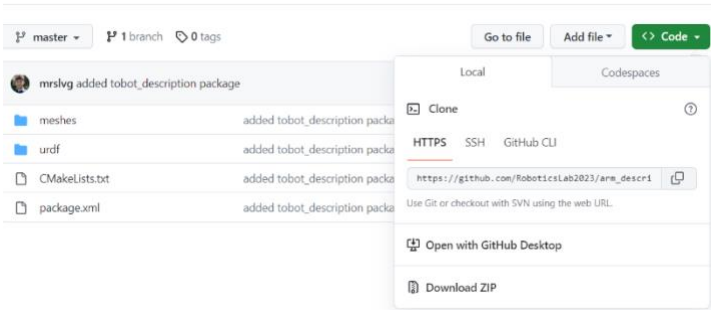
Report – Homework 1: Building your robot manipulator

Student:
Nicola Caliendo

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

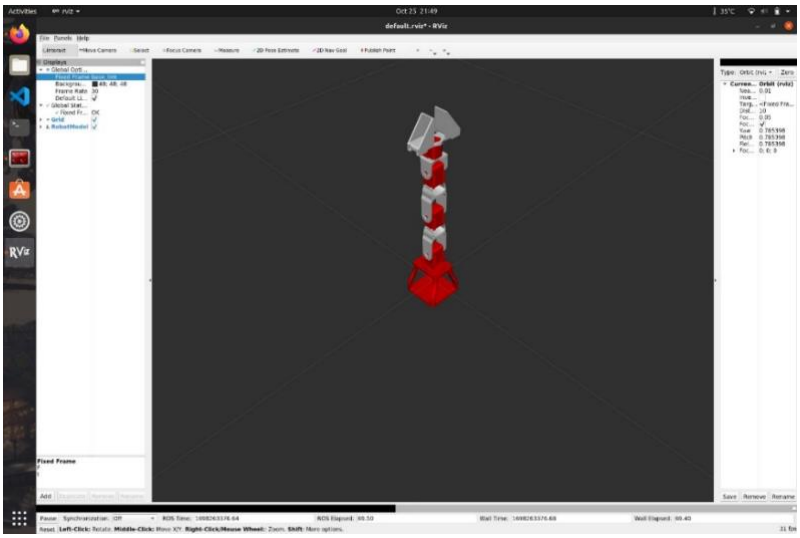
a) I downloaded the *"arm_description"* package by copying the repo in our catkin workspace folder.



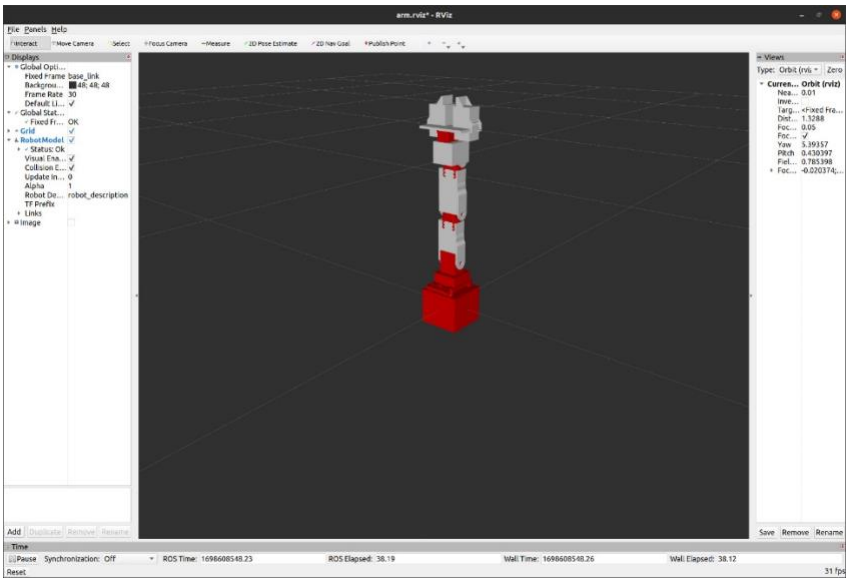
b) I created a launch folder containing a *"display.launch"* file.

```
<?xml version="1.0"?>
<launch>
  <param name="robot_description" command="$(find xacro)/xacro $(find arm_description)/urdf/arm.urdf.xacro"/>
     <node pkg="robot_state_publisher" type="robot_state_publisher" name="rob_st_pub"/>
     <node pkg="joint_state_publisher" type="joint_state_publisher" name="joi_st_pub"/>
     <node type="rviz" name="rviz" pkg="rviz" args="-d $(find arm_description)/rviz/arm.rviz" required="true" />
</launch>
```

This file loads the URDF as a *"robot_description"* ROS parameter. Also, it starts the *"robot_state_publisher", "joint_state_publisher"* and the rviz node. After launching the file for the first time, I added the "RobotModel" plugin interface, I saved the rviz configuration file and I put it as an argument to the node.



c) After getting the measurement from the original *".stl"* collision meshes files, we substituted all the collision geometries with boxes of similar dimensions.



d) I created *"arm.gazebo.xacro"* that contains a *"xacro:macro"* that includes all the *"<gazebo tags>"* of the URDF and we included it inside the original URDF

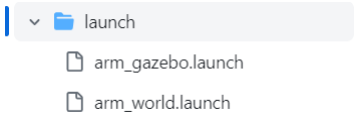
```
<?xml version="1.0"?>

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

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

    <!-- Load Gazebo lib and set the robot namespace -->
    <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">
      <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">
```

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

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

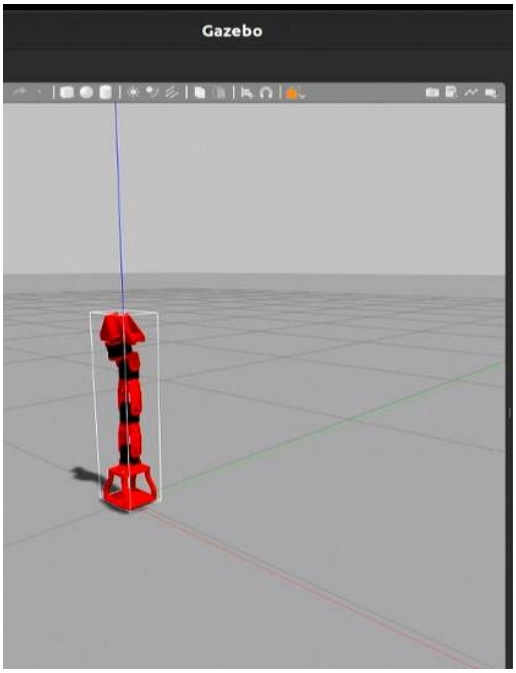
2. Add transmission and controllers to you robot and spawn it in Gazebo

a) I created the *"arm_gazebo"* package.

b) that contains a "launch" folder with the *"arm_world.launch"* file.



c) Following the example from the *"iiwa_stack"* package, I created the *"arm_world.lauch"* file. This file includes an *"arm_upload.launch"* that loads the URDF with the given hardware interface and robot name into the ROS Parameter Server. I proceeded to launch *"arm_world.launch"*.

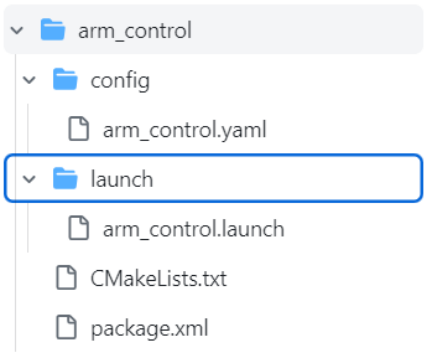


d) I created a *"arm.transmission.xacro"* file containing a *"xacro:macro"* with the hardware interface and I loaded it in my *"arm.urdf.xacro"* file.

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

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

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

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

    <transmission name="${robot_name}_tran_2">
```

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

e) I created the *"arm_control"* package.



f) I filled the *"arm_control.launch"* file with commands that load the joint controller configuration from the *".yaml"* file to the parameter server and spawn the controllers.

```
13        <!-- Loads joint controller configurations from YAML file to parameter server -->
14        <rosparam file="$(find arm_control)/config/arm_control.yaml" command="load" ns="/arm"/>
```

```
18        <!-- Loads the controllers -->
19        <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
20              output="screen" ns="/arm" args="joint_state_controller
21                    $(arg hardware_interface)_J0_controller
22                    $(arg hardware_interface)_J1_controller
23                    $(arg hardware_interface)_J2_controller
24                    $(arg hardware_interface)_J3_controller">
25        </node>
26
27        <!-- Converts joint states to TF transforms for rviz, etc -->
28        <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
29              respawn="false" output="screen">
30            <remap from="joint_states" to="/$(arg robot_name)/joint_states" />
31            <param name="publish_frequency" value="$(arg robot_state_frequency)" />
32        </node>
33
34    </launch>
```

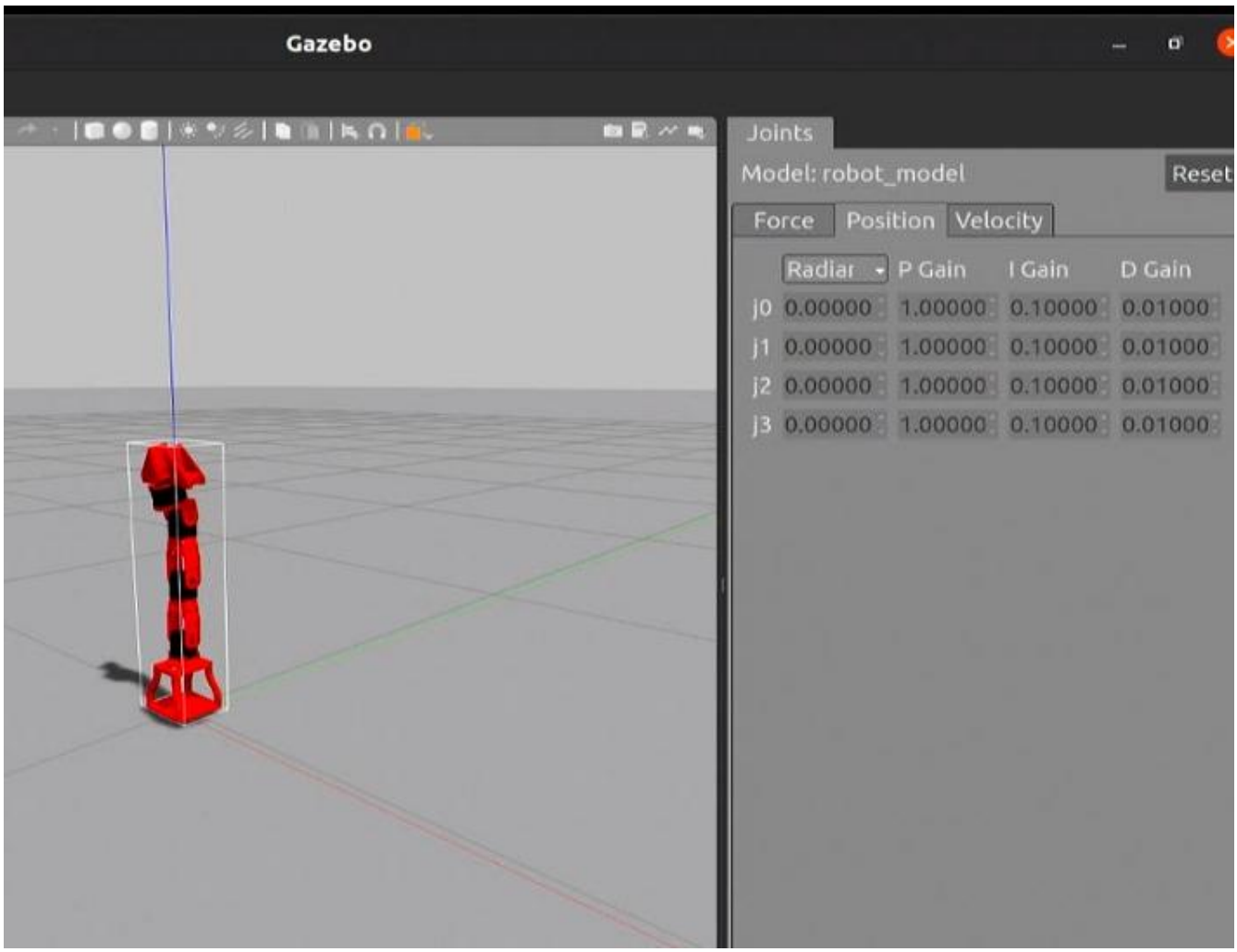
g) In the *".yaml"* file I added a *"joint_state_controller"* and *"JointPositionController*" to all the joints

```
1     #arm:
2       # Publish all joint states ----------------------------
3       joint_state_controller:
4         type: joint_state_controller/JointStateController
5         publish_rate: 50
6
7       # Controllers for singular joint ---------------------
8       #
9
10      # Position Controllers ---------------------------------
11      PositionJointInterface_J0_controller:
12        type: position_controllers/JointPositionController
13        joint: j0
14        pid: {p: 100.0, i: 0.01, d: 10.0}
15
16      PositionJointInterface_J1_controller:
17        type: position_controllers/JointPositionController
18        joint: j1
19        pid: {p: 100.0, i: 0.01, d: 10.0}
20
21      PositionJointInterface_J2_controller:
22        type: position_controllers/JointPositionController
23        joint: j2
24        pid: {p: 100.0, i: 0.01, d: 10.0}
25
26      PositionJointInterface_J3_controller:
27        type: position_controllers/JointPositionController
28        joint: j3
29        pid: {p: 100.0, i: 0.01, d: 10.0}
30
```

h) I created an *"arm_gazebo.launch"* file that includes both *"arm_world.launch"* and *"arm_control.launch"*.

```
21          <!-- Loads the Gazebo world. -->
22          <include file="$(find arm_gazebo)/launch/arm_world.launch"/>
23
24          <!-- Loads Controllers -->
25          <include file="$(find arm_control)/launch/arm_control.launch"/>
26
```

I proceeded to launch the file to load the simulation and check if the controllers were correctly loaded.



3. Add a camera sensor to your robot

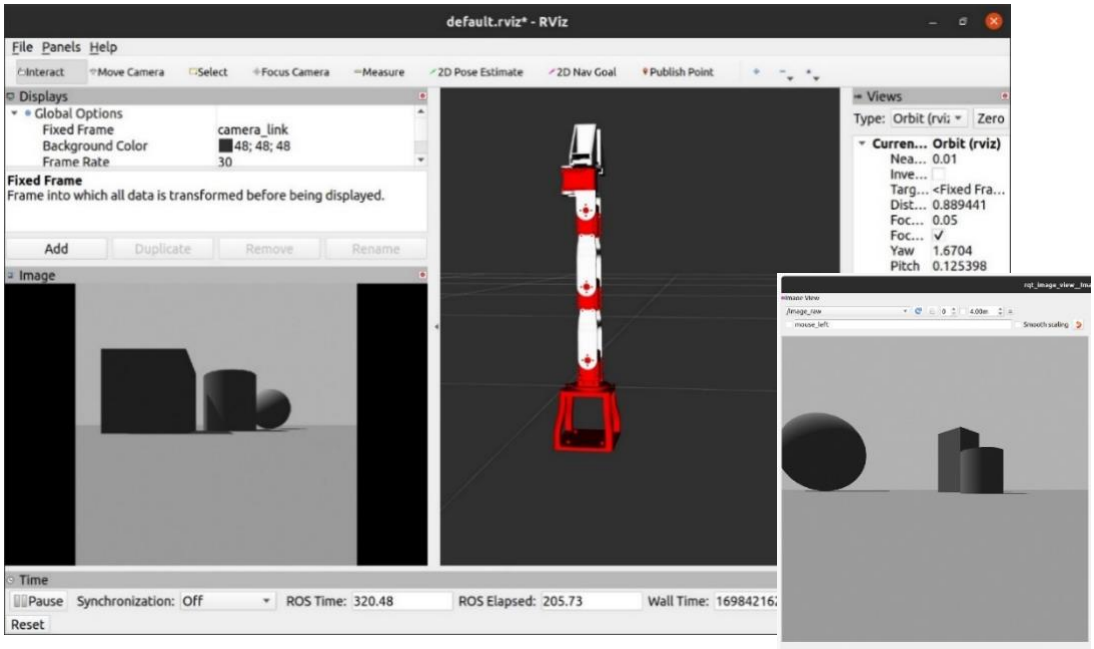
a) I directly created a *"camera.xacro"* file and I added to the robot "URDF" using *"<xacro:include>"*

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

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

  <xacro:macro name="arm_camera">

    <joint name="camera_sensor_joint" type="fixed">
        <axis xyz="0 1 0" />
        <origin xyz="-0.0021 -0.036 0.056" rpy = "0 0 -1.57"/>
        <parent link="base_link"/>
        <child link="camera_link"/>
    </joint>

    <link name="camera_link">
        <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="0.002 0.008 0.005"/>
            </geometry>
        </collision>
        <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="0.002 0.008 0.005"/>
            </geometry>
        </visual>
        <inertial>
        <mass value="0.0001" />
        <origin xyz="0 0 0" rpy="0 0 ${pi}"/>
        <inertia ixx="0.0000001" ixy="0" ixz="0" iyy="0.0000001" iyz="0" izz="0.0000001" />
        </inertial>
    </link>
  </xacro:macro>
</robot>
```

b) I added the sensor reference tags and *"libgazebo_ros_camera"* plugin to the *"arm.gazebo.xacro"*

```
14        <gazebo reference="camera_link">
15          <sensor type="camera" name="camera1">
16            <update_rate>30.0</update_rate>
17            <camera name="head">
18              <horizontal_fov>1.3962634</horizontal_fov>
19              <image>
20                <width>800</width> <height>800</height> <format>
21                R8G8B8</format>
22              </image>
23              <clip>
24                <near>0.02</near> <far>300</far>
25              </clip>
26              <noise>
27                    <type>gaussian</type> <mean>0.0</mean> <stddev>0.007
28                  </stddev>
29                </noise>
30              </camera>
31              <plugin name="camera_controller" filename="
32                libgazebo_ros_camera.so"> ... </plugin>
33              <plugin name="camera_controller" filename="
34                libgazebo_ros_camera.so">
35                <alwaysOn>true</alwaysOn>
36                <updateRate>0.0</updateRate>
37                <cameraName>camera</cameraName>
38                <imageTopicName>image_raw</imageTopicName>
39                <cameraInfoTopicName>camera_info</cameraInfoTopicName>
40                <frameName>camera_link_optical</frameName>
41                <hackBaseline>0.0</hackBaseline>
42                <distortionK1>0.0</distortionK1>
43                <distortionK2>0.0</distortionK2>
44                <distortionK3>0.0</distortionK3>
45                <distortionT1>0.0</distortionT1>
```

c) I proceeded to load the simulation with *"arm_gazebo.launch"* and check if the image was correctly published using the *"rqt_image_view"* plugin.



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

a) I created an *"arm_controller"* package containing the ROS C++ node named *"arm_controller_node"*. I proceeded to modify the "*CMakeLists.txt"* to compile the node with the required dependencies.

```
10    find_package(catkin REQUIRED COMPONENTS
11      roscpp
12      sensor_msgs
13      std_msgs
14    )
15
```

```
104   catkin_package(
105   #  INCLUDE_DIRS include
106   #  LIBRARIES arm_controller
107     CATKIN_DEPENDS roscpp sensor_msgs std_msgs
108   #  DEPENDS system_lib
109   )
```

```
117    include_directories(
118    # include
119      ${catkin_INCLUDE_DIRS}
120    )
```

```
136   add_executable(${PROJECT_NAME}_node src/arm_cont_node.cpp)
137   target_link_libraries(${PROJECT_NAME}_node ${catkin_LIBRARIES})
```

b) I created a subscriber to the topic *"joint_states"* and a callback function that prints the current joint positions.

```
7   void jointStatesCallback(const sensor_msgs::JointState::ConstPtr& msg)
8   {
9       ROS_INFO("Received joint states:");
10      for (size_t i = 0; i < msg->name.size(); i++)
11      {
12          ROS_INFO("%s: %f", msg->name[i].c_str(), msg->position[i]);
13      }
14  }
```

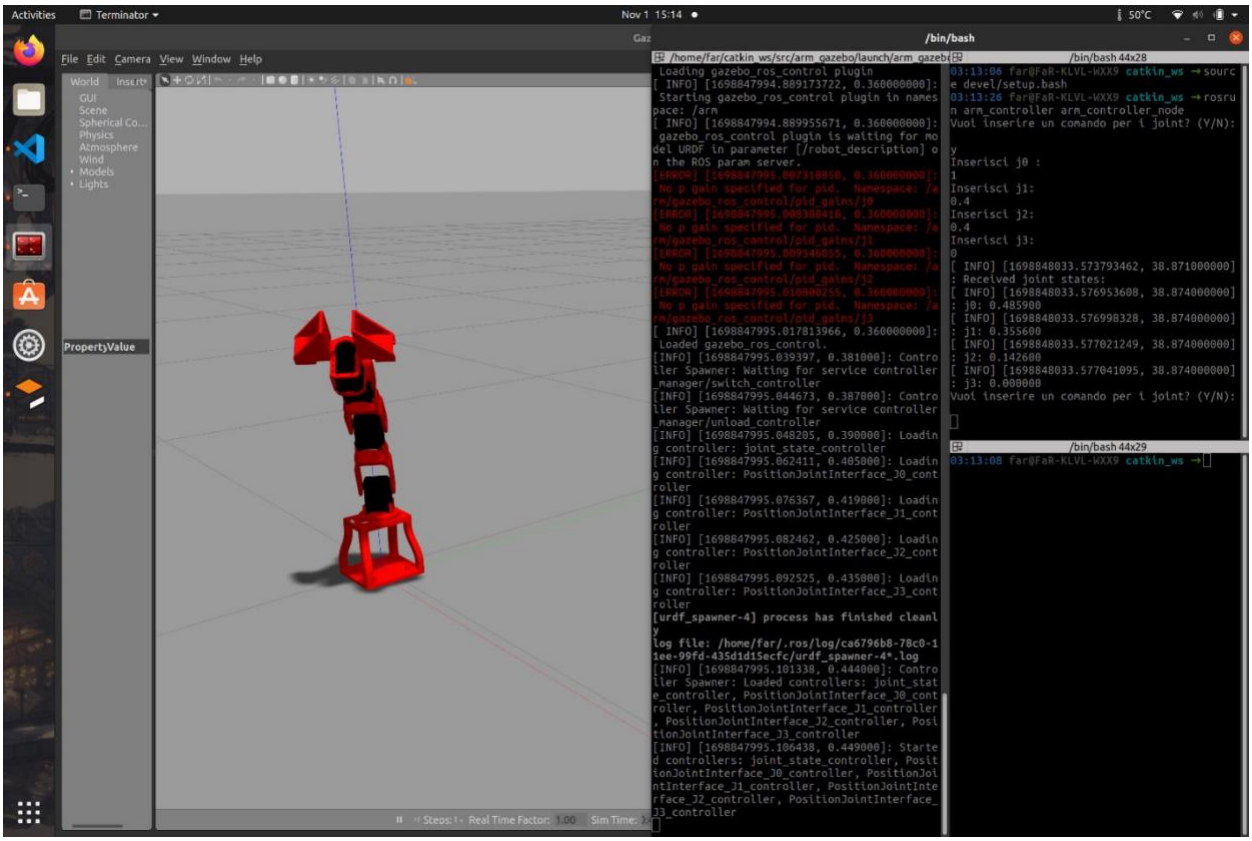
```
21      // Create a subscriber for the "joint_states" topic
22      ros::Subscriber sub = nh.subscribe("/arm/joint_states", 1, jointStatesCallback);
```

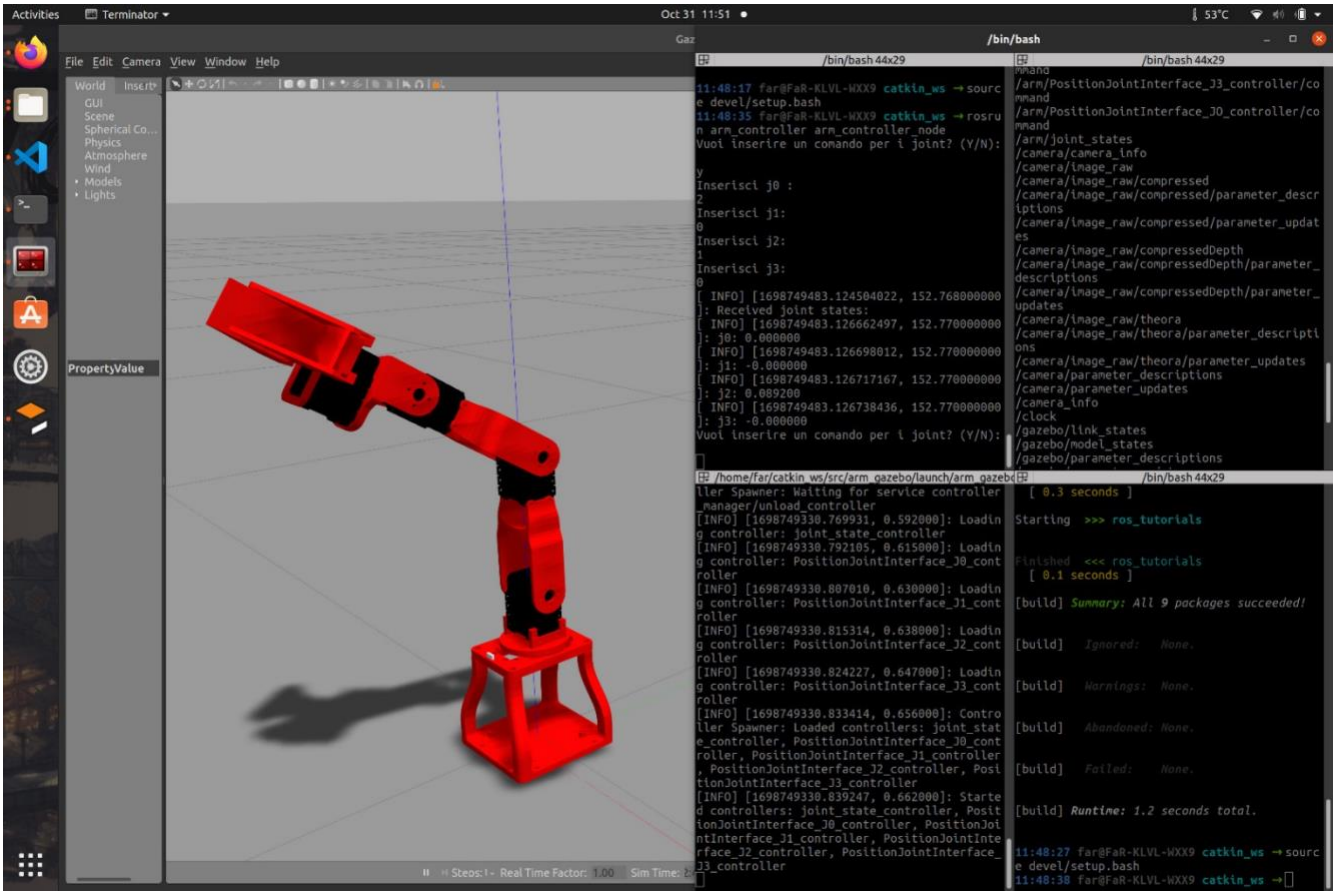
c) I created four subscribers that write commands onto the controller's *"/command"* topics.

```
25      ros::Publisher pub0 = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J0_controller/command", 1000);
26      ros::Publisher pub1 = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J1_controller/command", 1000);
27      ros::Publisher pub2 = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J2_controller/command", 1000);
28      ros::Publisher pub3 = nh.advertise<std_msgs::Float64>("/arm/PositionJointInterface_J3_controller/command", 1000);
```

```
35          while (ros::ok()) {
36              msg.data=0;
37              char r='0';
38
39              std::cout<<"Vuoi inserire un comando per i joint? (Y/N): "<<std::endl;
40              std::cin>>r;
41
42              //for(int i=0;i<4;i++){
43                  if(r=='Y' || r=='y'){
44                      std::cout<<"Inserisci j0 : "<<std::endl;
45                      std::cin>>msg.data;
46                      pub0.publish(msg);
47
48                      std::cout<<"Inserisci j1: "<<std::endl;
49                      std::cin>>msg.data;
50                      pub1.publish(msg);
51
52                      std::cout<<"Inserisci j2: "<<std::endl;
53                      std::cin>>msg.data;
54                      pub2.publish(msg);
55
56                      std::cout<<"Inserisci j3: "<<std::endl;
57                      std::cin>>msg.data;
58                      pub3.publish(msg);
59                  }
```

Now we can launch the controller node together with *"arm_gazebo.launch"* to submit requests to the position controllers.




Activities
Terminator
Oct 31 11:51
53°C
/bin/bash
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
PropertyValue
/bin/bash 44x29
11:48:17 far@FaR-KLVL-WXX9 catkin_ws → source devel/setup.bash
11:48:35 far@FaR-KLVL-WXX9 catkin_ws → rosrun arm_controller arm_controller_node
Vuoi inserire un comando per i joint? (Y/N):
y
Inserisci j0 :
2
Inserisci j1:
0
Inserisci j2:
1
Inserisci j3:
0
[INFO] [1698749483.124504022, 152.768000000]: Received joint states:
[INFO] [1698749483.126662497, 152.770000000]: j0: 0.000000
[INFO] [1698749483.126698012, 152.770000000]: j1: -0.000000
[INFO] [1698749483.126717167, 152.770000000]: j2: 0.089200
[INFO] [1698749483.126738436, 152.770000000]: j3: -0.000000
Vuoi inserire un comando per i joint? (Y/N):
/arm/PositionJointInterface_J3_controller/command
/arm/PositionJointInterface_J0_controller/command
/arm/joint_states
/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
/camera_info
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/home/far/catkin_ws/src/arm_gazebo/launch/arm_gazebo
[INFO] [1698749330.769931, 0.592000]: Loading controller: joint_state_controller
[INFO] [1698749330.792105, 0.615000]: Loading controller: PositionJointInterface_J0_controller
[INFO] [1698749330.807010, 0.630000]: Loading controller: PositionJointInterface_J1_controller
[INFO] [1698749330.815314, 0.638000]: Loading controller: PositionJointInterface_J2_controller
[INFO] [1698749330.824227, 0.647000]: Loading controller: PositionJointInterface_J3_controller
[INFO] [1698749330.833414, 0.656000]: Controller Spawner: Loaded controllers: joint_state_controller, PositionJointInterface_J0_controller, PositionJointInterface_J1_controller, PositionJointInterface_J2_controller, PositionJointInterface_J3_controller
[INFO] [1698749330.839247, 0.662000]: Started controllers: joint_state_controller, PositionJointInterface_J0_controller, PositionJointInterface_J1_controller, PositionJointInterface_J2_controller, PositionJointInterface_J3_controller
[0.3 seconds]
Starting >>> ros_tutorials
Finished <<< ros_tutorials [0.1 seconds]
[build] Summary: All 9 packages succeeded!
[build] Ignored: None.
[build] Warnings: None.
[build] Abandoned: None.
[build] Failed: None.
[build] Runtime: 1.2 seconds total.
11:48:27 far@FaR-KLVL-WXX9 catkin_ws → source devel/setup.bash
11:48:38 far@FaR-KLVL-WXX9 catkin_ws →

Github repository:

Sources:
Robot arm description package: https://github.com/RoboticsLab2023/arm_description.git
iiwa_stack robot example: https://github.com/IFL-CAMP/iiwa_stack/tree/master
Camera xacro: https://github.com/CentroEPiaggio/irobotcreate2ros/blob/master/model/camera.urdf.xacro
Other notes: *"Mastering ROS for Robotics Programming"*