Ausgabe am **12.04.2021**
Abgabe bis zum **25.04.2021 23:59 Uhr**

UNIVERSITÄT OSNABRÜCK

Priv. Doz. Dr. Thomas Wiemann, Alexander Mock

Robotik
Übungsblatt 1

Sommersemester 2021

Aufgabe 1.1 (Abgaben) (1P)

Die Abgaben der Übungsblätter erfolgt in 5er Gruppen. Erstellen Sie für jedes Übungsblatt ein neues ROS-Paket mit dem Namen des Blattes gefolgt von Ihrer Gruppe. Beispiel: Wenn Ihre Gruppe `Gruppe_1` heißt, wäre der Name für das aktuelle Übungsblatt: `blatt01_gruppe01`. Achten Sie darauf, dass nicht nur der Ordner so heißt sondern auch das Paket (`package.xml`, `CMakeLists.txt`). Andernfalls würde es mehrdeutige Paketnamen innerhalb unseres ROS-Systems geben.

Legen Sie alle handschriftlichen Lösungen mit in das Paket. Als Abgabe laden Sie dieses ROS-Paket dann als ZIP-Archiv verpackt auf StudIP hoch. Achten Sie darauf, dass Sie die Fristen einhalten. Diese sind verbindlich. Für eine Klausurzulassung muss Ihre Gruppe in jedem Übungsblatt mindestens 50% der Punkte erreichen.

Weitere Anmerkungen:

- Alles was in Ihrer Abgabe liegt zählt. Wenn eine Frage mit Ihrer Abgabe nicht beantwortet werden kann, gibt es dafür keine Punkte.
- Ergebnisse, die wir reproduzieren können, müssen nicht mit abgegeben werden. Beispiel: Ausgaben auf der Kommandozeile. Das gilt nicht für Aufgaben, in denen dies explizit widersprochen wird.

Aufgabe 1.2 (Installation von ROS) (4P)

(a) Installieren Sie Ubuntu 20.04 auf ihrem Computer, welches Sie kostenlos auf `https://www.ubuntu.com` herunterladen können.

(b) Informieren Sie sich über die Robotersoftware ROS auf der Webseite `http://www.ros.org` sowie im zugehörigen Wiki `http://wiki.ros.org`.

(c) Installieren Sie die Robotersoftware ROS in der Version *noetic*, wie in `http://wiki.ros.org/noetic/Installation/Ubuntu` beschrieben.

(d) Bearbeiten Sie **mindestens** die Tutorials 1 - 6 und 11, 13 auf der Website `http://wiki.ros.org/ROS/Tutorials`. Die Tutorials helfen Ihnen bei der Einrichtung der Arbeitsumgebung und führen Sie in die grundlegenden Tools und Konzepte von ROS ein. Achten Sie bei jedem der Tutorials darauf, dass der Reiter `catkin` ausgewählt ist.

(e) In dieser Teilaufgabe werden Sie Ihren ersten Subscriber und Publisher implementieren.

Paket

Erstellen Sie dazu mit `catkin_create_pkg` ein Paket benannt nach Ihrer Gruppe wie in Aufgabe 1.1 erklärt. Die Abhängigkeiten Ihres Paketes sind lediglich `roscpp` und `rospy`.

Quelltext

Erstellen Sie innerhalb dieses Pakets die Dateien `src/talker.cpp` und `src/listener.cpp`. Übernehmen Sie den Quelltext aus dem Tutorial 11. Achten Sie darauf, dass Sie immer die C++-Variante eines Tutorials verwenden.

Kompilierung

Die Kompilierung eines Quelltexts zu einer ausführbaren ROS-Node geschieht mit `CMake`. Ergänzen Sie dazu die `CMakeLists.txt` um die folgenden Zeilen:

```
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
```

Ausführung

Starten Sie Ihre Nodes wie in Tutorial 13 beschrieben. Ersetzen Sie `beginner_tutorials` mit Ihrem Paketnamen. (4P)

Aufgabe 1.3 (Visualisierung von Sensordaten mit `rviz`) (2P)

Vermutlich steht Ihnen Zuhause kein ROS-fähiger Roboter zur Verfügung, um praktische Aufgaben zu bearbeiten. Sie können daher eine aufgezeichnete Roboterfahrt (*rosbag*) verwenden oder – auf späteren Übungsblättern – den Roboter in einer Simulationsumgebung betreiben.

Für dieses Übungsblatt steht Ihnen eine *rosbag*-Datei `ceres.bag` des Roboters *ceres* (siehe Abbildung 1) mit aufgezeichneten Sensordaten in Stud.IP zur Verfügung.

Um ein Bagfile abspielen zu können, müssen Sie ROS zunächst starten:

```
$ roscore
```

Viele Messages besitzen einen Zeitstempel. Da Sie eine Aufzeichnung verwenden, müssen Sie ROS mitteilen, dass die Zeitstempel möglicherweise weit in der Vergangenheit liegen:

```
$ rosparam set use_sim_time true
```

Nun können Sie das Bagfile ins System einspielen:

```
$ rosbag play --loop --clock <bagfile>
```

Das Argument `--loop` sorgt dafür, dass die Daten in einer Endlosschleife abgespielt werden, d.h. sobald das Ende der Aufzeichnung erreicht ist, wird wieder mit dem Anfang begonnen. Argument `--clock` sorgt dafür, dass die im Bagfile aufgezeichneten Zeitstempel ebenfalls veröffentlich werden.

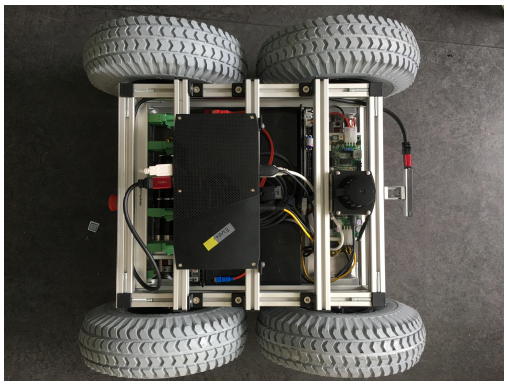

Abbildung 1: Ein *ceres*-Roboter

Weitere Informationen zu Bagfiles finden sie unter `http://wiki.ros.org/Bags` und `http://wiki.ros.org/rosbag`.

Die Sensordaten können nun mit dem Programm `rviz` visualisiert werden.

(a) Machen Sie sich mit der Bedienung von `rviz` vertraut.

(b) Visualisieren Sie das Laserscan-Topic `/scan` mit `rviz`. Als Fixed Frame müssen Sie dazu `/base_link` einstellen.

(c) Stellen Sie als Fixed Frame nun `/odom_combined` ein. Erhöhen Sie die Decay Time des Laserscan-Topics in `rviz` auf 30 (s). Was fällt Ihnen im Vergleich mit Aufgabenteil (b) auf?

Aufgabe 1.4 (Sensordaten) **(7P)**

Wie Sie in den Tutorials bereits erfahren haben, funktioniert ROS nach dem Publisher / Subscriber-Prinzip, wobei *Messages* wie z.B. Sensorwerte zwischen verschiedenen Programmen (*Nodes*) ausgetauscht werden. In dieser Aufgabe werden Sie Ihre erste (sinnvolle) Node schreiben um die Daten einer *Inertial Measurement Unit* (IMU) auszulesen und weiterzuverarbeiten.

Starten Sie das Bagfile `imu.bag` aus dem Stud.IP mittels:

```
$ rosparam set use_sim_time true
$ rosbag play --loop --clock <pfad/zur/imu.bag>
```

(a) Versuchen nun Sie auf der Kommandozeile etwas mehr über die Topics der Bagfile zu erfahren. Mit welchem Befehl

- lassen sich die vorhandenen Topics anzeigen?
- lassen sich die Messages eines Topics auf die Kommandozeile schreiben?
- lässt sich die Publizierungs-Rate eines Topics ermitteln?
- lässt sich der Message Typ eines Topics ermitteln?

(2P)

(b) Wenn noch nicht vorhanden, erstellen Sie ein neues Paket namens `blatt01`. Legen Sie in diesem Paket eine Node namens `imu_print` mit der Quelltext-Datei `imu_print.cpp` an. Diese Node soll sich lediglich auf das `/imu`-Topic subscriben und im Callback die Message über `ROS_INFO` auf die Kommandozeile schreiben.

(c) Erstellen Sie nun eine Node namens `imu_mean` mit der Quelltext-Datei `imu_mean.cpp`. Diese Node soll sich auf das Topic `/android/imu` subscriben. Ermitteln Sie im Callback den Mittelwert $\bar{X}$ und die Varianz V der linearen Beschleunigung seperat für alle 3-Achsen. Nutzen Sie für die Berechnung den Algorithmus für auflaufende Messwerte wie in Wikipedia [1] beschrieben:

$$\bar{X}_n = \frac{n-1}{n}\bar{X}_{n-1} + \frac{1}{n}X_n$$

$$V_n = \frac{n-1}{n}\left(V_{n-1} + \bar{X}_{n-1}^2\right) + \frac{1}{n}X_n^2 - \bar{X}_n^2$$

Lassen Sie die Formeln auf sich wirken. Erklären Sie,

- warum sich der Algorithmus für die vorliegende Fragestellung eher eignet als die klassische Berechnung des Mittelwerts und der Varianz.
- warum in dieser rekursiven Formel $\bar{X}_1$ und V_1 berechnet werden kann, ohne dass Sie $\bar{X}_0$ und V_0 kennen.

Implementieren Sie nun die Formeln in Ihrem Callback. Geben Sie für jedes einkommende Sensordatum den aktuellen Mittelwert und die Varianz auf der Kommandozeile aus. (5P)

Hinweis: Für die aktuelle Aufgabe müssen Sie gegebenfalls Abhängigkeiten in Ihre `CMakeLists.txt` und `package.xml` eintragen.

[1] `https://de.wikipedia.org/wiki/Stichprobenvarianz`

Aufgabe 1.5 (Transformationen) **(6P)**

Sensordaten (und alles andere) wurden immer an einem gewissen Ort zu einer gewissen Zeit aufgenommen. Wie Sie schon erfahren haben, sind `topics` gut für den Austausch von Sensordaten geeignet. Für die Definition von Orten bzw. Koordinatensystemen oder Bezugssystemen bietet ROS `tf` an. Für die nachfolgenden Teilaufgaben helfen die ROS-Tutorials `http://wiki.ros.org/tf2/Tutorials`.

(a) Ihr Robotersystem besteht diesmal aus einer Basis (`base`), einem Sensor (`sensor`) und einem Greifer `gripper`. Schreiben Sie eine Node `robot_tf`, welche die Koordinatensysteme innerhalb von ROS über `tf2` definiert und broadcastet.

Setzen Sie dabei die folgenden Aussagen um:

- Das gemeinsame Bezugssystem des Greifers und Sensors ist die Basis.
- Der Greifer ist um 0.3 Meter nach vorne verschoben und zeigt nach vorne
- Der Sensor ist um 0.5 Meter nach oben verschoben und zeigt um 45 Grad nach unten.

Führen Sie Ihre Node aus und visualisieren Sie den TF-Baum über RViz. Visualisieren Sie danach den TF-Baum über die Node `rqt_tf_tree` aus dem gleichnamigen Paket `rqt_tf_tree`. Machen Sie von beiden Visualisierungen einen Screenshot und legen Sie diesen mit zu Ihren Abgaben. (2P)

(b) Ihr Sensor kann Punkte detektieren. Schreiben Sie eine Node `sensor_node`, welche 10 mal pro Sekunde eine Message vom Typ `geometry_msgs/PointStamped` auf ein neues Topic publisht. Der Sensor kann den Punkt nur auf seiner x-Achse messen. Er liegt 0.7071 m vor ihm. Achten Sie darauf, dass der Punkt das Bezugssystem `sensor` erhält. Visualisieren Sie sich die Sensordaten mit RViz. Erklären Sie, inwiefern sich `geometry_msgs/PointStamped` von `geometry_msgs/Point` unterscheidet und wie dies mit dem TF-System zusammenhängt. (2P)

(c) Sie wollen wissen, wie weit der vom Sensor detektierte Punkt von Ihrem Greifer entfernt ist. Schreiben Sie dazu eine Node `gripper_distance`. Diese soll sich auf das Sensordaten-Topic subscriben. Transformieren Sie den Punkt im Callback zunächst in das `gripper`-Koordinatensystem. Nutzen Sie dazu die Funktionalitäten von `tf2`. Insbesondere ein `tf2`-listener wird hier benötigt. Berechnen Sie anschließend die Distanz vom transformierten Punkt zum Greifer-Ursprung und geben Sie das Ergebnis auf der Kommandozeile aus. (2P)

Hinweis: Für die aktuelle Aufgabe müssen Sie gegebenfalls Abhängigkeiten in Ihre `CMakeLists.txt` und `package.xml` eintragen.