

Robotics Lab: Homework 1

Students: Anzalone Claudio, Maisto Paolo, Manzoni Antonio

Here is the link to my public repo on github:

https://github.com/PaoloMaisto/HW1_RL_Maisto_Paolo.git

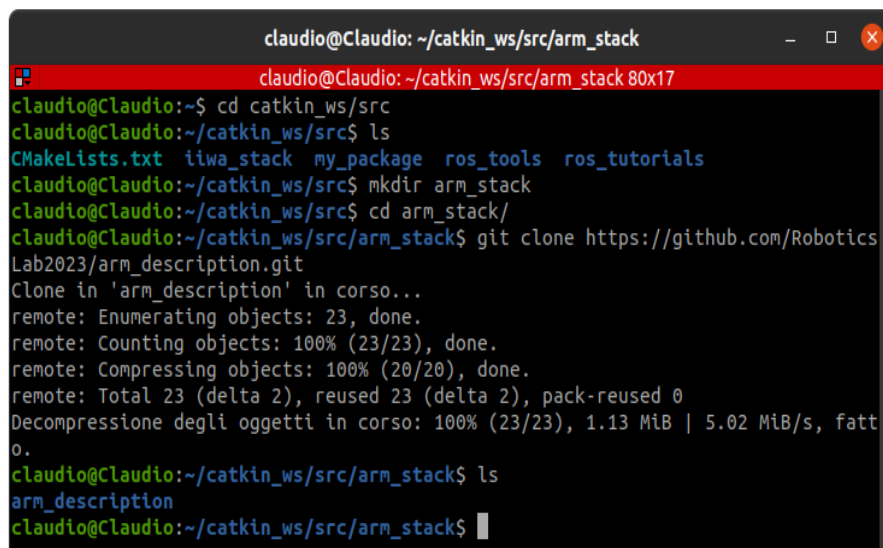
We want to specify that all the participants of the group worked at each stage of the development of the project. In order to simplify the drafting of the report (as recommended by the professor) we have fairly divided the writing of the development of the various points.

Building your robot manipulator

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

In first place it has been downloaded the “arm_description” folder available on GitHub with the command “git clone https://github.com/RoboticsLab2023/arm_description.git” inside the directory “catkin_ws/src/arm_stack”, which contains all the folders that are needed for operating the robot .



```
claudio@Claudio: ~/catkin_ws/src/arm_stack
claudio@Claudio: ~/catkin_ws/src/arm_stack 80x17
claudio@Claudio:~$ cd catkin_ws/src
claudio@Claudio:~/catkin_ws/src$ ls
CMakeLists.txt  iiwa_stack  my_package  ros_tools  ros_tutorials
claudio@Claudio:~/catkin_ws/src$ mkdir arm_stack
claudio@Claudio:~/catkin_ws/src$ cd arm_stack/
claudio@Claudio:~/catkin_ws/src/arm_stack$ git clone https://github.com/Robotics
Lab2023/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 | 5.02 MiB/s, fatt
o.
claudio@Claudio:~/catkin_ws/src/arm_stack$ ls
arm_description
claudio@Claudio:~/catkin_ws/src/arm_stack$
```

- (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, that automatically loads the RobotModel plugin by default, and give it as an argument to your node in the display.launch file

It has been created a folder named “launch” that contains the launch files necessary to define and configure the start of the system’s parameters and nodes. This point has been done firstly through the command “mkdir launch” in order to create the folder and then with the command “touch

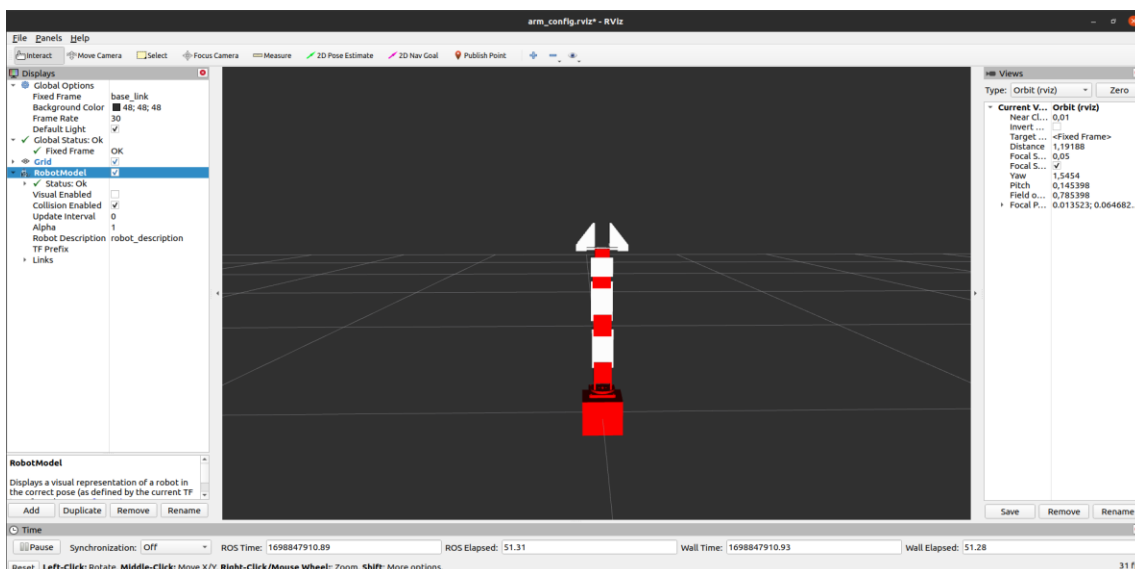
display.launch” to create the file that contains the URDF model with all the parameters of the robot and the boot parameters of the nodes: **"robot_state_publisher"**, **"joint_state_publisher"** and **"rviz"**. The configuration file **"arm_config.rviz"** that contains the **"RobotModel plugin"** has been saved inside the directory **"/arm_stack/arm_description/config"**.

```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/launch
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/launch 80x17
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description$ mkdir launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description$ ls
CMakeLists.txt  launch  meshes  package.xml  urdf
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description$ cd launch/
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/launch$ touch display.launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/launch$ ls
display.launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/launch$
```

To start the visualization of the robot in rviz environment it has been used the command **"roslaunch arm_description display.launch"**.

- (c) Substitute the collision meshes of your URDF with primitive shapes. Use **<box>** geometries of reasonable 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

It has been substituted the collision meshes of the URDF with primitive shapes of reasonable size. To show the results of this point it is displayed the following image:



- (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

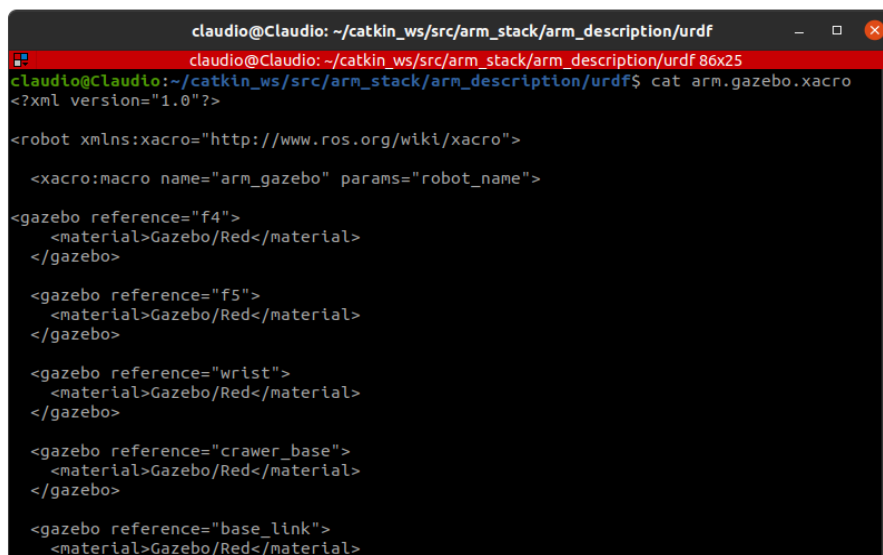
It has been created the file “**arm_gazebo.xacro**” inside the package “**arm_description**”.



```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf 90x7
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/urdf$ touch arm.gazebo.xacro
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/urdf$ ls
arm.gazebo.xacro  arm.urdf
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/urdf$
```

Inside this file it has been added the `xacro:macro` `<xacro:macro name="arm_gazebo" params="robot_name">` that contains all the Gazebo tags present inside the file “**arm.urdf**”.

When the `arm.urdf` file is compiled, what is inside the “`arm.gazebo.xacro`” file will be included inside the Xacro file that contains this instruction: “`<xacro:include filename="$(find arm_description)/urdf/arm.gazebo.xacro"/>`”. This is useful in order to split the URDF model in smaller and manageable parts, so we can have a better organization of the code and also allowing the reutilization of the components.



```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf 86x25
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/urdf$ cat arm.gazebo.xacro
<?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>

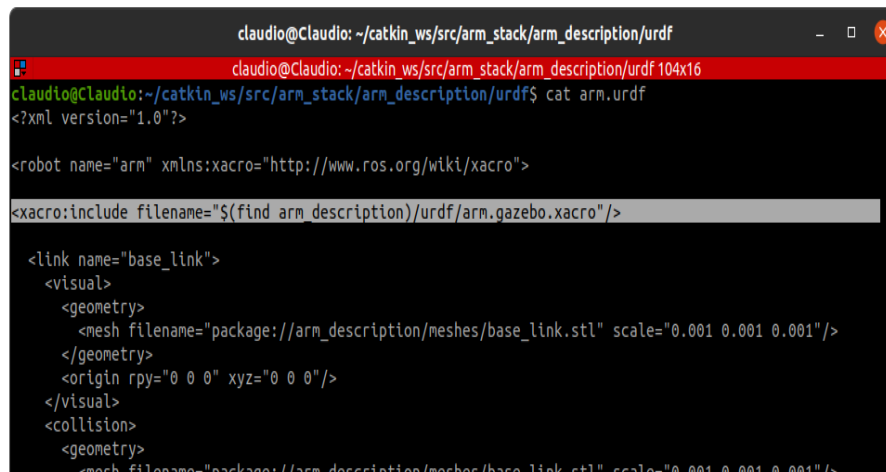
<gazebo reference="f5">
  <material>Gazebo/Red</material>
</gazebo>

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

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

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

The file “**arm.gazebo.xacro**” has been included inside “**arm.urdf**” through the xacro:include marked on the image.

A terminal window titled 'claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf' with a red header bar. The terminal shows the command 'cat arm.urdf' and its output, which is an XML file for a robot arm. The output includes an xacro:include tag that references 'arm.gazebo.xacro'.

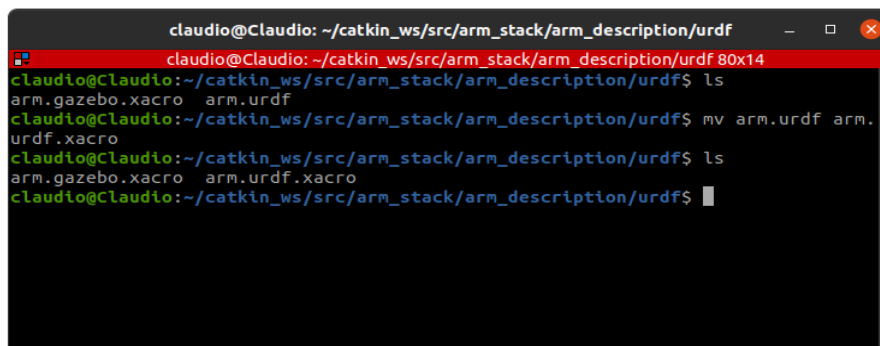
```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf$ cat arm.urdf
<?xml version="1.0"?>

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

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

  <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>
        <mesh filename="package://arm_description/meshes/base_link.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
  </link>
</robot>
```

At last the file “**arm.urdf**” has been renamed in “**arm.urdf.xacro**”

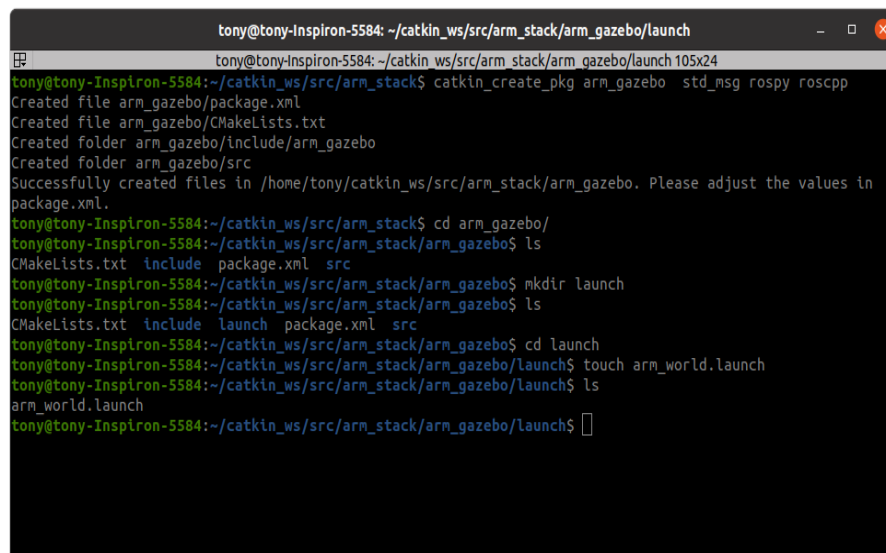
A terminal window titled 'claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf' with a red header bar. The terminal shows the command 'ls' followed by 'mv arm.urdf arm.urdf.xacro', and then 'ls' again to confirm the change. The output shows 'arm.gazebo.xacro' and 'arm.urdf.xacro' in the directory.

```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf$ ls
arm.gazebo.xacro  arm.urdf
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf$ mv arm.urdf arm.urdf.xacro
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf$ ls
arm.gazebo.xacro  arm.urdf.xacro
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf$
```

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

- (a) Create a package named `arm_gazebo`
- (b) Within this package create a launch folder containing a `arm_world.launch` file

In those passages it has been created in first the package `arm_gazebo` with the command “`catkin_create_pkg arm_gazebo std_msgs roscpp`” and then it was filled with the launch folder. In second place it was created the file “`arm_world.launch`”.



```
tony@tony-Inspiron-5584: ~/catkin_ws/src/arm_stack/arm_gazebo/launch
tony@tony-Inspiron-5584: ~/catkin_ws/src/arm_stack/arm_gazebo/launch 105x24
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack$ catkin_create_pkg arm_gazebo std_msgs roscpp
Created file arm_gazebo/package.xml
Created file arm_gazebo/CMakeLists.txt
Created folder arm_gazebo/include/arm_gazebo
Created folder arm_gazebo/src
Successfully created files in /home/tony/catkin_ws/src/arm_stack/arm_gazebo. Please adjust the values in
package.xml.
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack$ cd arm_gazebo/
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ ls
CMakeLists.txt include package.xml src
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ mkdir launch
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ ls
CMakeLists.txt include launch package.xml src
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ cd launch
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo/launch$ touch arm_world.launch
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo/launch$ ls
arm_world.launch
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/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

To accomplish this point it was used as example the `iiwa`. To show the results in Gazebo it is required to launch the “`arm_world.launch`” file.

```
/home/tony/catkin_ws/src/arm_stack/arm_gazebo/launch/arm_world.launch http://localhost:11311
/home/tony/catkin_ws/src/arm_stack/arm_gazebo/launch/arm_world.launch http://localhost:11311 105x24
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ roslaunch arm_gazebo arm_world.launch
... logging to /home/tony/.ros/log/27852fd8-78c9-11ee-8c93-a57d0f2aa487/roslaunch-tony-Inspiron-5584-9668
.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

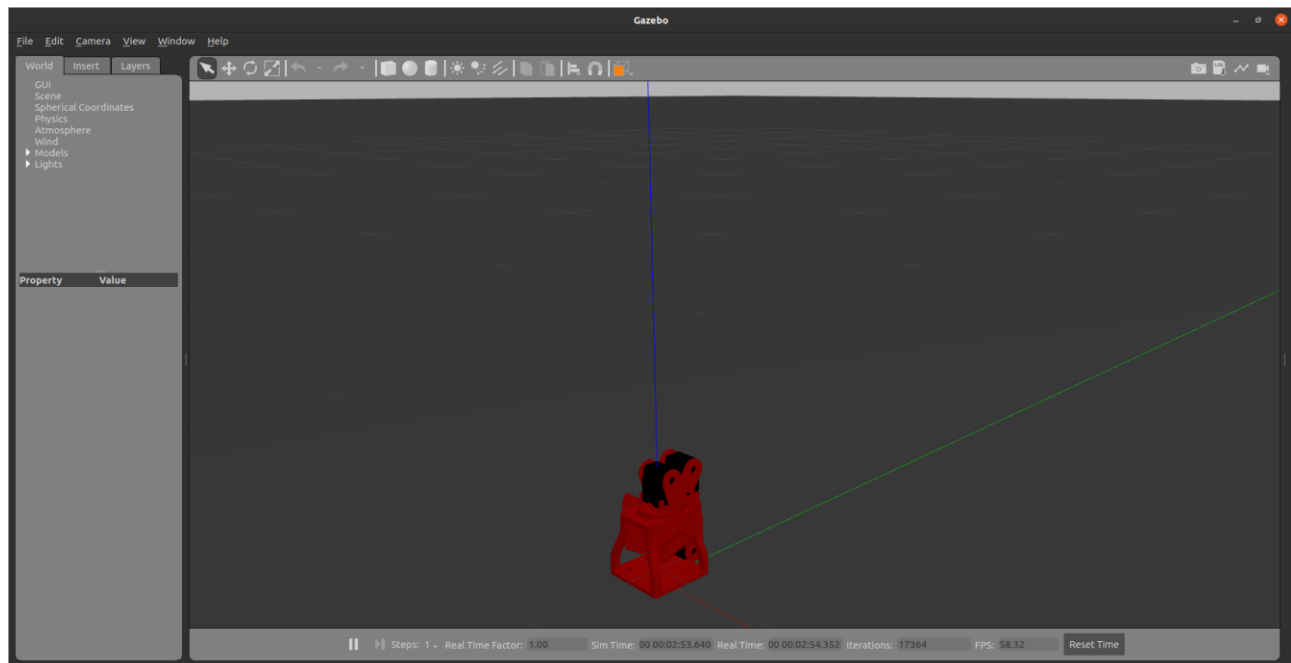
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://tony-Inspiron-5584:35233/

SUMMARY
=====

PARAMETERS
* /gazebo/enable_ros_network: True
* /robot_description: <?xml version="1...
* /roscdistro: noetic
* /rosversion: 1.16.0
* /use_sim_time: True

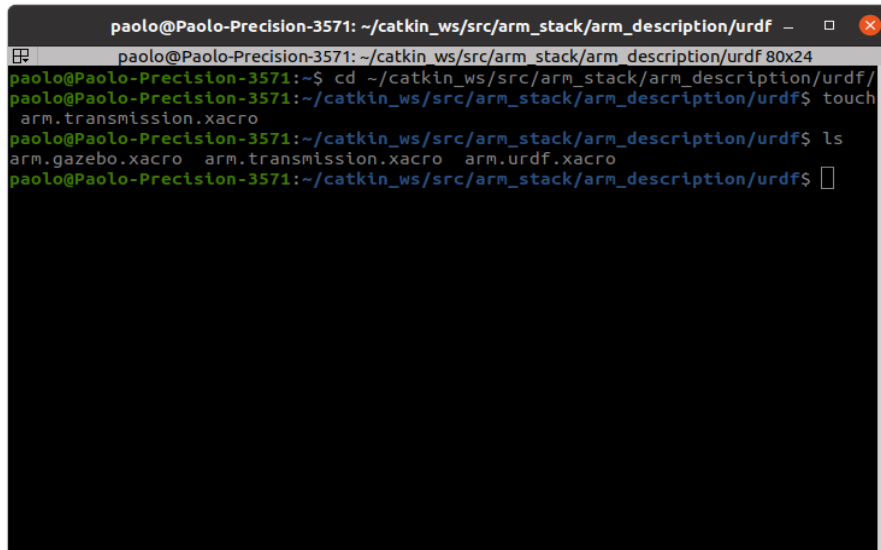
NODES
/
  gazebo (gazebo_ros/gzserver)
  gazebo_gui (gazebo_ros/gzclient)
```

As we can see the robot collapses on itself because it must be added the transmissions and the controllers.



- (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

The file “**arm.transmission.xacro**” has been created with the command “**touch**” in the following path “**~/catkin_ws/src/arm_stack/arm_description/urdf**”.

A terminal window titled "paolo@Paolo-Precision-3571: ~/catkin_ws/src/arm_stack/arm_description/urdf" shows the following commands and output:

```
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_description/urdf$ touch arm.transmission.xacro
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_description/urdf$ ls
arm.gazebo.xacro  arm.transmission.xacro  arm.urdf.xacro
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_description/urdf$
```

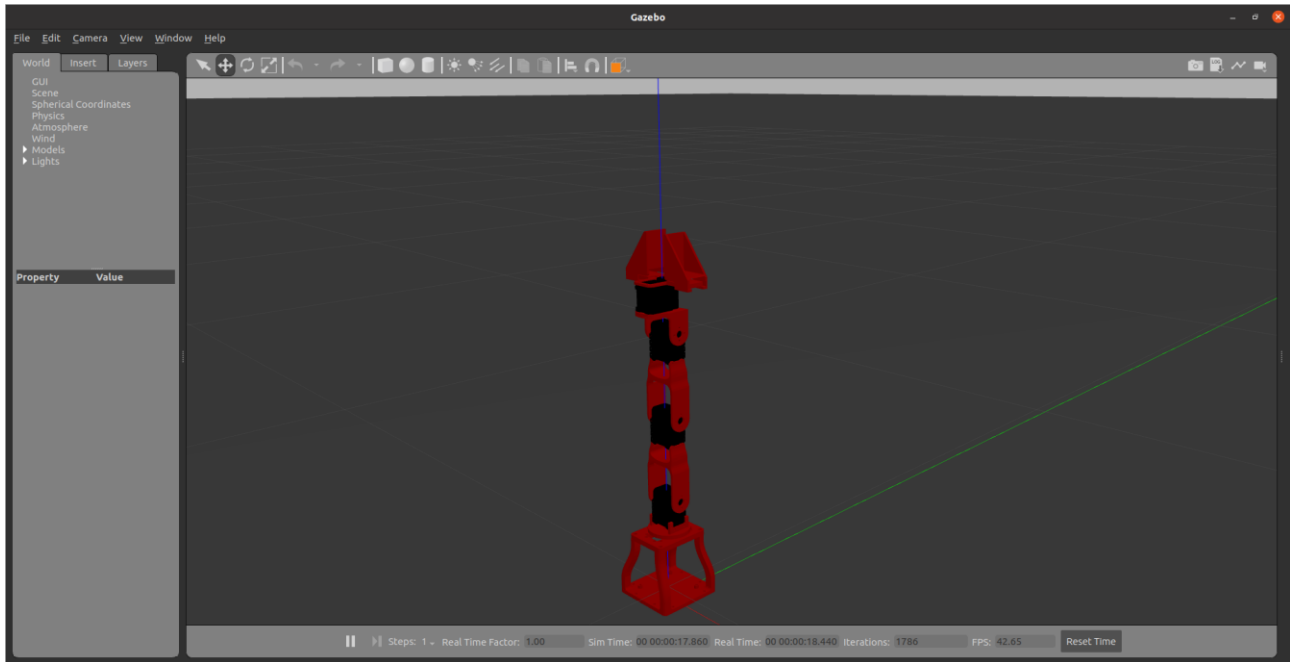
Then the “**arm.transmission.xacro**” file has been correctly filled; and then it has been launched, with the following commands : “**catkin build**”, “**source devel/setup.bash**”, “**roslaunch arm_description display.launch**” and “**roslaunch arm_gazebo arm_world.launch**”.

A terminal window titled "paolo@Paolo-Precision-3571: ~/catkin_ws" shows the following commands and output:

```
Starting >>> arm_gazebo
Finished <<< iiwa_control [ 0.2 seconds ]
Finished <<< iiwa_description [ 0.2 seconds ]
Starting >>> iiwa_gazebo
Finished <<< ros_tools [ 0.2 seconds ]
Finished <<< my_package [ 0.4 seconds ]
Finished <<< iiwa_msgs [ 0.7 seconds ]
Starting >>> iiwa_ros
Finished <<< arm_gazebo [ 0.2 seconds ]
Finished <<< iiwa_gazebo [ 0.2 seconds ]
Finished <<< iiwa_ros [ 0.3 seconds ]
Starting >>> iiwa_hw
Finished <<< iiwa_hw [ 0.3 seconds ]
Starting >>> iiwa_moveit
Finished <<< iiwa_moveit [ 0.2 seconds ]
[build] Summary: All 13 packages succeeded!
[build] Ignored: None.
[build] Warnings: None.
[build] Abandoned: None.
[build] Failed: None.
[build] Runtime: 2.2 seconds total.
paolo@Paolo-Precision-3571:~/catkin_ws$ source devel/setup.bash
paolo@Paolo-Precision-3571:~/catkin_ws$ roslaunch arm_description display.launch
```

Then it was added the xacro inside “**arm.urdf.xacro**” using “**<xacro:include filename="\$ (find arm_description)/urdf/arm.transmission.xacro"/>**”.

With “**roslaunch arm_gazebo arm_world.launch**” command the following image is displayed.



At the begin of the simulation we can notice that the manipulator stays standing on itself before it collapses due the absence of the controllers.

The same result is obtained launching the command: “**roslaunch arm_description display.launch**”.

- (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

To create the package, move to the following path “**~/catkin_ws/src/arm_stack/**”; it has been made with the following command “**catkin_create_pkg arm_control**”. Then launch and config folders must be made with the “**mkdir**” command, inside the new package and the “**arm_control.launch**” and “**arm_control.yaml**” files has been created in the respective folders: “**launch**” and “**config**”.

```

paolo@Paolo-Precision-3571: ~/catkin_ws/src/arm_stack/arm_control/config
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/config 105x55
paolo@Paolo-Precision-3571:~/catkin_ws$ ls
build devel logs src
paolo@Paolo-Precision-3571:~/catkin_ws$ cd src
paolo@Paolo-Precision-3571:~/catkin_ws/src$ ls
arm_stack docker_scripts iiwa_stack ros_package ros_tools
paolo@Paolo-Precision-3571:~/catkin_ws/src$ cd arm_stack/
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ ls
arm_description arm_gazebo
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ catkin_create_pkg arm_control
Created file arm_control/package.xml
Created file arm_control/CMakeLists.txt
Successfully created files in /home/paolo/catkin_ws/src/arm_stack/arm_control. Please adjust the values i
n package.xml.
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ ls
arm_control arm_description arm_gazebo
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ cd arm_control/
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ ls
CMakeLists.txt package.xml
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ mkdir launch
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ cd launch/
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/launch$ touch arm_control.launch
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/launch$ cd ..
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ cd launch
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/launch$ ls
arm_control.launch
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/launch$ cd ..
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ ls
CMakeLists.txt launch package.xml
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ mkdir config
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control$ cd config
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/config$ touch arm_control.yaml
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_control/config$ ls
arm_control.yaml
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/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

Inside the “`arm_control.launch`” file it has been uploaded the parameters from the “`arm_control.config`” file through the following code line:

```
<rosparam file="$(find arm_control)/config/arm_control.yaml" command="load" />.
```

The controllers have been spawned thorough the following node:

```
<node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false" output="screen" ns="$(arg robot_name)" args="$(arg controllers)" />.
```

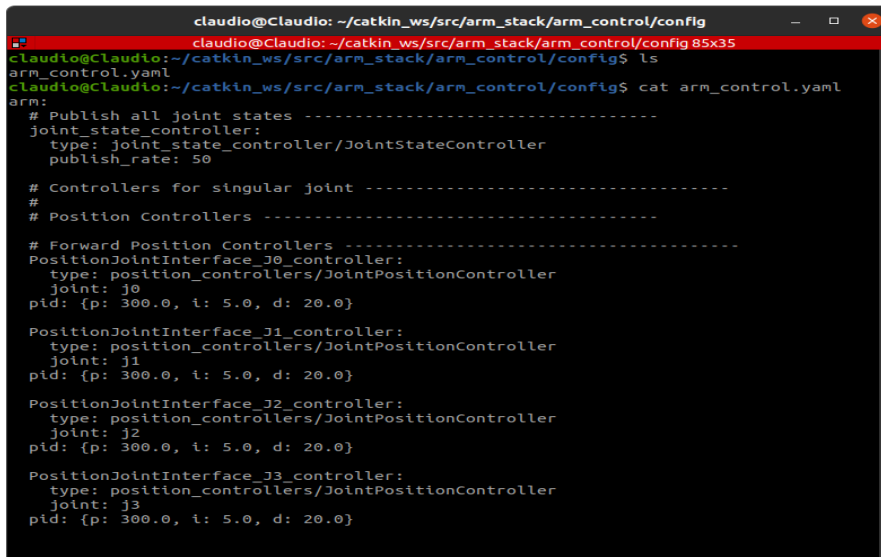
```

~/catkin_ws/src/arm_stack/arm_control/launch/arm_control.launch - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
arm_control.launch x
1 <?xml version="1.0"?>
2 <launch>
3
4   <arg name="hardware_interface" default="PositionJointInterface"/>
5   <arg name="controllers" default="PositionJointInterface_J0_controller PositionJointInterface_J1_controller PositionJointInterface_J2_controller
6   PositionJointInterface_J3_controller joint_state_controller"/>
7   <arg name="robot_name" default="arm" />
8
9   <!-- Loads joint controller configurations from YAML file to parameter server -->
10  <rosparam file="$(find arm_control)/config/arm_control.yaml" command="load" />
11
12  <!-- Loads the controllers -->
13  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" output="screen" ns="$(arg robot_name)" args="$(arg controllers)" />
14
15  <!-- Converts joint states to TF transforms for rviz, etc -->
16
17  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen">
18    <remap from="/joint_states" to="/arm/joint_states" />
19  </node>
20
21 </launch>
22
23
24
25
26
27
28
29

```

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

The file “**arm.control.yalm**” has been filled with a `joint_state_controller` and a `JointPositionController` to all the joints as it is shown in the following picture.



```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_control/config
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_control/config 85x35
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_control/config$ ls
arm_control.yaml
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_control/config$ cat arm_control.yaml
arm:
# Publish all joint states -----
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

# Controllers for singular joint -----
# Position Controllers -----

# Forward Position Controllers -----
PositionJointInterface_J0_controller:
  type: position_controllers/JointPositionController
  joint: j0
  pid: (p: 300.0, i: 5.0, d: 20.0)

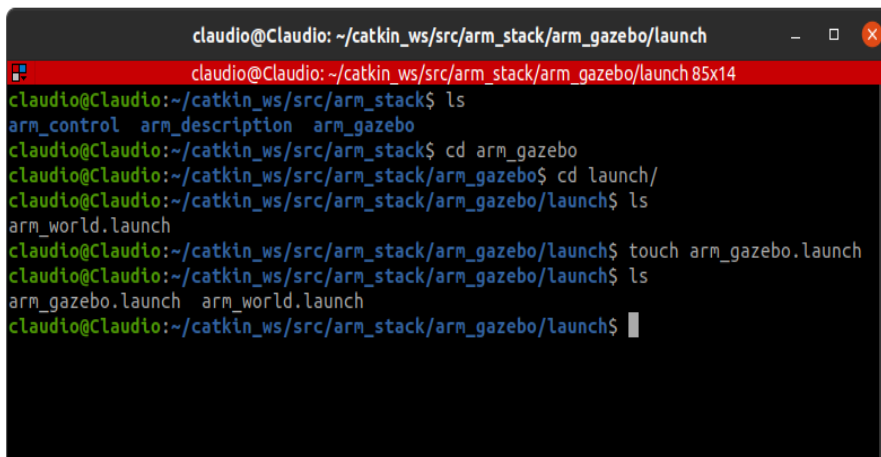
PositionJointInterface_J1_controller:
  type: position_controllers/JointPositionController
  joint: j1
  pid: (p: 300.0, i: 5.0, d: 20.0)

PositionJointInterface_J2_controller:
  type: position_controllers/JointPositionController
  joint: j2
  pid: (p: 300.0, i: 5.0, d: 20.0)

PositionJointInterface_J3_controller:
  type: position_controllers/JointPositionController
  joint: j3
  pid: (p: 300.0, i: 5.0, d: 20.0)
```

- (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

Firstly it has been created the “**arm_gazebo.launch**” file inside the launch folder of the “**arm_gazebo**” package.



```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_gazebo/launch
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_gazebo/launch 85x14
claudio@Claudio:~/catkin_ws/src/arm_stack$ ls
arm_control  arm_description  arm_gazebo
claudio@Claudio:~/catkin_ws/src/arm_stack$ cd arm_gazebo
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_gazebo$ cd launch/
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_gazebo/launch$ ls
arm_world.launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_gazebo/launch$ touch arm_gazebo.launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_gazebo/launch$ ls
arm_gazebo.launch  arm_world.launch
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_gazebo/launch$
```

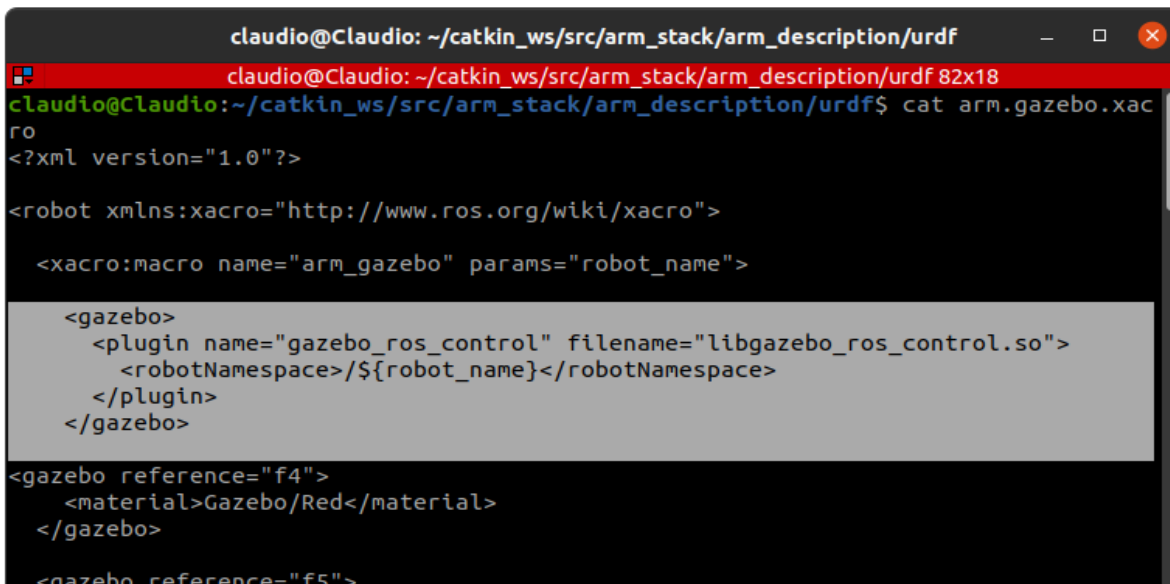
Then we get inside the `arm_control.launch` and to load the controllers it has been used:

```
<arg name="controllers" default="PositionJointInterface J0_controller  
PositionJointInterface J1_controller PositionJointInterface J2_controller  
PositionJointInterface J3_controller joint_state_controller"/>
```

A terminal window titled 'claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_control/launch' showing the command 'cat arm_control.launch' and its output. The output is an XML launch file with arguments for hardware_interface, controllers, and robot_name.

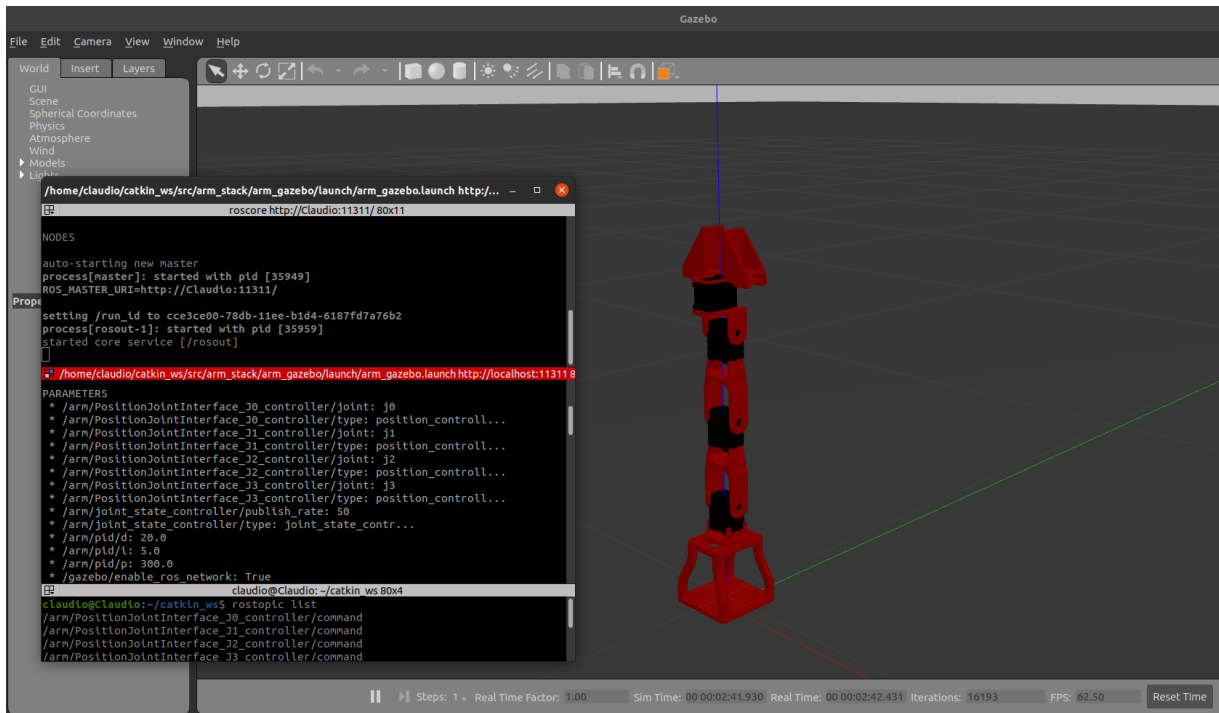
```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_control/launch  
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_control/launch$ cat arm_control.launch  
<?xml version="1.0"?>  
<launch>  
  
  <arg name="hardware_interface" default="PositionJointInterface"/>  
  <arg name="controllers" default="  
    PositionJointInterface_J0_controller  
    PositionJointInterface_J1_controller  
    PositionJointInterface_J2_controller  
    PositionJointInterface_J3_controller joint_state_controller"/>  
  <arg name="robot_name" default="arm" />  
</launch>
```

At this point we went to the “`arm_description`” package and we added the `gazebo_ros_control` plugin to our main URDF into the “`arm.gazebo.xacro`” file.

A terminal window titled 'claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf' showing the command 'cat arm.gazebo.xacro' and its output. The output is an xacro file that includes the gazebo_ros_control plugin.

```
claudio@Claudio: ~/catkin_ws/src/arm_stack/arm_description/urdf  
claudio@Claudio:~/catkin_ws/src/arm_stack/arm_description/urdf$ cat arm.gazebo.xacro  
<?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>  
    <gazebo reference="f4">  
      <material>Gazebo/Red</material>  
    </gazebo>  
    <gazebo reference="f5">
```

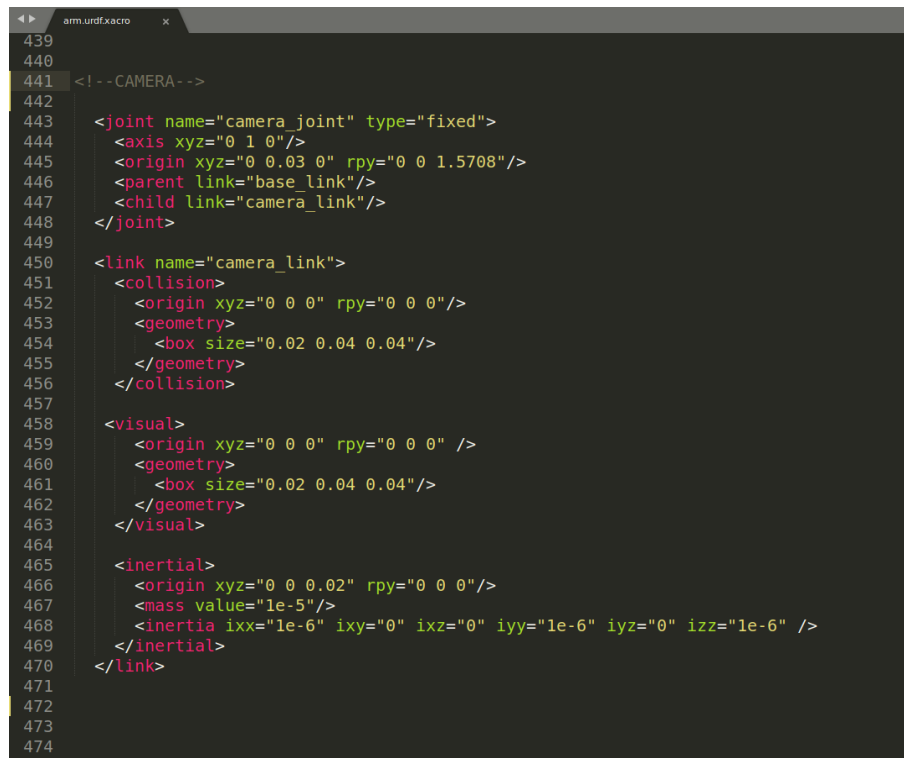
At last, it has been launched the simulation and it has been checked if our controllers were correctly loaded.



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 went inside our “`arm.urdf.xacro`” file and we have added a “`camera_link`” and a fixed “`camera_joint`” with “`base_link`” as a parent link. After that we have chosen the size and position of the camera link opportunely.



```
439
440
441 <!-- CAMERA -->
442
443 <joint name="camera_joint" type="fixed">
444   <axis xyz="0 1 0"/>
445   <origin xyz="0 0.03 0" rpy="0 0 1.5708"/>
446   <parent link="base_link"/>
447   <child link="camera_link"/>
448 </joint>
449
450 <link name="camera_link">
451   <collision>
452     <origin xyz="0 0 0" rpy="0 0 0"/>
453     <geometry>
454       <box size="0.02 0.04 0.04"/>
455     </geometry>
456   </collision>
457
458   <visual>
459     <origin xyz="0 0 0" rpy="0 0 0" />
460     <geometry>
461       <box size="0.02 0.04 0.04"/>
462     </geometry>
463   </visual>
464
465   <inertial>
466     <origin xyz="0 0 0.02" rpy="0 0 0"/>
467     <mass value="1e-5"/>
468     <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
469   </inertial>
470 </link>
471
472
473
474
```

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

As shown by the slides (74-75) we went inside the “`arm_gazebo.xacro`” file and we have added the gazebo sensor reference tags and the `libgazebo_ros_camera` plugin.

```

14 <!-- camera gazebo -->
15
16 <gazebo reference="camera_link">
17 <material>Gazebo/Yellow</material>
18 <sensor type="camera" name="camera1">
19 <update_rate>30.0</update_rate>
20 <camera name="head">
21 <horizontal_fov>1.3962634</horizontal_fov>
22 <image>
23 <width>800</width>
24 <height>800</height>
25 <format>R8G8B8</format>
26 </image>
27 <clip>
28 <near>0.02</near>
29 <far>300</far>
30 </clip>
31 <noise>
32 <type>gaussian</type>
33 <mean>0.0</mean>
34 <stddev>0.007</stddev>
35 </noise>
36 </camera>
37
38 <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
39 <alwaysOn>true</alwaysOn>
40 <updateRate>0.0</updateRate>
41 <cameraName>camera</cameraName>
42 <imageTopicName>image_raw</imageTopicName>
43 <cameraInfoTopicName>camera_info</cameraInfoTopicName>
44 <frameName>camera_link_optical</frameName>
45 <hackBaseline>0.0</hackBaseline>
46 <distortionK1>0.0</distortionK1>
47 <distortionK2>0.0</distortionK2>
48 <distortionK3>0.0</distortionK3>
49 <distortionT1>0.0</distortionT1>
50 <distortionT2>0.0</distortionT2>
51 <CxPrime>0</CxPrime>
52 <Cx>0.0</Cx>
53 <Cy>0.0</Cy>
54 <focalLength>0.0</focalLength>
55 </plugin>
56 </sensor>
57 </gazebo>
58

```

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

We have launched the simulation with the command “`roslaunch arm_gazebo.launch`” and we have checked if the image topic is correctly published using “`rqt_image_view`”.

```

tony@tony-Inspiron-5584: ~/catkin_ws/src/arm_stack/arm_gazebo
$ /home/tony/catkin_ws/src/arm_stack/arm_gazebo/launch/arm_gazebo.launch http://localhost:11311/
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_gazebo$ roslaunch arm_gazebo arm_gazebo.launch
... logging to /home/tony/.ros/log/1df41676-78df-11ee-8c93-a57d0f2aa487/r
roslaunch-tony-Inspiron-5584-21443.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop th
e option.
started roslaunch server http://tony-Inspiron-5584:41975/

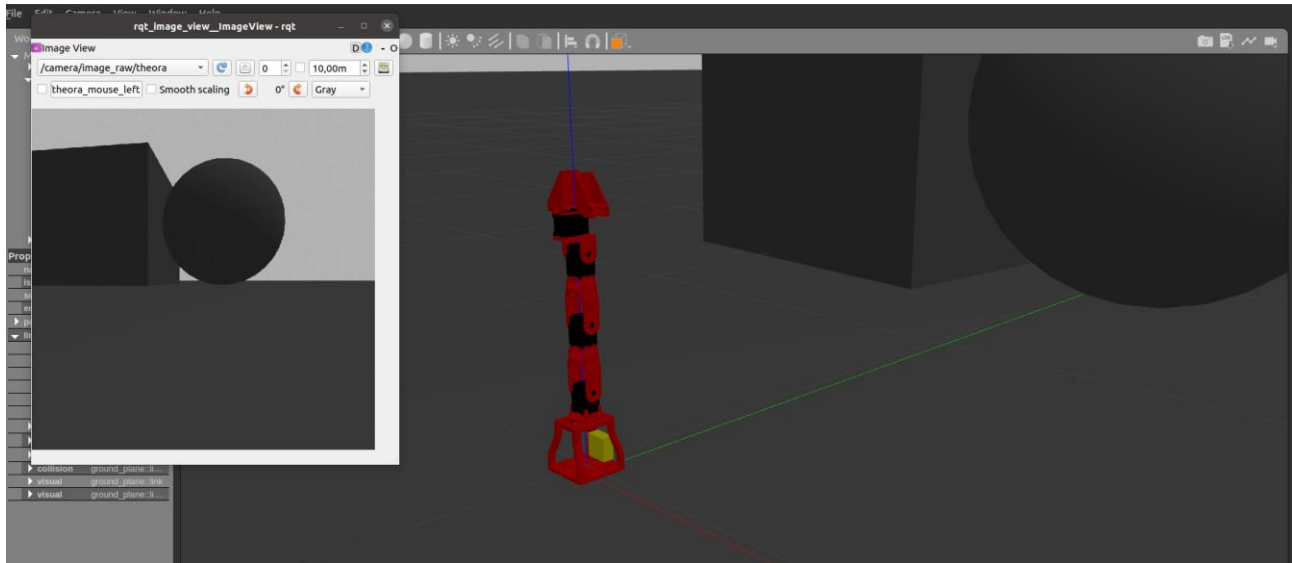
SUMMARY
=====
PARAMETERS
 * /roscdistro: noetic
 * /rosversion: 1.16.0

NODES
auto-starting new master
process[master]: started with pid [19959]
ROS_MASTER_URI=http://tony-Inspiron-5584:11311/

setting /run_id to 1df41676-78df-11ee-8c93-a57d0f2aa487
process[roscout-1]: started with pid [19969]
started core service [/roscout]

tony@tony-Inspiron-5584:~/catkin_ws$ rqt_image_view
/camera/camera_info
/camera/image_raw
/camera/image_raw/compressed
/camera/image_raw/compressed/parameter_descriptions
/camera/image_raw/compressed/parameter_updates
/camera/image_raw/compressedDepth
/camera/image_raw/compressedDepth/parameter_descriptions
/camera/image_raw/compressedDepth/parameter_updates
/camera/image_raw/theora
/camera/image_raw/theora/parameter_descriptions
/camera/image_raw/theora/parameter_updates
/camera/parameter_descriptions
/camera/parameter_updates
/clicked point

```



- (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>`

We have created the “**camera.xacro**” file and then we have filled it with the one downloaded from the GitHub link. At last, we have modified opportunely the box sizes and the color.

```

tony@tony-Inspiron-5584: ~/catkin_ws/src/arm_stack/arm_description/urdf
tony@tony-Inspiron-5584: ~/catkin_ws/src/arm_stack/arm_description/urdf 97x24
tony@tony-Inspiron-5584:~$ roscd arm_description
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_description$ ls
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_description/urdf$ touch camera.urdf.xacro
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_description/urdf$ ls
arm.gazebo.xacro  arm.transmission.xacro  arm.urdf.xacro  camera.urdf.xacro
tony@tony-Inspiron-5584:~/catkin_ws/src/arm_stack/arm_description/urdf$ 

```


In “**camera.urdf.xacro**” file there is a **<xacro:macro>** **camera_sensor** to pass xyz, rpy and parent in order to we can assign from my robot file “**arm.urdf.xacro**” the coordinates xyz of the camera origin, the orientation rpy and as parent the link base_link.

```

1 <?xml version="1.0"?>
2 <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="camera">
3
4
5
6 <xacro:macro name="camera_sensor" params="xyz rpy parent">
7
8   <joint name="camera_sensor_joint" type="fixed">
9     <axis xyz="0 1 0" />
10    <origin xyz="{xyz}" rpy="{rpy}" />
11    <parent link="{parent}" />
12    <child link="camera_link" />
13  </joint>
14
15  <link name="camera_link">
16    <collision>
17      <origin xyz="0 0 0" rpy="0 0 0" />
18      <geometry>
19        <box size="0.02 0.04 0.04" />
20      </geometry>
21    </collision>
22    <visual>
23      <origin xyz="0 0 0" rpy="0 0 0" />
24      <geometry>
25        <box size="0.02 0.04 0.04" />
26      </geometry>
27    </visual>
28    <inertial>
29      <mass value="0.0001" />
30      <origin xyz="0 0 0" rpy="0 0 {pi}" />
31      <inertia ixx="0.0000001" ixy="0" ixz="0" iyy="0.0000001" iyz="0" izz="0.0000001" />
32    </inertial>
33  </link>
34
35  <gazebo reference="camera_link">
36    <material>Gazebo/Yellow</material>
37    <sensor type="camera" name="camera">
38      <update_rate>30.0</update_rate>
39      <camera name="head">
40        <horizontal_fov>1.3962634</horizontal_fov>
41        <image>
42          <width>800</width>
43          <height>600</height>
44          <format>R8G8B8</format>
45        </image>
46      </camera>
47    </sensor>
48  </gazebo>
49 </robot>

```

We have included the camera inside the “**arm.urdf.xacro**” file through this command:
<xacro:include filename="\$ (find arm_description)/urdf/camera.urdf.xacro"/> .

In file “**arm.urdf.xacro**” we have assigned “0 0 1.5708” at *rpy* parameter because we wanted that the camera to be directed along the y axis, this is the axis along which the robot can move its links in the initial condition and so we have rotated the camera around z axis by 90° (1.5708 radians).

```

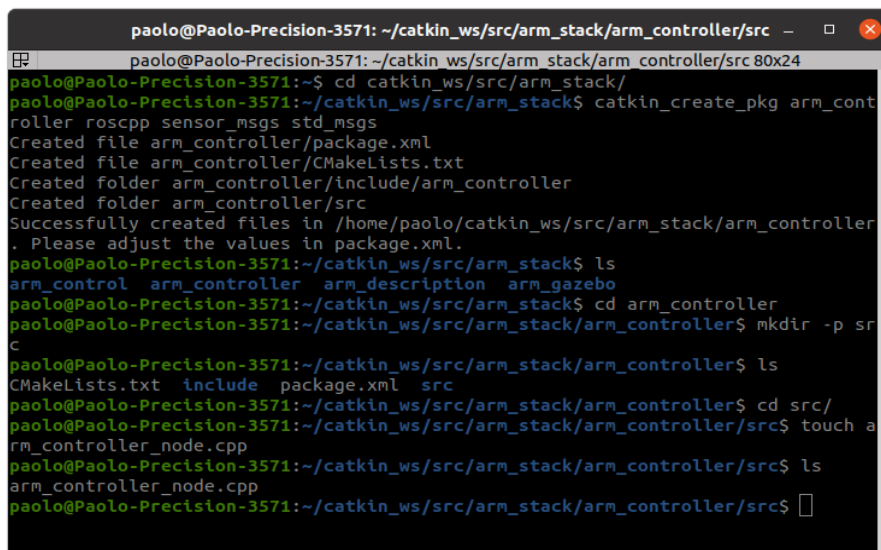
1 <?xml version="1.0"?>
2
3 <robot name="arm" xmlns:xacro="http://www.ros.org/wiki/xacro">
4
5   <xacro:include filename="$ (find arm_description)/urdf/arm.gazebo.xacro"/>
6   <xacro:include filename="$ (find arm_description)/urdf/arm.transmission.xacro"/>
7   <xacro:include filename="$ (find arm_description)/urdf/camera.urdf.xacro"/>
8
9   <xacro:camera_sensor xyz="0 0.03 0" rpy="0 0 1.5708" parent="base_link"/>
10
11   <xacro:arm_gazebo robot_name="arm" />
12   <xacro:arm_transmission robot_name="arm" hardware_interface="PositionJointInterface"/>
13
14 </robot>

```

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

The “`arm_controller`” package has been created with the following command “`catkin_create_pkg arm_controller roscpp sensor_msgs std_msgs`”, where the last three represent dependencies, in the following folder “`~/catkin_ws/src/arm_stack`”. Subsequently to create the “`arm_controller_node.cpp`” file the “`src`” folder is created in the “`~/catkin_ws/src/arm_stack/arm_controller`” path and then create the “`arm_controller_node.cpp`” file inside with the “`touch`” command.

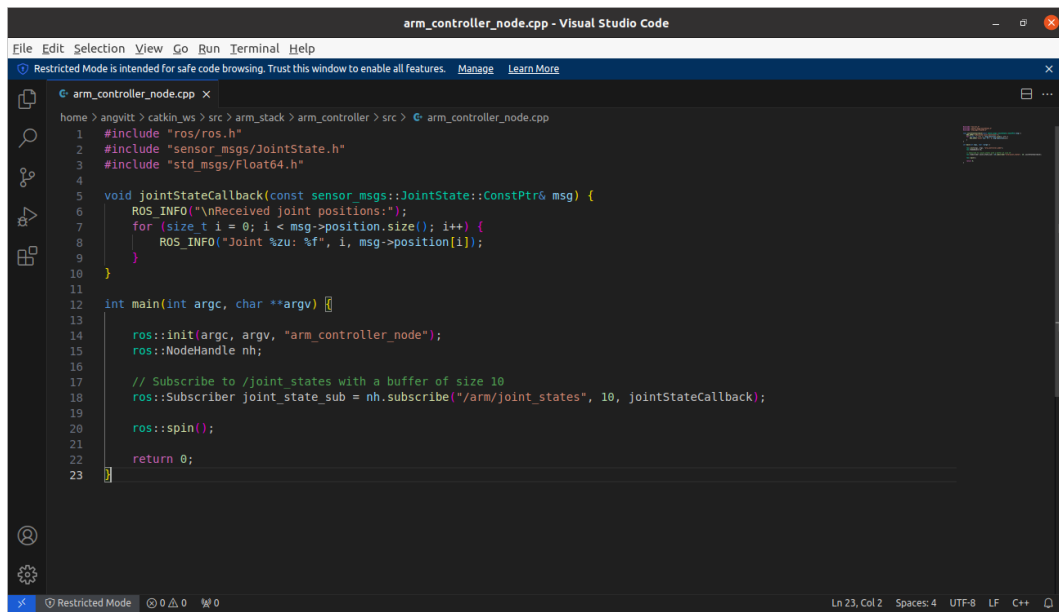


```
paolo@Paolo-Precision-3571: ~/catkin_ws/src/arm_stack/arm_controller/src - 80x24
paolo@Paolo-Precision-3571:~$ cd catkin_ws/src/arm_stack/
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ catkin_create_pkg arm_controller roscpp sensor_msgs std_msgs
Created file arm_controller/package.xml
Created file arm_controller/CMakeLists.txt
Created folder arm_controller/include/arm_controller
Created folder arm_controller/src
Successfully created files in /home/paolo/catkin_ws/src/arm_stack/arm_controller. Please adjust the values in package.xml.
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ ls
arm_control  arm_controller  arm_description  arm_gazebo
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack$ cd arm_controller
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller$ mkdir -p src
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller$ ls
CMakeLists.txt  include  package.xml  src
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller$ cd src/
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller/src$ touch arm_controller_node.cpp
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller/src$ ls
arm_controller_node.cpp
paolo@Paolo-Precision-3571:~/catkin_ws/src/arm_stack/arm_controller/src$
```

The “`CmakeLists.txt`” file has been modified: “`add_executable`” and “`target_link_libraries`” lines are uncommented. So the following lines have been added: “`add_executable(arm_controller_node src/arm_controller_node.cpp)`” and “`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`

The subscriber is created in the "arm_controller_node.cpp" file with the name "joint_state_sub" to the topic "/arm/joint_states" with a buffer of size 10 using the command "ros::Subscriber". The subscriber calls the callback called "jointStateCallback" which prints the current joint position.



```

arm_controller_node.cpp - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

arm_controller_node.cpp x
home > angvitt > catkin_ws > src > arm_stack > arm_controller > src > arm_controller_node.cpp
1 #include "ros/ros.h"
2 #include "sensor_msgs/JointState.h"
3 #include "std_msgs/Float64.h"
4
5 void jointStateCallback(const sensor_msgs::JointState::ConstPtr& msg) {
6     ROS_INFO("\nReceived joint positions:");
7     for (size_t i = 0; i < msg->position.size(); i++) {
8         ROS_INFO("Joint %zu: %f", i, msg->position[i]);
9     }
10 }
11
12 int main(int argc, char **argv) {
13     ros::init(argc, argv, "arm_controller_node");
14     ros::NodeHandle nh;
15
16     // Subscribe to /joint_states with a buffer of size 10
17     ros::Subscriber joint_state_sub = nh.subscribe("/arm/joint_states", 10, jointStateCallback);
18
19     ros::spin();
20
21     return 0;
22 }
23
Ln 23, Col 2 Spaces: 4 UTF-8 LF C++

```

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

Four publishers are created with the name "joint{n}_pub" (where n goes from 0 to 3), one for each joint within the "arm_controller_pub.cpp" file, with the topic "/arm/PositionJointInterface_J{n}_controller/command" using the following command: "ros::Publisher". The default values of the joint positions are fixed at 0. The "arm_controller_pub.cpp" file is created in the "~/catkin_ws/src/arm_stack/arm_controller/src" path with the "touch" command. As in point 4.a the "CmakeLists.txt" file is modified, the following lines have been added: "add_executable(arm_controller_pub src/arm_controller_pub.cpp)" and "target_link_libraries(arm_controller_pub \${catkin_LIBRARIES})".

At the end several tests has been carried out, in which the joint angle of joints of the robot are modified to rotate it to a defined configuration: the angle of the joints are controlled where they are: -90° for joint 1 (shoulder), 90° for joint 2 (elbow), -90° for joint 3 (wrist), 90° for joint 0 (base).

```
arm_controller_pub.cpp - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

arm_controller_pub.cpp x
home > tony > catkin_ws > src > arm_stack > arm_controller > src > arm_controller_pub.cpp
1 #include "ros/ros.h"
2 #include "std_msgs/Float64.h"
3
4 int main(int argc, char **argv) {
5
6     ros::init(argc, argv, "arm_controller_pub");
7     ros::NodeHandle nh;
8
9     ros::Publisher joint0_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J0_controller/command", 1);
10    ros::Publisher joint1_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J1_controller/command", 1);
11    ros::Publisher joint2_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J2_controller/command", 1);
12    ros::Publisher joint3_pub = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J3_controller/command", 1);
13
14    ros::Rate loop_rate(10);
15
16    while (ros::ok())
17    {
18
19        std_msgs::Float64 joint0_command;
20        joint0_command.data = 0;
21        joint0_pub.publish(joint0_command);
22
23        std_msgs::Float64 joint1_command;
24        joint1_command.data = 0;
25        joint1_pub.publish(joint1_command);
26
27        std_msgs::Float64 joint2_command;
28        joint2_command.data = 0;
29        joint2_pub.publish(joint2_command);
30
31        std_msgs::Float64 joint3_command;
32        joint3_command.data = 0;
33        joint3_pub.publish(joint3_command);
34
35        ros::spinOnce();
36        loop_rate.sleep();
37
38    }
39
40    return 0;
41 }
```

```
paolo@Paolo-Precision-3571: ~
/home/paolo/catkin_ws/src/arm_stack/arm_gazebo/launch/arm_gazebo.launch http://localhost:11311/80x25
paolo@Paolo-Precision-3571:~/catkin_ws$ rosrun arm_gazebo arm_gazebo.launch
... logging to /home/paolo/.ros/log/64c0153e-7978-11ee-b692-6960fecbece2/roslaunch-Paolo-Precision-3571-79147.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Paolo-Precision-3571:41161/

SUMMARY
=====

PARAMETERS
* /arm/PositionJointInterface_J0_controller/joint: j0
* /arm/PositionJointInterface_J0_controller/type: position_controll...
* /arm/PositionJointInterface_J1_controller/joint: j1
* /arm/PositionJointInterface_J1_controller/type: position_controll...
* /arm/PositionJointInterface_J2_controller/joint: j2
* /arm/PositionJointInterface_J2_controller/type: position_controll...
* /arm/PositionJointInterface_J3_controller/joint: j3

auto-starting new master
paolo@Paolo-Precision-3571: ~ 80x25
paolo@Paolo-Precision-3571:~$ roscore
... logging to /home/paolo/.ros/log/64c0153e-7978-11ee-b692-6960fecbece2/roslaunch-Paolo-Precision-3571-79147.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Paolo-Precision-3571:38397/
ros_comm version 1.16.0

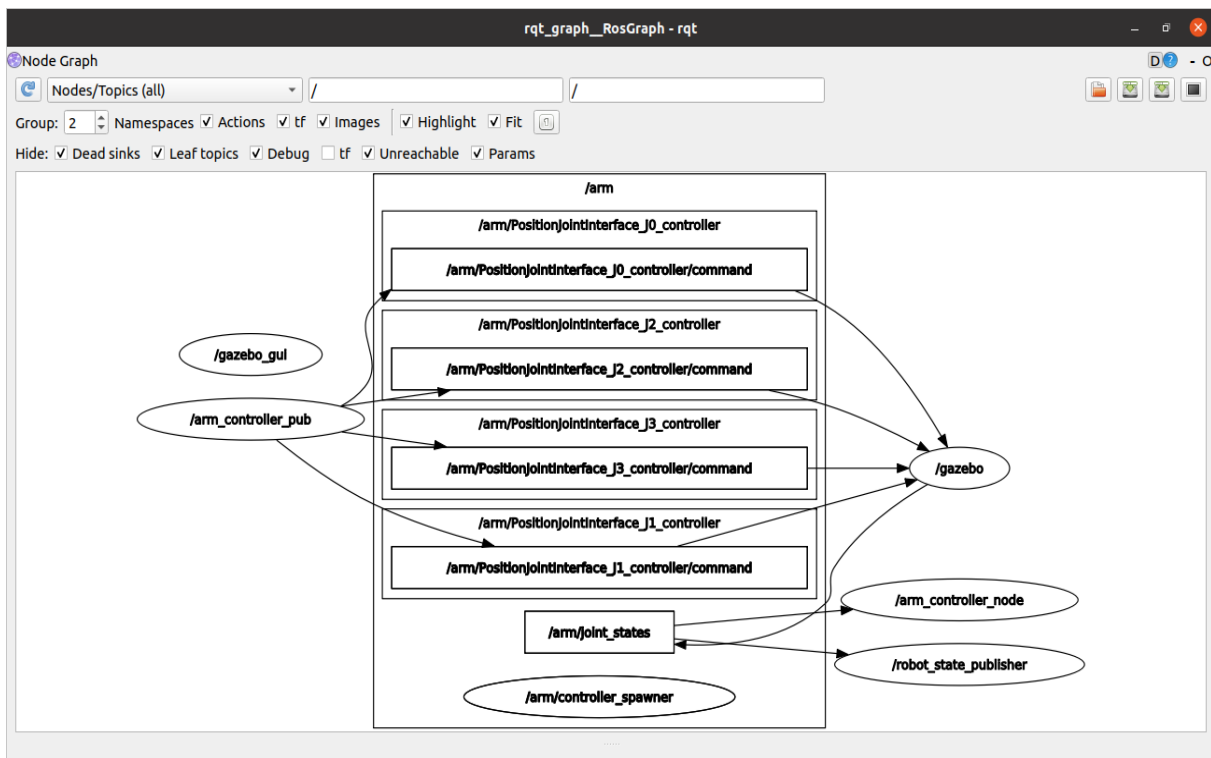
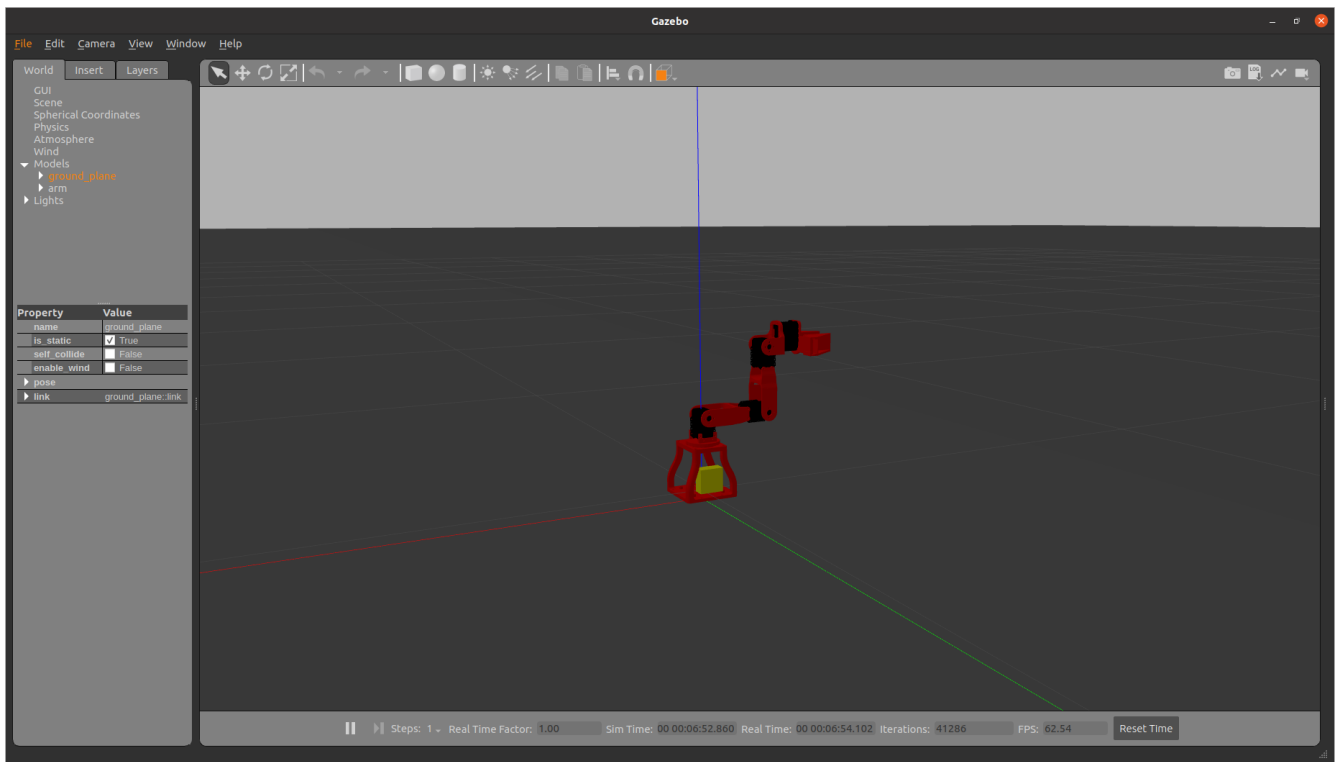
SUMMARY
=====

PARAMETERS
* /roscdistro: noetic
* /rosversion: 1.16.0

NODES

paolo@Paolo-Precision-3571:~$ rostopic pub /arm/PositionJointInterface_J0_controller/command std_msgs/Float64 "data: 1.57"
publishing and latching message. Press ctrl-C to terminate
^Cpaolo@Paolo-Precision-3571:~$ rostopic pub /arm/PositionJointInterface_J1_controller/command std_msgs/Float64 "data: -1.57"
publishing and latching message. Press ctrl-C to terminate
^Cpaolo@Paolo-Precision-3571:~$ rostopic pub /arm/PositionJointInterface_J2_controller/command std_msgs/Float64 "data: 1.57"
publishing and latching message. Press ctrl-C to terminate
^Cpaolo@Paolo-Precision-3571:~$ rostopic pub /arm/PositionJointInterface_J3_controller/command std_msgs/Float64 "data: -1.57"
publishing and latching message. Press ctrl-C to terminate
^Cpaolo@Paolo-Precision-3571:~$

[ INFO] [1698927253.594609907, 387.820000000]: Received joint positions:
[ INFO] [1698927253.594862785, 387.820000000]: Joint 0: 1.570000
[ INFO] [1698927253.594920820, 387.820000000]: Joint 1: -1.570000
[ INFO] [1698927253.594967955, 387.820000000]: Joint 2: 1.570000
[ INFO] [1698927253.594998164, 387.820000000]: Joint 3: -1.570000
[ INFO] [1698927253.615527147, 387.840000000]: Received joint positions:
[ INFO] [1698927253.615723913, 387.840000000]: Joint 0: 1.570000
[ INFO] [1698927253.615766031, 387.840000000]: Joint 1: -1.570000
[ INFO] [1698927253.615795580, 387.840000000]: Joint 2: 1.570000
[ INFO] [1698927253.615821694, 387.840000000]: Joint 3: -1.570000
[ INFO] [1698927253.635263632, 387.860000000]: Received joint positions:
[ INFO] [1698927253.635494058, 387.860000000]: Joint 0: 1.570000
[ INFO] [1698927253.635533384, 387.860000000]: Joint 1: -1.570000
[ INFO] [1698927253.635570126, 387.860000000]: Joint 2: 1.570000
[ INFO] [1698927253.635608660, 387.860000000]: Joint 3: -1.570000
[ INFO] [1698927253.654999523, 387.880000000]: Received joint positions:
[ INFO] [1698927253.655147369, 387.880000000]: Joint 0: 1.570000
[ INFO] [1698927253.655219704, 387.880000000]: Joint 1: -1.570000
[ INFO] [1698927253.655313995, 387.880000000]: Joint 2: 1.570000
[ INFO] [1698927253.655384766, 387.880000000]: Joint 3: -1.570000
```



By rqt_graph you can see the logic of the communication between nodes and topics. In our case the publishers node gives commands to controllers (/arm/PositionJointInterface_Jx_controller/command topics), gazebo node receives information by the controllers and it provides this data to arm/joint_states topic that provides them to subscriber node (arm_controller_node).