UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Robotic Games

Sommersemester 2020

holger.dieterich@ziti.uni-heidelberg.de

3. Aufgabenblatt

vom 06.05.2020
Lernziele: Kollisionsvermeidung, Homing
Besprechung der Hausaufgaben am 13.05.2020

3.1) launch-File

In moodle wurde in einem zip-File ein ros-package names *exercise_3* zur Verfügung gestellt. Bitte fügt es in den Source-Folder in Eurem catkin-Workspace ein. Im Ordner *scripts* befindet sich ein File names *collision_avoidance.py*. Es enthält ein Code-Gerüst für Aufgabe 3.2). Bitte beachtet die entsprechenden Zugriffsrechte, um es ausführen zu können:

```
$ cd ~/catkin_ws/src/exercise_3/scripts
$ chmod +x collisison_avoidance.py
```

Schreibt ein launch-File names *coll_avoid.launch* mit folgendem Inhalt:

- Es nimmt eine Zahl (1 ...5) als Argument entgegen (default = 1).
- Es ruft *turtlebot3_worldX.launch* im Package *exercise_3* auf. *X* steht im Folgenden für die Zahlen, die als Argument übergeben wird. Diese sind die Nummerierung der Worlds, in denen der Roboter Kollisisonen vermeiden soll.
- Es startet das Kollisionsvermeidungs- Skript *collision_avoidance.py* (zunächst noch ohne Funktion).
- Es zeichnet in einem rosbag das */odom*-Topic auf und speichert diese unter *~/.ros/odomX.bag*

`http://wiki.ros.org/roslaunch`

Hausaufgabe: Lade das launch-File in moodle hoch.

3.2) Kollisionsvermeidung

Ziel dieser Aufgabe ist es, ein Verfahren zu implementieren, das Kollisionen des Turtlebots mit der statischen Umgebung (Wände etc.) vermeidet. Die Lineargeschwindigkeit v_x des Roboters soll möglichst nicht unter $0{,}1 m/s$ fallen, d.h. der Roboter soll fortlaufend vowärtsfahren und nur im äußersten Notfall bremsen. Um Abstände zu den Hindernissen zu bestimmen, werden die Abstandsmessungen aus dem Laser-Scanner verwendet.
Ein Gerüst hierfür wird im Package *exercise_3.py* (siehe moodle) bereitgestellt. Im Ordner *scripts* befindet sich die Datei *collision_avoidance.py*. Bitte versucht zunächst den Aufbau des Skripts zu verstehen.

Die Funktion *calculate_force* in *collision_avoidance.py* soll eine 'Kraft' berechnen (siehe Vorlesung), die den Roboter daran hindert mit der Unwelt zu kollidieren. Hinweis: Triviale Lösungen (bspw. Kreisfahrt) werden nicht als Kollisionsvermeidung anerkannt. Die Rotationsgeschwindigkeit soll sinnvoll in Abhängigkeit der Laser-Scan-Messungen angepasst werden.

Hausaufgabe: Beschreibe den implementieren Ansatz zur Kollisisonsvermeidung in einem PDF anhand von mathematischen Formeln. Lade das PDF, undd das angepasste Skript *collision_avoidance.py* in moodle hoch.

Hausaufgabe: Verwende das launch-File aus Aufgabe 3.1 um je einen rosbag für die fünf Welten zu erzeugen. Erzeuge aus den rosbags und mit Hilfe des Skripts *create_visualization.py* für jede der fünf Welten ein Bild des Pfades. Lade die Bilder in moodle hoch.

3.3) Homing

Ziel des Homings ist es, den Roboter zu einem Zielpunkt fahren zu lassen. Starte zunähst die Turtlebot-Gazebo-Simulation in einer leeren Welt ohne Hindernisse.

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

Im Order scripts ind exercise_3 befindet sich das goal_publisher.py. Beachte auch hier wieder die Zugriffsrechte. Starte das Skript mit dem rosrun-Befehl.

```
$ rosrun exercise_3 goal_publisher.py
```

Unsere Ziel-Position wird jetzt unter dem Topic */goal_pose* ausgegeben. Standartgemäß fährt der Roboter dabei die Position [5,5] an. Zu Testzwecken ist jedoch Ratsam verschiedene Ziele zu Probieren, optimalerweise eines Pro Quadrant relativ zum roboter. Hierzu kann mit

```
$ srun rviz rviz -d ‘rospack find turtlebot3_description‘/rviz/model.rviz
```

Rviz Gestartet werden. dabei findet sich in der Oberen Leiste ein grüner Pfeil mit der Beschriftung *2D Nav Goal*. Dieses Tool lässt sich durch linksklick mit der Maus auswählen. Erst ausgewählt ermöglicht es mit linksklick auf dem Haptbildschirm ein Ziel relativ zum Roboter festzulegen.

Sowohl das Ziel als auch die Roboter Position sind bis jetzt im Bodenkoordinatensystem gegeben.

Hausaufgabe: Rechne das Ziel in Roboter-bezogene Polarkoordinaten $[\rho, \alpha]$ (siehe Abbildung 1) um. Lade deinen Rechenweg und die Lösung entweder als gescannte PDF oder über Latex erstellte PDF hoch.

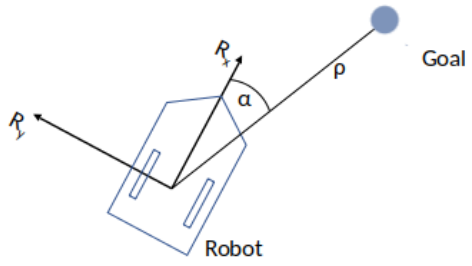

Abbildung 1: Skizze Polarkoordinaten

Die Positionsregelung kann dann durch negative, proportionale Rückführung von ρ und α auf die Sollgeschwindigkeit (v_x und ω_z) erfolgen.

$$v_x = K_v \rho$$

$$\omega_z = K_\omega \alpha$$

Der Wert K muss dabei selbst gewählt werden. Schreibe hierzu ein Skript *homing.py*, das die Topics der Zielposition (*/goal_pose*) und der Istposition (*/odom/pose/pose*) einliest und auf die Sollgeschwindigkeit (*cmd_vel*) schreibt. Teste das Programm für verschiedene Sollpositionen.
Tipp: Das einlesen und verarbeiten von mehreren Werten wird in der Übungsgruppe beispielhaft vorgeführt.

Hausaufgabe: Lade das Funktionierende Skript unter dem namen *homig.py* in moodle hoch. Erkläre wie du zur Wahl deiner $_v$, K_ω gekommen bist