ROS-I Basic Training “Mobility”
ROS Gazebo Tutorial

Instructor: MASCOR Institute

2017ff

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Contents

1 Introduction

This tutorial elaborates on how to use the simulation environment Gazebo. It includes a world, which presents a copy of the real-world playground and a model of the Turtlebot3 itself.

- Lines beginning with $ are terminal commands
- Lines beginning with # indicate the syntax of the commands

2 Terminal usage

- opening a new terminal : ctrl+alt+t
- opening a new tab inside an existing terminal : ctrl+shift+t
- killing an active process inside a terminal: ctrl+c

3 Prerequisites

Make sure to have the following Github repositories cloned in your workspace (please refer to the Filesystem Tutorial to understand how to clone git repositories):

```
https://github.com/nlimpert/turtlebot3.git
https://github.com/nlimpert/turtlebot3_simulations.git
```

4 Gazebo Simulation

The model of the playground and the model of the turtlebot3 can be found in the package **turtlebot3_gazebo**. Start the simulation by typing in a terminal on your desktop PC:

```
$ roslaunch turtlebot3_gazebo roundTrack_simulation.launch
```

The simulated turtlebot3 has an IMU, a RGB camera and a laser range finder available. All the actuators and sensors transmit and receive their data via the same topic names as their real-world counterparts.

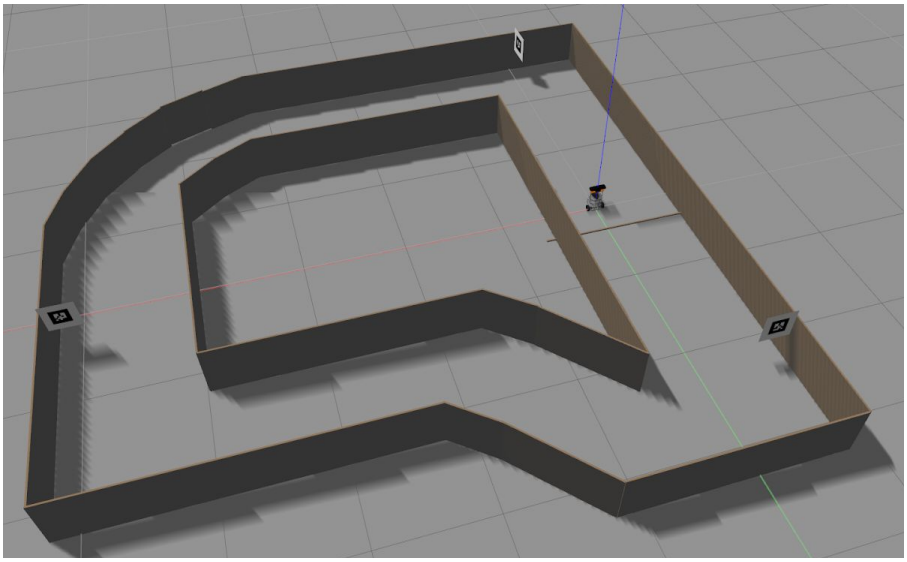

Figure 1: Gazebo simulation of turtlebot3 track

The simulation environment of Gazebo offers a physical model. The physical model includes characteristics like gravity or collision behaviour. For further information regarding Gazebo: `http://wiki.ros.org/gazebo_ros_pkgs`.

5 Interact with the simulated Turtlebot3

5.1 Motion control

To send messages of type geometry_msgs/Twist you can make use of a graphical tool called *rqt_robot_steering*:

```
$ rosrun rqt_robot_steering rqt_robot_steering
```

A window looking like Figure 2 should show up. You can send motion commands by manipulating the two sliders.

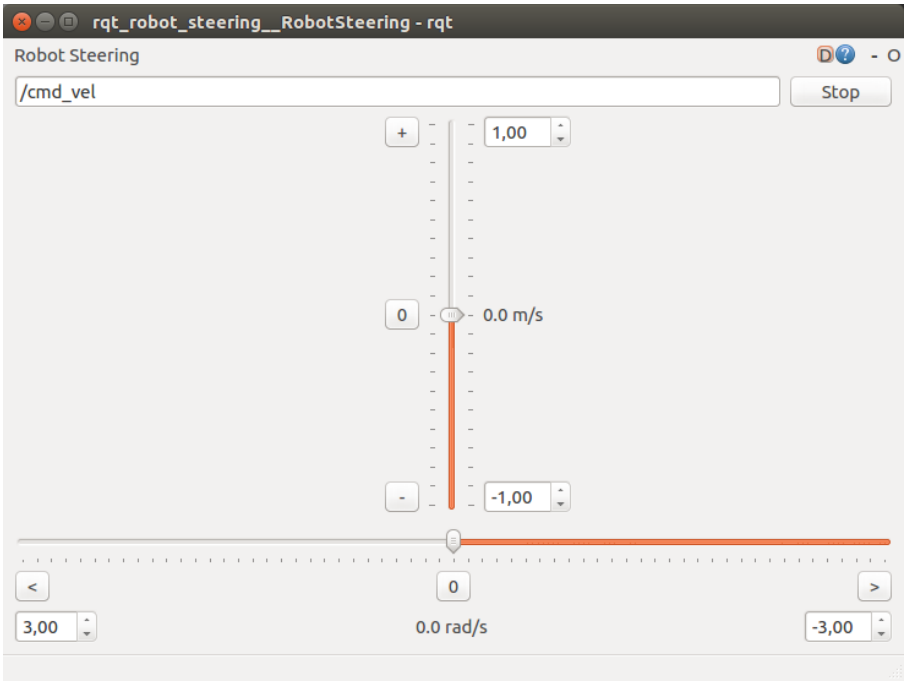

Figure 2: *rqt_robot_steering*

5.2 Image acquisition

To view the camera image do the following:

```
$ rosrun rqt_image_view rqt_image_view
```

A window looking like Figure 3 should show up. Set the topic name to */camera/image_raw*.

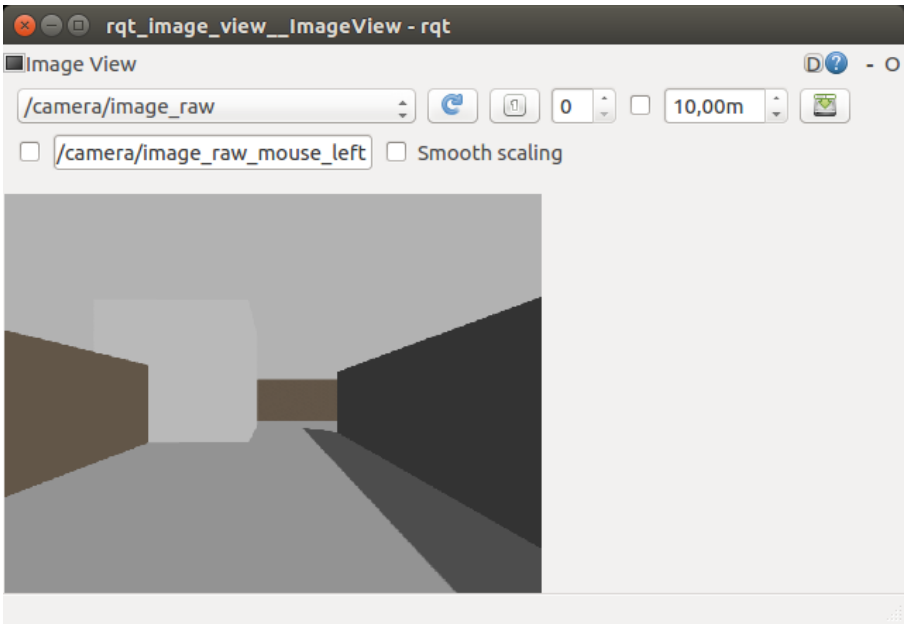

Figure 3: *rqt_image_view*

5.3 RQT Graph

The tool *rqt_graph* allows to visualize subscriptions and publications of nodes:

```
$ rosrun rqt_graph rqt_graph
```

After de-selecting "Debug" in the GUI (see Figure 4, you should get an overview about the different publications and subscriptions of topics (oval) on different nodes (rectangles).
Make sure to set "Nodes/Topics (all)" in the top left.

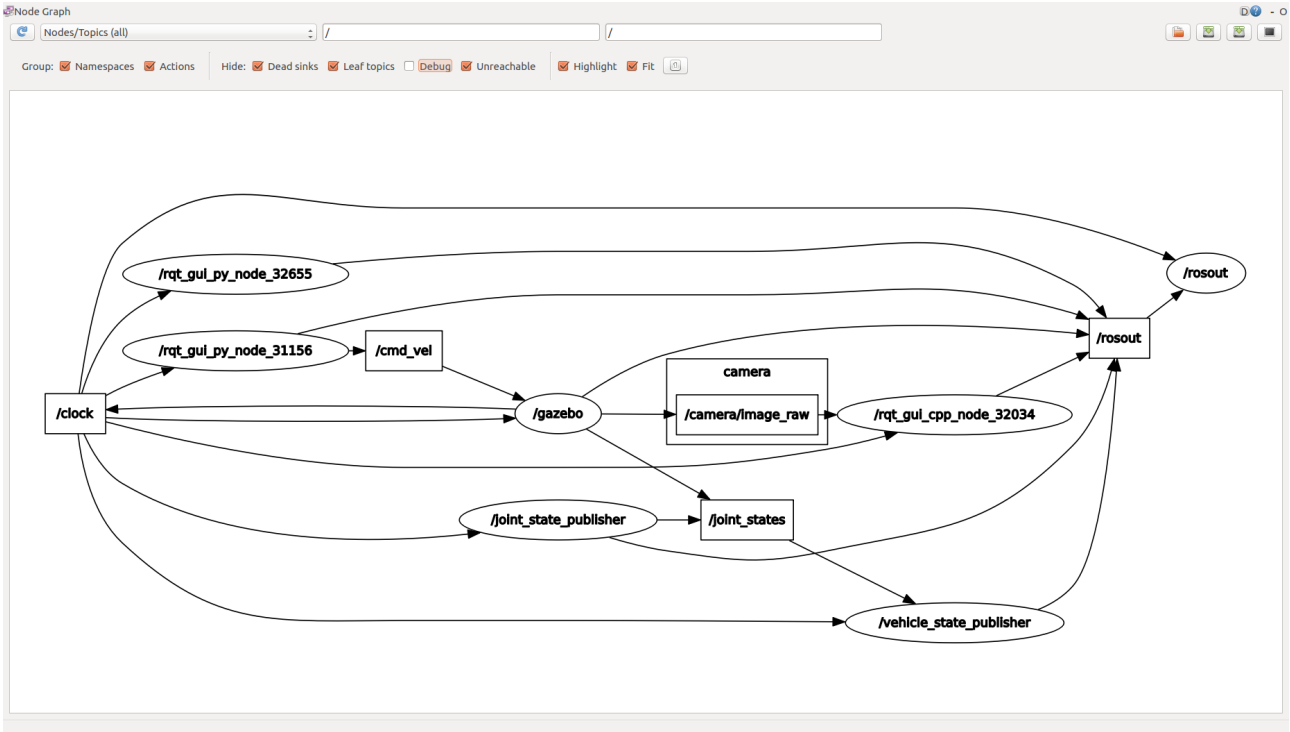

Figure 4: *rqt_graph* after running *rqt_robot_steering* and *rqt_image_view*

6 Modify the Turtlebot3 model

The **U**nified **R**obot **D**escription **F**ormat (URDF) is a XML based file format used in ROS to describe the robot's mechanical configuration. URDF can be used in Gazebo to visualize and simulate robots. To modify the robot model you have to edit the specific model file. Therefore, type in a terminal:

```
$ roscd turtlebot3_description
$ cd urdf
$ gedit turtlebot3_burger.urdf.xacro
```

As explained during the lecture the model consists of multiple links, which are connected via joints.

For more information regarding the URDF format, look at `http://wiki.ros.org/urdf`.
We want to add a flag to the robot model. Since the turtlebot3 model is not beginner friendly, we will use include mechanics to extend the model. Close all running processes and terminals at this point, except the gedit editor!
By adding the following line to the end of the *turtlebot3_burger.urdf.xacro* file, the flag model is included into the original turtlebot3 model.

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

Now you have to create the flag model. Create a new package in your workspace and name it **turtlebot_urdf**. It should have the dependencies **rospy**, **turtlebot3_description** and **urdf**. Create a folder inside your new package and name it *urdf*. Inside this folder create an empty document and name it *flag.urdf*. Add the following to the *flag.urdf* file:

Every link and joint of a robot needs to be defined inside a robot tag. Add them to your *flag.urdf*.

```
<robot>

<!-- INSERT YOUR ROBOT DESCRIPTION HERE -->

</robot>
```

Now add a pole to the flag model. Therefore, create a new link named *pole_link*. Make sure to insert the link definition inside the robot tag.

```
<link name="pole_link">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <cylinder radius="0.001" length="0.075"/>
        </geometry>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
            <cylinder radius="0.001" length="0.075"/>
        </geometry>
    </collision>

    <inertial>
        <mass value="0.001"/>
        <inertia ixx="0.001" ixy="0" ixz="0" iyy="0.001" iyz="0" izz="0.001"
            ↪ />
    </inertial>
</link>
```

The visual and the collision element are featuring the same cylindrical geometry. In most cases, it is recommended to use a simplified shape of the visual element to create a collision element.

It lowers the needed computation power by a lot. The inertial used is called null inertia. The link has a mass of 0.001 kg and the inertia matrix is close to zero, hence the link will not change the physics of the Turtlebot3.
Note: The inertial tag is mandatory for a gazebo model. If not specified gazebo will ignore the link.
Next step is to define a joint between the *pole_link* and the mountplate of the original turtlebot3 model. Be aware to insert the joint definition inside the robot tag.

```
<joint name="pole_mountplate_joint" type="fixed">
    <origin xyz="0 -0.05 0.135" rpy="0 -0.2 0" />
    <parent link="mountplate_link"/>
    <child link="pole_link" />
</joint>
```

The joint *pole_mountplate_joint* connects the links *mountplate_link* and *pole_link*. The parameter origin describes the pose of the new joint in relation to its parent frame. Since the joint is of the type fixed, we do not have to specify a rotation axis.
Add a flag at the top of the pole. Therefore, follow the same procedure. An example is given below.

```
<link name="flag_link">
    <visual>
        <origin xyz="-0.022 0 0" rpy="0 0 0"/>
        <geometry>
            <box size="0.045 0.001 0.03"/>
        </geometry>
    </visual>
    <inertial>
        <mass value="0.001"/>
        <inertia ixx="0.001" ixy="0" ixz="0" iyy="0.001" iyz="0" izz="0.001"
            ↪ />
    </inertial>
</link>
```

```
<joint name="flag_pole_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <origin xyz="0 0 0.02" rpy="0 0 0" />
    <parent link="pole_link"/>
    <child link="flag_link" />
</joint>
```

As compared to the link *pole_link*, no collision object is defined. In that case, no collision with other links will be detected. The joint is from type continuous and the z-axis is defined as a rotation axis, because the flag should be able to rotate.
To test your new robot model create a launch file *model.launch* in your package **turtlebot_urdf**.

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

<launch>

    <param name="robot_description" command="$(find xacro)/xacro '$(find
       ↪ turtlebot3_description)/urdf/turtlebot3_burger.urdf.xacro'" />

    <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" />

    <!-- rviz -->
    <node name="rviz" pkg="rviz" type="rviz" respawn="false" output="screen"
       ↪ />

</launch>
```

This launch file will start your robot model and the visualization tool RViz. In RViz choose the fixed frame base_footprint and Add a robot model. Is the flag correctly shown? If yes, restart the Gazebo simulation to simulate the modified robot model.

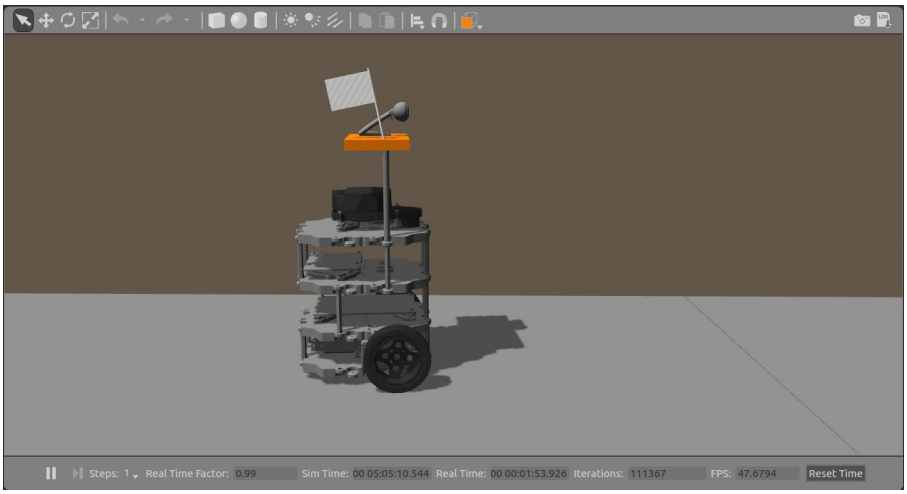

Figure 5: Turtlebot3 Burger with flag

7 Visualization in Rviz

7.1 tf tree

The links on the Turtlebot3 represent frames that have invariant coordinate transforms with each other, and form an overall tree structure.
To visualize the tf tree within Rviz:

```
$ rosrun rviz rviz
```

Click the **Add** button and choose **TF**, to visualize the tf information.
Hint: Switch your fixed_frame in RVIZ to the start frame of your tf tree, what is right now base_footprint.

7.2 Laserscan Data

Click the **Add** button and choose **Laserscan**, to visualize the laser data. The laser scan data is published with reference to the frame base_scan but is still available in this frame, because the transformation between the frames base_footprint and base_scan is given by the tf tree

(see fig. 6).

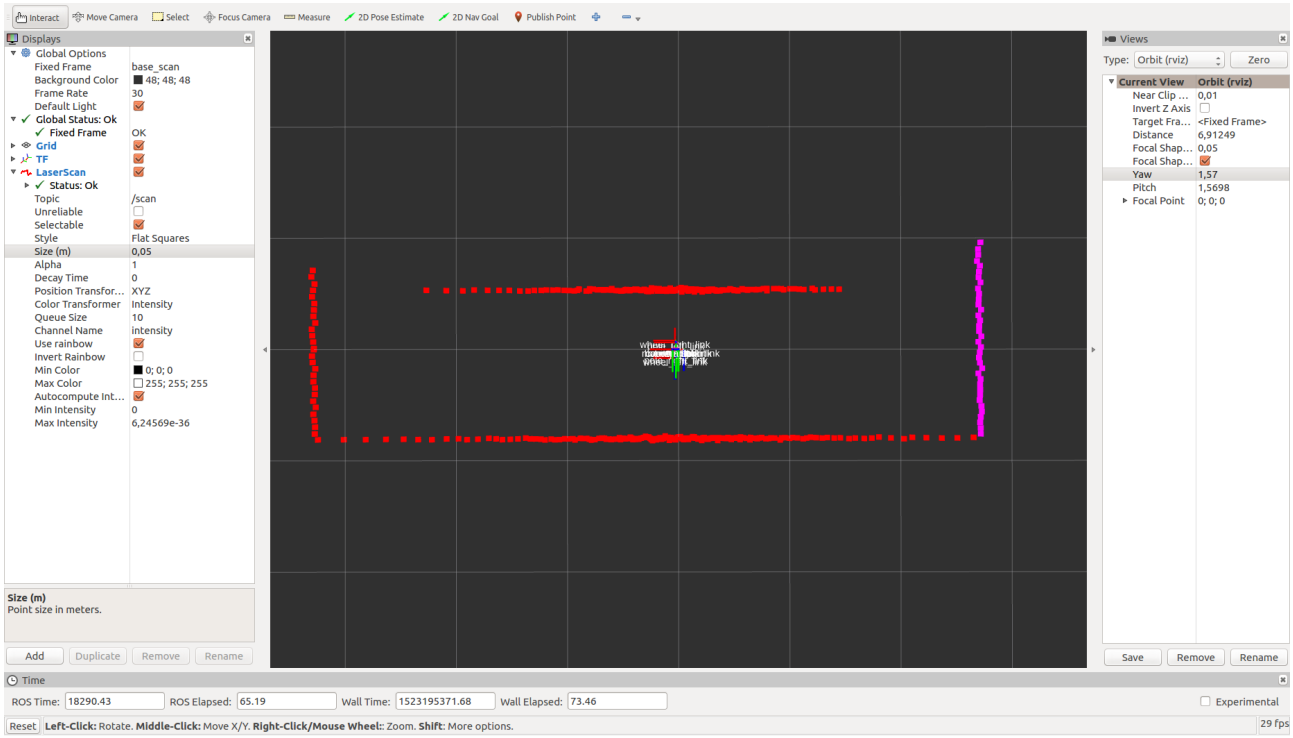

Figure 6: static TF tree of Turtlebot3 with laserscan