Exercise 3: The Turtlebot

Development of Intelligent Systems

2022

In this exercise you will familiarize yourself with the simulation of 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.

Unfortunately, the Turtlebot is not officialy supported in ROS Noetic, and in order to use it we have to manually install and manually build some packages.

1 Installing the Turtlebot packages

First, we need to install some Ubuntu packages that are either required by the packages used in the Turtlebot stack or we will use directly. You can install them with the following mega-command:

```
sudo apt install ros-noetic-ecl-exceptions ros-noetic-ecl-threads ros-noetic-ecl-geometry
    ros-noetic-kobuki-dock-drive ros-noetic-kobuki-driver ros-noetic-ecl-streams ros-
    noetic-position-controllers ros-noetic-effort-controllers ros-noetic-joint-trajectory
    -controller ros-noetic-costmap-2d ros-noetic-amcl ros-noetic-base-local-planner ros-
    noetic-carrot-planner ros-noetic-clear-costmap-recovery ros-noetic-dwa-local-planner
    ros-noetic-fake-localization ros-noetic-global-planner ros-noetic-map-server ros-
    noetic-move-base ros-noetic-move-base-msgs ros-noetic-move-slow-and-clear ros-noetic-
    nav-core ros-noetic-navfn ros-noetic-rotate-recovery ros-noetic-voxel-grid ros-noetic
    -slam-gmapping
```

For easier copying this command is available as a bash file (**ROS_packs.sh**) on učilnica. After we have all these packages installed download the **Turtlebot_packs.zip** file from učilnica into a warkspace and build them. These are ROS packages that are officially not available in ROS Noetic and we have to build them manually. After this, you can continue with the exercise.

2 Starting the simulation

ROS is integrated with the Gazebo simulator. Many times it is much more convenient to run a simulation of the Turtlebot robot instead of working on the real robot. For example, imagine there is a deadly virus, and you can not access the real robot. In Gazebo we need a model of the robot that we want to simulate, and a model of the world in which we simulate the robot. A model of the Turtlebot is already provided in the newly built Turtlebot packages. A simple turtlebot world representing a polygon in the classroom has been added inside the `worlds` folder in the exercise. It consists of a 3d model `.dae` file, as well as a folder containing graphics to be inluded in the model and a `.world` file

which Gazebo actually loads. You can create your own 3d models using any 3d modeling software (the provided file was made in Google SketchUp). There are a bit more detailed tutorials on the following links:

- Gazebo tutorials
- Importing a 3D model in Gazebo

To start a complete simulation of the Turtlebot robot in Gazebo, run the following command, after you have compiled the package for this exercise:

```
roslaunch exercise3 rins_world.launch
```

If this is the first time starting Gazebo it might take some time while Gazebo downloads the files that it needs from the internet. After this you should get a window with a 3D simulation of our robot in an environment containing a polygon.

Now we can use a lot of the existing functionalities just like we were using a real robot. We can use Rviz visualizer. In the package `turtlebot_rviz_launchers` we have already prepared Rviz visualizations of the robot. To see one, run the following command in a new terminal:

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

Try turning off and on different displays, and check what visualization options do the different displays offer. You should familiarize yourself with Rviz.

Let's now finally move the robot. You can find scripts for remote control of the Turtlebot using different devices in the `turtlebot_teleop` package. To control the robot with the keyboard run the following command:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

You should now be able the move the robot using the keys as shown in the terminal. Note that you can use the same script for moving the real robot, by simply redirecting the the commands to the correct topic. In fact, all that the script does is reading the keyboard presses and sending appropriate `Twist` messages.

3 Autonomous navigation

3.1 Map-building

Of course, our ultimate goal is to have the robot navigate autonomously. The first step towards this capability is to build a map of the working environment of the robot. For building a map we will use 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. Now close all the running programs except the Turtlebot simulation in Gazebo. Open a new terminal and run:

```
roslaunch exercise3 gmapping_simulation.launch
```

This will start the necessary nodes for building a map. The `gmapping_simulation.launch` file is built from the `gmapping_demo.launch` script from the `turtlebot_navigation` package which we use for the real robot. In `gmapping_simulation.launch` we do not start the nodes for the Kinect since we do not use a real Kinect sensor.

In order to view the map that we are building we can use Rviz and the prepared visualization in `turtlebot_rviz_launchers`:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

To build the map we should move the robot around the polygon, moving in straight lines and making 360° degree in place rotations. To move the robot use the same script as in the previous section:

```
roslaunch turtlebot_teleop keyboard.launch
```

Now move about the polygon until you get a relatively good map. To build a good map:

- 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.

Once you are satisfied with the map you can save it by executing:

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

Do not add any extensions to `<the_name_of_your_map>`. The `map_saver` will create two files for the map. The first one is a `.yaml` file containing the name of the image for the map, the origin of the image, the metric length of a pixel in the map and the thresholds for occupied and free space. The other file is a `.pgm` image file which you can open in any image editor. This is useful for fixing minor imperfections in the map.

If you have built a good enough map close all the running programs.

3.2 Navigation

If you have built a map of the polygon we are finally ready to let the robot drive itself. In one terminal start the simulation:

```
roslaunch exercise3 rins_world.launch
```

Next, we need to start the localization node from the Adaptive Monte Carlo Localization package, as well the navigation node that from the Move Base package that will actually drive the robot. You also need to specify the map file that you want to use-which would be the `.yaml` file generated in the previous section. You can do this in different ways,

right now, open the `amcl_simulation.launch` file and change the argument `map_file` to the complete path of your map. Now we can start the navigation by running:

```
roslaunch exercise3 amcl_simulation.launch
```

The `amcl_simulation.launch` file is built from the `amcl_demo.launch` script from the `turtlebot_navigation` package which we use for the real robot. Finally, to see what is going on and issue commands to the robot, start the visualization:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

In Rviz we can new see the robot as well as other information. In the beginning the position of the robot might be wrong. The initial position is the same position as when it began making the map. To initialize the robot in the right position you can use the "2D Pose Estimate" button in Rviz. Finally, if the position of the robot is relatively correct, use the "2D Nav Goal" button in Rviz to send the robot to a new position. Ta-da! We can use the same method for moving the real robot.

4 Sending movement goals from a node

In the `exercise3` package you have a C++ example 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`, write a script that 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 all 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.
- *Try to think of ways to automatically determine the goals that the robot should reach in order to explore the map and find objects in the map! This will be useful for the Tasks. You do not need to include this in the homework.