Note:

1. This Quiz is from the course, TF ROS 101 on Robot Ignite Academy
: https://www.robotigniteacademy.com/en/course/tf-ros-101/details/
2. Any contents of the quiz belong to Robot Ignite Academy except for the sample solution written by Samwoo Seong. I.e. I don't own any of quiz contents
3. Any work throughout the quiz is for learning purpose
4. The solution written by Samwoo Seong shouldn't be used to pass the quiz on this course

Samwoo Seong
samwoose@gmail.com

Quiz, TF ROS 101
June 28, 2020

1.0 Objectives

Quiz 1.

-Publish the "joint_states" of all the movable joints by activating the joint controllers already defined in the pi_robot_v2.urdf. [1]

Quiz 2.

-Find the frame that they share. [1]

-Create a launch file that publishes a static transform between both common frames. [1]

I.e. Connect TF of two robots using static transform so that the two robots can be represented in RVIZ at the same time.

1.1 Abstract and Motivation

A robot tends to have a lot of sensors and data from the sensors is used to perform specific tasks. For instance, we want a mobile robot to avoid a wall before it hits the wall and we are going to amount a sonar sensor to measure distance between the wall and our robot. The distance obtained by sensor can be different from an actual distance between the robot and the wall depending on the location of the sensor. Therefore, we need a way to represent robot's location and location of the sensor so that we can properly utilize data from the sonar sensor and calculate the actual distance between the robot and wall. This is when TF, Transform, in ROS comes in handy. Briefly speaking, there are 3 ways to use this TF.

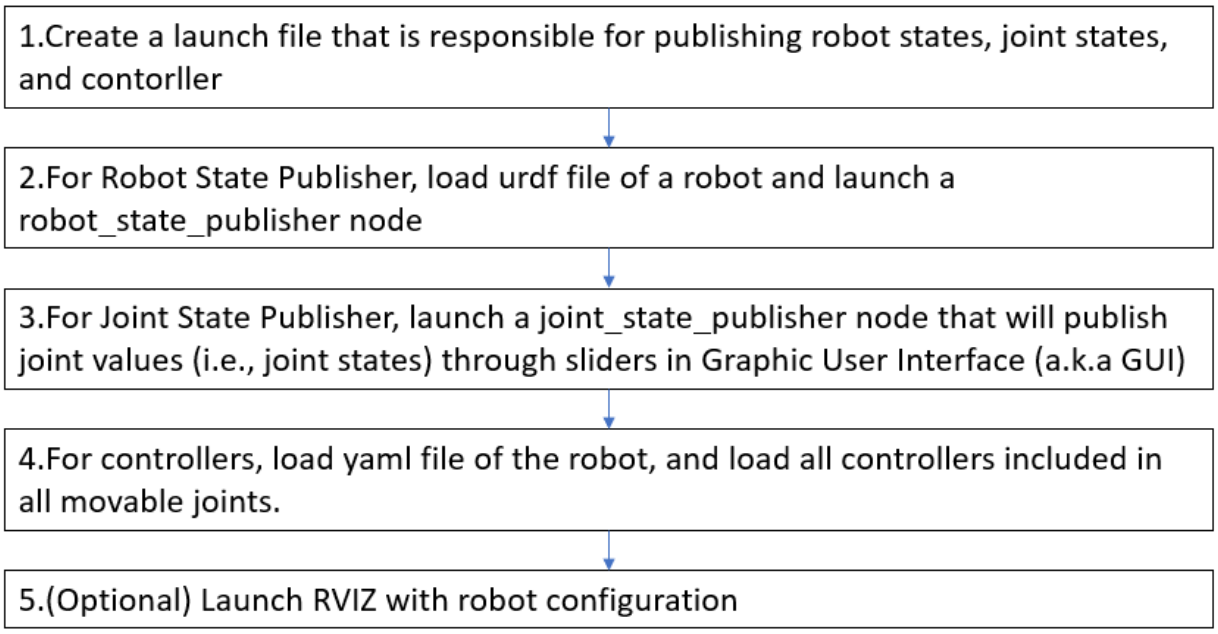
(1) We can manually broadcast and listen TF by defining nodes (also known as TF broadcaster and TF listener) properly.

(2) We can use Robot State Publisher and Joint State Publisher in case we have properly defined URDF files of robots.

(3) We can share frames from different robots using static transform publisher so that we are able to work with multiple robots in RVIZ with out defining unified URDF that contains information about different robots.
In this practice, we are going to make a use of method 2 for Quiz 1 and method 3 for Quiz 2

1.2 Approach and Procedures
<Quiz 1>

Our goal is to publish the "joint_states" of all the movable joints by activating the joint controllers defined in the pi_robot_v2.urdf. Since we already have pi_robot_v2.urdf that contains all details about robot's configuration from its design to all controllers included in it, we can utilize a great tool called robot state publisher which takes the urdf file and publishes appropriate TF based on the robot's morphology. To make a use of robot state publisher and joint state publisher, we can follow the procedure below.



Flowchart 1. Workflow of a launch file that launches robot state publisher, joint state publisher, and controllers

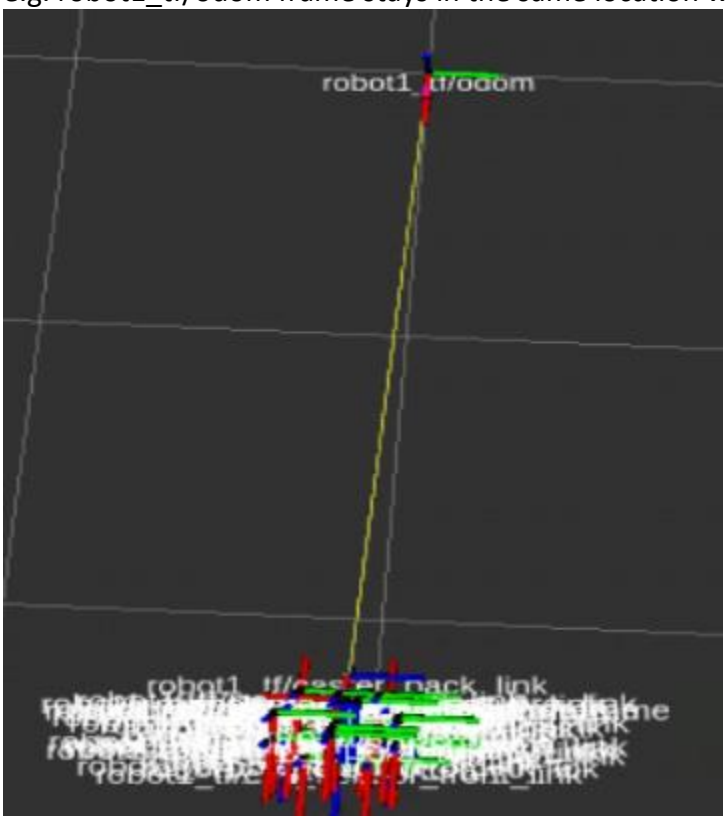
Each step will be further explored in 1.4 discussion.

<Quiz 2>

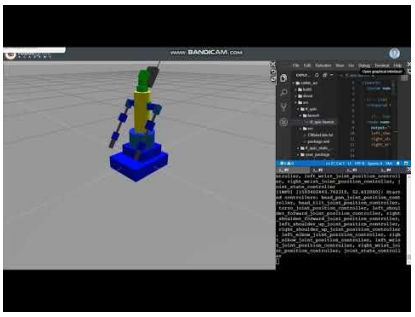
Briefly speaking of static transform, it is useful when we want to temporarily employ sensors without writing a whole urdf description and sensors are located outside of robot. In our practice, the goal is to publish static transform so that all TF of two turtlebots can exist in RVIZ at the same time without writing a new URDF file that contains all information about the two robots. Therefore, we can see one robot as main robot and the other robot as an external sensor. To

connect those two robots, we need to find a frame from each robot and connect those two frames publishing static transform. If you take a look at frames in RVIZ while you are moving around turtlebots by publishing values to /robot1/cmd_vel and /robot2/cmd_vel topics, you will soon realize that odom frame of each robot is staying in the same location while other frames are moving as we publish value to the topics(when you set /robot1/odom to fixed frame in RVIZ). Therefore, in our launch file, we will make a use of these odom frames and static transform publisher.

e.g. robot1_tf/odom frame stays in the same location while other frames are moving



1.3 Experimental Results
<Quiz 1>



https://www.youtube.com/watch?v=31LLv7v21WM
Video 1. Moving joints of Pi robot with GUI and terminal

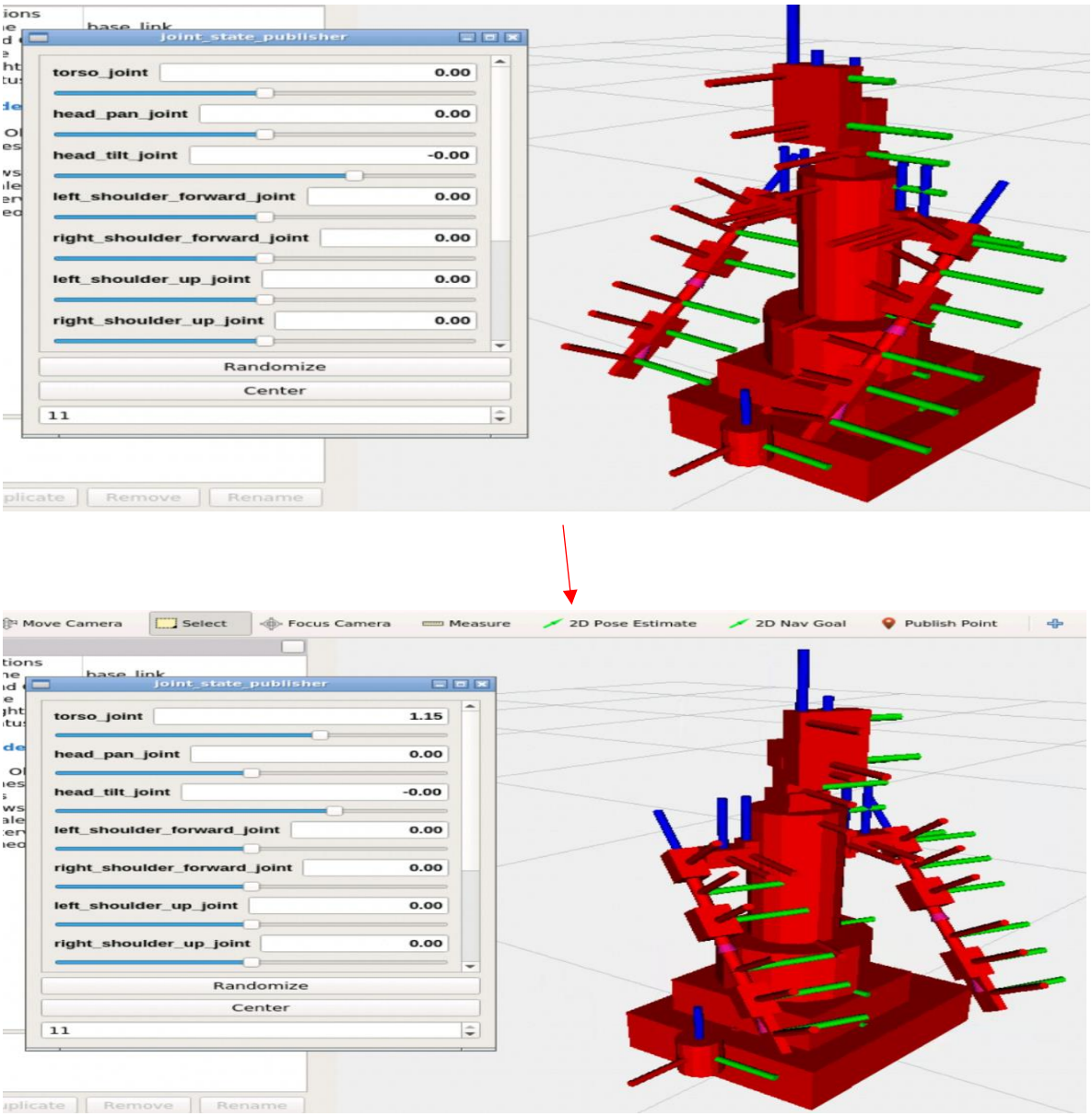


Fig 1. Before (1st image) and after (2nd image) publishing joint state (i.e., joint value) of torso_joint through GUI in RVIZ

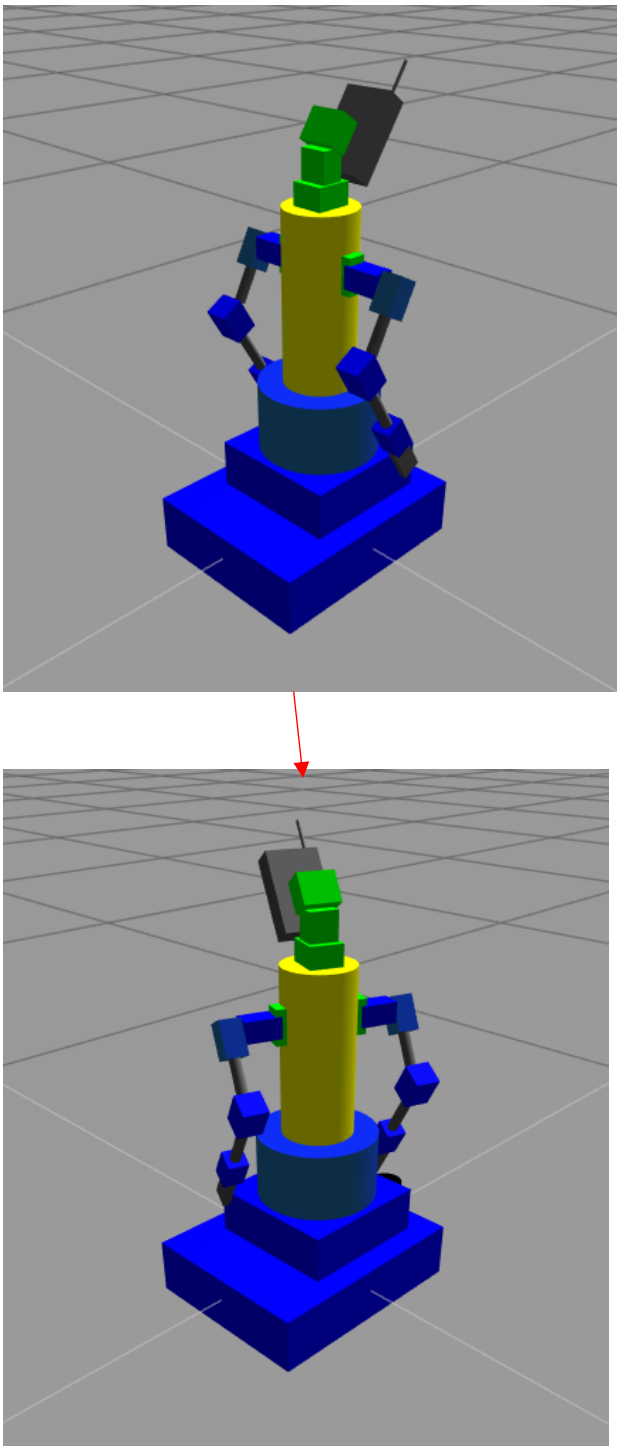
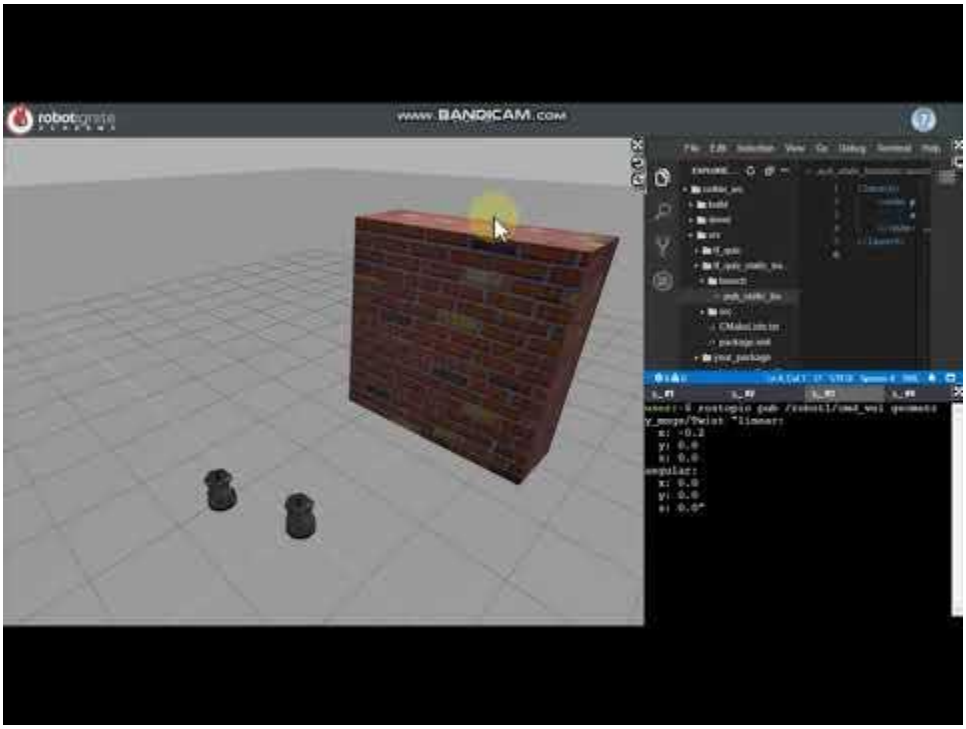


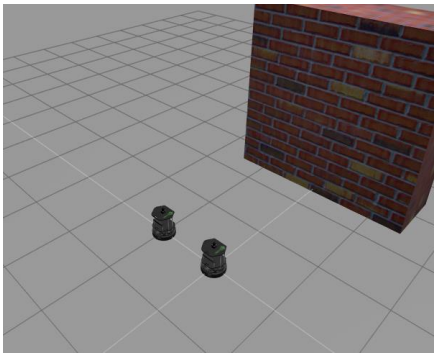
Fig 2. Before (1st image) and after (2nd image) publishing joint state (i.e., joint value) of torso_joint through command for Gazebo simulation

<Quiz2>



https://www.youtube.com/watch?v=ZTKHgYz8P3A

Video 2. Launching static transform publisher node and controlling two turtlebots



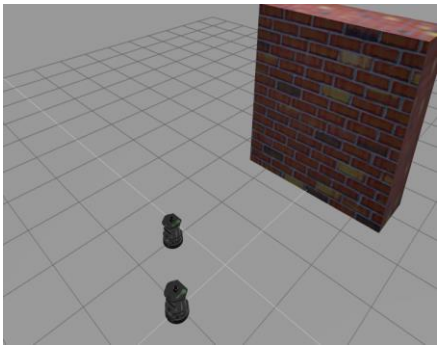


Fig 3. Before (1st image) and after (2nd image) publishing message to /cmd_vel topic of each robot in simulation

Fig 4. Before publishing static transform. As we can see, TF of only one robot can be shown at a time.

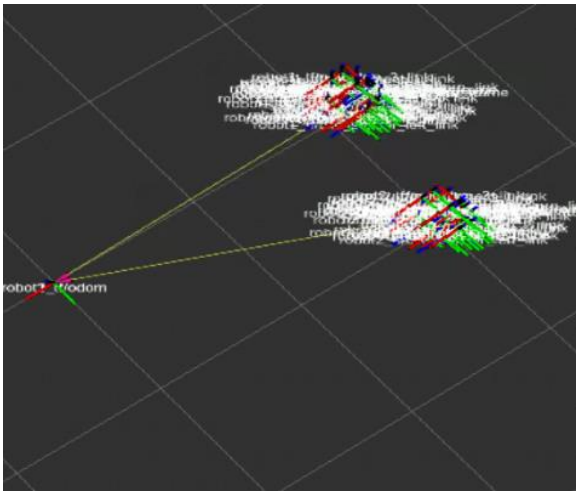


Fig 5. After publishing static transform and message to /cmd_vel topic. All TFs are shown at the same time.

1.4 Discussion

<Quiz 1>

It is worthy going through details in our launch file and see role(s) of each part. In launch file, we have robot_state_publisher node for robot state, joint_state_publisher node for joint state, controller_spawner node for controllers, and rviz node for launching RVIZ. Additionally, we load

yaml file so that our program can realize what controller are defined. Lastly, we have urdf file which is taken by robot_state_publisher node.

-robot_state_publisher node.
To properly launch this node, we need appropriate urdf file of target robot. This node's role is to automatically publish all TF for fixed joint for us in our robot based on robot's structure in our urdf file.

e.g. Taking urdf file of Pi robot as parameter

```
<launch>
  <param name="robot_description" command="cat $(find pi_robot_pkg)/urdf/pi_robot_v2.urdf" />
```

e.g. robot_state_publisher takes the urdf file as an input and publish TF for us.

```
<!-- Combine joint values -->
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
```

-joint_state_publisher node.
Even though robot_state_publisher does a tremendous job for us, we still need to publish TF for non-fixed joints. This is when joint_state_publisher node comes in handy. There are many ways to publish joint states (i.e., joint values). First of all, we can publish non-physical value via graphical user interface also known as GUI. In our practice, we will adjust sliders in user interface which adjusts value of joint states. Second, we can obtain joint value from real hardware such as encoder on motor. For instance, encoder can represent how much a motor has moved and this data from encoder can indicate joint state moved by the motor. However, in our practice, we will use partial of this method by putting some values via terminal command since we are working on only Gazebo simulation without encoder-like devices.

e.g. Launcing joint_state_publisher and value of joint state will be adjusted by use_gui (GUI)

```
<!-- send fake joint values -->
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
  <param name="use_gui" value="TRUE"/>
</node>
```

e.g. First method to publish joint state with GUI

e.g. Second method to publish joint state with arbitrary value through terminal

```
user:~$ rostopic pub /pi_robot/torso_joint_position_controller/command std_msgs/Float64 "data: 1"
```

-controller_spawner node

Even though we are able to get all TF from fixed and non-fixed joint (thanks to robot_state_publisher and joint_state_publisher) we cannot actually move the robot's joints because we have not launched its controllers that control the joints. To achieve this, we need yaml file of the robot that defines all details of controllers (it is highly correlated with urdf file of the robot) and controller_spawner node.

e.g. Loading pirobot_control.yaml file

```
<!-- Load joint controller configurations from YAML file to parameter server -->
<rosparam file="$(find pi_robot_pkg)/config/pirobot_control.yaml" command="load"/>
```

e.g. Launching controller_spawner node. Note that all movable controllers are in "args"

```
  <!-- load the controllers -->
<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
  output="screen" ns="/pi_robot" args="head_pan_joint_position_controller head_tilt_joint_position_controller
  torso_joint_position_controller left_shoulder_forward_joint_position_controller
  right_shoulder_forward_joint_position_controller left_shoulder_up_joint_position_controller
  right_shoulder_up_joint_position_controller left_elbow_joint_position_controller
  right_elbow_joint_position_controller left_wrist_joint_position_controller
  right_wrist_joint_position_controller joint_state_controller"/>
```

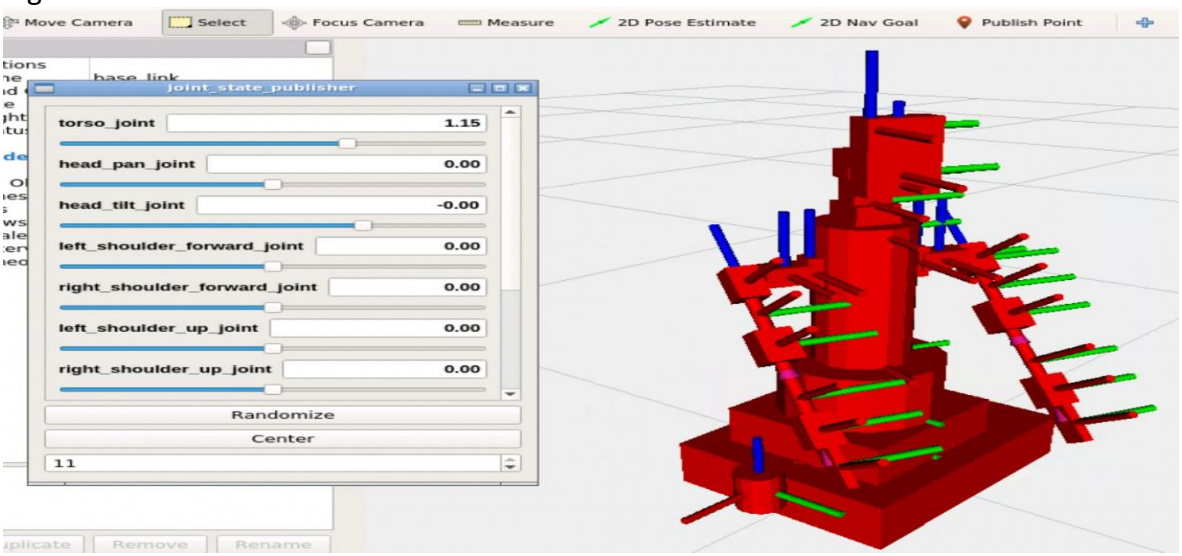
-rviz node

This node launches the ROS visualization tool, RVIZ so that we can see any changes while adjusting joint states with GUI.

e.g. Launching rviz node

```
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find pi_robot_pkg)/launch/pi_robot.rviz"/>
```

e.g. GUI and RVIZ for Pi robot



<Quiz 2>

To connect two robots by publishing static transform, we need to write a launch file that run node called world_frames_connection which has static_transform_publisher type. This node also takes several arguments such as x, y, z, yaw, pitch, roll, frame ID, child frame ID, publish frequency in msec.

e.g. node that publish static transform.

```
<launch>
    <node pkg="tf" type="static_transform_publisher" name="world_frames_connection"
          args="0 0 0 0 0 0 robot1_tf/odom robot2_tf/odom 1">
    </node>
</launch>
```

In this node, arguments are defined as follows.
x = 0, y = 0, z = 0,
yaw = 0, pitch = 0, roll = 0,
frame ID = robot1_tf/odom,
child frame ID = robot2_tf/odom,
publish frequency in msec = 1

Note: we assume the two turtlebots are already running on simulation.

References

[1] Quiz Contents Credit: Robot Ignite Academy

https://www.robotigniteacademy.com/en/course/tf-ros-101/details/