Robotic Systems Programming
Practical Session 2 (TP-2)
ROS and Gazebo Simulation; Introduction to TurtleBot2

Panagiotis PAPADAKIS

1 Introduction

The objective of TP-2 is to extend your experience with ROS by making use of its physics simulation software named **Gazebo**. Simulating the robot and its behaviour within a given environment is useful in a number of aspects:

- It accelerates **algorithm prototyping**
- It **protects** the real robot and surroundings from unrecoverable damage
- It allows performing multiple **repetitive** experiments
- It sets an **upper bound** of what the real robot can do;
  If it does not work in simulation, it will never work in practice. If it does works in simulation, it will hopefully work in practice.

Gazebo is originally developed outside ROS and can be used independently. Using Gazebo inside ROS, will allow us to use the exact same code to make a robot run in simulation or the real robot, by merely interchanging the names of the ROS topics that are used by the simulated robot and the real robot.

Setting up ROS simulation packages Unless you have installed the full ROS version, you will need to download and install new ROS packages for the purposes of the TP. Make sure you have working internet connection and perform the following:

```
% sudo apt-get update
% sudo apt-get install ros-kinetic-turtlebot-gazebo
% source ~/.bashrc
```

The above should download and install the necessary new packages for using the turtlebot simulation (along with many other ROS packages which are its dependencies).

You can explore the new ROS packages that have just been installed. For any ROS package that resides in a catkin workspace (see TP-1), you can type in the terminal:

```
% roscd <thenameoftheROSpackage>
```

to change to the corresponding directory. Try it for the `tp1_ros_package` that you created in the previous TP-1 and then for the newly installed package `turtlebot_gazebo` and quickly explore the content of the folder.

2 Gazebo simulation of TurtleBot2

In the end of TP-1, you were asked to launch a very simple simulation environment (`turtlesim`) and experiment by publishing ROS velocity commands. You will now use the Gazebo simulator to operate on a full-scale simulation of the TurtleBot2 robot, which models its exact kinematics and sensor capacities and its physical interaction with the environment.

This requires launching the Gazebo simulator environment and loading into the simulation an example 3D world and the TurtleBot2 robot within that world. The `turtlebot_gazebo` package will run all necessary ROS nodes with the necessary arguments, if you execute:

```
% roslaunch turtlebot_gazebo turtlebot_world.launch
```

This command should start Gazebo in the ROS ecosystem. Figure 1 shows a screen-shot of the environment.

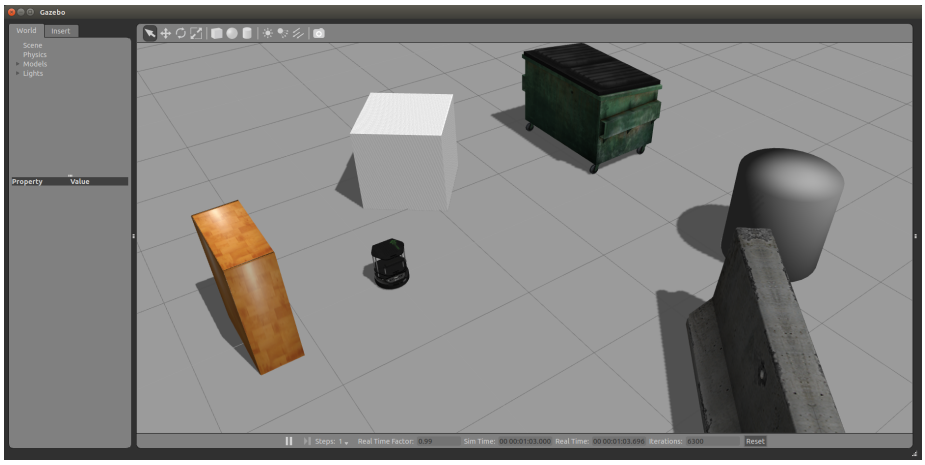

Figure 1: Gazebo simulation with TurtleBot2

3 Exercises

3.1 Getting the robot to move

1. **Adapt the velocity command** that you gave from the terminal to the Turtle in TP-1 (Section 5) so that you can give velocity commands to the TurtleBot2 inside Gazebo. The new ROS topic where you have to publish the velocity commands is:

   `/cmd_vel_mux/input/navi`

2. **Program a keyboard-based teleoperation** for the simulated TurtleBot2. To do so, create a new ROS package named as `tp2_ros_package` within your catkin workspace, `src` folder (created from TP-1). Then create a new ROS node that will receive as input from the keyboard the velocity commands and will publish a ROS message of type `geometry_msgs/Twist` to the topic where the TurtleBot2 expects to receive velocity commands.

   Your ROS node should have the following functionality:

   - Move the robot forward when the 'u' button is pressed
   - Move the robot backward when the 'j' button is pressed
   - Rotate the robot clockwise (right) when the 'k' button is pressed
   - Rotate the robot counter-clockwise 'left) when the 'h' button is pressed
   - Increase or decrease the linear and angular speed of the robot by 10% by pressing the buttons 'f' or 's' respectively

   Use the partially completed code provided in `https://moodle.imt-atlantique.fr/pluginfile.php/32286/mod_folder/content/0/teleoperation_node-partial.py?forcedownload=1`

3.2 Using ROS launch files

As you saw earlier in Section 2, there is a ROS command line tool called `roslaunch` which is helpful in setting up and running multiple ROS nodes, by putting everything together in a `.launch` file[1], with XML syntax[2].

In the following, you can see a minimal example of a ROS launch file[3] that starts two ROS nodes of the `tp1_ros_package`. If the ROS core process is not already running, the .launch file will start it as well:

```
<launch>
    <node pkg="tp1_ros_package" type="my_first_ros_node.py" name="my_publisher_node" output="screen" />
    <node pkg="tp1_ros_package" type="my_listener_ros_node.py" name="my_listener_node" output="screen" />
</launch>
```

[1] http://wiki.ros.org/roslaunch/Commandline%20Tools

[2] http://wiki.ros.org/roslaunch/XML

[3] You can download it from `https://moodle.imt-atlantique.fr/pluginfile.php/32286/mod_folder/content/0/example_launch_file.launch?forcedownload=1`

ROS launch files have to be placed in a special directory called "launch" within the folder of the respective ROS package. Create a "launch" folder inside the `tp1_ros_package` and copy the above file within that folder.

Execute in the terminal:

```
% roslaunch tp1_ros_package example_launch_file.launch
```

and inspect what is being executed (by ``rosnode list''), that should be equivalent to running the following three separate commands:

```
% roscore
% rosrun tp1_ros_package my_first_ros_node.py
% rosrun tp1_ros_package my_listener_ros_node.py
```

1. In the above ROS launch file, we only do limited use of possible XML *tags*. By referring to the documentation in `http://wiki.ros.org/roslaunch/XML` and `http://wiki.ros.org/roslaunch/XML/param`, use the `<param>` XML tag, in order to provide as parameters the associations between the keyboard buttons ('u', 'j', 'k', 'h', etc) and the control commands of your `tp2_ros_package`. In other words, make your package retrieve the keyboard mappings from within a ROS launch file, instead of hardc-oding them in your code.

   HINT: you will have to use the `rospy.get_param()` function in your python code in order to retrieve the parameter value (see `http://docs.ros.org/api/rospy/html/rospy-module.html#get_param`)

3.3 Exploring TurtleBot2 sensors

The simulation of the TurtleBot includes simulated models of its sensing capacities, from proprioceptive to exteroceptive (passive and active). An overview of its technical specifications can be accessed at `http://www.robotnik.es/web/wp-content/uploads/2014/04/TB_robot.pdf`.

Using the ROS command-line tools that you have learned so far and guided by the above documentation, you should gradually discover what the robot can sense. For example, the `/mobile_base/sensors/core` ROS topic incorporates a series of internal robot states and the robot is continuously publishing ROS messages in that topic to give information of its internal state.

Using the ROS command `rosmsg show -r <yourROSmessagetype>`, you can have a detailed (commented) description of the underlying ROS message.

1. Use the `rosmsg` command to discover what kind of observations (ROS messages) the TurtleBot2 is actually doing, by inspecting the total list of ROS topics that is being published within the ROS system that runs the simulation.

2. For ROS built-in sensing modalities (camera images, range scans, 3D point clouds, etc), ROS allows the visualization of the sensor readings via the ROS tool `rviz`. Download an rviz configuration file from `https://moodle.imt-atlantique.fr/pluginfile.php/32286/mod_folder/content/0/tp-2.rviz?forcedownload=1` and run:

   ```
   % rosrun rviz rviz -d <arg>
   ```

   by replacing `<arg>` with the path of the downloaded rviz config file. The rviz visualization tool should run with a preconfigured set of visualizations, as listed at the left part of the rviz window.

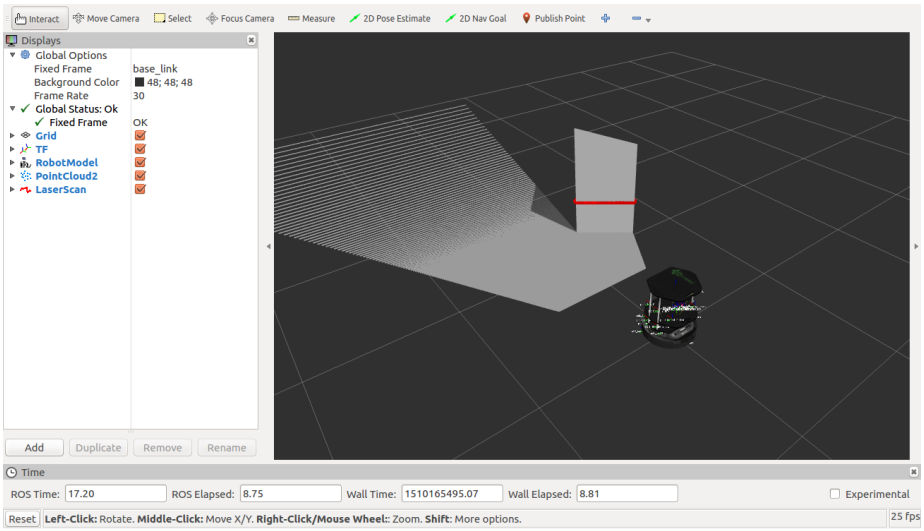

Figure 2: ROS rviz visualization of simulated TurtleBot2

By referencing to the rviz documentation[4], try to add to rviz visualization the following sensor data:

- RGB image
- Depth image
- IMU data

Once finished, save your modified .rviz configuration file to a new .rviz file named `tp_2-config.rviz`.

[4] http://wiki.ros.org/rviz/UserGuide