

Visual URDF Robot Models

ECE 495/595 Lecture Slides

Winter 2017

Instructor: Micho Radovnikov

Summary and Quick Links

These slides contain the following concepts:

- ▷ What is URDF? (Slide [3](#))
- ▷ Basic components of a URDF file (Slide [4](#))
- ▷ Xacro macros (Slide [11](#))
- ▷ CAD mesh files (Slide [14](#))
- ▷ Loading URDF models (Slide [16](#))
- ▷ TF frames of a robot model (Slide [17](#))

What is URDF?

- ▷ URDF is a standard XML format for describing robot models.
- ▷ User defines the shape of the various links of a robot, the joints that connect them, and the behavior of the joints (revolute, prismatic, fixed).
- ▷ Collision and inertial properties of links can also be specified for simulation in Gazebo, but that will be discussed in another set of slides.
- ▷ It is recommended to go through the official tutorials on the ROS wiki: <http://wiki.ros.org/urdf/Tutorials>

Components of a URDF Model

- ▷ The two fundamental elements of a URDF model are links and joints.
- ▷ Any number of links and joints are specified in separate XML tags.
- ▷ Links specify the physical properties of each separate piece of the robot. Their names match the TF frames that will eventually be published.
- ▷ Joints specify the kinematic relationship between links, including the type and limits of the joint.

Components of a URDF Model

- ▷ Example XML tags for two simple links:

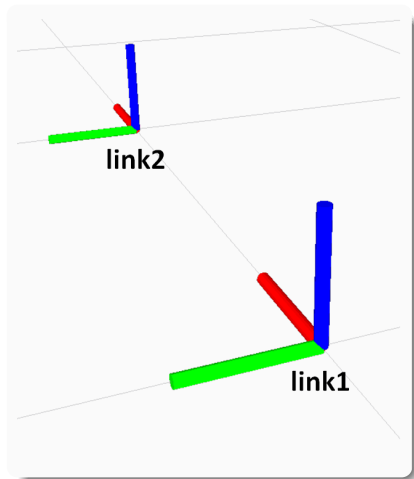
```
<link name="link1" >
  <visual>
    <geometry>
      <!-- Shape goes here -->
    </geometry>
    <material name="color_name" >
      <!-- Color goes here -->
    </material>
  </visual>
</link>

<link name="link2" >
  <visual>
    <geometry>
      <!-- Shape goes here -->
    </geometry>
    <material name="color_name" >
      <!-- Color goes here -->
    </material>
  </visual>
</link>
```

Components of a URDF Model

- ▷ Describe a fixed joint between these two links:

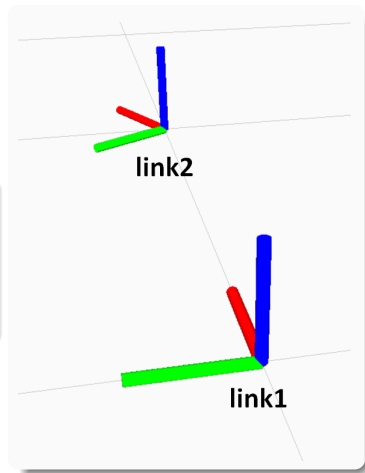
```
<joint name="joint1" type="fixed" >  
  <parent link="link1" />  
  <child link="link2" />  
  <origin xyz="1 0 0" rpy="0 0 0" />  
</joint>
```



Components of a URDF Model

- ▷ Describe a revolute joint between these two links:

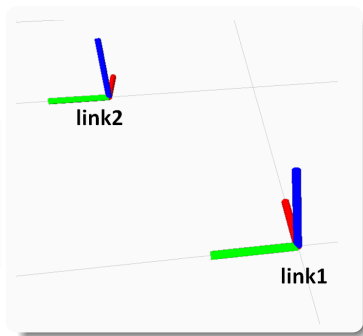
```
<joint name="joint1" type="revolute" >  
  <parent link="link1" />  
  <child link="link2" />  
  <origin xyz="1 0 0" rpy="0 0 0" />  
  <axis xyz="0 0 1" />  
  <limit effort="1000.0" velocity="3.0" />  
</joint>
```



Components of a URDF Model

- ▷ Describe a prismatic joint between these two links:

```
<joint name="joint1" type="prismatic" >  
  <parent link="link1" />  
  <child link="link2" />  
  <origin xyz="1 0 0" rpy="0 0 0" />  
  <axis xyz="0 1 0" />  
  <limit effort="1000.0" velocity="3.0" />  
</joint>
```



Components of a URDF Model

- ▷ Here is an example of a complete URDF model:

```
<robot name="example_robot" >

  <link name="link1" >
    <visual>
      <geometry>
        <box size="0.1 0.1 0.5" />
      </geometry>
      <material name="green" >
        <color rgba="0 1 0 1" />
      </material>
    </visual>
  </link>

  <link name="link2" >
    <visual>
      <geometry>
        <cylinder radius="0.2" length="1.0" />
      </geometry>
      <material name="red" >
        <color rgba="1 0 0 1" />
      </material>
    </visual>
  </link>
```

Components of a URDF Model

▷ Continued...

```
<joint name="joint1" type="fixed" >  
  <parent link="link1" />  
  <child link="link2" />  
  <origin xyz="1 0 0" rpy="0 0 0" />  
</joint>  
  
</robot>
```

▷ Notice how everything is inside of a <robot> </robot> tag.

Xacro Macros

- ▷ Even simple robots can require quite complicated URDF files.
- ▷ xacro macros help by allowing reusable modules and parameters, as well as mathematical expressions.
- ▷ In order to use the xacro capabilities, put the following in the robot tag:

```
xmlns:xacro="http://www.ros.org/wiki/xacro"
```

- ▷ Details about xacro are found here:
<http://wiki.ros.org/xacro>

Xacro Macros

- ▷ *xacro:property* – Defines a particular value:

```
<xacro:property name="a_number" value="6" />
```

- ▷ *xacro:macro* – Defines a reusable module:

```
<xacro:macro name="a_macro" params="a_parameter" >
  <link name="link" >
    <visual>
      <box size="${a_parameter} ${a_number} 0" />
    </visual>
  </link>
</xacro:macro>
```

- ▷ Math expressions:

```
<xacro:property name="half_number" value="${a_number/2}" />
```

Xacro Macros

Example use of a macro

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

  <xacro:macro name="link_macro" params="name y_scale" >
    <link name="${name}" >
      <visual>
        <geometry>
          <box size="1 ${y_scale} 1" />
        </geometry>
        <material name="blue" >
          <color rgba="0 0 1 1" />
        </material>
      </visual>
    </link>
  </xacro:macro>

  <xacro:link_macro name="link1" y_scale="1.0" />
  <xacro:link_macro name="link2" y_scale="0.5" />

  <joint name="joint1" type="continuous" >
    <parent link="link1" />
    <child link="link2" />
    <axis xyz="0 0 1" />
    <origin xyz="0 0 1.5" rpy="0 0 0" />
  </joint>
</robot>
```

CAD Mesh Files

- ▷ In a <geometry> tag, a mesh file exported from a CAD program can be specified in a <mesh> tag.
- ▷ Only .stl and .dae meshes are supported. Many CAD programs are capable of exporting in either of these formats, however.
- ▷ The imported mesh file is specified in the filename property of the <mesh> tag.
- ▷ A relative path to a ROS package is specified using package://

CAD Mesh Files

- ▷ Below is an example of a URDF link that uses a mesh file for visual geometry.

```
<link name="base_link" >
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
<mesh filename="package://mantis_model/meshes/mantis_base.stl" scale="1 1 1" />
    </geometry>
    <material name="gray" />
  </visual>
</link>
```

- ▷ In this case, particular mesh file *mantis_base.stl* is in the *meshes* folder in the ROS package *mantis_model*
- ▷ The *scale* property is used to independently scale the x, y and z axes of the imported mesh.

Loading a URDF Model

- ▷ A URDF model is loaded as a ROS parameter. Anything that needs the model simply looks up this parameter.
- ▷ This parameter is usually called /robot_description, but is sometimes loaded into a namespace.
- ▷ The model is loaded as a parameter in a launch file like this:

```
<param name="robot_description" command="$(find xacro)/xacro.py  
      '$(arg pkg_name)/urdf/robot.urdf.xacro' />
```

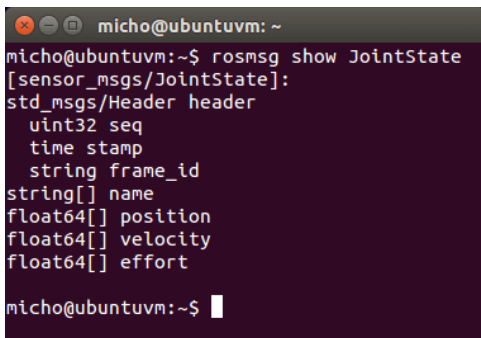
- ▷ xacro.py is a program that parses the URDF file.

TF Frame Management

- ▷ A node called *robot_state_publisher* publishes all the TF frames necessary for each link in the URDF file.
- ▷ For fixed joints, robot state publisher just publishes the offset and orientation as a static transform.
- ▷ For movable joints, the state publisher subscribes to a *sensor_msgs::JointState* message to receive what the current joint values are.
- ▷ This *JointState* message has to come from another node that is keeping track of the joint positions.

TF Frame Management

- ▷ *sensor_msgs::JointState* message structure:

A terminal window with a dark background and light text. The title bar shows a window icon, a minus icon, and a maximize icon, followed by the text 'micho@ubuntuvvm: ~'. The terminal content shows the command 'rosmmsg show JointState' being executed, which outputs the structure of the 'sensor_msgs/JointState' message. The output lists fields: 'std_msgs/Header header' (with sub-fields 'uint32 seq', 'time stamp', and 'string frame_id'), 'string[] name', 'float64[] position', 'float64[] velocity', and 'float64[] effort'. The prompt 'micho@ubuntuvvm:~\$' is shown at the bottom with a cursor.

```
micho@ubuntuvvm: ~  
micho@ubuntuvvm:~$ rosmmsg show JointState  
[sensor_msgs/JointState]:  
std_msgs/Header header  
  uint32 seq  
  time stamp  
  string frame_id  
string[] name  
float64[] position  
float64[] velocity  
float64[] effort  
  
micho@ubuntuvvm:~$
```

- ▷ The *name* vector corresponds to the joint names in the URDF file.
- ▷ The *position* vector controls the joint values.

TF Frame Management

- ▷ The velocity vector controls the velocity of the joints.
- ▷ The effort controls the force applied by the joint.
- ▷ A JointState message can be initialized like this:

```
sensor_msgs::JointState joints;  
joints.name.resize(3);  
joints.position.resize(3, 0);  
joints.velocity.resize(3, 0);  
joints.effort.resize(3, 0);  
  
joints.header.frame_id = "base_footprint";  
joints.name[0] = "joint1";  
joints.name[1] = "joint2";  
joints.name[2] = "joint3";
```

TF Frame Management

- ▷ To update the position of joint1 to spin at a constant rate, for example, a timer could be used.
- ▷ If a timer is set up to run at 50 Hz, the callback would look something like this:

```
void timerCallback(const ros::TimerEvent& event)
{
    double constant_speed = 2.0; // rad/s
    joints.header.stamp = event.current_real;
    joints.velocity[0] = constant_speed;
    joints.position[0] += 0.02 * constant_speed;
    pub_joint_states.publish(joints);
}
```