Report - Homework 1

Emanuele Cuzzocrea P38000196

Group:
Emanuele Cuzzocrea
Silvia Leo
Vito Daniele Perfetta
Giulia Gelsomina Romano

Link to the Github repository: `https://github.com/EmanueleCuzzocrea/Homework1.git`

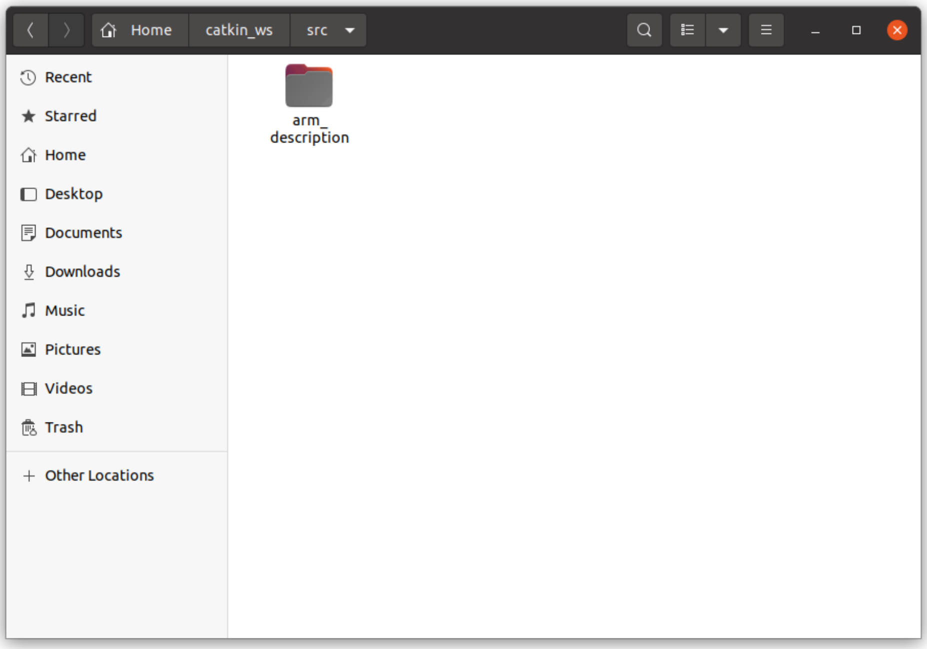
1. Create the description of your robot and visualize it in Rviz

(a) Download the arm_description package from the repo `https://github.com/RoboticsLab2023/arm_description.git` into your catkin_ws using git commands

After copying the link, the terminal commands that were executed to download the arm_description package into the catkin_ws workspace were as follows

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/RoboticsLab2023/arm_description.git
```

This way, the arm_description package was successfully downloaded into the folder ~/catkin_ws/src



(b) Within the package create a launch folder containing a launch file named display.launch that loads the URDF as a robot_description ROS param and starts the robot_state_publisher node, the joint_state_publisher node, and the rviz node. Launch the file using roslaunch. Note: To visualize your robot in rviz you have to changhe the Fixed Frame in the lateral bar and add the RobotModel plugin interface. Optional: save a .rviz configuration file, thad automatically loads the RobotModel plugin by default, and give it as an argument to your node in the display.launch file

```
$ cd ~/catkin_ws/src/arm_description
$ mkdir launch
$ cd launch
$ touch display.launch
```

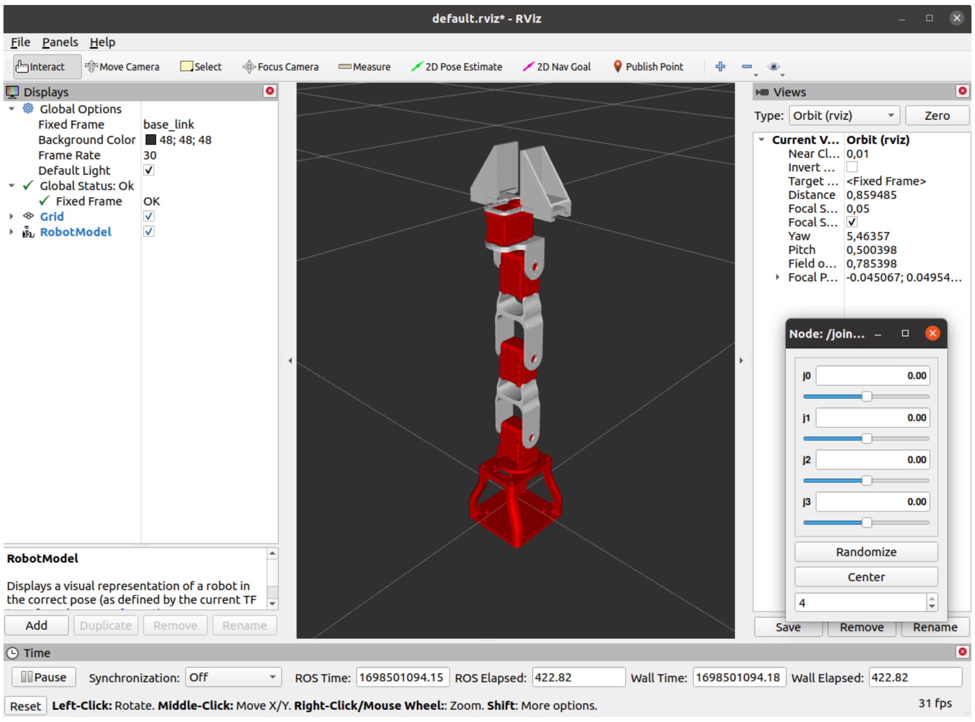
Inside the display.launch file, the following code was inserted

```
<?xml version="1.0" ?>

<launch>
  <param name="robot_description" textfile="$(find arm_description)/urdf
      /arm.urdf" />
```

```
    <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
        type="joint_state_publisher_gui" />
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
       robot_state_publisher" />
    <node name="rviz" pkg="rviz" type="rviz" required="true" />

</launch>
```

To visualize the robot within Rviz, it was necessary to change the Fixed Frame from map to base_link and then add the RobotModel plugin

```
$ cd ~/catkin_ws
$ catkin build
$ source devel/setup.bash
$ roslaunch arm_description display.launch
```


default.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Views
Node: /join...
Randomize
Center
Add Duplicate Remove Rename
Save Remove Rename
Time
Pause
Reset

Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓

RobotModel

Displays a visual representation of a robot in the correct pose (as defined by the current TF

Type: Orbit (rviz) Zero

Current V...	Orbit (rviz)
Near Cl...	0,01
Invert ...	☐
Target ...	<Fixed Frame>
Distance	0,859485
Focal S...	0,05
Focal S...	✓
Yaw	5,46357
Pitch	0,500398
Field o...	0,785398
Focal P...	-0.045067; 0.04954...

j0	0.00
j1	0.00
j2	0.00
j3	0.00

4

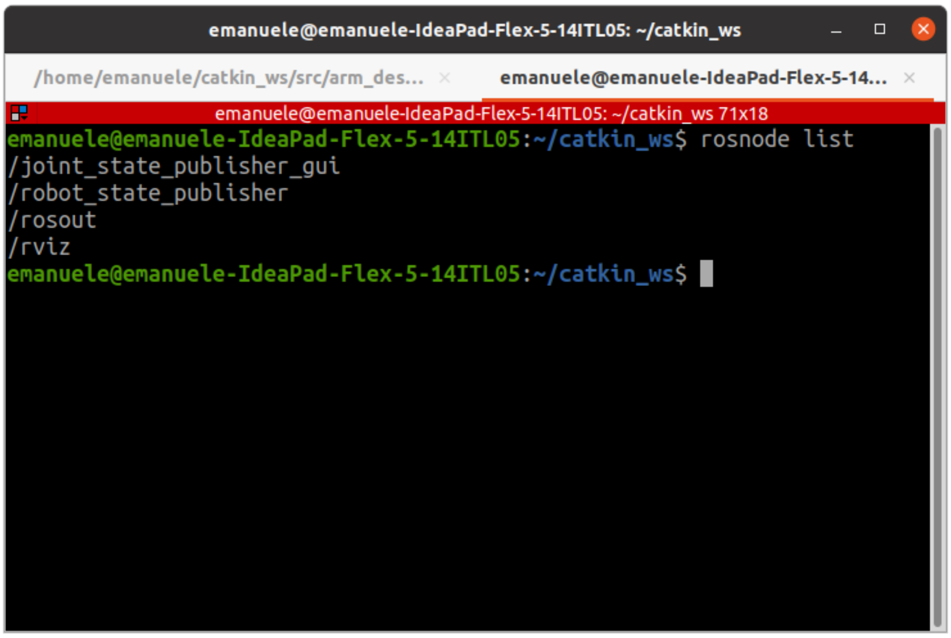
Synchronization: Off ROS Time: 1698501094.15 ROS Elapsed: 422.82 Wall Time: 1698501094.18 Wall Elapsed: 422.82

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options. 31 fps

At this point, the following configuration was saved, and the corresponding configuration.rviz file was inserted into the arm_description package. Passing the file as an argument to the Rviz node was then done in the following way

```
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find arm_description
    )/configuration.rviz" required="true"/>
```

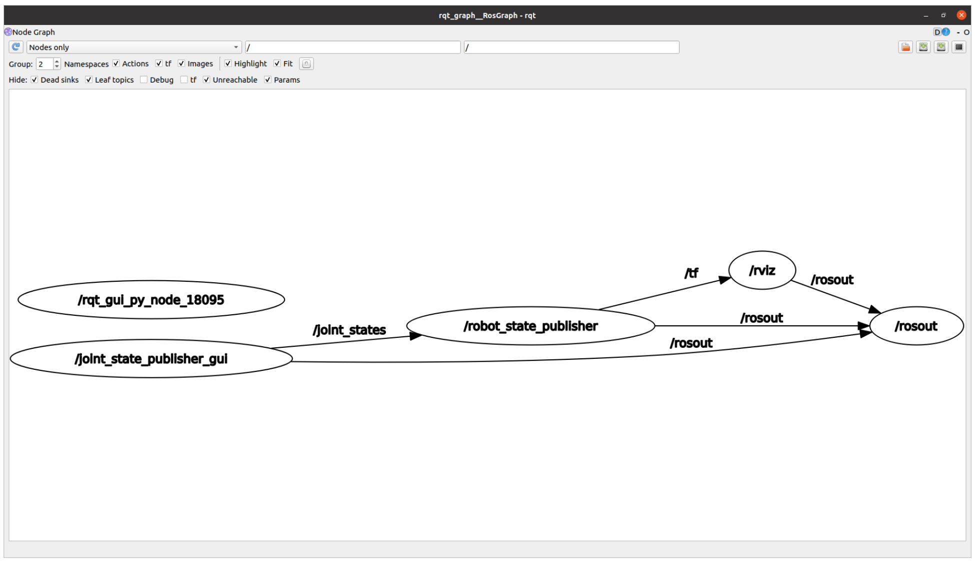
Let's visualize the nodes that are running at the moment



```
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws
emanuele@emanuele-IdeaPad-Flex-5-14ITL05:~/catkin_ws$ rosnode list
/joint_state_publisher_gui
/robot_state_publisher
/rosout
/rviz
emanuele@emanuele-IdeaPad-Flex-5-14ITL05:~/catkin_ws$ 
```

We can also run rqt_graph, in order to clearly understand what is going on between the nodes and the topics

```
$ rqt_graph
```



```
rqt_graph__RosGraph - rqt
Node Graph
Nodes only   /   /
Group: 2  Namespaces ✓ Actions ✓ tf ✓ Images | ✓ Highlight ✓ Fit
Hide: ✓ Dead sinks ✓ Leaf topics ☐ Debug ☐ tf ✓ Unreachable ✓ Params
```

(c) Substitute the collision meshes of your URDF with primitive shapes. Use <box> geometries of reasonabe size approximating the links. Hint: Enable collision visualization in rviz (go to the lateral bar > Robot model > Collision Enabled) to adjust the collision meshes size

Inside the arm.urdf file, to replace the link collisions with simple shapes, the <collision> tag for each link of the manipulator was modified. The following example is provided for the base_link

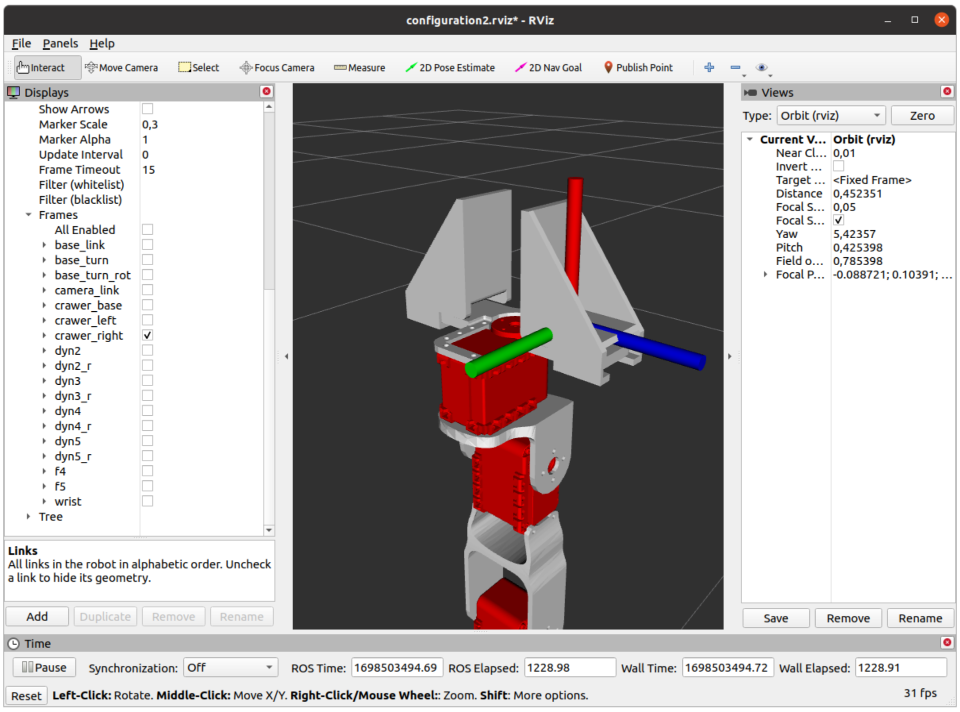
```
...
  <link name="base_link">
    <visual>
      <geometry>
```

```
        <mesh filename="package://arm_description/meshes/base_link.stl"
            scale="0.001 0.001 0.001"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </visual>
    <collision>
      <geometry>
        <box size = "0.1 0.1 0.1" />
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="1.06682889e+08" ixy="0.0" ixz="0.0" iyy="9.92165844e
         +07" iyz="0.0" izz="1.26939175e+08"/>
    </inertial>
  </link>
...
```

However, sometimes modifying only the dimensions was not sufficient, as the box moved too far from the desired position. Consequently, in some cases, the reference frame position was also adjusted, as in the case of crawler_right, for instance

```
...
  <link name="crawer_right">
    <visual>
      <geometry>
        <mesh filename="package://arm_description/meshes/the_crawer.stl"
             scale="0.001 0.001 0.001"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
          <box size = "0.055 0.04 0.045" />
        </geometry>
      <origin rpy="0 0 0" xyz="0.023 0 0.0075"/>
    </collision>
    <inertial>
      <mass value="0.1"/>
      <inertia ixx="6.90013881e+01" ixy="0.0" ixz="0.0" iyy="1.24554875e
         +02" iyz="0.0" izz="1.04295793e+02"/>
    </inertial>
  </link>
...
```

To correctly size and position the boxes in the right directions, it was crucial to visualize the reference frames. Thanks to them, it was possible to understand the directions of the x, y, and z axes, in order to adjust the values properly inside the arm.urdf file. The following is an example of reference frame visualization for the crawler_right link



configuration2.rviz* - RViz

Displays

Show Arrows	☐
Marker Scale	0,3
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Filter (whitelist)	
Filter (blacklist)	
Frames	
All Enabled	☐
base_link	☐
base_turn	☐
base_turn_rot	☐
camera_link	☐
crawer_base	☐
crawer_left	☐
crawer_right	✓
dyn2	☐
dyn2_r	☐
dyn3	☐
dyn3_r	☐
dyn4	☐
dyn4_r	☐
dyn5	☐
dyn5_r	☐
f4	☐
f5	☐
wrist	☐
Tree	

Links
All links in the robot in alphabetic order. Uncheck a link to hide its geometry.

Views

Type: Orbit (rviz)

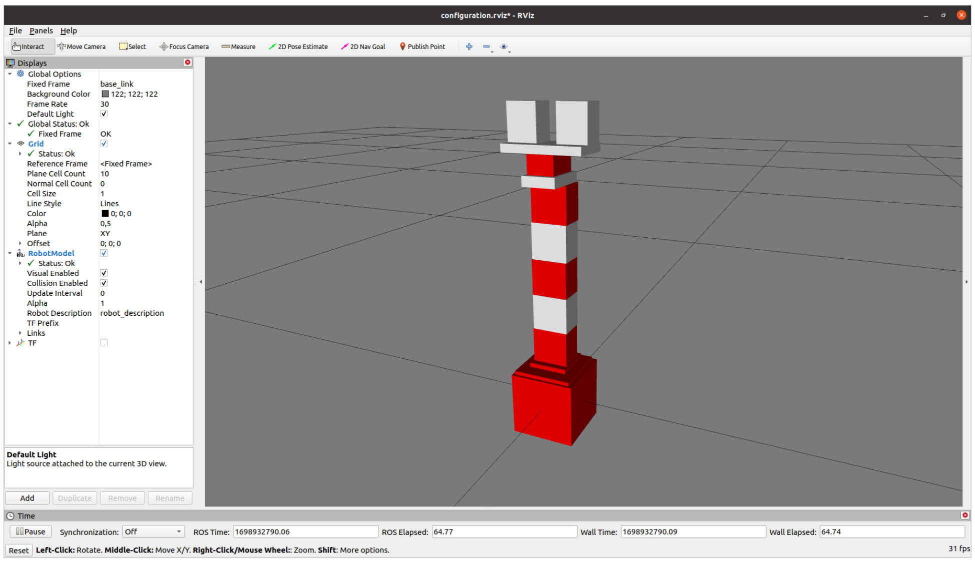
Current V...	Orbit (rviz)
Near Cl...	0,01
Invert ...	☐
Target ...	<Fixed Frame>
Distance	0,452351
Focal S...	0,05
Focal S...	✓
Yaw	5,42357
Pitch	0,425398
Field o...	0,785398
Focal P...	-0.088721; 0.10391; ...

Time

Synchronization: Off   ROS Time: 1698503494.69   ROS Elapsed: 1228.98   Wall Time: 1698503494.72   Wall Elapsed: 1228.91

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.   31 fps

The final result is as follows, where it can be seen that the collisions completely cover the actual robot meshes



configuration.rviz* - RViz

Displays

Global Options	
Fixed Frame	base_link
Background Color	122; 122; 122
Frame Rate	30
Default Light	✓
Global Status: Ok	
Fixed Frame	OK
Grid	✓
Status: Ok	
Reference Frame	<Fixed Frame>
Plane Cell Count	10
Normal Cell Count	0
Cell Size	1
Line Style	Lines
Color	■ 0; 0; 0
Alpha	0,5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	✓
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
TF	☐

Default Light
Light source attached to the current 3D view.

Time

Synchronization: Off   ROS Time: 1698932790.06   ROS Elapsed: 64.77   Wall Time: 1698932790.09   Wall Elapsed: 64.74

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.   31 fps

(d) Create a file named arm.gazebo.xacro within your package, define a xacro:macro inside your file containing all the <gazebo> tags you find within your arm.urdf and import it in your URDF using xacro:include. Remember to rename your URDF file to arm.urdf.xacro, add the string xmlns:xacro="http://www.ros.org/wiki/xacro" within the <robot> tag, and load the URDF in your launch file using the xacro routine

First of all, let's create the arm.gazebo.xacro file

```
$ cd ~/catkin_ws/src/arm_description/urdf
$ touch arm.gazebo.xacro
```

In this file, all the functions from the arm.urdf file containing the <gazebo> tag were inserted

```
<?xml version="1.0"?>

<robot xmlns:xacro="http://ros.org/wiki/xacro">

  <xacro:macro name="arm_gazebo" params="robot_name">

    <gazebo reference="f4">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="f5">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="wrist">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="crawer_base">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_link">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_turn">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="base_turn_rot">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="dyn2">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn3">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn4">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="dyn5">
        <material>Gazebo/Black</material>
    </gazebo>

    <gazebo reference="crawer_left">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo reference="crawer_right">
        <material>Gazebo/Red</material>
    </gazebo>
  </xacro:macro>

</robot>
```

At this point, the following include was added to the arm.urdf file

```
<xacro:include filename="$(find arm_description)/urdf/arm.gazebo.xacro"
   />
```

Of course, the <robot> tag inside the arm.urdf file was modified as follows

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

And the arm.urdf file was renamed as arm.urdf.xacro
Now, in order to effectively use the macro, the following command was added inside the arm.urdf.xacro file

```
<xacro:arm_gazebo robot_name="$(arg robot_name)"/>
```

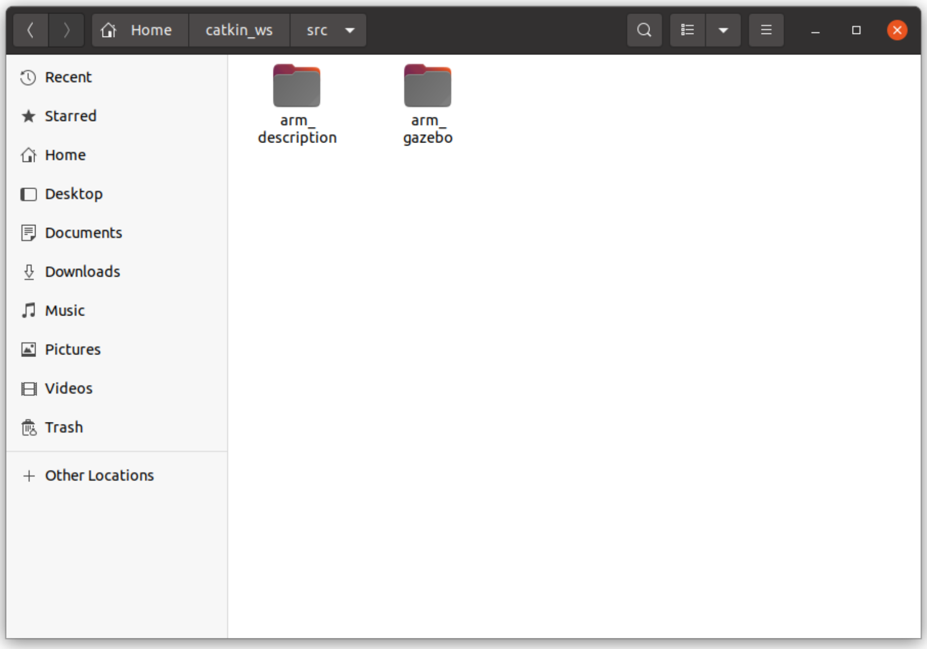
Finally, the display.launch must be changed as follows in order to convert the xacro file into an urdf file, and assign it to the robot_description parameter

```
<?xml version="1.0" ?>

<launch>

  <arg name="rvizconfig" default="$(find arm_description)/configuration.
     rviz"/>
  <arg name="robot_name" default="arm"/>

  <param name="robot_description" command="$(find xacro)/xacro --inorder
      '$(find arm_description)/urdf/arm.urdf.xacro'/>

  <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
      type="joint_state_publisher_gui" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="
     robot_state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)"
     required="true" />

</launch>
```

2. Add transmission and controllers to your robot and spawn it in Gazebo

(a) Create a package named arm_gazebo

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg arm_gazebo
```



(b) Within this package create a launch folder containing a arm_world.launch file

```
$ cd ~/catkin_ws/src/arm_gazebo
$ mkdir launch
$ cd launch
$ touch arm_world.launch
```

(c) Fill this launch file with commands that load the URDF into the ROS Parameter Server and spawn your robot using the spawn_model node. Hint: follow the iiwa_world.launch example from the package iiwa_stack: `https://github.com/IFLCAMP/iiwa_stack/tree/master`. Launch the arm_world.launch file to visualize the robot in Gazebo

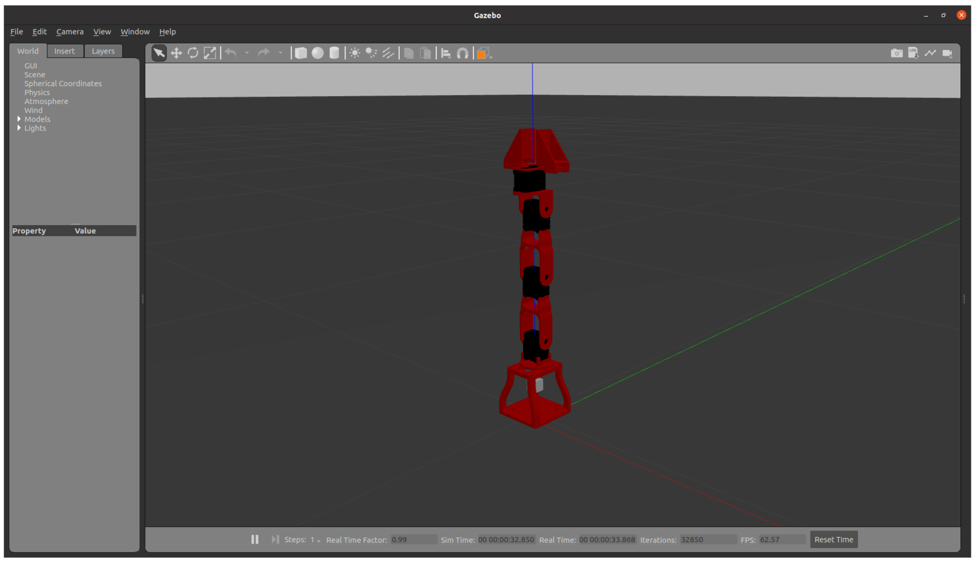
The arm_world.launch file was filled in this way

```
<?xml version="1.0" ?>

<launch>

  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="robot_name" default="arm" />
  <arg name="model" default="arm_model"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- Load the URDF into the ROS Parameter Server-->
  <param name="robot_description" command="$(find xacro)/xacro --inorder
      '$(find arm_description)/urdf/arm.urdf.xacro'/>

  <!-- Run a python script to send a service call to gazebo_ros to spawn
      a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn=
     "false" output="screen"
        args="-urdf -model arm -param robot_description"/>

</launch>
```

Now, we launch the arm_world.launch file to visualize the robot inside Gazebo

```
$ roslaunch arm_gazebo arm_world.launch
```

The visualization we obtain is as follows



We can observe that the robot's colors have been imported correctly, indicating that the steps taken in section 1.d have worked

(d) Now add a PositionJointInterface as hardware interface to your robot: create a arm.transmission.xacro file into your arm_description/urdf folder containing a xacro:macro with the hardware interface and load it into your arm.urdf.xacro file using xacro:include. Launch the file

Let's create the arm.trasmission.xacro file inside the arm_description/urdf folder

```
$ cd catkin_ws/src/arm_description/urdf
$ touch arm.trasmission.xacro
```

This file was filled with the following code

```
<?xml version="1.0"?>

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

  <xacro:macro name="arm_transmission" params="robot_name
     hardware_interface">

    <transmission name="${robot_name}_tran_0">
      <robotNamespace>/${robot_name}</robotNamespace>
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="j0">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
      </joint>
      <actuator name="${robot_name}_motor_0">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

    <transmission name="${robot_name}_tran_1">
      <robotNamespace>/${robot_name}</robotNamespace>
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="j1">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
      </joint>
      <actuator name="${robot_name}_motor_1">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

    <transmission name="${robot_name}_tran_2">
      <robotNamespace>/${robot_name}</robotNamespace>
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="j2">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
      </joint>
      <actuator name="${robot_name}_motor_2">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

    <transmission name="${robot_name}_tran_3">
      <robotNamespace>${robot_name}</robotNamespace>
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="j3">
        <hardwareInterface>hardware_interface/${hardware_interface}</
           hardwareInterface>
      </joint>
      <actuator name="${robot_name}_motor_3">
```

```
        <hardwareInterface>hardware_interface/${hardware_interface}</
            hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

  </xacro:macro>

</robot>
```

Meanwhile, the arm.urdf.xacro file was updated in the following way

```
...
<xacro:include filename="$(find arm_description)/urdf/arm.transmission.
   xacro"/>
...
<xacro:arm_transmission robot_name="$(arg robot_name)"
   hardware_interface="$(arg hardware_interface)"/>
...
```

(e) Add joint position controllers to your robot: create a arm_control package with a arm_control.launch file inside its launch folder and a arm_control.yaml file within its config folder

Let's create the arm_control package with its corresponding launch file inside the launch folder

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg arm_control
$ cd arm_control
$ mkdir launch
$ cd launch
$ touch arm_control.launch
```

And now, let's create the config folder with the arm_control.yaml file inside

```
$ cd ~/catkin_ws/src/arm_control
$ mkdir config
$ cd config
$ touch arm_control.yaml
```

(f) Fill the arm_control.launch file with commands that load the joint controller configurations from the .yaml file to the parameter server and spawn the controllers using the controller_manager package. Hint: follow the iiwa_control.launch example from corresponding package

The arm_control.launch file was filled in this way

```
<?xml version="1.0" ?>

<launch>

  <arg name="hardware_interface" default="PositionJointInterface"/>
  <arg name="controllers" default="joint_state_controller $(arg
     hardware_interface)_J0_controller $(arg hardware_interface)
     _J1_controller $(arg hardware_interface)_J2_controller $(arg
     hardware_interface)_J3_controller"/>
  <arg name="robot_name" default="arm"/>
  <arg name="model" default="arm_model"/>

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

  <!-- Load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner
     " respawn="false"
          output="screen" args="$(arg controllers)"/>

  <!-- Convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="
     robot_state_publisher"
    respawn="false" output="screen">
    <remap from="/joint_states" to="/$(arg robot_name)/joint_states"/>
  </node>

</launch>
```

(g) Fill the arm arm_control.yaml adding a joint_state_controller and a JointPositionController to all the joints

The arm_control.yaml file was configured in this way

```
#arm:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 100

  PositionJointInterface_J0_controller:
    type: position_controllers/JointPositionController
    joint: j0
    pid: {p: 100.0, i: 0.01, d: 10.0}
  PositionJointInterface_J1_controller:
    type: position_controllers/JointPositionController
    joint: j1
    pid: {p: 100.0, i: 0.01, d: 10.0}
  PositionJointInterface_J2_controller:
    type: position_controllers/JointPositionController
    joint: j2
    pid: {p: 100.0, i: 0.01, d: 10.0}
  PositionJointInterface_J3_controller:
    type: position_controllers/JointPositionController
    joint: j3
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

(h) Create an arm_gazebo.launch file into the launch folder of the arm_gazebo package loading the Gazebo world with arm_world.launch and spawning the controllers within arm_control.launch. Go to the arm_description package and add the gazebo_ros_control plugin to your main URDF into the arm.gazebo.xacro file. Launch the simulation and check if your controllers are correctly loaded

First of all, let's create the arm_gazebo.launch file inside the launch folder of the arm_gazebo package

```
$ cd catkin_ws/src/arm_gazebo/launch
$ touch arm_gazebo.launch
```

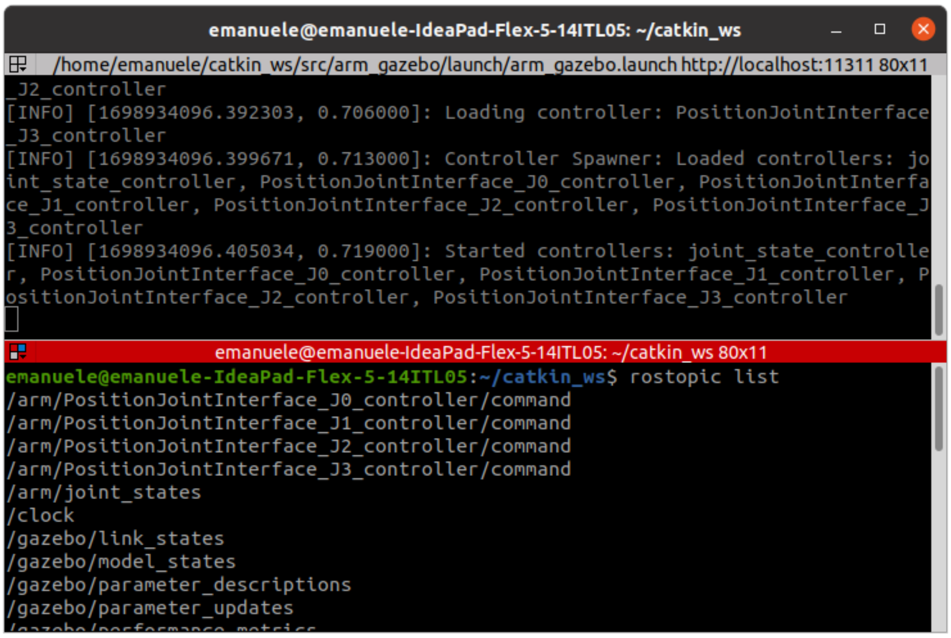
Inside this file, the following code has been inserted

```
<?xml version="1.0"?>

<launch>

    <arg name="hardware_interface" default="PositionJointInterface" />
    <arg name="robot_name" default="arm" />
    <arg name="model" default="arm_model"/>

    <include file="$(find arm_gazebo)/launch/arm_world.launch">
        <arg name="hardware_interface" value="$(arg hardware_interface)"
            />
        <arg name="robot_name" value="$(arg robot_name)" />
        <arg name="model" value="$(arg model)" />
    </include>

    <group ns="$(arg robot_name)">
        <include file="$(find arm_control)/launch/arm_control.launch">
            <arg name="hardware_interface" value="$(arg
               hardware_interface)" />
            <arg name="controllers" value="joint_state_controller
                 $(arg hardware_interface)_J0_controller
                 $(arg hardware_interface)_J1_controller
                 $(arg hardware_interface)_J2_controller
                 $(arg hardware_interface)_J3_controller"/>
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="model" value="$(arg model)" />
        </include>
    </group>

</launch>
```

At this point, let’s add the gazebo_ros_control plugin to the arm.gazebo.xacro file

```
<?xml version="1.0"?>

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

  <xacro:macro name="arm_gazebo" params="robot_name">

    <gazebo>
      <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.
         so">
        <robotNamespace>/${robot_name}</robotNamespace>
      </plugin>
    </gazebo>
...
```

Now let’s launch the arm_gazebo.launch file. First, by running rostopic list, we can see that all the necessary topics to move the robot are indeed present

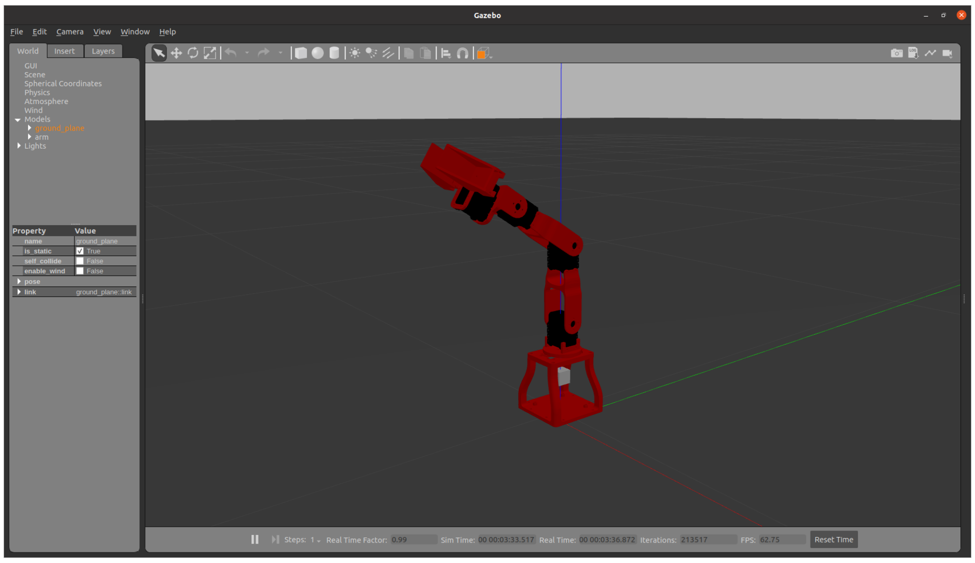


```
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws
/home/emanuele/catkin_ws/src/arm_gazebo/launch/arm_gazebo.launch http://localhost:11311 80x11
_J2_controller
[INFO] [1698934096.392303, 0.706000]: Loading controller: PositionJointInterface
_J3_controller
[INFO] [1698934096.399671, 0.713000]: Controller Spawner: Loaded controllers: jo
int_state_controller, PositionJointInterface_J0_controller, PositionJointInterfa
ce_J1_controller, PositionJointInterface_J2_controller, PositionJointInterface_J
3_controller
[INFO] [1698934096.405034, 0.719000]: Started controllers: joint_state_controlle
r, PositionJointInterface_J0_controller, PositionJointInterface_J1_controller, P
ositionJointInterface_J2_controller, PositionJointInterface_J3_controller

emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws 80x11
emanuele@emanuele-IdeaPad-Flex-5-14ITL05:~/catkin_ws$ rostopic list
/arm/PositionJointInterface_J0_controller/command
/arm/PositionJointInterface_J1_controller/command
/arm/PositionJointInterface_J2_controller/command
/arm/PositionJointInterface_J3_controller/command
/arm/joint_states
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
```

Now, let's try to move the robot with this command

```
$ rostopic pub /arm/PositionJointInterface_J2_controller/command
    std_msgs/Float64 1.0
```

Indeed, the robot moves correctly



3. Add a camera sensor to your robot

(a) Go into your arm.urdf.xacro file and add a camera_link and a fixed camera_joint with base_link as a parent link. Size and position the camera link opportunely

Let's update the arm.urdf.xacro file, adding a camera_link and a fixed camera_joint with base_-link as a parent link

```
...
  <joint name="camera_joint" type="fixed">
    <axis xyz="0 1 0"/>
    <origin xyz="0 0 0.02" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="camera_link"/>
  </joint>

  <link name="camera_link">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.02 0.02 0.02" />
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.02 0.02 0.02" />
      </geometry>
      <material name="Red"/>
    </visual>

    <inertial>
      <mass value="1e-5"/>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6"
         />
    </inertial>

  </link>
...
```

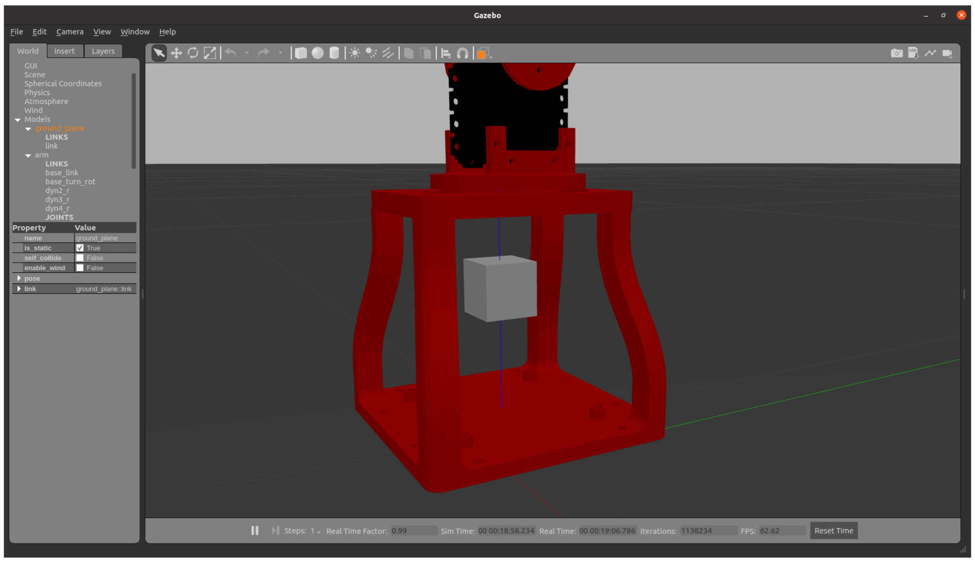
(b) In the arm.gazebo.xacro add the gazebo sensor reference tags and the libgazebo_ros_camera plugin to your xacro (slide 74-75)

```
...
    <gazebo reference="camera_link">
        <sensor type="camera" name="camera1">
            <update_rate>30.0</update_rate>
            <camera name="head">
            <horizontal_fov>1.3962634</horizontal_fov>
            <image>
                <width>800</width> <height>800</height> <format>
                R8G8B8</format>
            </image>
            <clip>
                <near>0.02</near> <far>300</far>
            </clip>
            <noise>
                <type>gaussian</type> <mean>0.0</mean> <stddev>0.007
                </stddev>
            </noise>
            </camera>

            <plugin name="camera_controller" filename="
               libgazebo_ros_camera.so">
                <alwaysOn>true</alwaysOn>
                <updateRate>0.0</updateRate>
                <cameraName>camera</cameraName>
                <imageTopicName>image_raw</imageTopicName>
                <cameraInfoTopicName>camera_info</cameraInfoTopicName>
                <frameName>camera_link_optical</frameName>
                <hackBaseline>0.0</hackBaseline>
                <distortionK1>0.0</distortionK1>
                <distortionK2>0.0</distortionK2>
                <distortionK3>0.0</distortionK3>
                <distortionT1>0.0</distortionT1>
                <distortionT2>0.0</distortionT2>
                <CxPrime>0</CxPrime>
                <Cx>0.0</Cx>
                <Cy>0.0</Cy>
                <focalLength>0.0</focalLength>
            </plugin>
        </sensor>
    </gazebo>
...
```

(c) Launch the Gazebo simulation with using arm_gazebo.launch and check if the image topic is correctly published using rqt_image_view

```
$ roslaunch arm_gazebo arm_gazebo.launch
```

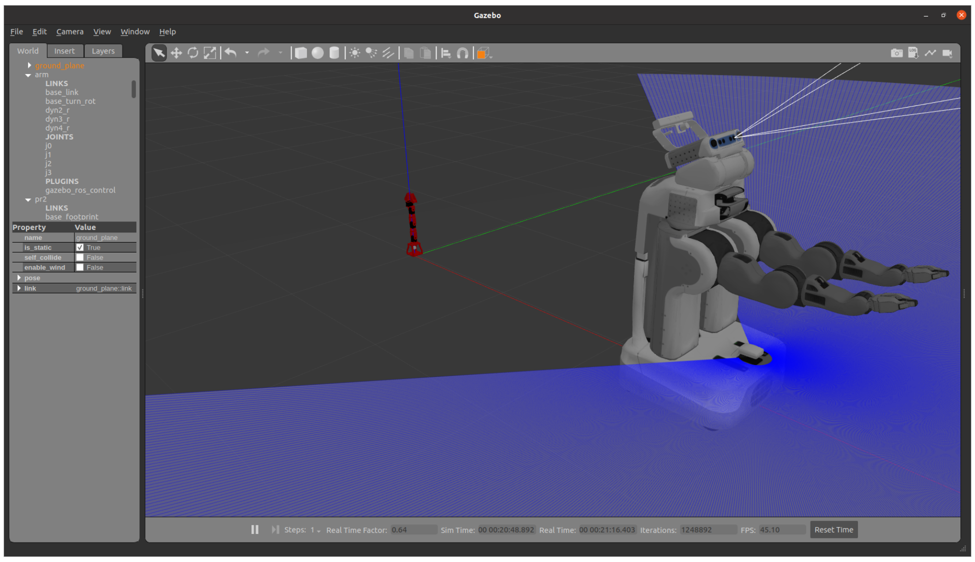
First of all, the camera is visible within the simulation



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wind	False
pose	
link	ground_plane::link

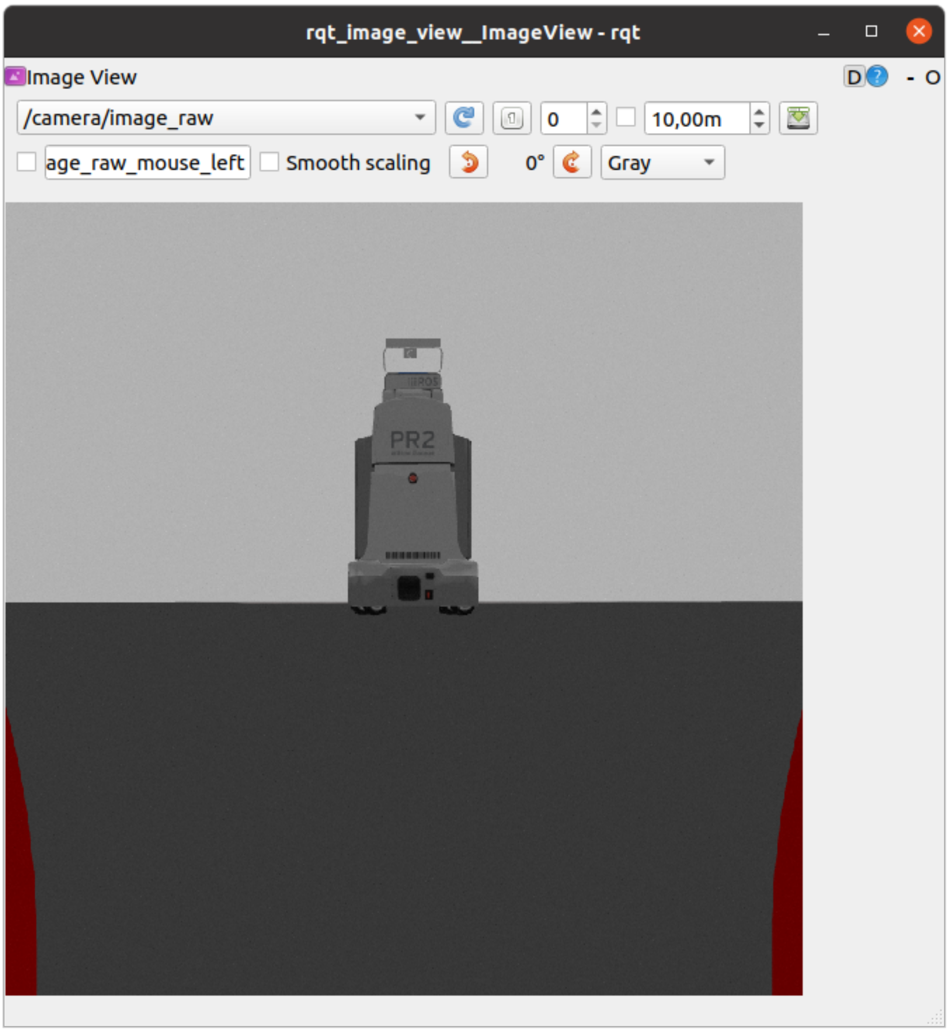
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:18:58.234 Real Time: 00 00:19:06.786 Iterations: 1138234 FPS: 62.62

Let's place an object in front of the camera, such as a PR2 robot



And let's visualize the image topic using the command

```
$ rqt_image_view
```



(d) Optionally: You can create a camera.xacro file (or download one from `https://github.com/CentroEPiaggio/irobotcreate2ros/blob/master/model/camera.urdf.xacro`) and add it to your robot URDF using <xacro:include>

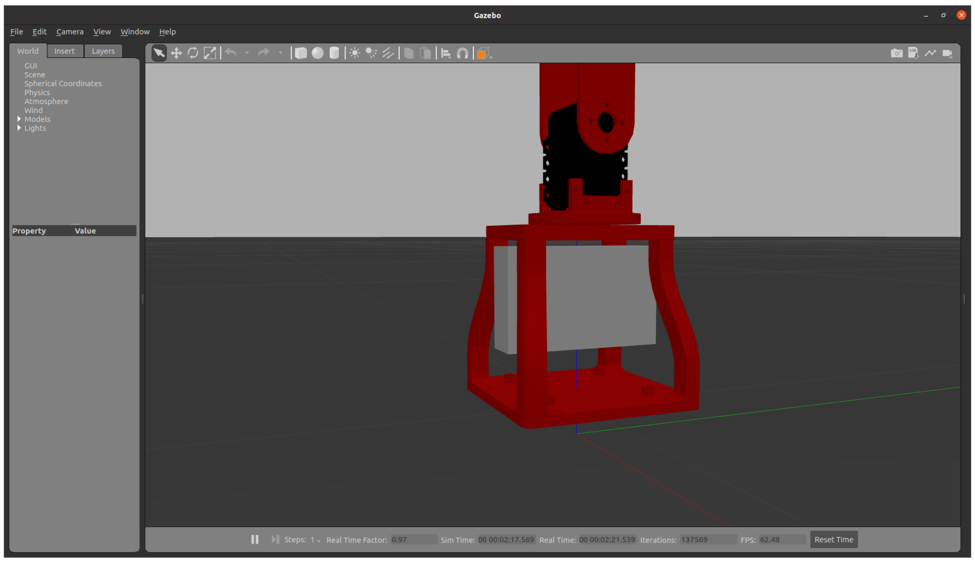
it was decided to directly import the camera.xacro file from the provided link. The imported code is shown below

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="camera">

    <xacro:property name="pi" value="3.1415926535897931"/>

    <xacro:macro name="camera_sensor" params="xyz rpy parent">
        <joint name="camera_sensor_joint" type="fixed">
            <axis xyz="0 1 0" />
            <origin xyz="${xyz}" rpy="${rpy}"/>
            <parent link="${parent}"/>
            <child link="camera_link"/>
        </joint>

        <link name="camera_link">
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.02 0.08 0.05"/>
                </geometry>
            </collision>
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <box size="0.02 0.08 0.05"/>
                </geometry>
                <material name="iRobot/Green"/>
            </visual>
            <inertial>
                <mass value="0.0001" />
                <origin xyz="0 0 0" rpy="0 0 ${pi}"/>
                <inertia ixx="0.0000001" ixy="0" ixz="0" iyy="0.0000001"
                    iyz="0" izz="0.0000001" />
            </inertial>
        </link>

        <gazebo reference="camera_link">
            <sensor type="camera" name="camera">
                <update_rate>30.0</update_rate>
                <camera name="head">
                    <horizontal_fov>1.3962634</horizontal_fov>
                    <image>
                        <width>800</width>
                        <height>600</height>
                        <format>R8G8B8</format>
                    </image>
                    <clip>
                        <near>0.02</near>
                        <far>300</far>
                    </clip>
                    <noise>
                        <type>gaussian</type>
                        <mean>0.0</mean>
                        <stddev>0.007</stddev>
                    </noise>
                </camera>
                <plugin name="camera_controller" filename="
                   libgazebo_ros_camera.so">
                    <alwaysOn>true</alwaysOn>
                    <updateRate>0.0</updateRate>
                    <cameraName>iRobot/camera</cameraName>
                    <imageTopicName>image_raw</imageTopicName>
                    <cameraInfoTopicName>camera_info</
                       cameraInfoTopicName>
                    <frameName>camera_link</frameName>
                    <hackBaseline>0.07</hackBaseline>
                    <distortionK1>0.0</distortionK1>
                    <distortionK2>0.0</distortionK2>
                    <distortionK3>0.0</distortionK3>
                    <distortionT1>0.0</distortionT1>
                    <distortionT2>0.0</distortionT2>
                </plugin>
            </sensor>
    </gazebo>

    </xacro:macro>

</robot>
```

Once this file was defined, the definition of the link and joint related to the camera was removed from the arm.urdf.xacro file, and the following two lines of code were added

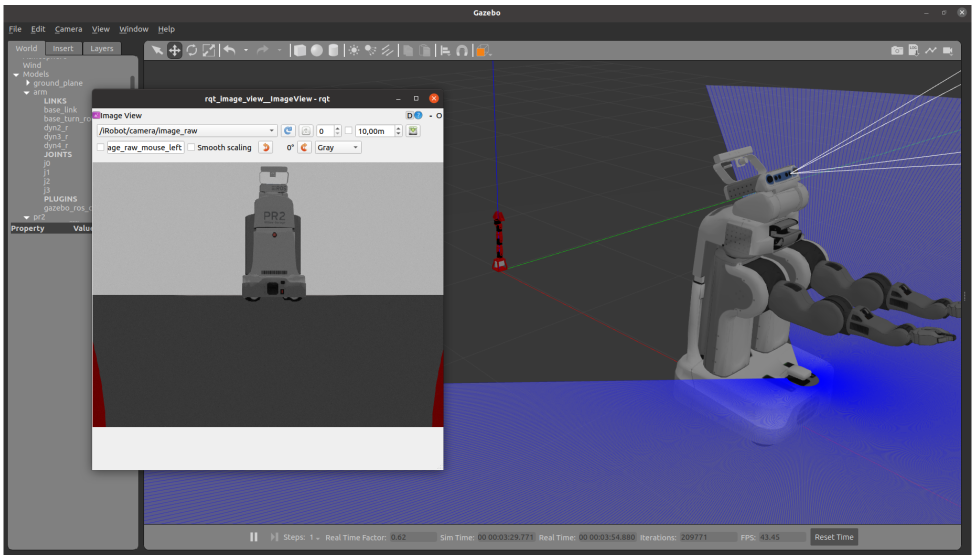
```
...
<xacro:include filename="$(find arm_description)/urdf/camera.xacro"/>
...
<xacro:camera_sensor xyz="0 0 0" rpy="0 0 0" parent="base_link"/>
...
```

From a graphical perspective, the new camera has the following appearance



And we can verify that it functions correctly, still using the command

```
$ rqt_image_view
```


Gazebo
File Edit Camera View Window Help
World Insert Layers
Wind
Models
ground_plane
arm
LINKS
base_link
JOINTS
PLUGINS
pr2
Property
rqt_image_view__ImageView - rqt
Image View
/iRobot/camera/image_raw
10,00m
Smooth scaling
Gray
PR2
Reset Time

4. Create a ROS publisher node that reads the joint state and sends joint position commands to your robot

(a) Create an arm_controller package with a ROS C++ node named arm_controller_node. The dependencies are roscpp, sensor_msgs and std_msgs. Modify opportunely the CMakeLists.txt file to compile your node. Hint: uncomment add_executable and target_link_libraries lines

```
$ ~/catkin_ws/src
$ catkin_create_pkg arm_controller roscpp sensor_msgs std_msgs
$ cd arm_controller/src
$ touch arm_controller_node.cpp
```

To compile the node correctly, we need to modify the Makefile, specifically, we must uncomment the following lines

```
...
add_executable(arm_controller_node src/arm_controller_node.cpp)
...
target_link_libraries(arm_controller_node ${catkin_LIBRARIES})
...
```

(b) Create a subscriber to the topic joint_states and a callback function that prints the current joint positions (see Slide 45). Note: the topic contains a sensor_msgs/JointState

Let's fill the node with the following code

```
#include <ros/ros.h>
#include <sensor_msgs/JointState.h>

void jointStateCallback(const sensor_msgs::JointState::ConstPtr&
   joint_state) {
    ROS_INFO("Received JointState message");

    for (size_t i = 0; i < joint_state->position.size(); ++i) {
        ROS_INFO("Position of joint %zu: %f", i, joint_state->position[i
           ]);
    }
}

int main(int argc, char* argv[]) {
    ros::init(argc, argv, "arm_controller_node");
    ros::NodeHandle nodeHandle;
    ros::Subscriber subscriber = nodeHandle.subscribe("arm/joint_states"
       , 10, jointStateCallback);
    ros::spin();
    return 0;
}
```

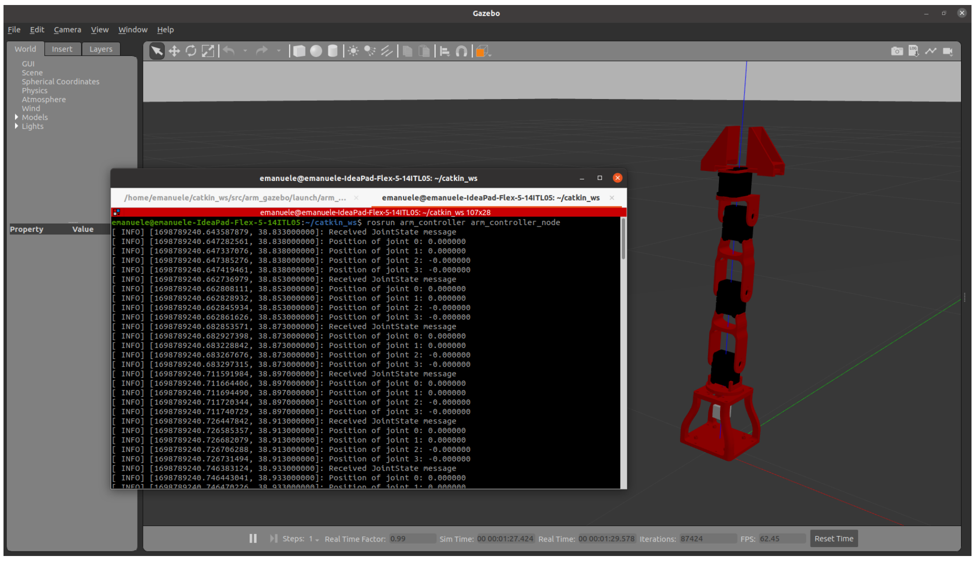
Let's verify its correct functioning. Launch the robot in Gazebo

```
$ roslaunch arm_gazebo arm_gazebo.launch
```

And run the node

```
$ rosrun arm_controller arm_controller_node
```

We notice how the values of the four joint variables are correctly printed on the terminal



```
emanuele@emanuele-IdeaPad-Flex-5-14ITL05:~/catkin_ws$ rosrun arm_controller arm_controller_node
[ INFO] [1698789240.643587879, 38.833000000]: Received JointState message
[ INFO] [1698789240.647282561, 38.838000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.647337076, 38.838000000]: Position of joint 1: 0.000000
[ INFO] [1698789240.647385276, 38.838000000]: Position of joint 2: -0.000000
[ INFO] [1698789240.647419461, 38.838000000]: Position of joint 3: -0.000000
[ INFO] [1698789240.662736979, 38.853000000]: Received JointState message
[ INFO] [1698789240.662808111, 38.853000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.662828932, 38.853000000]: Position of joint 1: 0.000000
[ INFO] [1698789240.662845934, 38.853000000]: Position of joint 2: -0.000000
[ INFO] [1698789240.662861626, 38.853000000]: Position of joint 3: -0.000000
[ INFO] [1698789240.682853571, 38.873000000]: Received JointState message
[ INFO] [1698789240.682927398, 38.873000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.683228842, 38.873000000]: Position of joint 1: 0.000000
[ INFO] [1698789240.683267676, 38.873000000]: Position of joint 2: -0.000000
[ INFO] [1698789240.683297315, 38.873000000]: Position of joint 3: -0.000000
[ INFO] [1698789240.711591984, 38.897000000]: Received JointState message
[ INFO] [1698789240.711664406, 38.897000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.711694490, 38.897000000]: Position of joint 1: 0.000000
[ INFO] [1698789240.711720344, 38.897000000]: Position of joint 2: -0.000000
[ INFO] [1698789240.711740729, 38.897000000]: Position of joint 3: -0.000000
[ INFO] [1698789240.726447842, 38.913000000]: Received JointState message
[ INFO] [1698789240.726585357, 38.913000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.726682079, 38.913000000]: Position of joint 1: 0.000000
[ INFO] [1698789240.726706288, 38.913000000]: Position of joint 2: -0.000000
[ INFO] [1698789240.726731494, 38.913000000]: Position of joint 3: -0.000000
[ INFO] [1698789240.746383124, 38.933000000]: Received JointState message
[ INFO] [1698789240.746443041, 38.933000000]: Position of joint 0: 0.000000
[ INFO] [1698789240.746470226, 38.933000000]: Position of joint 1: 0.000000
```

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws
/home/emanuele/catkin_ws/src/arm_gazebo/launch/arm_...
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws 107x28
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:01:27.424 Real Time: 00 00:01:29.578 Iterations: 87424 FPS: 62.45 Reset Time

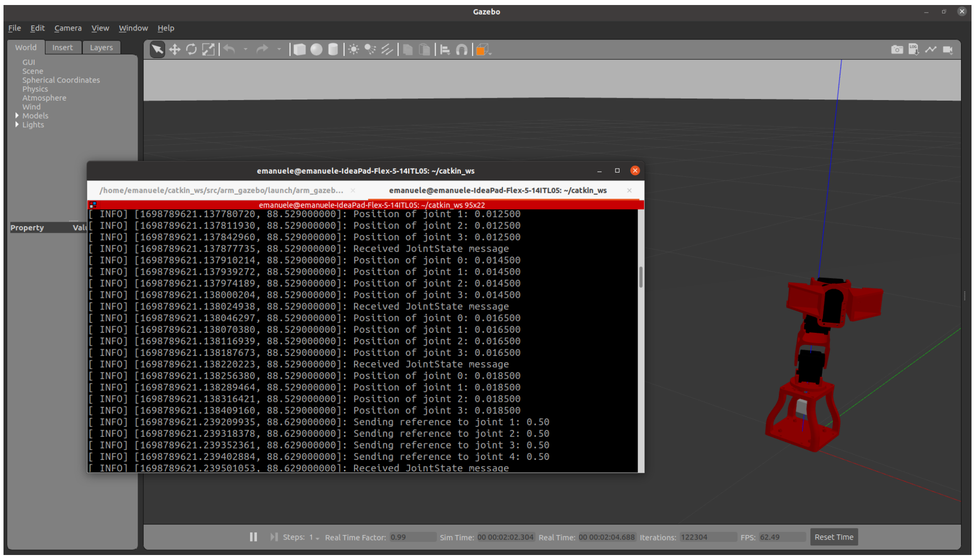
(c) Create publishers that write commands onto the controllers' /command topics (see Slide 46). Note: the command is a std_msgs/Float64

The arm_controller_node.cpp file is modified as shown. The following implementation is based on the assumption that the desired values of joints positions are assigned through the terminal

```
#include <ros/ros.h>
#include <std_msgs/Float64.h>
#include <sensor_msgs/JointState.h>

void jointStateCallback(const sensor_msgs::JointState::ConstPtr&
    joint_state) {
     ROS_INFO("Received JointState message");

     for (size_t i = 0; i < joint_state->position.size(); ++i) {
         ROS_INFO("Position of joint %zu: %f", i, joint_state->position[i
             ]);
     }
}

int main(int argc, char* argv[]) {
     ros::init(argc, argv, "arm_controller_node");
     ros::NodeHandle nh;

     if (argc != 5) {
         ROS_ERROR("Please insert the value of the reference");
         return 1;
     }

     ros::Publisher jointPublisher1 = nh.advertise<std_msgs::Float64>("/
        arm/PositionJointInterface_J0_controller/command", 1);
     ros::Publisher jointPublisher2 = nh.advertise<std_msgs::Float64>("/
        arm/PositionJointInterface_J1_controller/command", 1);
     ros::Publisher jointPublisher3 = nh.advertise<std_msgs::Float64>("/
        arm/PositionJointInterface_J2_controller/command", 1);
     ros::Publisher jointPublisher4 = nh.advertise<std_msgs::Float64>("/
        arm/PositionJointInterface_J3_controller/command", 1);
    double joint1_position = std::stod(argv[1]);
    double joint2_position = std::stod(argv[2]);
    double joint3_position = std::stod(argv[3]);
    double joint4_position = std::stod(argv[4]);

    ros::Subscriber subscriber = nh.subscribe("arm/joint_states", 10,
       jointStateCallback);
    ros::Rate loopRate(10);

    while (ros::ok()) {
        std_msgs::Float64 jointCommand1, jointCommand2, jointCommand3,
           jointCommand4;
        jointCommand1.data = joint1_position;
        jointCommand2.data = joint2_position;
        jointCommand3.data = joint3_position;
        jointCommand4.data = joint4_position;

        ROS_INFO("Sending reference to joint 1: %.2f", jointCommand1.
           data);
        ROS_INFO("Sending reference to joint 2: %.2f", jointCommand2.
           data);
        ROS_INFO("Sending reference to joint 3: %.2f", jointCommand3.
           data);
        ROS_INFO("Sending reference to joint 4: %.2f", jointCommand4.
           data);

        jointPublisher1.publish(jointCommand1);
        jointPublisher2.publish(jointCommand2);
        jointPublisher3.publish(jointCommand3);
        jointPublisher4.publish(jointCommand4);

        ros::spinOnce();
        loopRate.sleep();
    }

    return 0;
}
```

Let’s verify that the node functions correctly

```
$ rosrun arm_controller arm_controller_node 0.5 0.5 0.5 0.5
```


Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property
Valu
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws
/home/emanuele/catkin_ws/src/arm_gazebo/launch/arm_gazeb...
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws
emanuele@emanuele-IdeaPad-Flex-5-14ITL05: ~/catkin_ws 95x22
Steps: 1
Real Time Factor: 0.99
Sim Time: 00 00:02:02.304
Real Time: 00 00:02:04.688
Iterations: 122304
FPS: 62.49
Reset Time

```
[ INFO] [1698789621.137780720, 88.529000000]: Position of joint 1: 0.012500
[ INFO] [1698789621.137811930, 88.529000000]: Position of joint 2: 0.012500
[ INFO] [1698789621.137842960, 88.529000000]: Position of joint 3: 0.012500
[ INFO] [1698789621.137877735, 88.529000000]: Received JointState message
[ INFO] [1698789621.137910214, 88.529000000]: Position of joint 0: 0.014500
[ INFO] [1698789621.137939272, 88.529000000]: Position of joint 1: 0.014500
[ INFO] [1698789621.137974189, 88.529000000]: Position of joint 2: 0.014500
[ INFO] [1698789621.138000204, 88.529000000]: Position of joint 3: 0.014500
[ INFO] [1698789621.138024938, 88.529000000]: Received JointState message
[ INFO] [1698789621.138046297, 88.529000000]: Position of joint 0: 0.016500
[ INFO] [1698789621.138070380, 88.529000000]: Position of joint 1: 0.016500
[ INFO] [1698789621.138116939, 88.529000000]: Position of joint 2: 0.016500
[ INFO] [1698789621.138187673, 88.529000000]: Position of joint 3: 0.016500
[ INFO] [1698789621.138220223, 88.529000000]: Received JointState message
[ INFO] [1698789621.138256380, 88.529000000]: Position of joint 0: 0.018500
[ INFO] [1698789621.138289464, 88.529000000]: Position of joint 1: 0.018500
[ INFO] [1698789621.138316421, 88.529000000]: Position of joint 2: 0.018500
[ INFO] [1698789621.138409160, 88.529000000]: Position of joint 3: 0.018500
[ INFO] [1698789621.239209935, 88.629000000]: Sending reference to joint 1: 0.50
[ INFO] [1698789621.239318378, 88.629000000]: Sending reference to joint 2: 0.50
[ INFO] [1698789621.239352361, 88.629000000]: Sending reference to joint 3: 0.50
[ INFO] [1698789621.239402884, 88.629000000]: Sending reference to joint 4: 0.50
[ INFO] [1698789621.239501053, 88.629000000]: Received JointState message
```