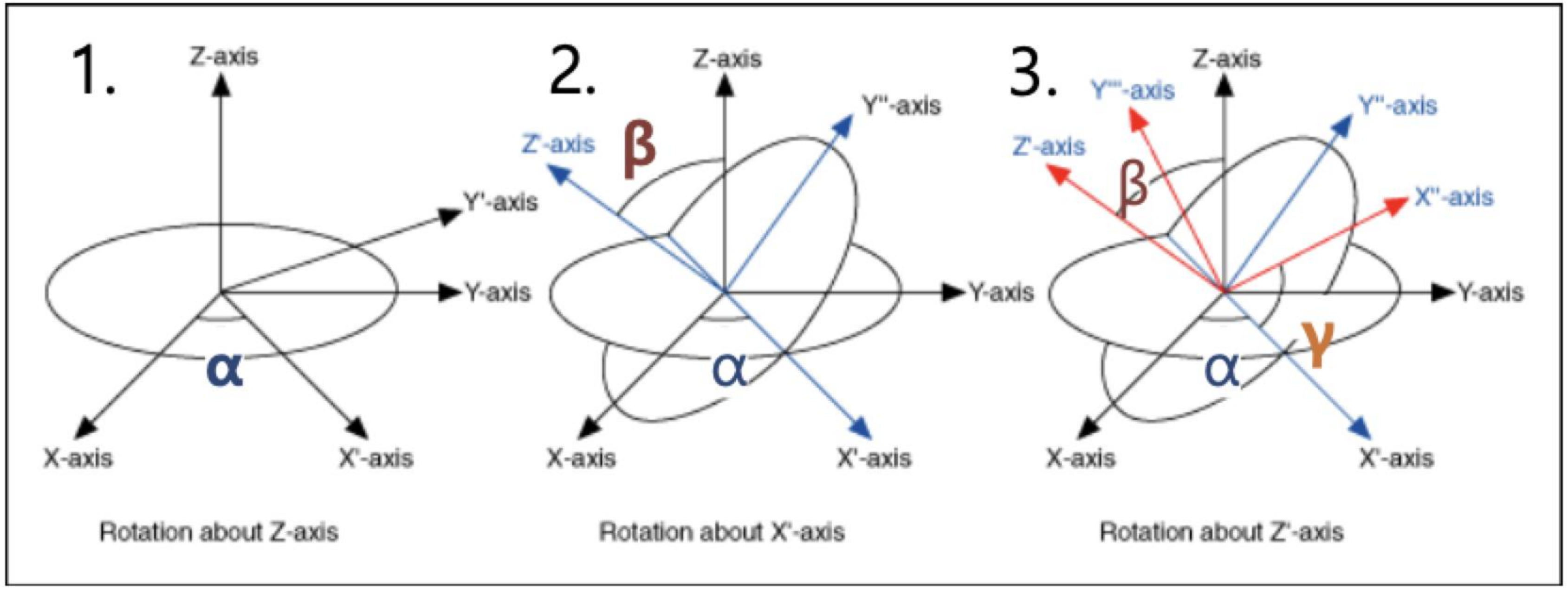


$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = \quad R_z(\alpha) \quad R_x(\beta) \quad R_z(\gamma)$$

TF2

- How to compute the rotation matrix



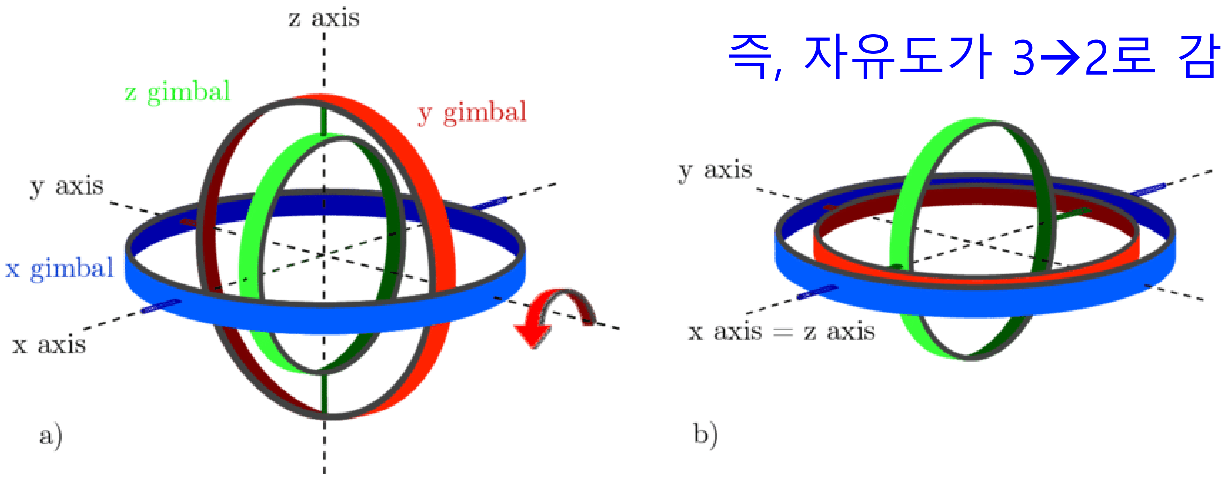
$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Roll **Pitch** **Yaw**

$$R = R_z(\alpha) \quad R_x(\beta) \quad R_z(\gamma)$$

TF2

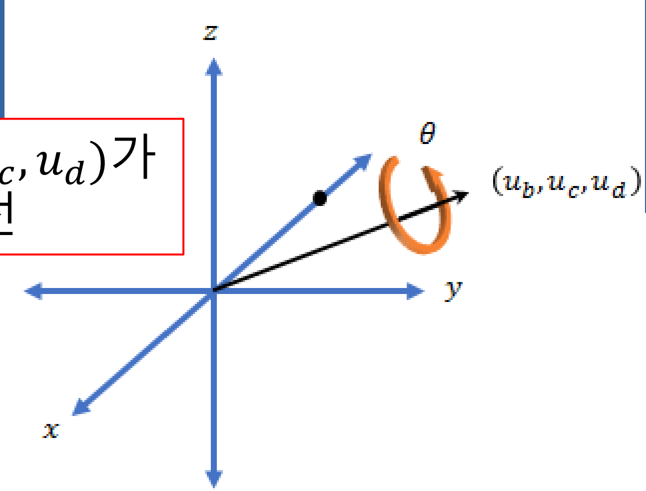
- How to compute the rotation matrix
- **The problems with using Roll, Pitch, and Yaw (Euler angles)**
  - **Discontinuity:**
    - Euler angles $(-180^{\circ} \sim +180^{\circ})$
    - E.g., $+180^{\circ}$ → $+181^{\circ} = -179^{\circ}$
  - **Gimbal lock**



즉, 자유도가 3→2로 감소

TF2

Unit vector (u_b, u_c, u_d)가
θ 만큼 회전



- Quaternion
  - $q = w + xi + yj + zk$

    $w = \cos\left(\frac{\theta}{2}\right),$

    $(x, y, z) = (u_b, u_c, u_d)\sin\left(\frac{\theta}{2}\right)$
  - w: scalar part
  - x, y, z: vector part

- A Unit Quaternion, which is a quaternion with a magnitude (or length) of 1, can represent a specific rotation in 3D space:
  - Euler angle (roll, pitch, yaw) → Quaternion
  - Quaternion → Rotation matrix
  - Rotation matrix + Translation vector → Transformation matrix

$$R = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-zw) & 2(xz+yw) \\ 2(xy+zw) & 1-2(x^2+z^2) & 2(yz-xw) \\ 2(xz-yw) & 2(yz+xw) & 1-2(x^2+y^2) \end{bmatrix}$$

TF2

- How does TF2 manage coordinate frames?
  - **Publishing static transforms** is useful to define the relationship between a robot based and its sensors or non-moving parts: StaticTransformBroadcaster
  - **Handling moving parts**: TransformBroadcaster
  - **Listening transforms**: TransformListener

Download the source code & build it!

TF2

- **Publishing static transforms:**
  - After overlaying,

```
$ ros2 run turtle_tf2_py static_turtle_tf2_broadcaster mystaticturtle 0 0 1 0 0 0
```

  - Echo the tf_static topic:

```
$ ros2 topic echo /tf_static
```

```
$ ros2 topic echo /tf_static
transforms:
- header:
   stamp:
      sec: 1622908754
      nanosec: 208515730
   frame_id: world
child_frame_id: mystaticturtle
transform:
   translation:
      x: 0.0
      y: 0.0
      z: 1.0
   rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
```

TF2

- **Publishing static transforms:**
  - We can publish static transforms using the CLI command:

```
$ ros2 run tf2_ros static_transform_publisher --x 0 --y 0 --z 1 --qx 0 --qy 0 --qz 0 --qw 1 --frame-id world --child-frame-id mystaticturtle
```

  - Launch file:

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

TF2

- **Handling moving parts & Listening transforms:**
  - After overlaying,

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

  - We can control one turtle and watch another one follow it:

```
$ ros2 run turtlesim turtle_teleop_key
```

```
t = self.tf_buffer.lookup_transform(
    to_frame_rel,
    from_frame_rel,
    rclpy.time.Time())
```

to_frame_rel → ‘turtle2’ coordinate (ref)

from_frame_rel → ‘turtle1’ coordinate (target)

TF2를 듣는 부분