# XACRO Basics

Estimated time to completion: **14 minutes**

## 7.9 Spawning Multiple XACRO Robot Models into Gazebo

In Gazebo, you can create a simulation with multiple robot models, including multiple instances of the same robot model. However, creating a simulation with multiple robots in the same world is not as simple as spawning a model multiple times. This is because each robot would listen to the same command topic, publish their sensor data to the same topics, and define the same TF frames. Therefore, any ROS nodes that rely on sensor data would not function properly, and you would not be able to control the robots individually. To fix this, you must use a different **namespace** for each robot. You can avoid naming collisions between different **topics**, **TF frames**, and **nodes** using namespaces.

Start by creating an empty file for your launch file script.

▶ Execute in Webshell 1

In [ ]:
```
cd ~/ros2_ws/src/my_box_bot_description/launch
```

In [ ]:
```
touch box_bot_spawn_two_robots.launch.py
```

Since the **robot_state_publisher** is required, you must include two instances of it. Additionally, you need two calls to the **spawn_entity.py** script.

box_bot_spawn_two_robots.launch.py

In [ ]:

```
import os
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, ExecuteProcess, IncludeLaunchDescription
from launch.substitutions import Command, LaunchConfiguration
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
from ament_index_python.packages import get_package_prefix

def generate_launch_description():

    pkg_box_bot_gazebo = get_package_share_directory('my_box_bot_gazebo')
    description_package_name = "my_box_bot_description"
    install_dir = get_package_prefix(description_package_name)

    # This is to find the models inside the models folder in my_box_bot_gazebo package
    gazebo_models_path = os.path.join(pkg_box_bot_gazebo, 'models')
    if 'GAZEBO_MODEL_PATH' in os.environ:
        os.environ['GAZEBO_MODEL_PATH'] = os.environ['GAZEBO_MODEL_PATH'] + \
            ':' + install_dir + '/share' + ':' + gazebo_models_path
    else:
        os.environ['GAZEBO_MODEL_PATH'] = install_dir + \
            "/share" + ':' + gazebo_models_path

    if 'GAZEBO_PLUGIN_PATH' in os.environ:
        os.environ['GAZEBO_PLUGIN_PATH'] = os.environ['GAZEBO_PLUGIN_PATH'] + \
            ':' + install_dir + '/lib'
    else:
        os.environ['GAZEBO_PLUGIN_PATH'] = install_dir + '/lib'

    print("GAZEBO MODELS PATH=="+str(os.environ["GAZEBO_MODEL_PATH"]))
    print("GAZEBO PLUGINS PATH=="+str(os.environ["GAZEBO_PLUGIN_PATH"]))

    use_sim_time = LaunchConfiguration('use_sim_time', default='true')

    # Declare a new launch argument for the world file
    world_file_arg = DeclareLaunchArgument(
        'world',
        default_value=[get_package_share_directory(
```

```
            'my_box_bot_gazebo'), '/worlds/box_bot_empty.world'],
        description='Path to the Gazebo world file'
    )

    # Define the launch arguments for the Gazebo launch file
    gazebo_launch_args = {
        'verbose': 'false',
        'pause': 'false',
        'world': LaunchConfiguration('world')
    }

   # Include the Gazebo launch file with the modified launch arguments
    gazebo = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
            get_package_share_directory('gazebo_ros'), 'launch'), '/gazebo.launch.py']),
        launch_arguments=gazebo_launch_args.items(),
    )

    # Define the robot model files to be used
    robot_desc_file = "box_bot_final.urdf"
    robot_desc_path = os.path.join(get_package_share_directory(
        "my_box_bot_description"), "urdf", robot_desc_file)

    robot_name_1 = "robot1"
    robot_name_2 = "robot2"

    rsp_robot1 = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='robot_state_publisher',
        namespace=robot_name_1,
        parameters=[{'frame_prefix': robot_name_1+'/', 'use_sim_time': use_sim_time,
                     'robot_description': Command(['xacro ', robot_desc_path, ' robot_name:=', robot_name_1])}],
        output="screen"
    )

    rsp_robot2 = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='robot_state_publisher',
```

```
        namespace=robot_name_2,
        parameters=[{'frame_prefix': robot_name_2+'/', 'use_sim_time': use_sim_time,
                     'robot_description': Command(['xacro ', robot_desc_path, ' robot_name:=', robot_name_2])}],
        output="screen"
    )

    spawn_robot1 = Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        arguments=['-entity', 'robot1', '-x', '0.0', '-y', '0.0', '-z', '0.0',
                   '-topic', robot_name_1+'/robot_description']
    )

    spawn_robot2 = Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        arguments=['-entity', 'robot2', '-x', '1.0', '-y', '1.0', '-z', '0.0',
                   '-topic', robot_name_2+'/robot_description']
    )

    return LaunchDescription([
        world_file_arg,
        gazebo,
        rsp_robot1,
        rsp_robot2,
        spawn_robot1,
        spawn_robot2
    ])
```

The launch file mentioned earlier has the added advantage of reading robot model descriptions written in both URDF and XACRO formats. To test this, all you need to do is locate this part:

In [ ]:

```
# Define the robot model files to be used
robot_desc_file = "box_bot_final.urdf"
```

And replace the `.urdf` file with a `.xacro` version of the robot:

In [ ]:

```
# Define the robot model files to be used
robot_desc_file = "box_bot.xacro"
```

However, you are not quite ready yet. You might have noticed that you only have one cmd_vel topic and one Odom topic. This does not allow you to control each robot independently. Additionally, you might have noticed the following warning message in the console during startup.

```
[gzserver-1] [WARN] [1679423567.337457462] [rcl.logging_rosout]: Publisher already registered for provided node name. If this is due to multiple
nodes with the same name then all logs for that logger name will go out over the existing publisher. As soon as any node with that name is destr
oyed it will unregister the publisher, preventing any further logs for that name from being published on the rosout topic.
```

Let us address the issue of having only one `cmd_vel` topic and one `odom` topic. This involves adding a namespace to both Gazebo plugins. To do so, add a `robot_name` property whose value is an argument you will call `robot_name` . So, you need to add the following code at the top of the XACRO file, below the opening `<robot ... >` element:

In [ ]:

```
<xacro:property name="robot_name" value="$(arg robot_name)"/>
```

In the above line of code, if the `robot_name` argument is provided, the value of the `robot_name` property will be set to the value of the `robot_name` argument.

The next step to avoid name conflicts in ROS2 topics is to use namespaces within the plugins that publish and subscribe to topics. In ROS2, some Gazebo plugins use `<ros>` tags to specify ROS2-specific settings for the plugin, such as the ROS2 node namespace, remappings, and other ROS2-related parameters. By defining these settings in the tag, the plugin can communicate with other ROS2 nodes and publish and subscribe to ROS2 topics.

For example, in the case of the Joint State Publisher node, you would need to add these XML tags:

In [ ]:

```
<ros>
    <namespace>/${robot_name}</namespace>
    <remapping>~/out:=joint_states</remapping>
</ros>
```

In the snippet above, the `<ros>` tag is used to specify the namespace for the ROS2 node associated with the gazebo_ros_joint_state_publisher plugin. In this case, the namespace is set to `/${robot_name}` to use the value of the `robot_name` argument, which is passed in when launching the robot.

The modified Gazebo plugins with the added `<ros>` and `<namespace>` tags are shown below:

In [ ]:

```
<gazebo>
  <plugin name="box_bot_joint_state" filename="libgazebo_ros_joint_state_publisher.so">
    <ros>
        <namespace>/${robot_name}</namespace>
        <remapping>~/out:=joint_states</remapping>
    </ros>
    <update_rate>30</update_rate>

    <joint_name>joint_left_wheel</joint_name>
    <joint_name>joint_right_wheel</joint_name>
    <joint_name>front_yaw_joint</joint_name>
    <joint_name>back_yaw_joint</joint_name>
    <joint_name>front_roll_joint</joint_name>
    <joint_name>back_roll_joint</joint_name>
    <joint_name>front_pitch_joint</joint_name>
    <joint_name>back_pitch_joint</joint_name>

  </plugin>
</gazebo>

<gazebo>
  <plugin filename="libgazebo_ros_diff_drive.so" name="differential_drive_controller">
      <ros>
          <namespace>/${robot_name}</namespace>
          <remapping>/cmd_vel:=cmd_vel</remapping>
      </ros>

      <update_rate>100</update_rate>
      <!-- Wheel joints -->
      <left_joint>joint_left_wheel</left_joint>
      <right_joint>joint_right_wheel</right_joint>
      <!-- Kinematics -->
      <wheel_separation>0.1</wheel_separation>
      <wheel_diameter>0.07</wheel_diameter>
      <publish_odom>true</publish_odom>
      <!-- TF frames -->
      <odometry_frame>${robot_name}/odom</odometry_frame>>
      <robot_base_frame>${robot_name}/base_link</robot_base_frame>
      <publish_odom_tf>true</publish_odom_tf>
```

```
        <publish_wheel_tf>false</publish_wheel_tf>
        <!-- Limits -->
        <max_wheel_torque>1.0</max_wheel_torque>
        <max_wheel_acceleration>2.0</max_wheel_acceleration>

    </plugin>
  </gazebo>
```

Note that in the code above, the `${robot_name}` parameter is also used to define the namespace inside the `<odometry_frame>` and `<robot_base_frame>` inside the differential drive controller plugin. By using the robot name inside the tags that define the names of the odometry and base_link frames for the robot, you can also keep separated TF frames for the odometry transform.

If you want to verify the latest changes, you must recompile the workspace and launch it again:

▶ Execute in Webshell 1

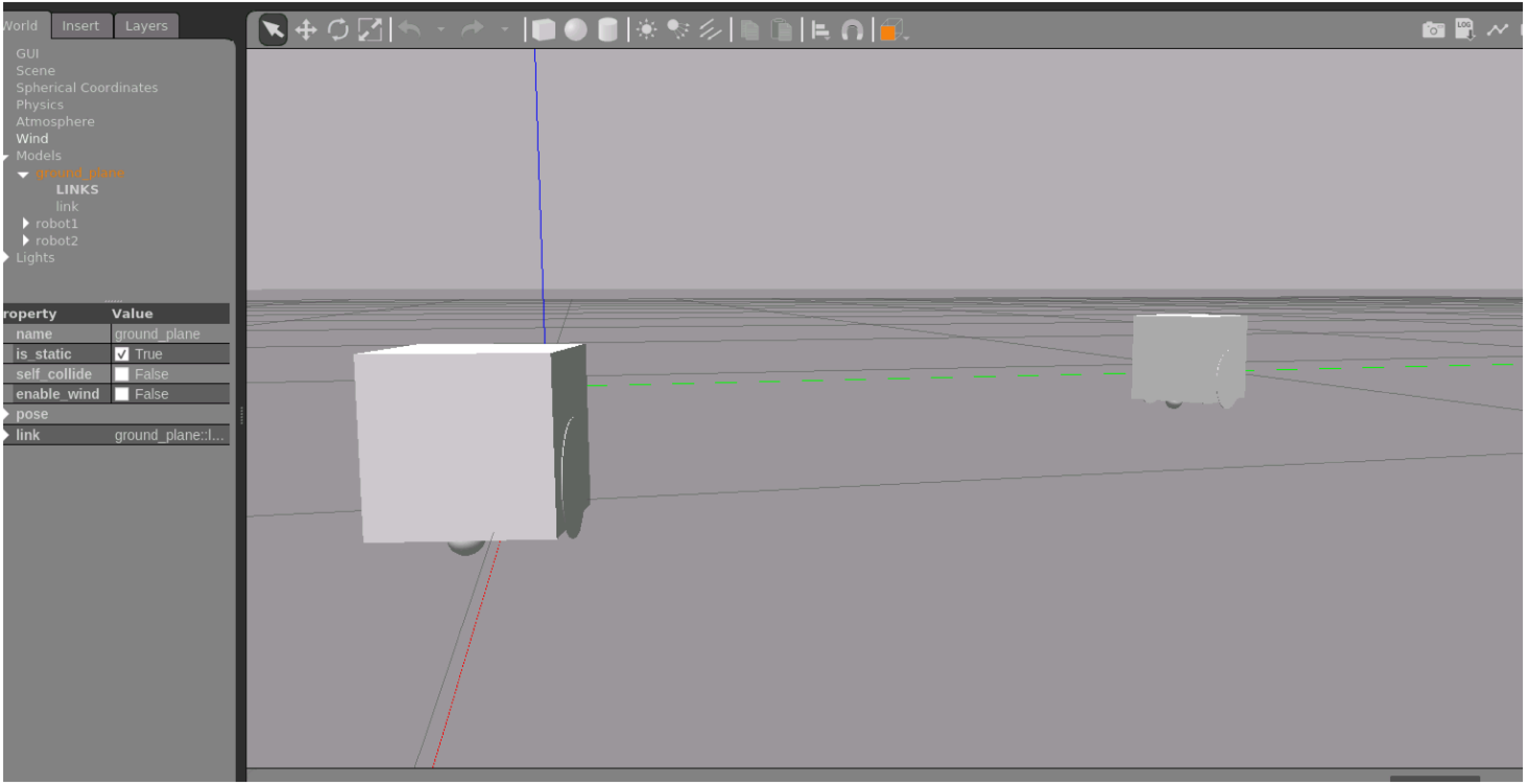
In [ ]:

```
cd ~/ros2_ws && colcon build && source install/setup.bash
```

In [ ]:

```
ros2 launch my_box_bot_description box_bot_spawn_two_robots.launch.py
```

You should see **two robots now**:




Verify namespaces separate the topics:

Two robots should appear in Gazebo.

▶ Execute in Webshell 2

In [ ]:

```
ros2 topic list
```

You should be able to see the following topics, which confirms that you have two separate `cmd_vel` , `odom` , `joint_states` and `robot_description` topics, one for each robot:

Expected output:

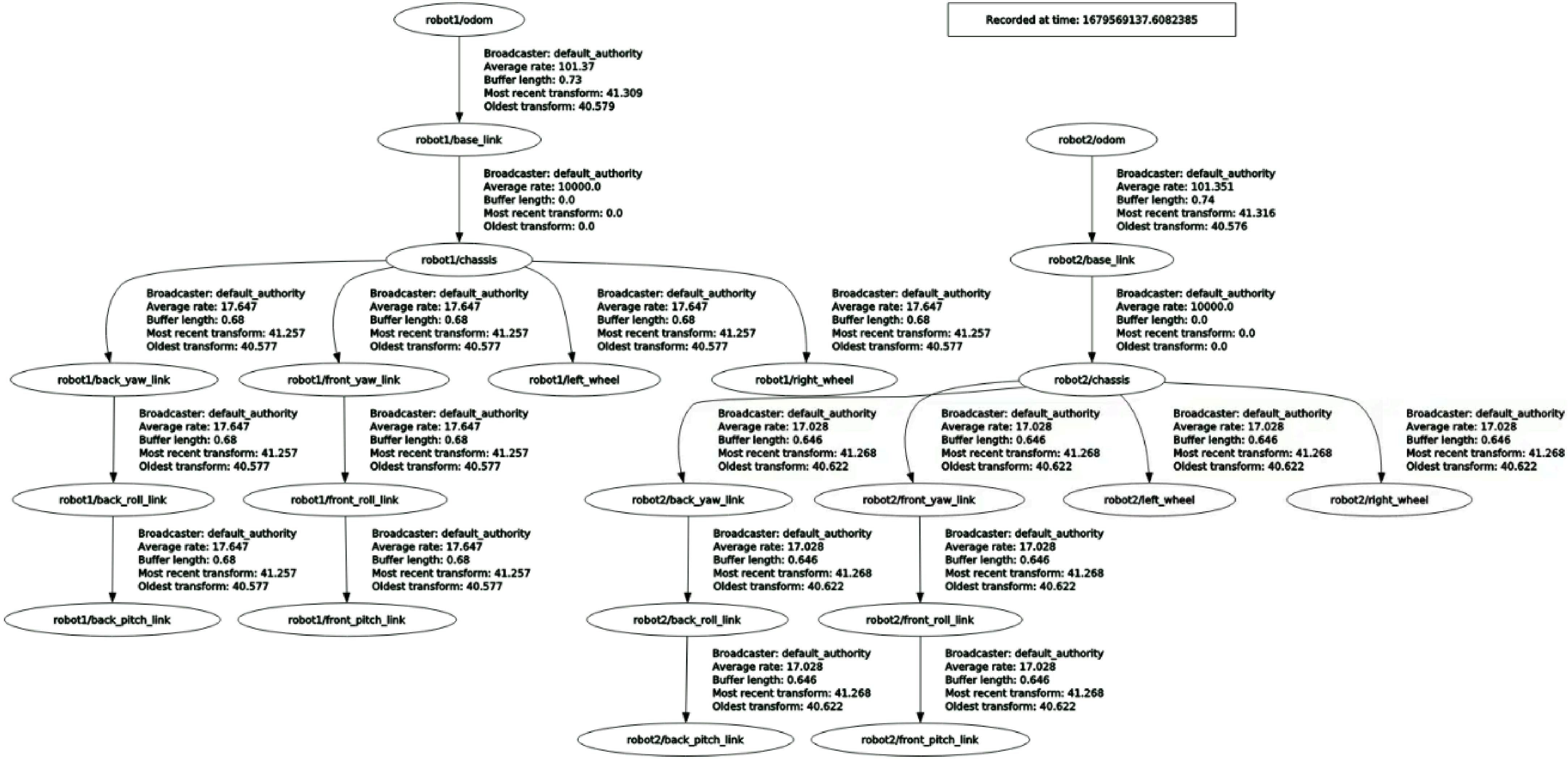
```
/clock
/joint_states
/parameter_events
/performance_metrics
/robot1/cmd_vel
/robot1/joint_states
/robot1/odom
/robot1/robot_description
/robot2/cmd_vel
/robot2/joint_states
/robot2/odom
/robot2/robot_description
/rosout
/tf
/tf_static
```

Inspect the full TF for this ROS2 system with two robots:

▶ **Execute in Webshell 2**

In [ ]:
```
ros2 run rqt_tf_tree rqt_tf_tree
```

You should see that the graphical tools window pops up automatically. A few seconds later, you should see a TF tree similar to this (use the Mouse wheel to zoom in):




Last but not least, confirm you can drive each robot around:

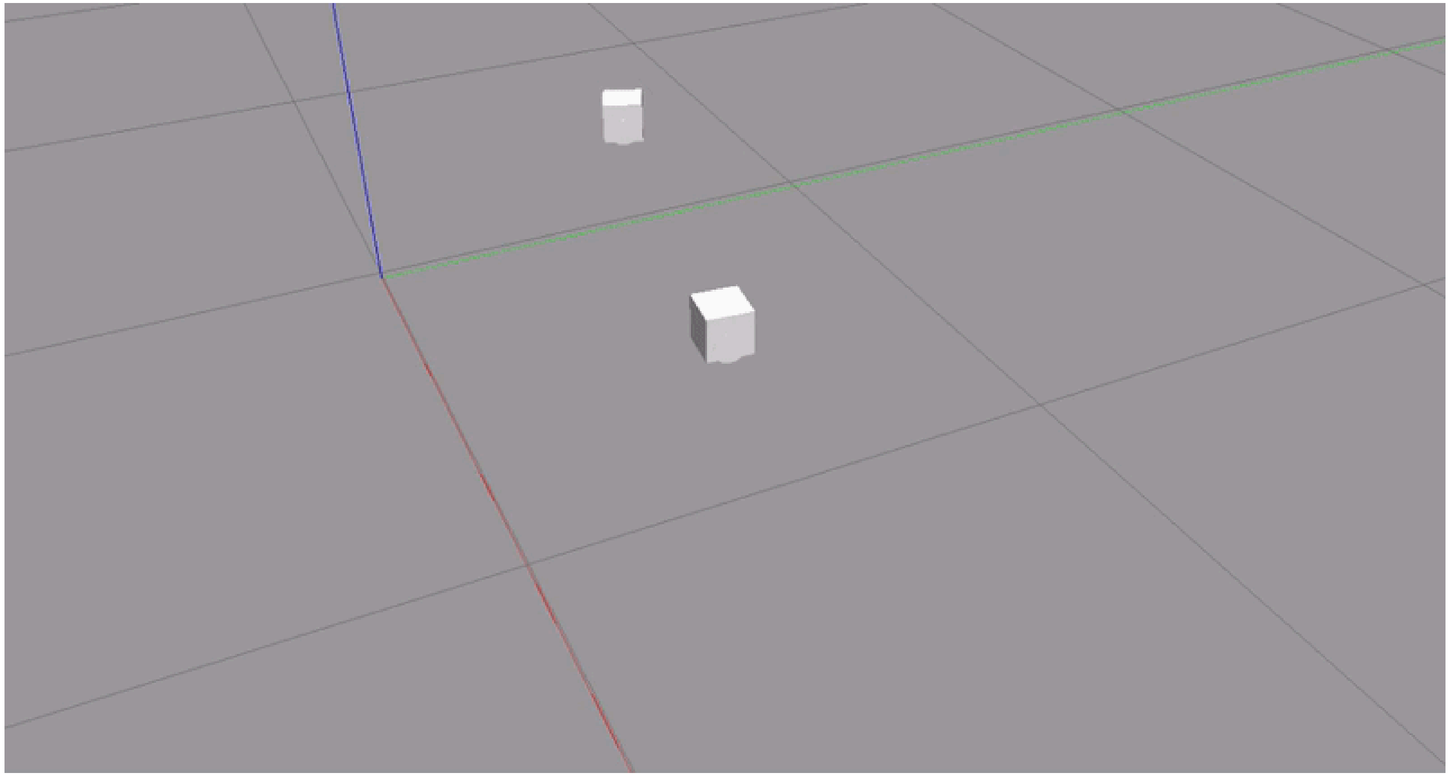
▶ Execute in Webshell 3

In [ ]:
```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap cmd_vel:=/robot1/cmd_vel
```

▶ Execute in Webshell 4

In [ ]:
```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap cmd_vel:=/robot2/cmd_vel
```

Press the keys shown with the focus on the window in which you executed the teleop_twist_keyboard program.



16/11/2023