Workshop “ROS for Engineers” - deel 3

Author (NL): Eric Dortmans (e.dortmans@fontys.nl)

Translation (EN): Kris Piters (k.piters@fontys.nl)

Preparation

Update the ros_examples stack you installed in the first session:

```
cd ~/catkin_ws/src/ros_examples
git pull
cd ~/catkin_ws
catkin_make
```

Install the required binary ROS packages:

```
sudo apt-get install ros-kinetic-urdf
sudo apt-get install ros-kinetic-urdf-tutorial
sudo apt-get install liburdfdom-tools
sudo apt-get install ros-kinetic-turtlebot
sudo apt-get install ros-kinetic-gazebo-ros-pkgs
sudo apt-get install ros-kinetic-gazebo-ros-control
sudo apt-get install ros-kinetic-joint-state-controller
sudo apt-get install ros-kinetic-effort-controllers
sudo apt-get install ros-kinetic-position-controllers
sudo apt-get install ros-kinetic-moveit
```

Install the Gazebo demo package and turtlebot_arm from source:

```
cd ~/catkin_ws/src/
git clone
   https://github.com/ros-simulation/gazebo_ros_demos.git
git clone https://github.com/turtlebot/turtlebot_arm.git
cd ..
catkin_make
```

URDF files

All the files for this excercise can be found in the package *urdf_examples* . You can go to the relevant folder with URDF files via the following command:

```
roscd urdf_examples/urdf
```

URDF test

View the file *test_robot.urdf* in your editor:

```
gedit test_robot.urdf
```

Create a PDF file with a graphic representation of the model:

```
urdf_to_graphiz test_robot.urdf
evince test_robot.pdf
```

Now visualize this model in RViz (set *Fixed Frame* to *link1*):

```
roslaunch urdf_tutorial display.launch
   model:=test_robot.urdf
```

- What do and/or don't you see?

Open a new terminal and view the ROS Graph to see what was launched by the *display.launch* file:

```
rqt_graph
```

Position a Link

We are now going to look at a robot with a little more shape. Open the model *block.urdf* in an editor:

```
gedit block.urdf
```

Open a new terminal to visualize this model in RViz (set *Fixed Frame* to *world*):

```
roslaunch urdf_tutorial display.launch model:=block.urdf
```

As you can see, the model contains 1 block.

Uncheck *RobotModel* if you only want to see the TF output or uncheck *TF* if you only want to see the *RobotModel*.

If you look closely, you will see that this block is not placed well on the ground (*world*). It is, as it were, half submerged. Why does this happen?

Adjust the URDF file, so that the block is properly placed on the ground, and restart the visualization.

Connecting Links in URDF and URDF+XACRO model

View the model *blocks.urdf* in an editor:

```
gedit blocks.urdf
```

Create a PDF file with a graphic representation of the model and view it:

```
urdf_to_graphiz blocks.urdf
evince blocks.pdf
```

Visualize this model in RViZ (set *Fixed Frame* to *world*):

```
roslaunch urdf_tutorial display.launch model:=blocks.urdf
```

You now see 3 blocks: a red one, a green one and a blue one.

Uncheck the *RobotModel* and take a look at the *TF* output.

Adjust the URDF file to stack the blocks, making sure the whole stack is placed neatly on the ground: the green block on top of the red block, the blue block on top of the green block. Relaunch RViz to visualize the model after you are (partly) done making adjustments.

If successful, open the model *blocks.urdf.xacro*:

```
gedit blocks.urdf.xacro
```

Modify this URDF + XACRO model to neatly stack the blocks. You will notice that this is much less work.

If successful in modifying the URDF + XACRO model, restart the visualization with the extra *gui:=true* parameter:

```
roslaunch urdf_tutorial display.launch
   model:=blocks.urdf.xacro gui:=true
```

Adjust the *sliders* and press the *random* and *center* buttons in the GUI. Observe what happens to the visualisation.

The robot_description parameter

You can also request the parameter server for a copy of the current robot model: Let the visualization (*blocks.urdf.xacro*) run and open a new terminal. Then enter the following command:

```
rosparam get -p /robot_description | tail -n +2 |cut -c
   3- > robot_description.urdf
```

You simply requested the current value of the ‘robot_description’ parameter and stored it in *robot_description.urdf*.

Open the file you have created:

```
gedit robot_description.urdf
```

Also open the blocks.urdf file (from the *urdf_examples* package):

```
gedit blocks.urdf
```

Is there a difference?

Modeling a mobile robot

We are going to build a mobile robot, or at least model it.

Open the modelfile *mobile_robot.urdf.xacro* in an editor:

```
gedit mobile_robot.urdf.xacro
```

Visualize this model:

```
roslaunch urdf_tutorial display.launch
    model:=mobile_robot.urdf.xacro
```

The model is off, it’s wheels aren’t properly attached to the base.

- Correctly position the wheels.

Next, add a Kinect and it’s support to your model by adding the following lines to your modelfile:

```
<joint name="kinect_support_joint" type="fixed">
  <parent link="base_link" />
  <child link="kinect_support" />
  <origin xyz="-0.118 0 ${base_height/2}" rpy="0 0 0" />
</joint>

<link name="kinect_support">
  <visual>
    <origin xyz="0 0 ${kinect_support_height/2}" rpy="0
        0 0" />
    <geometry>
      <box size="0.05 0.05 ${kinect_support_height}" />
    </geometry>
      <material name="grey_blue">
        <color rgba="0.4 0.4 1.0 1"/>
      </material>
    </visual>
```

```
</link>

<xacro:include filename="$(find
   turtlebot_description)/urdf/sensors/kinect.urdf.xacro"/>
<sensor_kinect parent="base_link"/>
```

Restart the visualization:

```
roslaunch urdf_tutorial display.launch
   model:=mobile_robot.urdf.xacro
```

Uncheck TF to get a good look of the model. If all went well, everything should fit together neatly.

Stage simulation

We have now modeled a nice robot platform, but can it also move together with a real or simulated robot?

Keep the visualization running and start a simulated world in Stage from a new terminal:

```
roslaunch stage_worlds kinect_world.launch
```

Set the Fixed Frame in RViz to *odom*.

Start the *teleop_twist_keyboard* **OR** *mouse_teleop* node to control the robot:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py

roslaunch mouse_teleop mouse_teleop.launch
   mouse_vel:=cmd_vel
```

You will notice that the robot model moves together with the simulated robot (as the transformation between *base_link* and *odom* is being updated by Stage).

Adding an arm to the mobile robot

We will add an arm to complete our robot. Add the following to the model file (source: *turtlebot_arm.urdf.xacro*):

```
<xacro:property name="joints_vlimit" value="1.571"/>
<xacro:property name="pan_llimit" value="-2.617"/>
<xacro:property name="pan_ulimit" value="2.617"/>
<xacro:property name="shoulder_llimit" value="-2.617"/>
<xacro:property name="shoulder_ulimit" value="2.617"/>
<xacro:property name="elbow_llimit" value="-2.617"/>
```

```
<xacro:property name="elbow_ulimit" value="2.617"/>
<xacro:property name="wrist_llimit" value="-1.745"/>
<xacro:property name="wrist_ulimit" value="1.745"/>

<include filename="$(find
   turtlebot_arm_description)/urdf/turtlebot_arm.xacro"
   />
<turtlebot_arm parent="base_link" color="White"
   gripper_color="Green"
               pincher_gripper="false"
                  turtlebot_gripper="true">
  <origin xyz="0.12 0 0.02"/>
</turtlebot_arm>
```

Can we also move the arm?

Restart the visualization with the *gui:=true* option:

```
roslaunch urdf_tutorial display.launch
   model:=mobile_robot.urdf.xacro gui:=true
```

Gazebo simulation

To experiment with Gazebo we use the RRBot (Revolute-Revolute Manipulator Robot). RRBot is a simple robot arm with 3 links and 2 joints. It is essentially a *double inverted pendulum*.

Take a look at the modelfile for the robot:

```
gedit `rospack find rrbot_description`/urdf/rrbot.xacro
```

Start RRBot in RViz to see if the model works:

```
roslaunch rrbot_description rrbot_rviz.launch
```

Control the joints. As you can see there is no gravity in RViz and all movements are infinitely fast.

Next, start the RRBot in Gazebo (this may take a while):

```
roslaunch rrbot_gazebo rrbot_world.launch
```

The joints' motors are still unarmed, so gravity has free play. Wait and see what happens under the influence of gravity.

Load and start the joint position controllers:

```
roslaunch rrbot_control rrbot_control.launch
```

Use the following commands as an example for controlling joint1 and/or joint2:

```
rostopic pub -1
   /rrbot/joint1_position_controller/command
   std_msgs/Float64 "data: 1.5"
rostopic pub -1
   /rrbot/joint2_position_controller/command
   std_msgs/Float64 "data: 1.0"
```

MoveIt!

Next, we are going to do motion planning with the arm of our mobile robot.

Start MoveIt in RViz:

```
roslaunch turtlebot_arm_moveit_config
   turtlebot_arm_moveit.launch sim:=true --screen
```

Go to the *Planning* tab. Pull the interactive markers to select a target (*Goal State*), then press *Plan* to have a trajectory calculated. Finally press *Execute* to have the plan executed by the arm.

If you are here, you have hopefully learned how to do a lot with ROS without any programming.

References

- rviz userguide
- ROS Tutorials
- ROS-Industrial Training Exercises
- XML Robot Description Format (URDF)
- URDF Tutorials
- Create a URDF for an Industrial Robot
- List of moments of inertia
- Specification for TurtleBot Compatible Platforms
- Tutorial: Using a URDF in Gazebo
- Tutorial: ROS Control
- MoveIt