Lab1 - Introduction to Turtlebot

1 Introduction

This lab is designed to do a first approach to the Turtlebot 2 seen in Figure 1 and to get use to the ROS network configuration. This robot will be used in the upcoming lab sessions. You can find all the information related to this robot in `http://wiki.ros.org/Robots/TurtleBot`. Furthermore, this lab session is also an introduction to the ROS launch system known as launch files.

Figure 1: Turtlebot 2 with a Kubuki base

2 Lab work

Most of the work asked here is based on the TurtleBot tutorials from `http://wiki.ros.org/Robots/TurtleBot`. For the robots use *turtlebot* as user name and *turtlebot* as password.

2.1 Network configuration

In ROS, two main variables have to be set up before start working: the `ROS_MASTER_URI`, which tells the system which robot/computer will host the roscore, and the `ROS_HOSTNAME`, which hosts the pc's IP or name. Pay special attention to this two variables if the messages are not properly sent or received over the network. If this happens, it means that the network configuration is wrong and some of the variables can not be resolved to an IP address.

2.1.1 Working with your own computer

In case you will work with your own computers and want to command the robots, you will have to make the link between your computer and the robot's computer. In order to set up the network, connect to the appropriate wifi according to the robot's computer (*edurom*). Then, configure your network according to the IP of your turtlebot. To find the IP of your TurtleBot run in terminal `ifconfig` and look for the wifi address. To find the robot's name run in terminal the command `who`. For the configuration, three steps are done. First add the TurtleBot to the hosts file in */etc/hosts* of your laptop:

```
$ sudo sh -c 'echo TURLEBOT_IP ROBOT_NAME >> /etc/hosts'
```

Use the TURTLEBOT_IP and ROBOT_NAME from before and your computer's IP for YOUR_IP (you can see your computer's IP by using the `ifconfig` command). Then, set up your `ROS_MASTER_URI`:

```
$ echo export ROS_MASTER_URI=http://ROBOT_NAME:11311 >> /home/$(whoami)/.bashrc
```

finally set up your `ROS_HOSTNAME`:

```
$ echo export ROS_HOSTNAME=YOUR_IP >> /home/$(whoami)/.bashrc
```

For all this changes to take effect you have to reload your `.bashrc` file:

```
$ source /home/$(whoami)/.bashrc
```

To check if the connection is properly done, first we open an ssh connection to the robot:

```
$ ssh turtlebot@ROBOT_NAME
```

and there, we run a roscore. Then run in your pc's terminal:

```
$ rostopic list
```

This command should return a list of topics if a the ROS master has a core running. If the ROS core is running in the master, and the `rostopic list` does not connects to it, the network configuration is not well done.

2.2 Bringup

In order to start the robot, `ssh` into it's computer (if you use your own computer) OR open the terminal on the turtlebot's laptop (be sure the turtlebot is power ON) and run:

```
$ cd catkin_ws
$ source devel/setup.bash
$ roslaunch turtlebot_bringup minimal.launch
```

To start the Kinect on the robot run:

```
$ roslaunch turtlebot_bringup 3dsensor.launch
```

and, in a new terminal run rviz:

```
$ rviz
```

Now be sure to change the frame to `base_frame`, and add a camera image selecting the corresponding topic. You can also see your robot in rviz if you load the model.

2.3 Teleoperation

For the keyboard teleoperation run:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

2.4 Navigation and mapping

This robot has already built up packages which allow to build maps using SLAM and also to do autonomous navigation on a given map. First, we are going to use the mapping node to obtain a small map. After bringing up the turtlebot and setting up the keyboard teleoperation, run (in a new terminal):

```
$ roslaunch turtlebot_navigation gmapping_demo.launch
```

In order to visualize the map and the robot's position run again rviz and add the corresponding components (map, robot model) with the corresponding topics.

Drive the robot around to build a small map and capture a screen shot for the report. To save the map use, on the robot's pc:

```
$ rosrun map_server map_saver -f /tmp/my_map
```

In order to navigate the robot autonomously on that map launch (on the robot):

```
$ roslaunch turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml
```

and close the teleoperation. If you have closed Rviz on your workstation use the same command as for the mapping to open it again. Now click on "2D Pose Estimate" and click where the robot is within the map (click for the position, and point for the direction). Now tell the robot where to go using "2D Nav Goal". Capture the screen while the robot is computing the trajectory for the report.

2.5 Follower

Let's make the robot follow you. Run (on the robot):

```
$ roslaunch turtlebot_follower follower.launch
```

Now stand in front of the robot and move, it should follow you if you are not moving fast.

2.6 Turtlebot driver

Along with the lab documentation you can find a ROS package called `lab1_turtlebot`. This package contains the script where the lab programming will be done. Please modify the driver.py (`lab1_turtlebot/src/driver.py`)

in order to be able to make the Turtlebot go to a given position. For this purpose, please modify the constructor of the driver class where the definition of the subscriber, publisher and velocity message are missing. You will have to modify also the functions `read_position`, which stores the position received on a message to the class variables, and `compute_velocity` where the velocity message is created.

To launch the node run:

```
$ roslaunch lab1_turtlebot driver.launch
```

If no arguments are given to the `driver.launch`, the goal point is $(0, 0, 0)$ for (x, y, θ). If you want this points to change use:

```
$ roslaunch lab1_turtlebot driver.launch x:=x y:=y theta:=θ
```

3 Optional

Modify your `turtlebot_driver.py` in order to be able to read from a .txt file (given as input for the launch file) a whole trajectory instead of just going to one point. In this case you will have to modify also the function `load_goals` and `next_goal`. The trajectory file will have the following structure:

x_0,y_0,θ_0

x_1,y_1,θ_1

. . .

x_n,y_n,θ_n

and will be passed as an argument called `file` to the launch file. Please find an example of a trajectory in `config/list.txt`

4 Lab report

Upload a zip file containing the report (2 pages in pdf format) as well as the final `driver.py` file. If you have done the optional part, please submit the new launch file also.