1 Creating RazBot description

Next: 2 Expanding RazBot Description for Simulation

Before simulating or controlling a robot, we must create a representation of the robot within ROS. This tutorial will take you through the initial steps of creating a description package that will describe the robot using the ROS standard Unified Robot Description Format (URDF) language. You can find the output for this stage of the tutorials in https://github.com/clearpathrobotics/razbot_tutorials/tree/visual-only

1. Start by creating the a skeleton package for the description:

```
$ catkin_create_pkg razbot_description roslaunch
$ mkdir razbot_description/meshes razbot_description/launch
razbot_description/urdf
```

2. Modify razbot_description/package.xml. This file contains dependencies and other relevant information about our package:

```
<?xml version="1.0"?>
<package>
  <name>razbot_description</name>
  <version>0.0.1</version>
  <description>Robot description for the RazBot</description>

  <maintainer email="pbovbel@clearpathrobotics.com">Paul Bovbel</maintainer>
  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roslaunch</build_depend>
  <run_depend>robot_state_publisher</run_depend>
  <run_depend>urdf</run_depend>
  <run_depend>xacro</run_depend>
</package>
```

3. Modify razbot_description/CMakeLists.txt. This file tells catkin, the ROS build system, how to build our package. Since our package contains a robot description, there's not a whole lot to compile:

```
cmake_minimum_required(VERSION 2.8.3)
project(razbot_description)
find_package(catkin REQUIRED COMPONENTS roslaunch)

catkin_package()

# We add a roslaunch_add_file_check(...) macro that will do some basic validation
on our launch files.
roslaunch_add_file_check(launch)

install(
  DIRECTORY launch meshes urdf
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
)
```

4. Create launch/description.launch. When started, this launch file will store our robot description on the ROS parameter server, and start a robot_state_publisher node:

```
<?xml version="1.0"?>
<launch>

  <!-- Send the RazBot URDF to param server -->
  <param name="robot_description" command="$(find xacro)/xacro.py '$(find
razbot_description)/urdf/razbot.urdf.xacro'" />

  <!-- This node will publish transforms for any fixed joints -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />

</launch>
```

5. Create urdf//razbot.urdf.xacro. This file contains the top-level description for our robot, which is comprised of links and joints. Links represent discrete components of our robot, and joints describe how the links are connected. This description contains some basic visual information about the robot, see the inline comments for details:

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

  <!-- Include wheel macro -->
  <xacro:include filename="$(find razbot_description)/urdf/wheel.urdf.xacro" />

  <!-- <property> tags encode important robot paremeters. It's helpful to break
  these out from the rest of the description for easy manipulation -->
  <property name="M_PI" value="3.14159"/>
  <!-- Base Size -->
  <property name="base_length" value="0.184" />
  <property name="base_width" value="0.120" />
  <property name="base_height" value="0.068" />

  <!-- Wheel Properties -->
  <xacro:property name="wheel_base" value="0.123" />
  <xacro:property name="wheel_track" value="0.146" />
  <xacro:property name="wheel_vertical_offset" value="0" />
  <xacro:property name="wheel_width" value="0.025" />
  <xacro:property name="wheel_width" value="0.025" />
  <xacro:property name="wheel_radius" value="0.040" />

  <!-- Calculated Properties -->
  <xacro:property name="base_vertical_offset"
value="${-base_height/2+wheel_radius-wheel_vertical_offset}" />

  <!-- <material> tags are used to describe the colour of robot components -->
  <material name="Blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>
  <material name="Grey">
    <color rgba="0.7 0.7 0.7 1.0"/>
  </material>

  <!-- Some links don't have visual properties, and exist for convenience -->

  <!-- Footprint link, always on floor -->
  <link name="base_footprint"/>
```

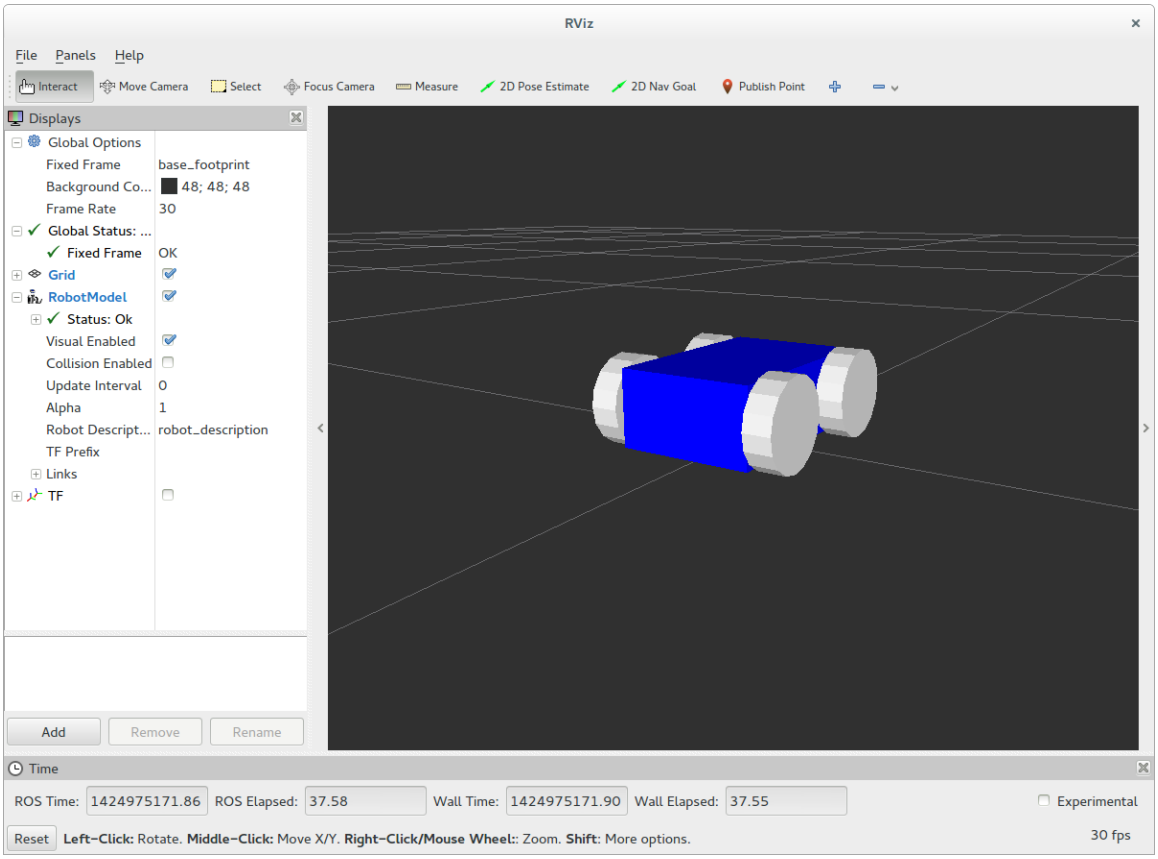
```
  <!-- Joints connect separate links together. Fixed joints do not move -->

  <!-- Footprint to base joint -->
  <joint name="footprint_joint" type="fixed">
    <origin xyz="0 0 ${base_vertical_offset}" rpy="0 0 0" />
    <parent link="base_footprint" />
    <child link="base_link" />
  </joint>

  <!-- Base link, represents center of mass for the robot -->
  <link name="base_link">

    <!-- <visual> tags describe how a link looks. They can contain a built-in
    shape (e.g. box, cylinder), or a complex shape (e.g. *.stl, *.dae files) -->

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <box size="${base_length} ${base_width} ${base_height}"/>
      </geometry>
      <material name="Blue" />
    </visual>
  </link>

  <!-- Xacro is an XML macro language that allows us to save effort when creating
similar
  XML blocks. Here, we create four wheels from the macro contained in the
urdf/wheel.urdf.xacro
  file -->

  <!-- Create wheels from macro -->
  <xacro:razbot_wheel wheel_prefix="front_left">
    <origin xyz="${wheel_base/2} ${wheel_track/2} ${wheel_vertical_offset}"
rpy="0 0 0" />
  </xacro:razbot_wheel>

  <xacro:razbot_wheel wheel_prefix="front_right">
    <origin xyz="${wheel_base/2} ${-wheel_track/2} ${wheel_vertical_offset}"
rpy="0 0 0" />
  </xacro:razbot_wheel>

  <xacro:razbot_wheel wheel_prefix="rear_left">
    <origin xyz="${-wheel_base/2} ${wheel_track/2} ${wheel_vertical_offset}"
rpy="0 0 0" />
  </xacro:razbot_wheel>

  <xacro:razbot_wheel wheel_prefix="rear_right">
    <origin xyz="${-wheel_base/2} ${-wheel_track/2} ${wheel_vertical_offset}"
rpy="0 0 0" />
```

```
  </xacro:razbot_wheel>

</robot>
```

6. Create urdf//wheel.urdf.xacro. This file defines a wheel 'macro' using xacro (an XML macro language). This macro is used in the previous step to generate four independent wheel links, see the inline comments for details:

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

  <!-- Wheel macro definition. The entirety of the code below is reproduced
  whenever the macro is called. We use 'params' to add additional arguments
  to the macro call, which can be simple parameters, or entire code blocks.
  See http://wiki.ros.org/xacro for more details. -->
  <xacro:macro name="razbot_wheel" params="wheel_prefix *joint_pose">

    <!-- Wheel link -->
    <link name="${wheel_prefix}_wheel_link">
      <visual>
        <origin xyz="0 0 0" rpy="${M_PI/2} 0 0" />
        <geometry>
          <cylinder length="${wheel_width}" radius="${wheel_radius}" />
        </geometry>
        <material name="Grey" />
      </visual>
    </link>

    <!-- Wheel to base joint. This joint is continuous so that the wheel can
    spin freely -->
    <joint name="${wheel_prefix}_wheel" type="continuous">
      <parent link="base_link"/>
      <child link="${wheel_prefix}_wheel_link"/>
      <xacro:insert_block name="joint_pose"/>
      <axis xyz="0 1 0" rpy="0 0 0" />
    </joint>

  </xacro:macro>
</robot>
```

7. Now you're ready to take a look at the RazBot!
    a. Run roslaunch razbot_description description.launch
    b. In a separate window, run rosrun joint_state_publisher joint_state_publisher. This additional node is needed to connect our continuous joint to the rest of the model. Later on, our simulator will take care of this.
    c. In a separate window, run rosrun rviz rviz. Click Add->Robot Model->Okay. You should see something like the below image:


RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
base_footprint
Background Co...
48; 48; 48
Frame Rate
30
Global Status: ...
Fixed Frame
OK
Grid
RobotModel
Status: Ok
Visual Enabled
Collision Enabled
Update Interval
0
Alpha
1
Robot Descript...
robot_description
TF Prefix
Links
TF
Add
Remove
Rename
Time
ROS Time:
1424975171.86
ROS Elapsed:
37.58
Wall Time:
1424975171.90
Wall Elapsed:
37.55
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
30 fps