Robotics Lab: Homework 1

Building your robot manipulator

Giuseppe Saggese
P38000213

Building your robot manipulator

The goal of this homework is to build ROS packages to simulate a robotic manipulator arm into the Gazebo environment.

All the files used in this report are available at the following GitHub link: https://github.com/peppesagg/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:

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src$ cd arm
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ git clone https://github.com/RoboticsLab2023/arm_description.git
Clone in 'arm_description' in corso...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 23 (delta 2), reused 23 (delta 2), pack-reused 0
Decompressione degli oggetti in corso: 100% (23/23), 1.13 MiB | 1.55 MiB/s, fatto.
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$
```

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 change the Fixed Frame in the lateral bar and add the RobotModel plugin interface. Optional: save a .rviz configuration file, then automatically loads the RobotModel plugin by default, and give it as an argument to your node in the display.launch file:

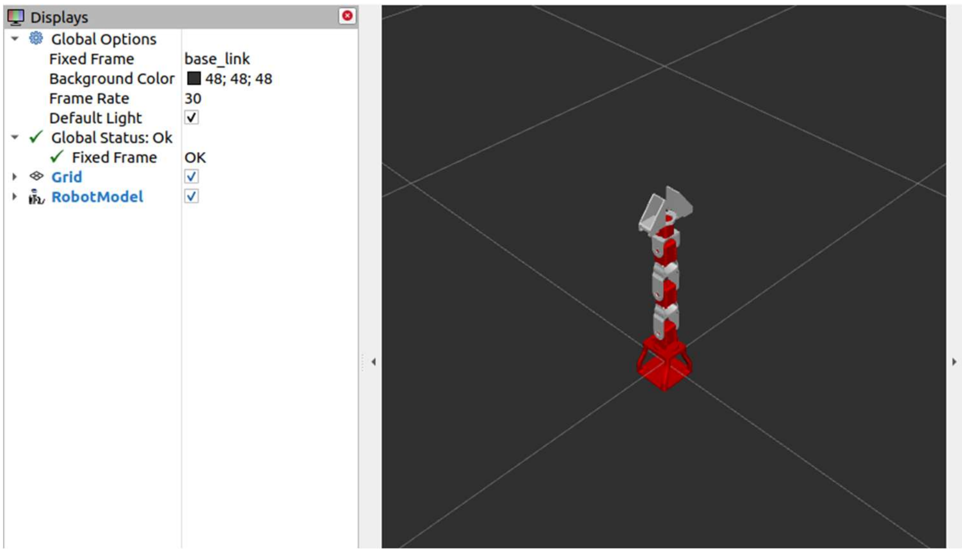
I created the xacro file arm.urdf.xacro adding the string xmlns:xacro="http://www.ros.org/wiki/xacro" within the <robot> tag to the arm.urdf file

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

I created a launch folder, in the arm_description package, containing the following launch file display.launch:

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

  <!-- This lauch file just loads the URDF with the given hardware interface and robot name into the ROS Parameter Server -->
  <arg name="hardware_interface" default="PositionJointInterface"/>
  <arg name="robot_name" default="arm"/>
  <arg name="origin_xyz" default="'0 0 0'"/> <!-- Note the syntax to pass a vector -->
  <arg name="origin_rpy" default="'0 0 0'"/>

  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find arm_description)/urdf/arm.urdf.xacro'
  hardware_interface:=$(arg hardware_interface) robot_name:=$(arg robot_name) origin_xyz:=$(arg origin_xyz) origin_rpy:=$(arg origin_rpy)"/>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" output="screen"/>
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" output="screen"/>
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find arm_description)/rviz/arm_config.rviz"/>
</launch>
```

This file load the urdf.xacro file as a robot_description ROS param and starts the robot_state_publisher node, the joint_state_publisher node and the rviz node. Then I launched this file and I saved the rviz configuration in a file named arm_config.rviz that I gived as an argument to the rviz node.

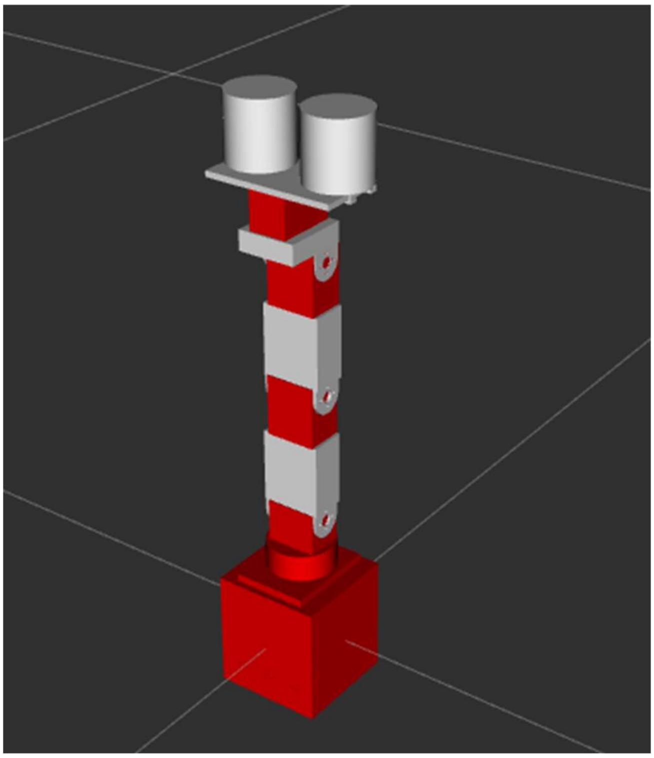


c. **Substitute the collision meshes of your URDF with primitive shapes. Use geometries of reasonabe size approximating the links. Hint: Enable collision visualization in rviz (go to the lateral bar > Robot model > Collision Enabled) to adjust the collision meshes size.**

In the URDF file I changed the collision meshes substituting them with the <box> and <cylinder> tags and I chose the right dimensions of the shapes for every link. For example, for the base_link I wrote:

```
<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="0.09 0.09 0.0975"/>
    </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>
```

The final result is:



d. Create a file named arm.gazebo.xacro within your package, define a xacro:macro inside your file containing all the 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 tag, and load the URDF in your launch file using the xacro routine.

I have already renamed the arm.urdf file in arm.urdf.xacro file in the point a.
I created an arm.gazebo.xacro file containing all the <gazebo> tags of the URDF file:

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

<robot xmlns:xacro="http://www.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>
```

Then I included this file in the arm.urdf.xacro file:

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

and in the same file I used this code string:

```
<xacro:arm_gazebo robot_name="arm"/>
```

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

a. Create a package named arm_gazebo

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ catkin_create_pkg arm_gazebo
Created file arm_gazebo/package.xml
Created file arm_gazebo/CMakeLists.txt
Successfully created files in /home/giuseppe/catkin_ws/src/arm/arm_gazebo. Please adjust the values in package.xml.
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$
```

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

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ cd arm_gazebo
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo$ mkdir launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo$ cd launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo/launch$ touch arm_world.launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo/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/IFL-CAMP/iiwa_stack/tree/master. Launch the arm_world.launch file to visualize the robot in Gazebo

In order to fill the arm_world.launch file I created a .world file within a worlds folder and the arm_upload.launch file simillary to the display.launch without the three nodes that I previously created. This file is necessary to load the URDF file without launch the nodes.

arm.world:

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <!-- We use a custom world for the arm so that the camera angle is launched correctly. -->
  <!-- One can change this world to his needs or use another one. -->

  <world name="default">

    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- Global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <physics name="default_physics" default="0" type="ode">
      <max_step_size>0.01</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>100</real_time_update_rate>
      <ode>
        <solver>
          <type>quick</type>
          <iters>50</iters>
          <sor>1.0</sor> <!-- Important, see issue #2209 -->
          <use_dynamic_moi_rescaling>false</use_dynamic_moi_rescaling>
        </solver>
      </ode>
    </physics>

    <!-- Focus camera -->
    <gui fullscreen='0'>
      <camera name='user_camera'>
    <pose>4.927360 -4.376610 3.740080 0.000000 0.275643 2.356190</pose>
    <view_controller>orbit</view_controller>
      </camera>
    </gui>

  </world>

</sdf>
```

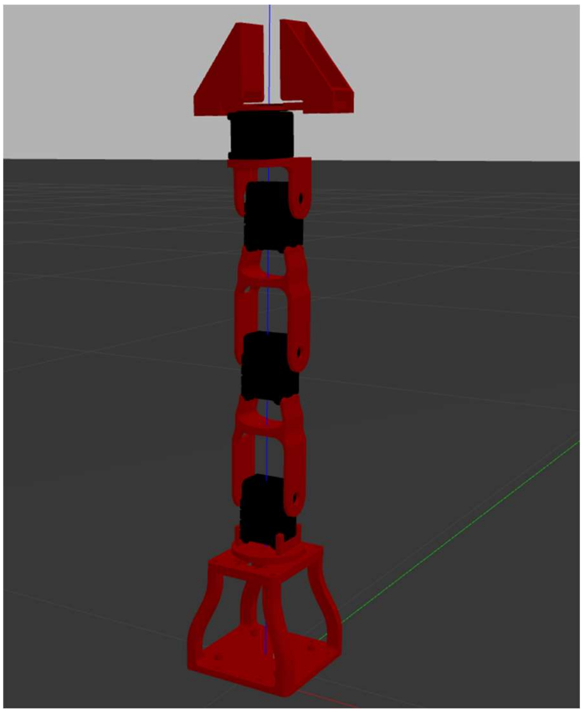
arm_upload.launch:

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

  <!-- This lauch file just loads the URDF with the given hardware interface and robot name into the ROS Parameter Server -->
  <arg name="hardware_interface" default="PositionJointInterface"/>
  <arg name="robot_name" default="arm"/>
  <arg name="origin_xyz" default="'0 0 0'"/> <!-- Note the syntax to pass a vector -->
  <arg name="origin_rpy" default="'0 0 0'"/>

  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find arm_description)/urdf/arm.urdf.xacro'
  hardware_interface:=$(arg hardware_interface) robot_name:=$(arg robot_name) origin_xyz:=$(arg origin_xyz) origin_rpy:=$(arg origin_rpy)"/>
</launch>
```

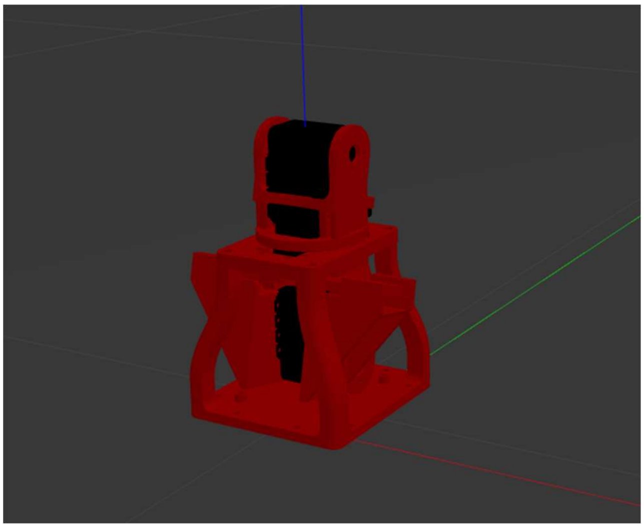
arm_world.launch:

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


    <!-- These are the arguments you can pass this launch file, for example paused:=true -->
    <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="hardware_interface" default="PositionJointInterface"/>
    <arg name="robot_name" default="arm" />
    <arg name="model" default="arm"/>

      <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
      <include file="$(find gazebo_ros)/launch/empty_world.launch">
            <arg name="world_name" value="$(find arm_gazebo)/worlds/arm.world"/>
            <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 with the given hardware interface into the ROS Parameter Server -->
     <include file="$(find arm_description)/launch/$(arg model)_upload.launch">
        <arg name="hardware_interface" value="$(arg hardware_interface)"/>
        <arg name="robot_name" value="$(arg robot_name)" />
    </include>


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


    </launch>
```

I launched the arm_world.launch file to visualize the robot in Gazebo:



After few minutes the robot collapsed on itself:



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

I created the arm.transmission.xacro file:

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

        <transmission name="$(arg robot_name)_tran_1">
        <robotNamespace>/$(arg robot_name)</robotNamespace>
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="j0">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
        </joint>
        <actuator name="$(arg robot_name)_motor_1">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        </transmission>

        <transmission name="$(arg robot_name)_tran_2">
        <robotNamespace>/$(arg robot_name)</robotNamespace>
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="j1">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
        </joint>
        <actuator name="$(arg robot_name)_motor_2">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        </transmission>

        <transmission name="$(arg robot_name)_tran_3">
        <robotNamespace>/$(arg robot_name)</robotNamespace>
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="j2">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
        </joint>
        <actuator name="$(arg robot_name)_motor_3">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        </transmission>

        <transmission name="$(arg robot_name)_tran_4">
        <robotNamespace>/$(arg robot_name)</robotNamespace>
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="j3">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
        </joint>
        <actuator name="$(arg robot_name)_motor_4">
            <hardwareInterface>hardware_interface/$(arg hardware_interface)</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        </transmission>

    </xacro:macro>
</robot>
```

Then I included this file in the arm.urdf.xacro file:

```xml
<!--Include arm.transmission.xacro -->
<xacro:include filename="$(find arm_description)/urdf/arm.transmission.xacro"/>
```

and in the same file I added the following line:

```xml
<xacro:arm_transmission hardware_interface="PositionJointInterface"/>
```

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

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ catkin_create_pkg arm_control
Created file arm_control/package.xml
Created file arm_control/CMakeLists.txt
Successfully created files in /home/giuseppe/catkin_ws/src/arm/arm_control. Please adjust the values in package.xml.
```

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control$ mkdir launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control$ mkdir config
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control$ cd launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control/launch$ touch arm_control.launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control/launch$ cd ..
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control$ cd config/
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control/config$ touch arm_control.yaml
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_control/config$
```

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

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

    <!-- Launches the controllers according to the hardware interface selected -->
    <!-- Everythings is spawned under a namespace with the same name as the robot's. -->

    <arg name="hardware_interface" default="PositionJointInterface"/>
    <arg name="controllers" default="joint_state_controller PositionJointInterface"/>
    <arg name="robot_name" default="arm" />
    <arg name="model" default="arm" />
    <arg name="joint_state_frequency" default="100" />
    <arg name="robot_state_frequency" default="100" />

    <!-- Loads joint controller configurations from YAML file to parameter server -->
    <rosparam file="$(find arm_control)/config/arm_control.yaml" command="load" />
    <param name="/$(arg robot_name)/joint_state_controller/publish_rate" value="$(arg joint_state_frequency)" />

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

    <!-- Converts 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="/$(arg robot_name)/joint_states" />
        <param name="publish_frequency" value="$(arg robot_state_frequency)" />
    </node>

</launch>
```

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

```yaml
#arm:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
```

```yaml
# Controllers for singular joint -----------------------------------
#
# Effort Position Controllers --------------------------------------

# VALUES ARE NOT CORRECT !
EffortJointInterface_J1_controller:
  type: effort_controllers/JointPositionController
  joint: j0
  pid: {p: 800.0, i: 100, d: 80.0, i_clamp_min: -10000, i_clamp_max: 10000}

EffortJointInterface_J2_controller:
  type: effort_controllers/JointPositionController
  joint: j1
  pid: {p: 800.0, i: 1000, d: 100.0, i_clamp_min: -10000, i_clamp_max: 10000}

EffortJointInterface_J3_controller:
  type: effort_controllers/JointPositionController
  joint: j2
  pid: {p: 800.0, i: 10, d: 5.0, i_clamp_min: -10000, i_clamp_max: 10000}

EffortJointInterface_J4_controller:
  type: effort_controllers/JointPositionController
  joint: j3
  pid: {p: 800.0, i: 10, d: 80.0, i_clamp_min: -10000, i_clamp_max: 10000}


# Forward Position Controllers -------------------------------------
PositionJointInterface_J1_controller:
  type: position_controllers/JointPositionController
  joint: j0

PositionJointInterface_J2_controller:
  type: position_controllers/JointPositionController
  joint: j1

PositionJointInterface_J3_controller:
  type: position_controllers/JointPositionController
  joint: j2

```

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

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws$ cd src/arm/arm_gazebo/launch/
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo/launch$ touch arm_gazebo.launch
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_gazebo/launch$ 
```

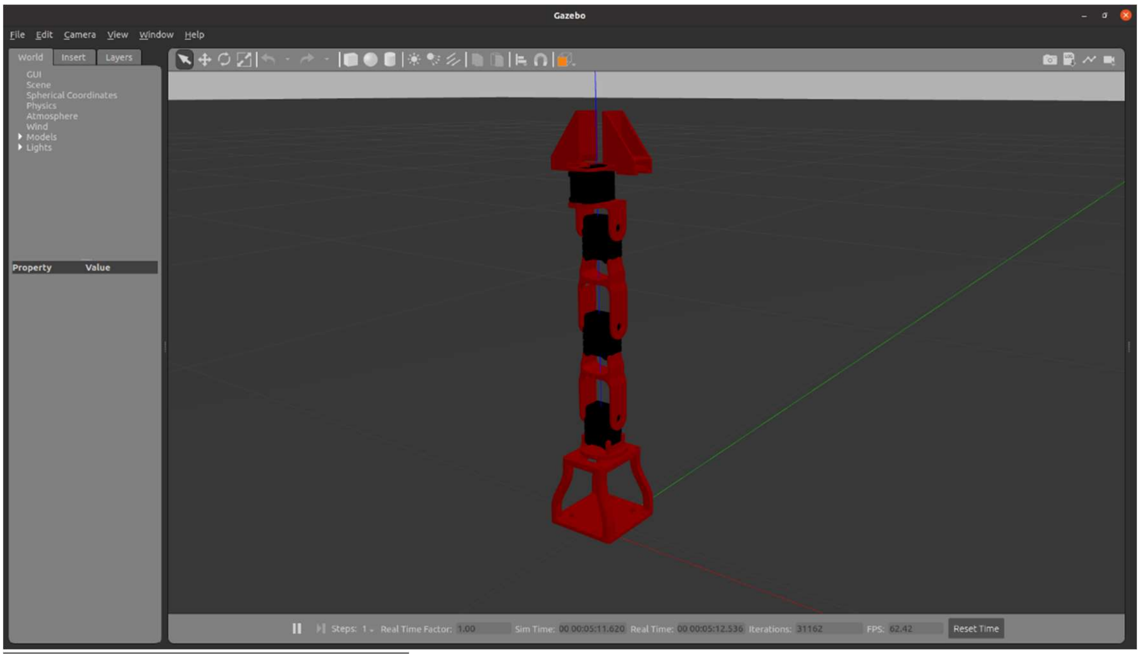
The arm_gazebo.launch file is:

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

    <!-- ==================================================================================== -->
    <!-- |    Lauch file to start Gazebo with an ARM using various controllers.          | -->

    <!-- |    It allows to customize the name of the robot, for each robot                   | -->
    <!-- |    its topics will be under a nameespace with the same name as the robot's.       | -->

    <!-- |    One can choose to have a joint trajectory controller or                        | -->
    <!-- |    controllers for the single joints, using the "trajectory" argument.            | -->
    <!-- ==================================================================================== -->

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

    <!-- Loads the Gazebo world. -->
    <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>

    <!--Spawn controllers - it uses a JointTrajectoryController >
    <group  ns="$(arg robot_name)" if="$(arg trajectory)">

        <include file="$(find arm_control)/launch/arm_control.launch">
            <arg name="hardware_interface" value="$(arg hardware_interface)" />
            <arg name="controllers" value="joint_state_controller $(arg hardware_interface)_trajectory_controller" />
            <arg name="robot_name" value="$(arg robot_name)" />
            <arg name="model" value="$(arg model)" />
        </include>

    </group-->

    <!-- Spawn controllers - it uses an Effort Controller for each joint -->
    <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>
```

In order to add the gazebo_ros_control plug in I added the following lines into the arm.gazebo.xacro file:

```xml
<!-- Plugin -->
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/$(arg robot_name)</robotNamespace>
  </plugin>
</gazebo>
```

then I launched the arm_gazebo.launch file and the result was:



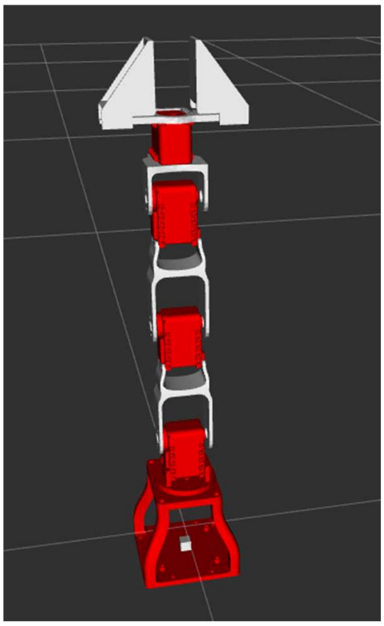
After 5 minutes of simulation the arm robot was still in the initial position.

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**

In order to add the camera to the arm I added this lines into the arm.urdf.xacro file that allow to create a camera_link and a camera_joint:

```
<!-- Add camera_link-->
 <link name="camera_link">
   <visual>
     <geometry>
       <box size="0.01 0.01 0.01"/>
     </geometry>
     <origin rpy="0 0 0" xyz="0 0 -0.02"/>
     <material name="white"/>
   </visual>
   <collision>
     <origin rpy="0 0 0" xyz="0 0 0"/>
     <geometry>
       <box size="0.01 0.01 0.01"/>
     </geometry>
   </collision>
 </link>

 <!-- Add camera_joint -->
   <joint name="camera_joint" type="fixed">
   <parent link="base_link"/>
   <child link="camera_link"/>
   <origin xyz="0 0 0" rpy="0 0 0"/>
 </joint>
```

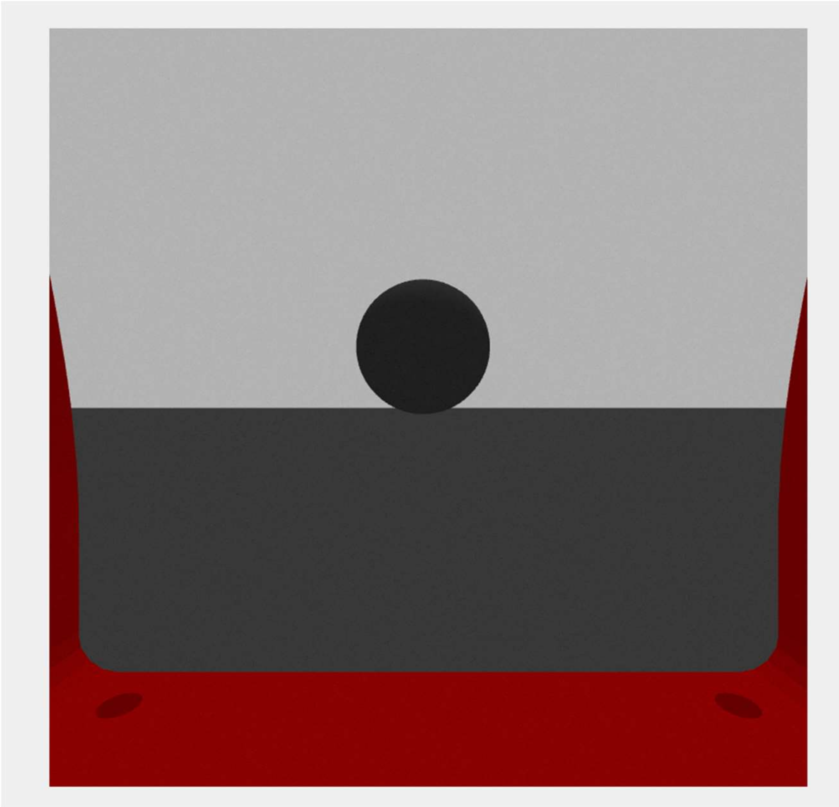


b. **In the arm.gazebo.xacro add the gazebo sensor reference tags and the libgazebo_ros_camera plugin to your xacro (slide 74-75)**

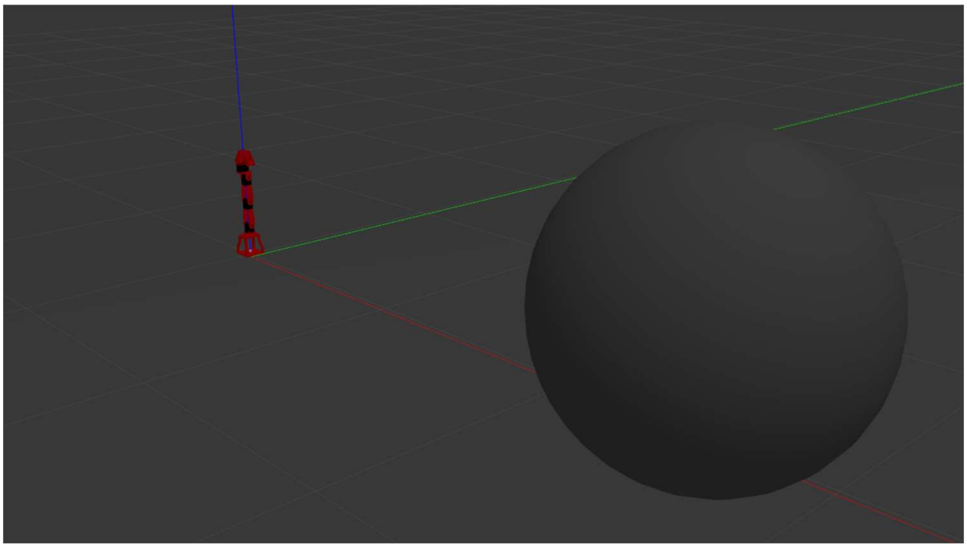
I added these lines into the arm.gazebo.xacro file:

```
<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">
    </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



I added a sphere in Gazebo to check if the camera correctly worked



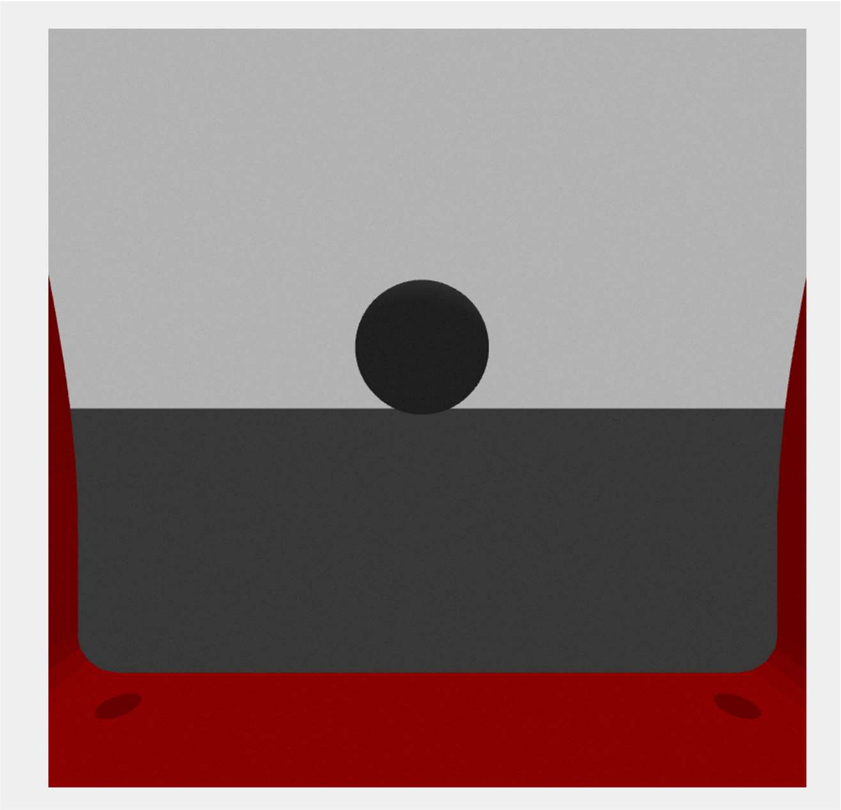
d. **Optionally**: **You can create a camera.xacro file (or download one from https://github.com/CentroEPiaggio/irobotcreate2ros/blob/master/model/camera.urdf.xacro) and add it to your robot URDF using <xacro:include>**

I added the camera.urdf.xacro file (that I downloaded from GitHub) in the urdf folder of the arm_description package so I wrote these lines in the URDF file:

```
<!--Include carmera.urdf.xacro-->
<xacro:include filename="$(find arm_description)/urdf/camera.urdf.xacro"/>
```

```
<xacro:camera_sensor xyz="0 0 0" rpy="0 0 0" parent="base_link" />
```

and I obtained the same results of the previously point:



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**

```
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ catkin_create_pkg arm_controller roscpp sensor_msgs std_msgs
Created file arm_controller/package.xml
Created file arm_controller/CMakeLists.txt
Created folder arm_controller/include/arm_controller
Created folder arm_controller/src
Successfully created files in /home/giuseppe/catkin_ws/src/arm/arm_controller. Please adjust the values in package.xml.
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm$ cd arm_controller/src/
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_controller/src$ touch arm_controller_node.cpp
giuseppe@HP-Pavilion-Laptop-15-cs0xxx:~/catkin_ws/src/arm/arm_controller/src$
```

I uncommented the add_executable and target_link_libraries lines of the CMakeList.txt file

```
add_executable(${PROJECT_NAME}_node src/arm_controller_node.cpp)
```

```
target_link_libraries(${PROJECT_NAME}_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**

I created a subscriber node in the arm_controller_node.cpp file with a callback function that prints the current joint positions of every joint:

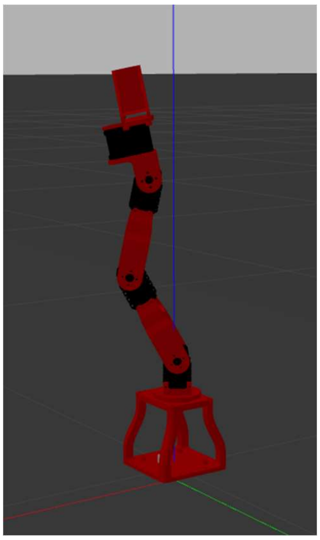
```cpp
#include <ros/ros.h>
#include <sensor_msgs/JointState.h>


void printJointStates(const sensor_msgs::JointState::ConstPtr& joint_states)
{
    // Print the current joint positions
    ROS_INFO("Current Joint Positions:");
    for (size_t i = 0; i < joint_states->name.size(); i++)
    {
        ROS_INFO("Joint Name: %s, Position: %f", joint_states->name[i].c_str(), joint_states->position[i]);
    }
}

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

    ros::Subscriber joint_states_sub = nh.subscribe("/arm/joint_states", 10, printJointStates);

    ros::spin();
    return 0;
}
```

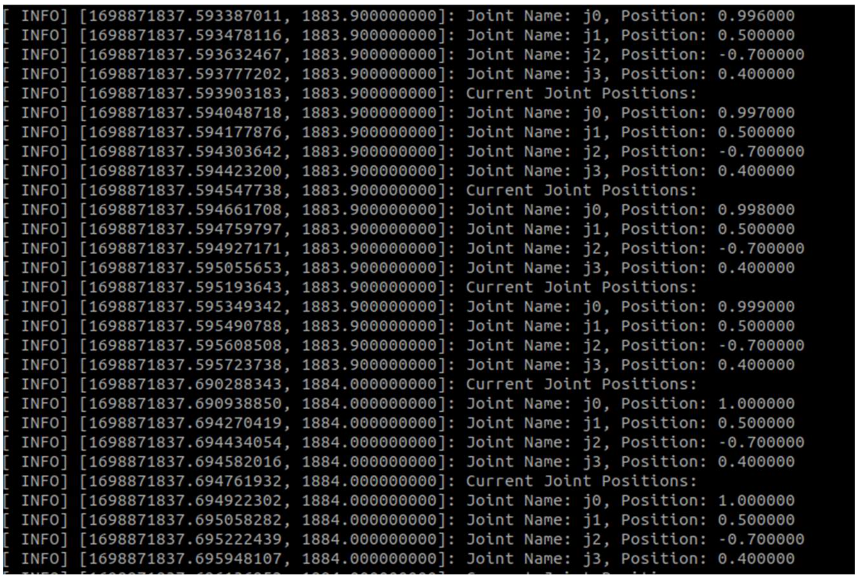
c. **Create publishers that write commands onto the controllers' /command topics (see Slide 46). Note: the command is a std_msgs/Float64**

In the same file I added 4 publischers (one for each joint) that write commands onto the controllers' /command, so the final C++ code is:

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


void printJointStates(const sensor_msgs::JointState::ConstPtr& joint_states)
{
    // Print the current joint positions
    ROS_INFO("Current Joint Positions:");
    for (size_t i = 0; i < joint_states->name.size(); i++)
    {
        ROS_INFO("Joint Name: %s, Position: %f", joint_states->name[i].c_str(), joint_states->position[i]);
    }
}

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

    ros::Subscriber joint_states_sub = nh.subscribe("/arm/joint_states", 10, printJointStates);

    ros::Publisher j0_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J0_controller/command", 10);
    ros::Publisher j1_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J1_controller/command", 10);
    ros::Publisher j2_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J2_controller/command", 10);
    ros::Publisher j3_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J3_controller/command", 10);

    ros::Rate loop_rate(10);

    while (ros::ok()) {
        // Publish commands to the controllers' /command topics
        std_msgs::Float64 j0_command;
        j0_command.data = 1;
        j0_pub.publish(j0_command);

        std_msgs::Float64 j1_command;
        j1_command.data = 0.5;
        j1_pub.publish(j1_command);

        std_msgs::Float64 j2_command;
        j2_command.data = -0.7;
        j2_pub.publish(j2_command);

        std_msgs::Float64 j3_command;
        j3_command.data = 0.4;
        j3_pub.publish(j3_command);

        ros::spinOnce();
        loop_rate.sleep();

    }

    return 0;
}
```

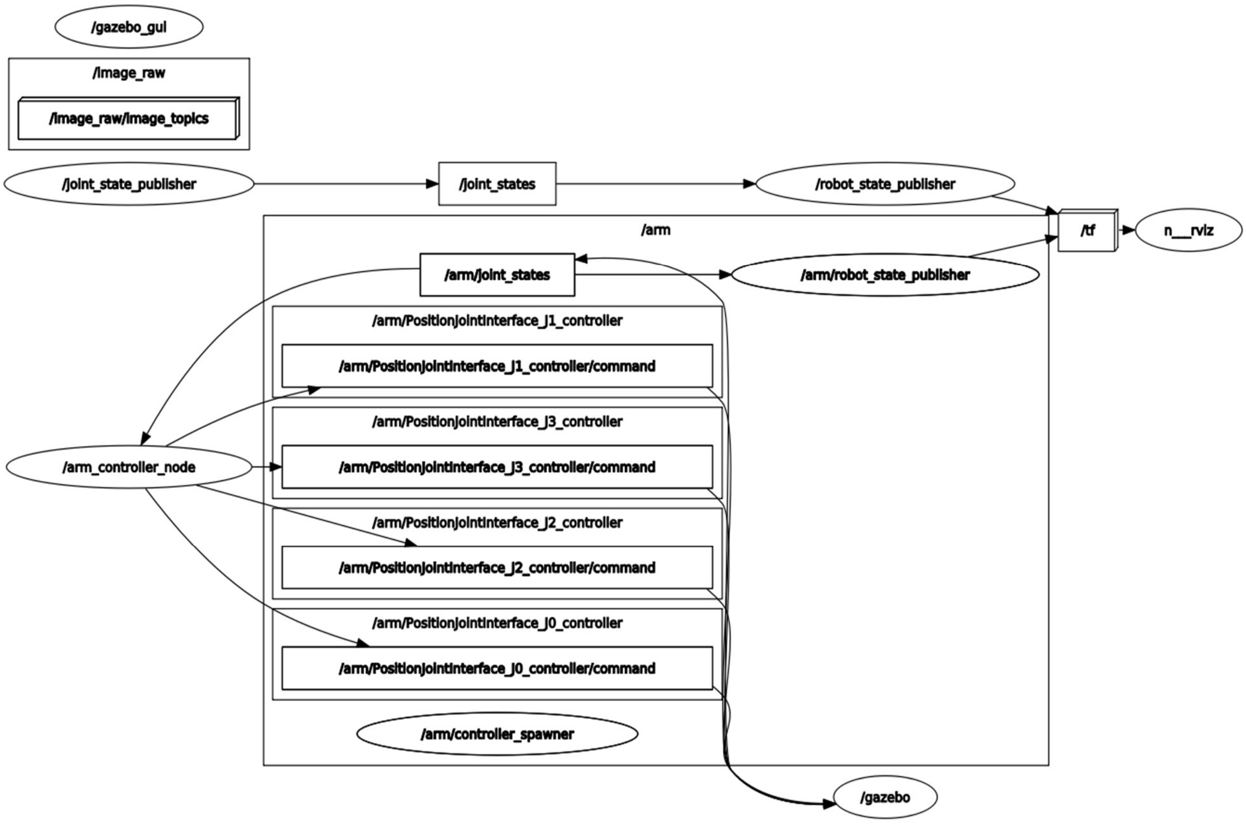
I assigned a different position for each joint and I obtained this configuration:




```
[ INFO] [1698871837.593387011, 1883.900000000]: Joint Name: j0, Position: 0.996000
[ INFO] [1698871837.593478116, 1883.900000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.593632467, 1883.900000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.593777202, 1883.900000000]: Joint Name: j3, Position: 0.400000
[ INFO] [1698871837.593903183, 1883.900000000]: Current Joint Positions:
[ INFO] [1698871837.594048718, 1883.900000000]: Joint Name: j0, Position: 0.997000
[ INFO] [1698871837.594177876, 1883.900000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.594303642, 1883.900000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.594423200, 1883.900000000]: Joint Name: j3, Position: 0.400000
[ INFO] [1698871837.594547738, 1883.900000000]: Current Joint Positions:
[ INFO] [1698871837.594661708, 1883.900000000]: Joint Name: j0, Position: 0.998000
[ INFO] [1698871837.594759797, 1883.900000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.594927171, 1883.900000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.595055653, 1883.900000000]: Joint Name: j3, Position: 0.400000
[ INFO] [1698871837.595193643, 1883.900000000]: Current Joint Positions:
[ INFO] [1698871837.595349342, 1883.900000000]: Joint Name: j0, Position: 0.999000
[ INFO] [1698871837.595490788, 1883.900000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.595608508, 1883.900000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.595723738, 1883.900000000]: Joint Name: j3, Position: 0.400000
[ INFO] [1698871837.690288343, 1884.000000000]: Current Joint Positions:
[ INFO] [1698871837.690938850, 1884.000000000]: Joint Name: j0, Position: 1.000000
[ INFO] [1698871837.694270419, 1884.000000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.694434054, 1884.000000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.694582016, 1884.000000000]: Joint Name: j3, Position: 0.400000
[ INFO] [1698871837.694761932, 1884.000000000]: Current Joint Positions:
[ INFO] [1698871837.694922302, 1884.000000000]: Joint Name: j0, Position: 1.000000
[ INFO] [1698871837.695058282, 1884.000000000]: Joint Name: j1, Position: 0.500000
[ INFO] [1698871837.695222439, 1884.000000000]: Joint Name: j2, Position: -0.700000
[ INFO] [1698871837.695948107, 1884.000000000]: Joint Name: j3, Position: 0.400000
```

5. Extra

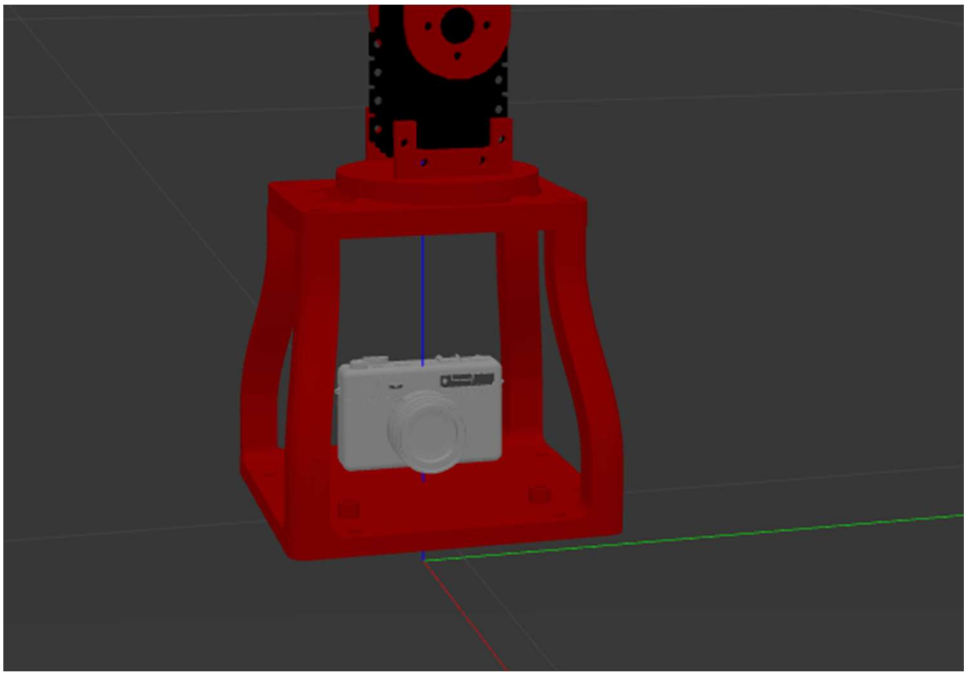
In order to complete this report I decided to show the graph obtained through the command "rqt_graph" when I ran arm_gazebo.lauch, display.launch and arm_controller_node.cpp together.



In conclusion I found a .stl file of a camera on internet and I added it to the meshes folder within the arm_description package then I changed the mesh of the camera_link in the arm.urdf.xacro file.

```
<!-- Add camera_link-->
  <link name="camera_link">
    <visual>
      <geometry>
        <!--box size="0.01 0.01 0.01"/-->
        <mesh filename="package://arm_description/meshes/YASHICA_ELECTRO_35.stl" scale="0.0003 0.0003 0.0003"/>
      </geometry>
      <origin rpy="0 0 1.571" xyz="0.015 0 -0.02"/>
      <material name="white"/>
    </visual>
```

The final result is:



Members of the group:

Davide Busco

Pietro Falco
Davide Rubinacci

Giuseppe Saggese