Homework 1

Mariateresa Ciuoffi, Giuseppe Cioffi, Alessia Iacono, Ciro Manfredonia

October 2024

Introduction

The goal of this homework is to build ROS packages to simulate a 4-degrees-of-freedom robotic manipulator arm into the Gazebo environment.

Here the links to the personal repositories.

Giuseppe Cioffi:
https://github.com/Peppecio/rl24_homework1

Mariateresa Ciuoffi:
https://github.com/meryciuo/RobLab_hw1

Ciro Manfredonia:
https://github.com/ciromanfry/RL24_homework1

Alessia Iacono
https://github.com/AlessiaIacono/Homework1_RoboticsLab/tree/main

Chapter 1

Create the description of your robot and visualize it in Rviz

a. Download the arm_description package into your ros2_ws using git commands

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~$ cd Docker
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/Docker$ ./docker_run_
container.sh rl24_image homework1_container  homework1
[WARNING] devel folder doesn't exists, creating a new one
non-network local connections being added to access control list
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
```

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/Docker/ros2_docker_sc
ripts$  ./docker_run_container.sh rl24_newimage homework1_container homework1
non-network local connections being added to access control list
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
```

We download the package from GitHub in the homework1 folder we created before.

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1$ git clone
https://github.com/RoboticsLab2024/arm_description.git
Cloning into 'arm_description'...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 33 (delta 5), reused 33 (delta 5), pack-reused 0 (from 0)
Receiving objects: 100% (33/33), 1.13 MiB | 570.00 KiB/s, done.
Resolving deltas: 100% (5/5), done.
```

In order to build the package, we launch the *colcon build* command.

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ colcon build
Starting >>> arm_description
Finished <<< arm_description [0.48s]

Summary: 1 package finished [0.60s]
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
```

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ source install/
local_setup.bash
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ls
build  install  log  src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ cd src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ ls
arm_description
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ cd arm_desc
ription
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_descript
ion$ ls
CMakeLists.txt  meshes  package.xml  urdf
```

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 rviz2 node. Launch the file using ros2 launch.

Before continuing, we needed to modify the CmakeLists.txt by adding some packages and declare what directories have to be installed with their destination.

```
cmake_minimum_required(VERSION 3.8)
project(arm_description)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_lifecycle REQUIRED)
find_package(rclpy REQUIRED)
#find_package(std_msgs REQUIRED)


install (
  DIRECTORY config launch meshes urdf
  DESTINATION share/${PROJECT_NAME}
)

ament_package()
```

After that, we add some other dependencies to package.xml.

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://
www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>arm_description</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="mario.selvaggio@unina.it">mrslvg</maintainer>
  <license>TODO</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <dependd>rclcpp</depend>
  <dependd>tf2_geometry_msgs</depend>

  <exec_depend>rclpy</exec_depend>
  <exec_depend>urdf</exec_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>


  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Now we need to create two new directories, one is launch and the second one is config, using *mkdir* command.

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1$ ls
arm_description
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1$ cd arm_des
cription
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_descrip
tion$ ls
CMakeLists.txt  meshes  package.xml  urdf
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_descrip
tion$ mkdir launch
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_descrip
tion$ ls
CMakeLists.txt  launch  meshes  package.xml  urdf
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_descrip
tion$ 
```

After this we use again *colcon build* and then *source/install setup.bash*

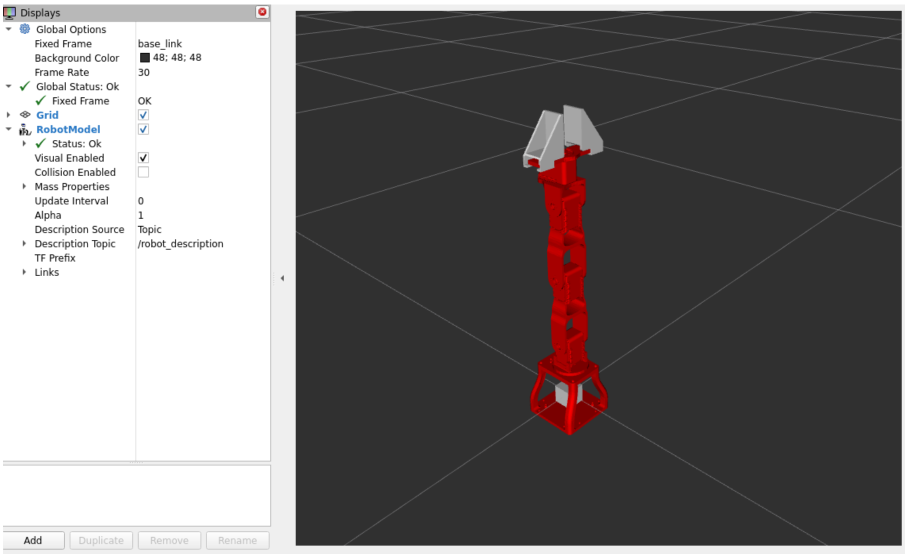
```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ colcon build
Starting >>> arm_description
Finished <<< arm_description [0.15s]

Summary: 1 package finished [0.27s]
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ source install/
setup.bash
```

We create a launch file named display_launch.py inside the folder launch.

```python
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import Command, LaunchConfiguration, PathJoinSubstitution
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
import os
from ament_index_python.packages import get_package_share_directory
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.actions import (
    DeclareLaunchArgument,
    IncludeLaunchDescription,
)

def generate_launch_description():

    declared_arguments = []

    declared_arguments.append(
        DeclareLaunchArgument(
            "rviz_config_file", #this will be the name of the argument
            default_value=PathJoinSubstitution(
                [FindPackageShare("arm_description") ,"config", "rviz", "arm_description.rviz"]
            ),
            description="RViz config file (absolute path) to use when launching rviz.",
        )
    )

    homework1_path = get_package_share_directory('arm.urdf')

    urdf_file = os.path.join(links_urdf_path, "urdf", "arm.urdf")

    with open(urdf_file, 'r') as infp:
        arm_desc = infp.read()

    robot_description_arm = {"robot_description": arm_desc}

    joint_state_publisher_node = Node(
        package="joint_state_publisher",
        executable="joint_state_publisher",
    )

    robot_state_publisher_node = Node(
        package="robot_state_publisher", #ros2 run robot_state_publisher robot_state_publisher
        executable="robot_state_publisher",
        output="both",
        parameters=[robot_description_arm,
                    {"use_sim_time": True},
            ],
        remappings=[('/robot_description', '/robot_description')]
    )
```

```
    rviz_node = Node(
        package="rviz2",
        executable="rviz2",
        name="rviz2",
        output="log",
        arguments=["-d", LaunchConfiguration("rviz_config_file")],
    )

    nodes_to_start = [
        joint_state_publisher_node,
        robot_state_publisher_node,
        rviz_node
    ]


    return LaunchDescription(declared_arguments + nodes_to_start)
```

Then we launch the file with *ros2 launch*, so we can open our robot in Rviz. To visualize it, we add the robotModel and the topic /robot_description. Finally we set the fixed frame as base_link.



Optional : After opening rviz file, we saved it as "arm_description.rviz" inside a folder rviz which is inside the folder config.

c. Substitute the collision meshes of your URDF with primitive shapes. Use <box>geometries of reasonable size approximating the links.

Firstly we enable collision, and we open the file arm.urdf where we need to modify the mesh part using box related to collision.

```
      <geometry>
        <box size = "0.1 0.1 0.08" />
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </collision>
    <inertial>
```

The box size has been chosen considering the real parameters value of the parts and scale it.

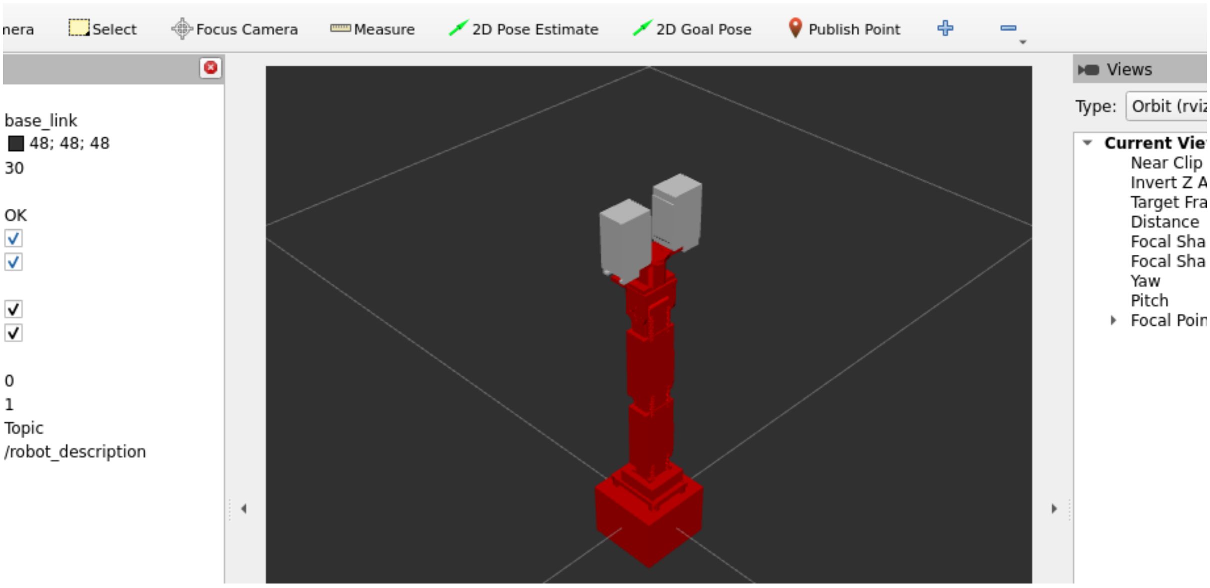


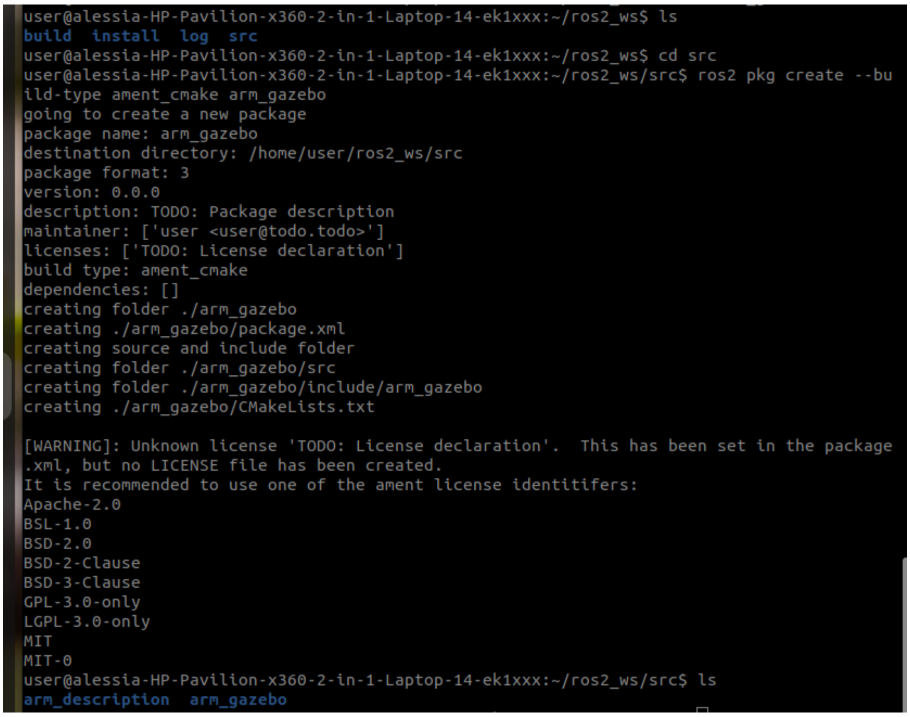
Figure 1.1: Collision boxes

Chapter 2

Add sensors and controllers to your robot and spawn it in Gazebo

a. Create a package named *arm_gazebo*

To create a new package called arm_gazebo, we use the command ros2 pkg create inside the src folder.
Now we return to the workspace and we launch *colcon build* and source *install/setup.bash*



```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ls
build  install  log  src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ cd src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ ros2 pkg create --bu
ild-type ament_cmake arm_gazebo
going to create a new package
package name: arm_gazebo
destination directory: /home/user/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['user <user@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./arm_gazebo
creating ./arm_gazebo/package.xml
creating source and include folder
creating folder ./arm_gazebo/src
creating folder ./arm_gazebo/include/arm_gazebo
creating ./arm_gazebo/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the package
.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ ls
arm_description  arm_gazebo
```

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

Now we open a terminal inside arm_gazebo, we create a folder with the name launch and inside launch with touch we create the launch file.

```
touch arm_world.launch.py
```

c. Fill this launch file with commands that load the URDF into the /robot_description topic and spawn your robot using the create node in the ros_gz_sim package

Firstly we take the launch file that we will modify, then we modify the cmake adding install

```
cmake_minimum_required(VERSION 3.8)
project(arm_gazebo)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)


install (
  DIRECTORY launch
  DESTINATION share/${PROJECT_NAME}
)


ament_package()
```

And we modify package.

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>arm_description</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="mario.selvaggio@unina.it">mrslvg</maintainer>
  <license>TODO</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>tf2_geometry_msgs</depend>

  <exec_depend>rclpy</exec_depend>
  <exec_depend>urdf</exec_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>


  <export>
    <build_type>ament_cmake</build_type>
    <gazebo_ros gazebo_model_path = "${prefix}/.."/>
  </export>
</package>
```

After we can use colcon and source and then ros2 to launch the arm file.
We can open Gazebo.

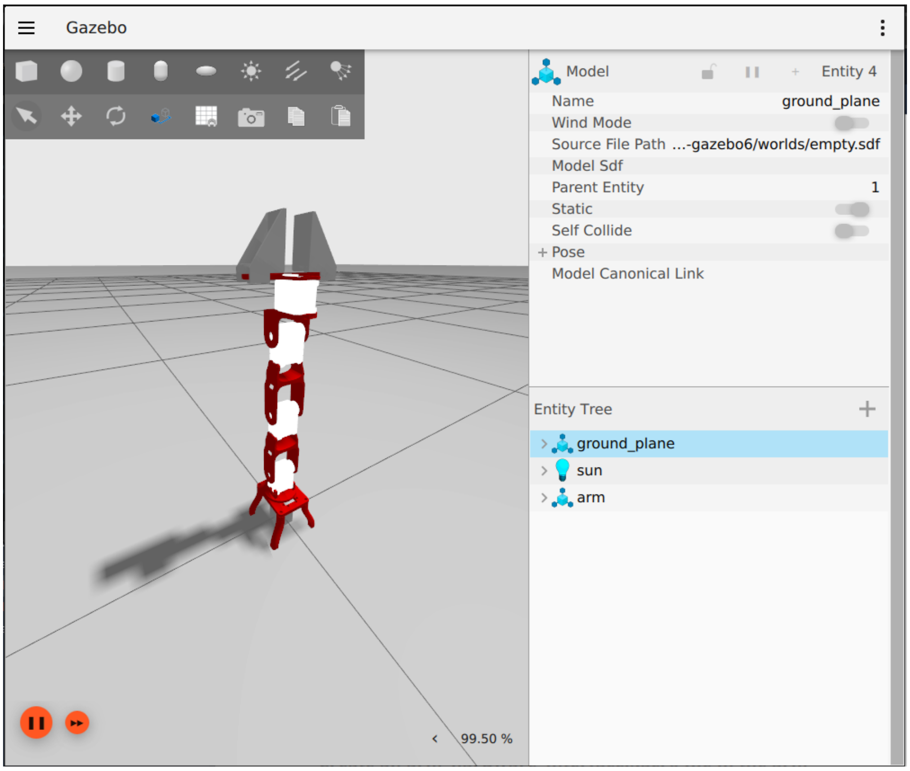


Figure 2.1: Robot manipulator in Gazebo

d. Add a PositionJointInterface as a hardware interface to your robot using ros2_control. Create an arm_hardware_interface.xacro file in the arm_description/urdf folder, containing a macro that defines the hardware interface for the joint, and include it in your main arm.urdf.xacro file using xacro:include. Specifically, define the joint using ros2_control and specify the hardware interface as PositionJointInterface.

We create arm_hardware_interface.xacro.

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~...
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_description/urdf$ touch arm_hardware_interface.xacro
```

After this we renamed our arm.urdf in arm.urdf.xacro and then we proceed to modify our launch file in order to consider the new file.

```
#    urdf_file= os.path.join(arm_description_path, "urdf", "arm.urdf")
    urdf_xacro = os.path.join(arm_description_path, "urdf", "arm.urdf.xacro")

#    with open(urdf_file, 'r') as infp:
#        arm_desc = infp.read()

#    robot_description_arm = {"robot_description": arm_desc}

    robot_description_arm = {"robot_description":Command(['xacro ', urdf_xacro])}
```

This is the arm_hardware_interface.xacro.

```xml
<?xml version="1.0" encoding="utf-8"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">


  <xacro:macro name="PositionJointInterface" params="name initial_pos">

      <joint name="${name}">
        <command_interface name="position"/>
        <state_interface name="position">
            <param name="initial_value">${initial_pos}</param>
        </state_interface>
      </joint>

  </xacro:macro>


</robot>
```

Our arm.urdf.xacro needs to include the xacro file so we add this.

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

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


  <material name="red">
    <color rgba="1 0 0 1"/>
```

and this too:

```xml
  <xacro:arg name = "j0_pos" default="0.0"/>
  <xacro:arg name = "j1_pos" default="0.0"/>
  <xacro:arg name = "j2_pos" default="0.0"/>
  <xacro:arg name = "j3_pos" default="0.0"/>

  <ros2_control name="HardwareInterface_Ignition" type = "system">
      <hardware>
          <plugin>ign_ros2_control/IgnitionSystem</plugin>
      </hardware>

      <xacro:PositionJointInterface name="j0" initial_pos="$(arg j0_pos)"/>
      <xacro:PositionJointInterface name="j1" initial_pos="$(arg j1_pos)"/>
      <xacro:PositionJointInterface name="j2" initial_pos="$(arg j2_pos)"/>
      <xacro:PositionJointInterface name="j3" initial_pos="$(arg j3_pos)"/>

  </ros2_control>
```

e. Add inside the arm.urdf.xacro the commands to load the joint controller configurations from the .yaml file and spawn the controllers using the controller_manager package. Then, launch the robot simulation in Gazebo and demonstrate how the hardware interface is correctly loaded and connected.

We add this part to arm.urdf.xacro in order to load the joint controller configurations.

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~/ros2_ws
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~...
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_description/config$ touch arm_controllers.yaml
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_description/config$
```

```
<gazebo>
  <plugin filename="ign_ros2_control-system" name="ign_ros2_control::IgnitionROS2ControlPlugin">
  <parameters>$(find arm_description)/config/arm_controllers.yaml</parameters>
    <controller_manager_prefix_node_name>controller_manager</controller_manager_prefix_node_name>
  </plugin>
</gazebo>
```

In order to demonstrate how the hardware interface is correctly loaded and connected we use:

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~/ros2_ws 80x14
 launched system
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ros2 control list_hardware_interfaces
command interfaces
        j0/position [available] [claimed]
        j1/position [available] [claimed]
        j2/position [available] [claimed]
        j3/position [available] [claimed]
state interfaces
        j0/position
        j1/position
        j2/position
        j3/position
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
</gazebo>
```

f. 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.

Firstly we need to go inside the src folder and then we create a package:

```
/spawner_joint_trajectory_controller
/spawner_position_controller
/spawner_position_controller
/transform_listener_impl_57130f03b120
/transform_listener_impl_593590970f20
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ cd ~/ros2_ws/src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ ros2 pkg create
arm_control --build-type ament_cmake --dependencies controller_manager joint_state_br
oadcaster position_controllers
going to create a new package
package name: arm_control
destination directory: /home/user/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['user <user@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['controller_manager', 'joint_state_broadcaster', 'position_controllers
']
creating folder ./arm_control
creating ./arm_control/package.xml
creating source and include folder
creating folder ./arm_control/src
creating folder ./arm_control/include/arm_control
creating ./arm_control/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the pac
kage.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$
```

After this we go inside arm_control and we create launch and config.

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src$ cd arm_control
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control$ mkdi
r launch config
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control$ ls
CMakeLists.txt  config  include  launch  package.xml  src
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control$
```

Now we create inside the launch folder, arm_control.launch and inside config, arm_control.yaml.

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control$ cd l
aunch
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control/launc
h$ touch arm_control.launch.py
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control/launc
h$ cd ..
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control$ cd c
onfig
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control/confi
g$ touch arm_control.yaml
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control/confi
g$ ls
arm_control.yaml
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws/src/arm_control/confi
g$
```

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

arm_gazebo.launch.py × arm.urdf.xacro × arm_world.launch.py × arm_control.yaml

```yaml
controller_manager:
  ros__parameters:
    update_rate: 225  # Hz

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

    position_controller:
      type: position_controllers/JointGroupPositionController

position_controller:
  ros__parameters:
    joints:
      - j0
      - j1
      - j2
      - j3
```

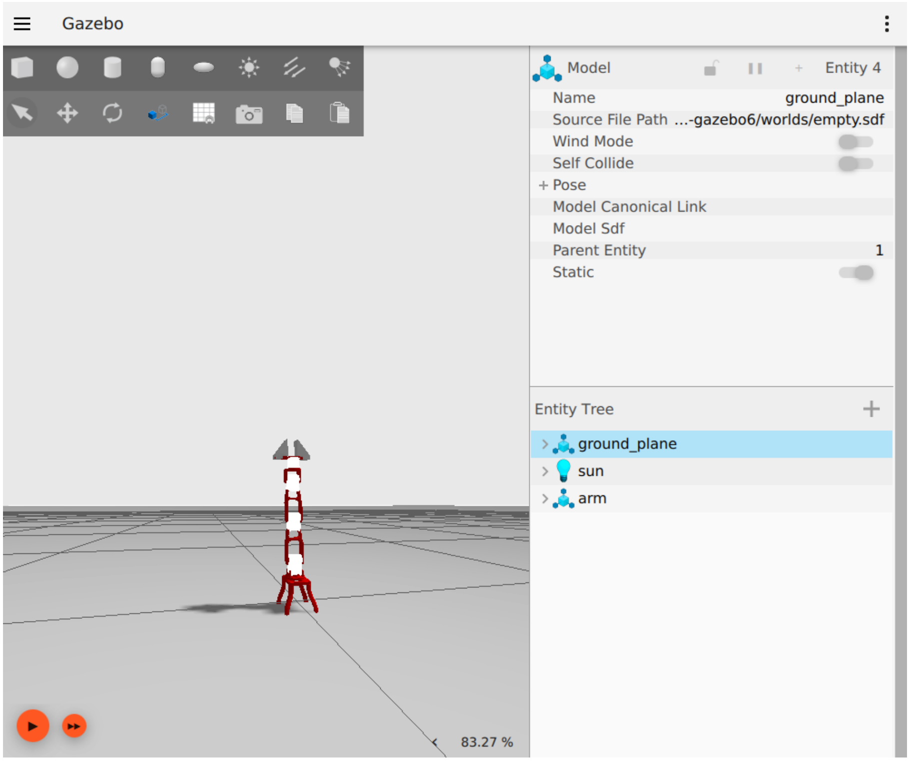
h. Create an arm_gazebo.launch file into the launch folder of the arm_gazebo package loading the Gazebo world with arm_world.launch and spawning the controllers within arm_control.launch. Launch the simulation and check if your controllers are correctly loaded

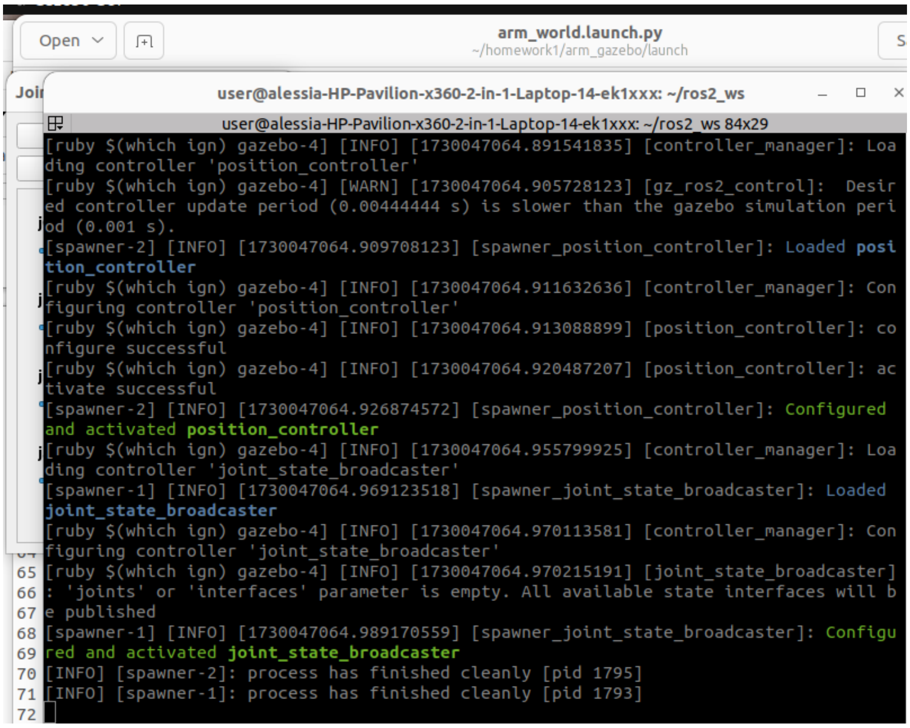
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~...

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_gazebo/launch$ touch arm_gazebo.launch.py
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_gazebo/
```

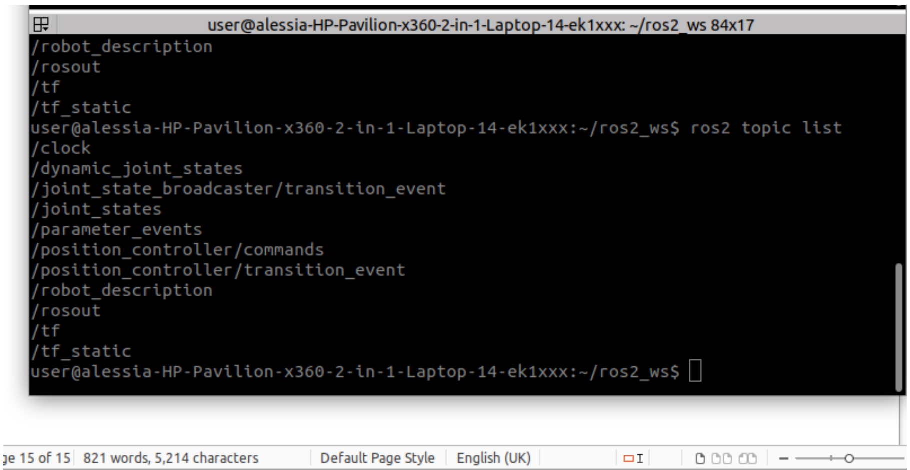
*arm_gazebo.launch.py × arm.urdf.xacro × arm_world.launch.py ×

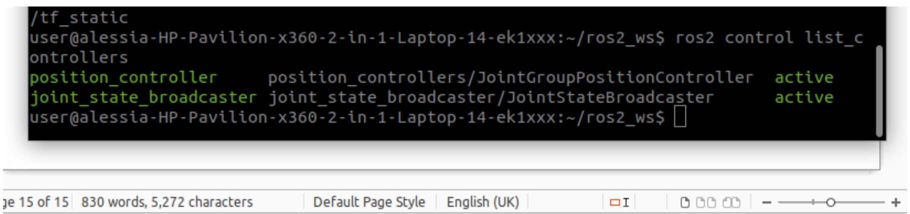
```python
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import Command, LaunchConfiguration, PathJoinSubstitution
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
import os
from ament_index_python.packages import get_package_share_directory
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.actions import (
    DeclareLaunchArgument,
    IncludeLaunchDescription,
)
from launch.actions import RegisterEventHandler
from launch.event_handlers import OnProcessExit



def generate_launch_description():

    declared_arguments = []


    gazebo_world = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [PathJoinSubstitution([FindPackageShare('arm_gazebo'),
                                       'launch',
                                       'arm_world.launch.py'])]),
    )

    controllers = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [PathJoinSubstitution([FindPackageShare('arm_control'),
                                       'launch',
                                       'arm_control.launch.py'])]),
    )


    nodes_to_start = [
        controllers,
        gazebo_world
    ]

    return LaunchDescription(nodes_to_start)
```





To verify that all the controllers are perfectly loaded:


user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx: ~/ros2_ws 84x17
/robot_description
/rosout
/tf
/tf_static
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ros2 topic list
/clock
/dynamic_joint_states
/joint_state_broadcaster/transition_event
/joint_states
/parameter_events
/position_controller/commands
/position_controller/transition_event
/robot_description
/rosout
/tf
/tf_static
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
15 of 15
821 words, 5,214 characters
Default Page Style
English (UK)


/tf_static
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ros2 control list_c
ontrollers
position_controller position_controllers/JointGroupPositionController active
joint_state_broadcaster joint_state_broadcaster/JointStateBroadcaster active
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$
15 of 15
830 words, 5,272 characters
Default Page Style
English (UK)

Chapter 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

We size and position the camera appropriately.

```
<joint name="camera_joint" type="fixed">
    <parent link="base_link"/>
    <child link="camera_link"/>
    <origin xyz="0 0 0.0008" />
</joint>

<link name="camera_link">
    <visual>
        <geometry>
            <box size="0.03 0.03 0.03"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="white"/>
    </visual>
</link>
```

b. Create an arm_camera.xacro file in the arm_gazebo/urdf folder, add the gazebo sensor reference tags and the gz-sim-sensors-system plugin to your xacro.

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_gazebo$ mkdir urdf
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_gazebo$ cd urdf
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_gazebo/urdf$ touch arm_camera.xacro
```

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

  <xacro:include filename="$(find arm_description)/urdf/arm_hardware_interface.xacro" />
  <xacro:include filename="$(find arm_gazebo)/urdf/arm_camera.xacro" />

```

In arm.urdf.xacro, we add these lines:

```
  </ros2_control>
 <gazebo>
          <plugin filename="gz-sim-sensors-system"
                name="gz::sim::systems::Sensors">
                <render_engine>ogre2</render_engine>
        </plugin>
  </gazebo>
  <xacro:arm_camera link = "camera_link" />

```

This is armc_camera.xacro:

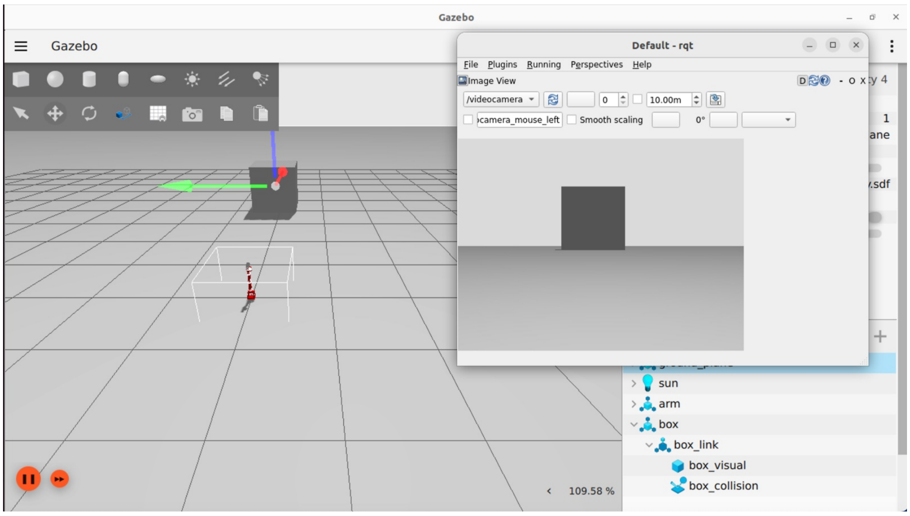
```
<?xml version="1.0" encoding="utf-8"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">


  <xacro:macro name="arm_camera" params="link">



  <gazebo reference="${link}">
  <sensor name="camera" type="camera">
    <camera>
      <horizontal_fov>1.047</horizontal_fov>
      <image>
        <width>320</width>
        <height>240</height>
      </image>
      <clip>
        <near>0.1</near>
        <far>100</far>
      </clip>
    </camera>
    <always_on>1</always_on>
    <update_rate>30</update_rate>
    <visualize>true</visualize>
    <topic>camera</topic>
  </sensor>
</gazebo>

  </xacro:macro>


</robot>

```

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

We need to add another node that we need to start related to camera.
Using rqt we can see the image:

We added a box, and, as it is possible to see, we can see with the camera the box.

```
    bridge_camera = Node(
        package='ros_ign_bridge',
        executable='parameter_bridge',
        arguments=[
            '/camera@sensor_msgs/msg/Image@gz.msgs.Image',
            '/camera_info@sensor_msgs/msg/CameraInfo@gz.msgs.CameraInfo',
            '--ros-args',
            '-r', '/camera:=/videocamera',
        ],
        output='screen'
    )


    nodes_to_start = [
        robot_state_publisher_node,
        *ign,
        joint_state_publisher_node,
    #   joint_state_broadcaster,
        bridge_camera,

    ]
```



d. Optionally: You can create a camera.xacro file and add it to your robot URDF using <xacro:include>

We create a camera.xacro file containing the definition of the camera_link and the camera_joint. And the we include it to arm_urdf.xacro using <xacro:include>.

arm.urdf.xacro camera.xacro

home > peppecio > homework1 > arm_description > urdf > camera.xacro

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

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

 <joint name="camera_joint" type="fixed">
      <parent link="base_link"/>
      <child link="camera_link"/>
      <origin xyz="0 0 0.0008"/>
    </joint>

    <link name="camera_link">
      <visual>
        <geometry>
          <box size="0.05 0.05 0.05"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="white"/>
      </visual>
    </link>

</robot>
```

Chapter 4

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

a. Inside the arm_controller package create a ROS C++ node named arm_controller_node. The dependencies are rclcpp, sensor_msgs and std_msgs. Modify opportunely the CMakeLists.txt file to compile your node

arm_control.launch.py × CMakeLists.txt × package.xml ×

```
cmake_minimum_required(VERSION 3.8)
project(arm_control)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(controller_manager REQUIRED)
find_package(joint_state_broadcaster REQUIRED)
find_package(position_controllers REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(sensor_msgs REQUIRED)



if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()


add_executable(talker_listener src/pub_sub.cpp)


ament_target_dependencies(talker_listener rclcpp std_msgs sensor_msgs)
ù


install (
  TARGETS talker listener
  DIRECTORY config launch
  DESTINATION share/${PROJECT_NAME}
)
ament_package()
```

After modifying the package and the cmake, we need to create a new file.cpp inside src of arm control.

```
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_control/src$ touch pub_sub.cpp
alessia@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/homework1/arm_control/src$
```

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

c. Create publishers that write commands onto the /position_

We create a single class containing both the publisher and the subscriber.

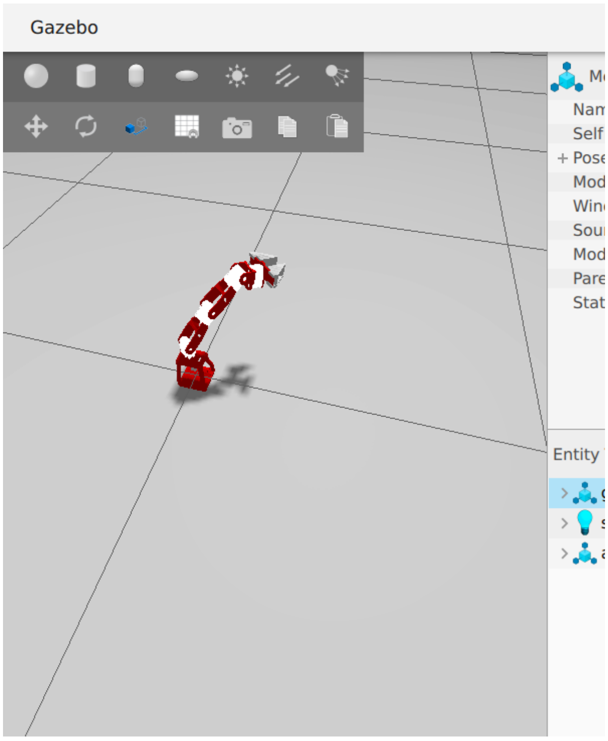
```
ome > peppecio > hw1fold > arm_control > src > pub_sub.cpp
#include <functional>
#include <memory>
#include <vector>
#include <chrono>

#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "std_msgs/msg/float64_multi_array.hpp"

using namespace std::chrono_literals;

using std::placeholders::_1;

class ArmController : public rclcpp::Node
{
public:
  ArmController() : Node("arm_controller_node")
  {
    subscriber_ = this->create_subscription<sensor_msgs::msg::JointState>(
      "joint_states", 10, std::bind(&ArmController::jointStateCallback, this, _1));

    publisher_ = this->create_publisher<std_msgs::msg::Float64MultiArray>(
      "/position_controller/commands", 10);

    timer_ = this->create_wall_timer(
      std::chrono::milliseconds(500), std::bind(&ArmController::publishCommand, thi
  }

private:
  void jointStateCallback(const sensor_msgs::msg::JointState::SharedPtr msg)
  {
    current_joint_positions_ = {msg->position[0], msg->position[1], msg->position[2
    RCLCPP_INFO(this->get_logger(), "joint_positions: [%f, %f, %f, %f]",
                current_joint_positions_[0], current_joint_positions_[1],
                current_joint_positions_[2], current_joint_positions_[3]);
  }
```

Using the command below it is possible to modify the arm configuration. Firstly we modify the values inside [] and we can obtain the new configuration.

```cpp
private:
  void jointStateCallback(const sensor_msgs::msg::JointState::SharedPtr msg)
  {
    current_joint_positions_ = {msg->position[0], msg->position[1], msg->position[2
    RCLCPP_INFO(this->get_logger(), "joint_positions: [%f, %f, %f, %f]",
                current_joint_positions_[0], current_joint_positions_[1],
                current_joint_positions_[2], current_joint_positions_[3]);
  }

  void publishCommand()
  {
    auto message = std_msgs::msg::Float64MultiArray();
    message.data = {1.0, 1.0, 1.0, 1.0};
    RCLCPP_INFO(this->get_logger(), "Publishing: [%f, %f, %f, %f]",
               message.data[0], message.data[1], message.data[2], message.data[3])
    publisher_->publish(message);
  }

  rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr subscriber_;
  rclcpp::Publisher<std_msgs::msg::Float64MultiArray>::SharedPtr publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
  std::vector<double> current_joint_positions_;
};

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<ArmController>());
  rclcpp::shutdown();
  return 0;
}
```

```
user@alessia-HP-Pavilion-x360-2-in-1-Laptop-14-ek1xxx:~/ros2_ws$ ros2 topic pub
/position_controller/commands std_msgs/msg/Float64MultiArray "data: [0.01, 0.0,
0.0, 0.0]"
```

An example:



Using

```
publishing #58: std_msgs.msg.Float64MultiArray(layout=std_msgs.msg.MultiArrayLayout(dim=[], data_offset=0), data=[1.01, 0.5, 0.2, 0.7])
```