Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Bachelorarbeit

Titel der Arbeit // Title of Thesis

Retrofitting der Hard- und Software der Kuka-youBots auf die aktuelle Linux-Version und das Robot Operating System (ROS - Kinetic), Autonome Navigation zum Werkstückstransport von/zu einer Fertigungs-Taktstraße unter Nutzung des Laserscanners und des Adaptive Monte Carlo Localization - Algorithmus und Matlab.

Retrofitting hard- and software of the Kuka-youBots to the current Linux version and Robot Operating System (ROS - Kinetic), Autonomous Navigation for work piece transport from/to a manufacturing line using the laserscanner and the Adaptive Monte Carlo Localization - Algorithmn and Matlab.

Akademischer Abschlussgrad: Grad, Fachrichtung (Abkürzung) // Degree

Bachelor of Engineering B.Eng.

Autorenname, Geburtsort // Name, Place of Birth

Flores Sebastian, Bocholt

Studiengang // Course of Study

Mechatronik // Mechatronics

Fachbereich // Department

Maschinenbau // Engineering

Erstprüfer // First Examiner

Prof. Dr.-Ing. Olaf Just

Zweitprüfer // Second Examiner

Prof. Dr.-Ing. Horst Toonen

Abgabedatum // Date of Submission

16. November 2017

Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Eidesstattliche Versicherung

Flores, Sebastian

Name, Vorname // Name, First Name

Ich versichere hiermit an Eides statt, dasss ich die vorliegende Abschlussarbeit mit dem Titel

„Retrofitting der Hard- und Software der Kuka-youBots auf die aktuelle Linux-Version und das Robot Operating System (ROS - Kinetic), Autonome Navigation zum Werkstückstransport von/zu einer Fertigungs-Taktstraße unter Nutzung des Laserscanners und des Adaptive Monte Carlo Localization - Algorithmus und Matlab.“

„Retrofitting hard- and software of the Kuka-youBots to the current Linux version and Robot Operating System (ROS - Kinetic), Autonomous Navigation for work piece transport from/to a manufacturing line using the laserscanner and the Adaptive Monte Carlo Localization - Algorithmn and Matlab.“

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum, Unterschrift // Place, Date, Signature

Inhaltsverzeichnis

Abkürzungsverzeichnis

AMCL Adaptive Monte Carlo Localization

Bash Bourne again Shell

BSD Berkeley Software Distribution

eog Eye of Gnome

EOL End of Live

IWK Industrie Werke Karlsruhe

KUKA Keller und Knappich Augsburg

LTS Long Term Support

MATLAB Matrix Laboratory

MCL Monte Carlo Localization

NUC Next Unit of Computing

ROS Robot Operating System

Abbildungsverzeichnis

1. Einleitung

> "Waren Roboter vor einigen Jahrzehnten noch bloße Science-Fiction, sind sie aus dem Leben heute kaum mehr wegzudenken. Sie bauen Autos, entschärfen Bomben und tauchen in die Tiefen der Ozeane. Auch die Raumfahrt ist bei ihren Missionen auf die Unterstützung von Robotern angewiesen. Doch bevor das erste Roboterfahrzeug auf dem Mars herumfahren konnte, mussten Forscher erst viele Jahre Entwicklungsarbeit leisten."[LEZ16]

1.1. Problemstellung

Die Westfälische Fachhochschule Gelsenkirchen Bocholt, Fachbereich 6, besitzt seit 2015 vier KUKA youBots. Diese dienen der Lehre und Forschung. Die youBots besitzen von Haus aus einen Intel®Atom als internen Computer. Das Ausführen rechenaufwendiger Programme muss ein zweiter externer Rechner übernehmen, da der interne PC zu langsam ist. Des Weiteren ist es nicht möglich mehrere Programme gleichzeitig rechnen zu lassen.

1.2. Zielsetzung

Bei einem der vier youBots soll der interne Intel®Atom Computer gegen einen neuen Intel®NUC ausgetauscht werden. Dies ist möglich, da die Universität Magdeburg den Austausch des internen Computers schon bewerkstelligt hat, "Wir haben einen Intel NUC mit Core i7-Prozessor der 5. Generation verbaut"[Sei]. Ein Retrofitting der Software wurde noch nicht durchgeführt, "Demnächst soll auf Ubuntu 16.04 und ROS Kinetic umgestellt werden."[Sei]. Dies soll in dieser Arbeit zusätzlich umgesetzt werden. Das System soll unter Ubuntu 16.04.2 und ROS-Kinetic laufen. Die youBot Software muss manuell an die ROS-Kinetic Software angepasst werden, da diese noch keinen youBot Support besitzt.
Mit der auf den neusten Stand gebrachten Hard- und Software, soll der youBot autonom von und zu einer Taktstraße verfahren. Dies soll mit Hilfe des AMCL- Algorithmus und

der Zielvorgabe durch MATLAB geschehen.

1.3. Vorgehensweise

Die Umsetzung zum Erreichen des Zieles soll wie folgt aussehen:

Als erstes soll der youBot soweit auseinander gebaut werden, dass es möglich ist den Computer auszutauschen. Wenn der youBot offen ist, soll die Schnittstelle des youBots zum Intel®Atom Computer analysiert werden, um den Einbau des Intel®NUC Computers zu ermöglichen. Mit Anpassung der Schnittstellen muss gerechnet werden.

Nachdem der neue PC eingebaut wurde und der youBot wieder zusammengebaut ist, soll die Anpassung der Software durchgeführt werden.

Das danach durchzuführende autonome Navigieren ist gleichzeitig ein Test ob die, auf den neusten Stand gebrachte, Hard- und Software richtig umgesetzt wurde.

2. Grundlagen

2.1. Linux Ubuntu

Das Betriebssystem Ubuntu wurde von der Firma Canonical entwickelt und basiert auf dem Projekt „Debian/GNU Linux“. Linux ist lediglich der Kernel des Betriebssystems, der als Vermittler zwischen Hardware und Software fungiert. Ubuntu ist eine Linux-Umgebung die es Entwicklern erlaubt, möglichst einfach, Software zu programmieren. Ubuntu bedeutet, Menschlichkeit gegenüber Anderen, und kommt aus der afrikanischen Sprache. [vgl. ubu]

Abb. 2.1.: Ubuntu-Logo

2.2. Robot Operating System

ROS ist eine Software Umgebung die es ermöglicht Roboter zu programmieren und wurde unter der Open Source BSD Lizenz veröffentlicht. Sie ist ähnlich aufgebaut wie das Linux System. Oft wird sie auch auf einem Linux basierten Rechner installiert. Diese Software erleichtert das Empfangen und Versenden von Daten der einzelnden Komponenten des Roboters. Dies wird alles über Topics und Nodes geregelt die als Vernetzungen fungieren. Als Beispiel versendet der Laserscanner die Daten der Umgebung an den Roboter, beziehungsweise die Lokalisation des Roboters und vergleicht diese Daten mit den intern gespeicherten Daten. Die neuste ROS-Version ist Lunar Loggerhead, sie wurde am 23 März 2017 veröffentlicht, erreicht das EOL (End of Live) Datum im Mai 2019. Des Weiteren besitzt sie auch keinen Long Term Support. Aus diesem Grund wurde auf dem Computer des youBots, Ros Kinetic Kame installiert. Diese Version wurde am 23 März 2016 veröffentlicht und hat erst im April 2021 sein EOL Datum erreicht. [vgl. Shab]

Abb. 2.2.:
ROS-Kinetic-Logo

2.3. Roboter und Firmen

2.3.1. KUKA

Abb. 2.3.: KUKA-Logo

Die Firma KUKA wurde 1898 gegründet von Johann Josef Keller und Jakob Knappich, es war ein Acetylenwerk für Beleuchtungen. Nach der Fusion im jahre 1970 mit der Firma IWK schreibt KUKA im Jahr 1973 Geschichte, als Robotik-Pionier mittels dem entwickelten FAMULUS. Dieser ist der erste Industrieroboter, der Welt, mit sechs elektromechanisch angetriebenen Achsen. Die offene PC-basierte Steuerung wird seit 1996 von KUKA, als erster Roboterhersteller, verwendet.[vgl. KUKa]

2.3.2. Locomotec

Gegründet wurde die Locomotec GmbH am 08.03.2010. Die Firma ist spezialisiert auf die Entwicklung und Vermarktung von Technologien im Bereich Mobilität. Für den KUKA youBot ist sie, seit Juli 2010, exklusiver Vertriebspartner der KUKA Laboratories GmbH. Daraus wurde im Juli 2011 die Tochtergesllschaft youBot Store GmbH gegründet. [vgl. Loca]

Abb. 2.4.: Locomotec-Logo

2.3.3. youBot Store

Abb. 2.5.: youBot-Store-Logo

Die youBot Store GmbH ist eine Tochtergesellschaft der Locomotec und wurde am 04.07.2011 gegründet. Die Aufgabe des Unternehmens ist der Vertrieb und die Weiterentwicklung des KUKA youBots, mit Hilfe der KUKA Laboratories GmbH. Die Internetseite des Unternehmens, `www.youBot-store.com`, dient zum Verkauf für Zusatzkomponenten wie Sensoren oder Travel Cases, aber auch als Plattform für open source software und Lehrmaterial. [vgl. Locb]

2.3.4. youBot Roboter

Es gibt den youBot in verschiedenen Ausführungen, in dieser Arbeit findet lediglich die Basis Verwendung. Sie wird in Kapitel 3.1.1 noch detaillierter erklärt. In Abbildung 2.6 sind vier Verschiedene Varianten des youBots dargestellt. Diese Varianten sind jeweils eigenständig programmierbar.

Abb. 2.6.: Verschiedene Varianten des youBots

2.4. MATLAB

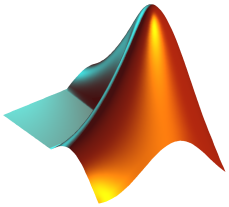


Abb. 2.7.: MATLAB-Logo

"Millions of engineers and scientists worldwide use MATLAB® to analyze and design the systems and products transforming our world. The matrix-based MATLAB language is the world's most natural way to express computational mathematics. Built-in graphics make it easy to visualize and gain insights from data. The desktop environment invites experimentation, exploration, and discovery. These MATLAB tools and capabilities are all rigorously tested and designed to work together.

MATLAB helps you take your ideas beyond the desktop. You can run your analyses on larger data sets, and scale up to clusters and clouds. MATLAB code can be integrated with other languages, enabling you to deploy algorithms and applications within web, enterprise, and production systems."[Matb]

2.5. Monte Carlo Localization- Algorithmus

"Mobile robot localization is the problem of determining a robot's pose from sensor data. Monte Carlo Localization is a family of algorithms for localization based on particle filters, which are approximate Bayes filters that use random samples for posterior estimation. Recently, they have been applied with great success for robot localization. Unfortunately, regular particle filters perform poorly in certain situations. Mixture-MCL, the algorithm described here, overcomes these problems by using a "dual" sampler, integrating two complimentary ways of generating samples in the estimation. To apply this algorithm for mobile robot localization, a kd-tree is learned from data that permits fast dual sampling. Systematic empirical results obtained using data collected in crowded public places illustrate superior performance, robustness, and efficiency, when compared to other state-of-the-art localization algorithms."[TFBD01]

3. Benötigte Hard- und Software

In diesem Kapitel werden die benötigten Hard- und Softwarekomponenten aufgelistet und auf das Wichtigste dieser wird eingegangen.

3.1. Hardware

Die Benötigten Komponenten für das Retrofitting der Hardware sind wie folgt geordnet:

- youBot Basis
- Laserscanner
- NUC PC-Kit
- Schnittstelle NUC-youBot
- Arena (Umgebung)

3.1.1. youBot Basis

Abb. 3.1.: Omnidirektionale mobile Plattform

Die youBot Basis ist mit omnidirektionalen Reifen ausgestattet und kann sich horizontal in eine beliebige Richtung bewegen. Der youBot ist ideal für die Lehre, Entwicklung und Forschung, speziell für die Logistik und Navigation. In Abbildung 3.1 ist die Basis ohne weitere Komponenten, wie Arm oder Laserscanner, dargestellt. Die youBot Basis besteht aus vier omnidirektionalen Reifen, die einen Radstand von 471mm haben. Die hinteren Reifen sind fest montiert, das bedeutet diese Reifen haben immer den gleichen Abstand zur Basis. Die vordere Achse hingegen lässt sich um den Mittelpunkt ‚ihrer selbst, leicht verdrehen. Dies dient dazu, dass Unebenheiten des Bodens ausgeglichen werden, da das

Fahrzeug mit vier Reifen statisch überbestimmt ist. Des Weiteren ist in der Basis ein Mini ITX PC-Board mit integrierten Intel®Atom Dual-Core CPU, 2 GB RAM und einer 32 GB SSD Festplatte verbaut. Die youBot Basis wiegt 20 kg und hat ein zulässiges Gesamtgewicht von 40 kg. Die maximale Geschwindigkeit liegt bei 0,8 m/s. Der Datenaustausch findet, mit 1ms Zyklen, über eine EtherCAT Schnittstelle statt. Die Stromversorgung wird mittels einer 24V, 5Ah Batterie geregelt. Es ist eine Wartungsfreie Bleibatterie. Der KUKA youBot kann damit ungefähr 90 Minuten lang betrieben werden, ohne geladen werden zu müssen. [vgl. KUKb]

Die genauen Abmaße zur Montage, von einem oder zwei Armen sowie der Plattform/ Ladefläche, werden in Abbildung 3.2 und 3.3 veranschaulicht.

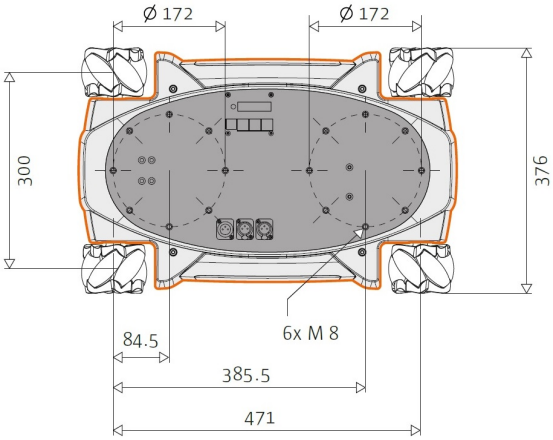


Abb. 3.2.: Maße Draufsicht

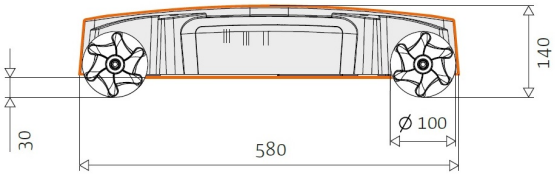


Abb. 3.3.: Maße Vorderansicht

In Abbildung 3.4 wurden die Reifen, die an dem youBot verbaut wurden, dargestellt. Der Reifen wurde von einem Ingenieur der schwedischen Firma Mecanum AB erfunden und hat so auch seinen Namen bekommen.

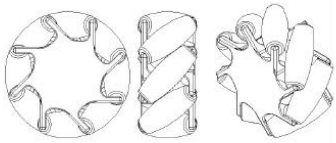


Abb. 3.4.: Mecanum-Reifen, basiert auf Ilon's Konzept

Das Mecanum-Rad besitzt keine geschlossene Lauffläche. Auf seiner Lauffläche sind, im Winkel von 45° zur Achse des Reifens, tonnenförmige Rollen gelagert. Durch die Wahl der Größe, Form und Abstände, ergibt sich eine durchgehende Abrollfläche. Mittels der Kombination der Drechrichtung und Drehgeschwindigkeit der einzelnen Räder, lässt sich eine Bewegungsrichtung erzeugen. [vgl. Mot]

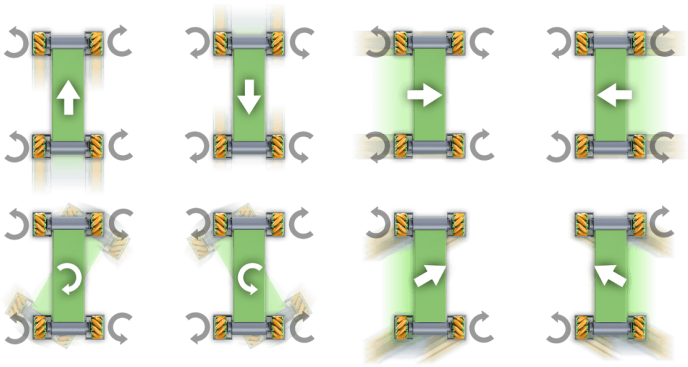


Abb. 3.5.: Reifenbewegung in Abhängigkeit zur Fahrtrichtung

Eine genaue Auflistung der einzelnen möglichen Bewegungsrichtungen wird in Abbildung 3.5 dargestellt. In dieser Darstellung wird die Drehrichtung der Reifen mit grauen Pfeilen dargestellt und die daraus resultierende Bewegungsrichtung, des youBots, mit weißen Pfeilen.

3.1.2. Laserscanner

Abb. 3.6.: Hokuyo URG-04LX-UG01

Der Hokuyo URG-04LX-UG01 ist ein 2D Entfernungsmesser, der eine Reichweite von 5600mm besitzt und über einen Winkel von 240° misst. Er ist nur für den Gebrauch in Gebäuden vorgesehen. Seine Genauigkeit für Distanzen bis 1000mm liegt bei ±30mm, für größere Distanzen bis 5600mm liegt sie bei ±3%. Die Scan Frequenz liegt bei 10 HZ und die Winkel Auflösung beträgt 0,352°. Er wurde speziell für die Robotik und das autonome Navigieren entwickelt. Abbildung 3.6 zeigt den verwendeten Laserscanner. [vgl. Hok]

Dieser Laserscanner kann für diverse Aufgaben benutzt werden, wie zum Beispiel zur Kartenerstellung oder zum Ausweichen von Hindernissen. In ROS gibt es dafür verfügbare Software, die neueste ist die urg_node.

3.1.3. NUC PC-Kit

Das für das Retrofitting der Hardware benötigte NUC PC-Kit besteht aus drei Komponenten. Die Grundkomponente ist der INTEL NUC7i3BNH PC der mit einer 128GB INTEL SSD 600P Series Model: SSDPEKKW128G7 und dem 8GB DDR4 SO-DIMM RAM Arbeitsspeicher ausgerüstet wird.

Aufgrund des erhöhten Stromverbrauchs der i5 und i7-Prozessoren wurde ein i3-Prozessor verwendet. Der erhöhte mögliche Nutzten der beiden anderen Prozessoren kompensiert nicht den erhöhten Stromverbrauch.
Die Laufzeit des youBots mittels Akku würde unnötig verringert werden. Diese Laufzeit liegt mit dem alten internen PC nur bei 90 Minuten, siehe Kapitel 3.1.1.

Abb. 3.7.: NUC

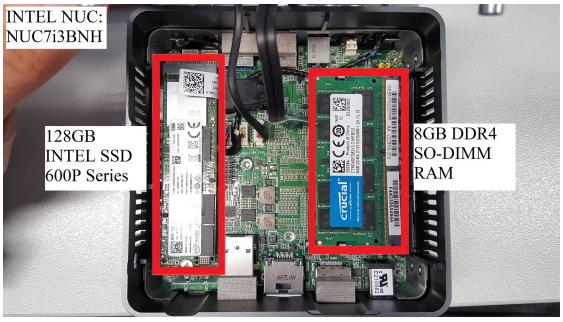


Abb. 3.8.: NUC Zusammenbau

- Mini PC Kit => NUC7i3BNH
- 8GB RAM Riegel => Crucial DDR4 SO-DIMM
- 128GB Festpaltte => INTEL SSD 600P Series Model: SSDPEKKW128G7

3.1.4. Schnittstelle NUC-youBot

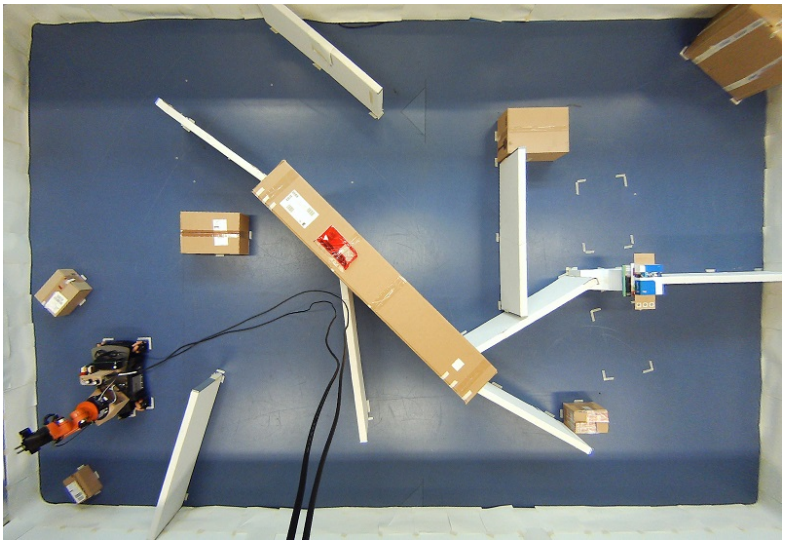
Um den NUC in den youBot einbauen sowie anschließen zu können, wurden folgende Komponenten benötigt:

- Aluminiumblech, Dicke t=2mm, 203 x 129mm
- Aluminiumblech, Dicke t=2mm, 50 x 92.5mm
- Lochplatten-Winkel, 80 x 80 x 80mm
- DC-Stecker, 2.5 x 5.5mm

- P4 Stromverlängerungskabel, 37cm
- HDMI Verlängerungskabel, 50cm
- 2x Intern USB 2.0-Kabel, 40cm

Die Benutzung dieser Komponenten wird in Kapitel 4.1 ausführlicher erklärt.

3.1.5. Arena (Umgebung)



Es wird eine Umgebung für den Roboter benötigt. Diese Umgebung ist in diesem Fall eine Fußball-Arena. Sie besteht aus 1,5m hohen Wänden und wurde auf der Höhe des Laserscanners abgeklebt. Dies war nötig, da die Oberflächen der Wände die Laserstrahlen ungünstig reflektieren. Die Arena misst 4 x 6m.

Abb. 3.9.: Arena Draufsicht

3.2. Software

Die im folgenden genanten Programme bilden, in dieser Arbeit, die Basis der Programmierung.

3.2.1. Linux Ubuntu

- Das benötigte Betriebssystem ist die Linux-Version Ubuntu 16.04.2 LTS. Es sollte eine LTS-Version verwendet werden, um zukünftigen möglichen Support zu erhalten.

3.2.2. ROS-Kinetic

- Nach Anleitung der wiki.ros Seite [DHo].
  Diese ROS-Version hat keinen youBot Support!

3.2.3. MATLAB

- Es wurde die MATLAB-Version 2017a verwendet.
  Zusätzlich wurde das Add-on 'Robotics System Toolbox' benötigt.

  Diese Toolbox beinhaltet Algorithmen und Verbindungen für das Entwickeln von autonomen Roboter Anwendungen, auf Basis von Hardware. Die Toolbox beinhaltet Algorithmen wie den Pfad Planer und das Ausweichen von Hindernissen. Des Weiteren schafft diese Toolbox eine Schnittstelle zwischen MATLAB und dem ROS. [vgl. Mata]

4. Durchführung

Im folgenden wird veranschaulicht wie die Aufgabe der Bachelorarbeit umgesetzt wurde. Dazu gehören das Retrofitting der Hard- und Software und das anschließende autonome Navigieren des youBots.

4.1. Retrofitting der Hardware

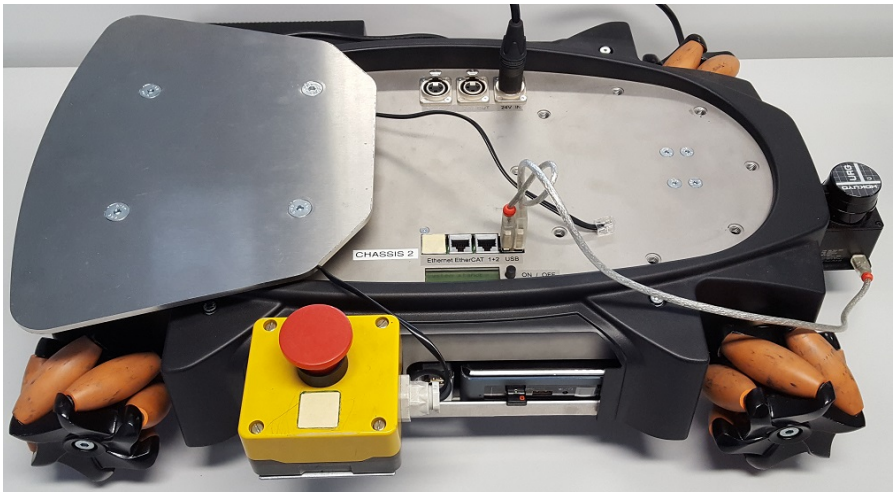


Abb. 4.1.: Fertig montierte youBot Basis

Das Durchführen zum Retrofitting der Hardware gliedert sich in vier Bereiche, die als Resultat den zusammengebauten youBot, wie in Abbildung 4.1, ergeben.

- Halterungen für HDMI, NUC und Not-Aus
- NUC-youBot Schnittstelle
- Computer Austausch
- Not-Aus Montage

Die nötigen Informationen zum Austausch, wurden von den folgenden Quellen bezogen: Nötige Stromversorgung:[Int], möglicher RAM:[cmt], Stromversorgungs-Anschluss:[Sch].

4.1.1. Halterungen für HDMI, NUC und Not-Aus

Es wurde für den neuen internen PC sowie für das HDMI-Kabel eine neue Halterung benötigt. Als erstes wurden Prototypen erstellt, wie in Abbildung 4.2 und 4.3 zu sehen ist. Diese Prototypen bestehen aus Pappe.

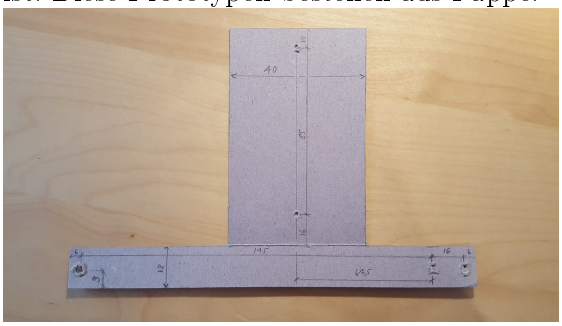


Abb. 4.2.: NUC-Halterung-Prototyp

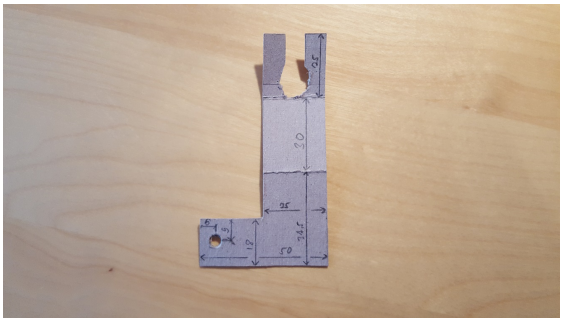


Abb. 4.3.: HDMI-Halterung-Prototyp

Die daraus resultierenden fertigen Halterungen sind aus Aluminium-Blechen gefertigt, mit der Dicke t=2mm, Abbildung 4.4 und 4.5.

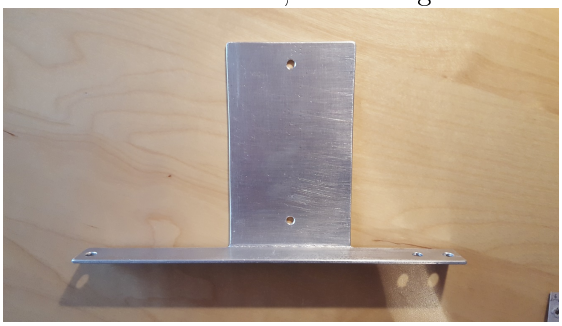
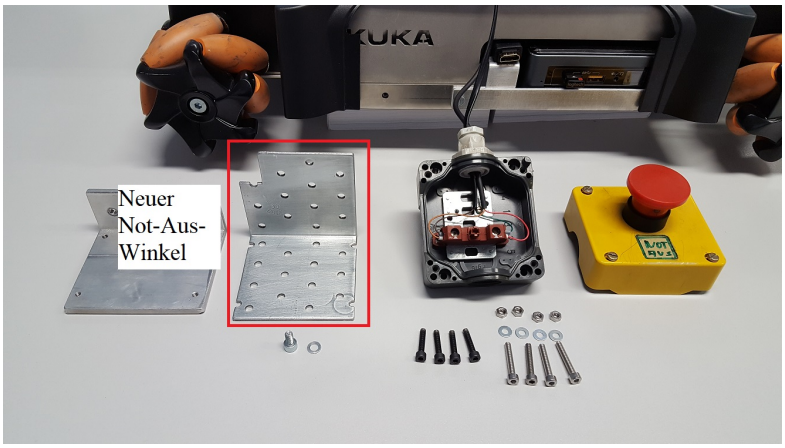


Abb. 4.4.: NUC-Halterung

Abb. 4.5.: HDMI-Halterung



Die Halterung für den Not-Aus-Schalter wurde durch eine neue ersetzt, wie in Abbildung 4.6 dargestellt ist. Da sonst die alte Halterung mit dem neuen Ausgang des HDMI-Kabels kollidieren würde. Die Position des Schalters wurde erhöht.

Abb. 4.6.: Halterung des Not-Aus-Schalters

4.1.2. NUC-youBot Schnittstelle

Um den NUC in den youBot einbauen zu können, wurden die Schnittstellen angepasst. Wie in Abbildung 4.12 zu sehen ist.

Der vorherige interne PC besaß einen internen Stromanschluss, daher wurde für den NUC der DC Stecker an 2 Polen des P4 Stromverlängerungskabels gelötet.

- DC Stecker 2.5 x 5.5mm
- P4 Stromverlängerungskabel 37cm

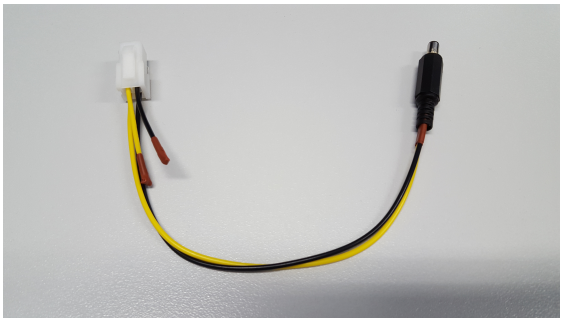


Abb. 4.7.: DC-Stecker am Stromkabel

Abb. 4.8.: HDMI-Kabel an HDMI-Winkel

Da der HDMI-Ausgang vom NUC hinten ist, wurde ein HDMI-Verlängerungskabel benötigt. Dieses Kabel ermöglicht ein flexibles Arbeiten, da ein schnelles wechseln eines HDMI-Kabels, vom Monitor, erfolgen kann.

- HDMI Verlängerungskabel 50cm

Zwei von vier USB-Anschlüssen sind hinten am NUC verbaut. Um diese auch nutzten zu können, beziehungsweise um die zwei USB-Ausgänge auf dem youBot nutzen zu können, wurden sie mit dem internen auf externen USB 2.0 Kabel verbunden.

- Intern USB 2.0-Kabel 40cm

Abb. 4.9.: 2x interner USB2.0 Kabel

4.1.3. Computer Austausch

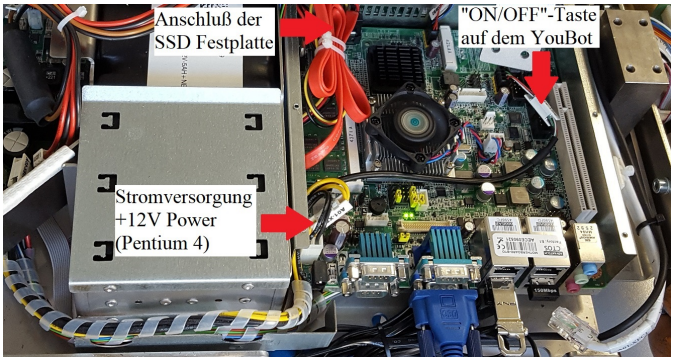


Abb. 4.10.: Intel®Atom Dual-Core

Der alte interne PC wurde mittels Pentium 4 Stromversorgung betrieben, wie in Abbildung 4.10 dargestellt ist. Die SSD Festplatte wurde über ein SATA Kabel angeschlossen und ist auf dem Bild nicht zu sehen, da sie unter dem PC verbaut wurde. Des Weiteren wurde der PC mit dem ON/OFF Taster auf der Basis ein- und ausgeschaltet.

Alle Anschlüsse zum internen PC wurden getrennt. Da der neue PC nur ein Ethernet-Anschluss besitzt wurde das zweite Ethernet Kabel, welches nicht mehr benötigt wurde, an die Unterseite des Deckel geklebt. Des Weiteren wurde Kabelmanagement mittels Kabelbinder betrieben, wie in Abbildung 4.11 zu sehen ist.

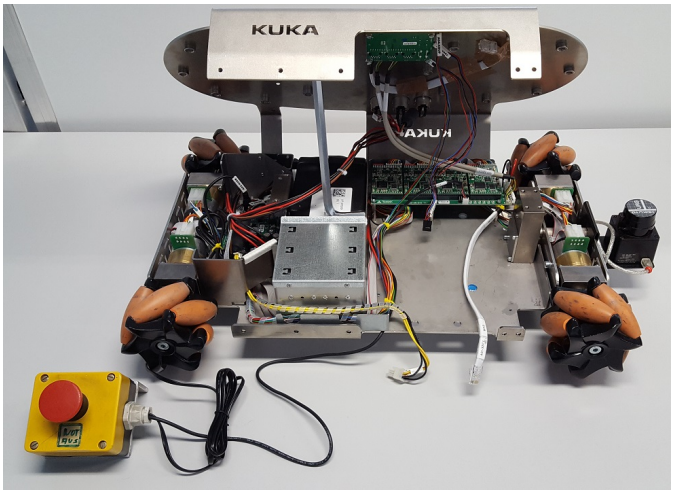


Abb. 4.11.: youBot ohne PC

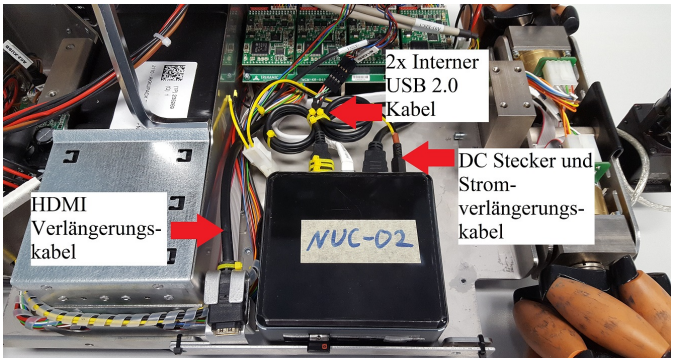


Abb. 4.12.: Intel®NUC7i3BNH

In Abbildung 4.12 ist der fertig eingebaute NUC dargestellt. Die Kabel wurden platzsparend gelegt beziehungsweise aufgerollt. Die Halterungen wurden für das Foto provisorisch mit Kabelbindern fixiert und anschließend mit schrauben festgeschraubt, siehe Abbildung 4.13 und 4.14.

4.1.4. Not-Aus Montage

Der Not-Aus-Schalter ist ein wichtiger Bestandteil für das Arbeiten mit dem Roboter. Der Roboter kann ungewollte Bewegungen ausführen und so eine Zerstörung von Bauteilen des youBots sowie von Objekten innerhalb der Arena herbeiführen. Mittels Not-Aus kann dies unterbunden werden.

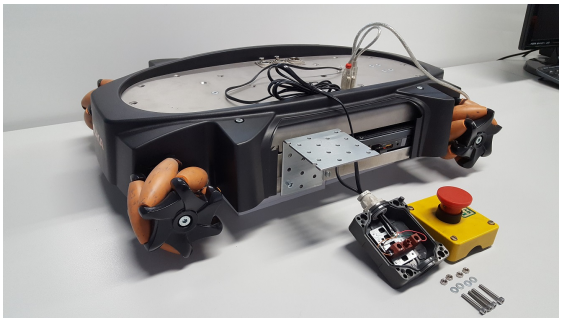


Abb. 4.13.: Winkel montiert

Abb. 4.14.: Not-Aus Unterseite montiert

In Abbildung 4.13 und 4.14 ist die Halterung des Not-Aus-Schalters montiert. Sie wird von einer Schraube gehalten, stützt sich aber zusätzlich auf die Halterungen vom HDMI und NUC ab, siehe Abbildung 4.14. Für die Belastung von oben beziehungsweise das Auslösen des Not-Aus-Schalters ist die Not-Aus-Halterung stabil verbaut worden.

4.2. Retrofitting der Software

Die Benutzung der Software basiert auf den Literaturen, ROS By Example [Goe15a][Goe15b] und Learning ROS for Robotics Programming [FCMM15a][FCMM15b][FCMM15c], sowie der wiki.ros Seite[Hoa].

Um das Retrofitting der Software durchführen zu können, wurde die Linux-Version Ubuntu 16.04.2 LTS installiert. Anschließend wurde die Software ROS-Kinetic installiert, da es der neueste anwendbare ROS-Treiber ist. Lediglich Lunar Loggerhead ist neuer aber noch im Entwicklungsstatus, beziehungsweise gibt es dafür keine LTS Version. Als nächstes wurde ein catkin_ws eingerichtet [Mar]. Catkin_ws bedeutet Arbeitsumgebung. Die Bash wurde zudem auch eingestellt, siehe nächstes Kapitel.

4.2.1. Bash Einstellungen

Die Bash (Bourne again Shell) dient als Konfiguration des Terminals/ der Shell.
In der Datei .bashrc wurden fünf Zeilen Code hinzugefügt. Die drei, für die youBot-Verbindung, wichtigsten Zeilen werden im folgenden erklärt.

- export ROS_PACKAGE_PATH=/home/youbot-02/catkin_ws/src:/opt/ros/kinetic/share
  - Diese Zeile verknüpft den src Ordner mit dem share Ordner, sodass Treiber aus dem catkin_ws auf benötigte Treiber aus dem kinetic Ordner zu greifen können.
- export ROS_MASTER_URI=http://127.0.0.1:11311
  - Die Vergabe einer IP-Adresse für den ROS-Master-Server, in diesem Fall die Lokale-IP-Adresse, da nicht mit anderen Computern kommuniziert wird.
- export ROS_HOSTNAME=127.0.0.1
  - Die Vergabe einer IP-Adresse für den ROS-Hostname. Da der Master-PC auch gleich der Host-PC ist, wurde wieder die Lokale-IP-Adresse vergeben.

4.2.2. youBot Treiber

Die youBot-Treiber wurden nach der Anleitung von Herrn Walter Nowak installiert. Herr Nowak arbeitet für die Locomotec Firma, dessen Tochterfirma die youBot Store GmbH ist. Lediglich der Treiber für die Navigation wurde zusätzlich, manuell installiert und angepasst.

- Nach Herrn Walter Nowak.
  - "ros-kinetic-pr2-msgs
  - ros-kinetic-controller-manager
  - manuel installiert im catkin_ws:
    - youbot_driver
    - youbot_driver_ros_interface
    - youbot_description
    - youbot_simulation
    - brics_actuator"
      [vgl. Now]
- Benötigte weiterführende Ergänzungen:
  - manuel installiert im catkin_ws:
    - youbot_navigation
  - Fehlerbehebung

  Das youbot_navigation Paket wurde für ROS-Indigo geschrieben. Die Pfade in ROS-Kinetic werden anders angelegt, dies wird im folgenden erklärt.

  Das Kompilieren durch catkin_make löste immer einen Fehler aus. Dieser Fehler war das Fehlen des Pfades zu den bestehenden Dateien im System Ordner. Das Problem sollte durch einstellen des ROS_PACKAGE_PATH in der Bash gelöst werden. Da, dadurch das Problem nicht gelöst wurde, wurden die Pfade alle manuell richtig eingestellt.

Beispiel anhand der Datei lowpass_filter.cpp:

- #include </ros/ros.h>
- #include </geometry_msgs/Twist.h>

wurde in das Folgende geändert

- #include </opt/ros/kinetic/include/ros/ros.h>
- #include </opt/ros/kinetic/include/geometry_msgs/Twist.h>

Der Pfad /opt/ros/kinetic/include wurde bei 41 Dateien geändert, siehe Anhang B.

4.2.3. Zusätzlich benötigte ROS-Treiber

Um die autonome Navigation durchführen zu können, wurden folgende Treiber benötigt:

- ros-kinetic-urg-node
  - Ist der Treiber für den Laserscanner.
- ros-kinetic-scan-tools
  - Zum Konvertieren der Scan-Daten in PointCloud2-Daten.
- cloud to cloud converter (manuel installiert im catkin_ws)
  - Zum Konvertieren der PointCloud2-Daten in PointCloud1-Daten.
- ros-kinetic-map-server
  - Zum Speichern und Laden einer Map.
- ros-kinetic-slam-gmapping
  - Zum Erstellen einer Map.
- ros-kinetic-amcl
  - Zum Ausführen des AMCL- Algorithmus.
- ros-kinetic-move-base
  - Wurde benötigt damit der youBot autonom verfahren kann, da der eigene Treiber einen Fehler hat: Der youBot bleibt am Ziel nicht stehen, er dreht sich mit minimaler Geschwindigkeit weiter. Das Problem sollte eigentlich gelöst worden sein:

    > "Ok. First off, I'll answer the question about turning at a really low angular speed. This is just a bug which should be fixed in the next patch release of navigation."[eit]

    Da der youBot_navigation Treiber aber nicht aktualisiert wurde, wurde das Problem, in dieser Arbeit, mit dem zusätzlichen move-base Treiber behoben.

Eine Liste aller ros-kinetic Treiber kann mittels folgendem Befehl angezeigt werden:

- apt-cache search ros-kinetic

4.3. Autonomes navigieren

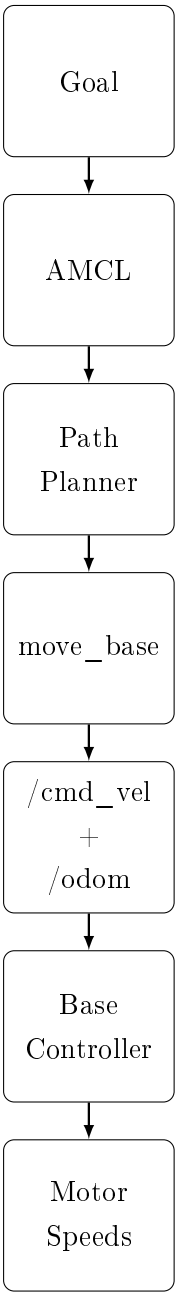


Abb. 4.15.: Motion Control Hierarchy

Der youBot soll autonom navigieren. Dies wird möglich aus dem Zusammenspiel verschiedenster Komponenten. In Abbildung 4.15 ist die Hierarchie dieser Komponenten abgebildet, mit der das System arbeitet.
Mittels Goal wird ein Ziel bestimmt, zudem der youBot fahren soll. Der AMCL Logarithmus vergleicht ständig den aktuellen Ort des youBots mittels Laserscanners mit der geladenen/erstellten Karte. In dieser Karte wird mittels Path Planner ein Weg berechnet der möglichst kurz ist.
Das move_base Programm verfährt mit den Daten der vorangegangenen Komponenten zum Ziel. Dies geschieht durch die Hilfe der Nodes, /cmd_vel + /odom, die den Base Controller Daten übergeben und der wiederum steuert die einzelnen Motor Geschwindigkeiten.

Im folgenden Kapitel werden die Verknüpfungen der Programme/ Daten erklärt.

4.3.1. Verknüpfung der Programme/ Daten (navigation stack)

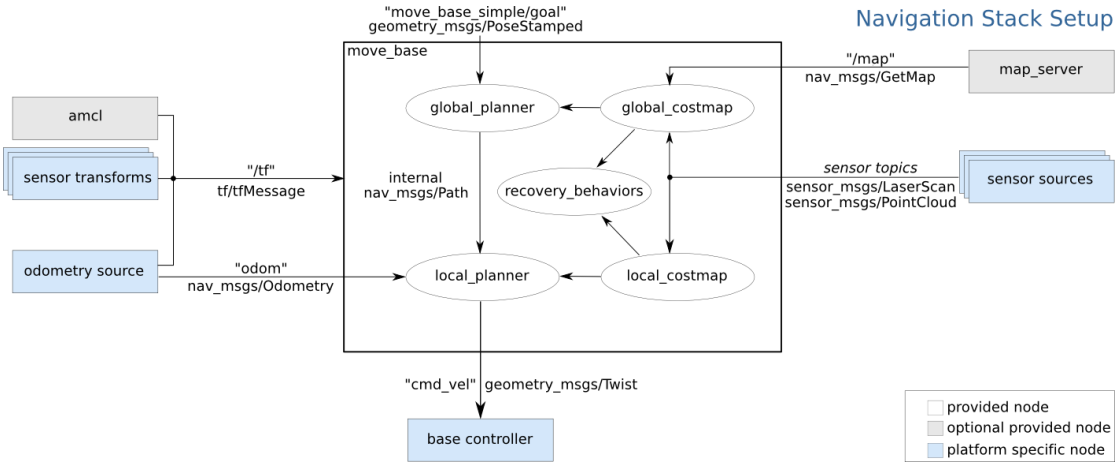


Abb. 4.16.: navigation stack

"In order to understand the navigation stack, you should think of it as a set of algorithms taht use the sensors of the robot and the odometry so that you can control the robot using a standard message. It can move your robot without any problems, such as crashing, getting stuck in a location, or getting lost to another position.

...

The robot must satisfy some requirements before it uses the navigation stack:

- The navigation stack can only handle a differential drive holonomic-wheeled robots. The shape of the robot must either be a square or rectangle. However, it can also do certian things with biped robots, such as robot localization, as long as the robot does not move sideways.
- It requires that the robot publishes information about the relationships between the positions of all the joints and sensors.
- The robot must send messages with linear and angular velocities.
- a planar laser must be on the robot to create the map and localization. Alternatively, you can generate something equivalent to several lasers or a sonar, or you can project the values to the ground if they are mounted at another place of the robot.

... [Abbildung 4.16] shows you how the navigation stack are organized. You can see three groups of boxes with colors (gray and white) and dotted lines.

The plain white boxes indicate the stacks that are provided by ROS, and they have all the nodes to make your robot really autonomous:“[FCMM15a, S. 303-304]

4.3.2. Funktion des move_base Programms

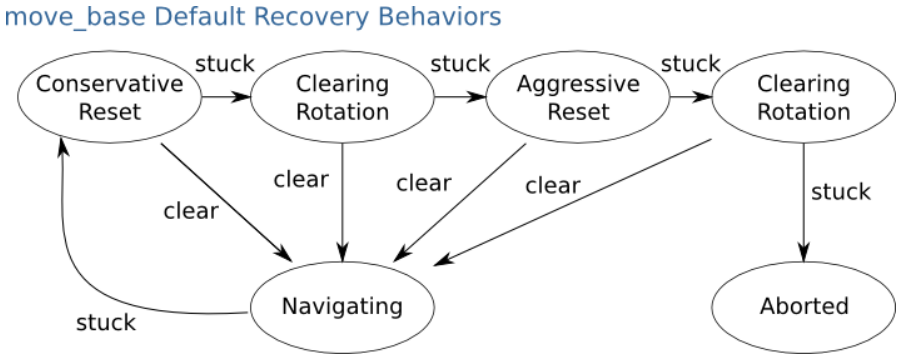


Abb. 4.17.: move_base

Im folgenden Zitat wird die Funktion des move_base Programms erklärt, das in Abbildung 4.17 schematisch dargestellt wurde.

> “Running the move_base node on a robot that is properly configured (please see navigation stack documentation for more details) results in a robot that will attempt to achieve a goal pose with its base to within a user-specified tolerance. In the absence of dynamic obstacles, the move_base node will eventually get within this tolerance of its goal or signal failure to the user. The move_base node may optionally perform recovery behaviors when the robot perceives itself as stuck. By default, the move_base node will take the following actions to attempt to clear out space:
>
> First, obstacles outside of a user-specified region will be cleared from the robot's map. Next, if possible, the robot will perform an in-place rotation to clear out space. If this too fails, the robot will more aggressively clear its map, removing all obstacles outside of the rectangular region in which it can rotate in place. This will be followed by another in-place rotation. If all this fails, the robot will consider its goal infeasible and notify the user that it has aborted. These recovery behaviors can be configured using the recovery_behaviors parameter, and disabled using the recovery_behavior_enabled parameter.“[Mor]

4.3.3. AMCL- Algorithmus

Die Anwendung des Algorithmus wurde bewerkstelligt, mittels der wiki.ros Seite [Shaa] und den Büchern [FCMM15b, S.357-365] und [Goe15b, S.108-122].

> "amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.
> ...
> amcl takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, amcl initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0). [In Kapitel 4.3.5 wird erklärt wie die Parameter von initial_pose_x, initial_pose_y und initial_pose_a eingestellt wurden.]
> ...
> amcl transforms incoming laser scans to the odometry frame (~odom_frame_id). So there must exist a path through the tf tree from the frame in which the laser scans are published to the odometry frame.
> An implementation detail: on receipt of the first laser scan, amcl looks up the transform between the laser's frame and the base frame (~base_frame_id), and latches it forever. So amcl cannot handle a laser that moves with respect to the base.
> The drawing .. [Abbildung 4.18] shows the difference between localization using odometry and amcl. During operation amcl estimates the transformation of the base frame (~base_frame_id) in respect to the global frame (~global_frame_id) but it only publishes the transform between the global frame and the odometry frame (~odom_frame_id). Essentially, this transform accounts for the drift that occurs using Dead Reckoning. The published transforms are future dated. "[Shaa]

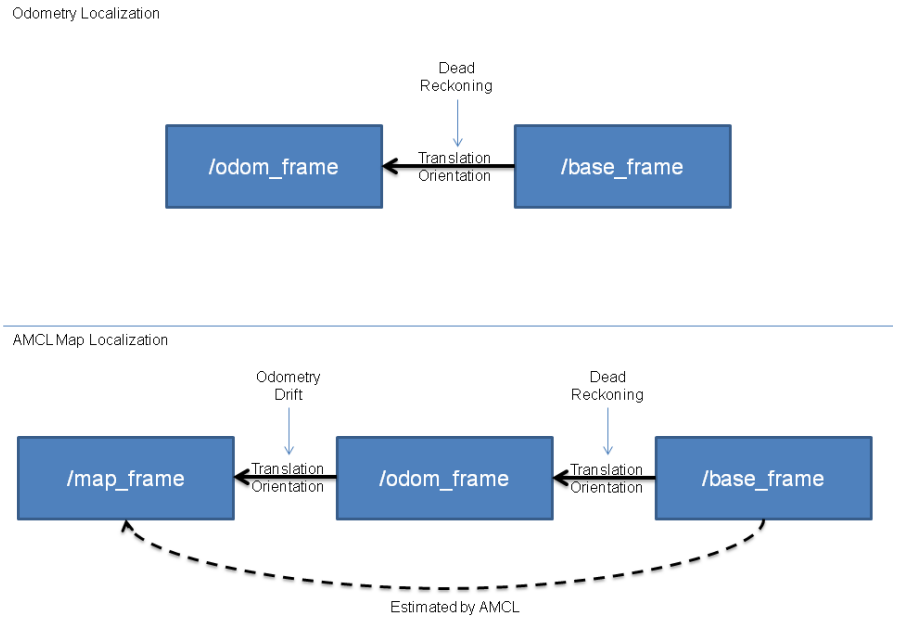


Abb. 4.18.: AMCL-Vergleichs-Schaubild

Nach dem Starten des amcl_sebe.launch Programms, das in Kapitel 4.3.7 angewendet wird, erstellt es mögliche Positionen des youBots mit roten Pfeilen. Nachdem der youBot verfahren wurde, hat das Programm eine Reduktion der möglichen Positionen errechnet. In Abbildung 4.19 zu 4.20 wird diese Reduktion dargestellt.

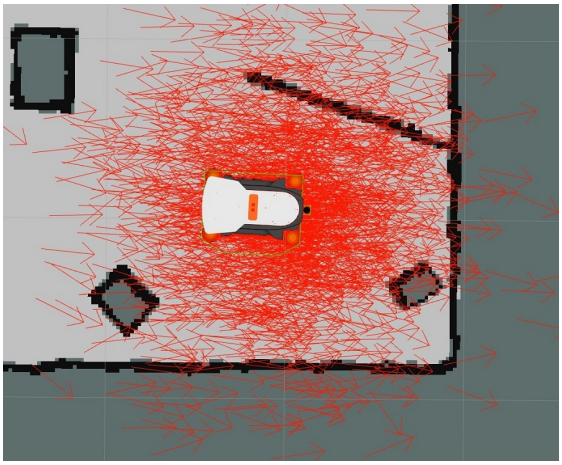


Abb. 4.19.: ParticleCloud nach Start der amcl_sebe.launch

Abb. 4.20.: ParticleCloud nach dem Verfahren des youBots

4.3.4. Laserscanner

Die Installation und Einrichtung des Laserscanners wurde mit Hilfe der folgenden Quelle absolviert:[FCMM15c, S.136-139]

Da das move_base Programm die Daten des Laserscanners als PointCloud1 Datentyp benötigt, musste der Datentyp konvertiert werden.

- Aus sensor_msgs/LaserScan wurde sensor_msgs/PointCloud2 mittels:
  - rosrun scan_to_cloud_converter scan_to_cloud_converter_node
- Und anschließend wurde aus PointCloud2 PointCloud1 mittels:
  - rosrun point_cloud_converter point_cloud_converter_node points2_in:=/cloud points_out:=/points_out_sebe

```
youbot-02@NUC-02:~$ rostopic list
/cloud
/diagnostics
/points2_out
/points_in
/points_out_sebe
/rosout
/rosout_agg
/scan
/urg_node/parameter_descriptions
/urg_node/parameter_updates
```

Abb. 4.21.: Topics des Laserscanners und der Konvertierung

In Abbildung 4.21 sind die Topics zu sehen, die erstellten wurden. Die Nodes kommunizieren/ streamen über diese Topics. Die Topics /points2_out und /points_in wurden nicht benötigt, aber automatisch mit erstellt.

4.3.5. Parameter- und Konfigurationseinstellungen

Die Parameter wurden mit Hilfe folgender Quellen ergänzt und eingestellt:

- wiki.ros amcl Seite [Shaa]
- wiki.ros base_local_planner Seite [Iva]
- ROS By Example Buch [Goe15b, S.75-81]
- Learning ROS for Robotics Programming Buch [FCMM15b, S.339-342]

Damit die youbot_driver.launch Datei nicht versucht den Arm-Treiber zu starten, wurde dieser in der Launch auf false geseztzt, siehe Abbildung 4.22.

```
<!-- Set relevant parameters. -->
<param name="youBotHasBase" type="bool" value="true"/>
<param name="youBotHasArms" type="bool" value="false"/>
```

Abb. 4.22.: Ausschnitt aus der youbot_driver.launch Datei

```
<!-- Startpunkt des YB in der Map Richtig setzten! -->
<param name="initial_pose_x" value="0.8"/>
<param name="initial_pose_y" value="-1.97"/>
<param name="initial_pose_a" value="-0.03"/>
```

Abb. 4.23.: Ausschnitt aus der amcl_sebe.launch Datei

Um in der Karte/ Map in der richtigen Position zu starten, siehe Abbildung 4.28, ist diese Position hart einprogrammiert. Die Werte in Abbildung 4.23 initial_pose_x, initial_pose_y und initial_pose_a geben die Position des youBots in der Karte/ Map an und wurden nachträglich einprogrammiert. Die Werte wurden aus RViz übernommen beziehungsweise so lange korrigiert, bis sie in der Markierung der youBot-StartPosition lagen.

Kompletter Code der beiden Launch Dateien siehe Anhang C.

In Abbildung 4.24 ist die Programmierung der Footprint des youBots zu sehen. In dem roten Kasten ist die nachträgliche Änderung zu sehen, wie es am Ende des Kapitels 4.3.8 erklärt wird.

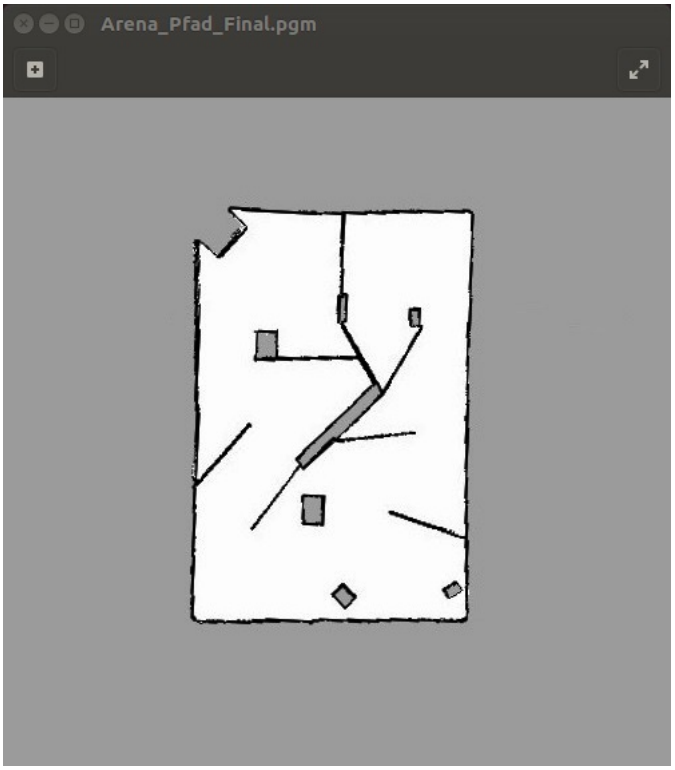
Die Einstellungen beziehungsweise wichtigen Dateien, der base_local_planner_params, arena_pfad, costmap_common_params und youbot_sebe.rviz sind im Anhang D zu finden.

```
#change this
footprint: [[0.26, 0.18],
            [0.26, 0.014],
            [0.31, 0.014],
            [0.31, -0.014],
            [0.26, -0.014],
            [0.26, -0.18],
            [0.15, -0.25],
            [-0.15, -0.25],
            [-0.27, -0.18],
            [-0.27, 0.18]]

controller_frequency: 10.0
controller_patience: 15.0
clearing_radius: 0.25
footprint_padding: 0.03
```

Abb. 4.24.:
move_base_params.yaml Datei

4.3.6. Erstellung der Karte



In Abbildung 4.25 ist die Karte/ Map dargestellt, welche anhand des unten aufgeführten Codes erstellt und geöffnet wurde. Mittels dem youBot-WASD-Programm von Karsten Flores wurde der youBot verfahren. Der youBot kann auch wie folgt gesteuert werden. Mittels Keyboard nach Literatur [Goe15a, S.71], Gamepad oder Joystick [Goe15a, S.72]/ [FCMM15c, S.130-136]. Die Darstellung der Aufnahme zur Erstellung der Karte/ Map siehe Anhang E.

Abb. 4.25.: Fertig erstellte Karte/ Map

Die Erstellung der Karte wurde teilweise mittels den Büchern [Goe15b], [FCMM15a, S.330-333] und der wiki.ros Seite[son] bewerkstelligt.

- roslaunch youbot_driver_ros_interface youbot_driver.launch
  - Startet den youBot Treiber. Wenn kein roscore (Master-Server) gestartet wurde, wird auch dieser gestartet.
- rosrun rviz rviz -d youbot_sebe_mapping.rviz
  - Startet RViz mit der Konfig-Datei Einstellungen aus youbot_sebe_mapping.rviz.
- sudo chmod a+rw /dev/ttyACM0
  - Erteilt dem Benutzer die Erlaubnis den Laserscanner zu benutzten. Beziehungsweise den USB Port.

- rosrun urg_node urg_node _frame_id:="base_laser_front_link" _angle_min:=-1.7978254834 _angle_max:=1.7978254834
  - Startet den Laserscanner + Ort des Lasers am youBot + verringerten Winkel, da der youBot sonst seine eigenen Reifen sieht und versucht mittels AMCL diesen auszuweichen.
  - Start und Ausgabe, im Terminal, siehe Anhang F.
- roslaunch youbot_navigation_global 2dslam.launch
  - Startet die 2dslam.launch.
- Strg + C
  - Beendet die 2dlsam.launch, da nur die Parameter geladen werden sollen.
- rosrun gmapping slam_gmapping _frame_id:="base_laser_front_link"
  - Startet slam_gmapping + Ort des Lasers, am youBot.

  Nun wurde der youBot in der zu scannenden Arena verfahren, mit dem youBot-WASD-Programm, bis die Karte vollständig in RViz aufgenommen wurde.

- rosrun map_server map_saver -f Arena_Pfad
  - Speichert die aufgenommene Karte aus RViz als pgm-Datei.
- eog Arena_Pfad.pgm
  - Öffnet die Karte mit eog (Eye of Gnome) siehe Abbildung 4.25.

Es ist möglich die Karte nachträglich zu bearbeiten, "However, you can always edit a map using your favorite graphics editor."[Goe15b, S. 107] hierzu wurde das Paint Programm von Windows benutzt.
Um die Karte mit Paint bearbeiten zu können, wurde vorher die PGM-Datei, mittels Convertio [Con], in eine JPEG-Datei konvertiert. Danach wurde in Paint mit Schwarz, Grau oder Weiß die Karte geändert, beziehungsweise verbessert, siehe Unterschied von Abbildung 4.26 zu 4.27. Nach der Änderung in Paint muss das Bild wieder in eine PGM-Datei konvertiert werden, dies wurde mit dem gleichen Programm bewerkstelligt.

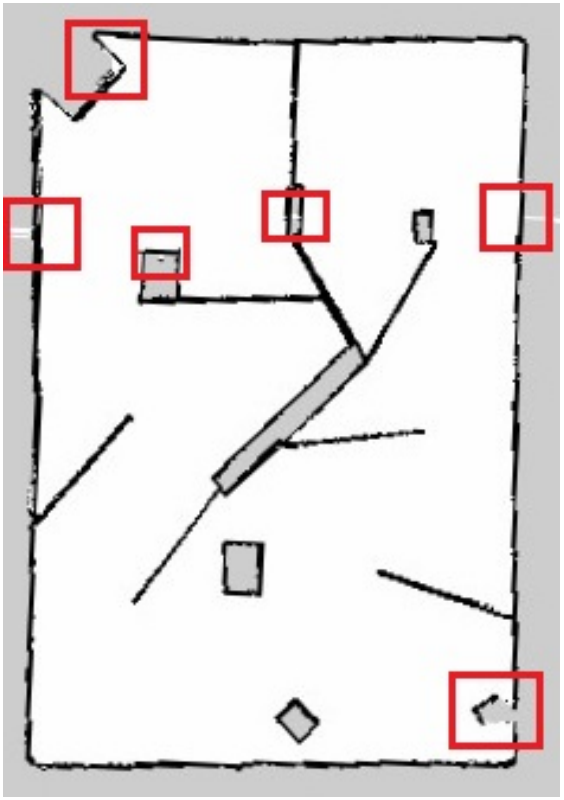


Abb. 4.26.: Karte unbearbeitet

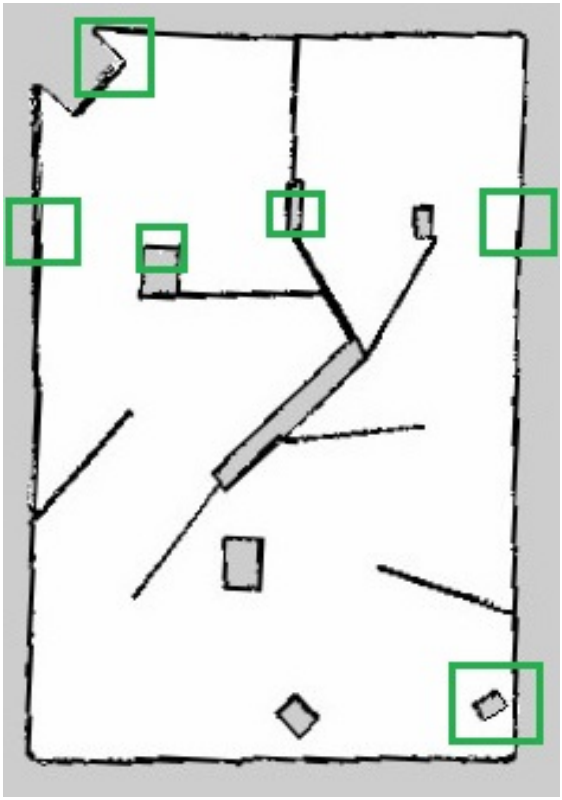


Abb. 4.27.: Karte bearbeitet

"For example, to keep your robot from entering a room, draw a black line across the doorway(s). (**Note**: There may be a bug in move_base that prevents this trick from working. You can also try setting all the pixels in forbidden region to mid-gray.) To remove a piece of furniture that has been moved, erase the corresponding pixels."[Goe15b, S. 107]

4.3.7. Programm zum autonomen Navigieren

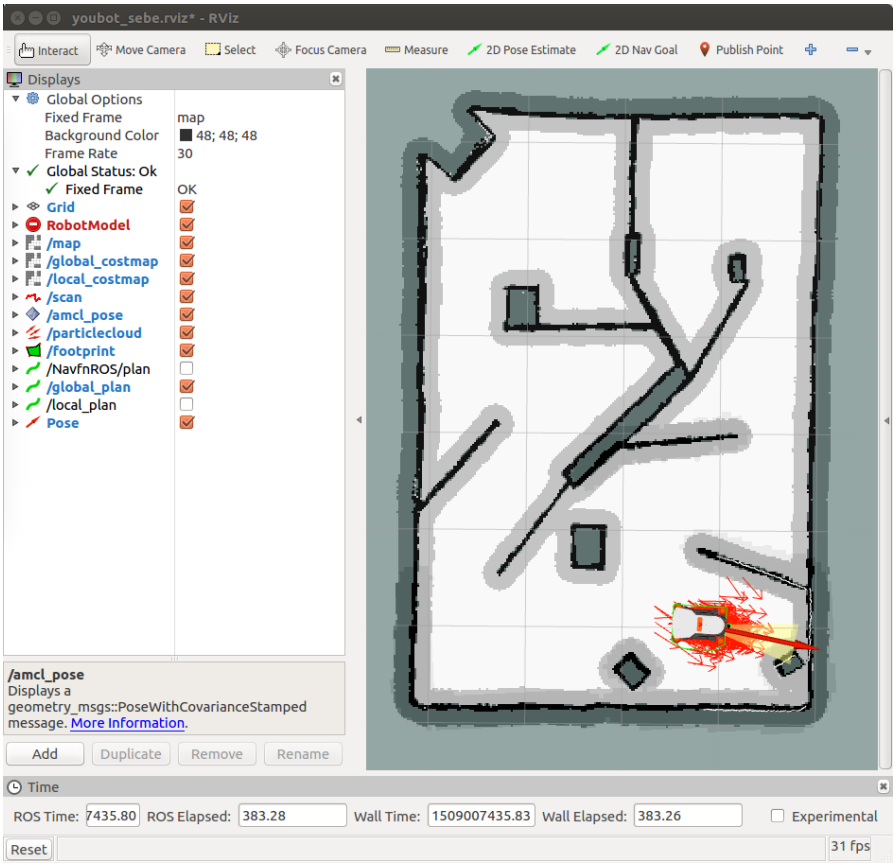


Abb. 4.28.: RViz autonomes navigieren

Die Abbildung 4.28 zeigt die RViz-Oberfläche, nachdem die folgenden Zeilen Code, in einem Terminal mit mehreren Fenstern, gestartet wurden:

- roslaunch youbot_driver_ros_interface youbot_driver.launch
  - Wie in Kapitel 4.3.6.
- rosrun rviz rviz -d youbot_sebe.rviz
  - Wie in Kapitel 4.3.6 allerdings mit einer anderen Konfig-Datei (youbot_sebe.rviz).
- sudo chmod a+rw /dev/ttyACM0
  - Wie in Kapitel 4.3.6.

- rosrun urg_node urg_node _frame_id:="base_laser_front_link" _angle_min:=-1.7978254834 _angle_max:=1.7978254834
  - Wie in Kapitel 4.3.6.
- rosrun scan_to_cloud_converter scan_to_cloud_converter_node
  - Konvertiert sensor_msgs/LaserScan in sensor_msgs/PointCloud2.
- rosrun point_cloud_converter point_cloud_converter_node points2_in:=/cloud points_out:=/points_out_sebe
  - Konertiert sensor_msgs/PointCloud2 in sensor_msgs/PointCloud1.
- rosrun map_server map_server Arena_Pfad.pgm
  - Lädt die Daten der Karte aus Arena_Pfad.pgm.
- rosrun map_server map_server Arena_Pfad.yaml
  - Sendet die Daten aus Arena_Pfad.pgm und Arena_Pfad.yaml in einem Intervall.
  - Start und Ausgabe, im Terminal, siehe Anhang F.
- roslaunch amcl_sebe.launch
  - Startet das Adaptive Monte Carlo Localization Programm.
- roslaunch youbot_navigation_global move_base_global.launch
  - Startet die move_base_global.launch.
- Strg + C
  - Beendet move_base_global.launch da nur die Parameter der local und global costmap geladen werden sollen.
- rosrun move_base move_base
  - Startet die move_base.
- rosrun rqt_reconfigure rqt_reconfigure
  - Startet rqt_reconfigure zum nachträglichen Einstellen von Parametern.
- rqt_graph
  - Startet den rqt_graph. Er zeigt die Verknüpfungen der einzelnen Nodes/Topics/ Programme untereinander an.

4.3.8. Arena

Die Arena hat den Nutzen, dass der youBot eine genau definierte Umgebung besitzt. Sie hat die Maße 4 x 6m. Die Hindernisse in der Arena können beliebig positioniert werden. In Abbildung 4.29 ist die Arena von vorne Fotografiert, dort steht der youBot an seiner Start- und Endposition. Auf der anderen Abbildung 4.30, sind Markierungen auf dem Boden zusehen. Diese Markierungen sind die Positionen die der youBot jeweils anfährt.

Abb. 4.29.: Vorne, Start/Ende Position

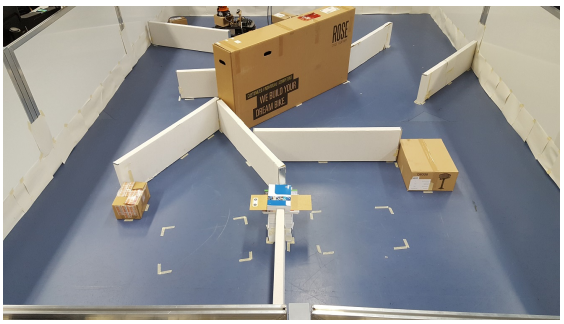


Abb. 4.30.: Hinten, Links- Rechts Position

In den Bilder 4.31 bis 4.33, sind die einzelnen Daten, die für move_base benötigt werden, dargestellt. Als erstes die Laser Daten, dann die local_map Daten, die aus den Laser Daten erstellt werden und als letztes die global_map Daten die aus gemappten Daten entstehen.

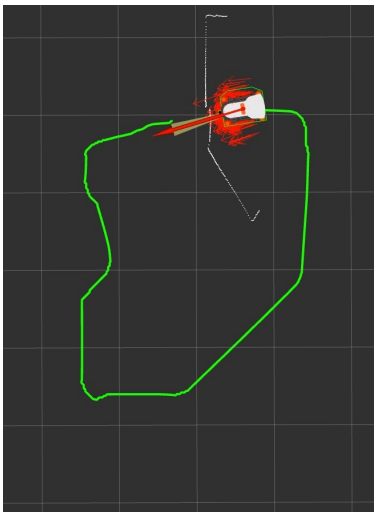


Abb. 4.31.: Laserscanner

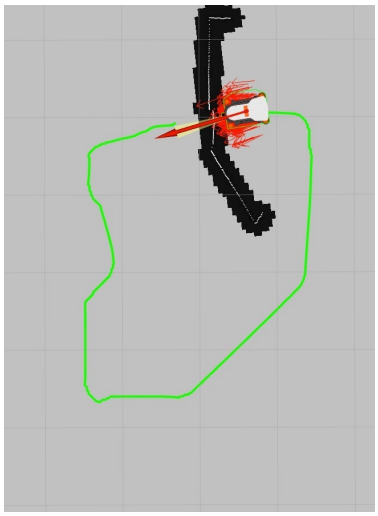


Abb. 4.32.: local_map

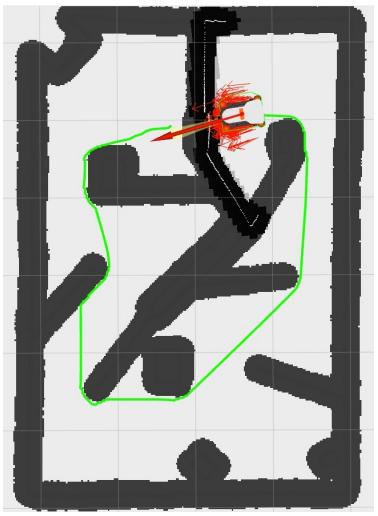


Abb. 4.33.: global_map

Die grüne Linie in den Abbildungen, gehört zum global_plan des Pathfinders und bezieht

sich als erstes auf die global_map, wo die statischen sich nicht verändernden Objekte vorhanden sind. Zudem bezieht es sich auf die local_map, wo sich neue und bewegende Objekte autonom umfahren lassen.

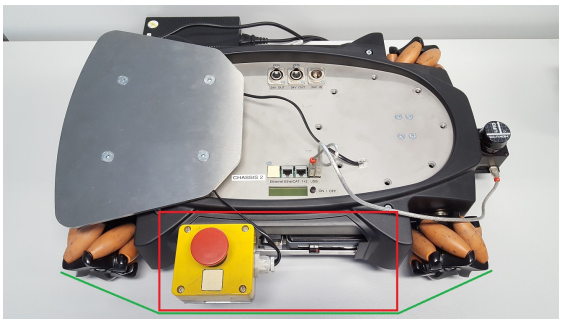


Abb. 4.34.: Footprint-Foto

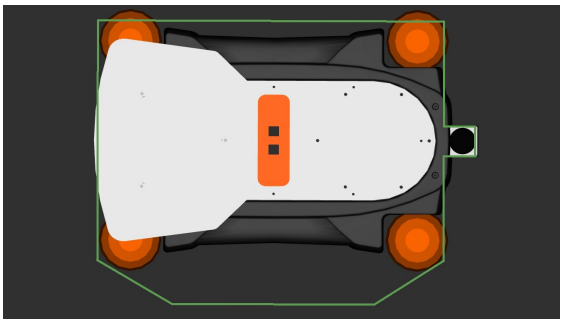


Abb. 4.35.: Footprint-RViz

Für das Navigieren wurde ein Footprint benötigt und angepasst, da der Not-Aus und das HDMI Kabel auf einer Seite, des youBots, überragen.
Das Footprint des youBots wurde angepasst, wie es in Abbildung 4.34 und 4.35 zusehen ist. In rot ist der Bereich umrandet, der ein Ändern des Footprints bedingt hat. In grün ist das geänderte Footprint dargestellt.
Das Footprint ist sehr wichtig, da move_base so verfährt, dass sich das Footprint nicht mit einem Hindernis überschneidet beziehungsweise die Pose des youBots sich nicht mit dem Inflatiuns Radius schneidet.

4.3.9. Ziele durch MATLAB publishen

Die Bestimmung des Ziels kann durch RViz geschehen, es wird auf 2D Nav Goal geklickt und dann der Ort in der Karte angeklickt, wo der Roboter hinfahren soll. Eine weitere Methode ist das Publishen über das Shell-Terminal, dies kann mittels abgewandeltem Code [Goe15b, S.84] wie folgt ausgeführt werden:

- rostopic pub /move_base/goal move_base_msgs/MoveBaseActionGoal '{header: {frame_id: "map"}, goal: {target_pose: {pose: {position: {x: -1.051, y: 2.430, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.01, w: 1.0}}}}}'

Des Weiteren ist es auch möglich in Pyton oder C++ zu schreiben. Im Folgenden wird die Möglichkeit gezeigt es über MATLAB zu publishen.

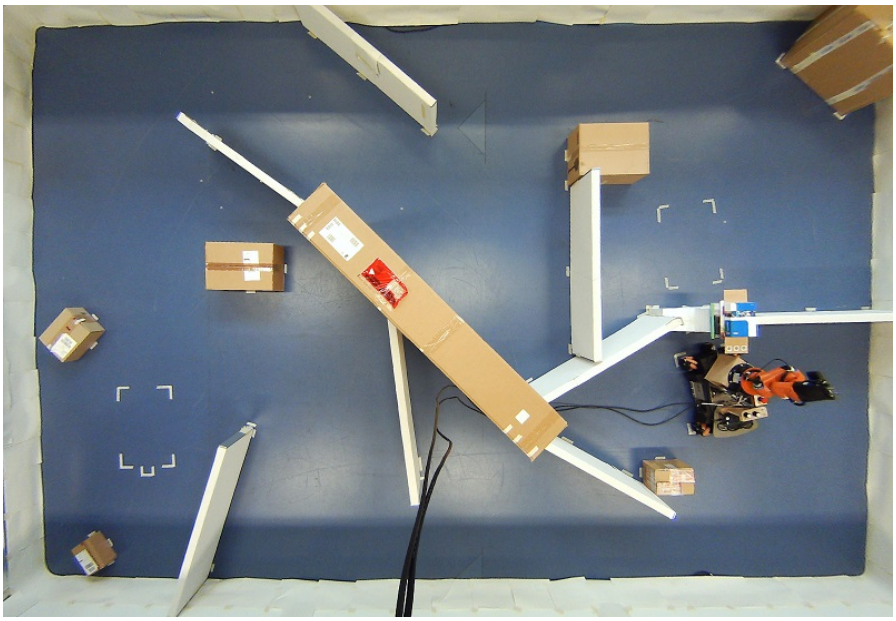


Abb. 4.36.: Start linker Pfad (End-Position rechter Pfad)

Das Folgende Beispiel ist das Anfahren der linken Position in der Arena, in Abbildung 4.36 Markierung oben rechts. Die Start-Position ist unten rechts, in der Abbildung die aktuelle Position des youBots.

- MATLAB mit ROS verbinden durch die Erstellung der Node Sebe:
  - rosinit('127.0.0.1','NodeName','Sebe');

- Zu den vorhandenen Nodes eine Verbindung erstellen:
  - statusSub = rossubscriber('/move_base/status', 'actionlib_msgs/GoalStatusArray');
  - goalPub = rospublisher('/move_base/goal', 'move_base_msgs/MoveBaseActionGoal');
- Eine kurze Pause für das Programm, da es sonst zu schnell ist und die folgenden Befehle nicht ausführen kann:
  - pause(0.1);
- Die Zielkoordinate wird jetzt gepackt:
  - msgsGoalPub = rosmessage(goalPub);
  - msgsGoalPub.Goal.TargetPose.Header.FrameId = 'map';
  - msgsGoalPub.Goal.TargetPose.Pose.Position.X = -1.051;
  - msgsGoalPub.Goal.TargetPose.Pose.Position.Y = 2.430;
  - msgsGoalPub.Goal.TargetPose.Pose.Position.Z = 0.000;
  - msgsGoalPub.Goal.TargetPose.Pose.Orientation.X = 0.000;
  - msgsGoalPub.Goal.TargetPose.Pose.Orientation.Y = 0.000;
  - msgsGoalPub.Goal.TargetPose.Pose.Orientation.Z = 0.001;
  - msgsGoalPub.Goal.TargetPose.Pose.Orientation.W = 1.000;
- Das Paket wird versendet:
  - send(goalPub, msgsGoalPub);

Des Weiteren muss der Befehl:

- rosservice call /move_base_node/clear_costmaps
  (Hier wurde die Shell-Terminal Eingabe benutzt und nicht MATLAB!)

alle paar Sekunden ausgeführt werden, da es sein kann, das die local_costmap den Pfad zwischen den Hindernissen blockiert. Das passiert aufgrund des AMCL-Programms, da der youBot sich versucht zu orientieren und bei der Drehbewegung in einer Richtung eine Verschiebung der local_costmap herbeiführt. Die kleine Verschiebung wird bei jeder weiteren Umdrehung immer größer, sodass es in dem gerade angesprochen Blockieren resultieren kann.

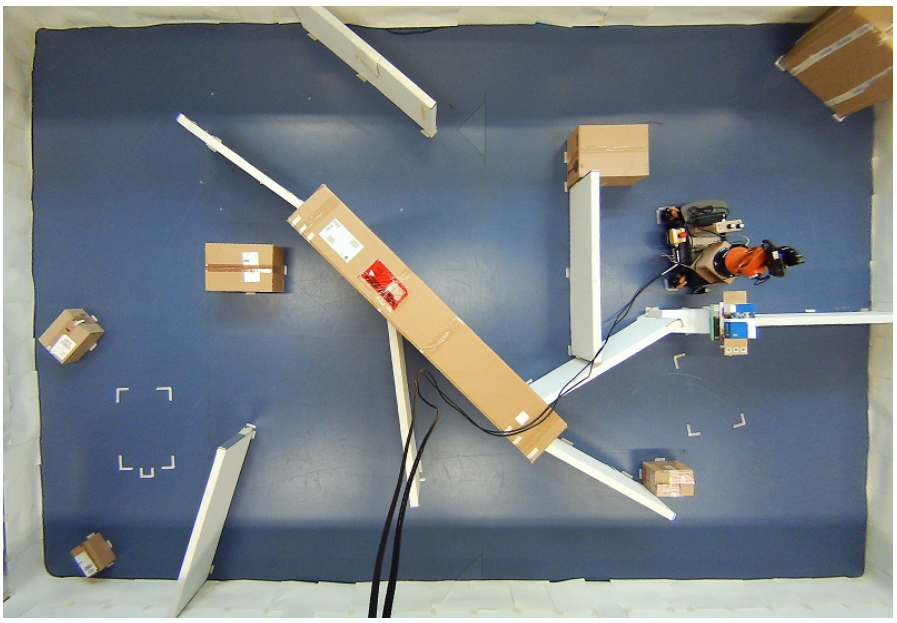


Abb. 4.37.: Ende linker Pfad (End-Position linker Pfad)

In Abbildung 4.37 wurde das Ziel erreicht, da der youBot auf der richtigen Bodenmarkierung steht.

Von den im Code erwähnten Zielkoordinaten sind lediglich die FrameId map, die Position X und Y sowie die Orientation Z und W wichtig, da die anderen Daten nicht benötigt werden, beziehungsweise gleich null sind. Im obigen Code sind sie nur der Vollständigkeit halber aufgelistet.
Die FrameId map gibt an, dass die Daten mit dieser Map/ Karte verglichen werden. Die Position X und Y geben die Raumkoordinaten an (Z bei dem youBot nicht möglich). Die Werte Z und W geben zusammen die Ausrichtung des youBots an.

5. Fazit

Im folgenden wird auf die Eingangsproblematik, die Probleme während der Umsetzung, die Kombination mit einer anderen Bachelorarbeit sowie zukünftige mögliche Arbeiten eingegangen.

5.1. Ergebnisse

Das Problem des langsamen, veralteten internen Computers wurde durch den Austausch mit einem neuen gelöst. Die Software wurde dementsprechend angepasst.
Das autonome Navigieren wurde an die neuste ROS-Version angepasst. Entsprechend der auf den neuesten Stand gebrachten Hard- und Software, ist es nun möglich alle Berechnungen beziehungsweise Programme, die in dieser Arbeit Verwendung finden, von dem neuen internen Computer ausführen zu lassen.

5.2. Probleme

Während der Umsetzung wurden folgende Probleme gelöst:

- Die Installation des youbot_navigation Treibers wurde durch die Einstellung von 41 Dateien angepasst, siehe Kapitel 4.2.2.
- Die move_base_global.launch Datei, beziehungsweise der Treiber für die autonome Bewegung, wurde nicht geupdatet und daher wurde ein zusätzlicher ros-kinetic-move-base Treiber verwendet, siehe Kapitel 4.2.3 und die dazugehörige Anwendung Kapitel 4.3.7.
- Der move_base Treiber benötigte einen anderen sensor_msgs Datentyp. Um diesen Datentyp zu generieren wurde der Ausgangsdatentyp des Laserscanners zweimal konvertiert, siehe Kapitel 4.3.4.

- Da die internen Akkus der youBots tiefen entleert sind, wurde dieser mittels Netzteil betrieben. Aufgrund dessen wurde darauf geachtet, dass der youBot seinen eigenen Strom- und HDMI-Kabel nicht überfährt beziehungsweise die Kabel nicht durch den Laserscanner aufgenommen werden.

5.3. Erweiterbar mit der Bachelorarbeit von Karsten Flores

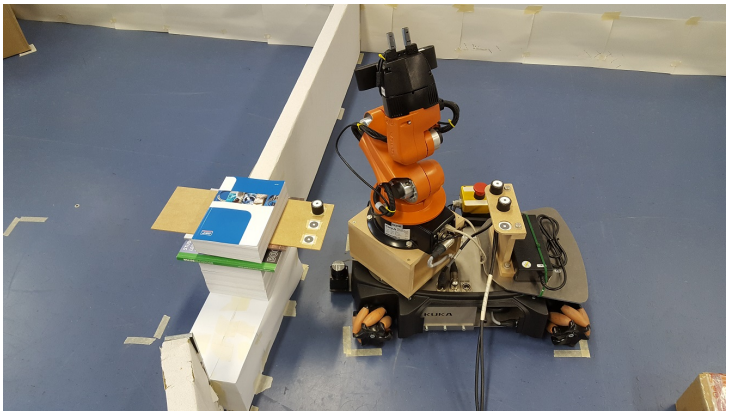


Abb. 5.1.: Kombination zweier Bachelorarbeiten

Die Bachelorarbeit von Karsten Flores behandelt das Thema der Erkennung zum Aufnehmen und Ablegen eines Werkstückes von und auf einer Taktstraße. Seine Arbeit konzentriert sich auf den youBot Arm. Durch das Zusammenbauen der Basis mit dem Arm wird es möglich beide Arbeiten gemeinsam auszuführen. Dies geschieht wenn die Befehle in Kapitel 4.3.7 alle gestartet werden und das jeweilige Ziel aus Kapitel 4.3.9 in dem MATLAB Programm von Karsten Flores aufgerufen wird. In Abbildung 5.1 ist der Ablage Ort dargestellt.

5.4. Ausblick

Da die Umsetzung des Umbaus erfolgreich war, werden die drei verbliebenen youBots der Fachhochschule, auf den Stand des youBots dieser Arbeit gebracht.

Diese Arbeit kann an folgenden Stellen verbessert werden:

- Das korrekte Einstellen der Pfade in der Bash (.bashrc).
  'export ROS_PACKAGE_PATH=...'
- Ein Script schreiben, das die einzeln auszuführenden Shell-Befehle automatisch ausführt.

- Einen move_base launcher schreiben, der ohne das vorherige Starten und Beenden des move_base Treibers, des youBots, richtig konfiguriert ist.
- Einen slam_gmapping launcher schreiben, der ohne das Laden der Parameter des youBot slam launchers auskommt.
- Einstellen der Zugriffsrechte des Hokuyo-Lasers, damit die Rechte nicht bei jedem Neustart wieder neu vergeben werden müssen, siehe Kapitel 4.3.6 und Kapitel 4.3.7.

Diese Arbeit kann zudem als Grundlage für wissenschaftliche Arbeiten oder Projekte dienen. Folgend sind einige Beispiele genant:

- Aufrüstung der Akkus durch den Umstieg von Blei- auf Lithium-Ionen-Akkus.
- Den Sichtbereich des youBots durch weitere Hokuyos, Kameras oder anderen Sensoren erweitern.
- Mehrere Roboter gleichzeitig in der Arena verfahren und zusammen Aufgaben bewältigen lassen.
- Das Verfahren des youBots in unbekannter Umgebung mit zu erreichenden definierten Zielen.
- Eine Objektverfolgung beziehungsweise Personenverfolgung programmieren.

Literaturverzeichnis

[cmt] CMTLAB: *Tested and Approved Modules.* http://www.cmtlabs.com/2012/memCertPartSearchResultsAll.asp?bNav=True&sManuf=Intel&outside=False&sMN=NUC7i3BNH%2FNUC7i3BNK&oSubmit=Search. – Zu letzt aufgerufen: 11.11.2017

[Con] CONVERTIO: *PGM in JPG Umwandeln. Online und Kostenlos.* https://convertio.co/de/pgm-jpg/. – Zu letzt aufgerufen: 11.11.2017

[DHo] DHOOD: *Ubuntu install of ROS Kinetic.* http://wiki.ros.org/kinetic/Installation/Ubuntu. – Zu letzt aufgerufen: 11.11.2017

[eit] EITAN: *Tuning Navigation - robot keeps rotating when close to its goal.* https://answers.ros.org/question/9795/tuning-navigation-robot-keeps-rotating-when-close-to-its-goal/). – Zu letzt aufgerufen: 11.11.2017

[FCMM15a] *Kapitel* The Navigation Stack - Robot Setups. In: FERNANDEZ, Enrique ; CRESPO, Luis S. ; MAHTANI, Anil ; MARTINEZ, Aaron: *Learning ROS for Robotics Programming.* 2. Packt Publishing Ltd., 2015, S. 303–334

[FCMM15b] *Kapitel* The Navigation Stack - Beyond Setups. In: FERNANDEZ, Enrique ; CRESPO, Luis S. ; MAHTANI, Anil ; MARTINEZ, Aaron: *Learning ROS for Robotics Programming.* 2. Packt Publishing Ltd., 2015, S. 335–365

[FCMM15c] *Kapitel* Using Sensors and Actuators with ROS. In: FERNANDEZ, Enrique ; CRESPO, Luis S. ; MAHTANI, Anil ; MARTINEZ, Aaron: *Learning ROS for Robotics Programming.* 2. Packt Publishing Ltd., 2015, S. 129–179

[Goe15a] *Kapitel* Controlling a Mobile Base. In: GOEBEL, R.Patrick: *ROS By Example.* 1. R.Patrick Goebel, 2015, S. 35–74

[Goe15b] *Kapitel* Navigation, Path Planning and SLAM. In: GOEBEL, R.Patrick: *ROS By Example.* 1. R.Patrick Goebel, 2015, S. 75–122

[Hoa] HoangGiang88: *Documentation*. http://wiki.ros.org. – Zu letzt aufgerufen: 11.11.2017

[Hok] Hokuyo: *Distance Data Output/URG-04LX-UG01*. https://www.hokuyo-aut.jp/search/single.php?serial=166. – Zu letzt aufgerufen: 11.11.2017

[Int] Intel: *Intel NUC Board NUC7i3BNB*. https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/NUC7i3BN_TechProdSpec.pdf. – Zu letzt aufgerufen: 11.11.2017

[Iva] IvanGavran: *base local planner*. http://wiki.ros.org/base_local_planner. – Zu letzt aufgerufen: 11.11.2017

[KUKa] KUKA: *Die KUKA Geschichte*. https://www.kuka.com/de-de/%C3%BCber-kuka/geschichte. – Zu letzt aufgerufen: 11.11.2017

[KUKb] KUKA: *YouBot Datenblatt*. http://www-home.htwg-konstanz.de/~bittel/LaborMobileRoboter/youBot_datenblatt_web_0514.pdf. – Zu letzt aufgerufen: 11.11.2017

[LEZ16] Lohrmann, Julia ; Eberhorn, Johannes ; Ziegler, Wiebke: Roboter. (2016), November. http://www.planet-wissen.de/technik/computer_und_roboter/roboter_mechanische_helfer/index.html

[Loca] Locomotec: *Geschichte*. http://www.locomotec.com/de/wir/. – Zu letzt aufgerufen: 11.11.2017

[Locb] Locomotec: *Tochterunternehmen*. http://www.locomotec.com/de/wir/. – Zu letzt aufgerufen: 11.11.2017

[Mar] Marguedas: *Installing and Configuring Your ROS Environment*. http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment. – Zu letzt aufgerufen: 11.11.2017

[Mata] MathWorks: *Design and test algorithms for robotics applications*. https://de.mathworks.com/products/robotics.html. – Zu letzt aufgerufen: 11.11.2017

[Matb] MathWorks: *MATLAB*. https://de.mathworks.com/help/matlab/index.html. – Zu letzt aufgerufen: 11.11.2017

[Mor] MoriKen: *move base*. http://wiki.ros.org/move_base. – Zu letzt aufgerufen: 11.11.2017

[Mot] Motion, Donkey: *Mecanum-Rad*. https://www.donkey-motion.de/donkeymotion/mecanum. – Zu letzt aufgerufen: 11.11.2017

[Now] Nowak, Walter: *Fwd: Re: youbot-driver-ros-interface für ROS-Kinetic*. – Sebastian.Flores@studmail.w-hs.de, E-Mail im Anhang A.

[Sch] Schnabel, Patrick: *Belegung der PC-Stromversorgungsstecker*. https://www.elektronik-kompendium.de/sites/com/0601151.htm. – Zu letzt aufgerufen: 11.11.2017

[Sei] Seidel, M.Sc. M.: *Der youBot*. http://www.robotto.ovgu.de/Team/Roboter.html#power. – Zu letzt aufgerufen: 11.11.2017

[Shaa] Shah, Vikrant: *amcl*. http://wiki.ros.org/amcl. – Zu letzt aufgerufen: 11.11.2017

[Shab] Shane, Loretz: *Distributions*. http://wiki.ros.org/Distributions. – Zu letzt aufgerufen: 11.11.2017

[son] sonictl: *How to Build a Map Using Logged Data*. http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData. – Zu letzt aufgerufen: 11.11.2017

[TFBD01] Thrun, Sebastian ; Fox, Dieter ; Burgard, Wolfram ; Dellaert, Frank: Robust Monte Carlo Localization for Mobile Robots. (2001), January. https://www.ri.cmu.edu/publications/robust-monte-carlo-localization-for-mobile-robots/. – Zu letzt aufgerufen: 11.11.2017

[ubu] ubuntuusers: *Was ist Ubuntu*. https://wiki.ubuntuusers.de/Was_ist_Ubuntu/. – Zu letzt aufgerufen: 11.11.2017

Anhang

A. E-Mail von Herrn Walter Nowak

"Betreff: **Fwd: Re: youbot_driver_ros_interface für ROSKinetic**
Absender: Olaf Just <olaf.just@w-hs.de>,
Empfänger: <karsten.flores@studmail.w-hs.de>,
<sebastian.flores@studmail.w-hs.de>,
Datum: 2017-06-22 11:27

−−−−−−−− Weitergeleitete Nachricht −−−−−−−−
Betreff: Re: youbot_driver_ros_interface für ROS-Kinetic
Datum: Thu, 22 Jun 2017 09:09:32 +0200
Von: Walter Nowak <nowak@locomotec.com>
An: Olaf Just <olaf.just@w-hs.de>

Sehr geehrter Herr Prof. Just,

die gute Nachricht ist, dass (unserer Erfahrung nach) die Software ohne Probleme mit ROS Kinetic läuft. Die Installation ist aber leider noch recht umständlich.

für Kinetic gibt es nämlich noch keine deb-Pakete zum installieren, die youBot Software muss daher über die Quellrepositories von github installiert werden. Das geht mit folgenden Kommandos.

Voraussetzung ist, dass Sie ROS kinetic installiert haben, am besten in der desktop-full Variante. Anschließend müssen Sie einen catkin workspace aufsetzen, s. http://wiki.ros.org/catkin/Tutorials/create_a_workspace

Unten bei den cd-Befehlen müssen Sie entsprechend den Pfad zu Ihrem catkin-Workspace einsetzen.

Danach führen Sie in einem Terminal aus:

sudo apt-get install ros-kinetic-pr2-msgs
sudo apt-get install ros-kinetic-controller-manager

cd ∼/(catkin-workspace)/src
git clone http://github.com/youbot/youbot_driver -b hydro-devel
git clone http://github.com/youbot/youbot_driver_ros_interface.git -b indigo-devel
git clone http://github.com/youbot/youbot_description.git -b jade-devel
git clone http://github.com/youbot/youbot_simulation.git
git clone http://github.com/wnowak/brics_actuator.git

cd ∼/(catkin workspace)
catkin_make
sudo setcap cap_net_raw+ep
devel/lib/youbot_driver_ros_interface/youbot_driver_ros_interface

Die Befehle zum starten etc. sind sonst alle genauso wie bei Hydro oder Indigo.

Ich hoffe es funktioniert bei Ihnen, wenn etwas nicht klappt geben Sie bitte Bescheid.

Es sollte auch irgendwann noch Pakete zum einfachen "apt-get install" instalieren geben, da kann ich jetzt aber noch keinen Zeitplan nennen.

Viele Grüße,
Walter Nowak

−−
Walter Nowak
youBot Store GmbH
Bergiusstr. 15
86199 Augsburg
Tel: +49 (0) 821 455159-702
Fax: +49 (0) 821 455159-700" [Now]

B. youBot-Treiber Ergänzung

Auflistung der 41 zu ändernden Dateien.
Einstellen der Pfade mittels '</opt/ros/kinetic/include/'.

- ~/catkin_ws/src/youbot_navigation/youbot_navigation_common/src:
  - lowpass_filter.cpp
- /opt/ros/kinetic/include/ros:
  - ros.h
  - time.h
  - rostime_decl.h
  - rate.h
  - console.h
  - assert.h
  - common.h
  - forwards.h
  - exceptions.h
  - roscpp_serialization_macros.h
  - node_handle.h
  - publisher.h
  - message.h
  - serialization.h
  - message_traits.h
  - builtin_message_traits.h
  - subscriber.h
  - subscription_callback_helper.h
  - parameter_adapter.h
  - message_event.h
  - service_server.h
  - service_client.h

  - timer_options.h
  - wall_timer_options.h
  - advertise_options.h
  - advertise_service_options.h
  - service_callback_helper.h
  - subscriber_options.h
  - transport_hints.h
  - service_client_options.h
  - spinner.h
  - init.h
  - single_subscriber_publisher.h
  - service.h
  - master.h
  - param.h
- /opt/ros/kinetic/include/xmlrpcpp:
  - XmlRpcValue.h
  - XmlRpcDecl.h
- /opt/ros/kinetic/include/geometry_msgs:
  - Twist.h
  - Vector3.h

C. Kompletter Code der Launch-Dateien

```
<?xml version="1.0"?>
<launch>
<!-- Set relevant parameters. -->
<param name="youBotHasBase" type="bool" value="true"/>
<param name="youBotHasArms" type="bool" value="false"/>
<param name="youBotDriverCycleFrequencyInHz" type="double" value="50.0"/>
<param name="youBotConfigurationFilePath" type="string" value="$(find youbot_driver)/config"/>

<param name="trajectoryActionServerEnable" type="bool" value="true"/>
<param name="trajectoryPositionGain" type="double" value="5.0"/>
<param name="trajectoryVelocityGain" type="double" value="0.0"/>

<!-- Default name values -->
<param name="youBotBaseName" type="str" value="youbot-base"/>
<param name="youBotArmName1" type="str" value="youbot-manipulator"/>


<!-- Start the driver. NOTE: Every joint topic is mapped to armName/joint_states -->
<node name="youbot_driver" pkg="youbot_driver_ros_interface"
        type="youbot_driver_ros_interface" output="screen">
        <remap from="base/joint_states" to="/joint_states"/>
        <remap from="arm_1/joint_states" to="/joint_states"/>
</node>

<!-- upoad URDF model to the parameter server -->
<param name="robot_description" command="$(find xacro)/xacro.py '$(find youbot_description)/
        robots/youbot.urdf.xacro'"/>

<!-- start robot_state_publisher -->
<node pkg="robot_state_publisher" type="state_publisher" name="robot_state_publisher"
        output="screen"/>
</launch>
```

Abb. C.1.: youbot_driver.launch

In der Abbildung C.2 ist im roten Kasten der hinzugefügte Code zu sehen.

```
<launch>
<node pkg="amcl" type="amcl" name="amcl">
  <!-- Publish scans from best pose at a max of 10 Hz -->
  <param name="odom_model_type" value="omni"/>
  <param name="odom_alpha5" value="0.1"/>
  <param name="transform_tolerance" value="0.2" />
  <param name="gui_publish_rate" value="1.0"/>
  <param name="laser_max_beams" value="30"/>
  <param name="min_particles" value="500"/>
  <param name="max_particles" value="5000"/>

  <!-- Startpunkt des YB in der Map Richtig setzten! -->
  <param name="initial_pose_x" value="0.8"/>
  <param name="initial_pose_y" value="-1.97"/>
  <param name="initial_pose_a" value="-0.03"/>

  <param name="kld_err" value="0.05"/>
  <param name="kld_z" value="0.99"/>
  <param name="odom_alpha1" value="0.2"/>
  <param name="odom_alpha2" value="0.2"/>
  <!-- translation std dev, m -->
  <param name="odom_alpha3" value="0.8"/>
  <param name="odom_alpha4" value="0.2"/>
  <param name="laser_z_hit" value="0.5"/>
  <param name="laser_z_short" value="0.05"/>
  <param name="laser_z_max" value="0.05"/>
  <param name="laser_z_rand" value="0.5"/>
  <param name="laser_sigma_hit" value="0.2"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_model_type" value="likelihood_field"/>
  <!-- <param name="laser_model_type" value="beam"/> -->
  <param name="laser_likelihood_max_dist" value="2.0"/>
  <param name="update_min_d" value="0.2"/>
  <param name="update_min_a" value="0.5"/>
  <param name="odom_frame_id" value="odom"/>
  <param name="resample_interval" value="1"/>
  <param name="transform_tolerance" value="0.1"/>
  <param name="recovery_alpha_slow" value="0.0"/>
  <param name="recovery_alpha_fast" value="0.0"/>
</node>
</launch>
```

Abb. C.2.: amcl_sebe.launch

D. Zusätliche Einstellungen von Parametern

```
# Robot Configuration Parameters
TrajectoryPlannerROS:
    acc_lim_x: 1.00
    acc_lim_y: 1.00
    acc_lim_theta: 1.2
    max_vel_x: 0.3
    min_vel_x: 0.1
    max_vel_theta: 0.7
    min_vel_theta: -0.7
    min_in_place_vel_theta: 0.1
    escape_vel: -0.1
    holonomic_robot: true

    # Goal Tolerance Parameters
    yaw_goal_tolerance: 0.05
    xy_goal_tolerance: 0.1

    # Forward Simulation Parameters
    sim_time: 1.7
    sim_granularity: 0.025
    vx_samples: 3
    vtheta_samples: 6

    # Trajectory Scoring Parameters
    pdist_scale: 0.6
    gdist_scale: 0.8
    occdist_scale: 0.01
    heading_lookahead: 0.325
    dwa: false

    # Oscillation Prevention Parameters
    oscillation_reset_dist: 0.01
```

Abb. D.1.: base_local_planner.params

```
image: Arena_Pfad_Final.pgm
resolution: 0.020000
origin: [-5.000000, -5.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Abb. D.2.: arena_pfad.yaml

Abbildung D.1 zeigt die Einstellung von zum Beispiel der Geschwindigkeit des youBots. Die Abbildung D.2 zeigt die Daten, die der map_server benötigt zum streamen der map.

In Abbildung D.3 sind die Parameter der costmap abgebildet, zum Beispiel sagt der inflation_radius: 0.15 aus, das die Hindernisse um einem Radius von 0,15m aufgeblasen werden.

```
map_type: costmap

#Set the tolerance we're willing to have for tf transforms
transform_tolerance: 0.2

#Obstacle marking parameters
obstacle_range: 2.5
raytrace_range: 3.0

#Cost function parameters
# included in both global and local costmaps params
inflation_radius: 0.15

#Configuration for the sensors that the costmap will use to update a map
# included in both global and local costmaps params
observation_sources: base_scan

#base_scan_marking: {sensor_frame: base_scan_link,
#                    data_type: PointCloud2,
#                    topic: /base_scan/marking,
#                    expected_update_rate: 0.2,
#                    observation_persistence: 0.0,
#                    marking: true,
#                    clearing: false,
#                    min_obstacle_height: 0.06,
#                    max_obstacle_height: 2.0}

base_scan: {sensor_frame: base_laser_front_link,
            data_type: PointCloud,
            topic: /points_out_sebe,
            expected_update_rate: 0.2,
            marking: true,
            clearing: true,
            min_obstacle_height: -0.10,
            max_obstacle_height: 2.0}
```

Abb. D.3.: costmap_common.params

```
Window Geometry:
  Displays:
    collapsed: false
  Height: 650
  Hide Left Dock: false
  Hide Right Dock: true
  QMainWindow State:
000000ff00000000fd00000
  Selection:
    collapsed: false
  Time:
    collapsed: false
  Tool Properties:
    collapsed: false
  Views:
    collapsed: true
  Width: 1000
  X: 90
  Y: 50
```

Abb. D.4.: youbot_sebe.rviz

Damit RViz immer in der gleichen definierten Größe und Position startet wurde folgendes eingestellt:

- Height: 650
- Width: 1000
- X: 90
- Y: 50

Abbildung D.4 zeigt lediglich einen kleinen Teil der Konfig-Datei.

E. Erstellung der Karte

Aufnahmen während der Erstellung der Karte.

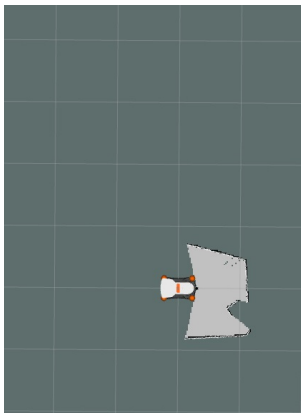


Abb. E.1.: Mapping 1

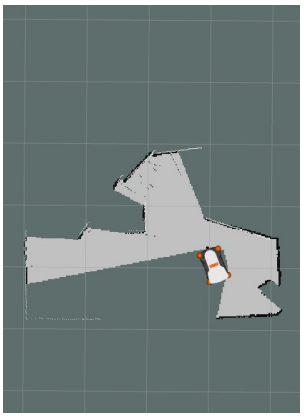


Abb. E.2.: Mapping 2

Abb. E.3.: Mapping 3

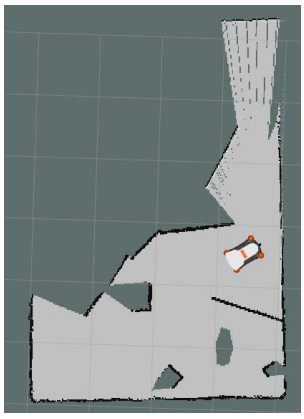


Abb. E.4.: Mapping 4

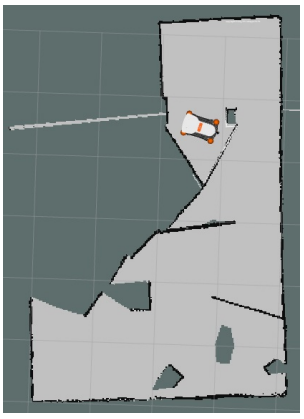


Abb. E.5.: Mapping 5

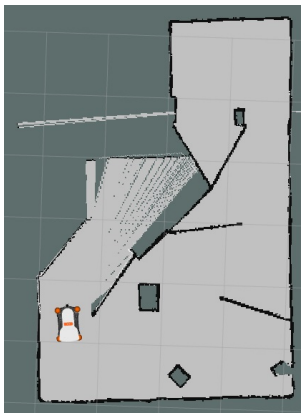


Abb. E.6.: Mapping 6

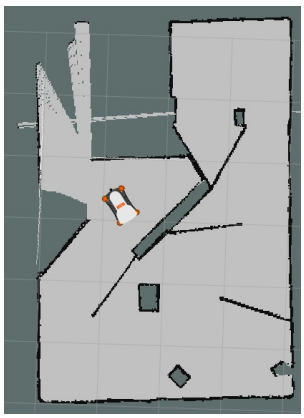


Abb. E.7.: Mapping 7

Abb. E.8.: Mapping 8

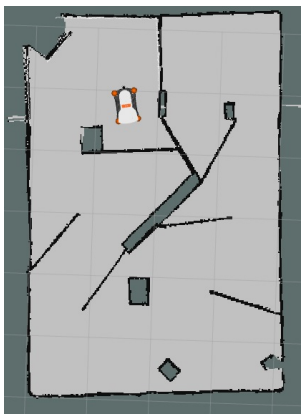


Abb. E.9.: Mapping 9

F. Starten von urg_node und map_server

```
youbot-02@NUC-02:~$ sudo chmod a+rw /dev/ttyACM0
[sudo] Passwort für youbot-02:
youbot-02@NUC-02:~$ rosrun urg_node urg_node _frame_id:="base_laser_front_link"
_angle_min:=-1.7978254834 _angle_max:=1.7978254834
[ INFO] [1509007076.128644022]: Connected to serial device with ID: H1620222
[ INFO] [1509007077.081492412]: Streaming data.
```

Abb. F.1.: urg_node

```
youbot-02@NUC-02:~$ rosrun map_server map_server Arena_Pfad_Final.pgm
youbot-02@NUC-02:~$ rosrun map_server map_server Arena_Pfad_Final.yaml
[ INFO] [1509007094.232997130]: Loading map from image "./Arena_Pfad_Final.pgm"
[ INFO] [1509007094.235379867]: Read a 500 X 500 map @ 0.020 m/cell
[ INFO] [1509007101.175664027]: Sending map
```

Abb. F.2.: map_server