Praktikum zu reaktiver Navigation

Christian Pfitzner

June 5, 2018

1 Voraussetzungen

- Grundlagen C++
- Grundlagen in ROS
- Kentnisse über reaktive Navigation aus der Vorlesung

2 Lernziele

- Vertiefung der Kentnisse in ROS
- Kennenlernen der reaktiven Navigation
- Kennenlernen des Simulators STDR in ROS

3 Aufgabe im Praktium

Sie verwenden einen Roboter in der Simulationsumgebung STDR. Dieser Roboter übernimmt Geschwindigkeitsbefehle und liefert Sensordaten (hier einen Laserscanner) über die ROS-üblichen Schnittstellen. Ihre Aufgabe ist es, einen ROS-Knoten zu schreiben, der den Roboter kollisionsfrei durch seine Umbegung navigiert.

Aufgabe 1.
Machen Sie sich mit der ROS-Nachricht für Laserscanner vertraut `http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html`

Bei Unklarheiten, zögern Sie bitte nicht ihren Praktikumsbetreuer zu fragen.

Template für die Aufgabe

Sie erhalten für das Praktikum eine vorgefertigtes ROS-Paket, welches Sie mittels `git` und dem Befehl `git clone https://github.com/christianpfitzner/reactive_navigation.git` in den Catkin workspace herunterladen können. Mittels des Befehls `catkin_make` können Sie das Paket bauen.

Sie starten den ROS-Knoten und die Simulation mittels des Befehls `roslaunch reactive_navigation reactive_laser_simulation.launch`. Ein Kreis stellt den Roboter dar; ein Strich zeigt die aktuelle Orientierung. Da der zu ergänzende ROS-Knoten noch keine Werte an den Roboter schickt, steht dieser still. Auch die Sensordaten des Laserscanners werden in der Simulation als rote Strahlen angezeigt.

Aufgabe 2.
Untersuchen Sie die gestarteten ROS-Knoten mittels der Befehle `rosnode` und `rostopic`. Welche Schnittstellen zum Steuern des Roboters bietet die Simmulation? Über welches topic können Sie die Daten des simulierten Laserscanners abfragen? Untersuchen Sie zum Beispiel, wie häufig das topic für den Laserscanner gesendet wird.

Schauen Sie sich das Template an (`src/reactive_navigation_node.cpp`): Dieses enthält eine Main-Funktion, eine Callback-Funktion zum Empfangen der Laserdaten, sowie weiterer Funktionen, die für die Berechnung der reaktiven Navigation notwendig sind.

Aufgabe 3.
Tragen Sie die gefundenen `topic`-Namen für die Subscriber und Publisher in der Main-Funktion ein. Nutzen Sie dazu die Variablen `vel_topic` und `laser_topic`. Bauen Sie das Paket und starten Sie nochmal das launch-File. Der Roboter sollte nun geradeaus fahren, wenn der Publisher korrekt verknüpft ist.

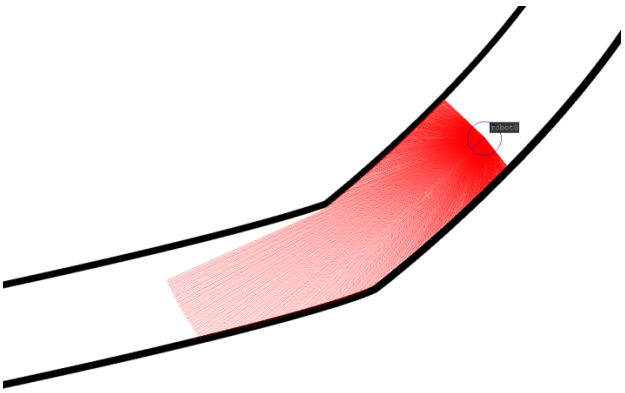

Figure 1: Roboter in der STDR-Simulationsumgebung

Beginnen Sie nun die reaktive Navigation zu implementieren. Nutzen Sie für die Änderung der Orientierung die Gleichung

$$\alpha = \arctan \frac{\sum \sin(\theta_i) f_1(\theta_1) f_2(r_i)}{\sum \cos(\theta_i) f_1(\theta_1) f_2(r_i)} \tag{1}$$

wobei θ_i der Winkel eines einzelnen i-ten Laserstrahls ist und r_i die gemessene Entfernung für einen Laserstrahl ist. Die beiden Funktionen $f_1(\cdot)$ und $f_2(\cdot)$ werden wie folgt eingesetzt:
Die Funktion $f_1(\cdot)$ gewichtet, ob sich ein Strahl des Laserscanners in Fahrtrichtung befindet. Hindernisse in Fahrtrichtung sollen somit höher gewichtet werden.
Die Funktion $f_2(\cdot)$ gewichtet dagegen die Distanz zu einem Hindernis. Nahe Hindernisse sollen höher Gewichtet werden, als Hindernisse mit einer größeren Entfernung. ja

Aufgabe 4.
Implementieren Sie die beiden Funktionen mit festen Schwellwerten; die Funktionen sollen entweder den Wert 0.0 oder 1.0 zurückliefern. Versuchen Sie ein möglichst gutes Verhalten zu erzielen.

Mit fest eingestellten Schwellwerten können Sie schnell zu einem Ergebnis kommen. Besser ist jedoch die Gewichtung über eine kontinuierliche Funktion.

Aufgabe 5.
Ändern Sie die beiden Funktionen ab. Der Winkel soll über

$$f_1(\theta_i) = 0.5 \cos(2\theta_i) + 1 \tag{2}$$

bestimmt werden, währen die Distanz mittels

$$f_2(r_i) = \frac{1}{1 + e^{-2 \cdot r_i + 2}} \tag{3}$$

gewichtet werden soll. Beide Funktionien liefern Werte zwischen 0 und 1 zurück. Wie ändert sich das Verhalten des Roboters?

Testen Sie die Lösung mit einer festen linearen Geschwindigkeit von 1.0 m/s. Ihr Roboter wir wahrscheinlich nach kurzer Zeit die Wand berühren.

Aufgabe 6.
Steuern Sie die Geschwindigkeit in Abhängigkeit der geringsten Distanz zu einem Hindernis. Beachten Sie, dass Hindernisse an der Seite des Roboters die Geschwindigkeit weniger beeinflussen sollten, als Hindernisse in Fahrtrichtung. Nutzen Sie hierfür die Funktion `isInDrivingWay`, die ausgeben soll, ob ein Punkt in Fahrtrichtung liegt, oder nicht. MerkenSie sich den Punkt mit der geringsten Entferung zum Roboter über alle Laserstrahlen. Nutzen Sie diesen Wert zum Regeln der Geschwindigkeit. In welcher Zeit schafft ihr Roboter nun eine Runder im Parcours?

Aufgabe 7.
Zeigen Sie ihrem Praktikumsbetreuer das Ergebnis Ihrer Navigation.
