3. Xacro

Xacro is an Xml macro that introduce the use of macros in an urdf file. It allow the use of variables, math and macros. And let us divide the robot model in different files.

3.1. Xacro syntax

3.1.1. Property

Instead of defining constant values, we can use variables, called property in xacro.

```
 <xacro:property name="width" value="0.2" />
 <xacro:property name="bodylen" value="0.6" />

 <link name="base_link">
     <visual>
         <geometry>
             <cylinder radius="${width}" length="${bodylen}"/>
         </geometry>
     </visual>
 </link>

In the previous snippet we create two properties with default values: ::

 <xacro:property name="width" value="0.2" />
 <xacro:property name="bodylen" value="0.6" />
```

In the `geometry` tag, instead of using constant values, we assign the properties just defined to the radius and the length. Note the use of `${}` .

3.1.2. Math

```
<cylinder radius="${wheeldiam/2}" length="0.1"/>
<origin xyz="${reflect*(width+.02)} 0 0.25" />
```

3.1.3. Macro

Marco are piece of code that can be used as functions.

```
<xacro:macro name="default_origin">
  <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:macro>
```

When need the macro can be called by name:

```
<xacro:default_origin />
```

When called, the macro content will placed where is called.

Macros can accept also input parameters:

```
<xacro:macro name="default_inertial" params="mass">
    <inertial>
            <mass value="${mass}" />
            <inertia ixx="1.0" ixy="0.0" ixz="0.0"
                 iyy="1.0" iyz="0.0"
                 izz="1.0" />
    </inertial>
</xacro:macro>
```

Can be called in this way:

```
<xacro:default_inertial mass="10"/>
```

Macros input parameters can be also blocks or macros. When we need to pass a block as a parameter to another macro, we precede the parameter name with `*`.

```
<xacro:macro name="blue_shape" params="name *shape">
    <link name="${name}">
        <visual>
            <geometry>
                <xacro:insert_block name="shape" />
            </geometry>
            <material name="blue"/>
        </visual>
        <collision>
            <geometry>
                <xacro:insert_block name="shape" />
            </geometry>
        </collision>
    </link>
</xacro:macro>
```

In this snippet, we have 2 parameters. The first parameter `name` is a string, the second one `shape` is a block.

A block is defined as follow:

```
<xacro:blue_shape name="base_link">
  <cylinder radius=".42" length=".01" />
</xacro:blue_shape>
```

The macro can be called ?????????????????:

```
<xacro:blue_shape mass="10" shape="blue_shape"/>
```

3.1.4. Example

Here is defined a macro called leg. As the model have two legs, in order to avoid duplicated code, we define a macro with two paramters, `prefix` that is part of the link and joint names. And `reflect` to define the position of the leg respect to the axis.

```
<xacro:macro name="leg" params="prefix reflect">

    <link name="${prefix}_leg">
        <visual>
            <geometry>
                <box size="${leglen} 0.1 0.2"/>
            </geometry>
            <origin xyz="0 0 -${leglen/2}" rpy="0 ${pi/2} 0"/>
            <material name="white"/>
        </visual>

        <!-- call to default_inertial macro -->
        <xacro:default_inertial mass="10"/>
    </link>

    <joint name="base_to_${prefix}_leg" type="fixed">
        <parent link="base_link"/>
        <child link="${prefix}_leg"/>
        <origin xyz="0 ${reflect*(width+.02)} 0.25" />
    </joint>

</xacro:macro>

<!-- call to legl macro -->
<xacro:leg prefix="right" reflect="1" />
<xacro:leg prefix="left" reflect="-1" />
```

3.2. Xacro to urdf

Once an xacro file is defined it must be converted to urdf file in order to be used.

```
rosrun xacro xacro scara.xacro --inorder > scara_generated.urdf
```

The command can be integrated inside a launch file.

```
<param name="robot_description" command="$(find xacro)/xacro --inorder $(find
epson_g3_description)/urdf/scara.xacro" />
```

Or better it can be parameterized:

```
<param name="robot_description" command="$(find xacro)/xacro --inorder $(arg
model)" />
```

We will defined a new launch file:

```
<launch>
<arg name="pkg" default="epson_g3_description"/>

  <arg name="model" default="$(find epson_g3_description)/urdf/scara.xacro"/>
  <arg name="rvizconfig" default="$(find epson_g3_description)/rviz/urdf.rviz" />

  <arg name="gui" default="true" />

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(arg
model)" />
  <param name="use_gui" value="$(arg gui)"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)"
required="true" />
</launch>
```

Download display3.launch

3.3. Scara robot with xacro

Download scara.xacro

3.4. Include files

When the robot model is complex, it is convenient to divide the definition in different files. Suppose we have a modile robot. We can create at least 2 files. One file to define the wheel and the other one the robot, where the wheel file will be included.

Let' create a file called `wheel.xacro` , where we define different marcos and properties.

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

  <!-- Wheels -->
  <xacro:property name="wheel_radius" value="0.04" />
  <xacro:property name="wheel_height" value="0.02" />
  <xacro:property name="wheel_mass" value="2.5" /> <!-- in kg-->

  <xacro:macro name="cylinder_inertia" params="m r h">
    <inertia  ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
              iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
              izz="${m*r*r/2}" />
  </xacro:macro>

   <xacro:macro name="wheel" params="fb lr parent translateX translateY flipY">
    <!-- other stuff -->
  </xacro:macro>

</robot>
```

This file can be included in other xacro file:

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

And the macro `wheel` defined in the file `wheel.xarco` can be called:

```
<!-- Wheel Definitions -->
<wheel fb="front" lr="right" parent="base_link" translateX="0" translateY="0.5"
flipY="1"/>
<wheel fb="front" lr="left" parent="base_link" translateX="0" translateY="-0.5"
flipY="1"/>
```