Exercise 3: The Turtlebot

Development of Intelligent Systems

2019

In this exercise you will familiarize yourself with the Turtlebot robot platform that you will be using throughout this course. The robot is composed of a heavily modified iRoomba vacuum cleaner, a depth sensor (Microsoft Kinect), a laptop, and some construction material. Download and build the packages for this exercise.

☞ Keep in mind that the same hardware is used not just by your group but also by other students this year as well as in the future years. So pay attention to the maintenance instructions to maximize the life-expectancy of components. Frequent violations of these instructions will be sanctioned by reducing the grade of your group.

1 Setting up connection to the robot computer and driving the robot

When running your code on a real robot you should place the faculty laptop on it and use your own computer to connect to it, control the robot, and run visualization tools. The computer on the robot will be called *robot computer* and is meant to handle most of the sensor processing and direct robot communication. The other computer on your desk will be called *workstation*.

☞ Set up the faculty laptops so that they do not go into any power saving mode when the lid is closed. When working with the robot the laptop should be on the lowest available shelf on the robot because a lower center of gravity will prevent robot from swinging during rapid acceleration and will therefore improve its odometry.

Because of reduced power-saving abilities it is necessary to plug the laptop to a charger whenever possible (when the robot is not moving for a longer amount of time because you are working on your code).

ROS supports distributed execution so, when properly configured, nodes can run on multiple computers and communicate between themselves. On the workstation you can use SSH to obtain a remote terminal on the robot computer by using the following command:

```
ssh -l <team_name> <robot_name>.local
```

or

```
ssh <team_name>@<robot_name>.local
```

Where `<team_name>` is the name of your team and `<robot_name>` is the name of the robot computer. When logging in you can run all the commands on a robot computer in the same way as in the local workstation terminal (you will not be able to run nodes that have any kind of GUI though, those should be run on your workstation).

☞ Do not remove the laptop from its designated space on the robot during the operation unless you really need direct access to the keyboard or when finishing your work or interrupting robot operation for a longer amount of time.

Now run the following commands on the robot computer (through ssh):

```
roslaunch turtlebot_bringup minimal.launch
```

This will start all the necessairy nodes for the control of the robot. **All nodes that communicate directly with hardware have to be run on the robot computer**. Then, open a new local terminal and set the `ROS_IP` and `ROS_MASTER_URI` environmental variables:

```
export ROS_IP=<IP address of the workstation>
export ROS_MASTER_URI="http://<IP address of the robot computer>:11311"
```

☞ You have to set these variables for every new shell. This gets tedious after a few repetitions and you will have to set up a lot of terminals. It is therefore wise to automate this. You can do this by adding the followwing two lines to `.bashrc`:

```
export ROS_IP=`ifconfig | grep 'inet addr:'| grep -v '127.0.0.1' | cut -d: -↩
    f2 | awk '{ print $1}' | head -1`
export ROS_MASTER_URI="http://<IP address of the robot computer>:11311"
```

The first line fethces the current IP address of your laptop and sets the value of the `ROS_IP` environment variable to that value. The second line hard-codes the address of the ROS MASTER to the IP address of the robot computer.

Then run the following command (on the workstation):

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

This will provide an interface for moving the robot directly from your keyboard. Try moving it around for a bit to get a feeling for how the robot moves.

2 Visualizing robot sensors

ROS contains a very useful tool for visualizing the state of the robot called rviz. This tool contains a lot of useful plugins and you will be using it often. We can also preconfigure the plugins and the displays we want to visualize for our robot. You can configure it yourself, but the Turtlebot includes some `.launch` files with pre-configured rviz visualizations, located in the `turtlebot_rviz_launchers` package. To visualize the state of your robot, on the workstation run the command:

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

When we are building a map, or navigating the robot through the environment we usually use the prepared visualizations in the `view_navigation.launch` file:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

3 Autonomous navigation

In order for the robot to be able to navigate to a given goal in a complex environment it needs to have a few basic capabilities. First, we need a map of the working environment of the robot and if that map is not provided beforehand we also need a way to build that map. One we have a map, we need an algorithm for localizing the robot in the given map. When we have a way of determining the robot location at each point in time, we need an algorithm capable of planning a path in the environment to a given goal location. If the possibility exists that the environment can change during the robot operation (obstacle move positions, there are new obstacles or moving objects) we also need an algorithm of on-line obstacle avoidance and re-planning the trajectories. Finally, we need control algorithms for translating the planned trajectory into motor movement commands. In ROS there are packages that implement each of these components.

3.1 Map-building

For the Turtlebot we build a map using the gmapping package which builds a map based on the laser-scan data (which we get from the kinect) and the odometry data for the robot movement. Once the map is built we use the map_server package to load the map and make it available to all the nodes that need it.

For this exercise we are going to use the `gmapping_demo.launch` file provided in the `turtlebot_navigation` package which loads the gmapping nodes with some parameters pre-configured for the Turtlebot.

On the robot computer run:

```
roslaunch turtlebot_bringup minimal.launch
```

and (make sure that the kinect is powered on):

```
roslaunch turtlebot_navigation gmapping_demo.launch
```

On the workstation run:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

and in another terminal run:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Some tips for good map-building:

- Move the robot slowly. When the robot is moving quickly it loses the connection between individual scans and is unable to merge them together. Because of this the map is not expanded.
- Stop frequently and rotate around the axis to capture the entire neighborhood.
- Observe the current state that is shown in Rviz. The map is not refreshed in real time but rather in steps therefore make sure that the map has indeed been updated before moving on.

When you are satisfied with the map, you can save it by running:

```
rosrun map_server map_saver -f <the_name_of_your_map>
```

3.2 Localization and navigation

For localization of the robot we use the amcl (Adaptive Monte Carlo Localization) package. This package containes the `amcl` node which reads the laser-scan and the odometry data and provides a probabilistic location of the robot.

To actually navigate to a given goal in our map we are going to use the move_base package. Among other things, this package implements the move_base action API. We are going to use the `amcl_demo.launch` file provided in the `turtlebot_navigation` package which loads both the amcl node and the `move_base` action server with some parameters pre-configured for the Turtlebot. Additionally, this launch file also starts the `move_base` node which provides the navigation and control capabilities for the Turtlebot. Let us now try the navigation stack.

On the robot computer run:

```
roslaunch turtlebot_bringup minimal.launch
```

and:

```
roslaunch turtlebot_navigation amcl_demo.launch
```

Inside the `amcl_demo.launch` file you should set the correct path to your map. You can also do that by setting the correct environment variable. Note that if you start the `3d_sensor` node on the robot computer (separate it from the `amcl_demo.launch` file), you can start `amcl` and `move_base` on the workstation laptop.

On the workstation run:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Now once you give the original position of the robot, you should be able to send goals from rviz.

4 Sending movement goals from a node

In the `exercise3` package you have both C++ and Python examples of sending a goal from a node. For this purpose we are using a SimpleActionClient to communicate with the SimpleActionServer that is available in `move_base`. This node is based on the actionlib server/client, which can be viewed as an additional type of ROS Service, for requests that have a longer execution time. It gives us the capability for monitoring the execution status of our requests, canceling the request during execution and other options.

5 Homework

For the homework you need to have a pre-built map of the polygon. Create a node that:

- Has pre-determined (hardcoded) 5 goal locations on the map. You should determine this locations while navigating the map.
- Using the interface to `move_base`, sends the robot at the first goal location, waits until the goal is reached, then sends the robot to the second goal location and so on, until goal locations have been visited. While the robot is moving you should print out the status of the goal.
- If `move_base` is not able to reach some goal, you should print some warning message in the console and continue with the rest of the goals.