

CSC398 Introduction to Autonomous Robots

- Introduction into ROS (3) -

Ubbo Visser

Department of Computer Science
College of Arts and Sciences
University of Miami

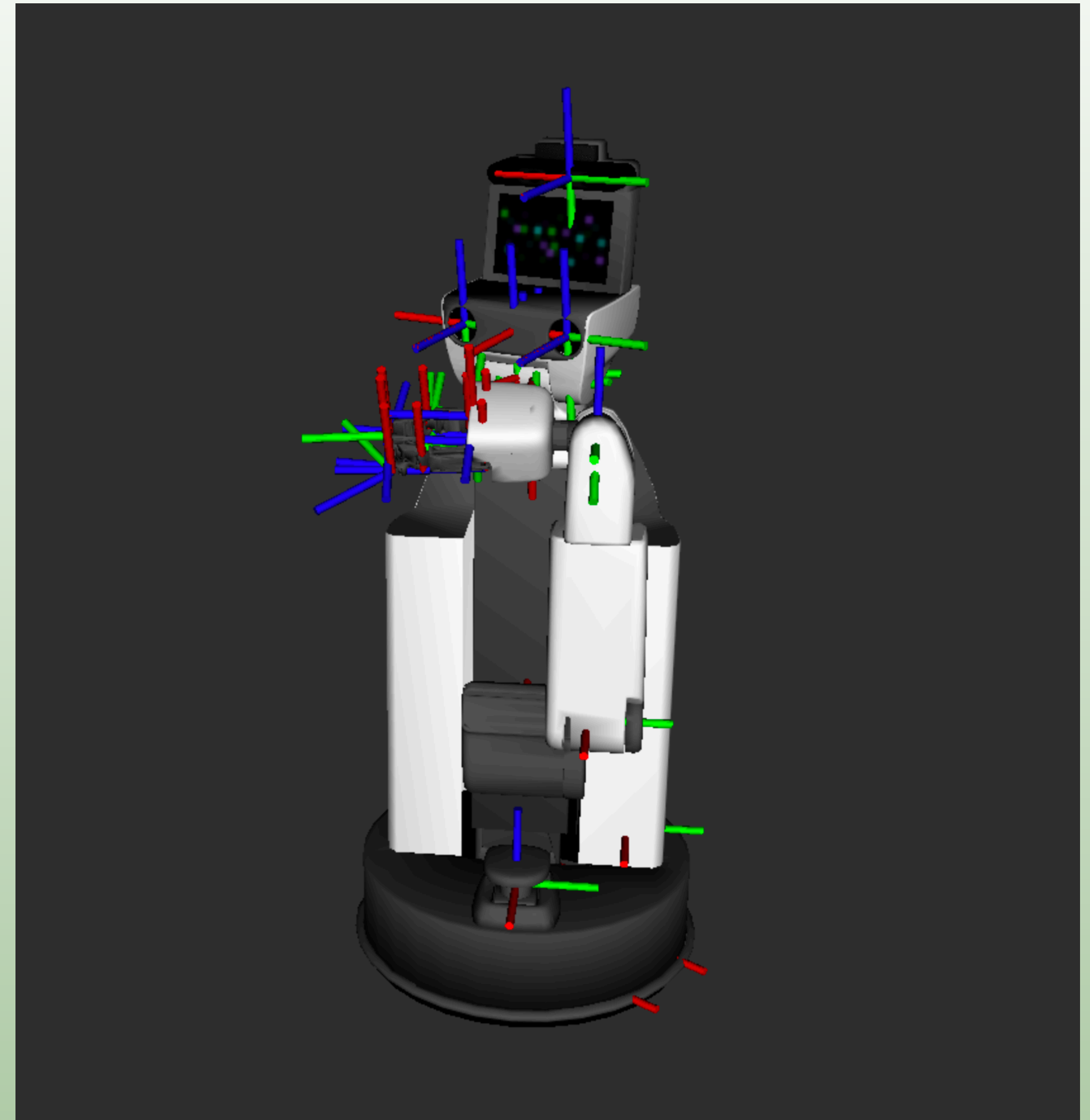
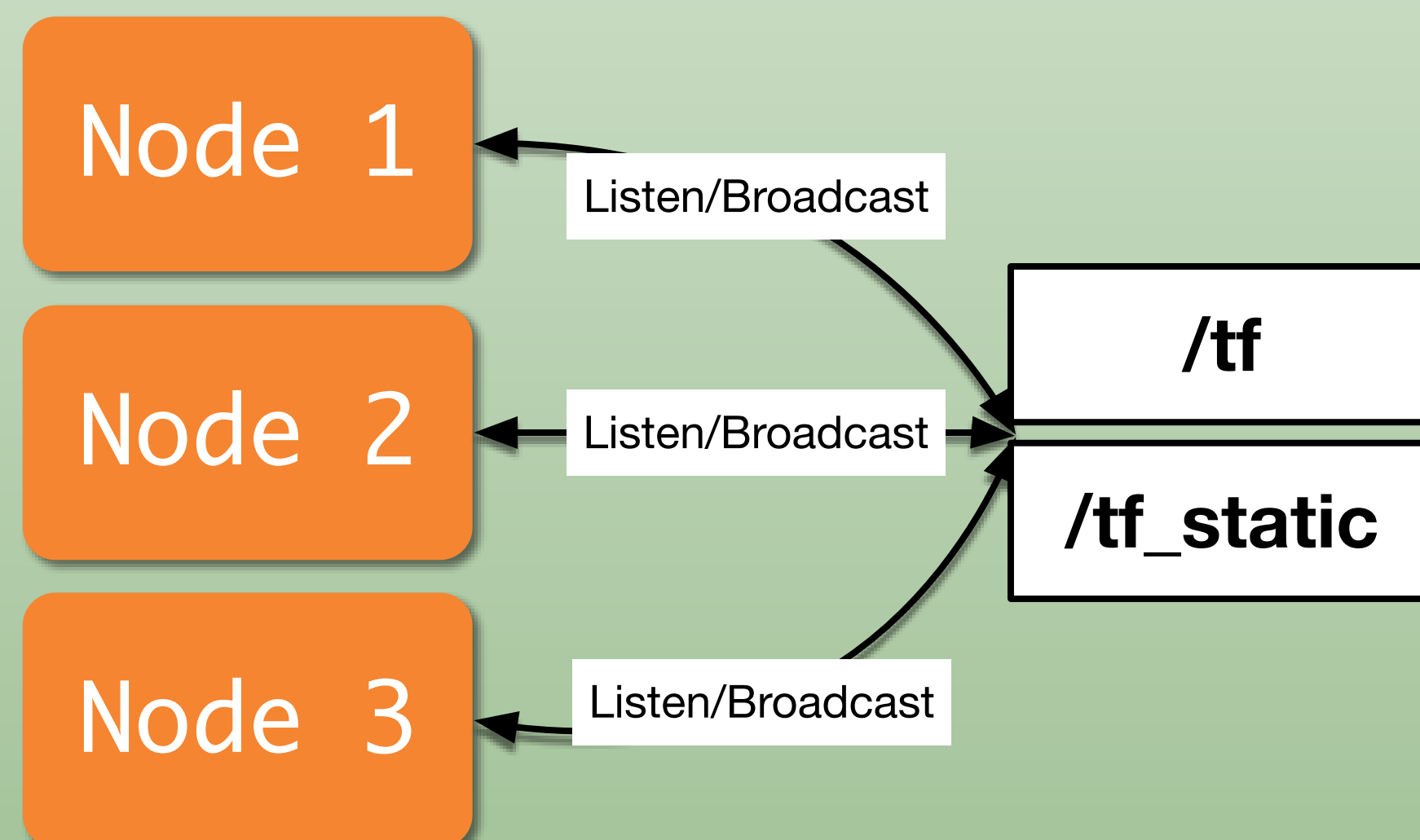
September 2026



- ▶ TF2 Transformation System
- ▶ rqt Graphical User Interface
- ▶ Markers
- ▶ Robot models (URDF)
- ▶ Simulation descriptions (SDF, USD)



- ▶ Tool for keeping track of **coordinate frames over time**
- ▶ Maintains relationship between coordinate frames in a **tree structure** buffered in time
- ▶ Allows points, vectors, poses, etc. to be transformed between coordinate frames at a desired time
- ▶ Transform information is exchanged using: **/tf** and **/tf_static**



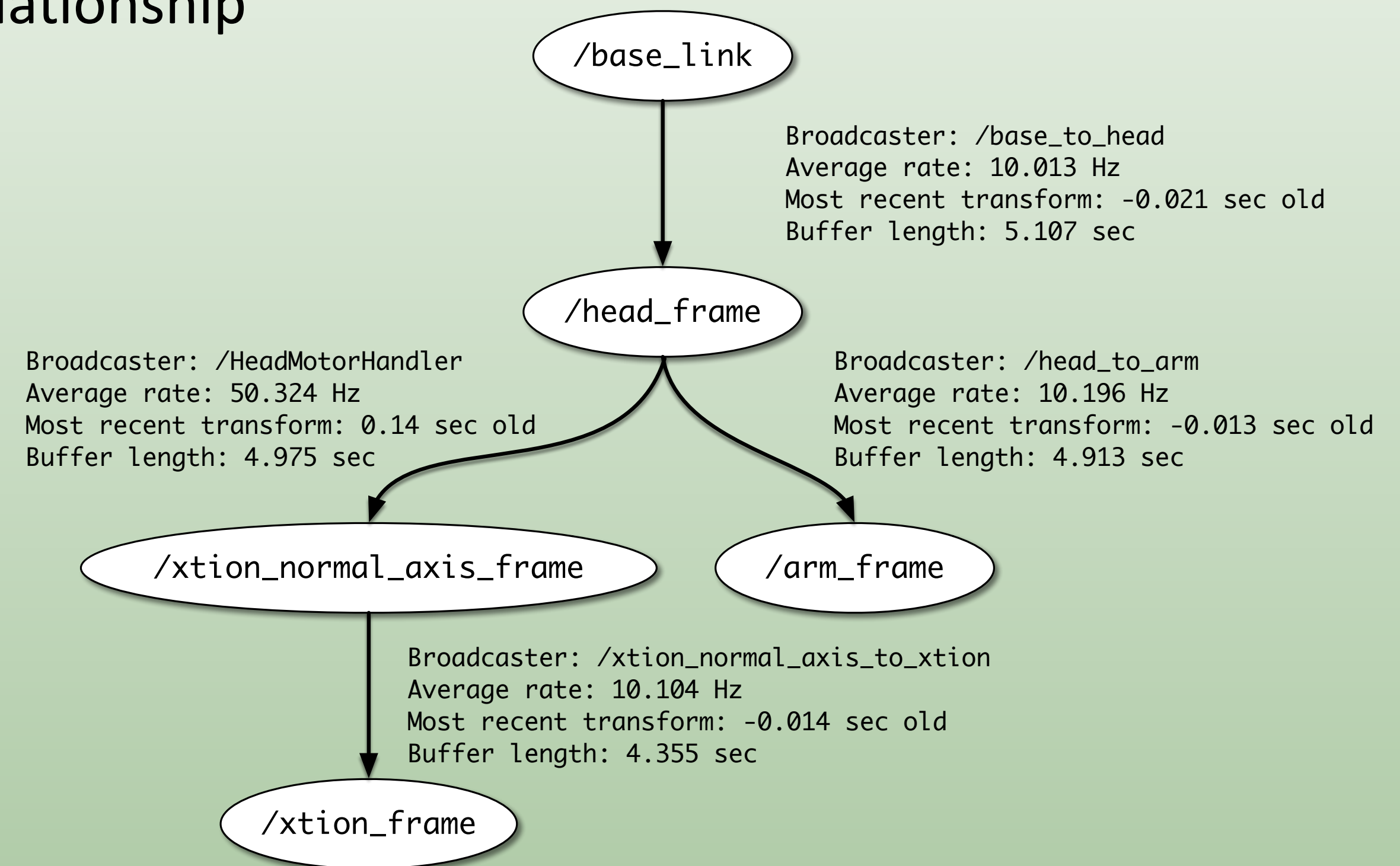
Details at: [TF2](#)

TF2 TRANSFORMATION SYSTEM — TRANSFORM TREE

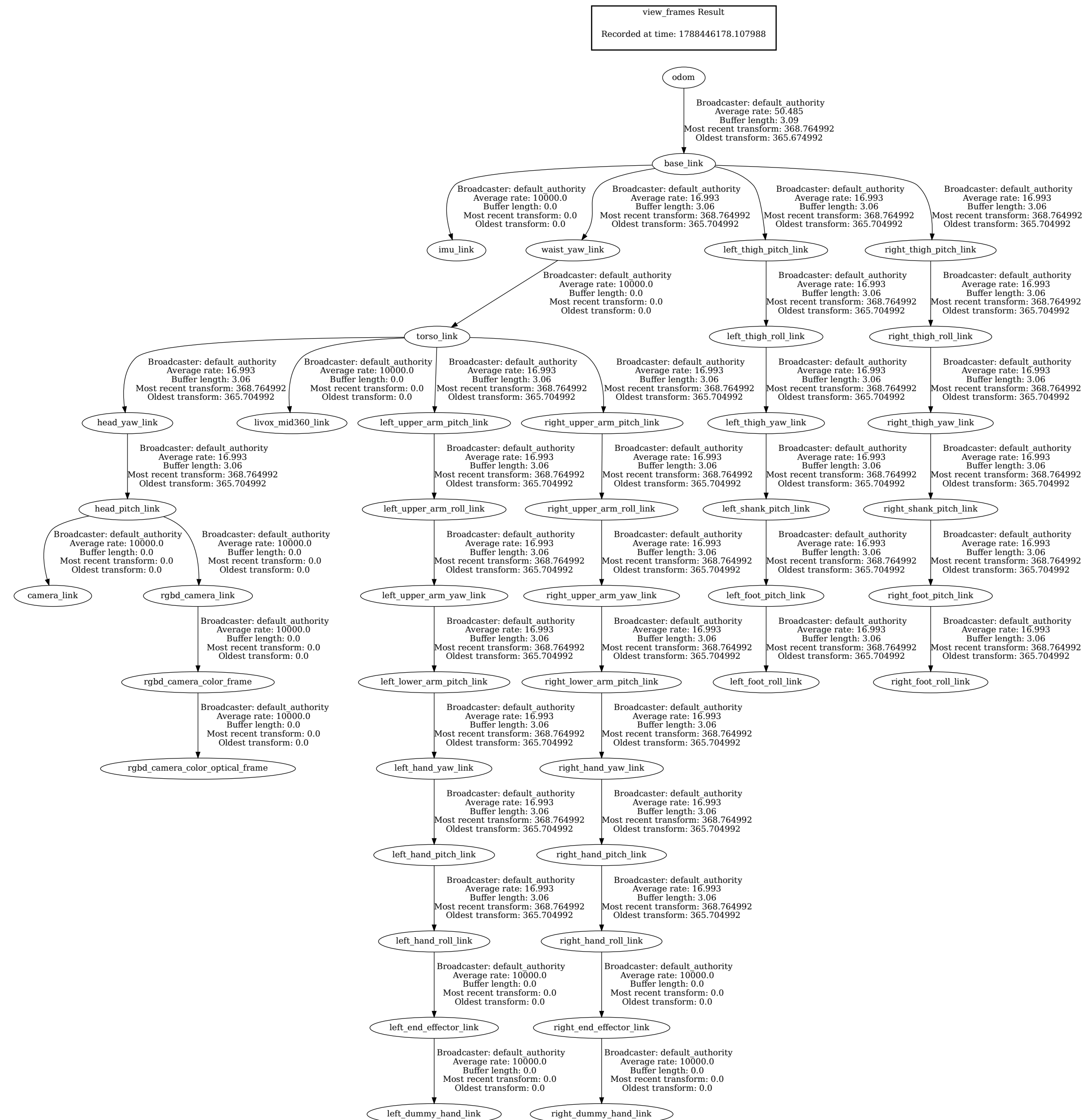
- ▶ TF2 listeners use a **buffer** to store received transforms
- ▶ Query the buffer for transforms between specific coordinate frames
- ▶ Transforms form a **tree of coordinate frames**
- ▶ Each transform defines a **parent frame** → **child frame** relationship

tf2_msgs/TFMessage.msg

```
tf2_msgs/TFMessage
├── geometry_msgs/TransformStamped[] transforms
│   ├── std_msgs/Header header
│   │   ├── builtin_interfaces/Time stamp
│   │   └── string frame_id
│   ├── string child_frame_id
│   └── geometry_msgs/Transform transform
│       ├── geometry_msgs/Vector3 translation
│       └── geometry_msgs/Quaternion rotation
```



TF2 TRANSFORMATION SYSTEM — TRANSFORM TREE



TF2 TRANSFORMATION SYSTEM — TRANSFORM TREE

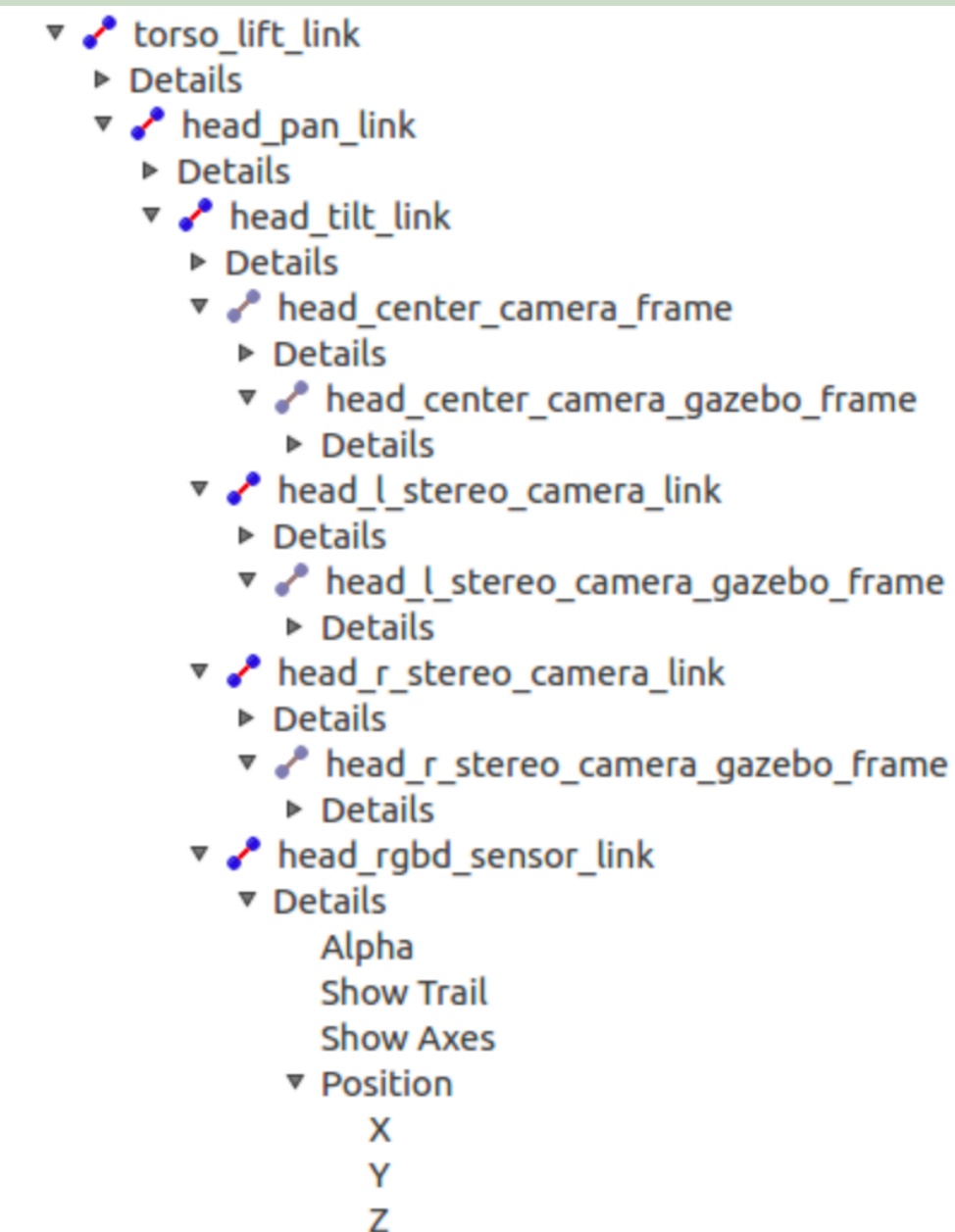
- ▶ TF2 listeners use a **buffer** to store received transforms
- ▶ Query the buffer for transforms between specific coordinate frames
- ▶ Transforms form a **tree of coordinate frames**
- ▶ Each transform defines a **parent frame** → **child** frame relationship

tf2_msgs/TFMessage

```
tf2_msgs/TFMessage
├── geometry_msgs/TransformStamped[] transforms
│   ├── std_msgs/Header header
│   │   ├── builtin_interfaces/Time stamp
│   │   └── string frame_id
│   ├── string child_frame_id
│   └── geometry_msgs/Transform transform
│       ├── geometry_msgs/Vector3 translation
│       └── geometry_msgs/Quaternion rotation
```

Details at: https://docs.ros.org/en/humble/p/tf2_msgs/msg/TFMessage.html

Partial link tree from HSRB, torso



TF2 TRANSFORMATION SYSTEM — TOOLS

Terminal

Get information about the current transform tree

```
~$ros2 run tf2_ros tf2_monitor
```

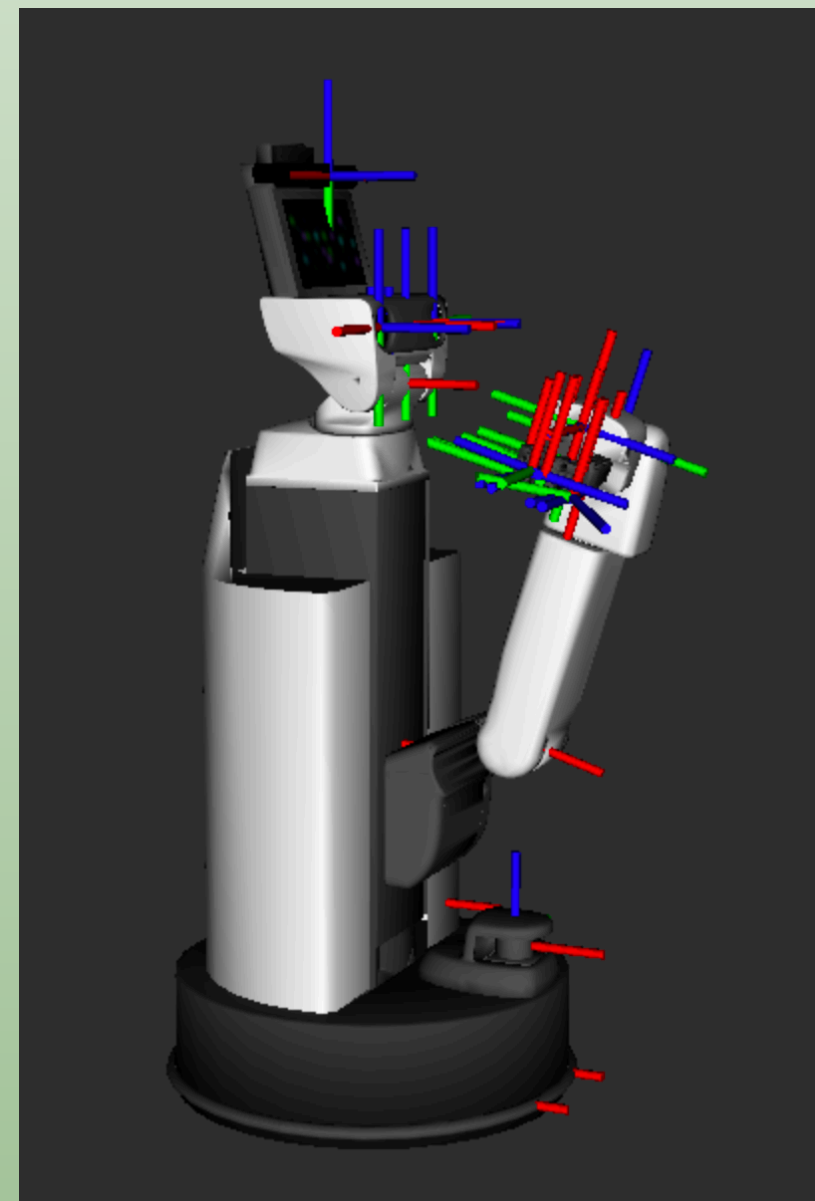
Get information about the transform between two frames

```
~$ros2 run tf2_ros tf2_echo source_frame target_frame
```

RViz2

3D visualization of the transforms

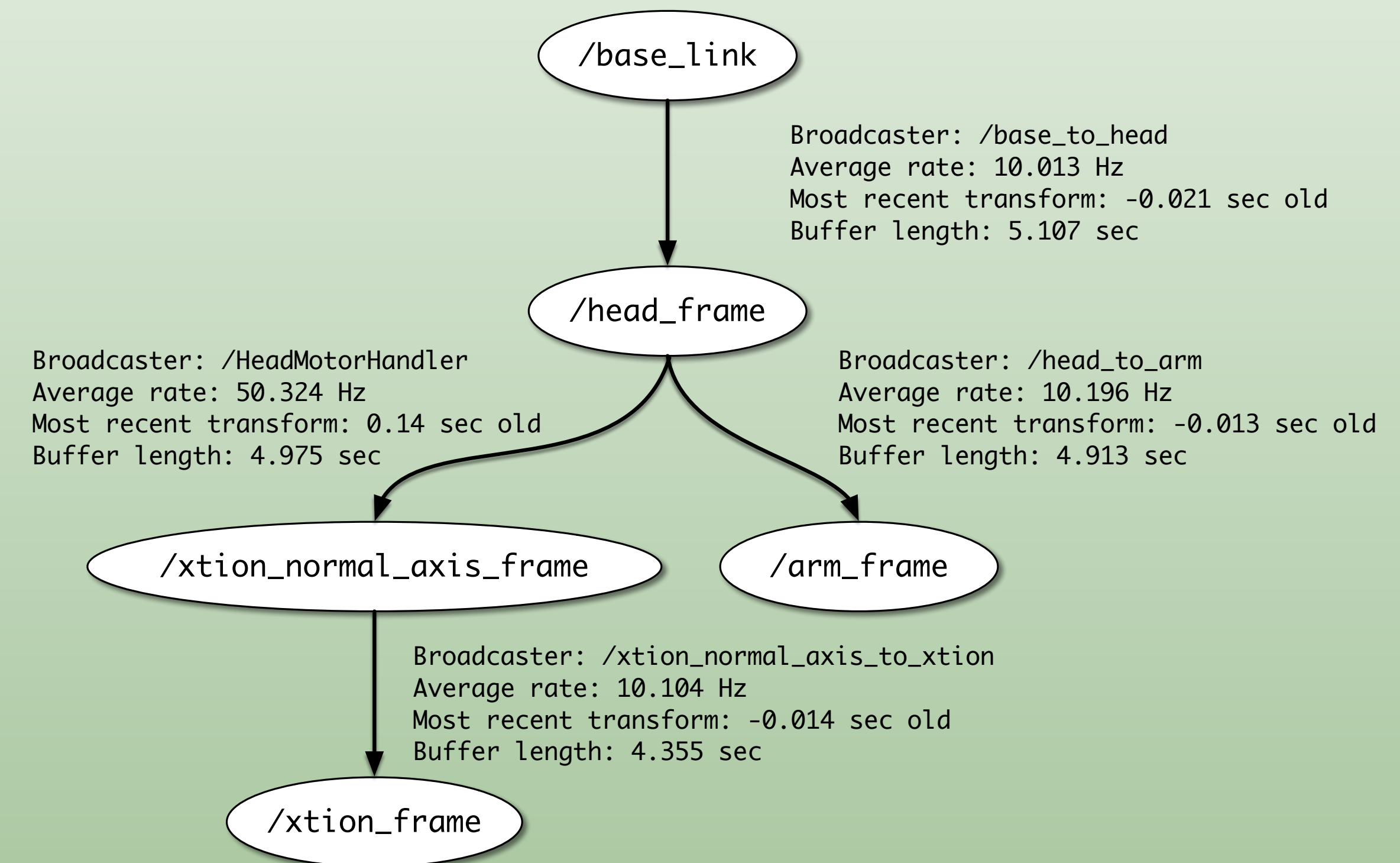
```
~$rviz2
```



View frames

Visual graph of the transform tree

```
~$ros2 run tf2_tools view_frames
```



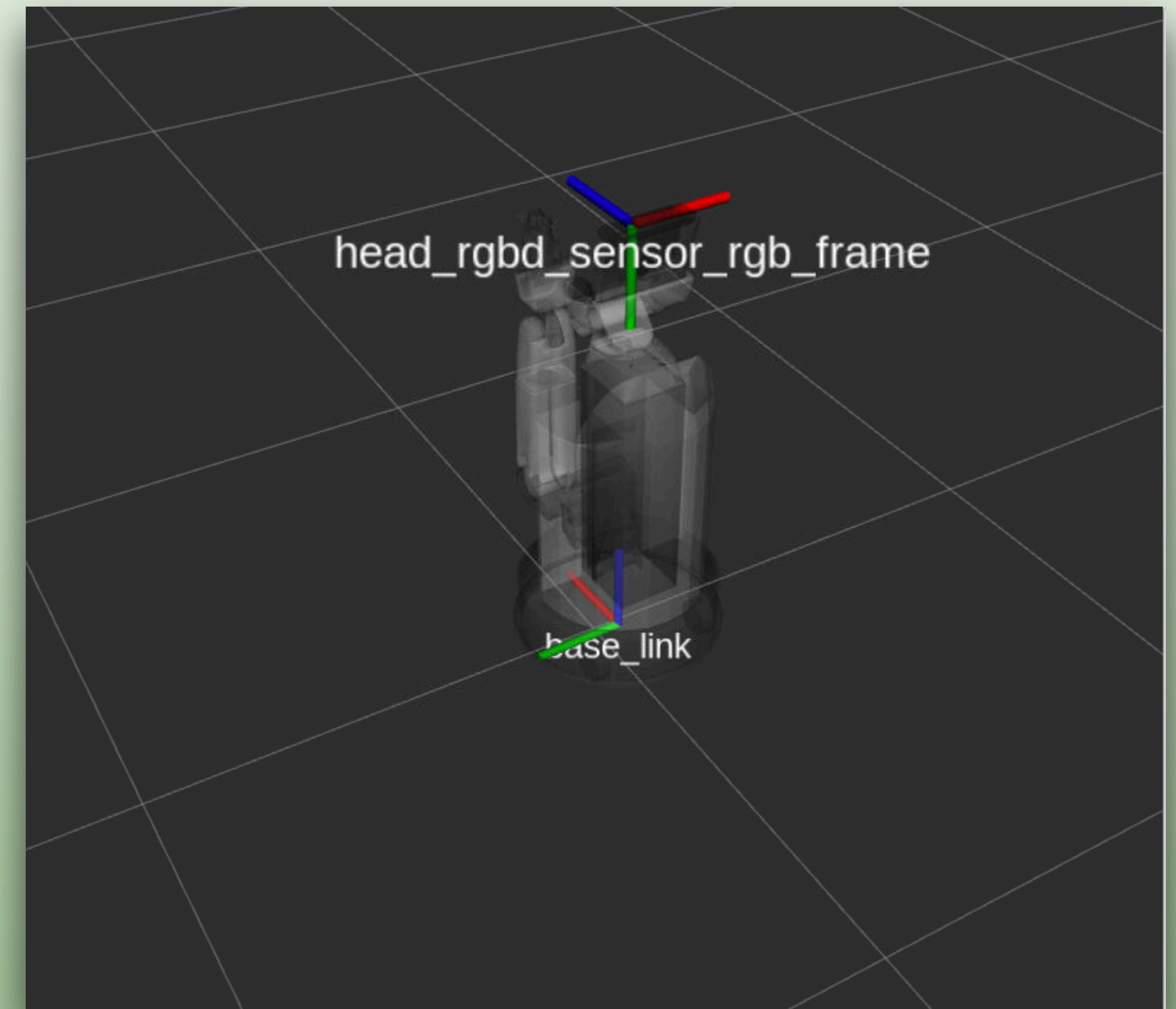
TF TRANSFORMATION SYSTEM - TOOLS

Terminal

Example

```
~$ros2 run tf2_ros tf2_echo base_link head_rgbd_sensor_rgb_frame
```

```
At time 489.133
- Translation: [-0.060, 0.022, 0.967]
- Rotation: in Quaternion [0.500, -0.500, 0.500, -0.500]
           in RPY (radian) [-1.571, -0.000, -1.571]
           in RPY (degree) [-90.000, -0.000, -90.000]
At time 490.133
- Translation: [-0.060, 0.022, 0.967]
- Rotation: in Quaternion [0.500, -0.500, 0.500, -0.500]
           in RPY (radian) [-1.571, -0.000, -1.571]
           in RPY (degree) [-90.000, -0.000, -90.000]
At time 491.133
- Translation: [-0.060, 0.022, 0.967]
- Rotation: in Quaternion [0.500, -0.500, 0.500, -0.500]
           in RPY (radian) [-1.571, -0.000, -1.571]
           in RPY (degree) [-90.000, -0.000, -90.000]
At time 492.133
- Translation: [-0.060, 0.022, 0.967]
- Rotation: in Quaternion [0.500, -0.500, 0.500, -0.500]
           in RPY (radian) [-1.571, -0.000, -1.571]
           in RPY (degree) [-90.000, -0.000, -90.000]
```



TF2 TRANSFORMATION SYSTEM — RVIZ2 TF DISPLAY

The screenshot shows the RViz2 TF Display window. The left panel displays the 'Displays' tree with the following structure:

- Displays
 - Description Source: /robot_description
 - Description Topic
 - TF Prefix
 - Links
 - Link Tree Style: Links in Alphabetic Order
 - Expand Link Details: ☒
 - All Links Enabled: ☒
 - base_link
 - Alpha: 1
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.22826; 0.018152; 0.80693
 - Orientation: 0.00036453; -0.017254; 0.020017; 0.99965
 - camera_link
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.13632; 0.021813; 1.4443
 - Orientation: 0.00036439; -0.017255; 0.020026; 0.99965
 - head_pitch_link
 - Alpha: 1
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.24573; 0.017431; 1.3424
 - Orientation: 0.00036439; -0.017255; 0.020026; 0.99965
 - head_yaw_link
 - Alpha: 1
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.24421; 0.017494; 1.2982
 - Orientation: 0.00036437; -0.017254; 0.020026; 0.99965
 - imu_link
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.28483; 0.015888; 0.73694
 - Orientation: 0.00036453; -0.017254; 0.020017; 0.99965
 - left_dummy_hand_link
 - Alpha: 1
 - Show Trail: ☐
 - Show Axes: ☐
 - Position: -0.13236; 0.2522; 0.69148
 - Orientation: 0.00428; -0.21168; 0.01957; 0.97713
 - left_end_effector_link

The right panel shows a 3D visualization of the robot arm model, with links colored in blue, red, and green. The bottom status bar displays the following information:

- Time
 - ROS Time: 175.86
 - ROS Elapsed: 175.86
 - Wall Time: 1788875753.38
 - Wall Elapsed: 395.14
- Experimental: ☐
- Reset:
- Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
- 31 fps

TF2TRANSFORMATION SYSTEM — TRANSFORM LISTENER C++ API

```
#include "rclcpp/rclcpp.hpp"
#include "tf2_ros/transform_listener.h"
#include "tf2_ros/buffer.h"

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);

    auto node = rclcpp::Node::make_shared("tf2_listener");

    tf2_ros::Buffer tf_buffer(node->get_clock());
    tf2_ros::TransformListener tf_listener(tf_buffer);

    rclcpp::Rate rate(10.0);

    while (rclcpp::ok())
    {
        try {
            auto transform = tf_buffer.lookupTransform(
                "base_link", "camera_link",
                tf2::TimePointZero);

            // use transform
        }
        catch (const tf2::TransformException & ex) {
            RCLCPP_WARN(node->get_logger(), "%s", ex.what());
        }

        rclcpp::spin_some(node);
        rate.sleep();
    }

    rclcpp::shutdown();
    return 0;
}
```

- Create a TF2 buffer and listener

```
tf2_ros::Buffer tf_buffer(node->get_clock());
tf2_ros::TransformListener tf_listener(tf_buffer);
```

- Beware of scope!

- Lookup transformations use this:

```
geometry_msgs::msg::TransformStamped transform =
    tf_buffer.lookupTransform(
        target_frame, source_frame, tf2::TimePointZero);
```

- For time: use **tf2::TimePointZero** to get latest available transform

Details at:

[http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20\(C++\)](http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20(C++))

TF2TRANSFORMATION SYSTEM — TRANSFORM LISTENER PYTHON API

```
import rclpy
from rclpy.node import Node
from tf2_ros import Buffer, TransformListener

rclpy.init()
node = Node('tf2_listener')

tf_buffer = Buffer()
tf_listener = TransformListener(tf_buffer, node)

def get_transform():
    try:
        transform = tf_buffer.lookup_transform(
            'base_link',
            'camera_link',
            rclpy.time.Time())

        node.get_logger().info(
            f'Translation: {transform.transform.translation}')

    except Exception as ex:
        node.get_logger().warning(str(ex))

timer = node.create_timer(1.0, get_transform)

rclpy.spin(node)
rclpy.shutdown()
```

- Create a TF listener to fill a buffer

```
tf_buffer = Buffer()
tf_listener = TransformListener(tf_buffer, node)
```

- Lookup transformations, **rclpy.time.Time()** for latest frame

```
transform = tf_buffer.lookup_transform(
    'base_link', 'camera_link', rclpy.time.Time())
```

Details at:

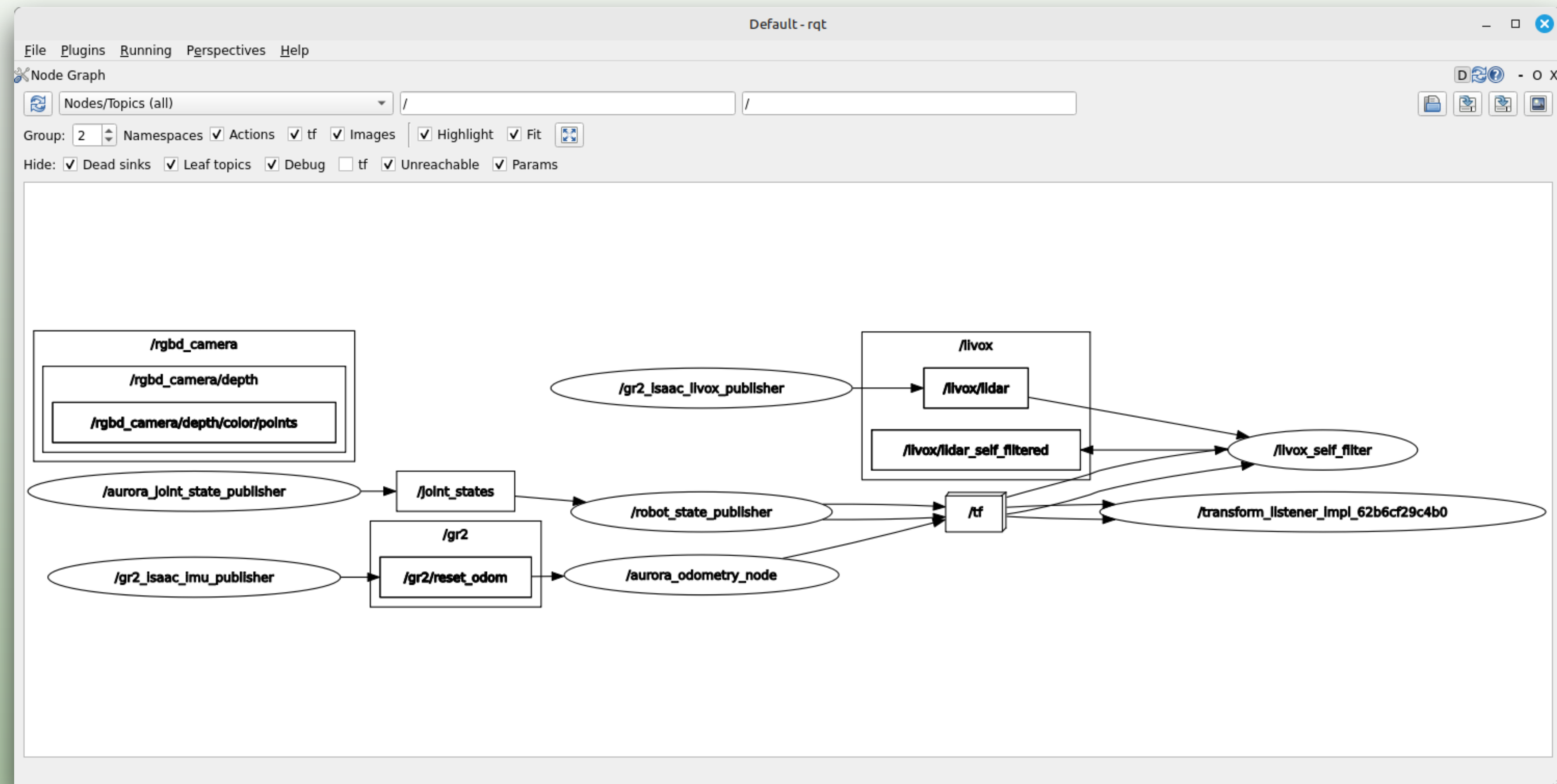
<https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Listener-Py.html>

RQT USER INTERFACE

- ▶ User interface based on Qt
- ▶ Custom interfaces possible
- ▶ Use existing plugins
- ▶ Create your own plugins

Run RQT

```
~$rqt
```



Details at: <http://wiki.ros.org/rqt/Plugins>

RQT PLOT

- ▶ Our GR-2 publishes standard sensor_msgs/msg/JointState on /joint_states
- ▶ rqt_plot can plot an array element directly.

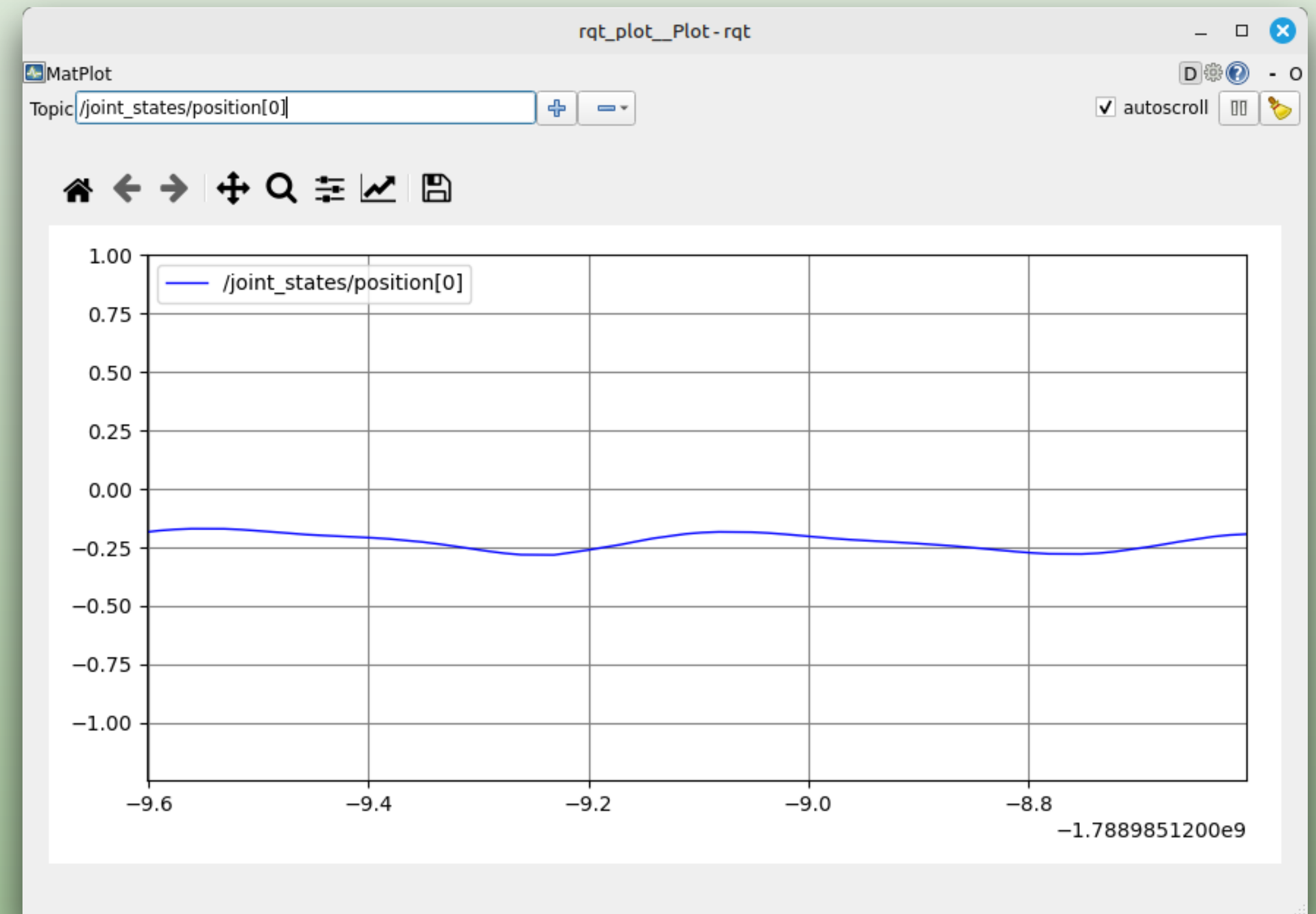
First check with echo

```
~$ros2 topic echo /joint_states --once
```

```
robocanes@rc2:~/rc_workspace$ ros2 topic echo /joint_states --once
header:
  stamp:
    sec: 521
    nanosec: 404988345
  frame_id: ''
name:
- left_hip_pitch_joint
- left_hip_roll_joint
...
position:
- -0.34530240297317505
- -0.05025877431035042
- -0.15378665924072266
```

- ▶ Then we start rqt_plot

```
~$ros2 run rqt_plot rqt_plot
```

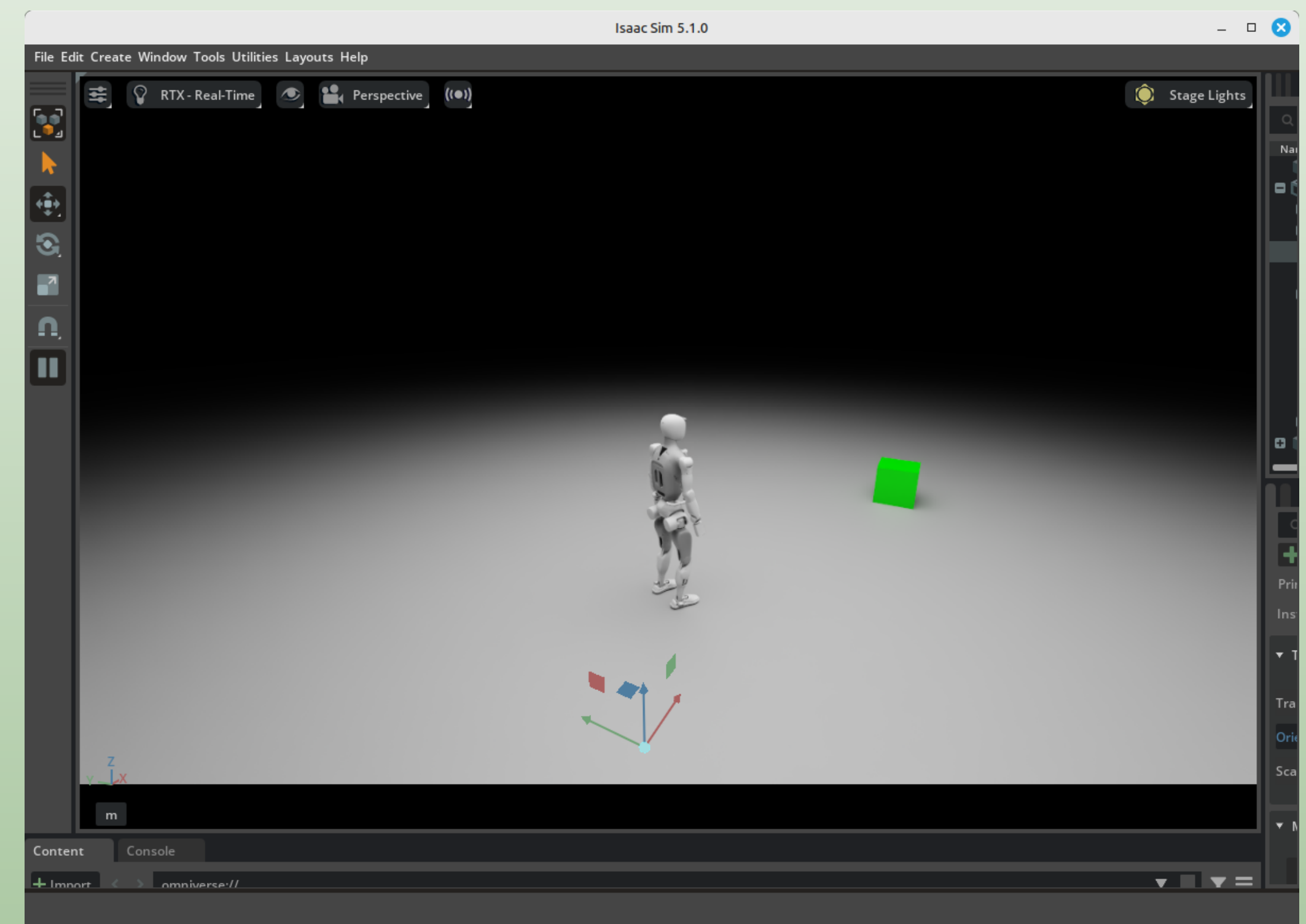
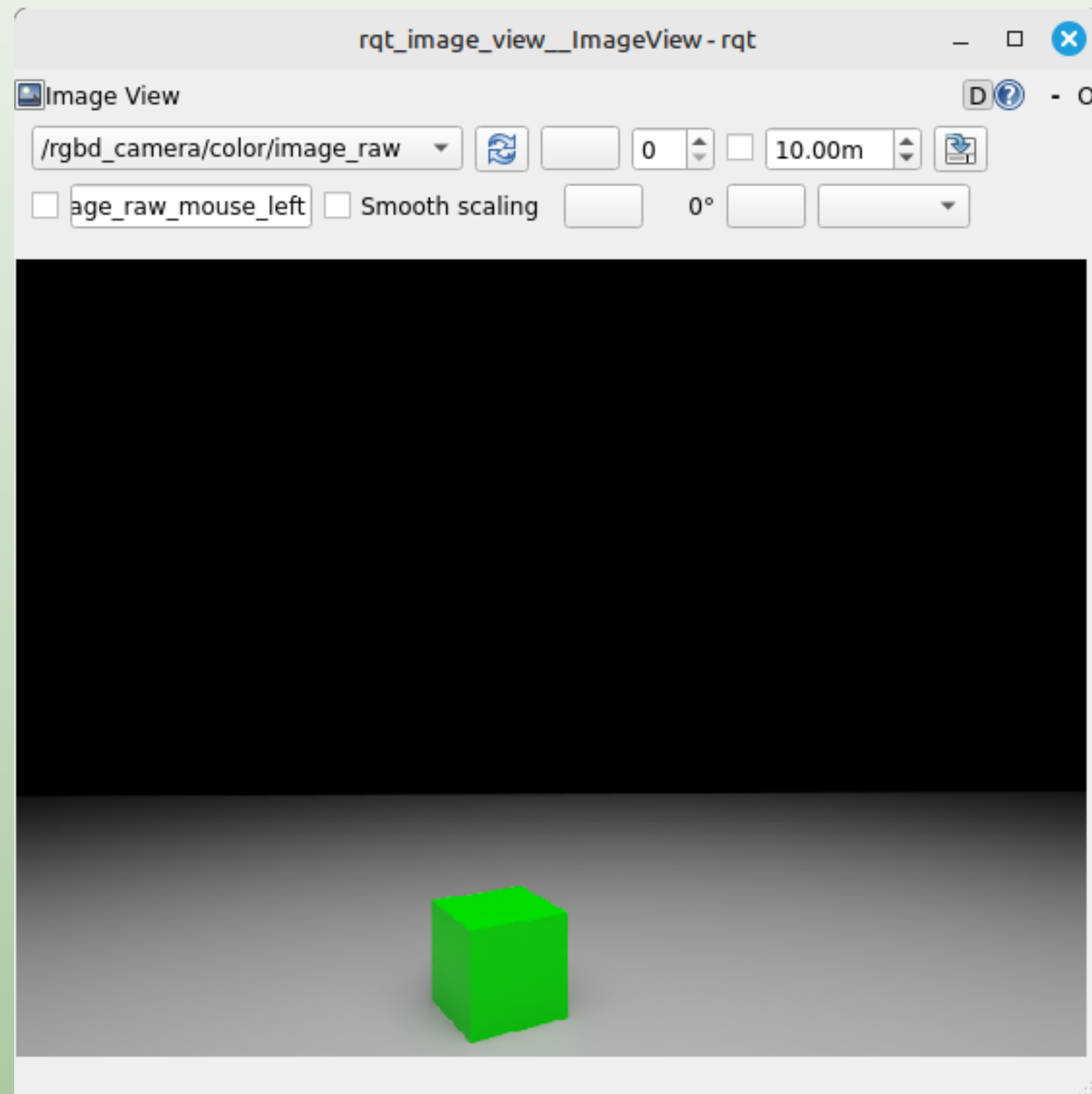


RQT USER INTERFACE IMAGE VIEW

► Visualizing images

Run rqt_image_view

```
~$ /usr/bin/python3 /opt/ros/humble/lib/rqt_image_view/rqt_image_view
```



Details at: http://wiki.ros.org/rqt_image_view

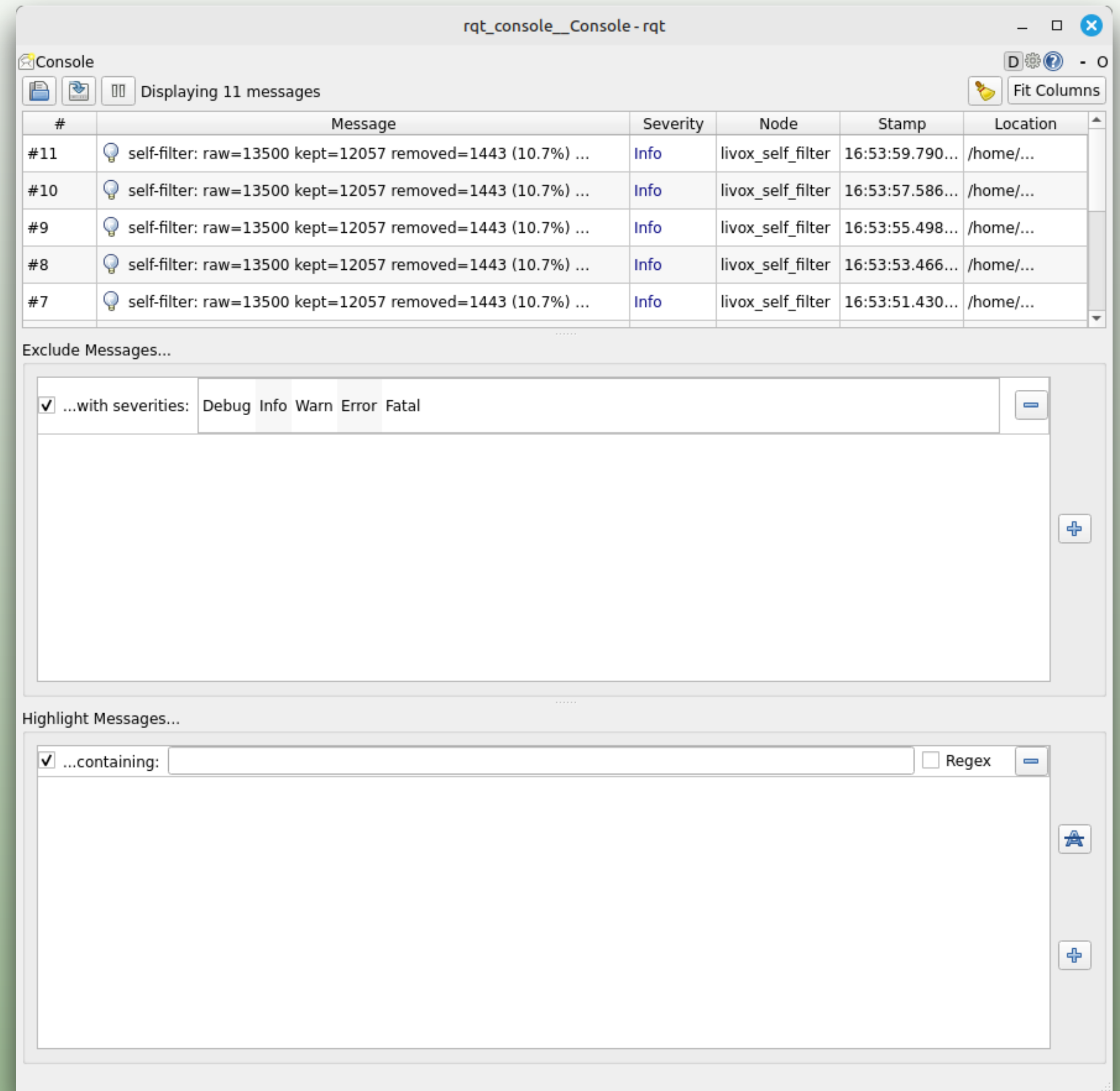
RQT USER INTERFACE — RQT_CONSOLE

- ▶ Displays ROS 2 log messages
- ▶ Filter messages by severity
- ▶ Filter by node/logger
- ▶ Useful for debugging running ROS 2 systems

Run rqt_console

```
~$ros2 run rqt_console rqt_console
```

Details at: https://docs.ros.org/en/humble/p/rqt_console/



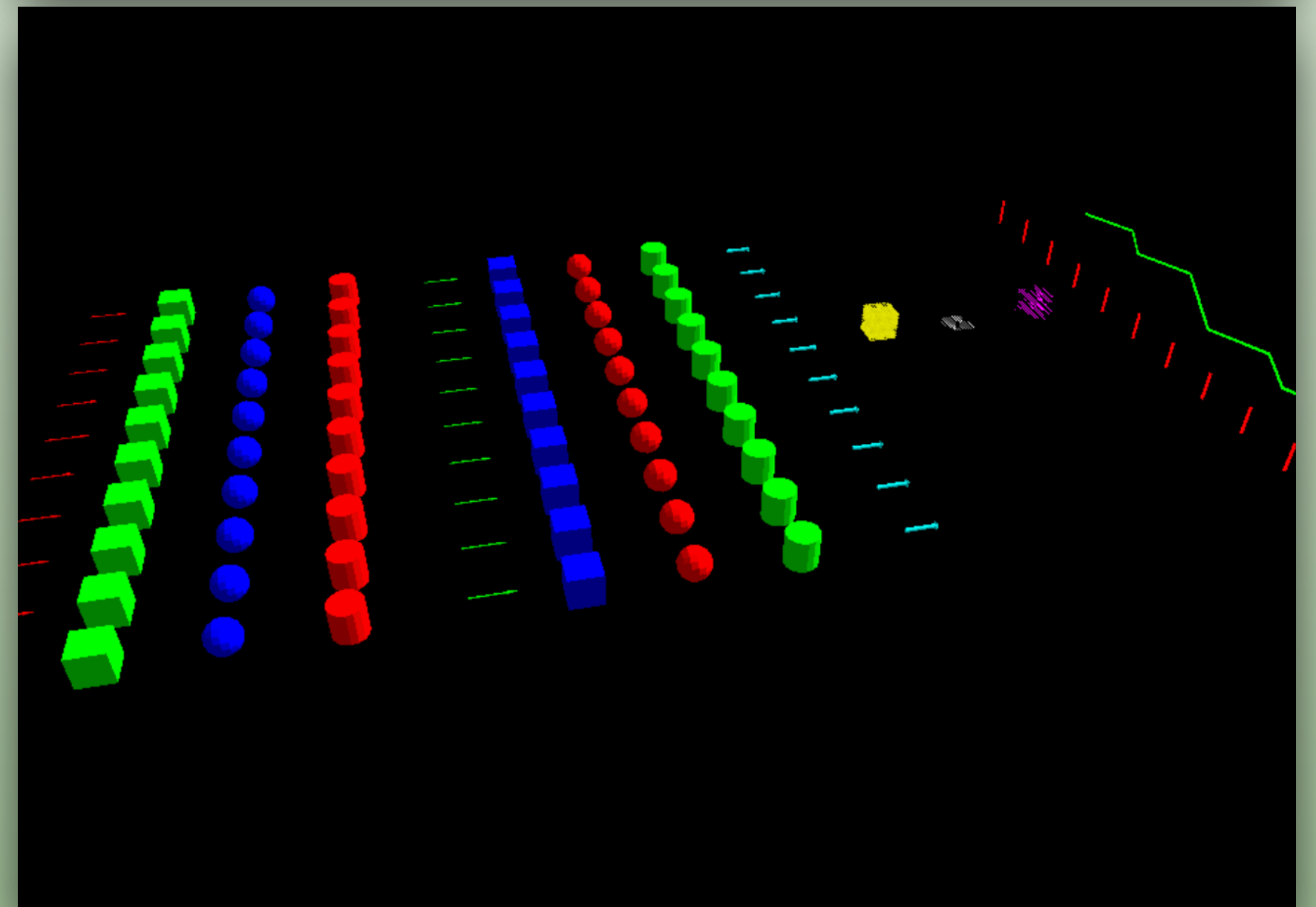
ROS 2 VISUALIZATION MARKERS

- ▶ Markers are messages used to display custom information in **RViz2**
- ▶ Useful for:
 - ▶ points and trajectories
 - ▶ arrows and vectors
 - ▶ bounding boxes
 - ▶ text labels
 - ▶ detected objects
 - ▶ planned paths / goals
- ▶ Published as `visualization_msgs/msg/Marker`
- ▶ Multiple markers can be published as `visualization_msgs/msg/MarkerArray`

Details at: [Markers](#)

Minimal structure

```
Marker
  header.frame_id
  ns
  id
  type
  action
  pose
  scale
  color
```



ROS 2 VISUALIZATION MARKERS

- ▶ RViz2 → Add → Marker → select marker topic
- ▶ Marker pose is expressed relative to `header.frame_id`, so this links directly back to TF2

```
from visualization_msgs.msg import Marker

marker = Marker()

marker.header.frame_id = "base_link"
marker.type = Marker.SPHERE
marker.action = Marker.ADD

marker.pose.position.x = 1.0
marker.pose.orientation.w = 1.0

marker.scale.x = 0.2
marker.scale.y = 0.2
marker.scale.z = 0.2

marker.color.r = 1.0
marker.color.a = 1.0

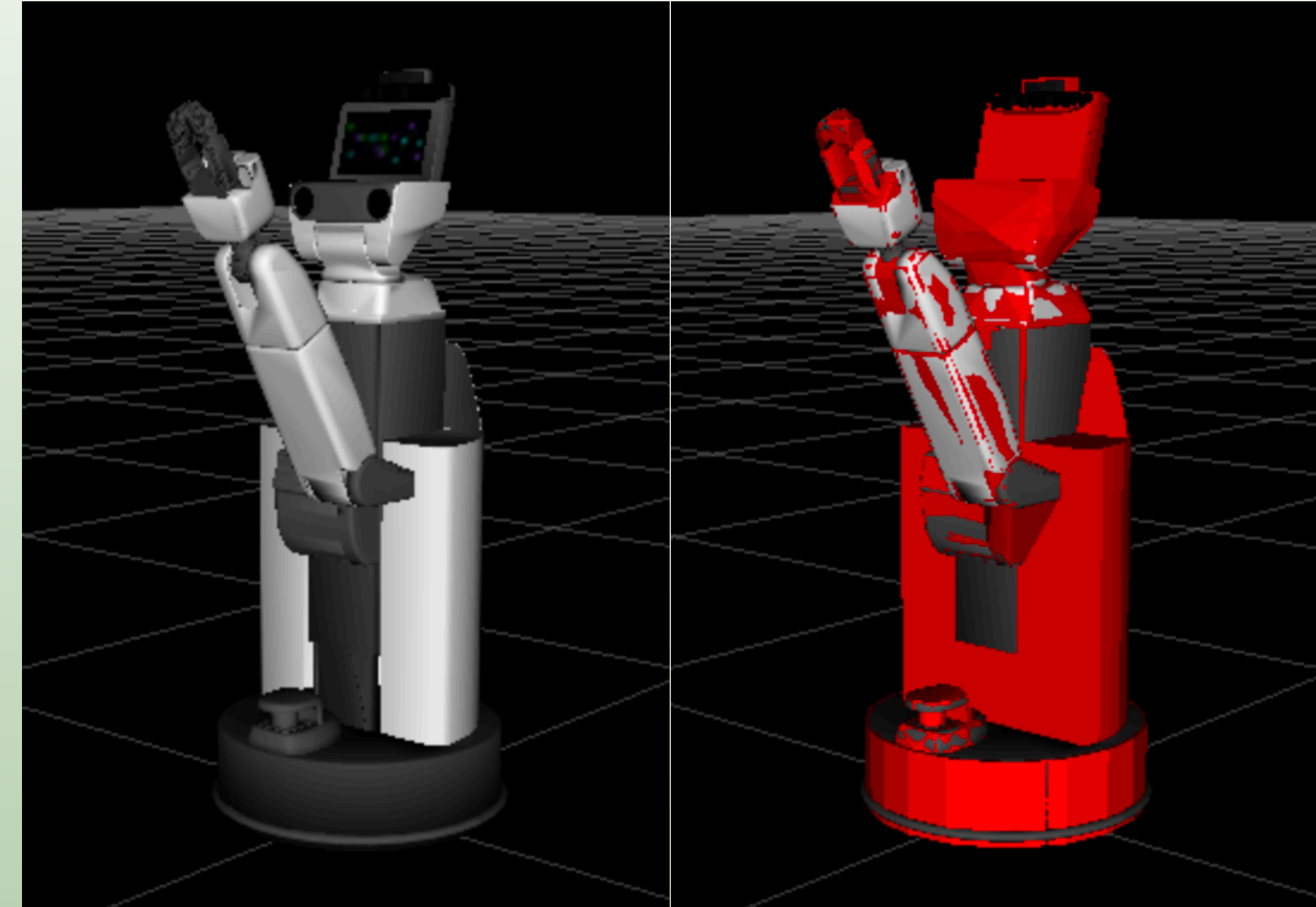
publisher.publish(marker)
```

ROS 2 VISUALIZATION MARKERS

- ▶ Common Marker Types
- ▶ **Example:** visualize a detected obstacle, target location, planned trajectory, or robot goal without modifying the URDF.

Type	Typical use
ARROW	direction/vector
CUBE	bounding box
SPHERE	point/object
CYLINDER	region/object
LINE_STRIP	path/trajectory
LINE_LIST	edges/connections
POINTS	point set
TEXT_VIEW_FACING	labels
MESH_RESOURCE	custom geometry

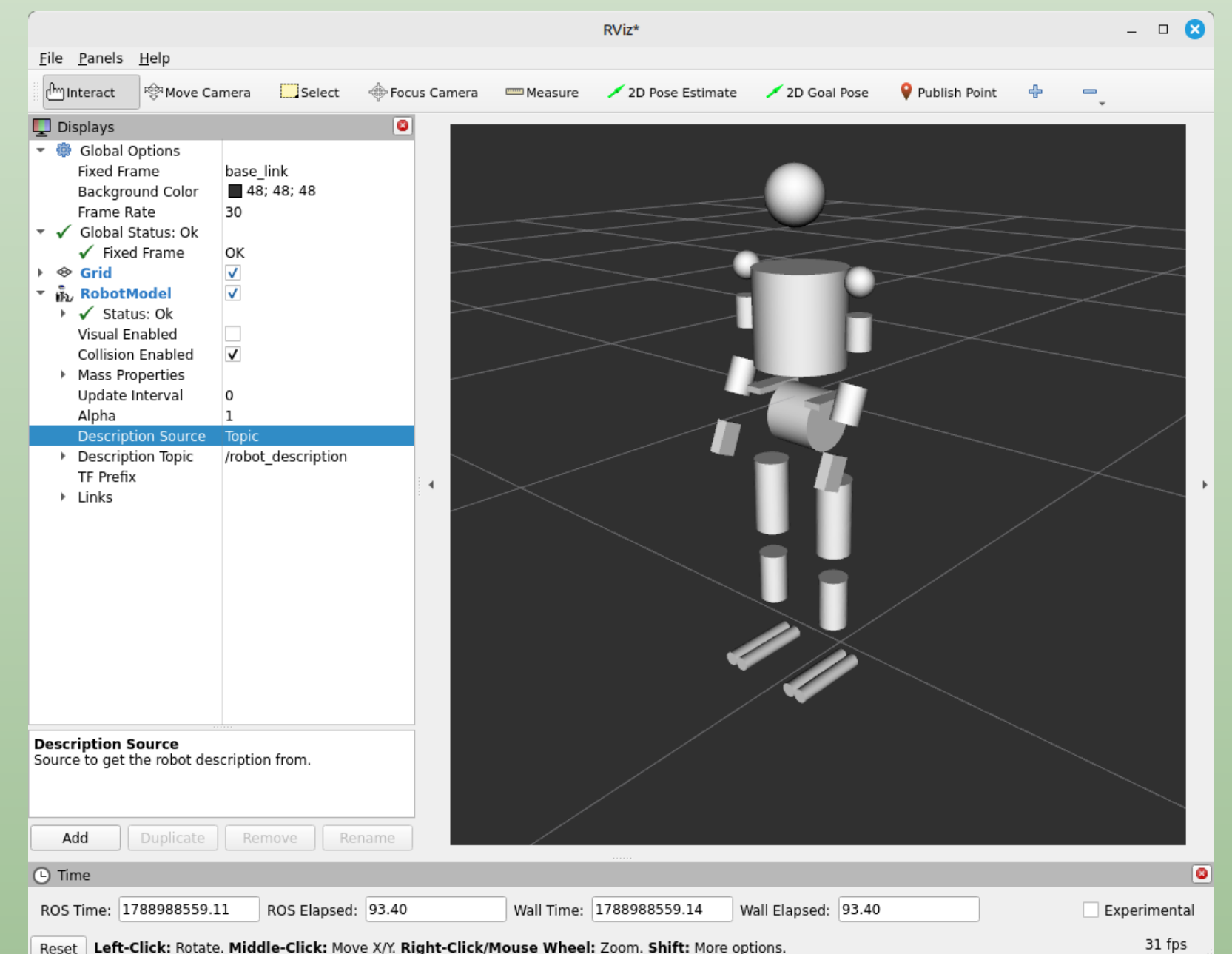
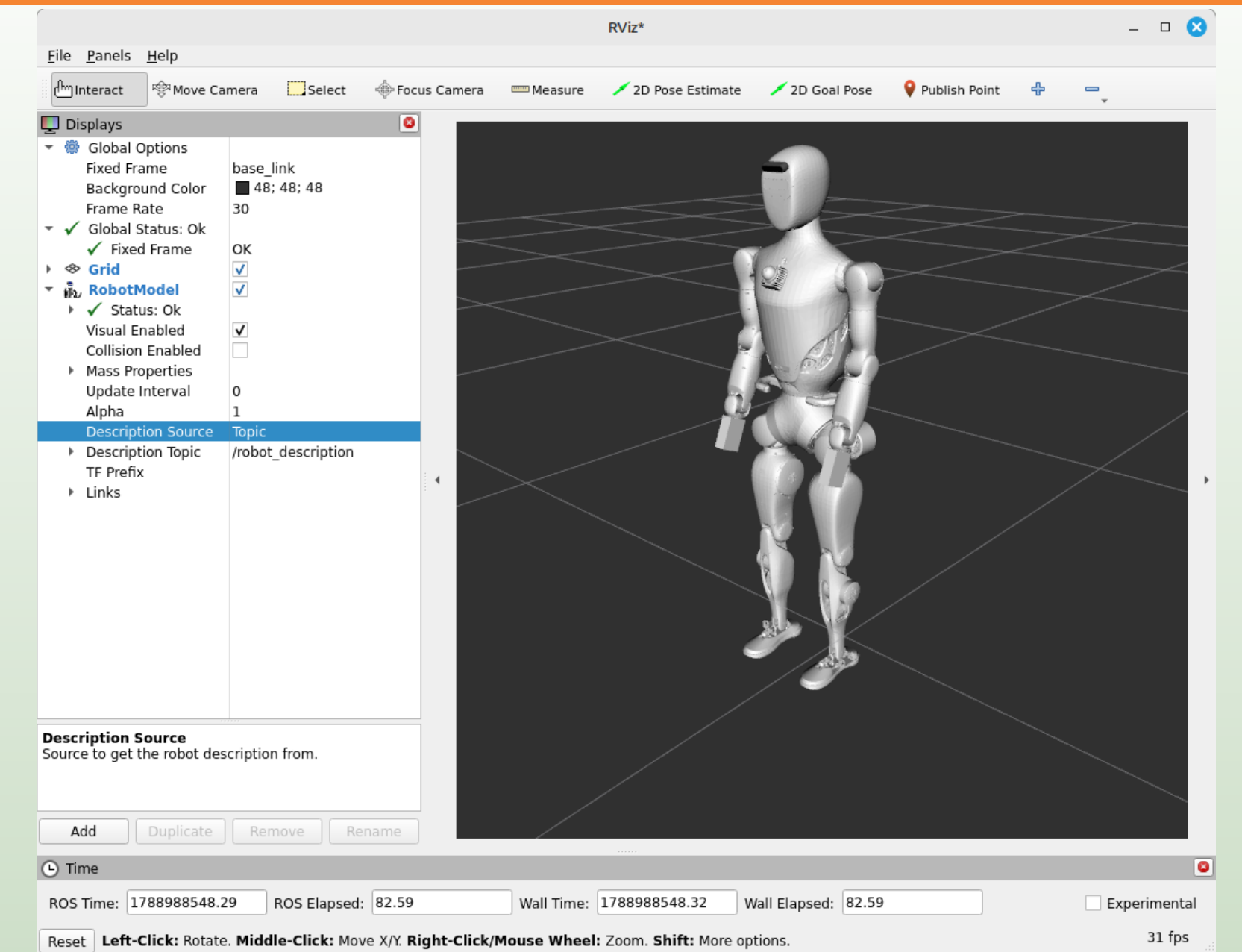
- ▶ **Unified Robot Description Format (URDF)**
- ▶ XML format for describing a robot
 - ▶ **Kinematic structure** — links and joints
 - ▶ **Inertial properties** — mass and inertia
 - ▶ **Visual geometry** — appearance/meshes
 - ▶ **Collision geometry** — simplified geometry for collision checking
- ▶ URDF descriptions can be generated using **Xacro**



ROBOT MODELS - URDF

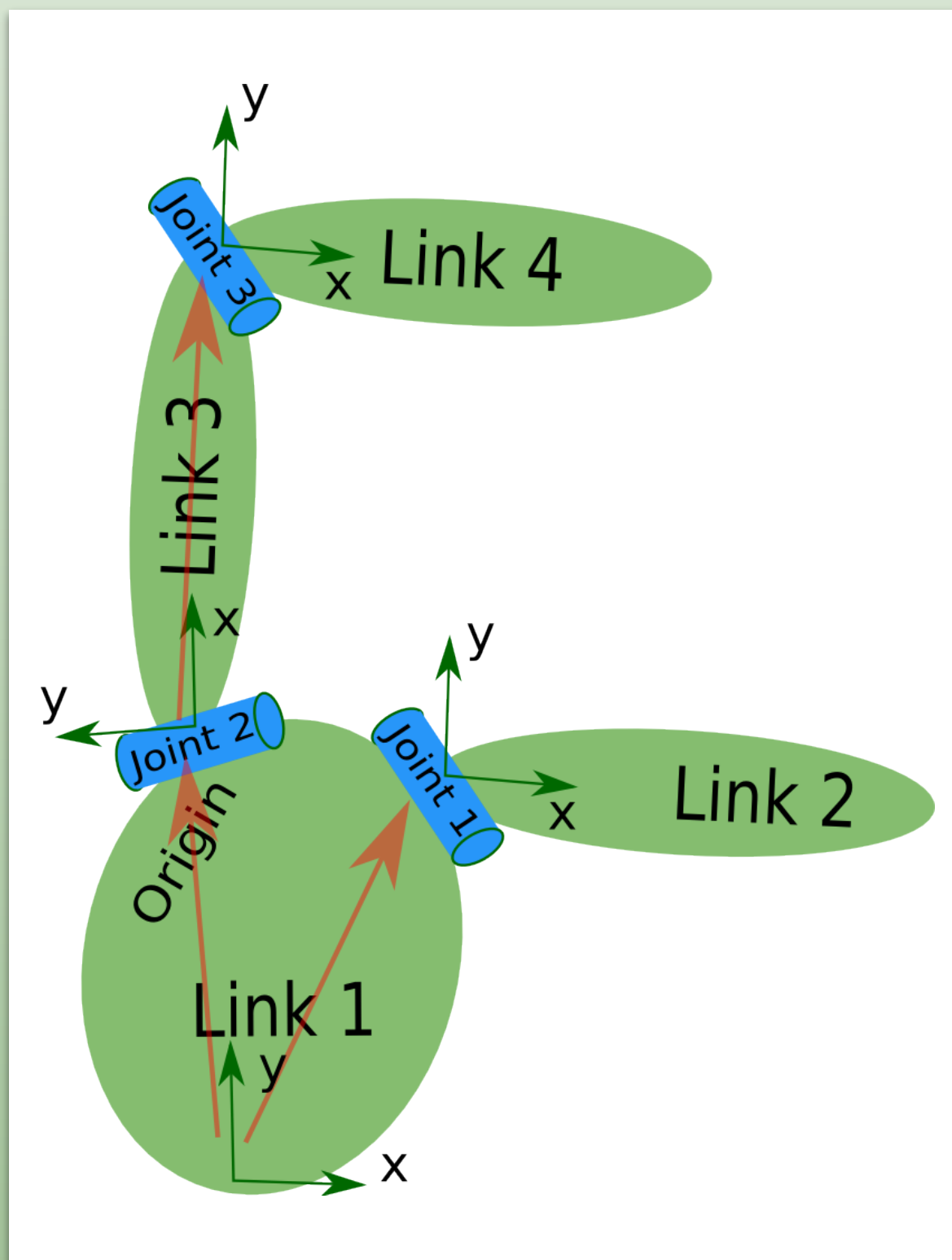
- ▶ **Unified Robot Description Format (URDF)**
- ▶ XML format for describing a robot
 - ▶ **Kinematic structure** — links and joints
 - ▶ **Inertial properties** — mass and inertia
 - ▶ **Visual geometry** — appearance/meshes
 - ▶ **Collision geometry** — simplified geometry for collision checking
- ▶ URDF descriptions can be generated using **Xacro**

Details at:
[URDF](#), [Xacro](#)



ROBOT MODELS - URDF

- ▶ A robot description consists of a set of **link** elements and **joint** elements
- ▶ **Joints connect links** and define their relative motion and transformation
- ▶ Links can contain:
 - ▶ inertial properties
 - ▶ visual geometry
 - ▶ collision geometry



```
<robot name="hsr">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint> .... </joint>
  <joint> .... </joint>
  <joint> .... </joint>
</robot>
```

```
<link name="my_link">
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100"
      iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

ROBOT MODELS — USAGE IN ROS 2

- ▶ The robot description (URDF) is typically provided as the `robot_description` **parameter** of `robot_state_publisher`
- ▶ `robot_state_publisher` uses the URDF and joint states to publish the robot's transforms to **TF2**
- ▶ The robot model can be visualized in **RViz2** using the **RobotModel** display
- ▶ **Xacro** is commonly used to generate the URDF

Example

```
<robot name="my_robot"
  xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:property name="wheel_radius" value="0.1"/>

  <link name="base_link">
    ...
  </link>
</robot>
```

Launch example

```
from launch_ros.actions import Node
import xacro

robot_description = xacro.process_file(
    'my_robot.urdf.xacro').toxml()

robot_state_publisher = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    parameters=[{
        'robot_description': robot_description
    }]
)
```

- ▶ The robot description (URDF) is typically provided as the `robot_description` parameter of `robot_state_publisher`
- ▶ `robot_state_publisher` uses the URDF and joint states to publish the robot's transforms to **TF2**
- ▶ The robot model can be visualized in **RViz2** using the **RobotModel** display
- ▶ **Xacro** is commonly used to generate the URDF

gr2_real.urdf.xacro

```
<?xml version='1.0' encoding='utf-8'?>
<robot name="gr2v3_8_7">
  <link name="base_link">
    <inertial>
      <origin xyz="-0.0030357 6.968E-05 -0.087284" rpy="0 0 0" />
      <mass value="6.6598" />
      <inertia ixx="0.048356" ixy="-1.314E-05" ixz="0.00041517" iyy="0.
030781" iyz="-3.242E-05" izz="0.041191" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://gr2_description/meshes/base_link.STL" />
      </geometry>
      <material name="">
        <color rgba="1 1 1 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0.125 0.01" rpy="0 0 0" />
      <geometry>
        <box size="0.18 0.06 0.02" />
      </geometry>
    </collision>
    <collision>
      <origin xyz="0 -0.125 0.01" rpy="0 0 0" />
      <geometry>
        <box size="0.18 0.06 0.02" />
      </geometry>
    </collision>
    <collision>
      <origin xyz="-0.03 0 -0.08" rpy="1.57 0 0" />
      <geometry>
        <cylinder radius="0.1" length="0.17" />
      </geometry>
    </collision>
  </link>

```

rsp_real.launch.py

```
import os

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.conditions import IfCondition
from launch.substitutions import Command, LaunchConfiguration
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    pkg_share = get_package_share_directory("gr2_description")
    xacro_file = os.path.join(pkg_share, "urdf", "gr2_real.urdf.xacro")

    use_gui = LaunchConfiguration("use_gui")

    robot_description = {
        "robot_description": Command([
            "xacro ",
            xacro_file,
        ])
    }

    return LaunchDescription([
        DeclareLaunchArgument(
            "use_gui",
            default_value="false",
            description="Start joint_state_publisher_gui for manual joint testing.",
        ),
        Node(
            package="robot_state_publisher",
            executable="robot_state_publisher",

```

UNIVERSAL SCENE DESCRIPTION (USD)

- ▶ OpenUSD is used to describe **complex 3D scenes and simulation environments**
- ▶ In ROS 2 workflows, USD is especially relevant when using **NVIDIA Isaac Sim / Isaac Lab**
- ▶ A USD scene can contain:
 - ▶ environments and world geometry
 - ▶ static and dynamic objects
 - ▶ robots
 - ▶ sensors
 - ▶ materials, lighting, and physics properties
- ▶ Robot models described in URDF can be **imported into Isaac Sim and converted to USD**
- ▶ ROS 2 nodes can interface with the simulated robot and sensors through the **Isaac ROS / ROS 2 bridge**
- ▶ USD is the native scene description format used by Isaac Sim

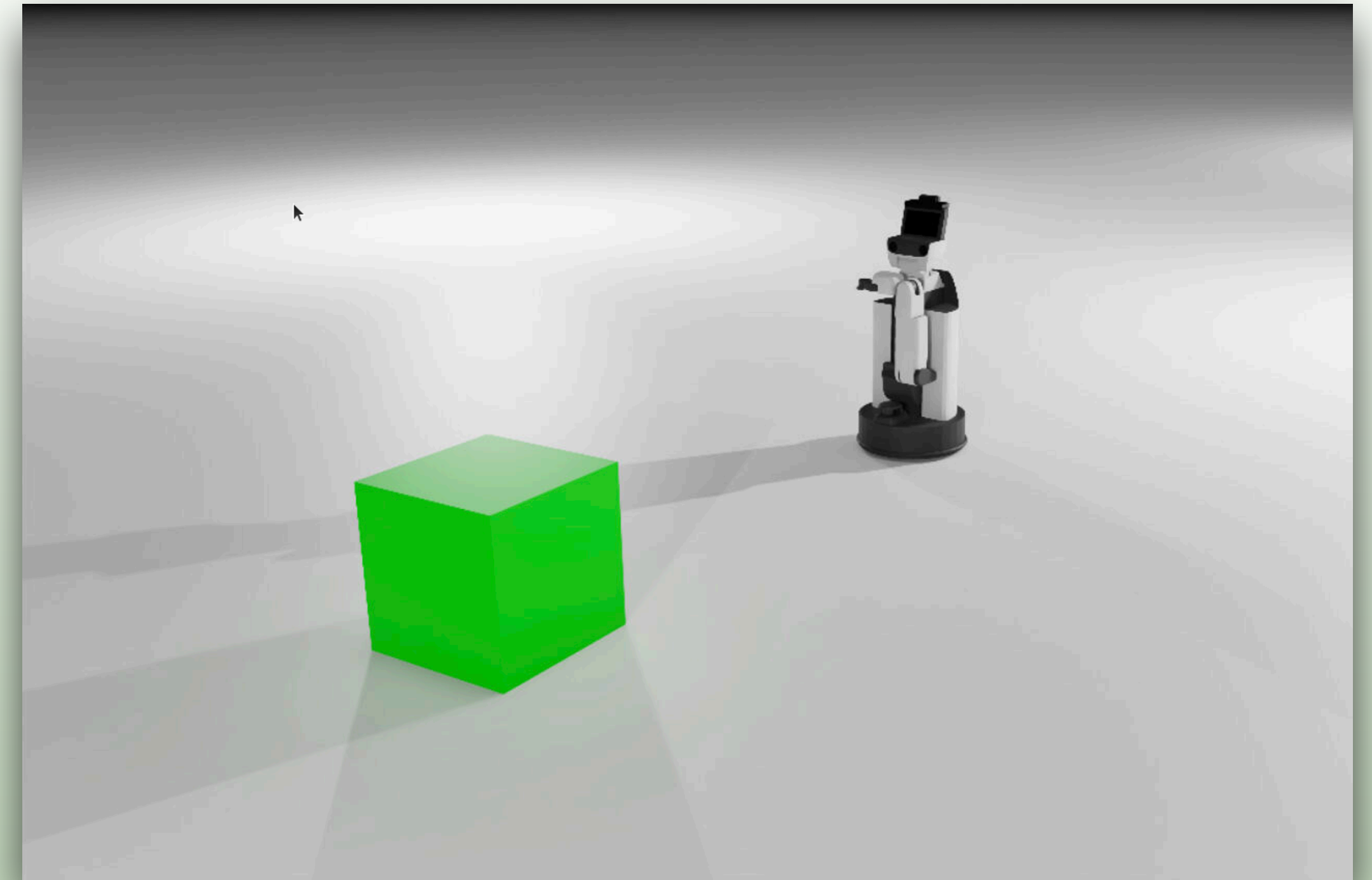
Details at:
<https://openusd.org/release/index.html>



```
#usda 1.0
(
  customLayerData = {
    dictionary cameraSettings = {
      dictionary Front = {
        double3 position = (0, 0, 500)
        double radius = 5
      }
      dictionary Perspective = {
        double3 position = (9.922647693462752, 1.695244378764257, -
        double3 target = (7.270295042185742, 0.9129468292180112, -1
      }
      dictionary Right = {
        double3 position = (-500, 0, 0)
        double radius = 5
      }
      dictionary Top = {
```

UNIVERSAL SCENE DESCRIPTION (USD)

- ▶ Encoding scalable, hierarchically organized, static and time-sampled data
- ▶ Rich, extensible scene-description framework
- ▶ Environments (geometry, lighting, physics, etc.)
- ▶ Objects (static and dynamic)
- ▶ Sensors
- ▶ Robots
- ▶ USD is the native scene-description format used by Isaac Sim
- ▶ Importers/converters to USD are available



```
from pxr import Usd, UsdGeom

stage = Usd.Stage.CreateNew('/path/to/HelloWorld.usda')
xformPrim = UsdGeom.Xform.Define(stage, '/hello')
spherePrim = UsdGeom.Sphere.Define(stage, '/hello/world')
# generic_spherePrim = stage.DefinePrim('/hello/world_generic', 'Sphere')
stage.GetRootLayer().Save()
```

```
#usda 1.0

def Xform "hello"
{
    def Sphere "world"
    {
    }
}
```

```
#usda 1.0
(
    defaultPrim = "bookshelf"
    upAxis = "Y"
)

def Xform "bookshelf"
{
    token ui:displayGroup = "Material Graphs"
    token ui:displayName = "bookshelf"
    int ui:order = 1024
    float3 xformOp:rotateXYZ = (0, -0, 0)
    float3 xformOp:scale = (1, 1, 1)
    double3 xformOp:translate = (0, 0, 0)
    uniform token[] xformOpOrder = ["xformOp:translate", "xformOp:rotateXYZ", "xformOp:scale"]

    def Xform "bookshelf" (
        apiSchemas = ["SemanticsAPI:Semantics_rm0w", "SemanticsAPI:Semantics_zY6Z",
            "SemanticsAPI:Semantics_Ql3t", "SemanticsAPI:QWQQ", "SemanticsAPI:QWQL",
            "SemanticsAPI:QWQC", "PhysicsRigidBodyAPI", "PhysicsMassAPI"]
    )
    {
    }
}
```

Details at:
<https://openusd.org/dev/intro.html>

FURTHER REFERENCES

- ▶ ROS 2 Documentation

- ▶ <https://docs.ros.org/en/humble/>

- ▶ Installation

- ▶ <https://docs.ros.org/en/humble/Installation.html>

- ▶ Tutorials

- ▶ <https://docs.ros.org/en/humble/Tutorials.html>

- ▶ Packages

- ▶ <https://index.ros.org/>

- ▶ tf2

- ▶ <https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html>

- ▶

- ▶ URDF / Xacro

- ▶ <https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html>

- ▶ rqt

- ▶ <https://docs.ros.org/en/humble/Concepts/Intermediate/About-RQt.html>

- ▶ Isaac Sim / USD

- ▶ <https://docs.isaacsim.omniverse.nvidia.com/>

- ▶ <https://openusd.org/>

Material is based on the official ROS2 Documentation