Creating and Visualizing Robot Models

Wyatt Newman

July, 2015

A robot model is valuable both for simulations and for visualization. Simulation models will be deferred until after introduction of visualization. Given a sufficiently detailed robot description, the pose of the robot in space can be constructed and illustrated based on sensor values of the robot's joints. Such visualization is identical whether the robot's joint-sensor values originate from a simulation of the robot or from communication with the actual robot.

The Unified Robot Description Format (URDF): A robot model can be specified in ROS using the "Unified Robot Description Format" (see http://wiki.ros.org/urdf). For visualization purposes, one does not need to specify any dynamic properties. It is only necessary to describe the shapes (surfaces) of links, and how the links interconnect via joints.

A minimal example URDF file is given below (and also appears in the package "minimal_robot_description" as file "minimal_robot_visual.urdf").

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

<!-- Used for fixing robot to Gazebo 'base_link' -->
  <link name="world"/>

  <joint name="glue_robot_to_world" type="fixed">
    <parent link="world"/>
    <child link="link1"/>
  </joint>

 <!-- Base Link -->
  <link name="link1">
    <visual>
     <origin xyz="0 0 0.5" rpy="0 0 0"/>
      <geometry>
       <box size="0.2 0.2 1"/>
      </geometry>
    </visual>
  </link>

<!-- Moveable Link -->
  <link name="link2">
    <visual>
     <origin xyz="0 0 0.5" rpy="0 0 0"/>
      <geometry>
        <cylinder length="1.0" radius="0.1"/>
      </geometry>
    </visual>
  </link>
```

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

In the above, a robot called “one_DOF_robot” is defined consisting of two links. Each element (link or and joint) of the robot is detailed in XML syntax.

Link1 is “glued” to the “world” frame via a fixed (non-movable) joint named “glue_robot_to_world.” The appearance of link1 is defined to be a rectangular prism (box) of dimensions (Lx,Ly,Lz) = (0.2, 0.2, 1.0). (In ROS, all dimensions are MKS by default). “Box” and “cylinder” are geometric entities than can be described in-line. More commonly an entire CAD file is referenced for the appearance of each robot element. A coordinate frame is associated with this link, and this coordinate frame is coincident with the “world” reference frame (as imposed by the “glue_robot_to_world” fixed joint). In defining a rectangular prism as the appearance of link1, the “box” entity is defined about an origin at the center of the box. We place the origin of this visualization entity to be elevated 0.5m above the link's reference frame. With this displacement (<origin xyz="0 0 0.5" rpy="0 0 0"/>), the bottom of the defined box aligns with the world frame. With roll, pitch and yaw angles (rpy) all 0, the z-axis of the box is parallel to the z axis of the world frame (as are the respective x and y axes).

A second link, “link2”, is defined to be a cylinder of length 1.0m and radius 0.1m. The origin of this visualization model is also at the center of the shape. We define the cylinder's frame relative to the link2 reference frame as <origin xyz="0 0 0.5" rpy="0 0 0"/>, which places the bottom of the cylindrical shape aligned with the link2 frame.

We must also define how link2's frame is related to link1's frame. This relationship is variable, since these links are connected via a revolute joint. Link1 and link2 are defined to be coupled via a joint called “joint1.” This is a revolute joint with a “parent” of link1 and a “child” of link2. The joint has an origin at (0,0,1) with respect to the link1 frame, which is at the “top” of the rectangular prism (1m above the world plane). The revolute joint has an axis of rotation, which is defined in terms of a vector through the joint origin. In the URDF file, this is defined as <axis xyz="0 1 0"/>, which is coincident with the y-axis of link2.

Importing a robot model into Rviz: To communicate a robot model to Rviz, the model is loaded into a “parameter server” (see http://wiki.ros.org/ParameterServer). The parameter server is yet another means of ROS communications. However, this is meant primarily for initializations. Nodes can get and set values from the parameter server, but nodes are not informed of such changes (but must actively examine these values, typically on start-up).

An entire robot model can be loaded into the parameter server, typically via a launch file. The following launch file, “minimal_robot_visual.launch” resides within package “minimal_robot_description.”

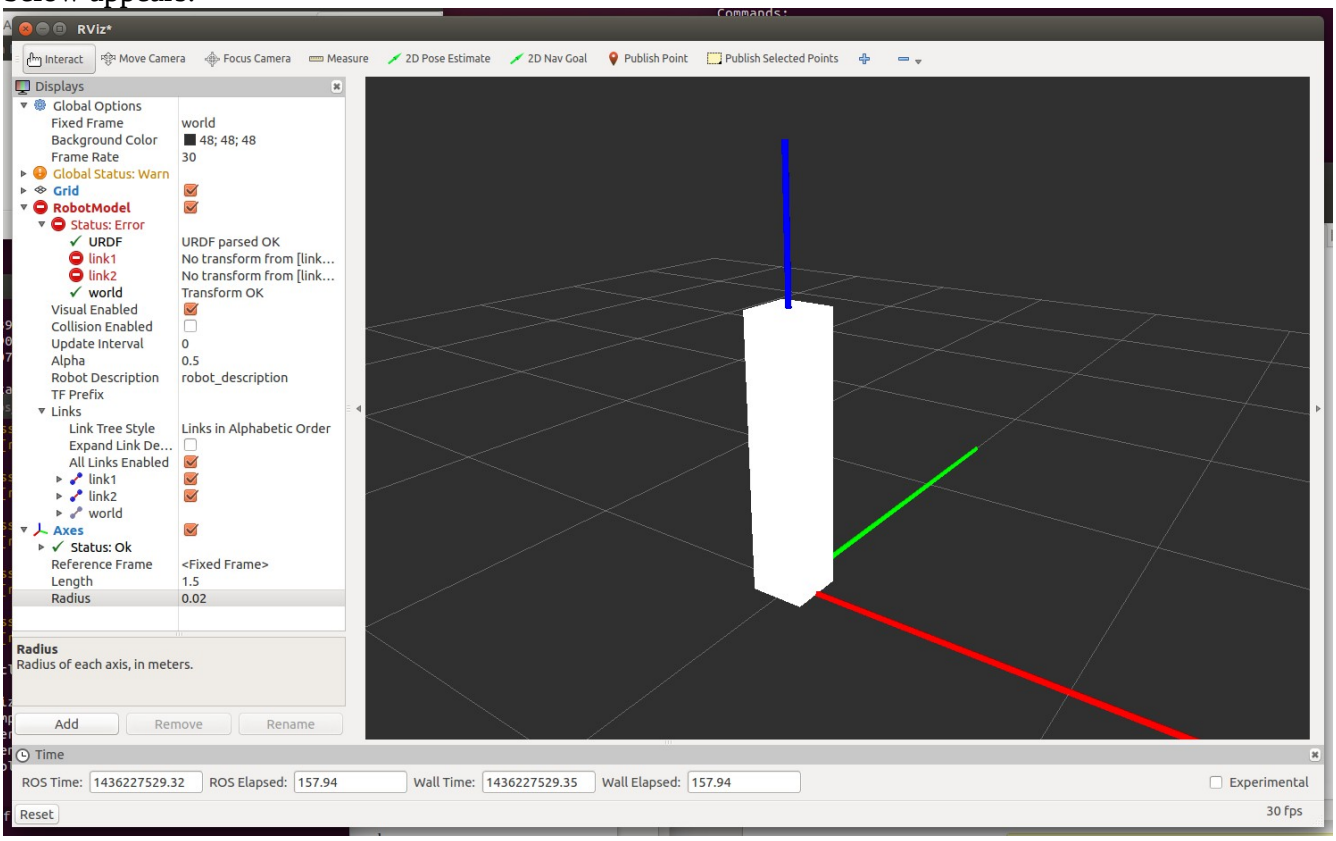
```
    <launch>
    <param name="robot_description"
    textfile="$(find minimal_robot_description)/minimal_robot_description.urdf"/>
    </launch>
```

Executing this launch file with:

roslaunch minimal_robot_description minimal_robot_visual.launch

loads the robot description into the parameter server. Nodes may access details of the robot model by querying the parameter server using the parameter named “robot_description.”

The robot model may be loaded into rviz by adding a display item “RobotModel” and setting (editing) the value of “Robot Description” to the parameter name “robot_description”. At this point, the view below appears:



As shown, the robot model is not successfully displayed. The Display panel indicates errors in the RobotModel item, complaining of “no transform...”. The problem is that link2's pose in space depends on the value of joint1's joint angle, and this has not been specified. This can be corrected with the help of additional nodes.

Joint_state_publisher and robot_state_publisher:

For Rviz to correctly place link2 in a 3D view, Rviz must know the full pose of link2. This is described as a “transform”--a relationship between two coordinate frames. The full 6-DOF pose of link2 can be deduced from knowledge of the robot model plus specification of the angle of “joint1.” Ordinarily, the joint angle would be obtained from a publication from a robot simulator or from an actual robot via the topic “joint_states.” At this point, we have neither a physical robot nor a simulated robot, but we can create a publisher to the topic “joint_states” to temporarily fill this roll. The value of joint angle to be published can be input via a slider using the following command:

rosrun joint_state_publisher joint_state_publisher _use_gui:=true

The above command shows an additional feature of "rosrun." Besides the package name and node name, one can specify optional parameters with "_" preceding the parameter name, and ":=" used to assign the value on the right-hand side to the parameter on the left-hand side.

Running the above command results in the following GUI appearing.

At this point, running:
rostopic list
will show that there is a new topic, "joint_states." Running:
rostopic info joint_states
shows that this topic carries messages of type "sensor_msgs/JointState." Running:
rosmsg show sensor_msgs/JointState
displays the format of this message type as:

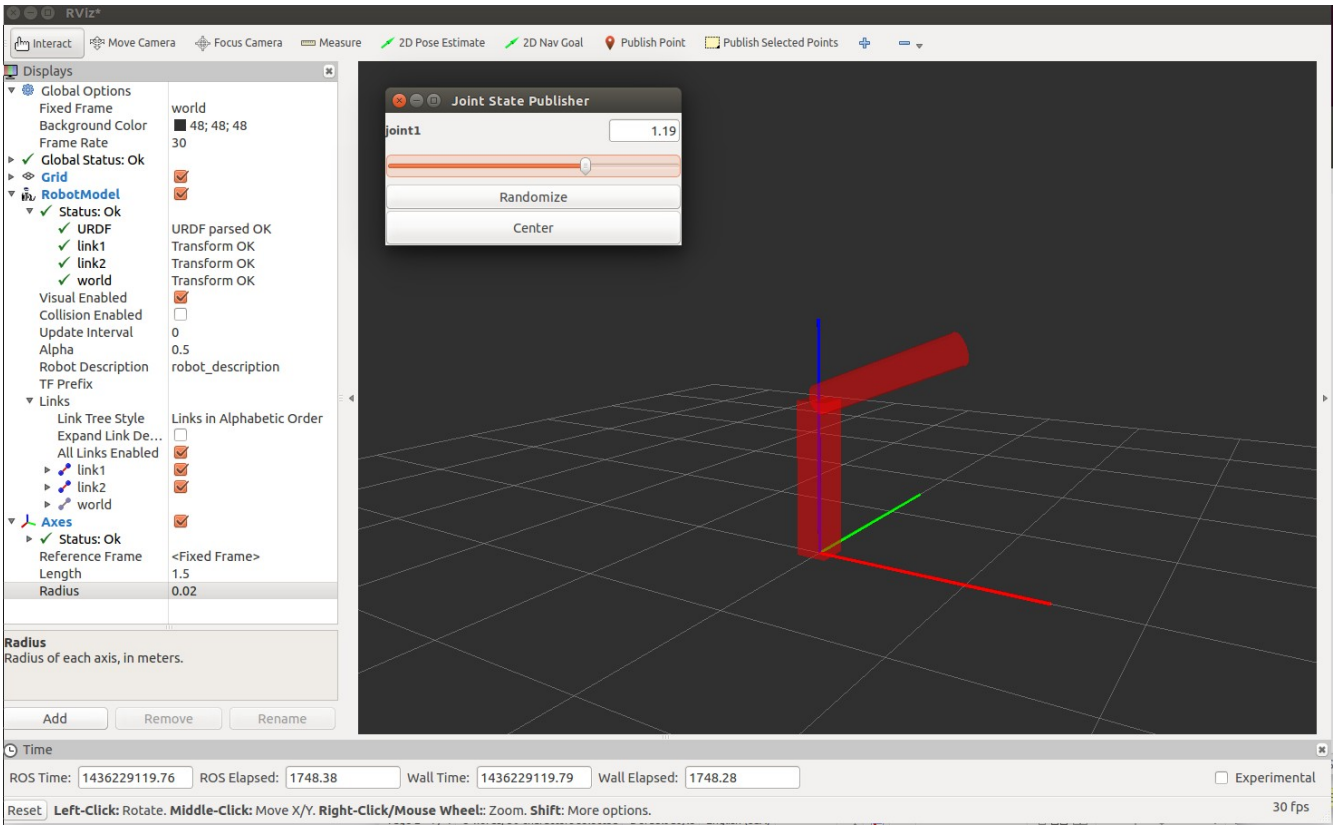
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Running:
rostopic echo joint_states
shows that there is a single joint named "joint1" and the corresponding value of "position" varies as the GUI slider is moved (agreeing with the value displayed in the GUI box). By this means, the value of the angle of joint1 (in radians) can be communicated to the ROS system.

This is not yet enough for Rviz to understand the 3D pose of link2. A second node is needed, which combines information from the robot model and from the joint1 value to compute the "transform" to get the pose of link2. Running:

rosrun robot_state_publisher robot_state_publisher

Results in publication of the missing transform, which enables Rviz to display the links with full knowledge of their 6D poses in world coordinates. As shown in the screenshot below, the RobotModel item in Rviz no longer displays errors. The slider GUI value of 1.19 results in link2 “cylinder” rotated 1.19 rad from vertical about a virtual joint (with axis parallel to the link2 y-axis) at the top of the link1 “box.”



Conclusion: Displaying robot models in Rviz is more complex than simple markers. A robot model must be described in detail, e.g. in URDF format, and the entire model specification gets loaded into the parameter server, from which Rviz can access the robot model. Since the joint-rotation values are variables, these must be specified by some means. Ordinarily, this information is published on the topic “joint_states” via a robot interface (physical or simulated), although this can be spoofed temporarily using the package joint_state_publisher. Additionally, the package robot_state_publisher is needed to combine model and joint-value information to compute link poses.

Although illustrated with only a single degree of freedom, this minimal example shows the process for defining and using robot models. More realistic robot models are considerably more detailed.

In the next step, “joint_state_publisher” is replaced with information from a robot simulator, “gazebo.” To get the simulator to provide this information, the robot model must be enhanced to incorporate dynamic information, notably, inertial properties.