Practice 7

Modelling robots with URDF

7.1 Background

7.1.1 URDF

URDF[1] stands for the Unified Robot Description Format, which is usually an XML based DSML (domain specific modelling language). It contains the parameters of the described robot (geometric, kinematic, dynamic) and other metadata. This description language is designed so that it is also readable for human and machine. The basic elements of a URDF files are the links and joints:

- **Links**[2] are the structural parts of a robot which are modelled as rigid bodies (modelled as non-deformable solid bodies in mechanics).
- **Joints**[3] are the parts of the robot that connect links. (Joints are usually categorized into two categories which are the rotation- and the translation joints.)

To determine the constraint that a joint has, in URDF we can use 6 types of joints:

- Fixed - this joint connects the two links rigidly (all 6 DOF are disabled, it is like "glued" together)
- Revolut - this kind of joint lets the two link to rotate relatively to each other around an axis, but only between a range.
- Continuous - this joint is the same as Revolut (enables rotation around one axis), only it has no limits in the rotation.
- Prismatic - enables translational movement along one axis.
- Planar - enables two dimensional movement
- Floating - enables all 6 DOF (degree of freedom), 3 rotations and 3 translations along the three axes.

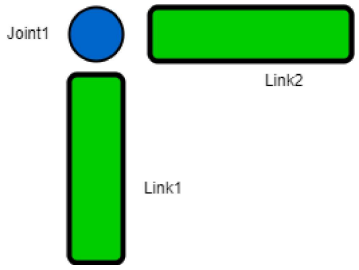


Figure 7.1: Two links connected by one joint.

One joint can only have one type at a time.
The main standard for the URDFs is the REP103[4] of which our main takeaway is:

- usage of right-handed coordinate system (X+ is the forward, Y+ is the left, Z+ is the upward)
- usage of SI units of measure (meter, radian etc.)

Usage of URDFs can be sometimes a little time-consuming because of the repeatedly written code. Later we will see how to use Xacro which is a macro language that can save us a lot of time.

7.1.2 RViz

The visualization will be in RViz which is a visualization package of ROS.

[1]URDF in ROS wiki
[2]Link in ROS Wiki
[3]Joint in ROS Wiki
[4]REP103

7.2 Building a robot model

We will build a robotic arm from scratch now. For this, we first need to install some related packages and make a package. Let's name it 'my_robotic_arm'.

```
$ sudo apt-get install ros-foxy-robot-state-publisher
$ sudo apt-get install ros-foxy-joint-state-publisher
$ sudo apt-get install ros-foxy-joint-state-publisher-gui
$ sudo apt-get install ros-foxy-xacro
$ sudo apt-get install ros-foxy-controller-manager
$ sudo apt-get install ros-foxy-joint-state-controller
$ sudo apt-get update
```

To install the 'urdf_launch' package we need to download it inside our 'src' directory of the workspace and then install it:

```
$ git clone https://github.com/ros/urdf_launch.git
$ col_mk urdf_launch
```

After the successful installations, we can create our package:

```
$ ros2 pkg create --build-type ament_cmake robot_modelling
```

And also make room in our package for our files, for which we will create an 'urdf', 'rviz', 'meshes' and 'launch' directories. We modify the 'CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)
project(robot_modelling)

find_package(ament_cmake REQUIRED)

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

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

The robot that we would like to create will look something like this:

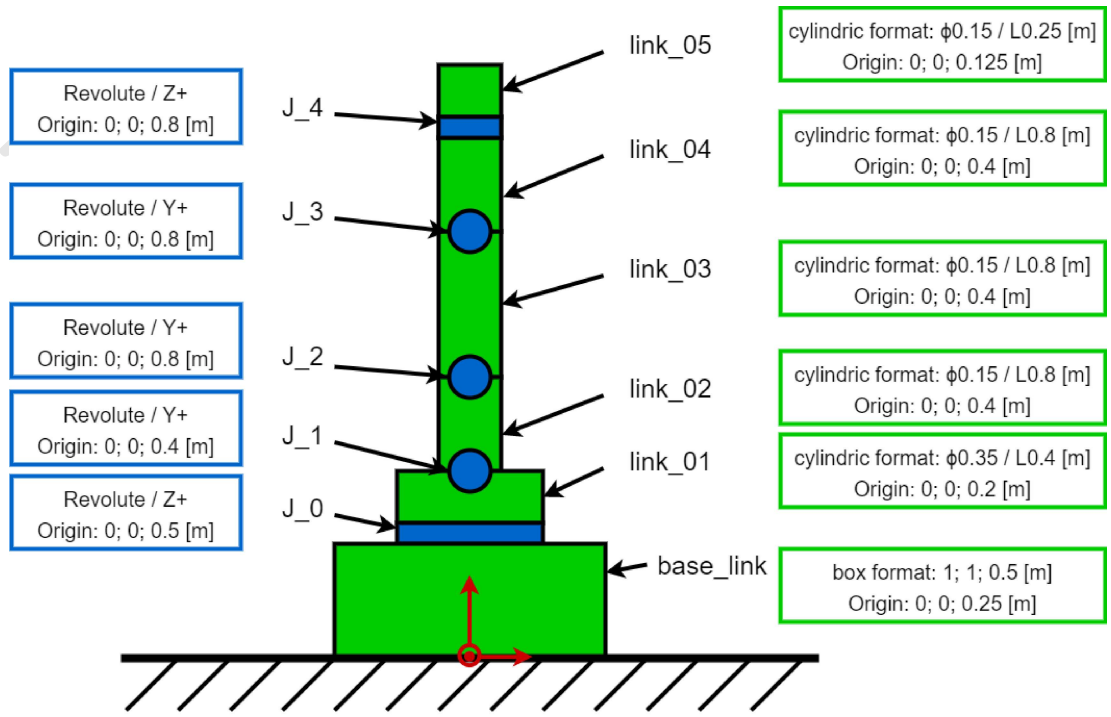


Figure 7.2: Our first robot

We will create our first model in the created 'urdf' directory, and give the name "robotarm1.urdf". The content of the file should look like the following:

```
<?xml version="1.0"?>
<robot name="robotarm_v1">

</robot>
```

Here we tell the system that we will use XML version 1.0 and we name our robot "robotarm_v1". Now we give the robot its first link, which name will be "base_link".

```
<?xml version="1.0"?>
<robot name="robotarm_v1">
    <link name ="base_link">
        <visual>
            <origin rpy ="0 0 0 " xyz="0 0 0.25"/>
            <geometry>
                <box size="1 1 0.5"/>
            </geometry>
        </visual>
    </link>
</robot>
```

Now we have something to visualize. For this, we will create a launch file that will open RViz (visualizer) and load the robot inside. Inside the 'launch' directory create a 'display.launch.py' file with the following content:

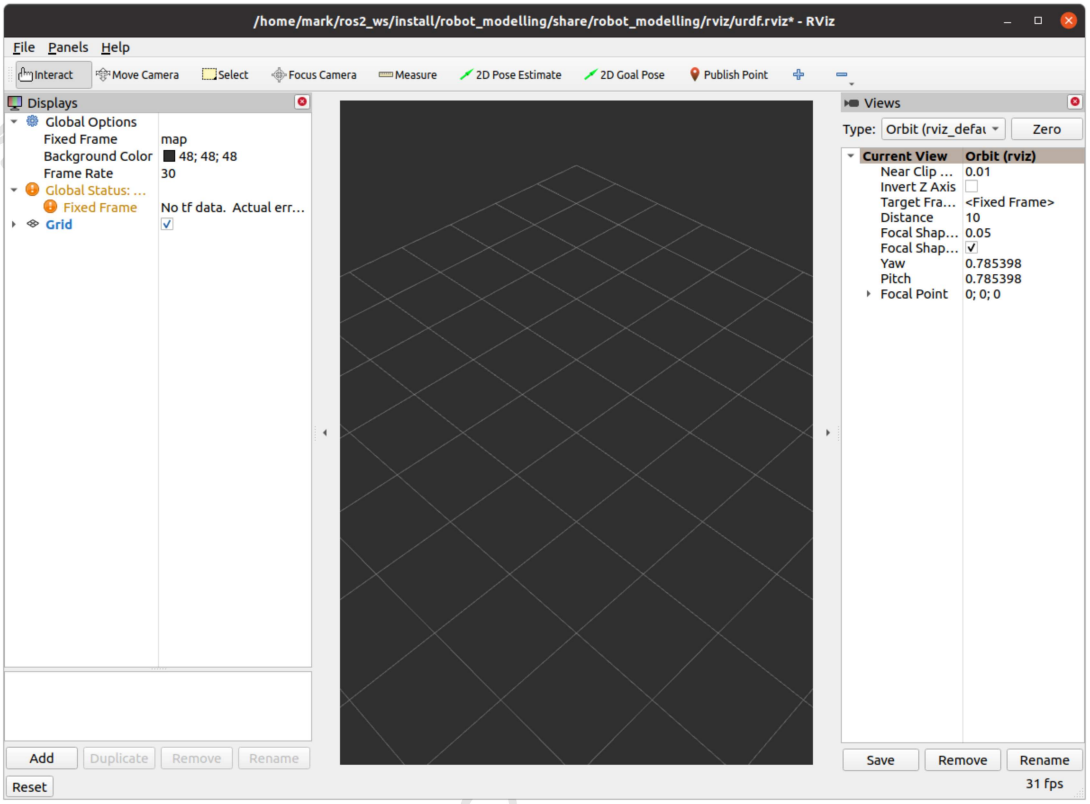
```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument,
    IncludeLaunchDescription
```

```
from launch.substitutions import LaunchConfiguration,
    PathJoinSubstitution
from launch_ros.substitutions import FindPackageShare


def generate_launch_description():
    ld = LaunchDescription()

    urdf_tutorial_path = FindPackageShare('robot_modelling')
    default_model_path = PathJoinSubstitution(['urdf', 'robotarm1.urdf'
    ])
    default_rviz_config_path = PathJoinSubstitution([urdf_tutorial_path,
    'rviz', 'urdf.rviz'])

    gui_arg = DeclareLaunchArgument(name='gui', default_value='true',
    choices=['true', 'false'],
                                    description='Flag to enable
    joint_state_publisher_gui')
    ld.add_action(gui_arg)
    rviz_arg = DeclareLaunchArgument(name='rvizconfig', default_value=
    default_rviz_config_path,
                                     description='Absolute path to rviz
    config file')
    ld.add_action(rviz_arg)

    ld.add_action(DeclareLaunchArgument(name='model', default_value=
    default_model_path,
                                        description='Path to robot urdf
    file relative to urdf_tutorial package'))

    ld.add_action(IncludeLaunchDescription(
        PathJoinSubstitution([FindPackageShare('urdf_launch'), 'launch',
    'display.launch.py']),
        launch_arguments={
            'urdf_package': 'robot_modelling',
            'urdf_package_path': LaunchConfiguration('model'),
            'rviz_config': LaunchConfiguration('rvizconfig'),
            'jsp_gui': LaunchConfiguration('gui')}.items()
    ))

    return ld
```

Now we can visualize our robot:

```
$ ros2 launch robot_modelling display.launch.py model:=urdf/robotarm1.
    urdf
```

This will open RVIz and we will see our robot. Or perhaps not, since we didn't set up RViz to do so.



To make our robot visible we need to do a few steps. First, on the 'Displays' panel, we change the Global Options's 'Fixed Frame' property from "map" to "base_link". Then, we click on "Add" at the bottom of this panel and select "TF" to add. Then again, we add "Robot Model". Finally, under Robot Model's 'description Topic' we select "/robot_description". We will see the following:

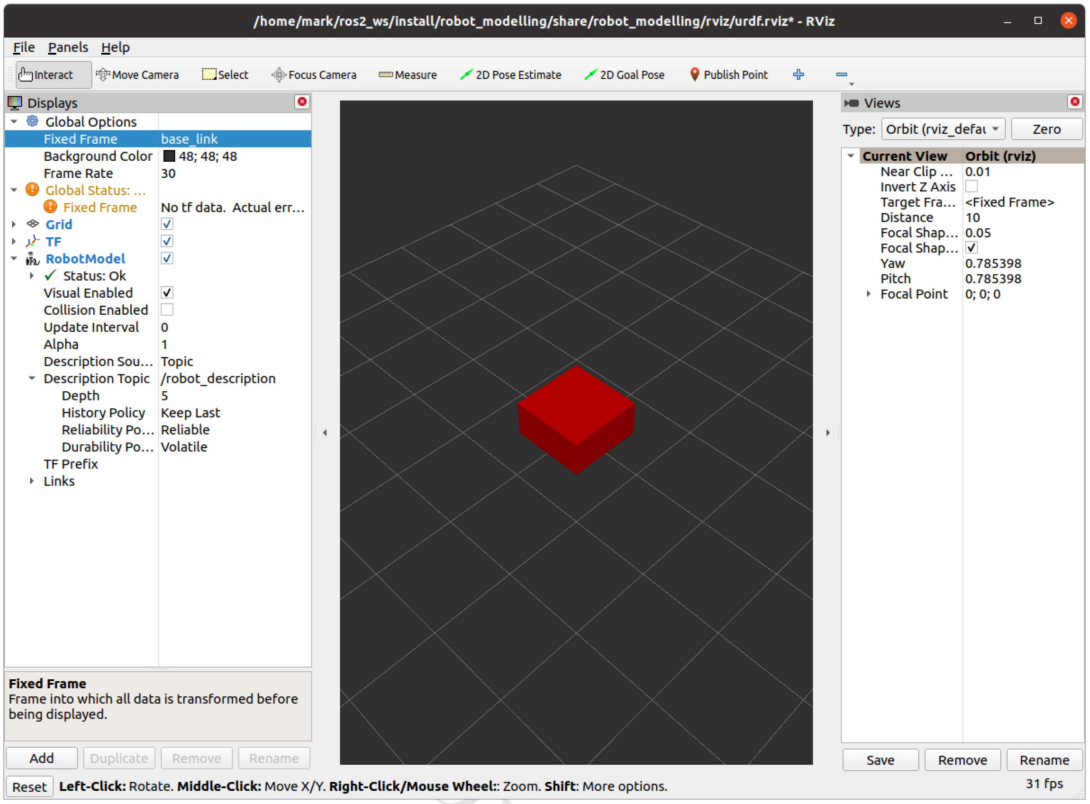
If we don't want to do these steps every time we can save this configuration by clicking on "File/Save Config As", then we give a name to the config. Let that be "urdf.rviz".

Now let's add a new link (after the previous one) to the robot description, which will be the "link_01":

```
    <link name = "link_01">
        <visual>
            <origin rpy = "0 0 0 " xyz="0 0 0.2"/>
            <geometry>
                <cylinder radius="0.35" length="0.4"/>
            </geometry>
        </visual>
    </link>
```

We have two links, meaning we can (and should) connect them. To do so we define (between the two links description, for easier reading) our first joint called "J_0", which will be a revolute type one:

```
    <joint name="J_0" type ="revolute">
        <axis xyz="0 0 1"/>
        <limit effort="1000.0" lower="-3.14" upper="3.14" velocity = "
    0.5"/>
```




```
        <origin rpy="0 0 0" xyz="0 0 0.5"/>
        <parent link="base_link"/>
        <child link="link_01"/>
    </joint>
```

At this point, if we want to check how the robot looks (which is a good practice), then we need to build our package, and with the previously used command, we can visualize our robot.

From now we also can move our joints with the 'Joint State Publisher'.

We can continue (now it is Your turn, based on Fig. 7.2) to add the next joints and links. (It is a good practice to build and visualize Your robot after certain changes during the work.) You can find the complete URDF in the appended: A.4.1. We can see, that this process is highly repetitive and easy to make mistakes and typos. At the end to describe this robot we needed 106 lines of code. At the end, You should see something like the following:

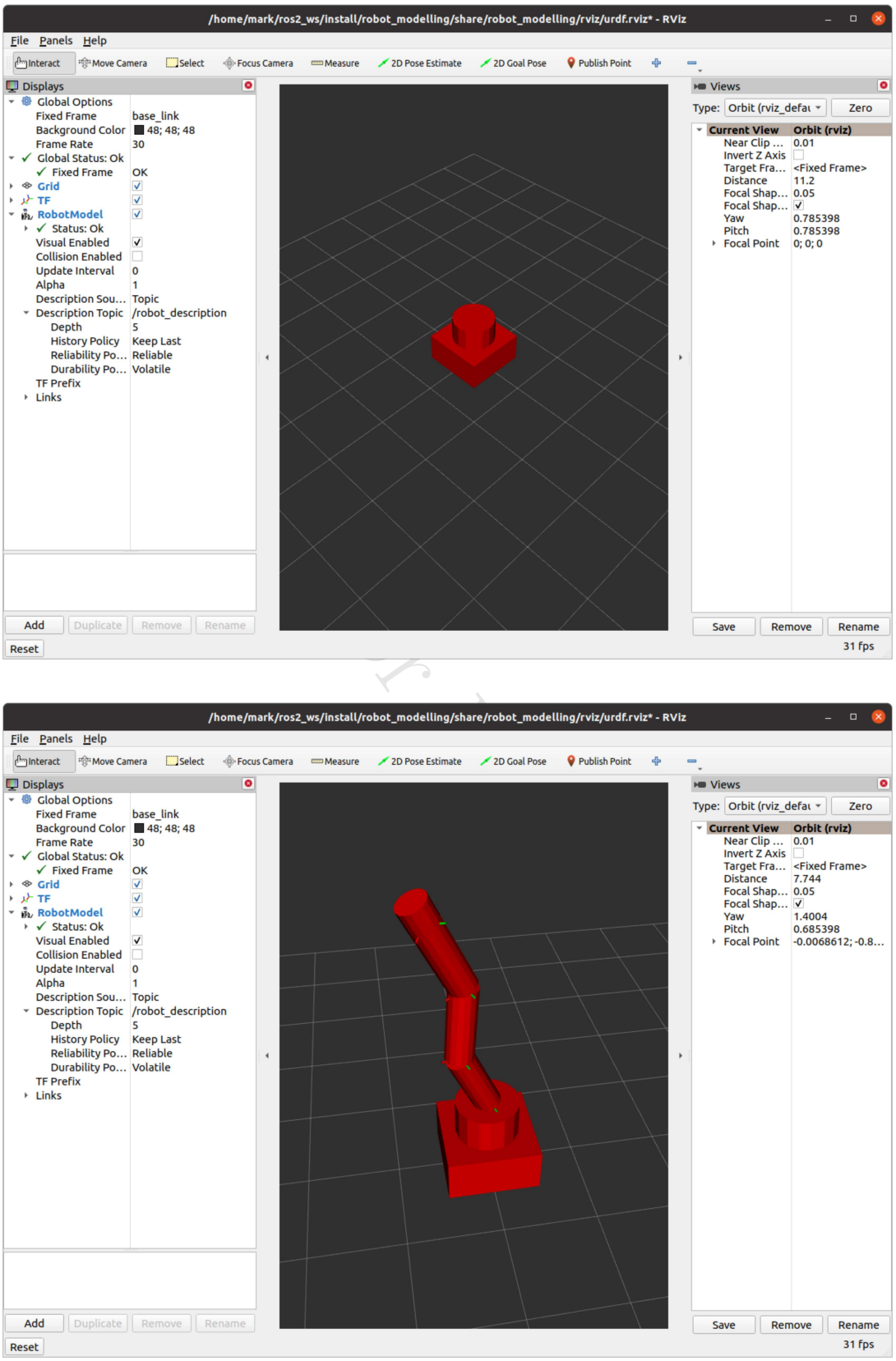


Figure 7.3: Half of our robot

7.2.1 Xacro

Previously we saw, that to model a simple robot takes 106 lines of XML code. To make our lives a little easier we can use Xacro, which is a macro-based version of the URDFs. Here we can create macros for similar elements, create constants, and use complex expressions to define values. Let's create two of them in our "urdf" folder. One will be called 'macros.urdf.xacro' and the second will be 'robot1.urdf.xacro'. Inside the first we will define a macro for the joints and later one for the links. In the latter, we will build up our robot again. Let's create the macro for the joints:

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


</robot>
```

We created the robot template, now we tell the system where to find the XML namespace. Inside the robot tags we will define the macro for the joints:

```
  <xacro:macro name="m_joint" params="name type axis_xyz origin_rpy
   origin_xyz parent child limit_e limit_l limit_u limit_v">
    <joint name="${name}" type="${type}">
      <axis xyz="${axis_xyz}" />
      <limit effort="${limit_e}" lower="${limit_l}" upper="${limit_u}"
   velocity="${limit_v}" />
      <origin rpy="${origin_rpy}" xyz="${origin_xyz}" />
      <parent link="${parent}" />
      <child link="${child}" />
    </joint>
  </xacro:macro>
```

The first line defines the name of the macro type, then all the variables used inside the macro. After this, we see that the macro looks exactly the same way as our original URDF joint definitions, just it is now parameterized. Now let's use this joint macro inside our "robot1.urdf.xacro"

```
<?xml version="1.0" ?>
<robot name="robot1" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Start of Includes-->
  <xacro:include filename="$(find robot_modelling)/urdf/macros.urdf.
   xacro" />
  <!-- End of Includes -->

</robot>
```

We named our robot to "robot1" similarly to the previous robot. Now we also added an include line to use our macros in the separated file. (Although, they can be defined in the same file as well.) Now let's add the first two links to the robot and the joint between them.

```
 <!-- Start of robot description -->
   <link name ="base_link">
        <visual>
            <origin rpy ="0 0 0 " xyz="0 0 0.25"/>
```

```
            <geometry>
                <box size="1 1 0.5"/>
            </geometry>
        </visual>
    </link>

  <xacro:m_joint name="J_0" type="revolute"
                 axis_xyz="0 0 1"
                 origin_rpy="0 0 0" origin_xyz="0 0 0.5"
                 parent="base_link" child="link_01"
                 limit_e="1000" limit_l="-3.14" limit_u="3.14" limit_v="
   0.5" />


    <link name = "link_01">
        <visual>
            <origin rpy = "0 0 0 " xyz="0 0 0.2"/>
            <geometry>
                <cylinder radius="0.35" length="0.4"/>
            </geometry>
        </visual>
    </link>
<!-- End of robot description -->
```

If we build this project and visualize it with the launch file, then we will see the same robot as on Fig. 7.3.

```
$ ros2 launch robot_modelling display.launch.py model:=urdf/robot1.urdf.
   xacro
```

Now it is Your turn, to continue with. Define the macro for the links in the correspondent file and add all the joints and the links from 'link_01' to it. At the end again You should see the same robot. Now, even in this simple case we saved around 15 lines of code and our code is more readable. Congratulations, You just built Your first robot model.

The original ROS2 Foxy tutorial series is similar to this one, only there You can build an R2D2-type robot.[1]

7.3 Cookbook

7.3.1 Constants in Xacro

You can define properties in Xacro which are acting as constants in the following manner:

```
<xacro:property name="item" value="link" />
.
.
.
<link name="${item}_${ID}" />
```

If we add this line to our macro file, we will get another way to define the name of our links. But we can have much better as well.

[1] ROS2 tutorials on URDF

7.3.2 Math in Xacro

If You were vigilant enough, then You saw that in our robotic case most of the time the origin is half of the cylindric typed link lengths. We can use this observation and add a little math to our definition, by simply changing the origin property's line.

```
1 <origin rpy ="${origin_rpy}" xyz="0 0 ${length/2}"/>
```