HAW HAMBURG
Fakultät TI, Department Informatik
Intelligente Sensorik
Prof. Dr. Tim Tiedemann, Max De Muirier

Übungsblatt 5
zur Veranstaltung „RobE – Einführung in die Robotik”

5. Navigation und Pfadplanung

In der Übung 5 geht es um Navigationsstrategien und Pfadplanung innerhalb bekannter und bereits kartierter Umgebungen.

Zur Bearbeitung der Übungsaufgaben: Bearbeiten Sie auf jeden Fall *alle* Übungsaufgaben. Ausgenommen hiervon sind lediglich die mit „optional” gekennzeichneten Textstellen. Lesen Sie sich die Aufgaben gut durch. Sollten Sie eine Aufgabe nicht lösen können, so beschreiben Sie zumindest, wie weit Sie gekommen sind und auf welche Weise Sie vorgegangen sind. Aufgaben mit der Randbemerkung *“Protokoll!”* sind im Protokoll zu beantworten. Als Lösung der Aufgaben ist (je nach Aufgabe) Programmcode abzugeben (kommentiert) oder ein kurzer Text.

5.1. Vorbereitungsaufgaben

1. Schauen Sie sich einmal die Beschreibungen zu den **ROS**-packages *AMCL*, *base_local_planner*, *dwa_local_planner* und *global_planner* an. Was wird zu den Algorithmen, zur Funktionsweise gesagt?

5.2. Praktikumsaufgaben

1. Bearbeiten Sie die Schritte **Setup for TurtleBot3 navigation** (Anhang A) und “Navigating within known territory” (Anhang B).
2. Welche Art Filter wird hier für die Lokalisierung verwendet? Beschreiben Sie, was sie sehen? Fahren Sie nun zur oberen rechten Ecke. Was ändert sich während der Fahrt? Wo sind die Parameter zu diesem Filter zu finden? *Protokoll!*
3. Beschreiben Sie, welche unterschiedlichen Elemente der Karte zu sehen sind. Überlegen Sie, welche markierten Areale für die globale und lokale Pfadplanung verwendet werden. *Protokoll!*
4. Setzen Sie nun die Zielposition so nah wie möglich an eine Wand. Welches Verhalten des Roboters ist zu beobachten und welche Strategie verwendet er, um sein Ziel zu erreichen? Überlegen Sie, worin hier ein Konflikt liegen könnte. *Protokoll!*
5. Ändern Sie nun die Toleranzwerte zur Zielerreichung (*goal tolerance*) in der Parameter-Datei *dwa_local_planner_params_burger.yaml* zu **0.1** für *x/y* und **0.5** für *yaw* und wiederholden Sie den vorherigen Schritt. Welches Verhalten ist nun zu beobachten? Wenn nötig, erhöhen Sie diese Werte noch weiter. *Protokoll!*
6. Bearbeiten Sie die Schritte zu **Modifying local and global costmaps** (Anhang C). Setzen Sie für die lokale Costmap die Parameter *cost_scaling_factor* auf **2** und *inflation_radius* auf **1**. Wie wirkt sich diese Änderung auf die Pfadplanung aus? Was passiert bei noch kleineren Werten? *Protokoll!*

7. Bearbeiten Sie die Schritte zu **Encountering an obstacle** (Anhang D). Fahren Sie an dem Hindernis vorbei. Welchen Einfluss nimmt das Hindernis auf die aktuelle und zukünftige Pfadplanung? *Protokoll!*
8. Bearbeiten Sie die Schritte zu **Path planning with A* and Dijkstra** (Anhang E).
9. **Optional**: Setzen Sie den von global planner verwendete Algorithmus auf **A*** und grid-basiert um. Ist ein Unterschied in der Pfadplanung zu erkennen?

A. Setup for TurtleBot3 navigation

In order to start with an equal base, a maze map is provided. Also included are *.world*, *.launch* and *model*-files. You can still use your previously generated map and model, if you like. Just copy them to the respective directories in the *robep5* package.

Please download and build the ROS-package *robep5* to *~/ros1_ws/src/*. It can be found in **Teams**. A *.bashrc*-file is included as well. Just add it to your existing *.bashrc* via

```
1 echo "source ~/ros1_ws/src/robep5/.bashrc" >> ~/.bashrc
2 source ~/.bashrc
```

The parameter files come in form of *yaml*-files and are very useful when wanting to pass parameters to the *rosparam* server. We will take a look into them later.

B. Navigating within known territory

Load the ROS setup file:

```
3 source ~/ros1_ws/devel/setup.bash
```

Define the TurtleBot3-model:

```
4 export TURTLEBOT3_MODEL=burger
```

Start the ROS master:

```
5 roscore
```

Start the Gazebo simulation with the provided maze world:

```
6 roslaunch robep5 turtlebot3_maze_navigation.launch
```

Now start the TurtleBot3-navigation with the provided maze map: (launch-file could be split in multiple)

```
7 roslaunch robep5 turtlebot3_navigation.launch
  map_file:=$HOME/ros1_ws/src/robep5/maps/map.yaml
```

NOTE: The *yaml*-file contains the absolute path to the map-image. It can be change to a relative path, such as *./map.pgm*.

Use the buttons **2D Pose Estimate** to provide an initial pose for the localization and **2D Nav Goal** to set the target position. The more exact the placement, the better will be the localization and path planning.

To change the resolution of the local costmap, edit the following parameter file:

```
 8 cd ~/ros1_ws/src/robep5/param/
 9 gedit local_costmap_params.yaml
```

Change it to *0.01*.

C. Modifying local and global costmaps

Costmaps assign certain costs to sections of an environment. These cost values increase the closer these parts are to walls and other obstacles. There are two types of costs: lethal and neutral costs. Lethal costs are assigned to walls and obstacles, neutral costs to free spaces. The inflation radius is used to create a limited passable buffer zone around walls and obstacles. Costs decrease along this radius. The cost scaling factor describes the ratio by which these costs decrease.

The *inflation radius* and *cost scaling factor* for the local and global costmap can be edited for the local costmap:

```
10 gedit costmaps_common_params_burger.yaml
```

and for the global costmap:

```
11 gedit global_costmap_params.yaml
```

Decreasing both will result in an “no-go”-Area, indicated by the red map section with high costs. Increasing the cost scaling factor results in an increasingly narrow area of high costs.

All accessible parameters can be modified on the go. For that, start **RQT GUI** with the tool with *reconfigure*

```
12 rosrun rqt_reconfigure rqt_reconfigure
```

and confirm your changes with **ENTER**. Changes will not be saved.

D. Encountering an obstacle

Since the robot has knowledge about its environment, it can plan its path though it using the map that was generated with SLAM. Dynamic or new obstacles are mapped

on the go and remembered during the navigation session.

A small example map shows, how a previously unknown obstacle is implemented into a known map. It then can be considered during path planning.

Start the Gazebo simulation with the provided maze world:

```
13 roslaunch robep5 turtlebot3_maze_navigation_obstacle.launch
```

Now start the navigation with the provided maze map:

```
14 roslaunch robep5 turtlebot3_navigation.launch
   map_file:=$HOME/ros1_ws/src/robep5/maps/map.yaml
```

You can continue to add another obstacle via the **toolbar** on the top and edit it with **right-click**. Save your obstacle and continue the simulation with the **Play**-button on the bottom of the window.

E. Path planning with A* and Dijkstra

The packages *base_local_planner* and *dwa_local_planner* are used to generate a global and local path using local and global costmaps. The path planning algorithm then uses these costmaps to find the best compromise between minimal costs and the shortest distance to the target position.

For more information, read the following paper: `https://arxiv.org/pdf/1706.09068.pdf`

The package *global_planner* provides an implementation of **A*** and **Dijkstra** for path planning. Its parameters can be edited in the parameter file.

To enable the global planner, uncomment the lines 12 and 13:

```
15 cd ~/ros1_ws/src/robep5/launch
16 gedit move_base.launch
```

Edit the global planner parameters:

```
17 cd ~/ros1_ws/src/robep5/param
18 gedit global_planner_params.yaml
```

Change the parameter *use_dijkstra* to false. This enables the A* algorithm for path planning. To enable grid-based path planning, change the parameter *use_grid* to true.

Now start the simulation again, as before, using:

```
19 roslaunch robep5 turtlebot3_maze_navigation.launch
20 roslaunch robep5 turtlebot3_navigation.launch
   map_file:=$HOME/ros1_ws/src/robep5/maps/map.yaml
```