Practice 8

Finalizing our URDF

8.1 Background

8.2 Adding physical properties to our robot for the simulations

In the previous practice, we learned how to build our basic robot arm model with URDF and Xacro. Since it is a basic model, it does not contain the properties necessary for a proper simulation. These properties include collision properties so that the simulation will have collision detection features. We also left out the physical properties of this model such as mass and inertia of the elements.

To solve these problems, we will step by step add these features. Inside the previous practice's package make a copy of the 'macros.urdf.xacro' and name it 'macros2.urfd.xacro', also do the same for the 'robot1.urdf.xacro'.

Now, starting with the collision, we will add this feature to the macro of our links. We can add different (even highly complex) shapes to the collision property, although most of the time it is a good practice to stay with simple shapes like boxes and cylinders since the calculation with them is much simpler.

We will need to modify both of our files. For the first link, since we don't have a macro for that part, we will do so in the 'robot2.urdf.xacro'. Inside the 'base_link' we add the following collision tag before the visual tag so the definition will look like this:

```
<link name ="base_link">
    <collision>
        <origin rpy="0 0 0" xyz="0 0 0.25" />
        <geometry>
            <box size="1 1 0.5" />
        </geometry>
    </collision>
    <visual>
        <origin rpy ="0 0 0 " xyz="0 0 0.25"/>
        <geometry>
            <box size="1 1 0.5"/>
        </geometry>
    </visual>
```

```
</link>
```

Inside 'macros2' we can also update the 'm_link' definition with the following:

```
<xacro:macro name="m_link" params="name origin_rpy origin_xyz radius length">
   <link name ="${name}">
     <collision>
         <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
         <geometry>
             <cylinder radius="${radius}" length="${length}"/>
         </geometry>
     </collision>
     <visual>
         <origin rpy ="${origin_rpy}" xyz="${origin_xyz}"/>
         <geometry>
             <cylinder radius="${radius}" length="${length}"/>
         </geometry>
     </visual>
  </link>
</xacro:macro>
```

We used the same values used in the visual tag since our collision shape should cover the complete shape of the links. Also don't forget to change the include part of the 'robot2' URDF to include the updated 'macros2' file.

```
<xacro:include filename="$(find robot_modelling)/urdf/macros2.urdf.xacro" />
```

It is time to check the soundness of the URDF.

```
$ ros2 launch robot_modelling display.launch.py model:=urdf/robot2.urdf.xacro
```

Now that we have our collision tags correctly set. We can continue with the physical properties. To do so we will add a new tag called 'inertia' to the robot's 'base_link' and the macros. For the first, we simply add this to the link, so our link now will look like the following:

```
    <link name ="base_link">
        <inertial>
            <mass value="1024" />
            <origin rpy="0 0 0" xyz="0 0 0.25" />
            <inertia ixx="170.667" ixy="0" ixz="0" iyy="170.667" iyz="0" izz="170.667" />
        </inertial>
        <collision>
            <origin rpy="0 0 0" xyz="0 0 0.25" />
            <geometry>
                <box size="1 1 0.5" />
            </geometry>
        </collision>
        <visual>
            <origin rpy ="0 0 0 " xyz="0 0 0.25"/>
            <geometry>
                <box size="1 1 0.5"/>
```

```
            </geometry>
        </visual>
    </link>
```

We add this to the macro definition as well:

```
<xacro:macro name="m_link" params="name origin_xyz origin_rpy radius length mass ixx ixy ixz iyy iyz izz">
    <link name="${name}">
      <inertial>
        <mass value="${mass}" />
        <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
        <inertia ixx="${ixx}" ixy="${ixy}" ixz="${ixz}" iyy="${iyy}" iyz="${iyz}" izz="${izz}" />
      </inertial>
      <collision>
        <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
        <geometry>
          <cylinder radius="${radius}" length="${length}"/>
        </geometry>
      </collision>
      <visual>
        <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
        <geometry>
          <cylinder radius="${radius}" length="${length}"/>
        </geometry>
      </visual>
    </link>
  </xacro:macro>
```

8.2.1 Simulating in a physical environment

To use the physical properties, we need a simulation tool with a built-in physical engine as well. Luckily, ROS2 has a compatible simulation tool called Gazebo. We will not go into details about Gazebo, since it will be the topic of the next practice, but for now, it is enough to say, that it is a simulation environment. To get full access to ROS2 we will need however some extra tools. Let's install them and then test the installation by opening a world file:

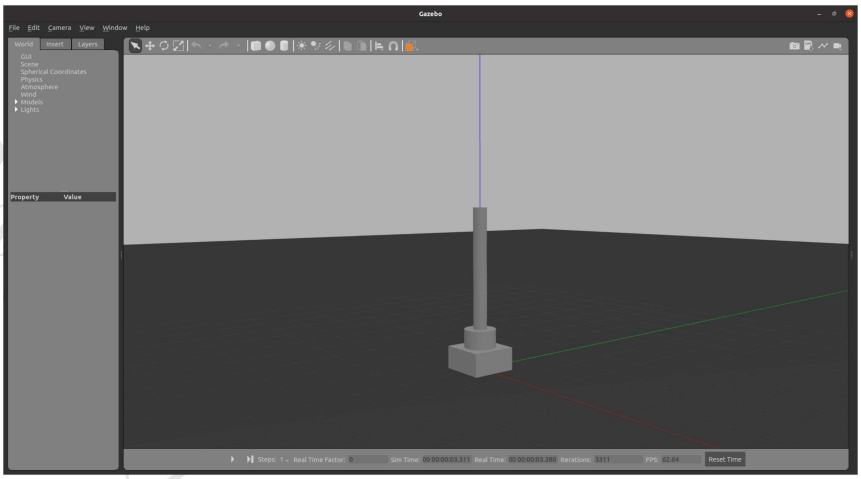
```
$ sudo apt install ros-foxy-gazebo-ros-pkgs -y
$ gazebo --verbose /opt/ros/foxy/share/gazebo_plugins/worlds/gazebo_ros_diff_drive_demo.world
```

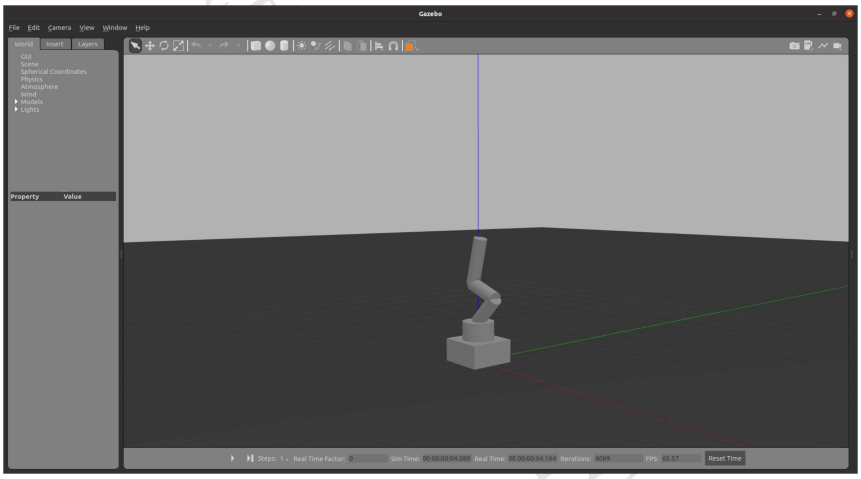
Now we will make a new launch file inside the 'launch' folder called 'gazebo.launch.py' with the following content:

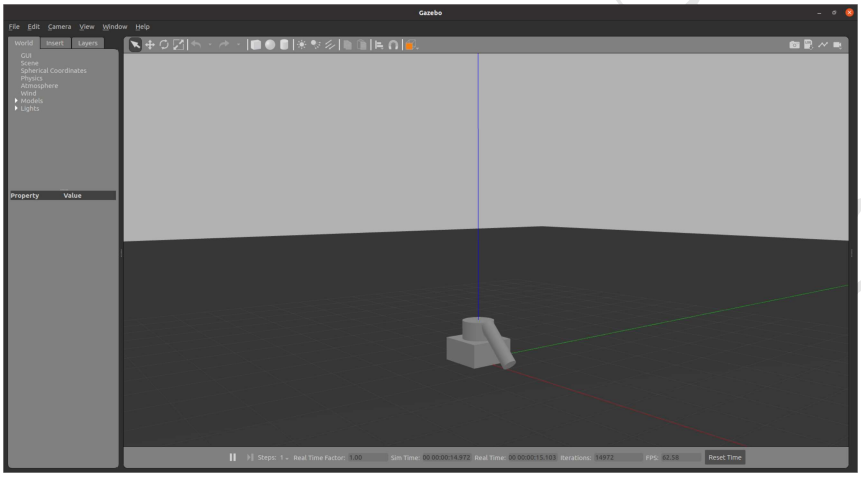
```
import launch
from launch.substitutions import Command, LaunchConfiguration
import launch_ros
import os

def generate_launch_description():
    pkg_share = launch_ros.substitutions.FindPackageShare(package='robot_modelling').find('robot_modelling')
```

```
    model_name = 'robot2.urdf.xacro'
    def_model_path = os.path.join(pkg_share, 'urdf', model_name)

    robot_state_publisher_node = launch_ros.actions.Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        parameters=[{'robot_description': Command(['xacro ',
LaunchConfiguration('model')])}]
    )

    joint_state_publisher_node = launch_ros.actions.Node(
        package='joint_state_publisher',
        executable='joint_state_publisher',
        name='joint_state_publisher'
    )

    spawn_robot = launch_ros.actions.Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        arguments=['-entity', 'RoboticArm', '-topic', 'robot_description
'],
        output='screen'
    )

    return launch.LaunchDescription([
        launch.actions.DeclareLaunchArgument(name='model', default_value
=def_model_path,
                                             description='Absolute paht
to the urdf file.'),
        launch.actions.ExecuteProcess(cmd=['gazebo', '--verbose', '-s',
'libgazebo_ros_init.so', '-s', 'libgazebo_ros_factory.so'],
                                      output='screen'),
        joint_state_publisher_node,
        robot_state_publisher_node,
        spawn_robot
    ])
```

In this file we define the default description file name ('robot2.urdf.xacro'), then we start a gazebo simulation and start the joint_state_publisher, the robot_state_publisher and we spawn the robot. Now what we should see is something similar process to the figures below:


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0 Sim Time: 00 00:00:03.311 Real Time: 00 00:00:03.380 Iterations: 3311 FPS: 62.64
Reset Time


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0 Sim Time: 00 00:00:04.089 Real Time: 00 00:00:04.164 Iterations: 4089 FPS: 62.57
Reset Time


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:14.972 Real Time: 00 00:00:15.103 Iterations: 14972 FPS: 62.58
Reset Time

What just happened? Did we do something wrong? It depends. We missing the actuation from the joints. But don't be afraid we will solve this problem as well, but before that, we need to talk about a warning regarding the KDL (Kinematics and Dynamics library). It states that KDL does not support base link with inertia. To correct this problem, we will add a dummy link with a fixed joint to our robot description ('robot2.urdf.xacro):

```
<!-- Dummy root for KDL-->
    <link name="dummy_root"/>

    <joint name="basejoint" type="fixed">
        <parent link="dummy_root"/>
        <child link="base_link"/>
        <origin xyz="0 0 0" rpy="0 0 0"/>
    </joint>

<!--END of Dummy root-->
```

8.3 Cooking tips

8.3.1 If Gazebo doesn't stops properly

You can use the following command to kill all the processes:

```
$ killall -9 gazebo gzserver gzclient
```