CARL VON OSSIETZKY universität OLDENBURG

DEPARTMENT FÜR INFORMATIK
SYSTEMSOFTWARE UND VERTEILTE SYSTEME

Entwurf und Implementierung einer Lokalisierungsstrategie für autonome Rasenmähroboter

Bachelorarbeit

23. Februar 2017

Christian Brunzendorf
Wilhelm-Busch-Str. 8
26842 Ostrhauderfehn

Erstprüfer	Prof. Dr.-Ing. Oliver Theel
Zweitprüfer	M.Sc. Robert Schadek

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 23. Februar 2017

Christian Brunzendorf

In dieser Bachelorarbeit wurde ein Mähroboter um die Fähigkeit erweitert, sich zu Lokalisieren. Dazu wurde ein stabilisierter Entfernungsscanner entworfen dessen Sensordaten zur Kartengenerierung und Lokalisierung dienten. Dabei wurde auf das Framework Robot Operating System zurückgegriffen, welches die erforderlichen Methoden für den SLAM- und Lokalisierungs-Prozess zur Verfügung stellte.

In this Bachelor thesis, a mowing robot was expanded to the ability to locate. For this purpose, a stabilized distance scanner was designed whose sensor data were used for map generation and localization. The framework Robot Operating System, which provided the necessary methods for the SLAM and localization process, was used.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die Orientierung in der Umwelt spielte für den Menschen schon immer eine große Rolle. So sind unsere Urahnen von einem Ort zum nächsten gewandert in der Hoffnung neues fruchtbares Land zu finden. Als Orientierungshilfe hatten sie keine technischen Geräte zur Verfügung, wie das heute der Fall ist, um von einem Punkt zum anderen zu finden. Naheliegende Ziele konnten sie mithilfe markanter Punkte in der Natur erreichen und weiter entfernte Ziele mithilfe der Sonne bzw. Sterne als Richtungshilfe. So hat die Menschheit es geschafft über Jahrtausende die Welt zu bevölkern. Allein die Möglichkeit, dass unser Gehirn dazu fähig ist zu kombinieren ermöglicht dem Menschen das Zurechtfinden in der Umwelt. Dem Menschen stehen bekanntlich die fünf Sinne Sehen, Schmecken, Richen, Hören und Tasten zur Verfügung. Jeder Sinn gekapselt von den restlichen würde eine Orientierung unmöglich machen und uns in die Irre führen. Jedoch bewirkt das Kombinieren von bereits gesammelten Erfahrungen und die eingehenden Informationen aus den Sinnesorganen, dass wir uns in einer bekannten oder unbekannten Umgebung orientieren können.
Mit dem Fortschreiten des Entwicklungsstandes der Menschheit sind nach und nach Hilfsmittel erfunden worden, die die Orientierung erleichterten. Es wurden Erfahrungen und Entdeckungen kartographiert. Die Erfindung des Kompasses oder Sextanten haben die Navigation revolutioniert. Das sogenannte Computerzeitalter hat letztendlich zu unserem heutigen Lebensstandard beigetragen. Die Raumfahrt hat wohl das bekannteste Navigationssystem, das Global Positioning System (GPS) über Satelliten bereitstellt, welches eine ca. zehn Meter genaue Bestimmung der Positionierung ermöglicht. Mit zusätzlichen Erweiterungen[1] kann dieses System genauer gemacht werden.
Eine Vielzahl von Maschinen werden zur Zeit entwickelt, die die Aufgabe haben dem Benutzer die täglich anfallenden Arbeiten im Haushalt abzunehmen. Dies hat schon mit Waschmaschinen und Geschirrspülern funktioniert und jetzt sollen die Geräte autonom und mobil durch den Haushalt bzw. Garten navigieren. Waschmaschinen, Geschirrspüler und andere Haushaltsgeräte haben viele Aufgaben übernommen und sie werden zunehmend intelligenter. Unter dem neuen Oberbegriff Smart Home versteht man ein vernetztes Zuhause, das informations- und sensortechnisch aufgerüstet ist. Ziel dieser Vernetzung ist es nach Wirtschaftslexikon [2016], die Lebens- und Wohnqualität, Betriebs- und Einbruchsicherheit und Energieeffizienz zu erhöhen. In den letzten Jahren unterstützen neben den genannten stationären Maschinen auch mobile autonome Roboter im Haushalt. Ein weit verbreiteter und bekannter Haushaltroboter ist der Saugroboter, der das Staubsaugen übernehmen soll. Im Gegensatz zu den stationären Maschinen müssen sich die Roboter in ihrer Umgebung bewegen, um die ihnen gestellten Aufgaben zu erledigen.
Die Wahrnehmung seiner Umgebung ist für einen Roboter ebenso wichtig, wie für den Menschen. Je mehr Sinne[2] ihm zur Verfügung stehen, deren Informationen er auswerten und kombinieren kann, desto besser kann er sich in einer Umgebung orientieren. Sensoren werden in aktive und passive Sensoren unterteilt. Der Unterschied zwischen den beiden Gruppen besteht darin, wie eine Messung ausgeführt wird. Bei ak-

[1] siehe dazu Abschnitt 3.5
[2] technisch: Sensoren

tiven Sensoren wird eine Energie an die Umgebung abgegeben und eine möglich entstandene Reaktion ausgewertet. Passive Sensoren hingegen benutzen eine von außen einwirkende Hilfsenergie, um eine Information dieser Hilfsenergie zu erzeugen. Neben den Sensoren stehen einem Roboter meist auch Aktoren bzw. Effektoren zur Verfügung, die es ihm erlauben mit der Umwelt zu interagieren. Diese wandeln ein elektrisches Signal in eine physikalische Energie um. Als Zentrales Element wird ein Controller benötigt, welcher die Informationen der Sensorik auswertet und kombiniert sowie das Steuern bzw. Regeln der Aktorik vornimmt.

In Kapitel 3 *Grundlagen* wird hierzu das Thema Informationsbeschaffung und -verarbeitung näher beschreiben.

Bevor der Controller überhaupt etwas macht, muss aus einer erstellten Problembeschreibung eine passende Lösung implementiert werden. Weil die Problembeschreibung ein essentiell wichtiger Punkt ist, befasst sich das nachfolgende Kapitel 2 *Motivation* mit der Erarbeitung dieser.

In dem Wettbewerb *RoboCup Soccer* wird sich nach Asada u. a. [2016] unter anderem mit der Selbstlokalisierung befasst. Dort treten zwei Teams mit maximal sechs Robotern in einem Fußballspiel gegeneinander an. Die genaue Selbstlokalisierung ist dort ein wichtiger Bestandteil um erfolgreich am Turniergeschehen mitwirken zu können. Die Wahl der Methode zur Selbstlokalisierung bleibt dabei jedem Team selbst überlassen. Je nach Umgebung und Sensorik existieren bereits viele Methoden zur Selbstlokalisierung. Dieses Themengebiet spielt bei der Problembeschreibung eine wichtige Rolle und wird ebenfalls im Kapitel 2 eingeführt und in späteren Kapiteln weiter spezifiziert.

Diese Arbeit befasst sich mit der Selbstlokalisierung eines Rasenmähroboters, der in der Diplomarbeit Lübke [2014] konstruiert wurde. Im Gegensatz zu den Saugrobotern, die in einem Haushalt agieren, muss ein Rasenmähroboter mit weiteren Problemen seiner Umgebung zurechtkommen. Diese wären z.B. die Beschaffenheit des Untergrundes, Grenzen der Umgebung, Witterungsverhältnisse sowie höhere Sicherheitsanforderungen.

Die schon zuvor erwähnte Implementierung des Controllers wird an dem Framework[3] Robot Operating System (ROS) ansetzen. Dieses bietet bereits eine ausgereifte Strukturierung sowie spezielle Erweiterungen, die die Entwicklung der Betriebssoftware für den Rasenmähroboter vereinfachen.

Diese Arbeit umfasst zusammen mit der Einleitung sechs Kapitel, dessen Inhalt sich wie folgt beschreiben lässt. Als Grundlage für diese Arbeit, wurde eine Problembeschreibung in Kapitel 2 erarbeitet aus der die Aufgabenstellung resultierte. Dabei befasst sich der Abschnitt 2.2 *Stand der Technik* damit, wie das Verhalten der heutigen Generation von Rasenmähroboter aussieht. Im darauffolgenden Kapitel 3 wurden Begriffe und Methoden eingeführt, die im weiteren Verlauf dieser Arbeit benutzt wurden. Einen in Bezug auf die Problemstellung eingeschränkten Einblick in die Sensorwelt bietet der Abschnitt 3.5. Das Kapitel 4 befasst sich schließlich mit der Umsetzung. In den einzelnen Abschnitten wurden zum einen die hard- und softwareseitigen Änderungen bezüglich der Ausgangskonfiguration beschrieben und zum anderen auf die zentralen Punkte dieser Arbeit, der Kartierung einer Umgebung und der Lokalisierung, eingegangen. Mit der Evaluation des erstellten Systems befasst sich das Kapitel 5 und anhand von ausgewerteten Sensordaten wurde die Genauigkeit der Modelle verifiziert und Schwachstellen der gewählten Konfiguration aufgezeigt. In dem Kapitel 6 wird diese Arbeit ihren Abschluss finden. Es beinhaltet eine, aus den vorher erarbeiteten

[3] Als Framework wird eine Grundstruktur bezeichnet, die dem Entwickler gewisse Softwarebausteine zur Nutzung bereitstellt.

Aussagen und gewonnenen Informationen, verfasste Zusammenfassung. Mögliche offene Punkte, die in weiteren Arbeiten ausgearbeitet werden können, sind mitaufgenommen worden.

2. Motivation

Mähroboter sind heutzutage keine Seltenheit mehr und können in vielen Vorgärten beobachtet werden. Schaut man etwas genauer hin, ist die Art und Weise mit der eine Fläche gemäht wird nicht effizient. Die meisten Modelle fahren zufällig auf einem Grundstück herum. Dadurch wird jedoch erst nach mehrmaligem Wiederholen eines Mähvorgangs die komplette Rasenfläche bewirtschaftet. Um diese Arbeitsweise schneller, ressourcensparender und verschleißarmer werden zu lassen, muss der Roboter intelligenter arbeiten. Diese Intelligenz setzt voraus, dass der Roboter weiß wo er sich befindet und ist das Thema dieser Arbeit.

2.1. Problembeschreibung

Angenommen der Roboter verfügt über eine Möglichkeit sich auf dem Grundstück zu orientieren und seine Position zu bestimmen (Localization), dann wäre dieser in der Lage eine optimale Strategie zum Mähen zu suchen (Planning). Der Roboter kann somit das Grundstück schnellstmöglich abfahren. Die Vorteile, die dadurch entstehen würden, sind zum einen die vorher angesprochenen Ressourcen (z.B Batterie) zu sparen und zum anderen wäre der Verschleiß erheblich geringer.
Was macht es so schwer eine so einfach klingende Tätigkeit wie das Rasenmähen intelligent und autonom von Robotern erledigen zu lassen? Voraussetzung für das intelligente Agieren eines Roboters ist die exakte Erfassung seiner Umwelt. Der Roboter muss seine Umgebung kennen und auf Veränderungen reagieren können. Durch verschiedene Sensoren, die dem Roboter zur Verfügung gestellt werden müssen, kann dieser eine Karte anlegen und sein Bewegungsmuster in ihr berechnen. Diese Sensoren wären z.B. Magnetometer, Gyroskop, Ultraschall, Infrarot, Laser, Druck, Odometrie und vieles mehr. Jeder dieser Sensoren liefert dem Roboter einen bestimmten Ausschnitt seiner Umwelt. Leider, und da liegt das eigentliche Problem, sind die gewonnenen Sensordaten mit Fehlern behaftet. Jeder noch so kleine Fehler sorgt über die Zeit dafür, dass eine genaue Bestimmung der Position nicht möglich ist. Ein Grund für die Ungenauigkeit eines Sensors wäre z.B. der Einfluss von Störungen auf ihn. So fängt das Gyroskop an zu Driften; die Odometrie bekommt es nicht mit, wenn Räder durchdrehen und Ultraschall empfängt nur Echos, wenn der Winkel zu einem Gegenstand stimmt (Stealth).
Mithilfe einer softwareseitigen Fusion verschiedener Sensorinformationen oder durch den Einsatz von Filtern beruhend auf mathematischer Gleichungen, können einige Fehlerentwicklungen eliminiert werden. Dadurch werden die Daten zuverlässiger und die Auswirkungen eines Drifts des Gyroskops werden verringert.
Die Aufgabenstellung lässt sich anhand der exemplarisch genannten Probleme wie folgt formulieren.

> Der Mähroboter soll in der Lage sein, seine Umgebung zu erkennen und sich diese durch eine geeignete Speicherstruktur zu merken.

Mithilfe der Sensorik kann der Roboter nun seine Umwelt wahrnehmen und sie speichern. Ein exaktes Navigieren ist unter der Voraussetzung möglich, dass der Roboter seine genaue Position kennt. Mittels Partikelfilter ist es möglich, die wahrscheinlichste Position zu bestimmen. Diese Positionsbestimmung wird anhand von Messwerten und auf Grundlage der zuvor generierten Umgebungskarte ermittelt. Dies ist sehr rechenintensiv. Wie genau diese Lokalisierung ist, hängt von der Genauigkeit des Sensormodels ab und muss durch Testergebnisse verifiziert werden. Da ein Mähroboter meist nur von einem Haushalt benutzt wird, kann die rechenintensive Anwendung von Partikelfiltern durch Positionsinformationen von außen (z.B. Real Time Kinematik (RTK)-GPS oder Triangulation von Bluetooth) unterstützt werden.
Die Aufgabenstellung wird somit um den folgenden Sachverhalt erweitert.

> Der Mähroboter kann selbstständig seine Position in einer bekannten Umgebung ermitteln.

Ist dem Mähroboter nun seine Umgebung bekannt und das Verfolgen seiner Bewegungen in ihr möglich, kann durch intelligente Anwendung von Planungs-Algorithmen die Optimale Route in ihr gefunden werden.
Somit wird die Aufgabenstellung dieser Bachelorarbeit mit dem letzten Absatz vervollständigt und ist damit auch gleichzeitig die **Anforderung** an das System.

> Der Mähroboter kann in einer bekannten Umgebung eine optimale Route finden, die zum gewünschten Ergebnis zu führt.

2.2. Stand der Technik

Die Diplomarbeit von Lübke [2014] die dieser Bachelorarbeit zugrunde liegt wurde 2014 erstellt. Bekanntlich sind zwei Jahre in der Technik heutzutage eine lange Zeit und es hat sich vieles in dieser Sparte getan. Laut der Pressemitteilung Rehm u. Röhr [2016] wuchs der Absatz von Mährobotern im ersten Halbjahr 2016 um 37,2 %[1]. Neue Hersteller sind mit eigenen Modellen dazugekommen und gestandene Hersteller haben ihre Modelle weiterentwickelt. Zudem existieren auch diverse Hobbyentwickler, die sich zusammengeschlossen haben, um Open Source Lösungen anzubieten, wie zum Beispiel Ardumower[2]. Dadurch wird es jedem ermöglicht seinen eigenen Rasenmähroboter zu bauen, zu konfigurieren und weiterzuentwickeln.
Nachfolgend werden 3 aktuelle Vertreter von kommerziellen Rasenmähroboter vorgestellt. Um einen Vergleich gegenüber der Abbildung C.1 von 2014 zu erhalten, wurden der Robowmow MC800, Bosch Indego 1000 Connect und Worx Landroid WG756E.1 als aktuelle Modelle ausgewählt. Der Vollständigkeit halber sollte erwähnt werden, dass neben den ausgewählten Vertretern zudem noch die namenhaften Hersteller wie Gardena, Husquarna, Ambrogio, Wolf und Wiper/Rumsauer u.v.m. existieren, die ebenfalls teils sehr erfolgreich mit ihren Produkten am Marktgeschehen mitwirken.

[1] Statistik bezieht sich auf die Länder Großbritanien, Deutschland, den Niederlanden, Frankreich und Belgien
[2] Ardumower ist ein Do It Yourself (DIY) Open Source Projekt, dass Hobbybastler den Eigenbau eines Mähroboters durch bereits entwickelte Hard- und Software einfacher macht http://www.ardumower.de/index.php/de/

(a) Robomow MC800 (b) Bosch Indego 1000 Connect (c) Worx Landroid WG756E.1

Abbildung 2.1.: aktuelle erhältliche Modelle

2.2.1. Robomow MC800

Laut Hersteller Robomow besitzt der Robomow MC800, aus Abbildung 2.1(a), eine Schnittbreite von 28 cm und die Besonderheit dieses Produktes soll in dem Kantenmodus liegen. Die maximale Größe der zu mähenden Fläche beträgt 800 m². Das sind 200 m² mehr als das Vergleichsmodell aus Abbildung C.1 aufweist. Verbessert hat sich unteranderem auch der Geräuschpegel von 77db auf 69 db. Eine intelligente Navigation wird nicht erwähnt, was den Rückschluss erlauben lässt, dass sie nicht vorhanden ist. Die Arbeitsweise hat sich zu der, wie sie noch in Lübke [2014] beschrieben wurde, nicht geändert.

2.2.2. Bosch Indego 1000 Connect

Das Modell vom Hersteller Bosch war 2014 das einzige, von den in Abbildung C.1 aufgeführten Modellen, welches versucht hatte, das Grundstück Bahn-für-Bahn zu mähen. In dem Testbericht Jäger [2014] schnitt dieses Exemplar jedoch nicht gut ab. Als Grund hierfür wurde die sehr lange Dauer beim Mähen angegeben. Das Navigationssystem „Logicut“ bringt den Roboter immer wieder zum stehen, um die Sensoren neu zu Kalibrieren und interne Berechnungen durchzuführen. Verwinkelte Grundstücke und enge Passagen sind ebenfalls K.O.-Kriterien für das Navigationssystem.

Der Bosch Indego 1000 Connect (siehe Abbildung 2.1(b)) Rasenroboter ist die momentan aktuelle Version, die seit 2015 auf dem Markt zur Verfügung steht. Der Roboter wurde um eine mobile App Steuerung erweitert, was den Namenszusatz Connect erklärt. Unter anderem wurde die Software weiterentwickelt. Die genaue Funktionsweise der Kartierung wird von Bosch leider nicht offengelegt. Bekannt aus der Arbeitsweise dieses Systems ist jedoch, dass dieser die äußere Begrenzungsschleife bei der Erstinstallation abfährt, um die interne Karte anzulegen. Dabei werden innere Hindernisse[3], die ebenfalls durch die Begrenzungsschleife gekennzeichnet werden müssen, in den darauffolgenden zwei Mähzyklen kartographiert. Bosch ist mit dem Indego auf einem guten Weg, ein Grundstück schnell und effizient abarbeiten zu können. Jedoch wird eine gewisse Struktur der Rasenfläche vorausgesetzt, damit Logicut ohne Probleme arbeiten kann.

[3] innere Hindernisse werden auch Inseln genannt

2.2.3. Worx Landroid WG756E.1

Laut Herstellerangaben Worx soll der Worx Landroid Rasenmäher (siehe Abbildung 2.1(c))im Vergleich zu anderen Rasenmäher Roboter aufgrund seines Artificial Intelligence Algorithm (AIA) 30 % schneller fahren. Mit deren Hilfe soll er in der Lage sein, Entscheidungen zu treffen und sich individuell auf den Rasen einzustellen. Im Gegensatz zum Bosch Indego versucht der Landroid keine Bahnen zu fahren, sondern mäht den Rasen mit dem Zufallsprinzip. Der Unterschied in der Navigation zu herkömmlichen Mährobotern verbirgt sich hinter dem AIA. Dieser verarbeitet in seiner Logik den Winkel, indem der Roboter auf das Begrenzungskabel trifft bzw. verlässt sowie den zurückgelegten Weg bis zum nächsten treffen der Begrenzungsschleife. Durch das Messen des Signals, welches durch die Begrenzungsschleife gesendet wird, kann der Landroid seine Ausrichtung feststellen.

2.3. Zusammenfassung

Der Vergleich zeigte, dass die Hersteller versucht sind die Effizienz ihrer Produkte zu verbessern. Bosch war der einzige Hersteller, der mit seinem Produkt die Linienfahrt dem Zufallsprinzip vorzog. Dabei wurden aber gewisse Voraussetzungen an die Form der Mähfläche gestellt. In dieser Arbeit sollte der Roboter mit mehr Informationen aus seiner Umgebung versorgt werden, damit der Rasen schneller gemäht werden konnte und kein besonderer Anspruch an die Form Grundstück gestellt werden musste. Zudem sollte eine interne Repräsentation der Umgebung dem Roboter zur Verfügung stehen, um sein Mähzyklus zu planen.

3. Grundlagen

In dem nachfolgendem Kapitel werden die wesentlichen Grundlagen, auf die diese Arbeit aufbaut, vorgestellt und die im vorherigen Kapitel erarbeitete Aufgabenstellung soll aufbauend auf diese erfüllt werden. Wie schon in Kapitel 2 beschrieben, wird zur Orientierung eine Karte der Umgebung benötigt. Abschnitt 3.1 befasst sich damit, wie diese Karte erstellt werden kann und welche Probleme dabei auftreten können. Sobald eine Karte verfügbar ist kann der Roboter anhand dieser und mithilfe eingehender Informationen seine Position bestimmen. Um diese Positionsbestimmung besser verstehen zu können wird im Abschnitt 3.2 thematisiert, welche Arten von Probleme bei der Lokalisierung existieren. Für die Lösung dieser Lokalisierungs-Probleme existieren eine Vielzahl von verschiedenen Ansätzen, die in Abschnitt 3.3 grundlegend kategorisiert werden. Auf das Thema bezogene Lokalosierungsmethoden runden den Abschnitt ab. Abgeschlossen wird dieses Kapitel mit einer Auswahl an Sensoren, die es ermöglichen den zuvor genannten Methoden mit Informationen aus der Umgebung zu versorgen.

3.1. Simultanious Localisation and Mapping

Ein großes Problem in der Robotik ist das Simultaneous Localization and Mapping (SLAM)-Problem, denn ein Roboter benötigt für die Lokalisierung in seiner Umwelt eine Karte. Damit diese Karte erzeugt werden kann, wird unteranderem jedoch die Position des Roboters benötigt. Dies ist ein Widerspruch und ist vergleichbar mit dem Henne-Ei-Problem. Um dieses Problem lösen zu können, existieren verschiedene Ansätze. Bezogen auf diese Arbeit werden nachfolgend zwei dieser Ansätze behandelt.
Der erste Ansatz ist das *scan matching* und nach Gutmann [1999], werden eingehende Laserscans mit einem Referenzmodell verglichen. Dabei wird dieser Scan gedreht und verschoben, um eine bestmögliche Übereinstimmung zu finden. Durch diesen Prozess kann die Position des Roboters bestimmt werden. Durch die Tatsache, dass dieses Verfahren nur Scans mit Referenzen vergleicht, um die Position zu bestimmen, macht es zu einer guten Lösung für mobile Roboter ohne Odometriedaten. Vorraussetzung ist allerdings, dass der Laserscan mit genügend Merkmalen des Referenzmodells übereinstimmt.
Das zweite Verfahren ist der Rao-Blackwellized Particle Filter (RBPF). Dieser baut auf einen Partikelfilter auf wobei von jedem Partikel eine eigene individuelle Karte repräsentiert wird Grisett u. a. [2005]. Die Partikel repräsentieren jeweils die Bewegungsverlauf des Roboters in ihrer Karte. Diese werden mittels eines Zustandsübergangsmodells bewegt und einer Observationsfunktion bewertet. Grzonka [2006]

3.2. Arten der Lokalisierung

Wie in Abschnitt 3.1 beschrieben, ist eine Lokalisierung nur möglich, wenn der Roboter über eine Karte seiner Umgebung verfügt. Liegt ihm diese Karte vor, kann eine genaue Lokalisierung durchgeführt werden. Dabei spielt die Wahrnehmungen der Umwelt eine entscheidend Rolle, wie genau und effizient eine Posi-

tionsbestimmung durchgeführt werden kann. Die Sensoren, mit denen äußere Einflüsse gemessen werden können, sind in Abschnitt 3.5 zu finden.
Das Problem der Lokalisierung ist nach Thrun u. a. [2001] in drei Unterarten klassifiziert werden. Das erste und auch einfachere Problem nennt sich die *lokale Lokalisierung* gefolgt von der *globalen Lokalisierung* und dem sogenannten *kidnapped-Robot*.

Lokale Lokalisierung

Bekannt als Positionsverfolgung ist unter den genannten Problemen die erste Hürde, die es zu meistern gilt, um auch die restlichen Probleme lösen zu können. Bei der Positionsverfolgung wird die anfänglich bekannte Position durch das Auswerten von eingehenden Sensordaten nachvollzogen. Primär werden hierfür die Daten aus der Odometrie herangezogen und ausgewertet. Bekanntlich liefern diese keine genauen Messwerte und es gilt durch Fusion dieser Daten mit anderen Sensoren (Gyroskop) die Fehlersumme zu minimieren.

Globale Lokalisierung

Anders als bei der lokalen Lokalisierung ist die Anfangsposition nicht bekannt und deshalb benötigt der Roboter eine Karte seiner Umgebung. Durch das Auswerten weiterer eingehender Sensormessungen (Entfernungsmessungen) und unter Verwendung der Lösung des lokalen Problems ist nun die Aufgabe des Roboters seine Position auf der Karte zu bestimmen.

kidnapped Robot

Dieses Problem beinhaltet das Erkennen, dass die angenommene Roboterposition falsch ist und folglich die richtige Position bestimmt werden muss. Der Lösungsansatz hierbei beinhaltet lediglich die Erweiterung der globalen Lokalisierungslösung um eine Methode zur Erkennung der falschen Position.

3.3. Methoden der Lokalisierung

Um aus den gewonnenen Umweltinformationen die Roboterposition zu bestimmen, werden Methoden herangezogen, die diese in einer Karte repräsentieren. Einige werden hier nachfolgend vorgestellt, um einen Einblick in die Arbeitsweise zu erlangen.
Diese werden in der Klasse der *probabilistischen*[1] *Verfahren* zusammengefasst und wählen die wahrscheinlichste Position aus mehreren möglichen aus. Dies geschieht durch den Abgleich von eingehenden Informationen der Umgebung auf eine vorhandene Karte Thrun [2001]. Das große Problem dieser Verfahren ist die hohe Komplexität, die entsteht, wenn eine große Menge von wahrscheinlichen Positionen gegen eine andere Menge abgeglichen werden muss. Um dieses Problem einzuschränken werden approximative Verfahren eingesetzt.

Überblick probabilistischer Ansätze

Nach Elfers [2008] besitzen probabilistische Ansätze die Möglichkeit, durch das Ausdrücken einer Wahrscheinlichkeit die Unsicherheit bezüglich der Informationen zur eigenen Position modellieren zu lassen. Dies ist insofern wichtig, da die Informationen aus den Messungen der Sensoren, wie in Abschnitt 3.5 beschrieben wird, oft ungenau sein kann. Die in Anderson u. a. [2002] grafische Zusammenfassung der

[1] Wahrscheinlichkeitstheorie oder Wahrscheinlichkeitsrechnung aus der Mathematik

probabilistischen Algorithmen ist hervorragend geeignet, um einen ersten Überblick über die Ansiedlung dieser Algorithmen zu bekommen.

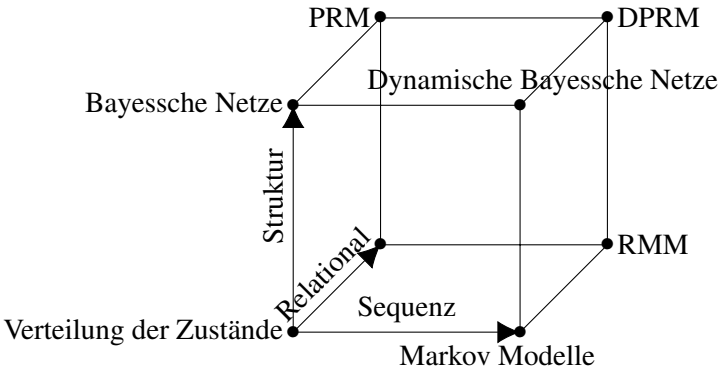


Abbildung 3.1.: Probabilstische Modellübersicht, adaptiert von Anderson u. a. [2002]

Um die gezeigte Grafik besser zu verstehen werden die einzelnen Knoten kurz besprochen. Das Modell ist auf die drei Eigenschaften Struktur, Sequenz und Relationalität laut Anderson u. a. [2002] zurückführen:

- sequenzell
  Aus dem sequenziellen Ansatz geht das Markov Modell (MM) hervor, welches Zustandssequenzen modellieren kann.
- strukturell
  Das Bayessches Netz (BN) verfügt über die Eigenschaft der strukturierten Unabhängigkeit zwischen Zustandsmengen.
- relational
  Mit dieser Eigenschaft werden Zustände mit der dazugehörigen Relation[2] modelliert.

Alle weiteren Knoten des Modells sind Kombinationen aus den hervorgehenden Algorithmen der 1. Generation. Da im Verlauf dieser Arbeit ausschließlich der sequenzielle Zweig Anwendung findet, wird der Vollständigkeit halber an dieser Stelle kurz auf die anderen Algorithmen eingegangen.

- BN
  Durch ein BN ist es möglich Systeme mit Unsicherheiten darzustellen. Sie stellen einen gerichteten azyklischen Graphen dar, dessen Knoten, Zufallsvariablen und Kanten direkte stochastische Abhängigkeiten repräsentieren. Alle Zufallsvariablen zusammengefasst stellen die Menge des unsicheren Wissens dar und das Netz dessen Wahrscheinlichkeitsverteilung.
- Dynamisches Bayessche Netz (DBN)
  Anders als bei dem BN beschränkt sich das DBN bei den zu verarbeiteten Beobachtungen nicht nur auf einen Zeitpunkt sondern weitet den Zustandsraum von Beobachtungen auf einer ganzen Zeitreihe (Sequenz von Beobachtungen) aus.

[2] aus der Mathematik: Eine Relation ist die Beziehung zwischen mindestens zwei Elementen eines geordneten Paares.

- Relationales Markov Modell (RMM)
  In Anderson u. a. [2002] werden das RMM als Verallgemeinerung vom Markov Modell beschrieben, wobei die Zustände von verschiedenen Typen sein können und jeder Typ durch eine andere Menge an Variablen beschrieben werden kann.

- Probabilistisches Relationale Modell (PRM)
  Dieses Modell ist auf Firedman u. a. [1999] zurückzuführen, der das BN erweiterte. In einem PRM sind alle Objekte in einer Menge von Klassen zerteilt und für jede dieser Klassen ist ein unterschiedliches probabilistisches Modell erstellt, welches spezifiziert, wie jedes Attribut in Relation zu anderen Attributen verwandter Klassen steht.

- Dynamische Probabilistische Relationale Modell (DPRM)
  Dieses Modell ist laut Sanghai u. a. [2003] eine Erweiterung zum DBN, wobei jeder Zeitabschnitt und die Abhängigkeiten zu vorherigen Zeitabschnitten durch eine PRM repräsentiert wird.

3.4. Markov-Kette

Die Markov-Kette wurde von dem russischen Mathematiker Andrej Andreevič Markov entwickelt und ist ein statistisches Werkzeug, um eine zukünftige Vorhersage bezogen auf gegenwärtige Informationen zu schließen. Um die Markov-Kette zu beschreiben, auf die im weiterem Verlauf aufgebaut wird, folgen nun einige Mathematische Hintergründe. Das zusammengetragene Wissen stammt aus dem Werk Wahrscheinlichkeitstheorie und Stochastische Prozesse Mürmann [2014].

Stochastischer Prozess: Ein stochastischer Prozess ist eine nicht-leere Familie $(X_t)_{t\in\mathcal{T}}$ von Zufallsvariablen auf einem Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ mit Werten in einem messbaren Raum $(E, \mathcal{B})$. $\mathcal{T}$ heißt die Zeitmenge und $(E, \mathcal{B})$ der Zustandsraum von $(X_t)_{t\in\mathcal{T}}$.

Die Markov-Ketten sind eine spezielle Klasse von stochastischen Prozessen, denn sie sind diskret in Zeit und Zustandsraum. Genau darin liegt der Vorteil, denn so lassen sie sich mit elementaren Methoden behandeln.

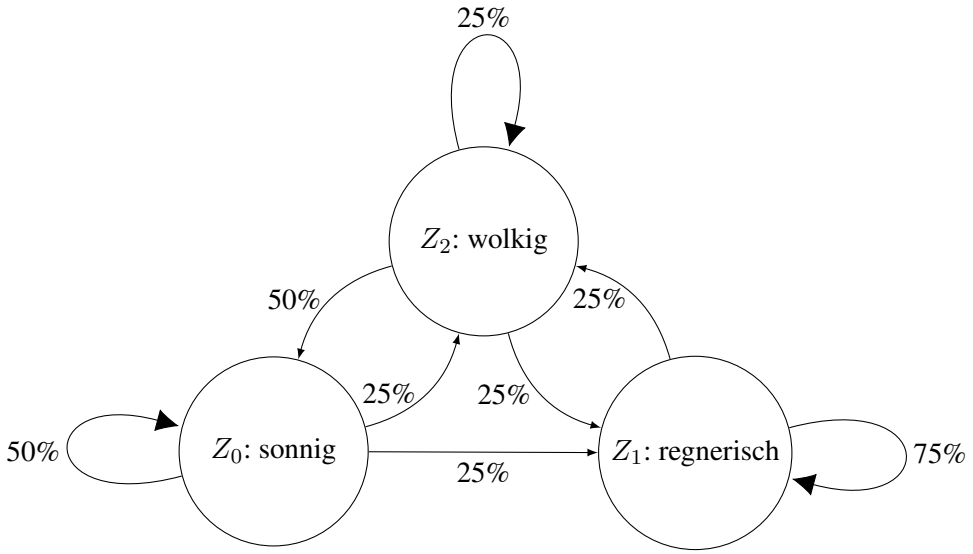
Markov-Kette: Eine Markov-Kette eine Folge $X_n, n \in \mathbb{Z}^+$ von diskreten Zufallsvariablen mit Werten in einem höchstens abzählbaren Zustandsraum E, der die folgende Markov-Eigenschaft hat:

$$\forall n \geq 0$$
und
$$i_0, \ldots, i_n, i_{n+1} \in E$$
mit
$$P(X_0 = i_0, \ldots, X_n = i_n) > 0$$
ist
$$P(X_{n+i} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n)$$

Um ein besseres Verständnis zu erlangen wird anhand eines einfachem Beispiels (einfaches Wettermodell) die Markov-Kette erläutert dieses Beispiel dient nicht als Mathematischer Beweis.
Angenommen das Wetter lässt sich anhand von drei Zuständen beschreiben , die da wären: wolkig, sonnig und regnerisch. Weiter wird angenommen, dass die Wahrscheinlichkeiten existieren, dass wenn es heute ...

- Z0: sonnig ist, es morgen mit der Wahrscheinlichkeit von 50% sonnig, 25% wolkig oder 25% regnerisch ist
- Z1: regnerisch ist, es morgen mit einer Wahrscheinlichkeit von 75% ebenfalls Regnet oder zu 25% wolkig ist
- Z2: wolkig ist, morgen zu 50% die Sonne scheint oder es zu jeweils 25% weiter wolkig ist oder gar regnet.

Diese Werte werden nun in einen Graphen dem Prozessdiagramm überführt, um diese zu visualisieren.



Anschließend wird die Überführungsmatrix aufgestellt, in der die Spaltenbeschriftung den Ausgangszustand wiederspiegelt und die Zeilenbeschriftung den jeweiligen Zielzustand. Die Einträge der Matrix ergeben sich dann schließlich aus den Übergangswahrscheinlichkeiten.

$$M = \begin{array}{c} \\ Z_0 \\ Z_1 \\ Z_2 \end{array} \begin{array}{c} \begin{array}{ccc} Z_0 & Z_1 & Z_2 \end{array} \\ \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.75 \end{pmatrix} \end{array}$$

Anhand von M kann vorausgesagt werden, wie das Wetter zu einem Zeitpunkt $t > 0$ mit welcher Wahrscheinlichkeit wird. Als Startzustand wird das aktuelle Wetter vor Ort gewählt, welches wolkig ist. Wir befinden uns also im Zustand Z_2 und der resultierende Vektor v_0 sieht wie folgt aus:

$$v_0 = \begin{array}{c} \\ Z_0 \\ Z_1 \\ Z_2 \end{array} \begin{array}{c} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{array}$$

Anschließend wird elementare Matrizenrechnung durchgeführt und M mit v_0 multipliziert. Der neue Vektor v_1 spiegelt das Wetter zum Zeitpunkt $t_0 + 1$ wieder.

$$M * v_0 = \begin{matrix} Z_0 \\ Z_1 \\ Z_2 \end{matrix} \begin{pmatrix} 0.0 \\ 0.25 \\ 0.75 \end{pmatrix} = v_1$$

Bei einem Blick auf die Zeitspanne t_n kann festgestellt werden, dass die Wahrscheinlichkeit, in welchem Zustand man sich zukünftig befinden, gegen einen festen Wert konvergieren und das unabhängig vom gewählten Startzustand.

$$M * v_1 = \begin{matrix} Z_0 \\ Z_1 \\ Z_2 \end{matrix} \begin{pmatrix} 1/8 \\ 1/4 \\ 5/8 \end{pmatrix} = v_2 * M = \cdots = v_{n-1} * M \approx \begin{matrix} Z_0 \\ Z_1 \\ Z_2 \end{matrix} \begin{pmatrix} 1/4 \\ 1/4 \\ 1/2 \end{pmatrix} = v_n$$

Dieses Ergebnis ist auf die Markov-Eigenschaft der sogenannten Gedächtnislosigkeit zurückzuführen, denn die Wahrscheinlichkeit $P(X_{n+1} = i_{n+1} | X_n = i_n)$ in den Zustand $X_{n+1} = i_{n+1}$ zu gelangen ist nur von dem Zustand $X_n = i_n$ abhängig. Nicht etwa davon, wie i_n erreicht wurde.

3.4.1. Markov Lokalisierung

Die zuvor beschriebene Morkov-Kette findet bei der Markov Lokalisierung (ML) nun ihre Anwendung. Bei dieser probabilistischen Lokalisierung wird nicht nur eine Annahme über die Roboterposition getroffen, sondern behält eine Wahrscheinlichkeitsverteilung über den ganzen Raum für viele verschiedene Annahmen bei.
Die Notation für die geschätzte Position l besteht nach Fox u. a. [1999] aus den Kartesischen Koordinaten (x, y) und der Ausrichtung θ mit,

$$l =< x, y, \theta >$$

Da der Roboter in der Regel seine genaue Position nicht kennt, sondern nur in dem Glauben (engl. belief) ist sie zu kennen, wurde für die geschätzte Position die Notation $Bel(l)$ eingeführt. Zu einem bestimmten Zeitpunkt ist also $Bel(l)$ die Wahrscheinlichkeit, sich auf einer bestimmten Position zu befinden. $Bel(l)$ wird durch zwei verschiedene Aktionen aktualisiert. Einmal durch das Empfangen von Sensordaten und zum anderen durch die Odometrie. Fox u. a. [1999]Thrun u. a. [2005]

3.4.2. Monte-Carlo-Lokalisierung

In Thrun u. a. [2005] wird die Monte Carlo Lokalisierung (MCL) als erweiterte ML mit einen Partikelfilter beschrieben. Dieser Partikelfilter hält mehrere wahrscheinliche Positionen (Partikel) vor und bei jeder Iteration durch diese Partikelmenge werden die unwahrscheinlichsten aus der Menge entfernt. Die entfernten Partikel werden dann auf eine zufällige Position in der Nähe der Position gesetzt, die die meiste Ansammlung Partikel besitzt. Wobei die Erweiterung aus einem Partikelfilter besteht, dass jede wahrscheinliche

Roboterposition zusätzlich gewichtet wird. Auch wird in Thrun u. a. [2001] beschreiben, dass mit dieser Methode das lokale und globale Lokalisierungsproblem gelöst werden kann.

3.4.3. Augmented Monte Carlo Localisation

Die Augmented Monte Carlo Localisation (AMCL) beruht nach Thrun u. a. [2005] auf der MCL, die in dem Abschnitt 3.4.2 bereits vorgestellt wurde. Es kommt lediglich neu hinzu, dass mit einer adaptiv angepassten Wahrscheinlichkeit zufällig neue Partikel hinzugefügt werden. Durch diese zufällig gestreuten Partikel ist es möglich, zusätzlich zu den durch die MCL gelösten Problem auch das Kidnaped-Robot-Problem zu lösen zu können.
Der resultierende Algorithmus ist in Listing 3.1 zu sehen und stammt aus Thrun u. a. [2005]. In Zeile 8 wird die empirsche Messwahrscheinlichkeit berechnet, mit der dann in Zeile 10 und 11 die kurz- und langfristigen Mittelwerte dieser Wahrscheinlichkeit bestimmt werden. Voraussetzung ist, dass $0 \leq \alpha_{slow} < \alpha_{fast}$, wobei $\alpha_{slow}, \alpha_{fast}$ die Zerfallrate für den Exponentialfilter ist. Das wesentliche passiert in Zeile 13, denn dort wird während des Resamplings mit der Wahrscheinlichkeit $max(0.0, 1.0 - w_{fast}/w_{slow})$ ein neues Zufälliges Partikel gestreut.

Listing 3.1: AMCL Algorithmus

```
1   Algorithm Augmented_MCL(χ_{t-1}, u_t, z_t, m):
        static w_slow, w_fast
3       χ̄_t = χ_t = ∅
        for m = 1 to M do
5           x_t^[m] = sample_motion_model(u_t, x_{t-1}^[m])
            w_t^[m] = measurment_model(z_t, x_t^[m], m)
7           χ̄_t = χ̄_t + < x_t^[m], w_t^[m] >
            w_avg = w_avg + 1/M w_t^[m]
9       endfor
        w_slow = w_slow + α_slow(w_avg − w_slow)
11      w_fast = w_fast + α_fast(w_avg − w_fast)
        for m = 1 to M do
13          with probability max(0.0, 1.0 − w_fast/w_slow) do
                add random pose to χ_t
15          else
                draw i ∈ {1, ..., N} with probability inf w_t^[i]
17              add x_t^[i]
            endwith
19      endfor
        return χ_t
```

3.5. Sensoren

Einleitend zu dieser Arbeit wurde bereits erwähnt, dass ein Roboter auf die Informationen aus seiner Umgebung angewiesen ist. In diesem Abschnitt werden einige Sensorgruppen vorgestellt, die dem Roboter mit genau den Informationen versorgen, die dieser für eine Lokalisierung benötigt.

3.5.1. Abstandssensoren

Diese Gruppe von Sensoren ermöglichen es dem Roboter die Entfernungen zu Objekten zu bestimmen und sind für die Lokalisierung und dem SLAM-Prozess unerlässlich. Nachfolgend sind 3 Sensoren aufgeführt, mit denen Entfernungen gemessen werden können.

Ultraschall

Ultraschallsensoren sind kostengünstig und werden unteranderem als Einparkhilfe in der Automobilbranche eingesetzt. Diese Sensoren senden eine Ultraschallwelle aus und werden von Objekten reflektiert. Dabei wird ein gerichtetes Ultraschallsignal an die Umwelt abgegeben, welches sich mit Schallgeschwindigkeit[3] ausbreitet. Trifft dieses auf einen Gegenstand so wird ein Teil des Signals reflektiert. Sobald das reflektierte Signal vom Empfänger registriert wird, kann anhand der Differenz zwischen Sendezeitpunkt und Empfangszeitpunkt errechnet werden, in welcher Entfernung sich ein Objekt zum Sensor befindet. Dieses Verfahren wird Time of Flight (ToF) genannt. Der Nachteil dieser Sensoren ist, dass diese einen großen Öffnungswinkel besitzen. Was soviel bedeutet, dass mit zunehmenden Abstand die Fläche größer wird, in der die Welle auf ein Objekt treffen kann.

Infrarot

Infrarot (IR)-Entfernungssensoren besitzen gegenüber der gerade genannten Ultraschallsensoren einen geringen Öffnungswinkel. Aus diesem Grund sind mit einem IR-Sensor punktuelle Entfernungsmessungen auch auf größeren Distanzen möglich. Der Sharp GP2Y0A710K0F IR-Sensor berechnet die Entfernung auf eine andere Weise. Nach dem Datenblatt Sharp [2006], wird nicht die Laufzeit des Signals gemessen sondern der Winkel, indem das Reflektierende Lichtsignal auf dem Empfänger trifft. Die Winkelmessung wird dabei von einer speziellen Kamera durchgeführt, die in Sektoren aufgeteilt ist. Ein Nachteil dieses Sensors ist, dass sich Teile der Sonnenstrahlen ebenfalls im Infrarotspektrum befinden und dadurch den Sensor stören könnten.

Laser

Die derzeit besten Entfernungsmesser in Bezug auf Reichweite und Genauigkeit bauen auf die Lasertechnik auf. Zur Berechnung der Entfernung wird auf die ToF-Methode zurückgegriffen oder eine Differenzmessung der Phasen vorgenommen. Diese Sensoren sind sehr teuer aufgrund der benötigten Hardware, die Signale verarbeiten müssen, die mit Lichtgeschwindigkeit reflektiert werden. Diese Sensoren reagieren unempfindlicher auf natürliche Lichtquellen und sind somit pregistiniert für den Außeneinsatz. Das System Light Distance And Ranging (LIDAR)-Lite v2 ist Entfernungsmesser, der auf die Lasertechnik aufbaut und hat eine Reichweite von 40m.

3.5.2. inertial measurement unit

Das unter Inertial Measurement Unit (IMU) bezeichnete System ist nach Woodman [2007], eine Fusion aus einem drei achsigen Beschleunigungssensor und drei achsigen Drehratensor (Gyroskop). Wobei der Beschleunigungssensor die Linearbewegungen in der x-, y- und z-Achse messen kann und der Drehratensensor misst dabei die Rotation um diese Achsen. Das Razor-IMU-Board von Sparkfunk bietet zu den bereits bekannten Sensoren noch einen zusätzlichen 3 achsiges Magnetometer. Dieses Sensor hat die Eigenschaft, dass das Erdmagnetfeld gemessen werden kann und somit ein weiterer wichtiger Faktor in der

[3] Die Schallgeschwindigkeit beträgt 343,2 m/s

Fehlerminimierung ist. Dieses Board besitzt zudem einen eignen Mikrocontroller und ist mit einer Firmware ausgestattet, die die Sensorfusion mithilfe des Direction Cosine Matrix (DCM)-Algorithmus durchführt Sparkfun

3.6. Zusammenfassung

In diesem Kapitel wurde eine Auswahl an Methoden präsentiert, die eine Lokalisierung und Kartengenerierung ermöglichen und Sensoren, die diese wiederum mit den benötigten Information versorgen. Für diese Arbeit wurde auf die teuren laserbasierten Sensoren verzichtet und auf eine Fusion von Ultraschall und Infrarot gebaut. Unteranderem wird dieses System im nächsten Kapitel vorgestellt.

4. Umsetzung

In diesem Teil der Arbeit geht es darum, den in Lübke [2014] entworfenen Rasenmähroboter zu erweitern, damit dieser in der Lage ist, eine Karte seiner Umgebung anzulegen und unter Beibehaltung der ihm gegebenen Grenzen einen geplanten Mähzyklus abzufahren. Um diese Aufgaben erfüllen zu können mussten grundlegende Änderungen an der Hardware und wird in Abschnitt 4.1 behandelt. Das mit unter wichtigste Sensorsystem dieser Arbeit, welches den Roboter mit Entfernungsmessungen versorgt, ist das entworfene self stabilized short distance (3sd)-LIDAR, welches in Abschnitt 4.2 vorgestellt wird. Der Roboter benötigte eine zusätzliche Steuereinheit aus Gründen die Abschnitt 4.3 dargelegt sind. Der neue leistungsstärkere Hauptcomputer, der für die Verarbeitung und Auswertung sämtlicher Daten zuständig ist, wurde mit dem modernen Roboter-Framework ROS aufgerüstet. Ein kurze Einweisung in dieses System ist in Anhang B zu finden, in dem neben einige wichtige Pakete auch nützliche Informationen über die Benutzung des Frameworks stehen. Der Abschnitt 4.4 baut auf dieses Framework auf und schildert alle nötigen Implementierungen, die zusammengenommen die Aufgabenstellung aus Abschnitt 2.1 lösen sollten.

4.1. Änderungen am bestehenden System

Der Mähroboter, der die Grundlage dieser Arbeit ist, war mit der installierten Konfiguration nicht dafür ausgelegt autonom zu arbeiten. Aus diesem Grund mussten Veränderungen an der vorhandenen Hardware gemacht und zusätzliche Hardware integriert werden. Diese Änderungen sind im Abschnitt 4.1.1 zu finden. Um die Distanzen zwischen dem Roboter und seiner Umgebung messen zu können, wurde ein eigenes System entworfen. Dieses Selbststabilisierende System ist auf der Grundlage eines, im Azimuth beweglichen, Infrarot-Entfernungssensor aufgebaut und im Abschnitt 4.1.2 aufgeführt.

4.1.1. Änderungen an der Hardwarekonfiguration

Der Computer

Der verbaute Raspberry PI 1B wurde gegen den aktuell auf dem Markt existierenden Raspberry PI 3B ausgetauscht. Durch die Kartierung, Lokalisierung und Planung des zukünftigen Softwaresystems war eine höhere Rechenleistung erforderlich. Diese bietet der neue Raspberry durch seinen 4-Kern-Prozessor und seinem größerem Arbeitsspeicher gegenüber dem älterem Modell. Das verbaute Adapterboard, welches dazu genutzt wurde die Pins der Platine besser zugänglich zu machen, wurde entfernt.

Verkabelung und Zusatzplatinen

Durch den Neuaufbau wurden Veränderungen an der Verkabelung vorgenommen. Gründe für diese Änderungen sind der Wegfall des Induktionsschleifensensors und somit auch der Serial Data Line (SDA)/Serial Clock Line (SCL)-Konverter; die Änderung der Datenabfrage der Front-Ultraschallsensoren und die Ände-

rungen am Stoßfänger. Der neue Schaltplan wurde unter der Verwendung des Programms Fritzing[1] erstellt, das leichter zu bedienen ist als das Programm Eagle[2]. Nachfolgend ist der neue Schaltplan aufgeführt.

Stoßfänger

Der 40 cm lange Stoßfänger, der an der Front des Roboters in einer Höhe von 20 cm angebracht war, ist 12 cm nach unten versetzt worden. Der Grund hierfür ist Nutzung als Sensor für Hindernisse im Bodenbereich, die durch die anderen Sensoren nicht bemerkt wurden. Der Mechanismus zum Auslösen des Sensors wurde durch die Verwendung von zwei Schließkontakten mit Federbeinchen (links und rechts) geändert, was ein Öffnen der Kontakte bei Nichtbelastung des Sensors garantieren sollte. Die zwei Kontakte sind parallel angeordnet und sorgen durch diese Redundanz dafür, dass eine ausreichend einwirkende Kraft auf eine beliebige Position des Sensors, das Schließen mindestens eines Kontaktes zur Folge hat. Hindernisse im Frontbereich des Roboters und mit einer Höhe von mindestens 8 cm über der Grasnarbe werden nun zusätzlich wahrgenommen und können in die Planung von Fahrmanövern mit einfließen.

Induktionsschleifenplatine

Die Induktionsschleife wie in Lübke [2014] beschrieben, wurde in dieser Arbeit nicht benötigt, da ein geplante virtuelle Begrenzung dafür sorgen sollte, dass der Mähroboter die vorgeschrieben Fläche nicht verlassen würde. Deshalb wurde der Arduino Uno mit Adapterboard entfernt.

Antrieb

Die Schrittmotorensteuerung wurde auf 1/16 Mikroschritte eingestellt, was den Vorteil hat, dass die Auswirkung von Schrittverlusten einen kleineren Fehler in der Berechnung der Odometriedaten mit sich bringt und eine Beschleunigung bzw. Verlangsamung der Umdrehungszahl weniger Schlupf verursacht. In Tabelle 4.1 sind die Einstellmöglichkeiten der Schrittmotorensteuerung aufgeführt.

a)	Schritt	4	3	2	1	Strom	b)	Strom	2	1
	1	0	0	0	0	100%		100%	1	1
	1/2	0	1	1	0	75%		75%	1	0
	1/8	1	1	0	1	50%		50%	0	1
	1/16	1	0	1	1	25%		25%	0	0

Tabelle 4.1.: a) Jumperstellung der jeweiligen Mirkroschritte b) Stromabsenkung

Versorgungsspannung

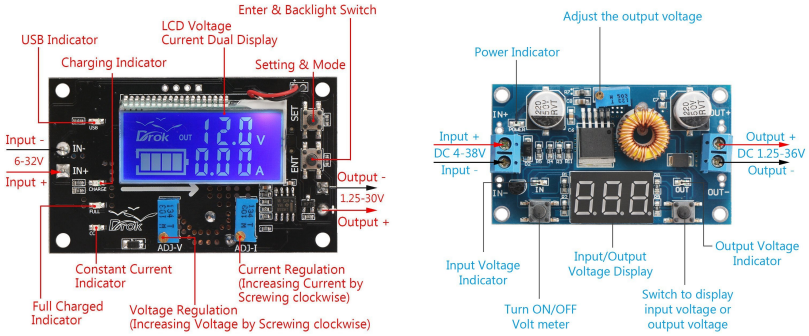
Das bisherige Spannungsversorgungsnetzt des Roboters bisher:

- **24 Volt Betriebsspannung** 2 in Reihe geschaltete 12 Volt 4500mAh großen NIMH Akkus
- **12 Volt Betriebsspannung** 1 x 12 Volt 4500 mAh Nickel-metal hybride (NIMH) Akku

[1] frei erhältlich unter http://fritzing.org/download/
[2] Programm zum erstellen von Schaltplänen. Wurde in der ersten Version benutzt

- **5 Volt Betriebsspannung** 5V/1A Spannungsregler

Nachdem alle Änderungen am System vorgenommen wurden, sieht das Versorgungsnetz wie folgt aus: Die Anordnung der Akkus wurde beibehalten. Ein Direct Current (DC)-DC Step-Down Spannungsregler, in Abbildung 4.1 sorgt für eine stabile 12 Volt Betriebspannung um das LIDAR-System zu versorgen. Ein



(a) genutzt für die 12 Volt Versorgung (b) genutzt für die 5 Volt Versorgung

Abbildung 4.1.: DC-DC Step-Down Spannungsregulatoren der Firma Drak

weiterer Regulator sorgt für die Bereitstellung der 5Volt Betriebsspannung, die nach wie vor der Raspberry Pi u.a. benötigen.

4.1.2. Zusätzlich integrierte Bauteile

Neben den schon erwähnten Änderungen an der bisherigen Konfiguration wurden zusätzliche Bauteile integriert, die zum Erfüllen der Aufgabenstellung nötig waren. Zu diesen Bauteilen zählen eine IMU, das Steuerboard und das LIDAR-System, welches in einem gesonderten Abschnitt vorgestellt wird.

Razor 9DOF IMU Board

Um die Ausrichtung des Mähroboters in seiner Umgebung messen zu können wurde das Razor-9DOF-IMU-Board[3] untergebracht. Da schon erste positive persönliche Erfahrungen mit diesem Board gesammelt wurden und eine vom Hersteller bereitgestellte Firmware für gefilterte Sensordaten sorgt, wurde dieses Board gewählt.

Arduino Steuerboard

Als zusätzlichen Microcontroller wurde ein Arduino Mega 2560 verbaut, der aufgrund seiner vielen digitalen und analogen Anschlüsse prädestiniert für die Steuerung von Hardware und den Empfang der Sensorrohdaten ist. Der Microcontroller übernimmt aus Gründen, die im Abschnitt 4.4 aufgeführt sind, die Ansteuerung der Schrittmotoren. Die Ultraschallsensoren und Schalter werden ebenfalls vom Arduino überwacht.

[3] siehe Kapitel 3

4.2. LIDAR-System

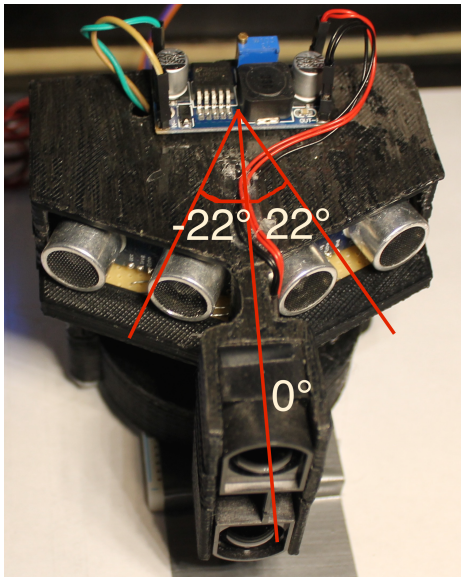


Abbildung 4.2.: LIDAR-Prototyp mit eingezeichnetem Sensor-Offset

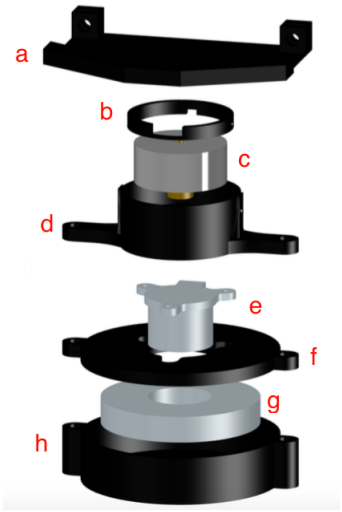
Im Abschnitt 3.5 wurden mehrere Sensoren vorgestellt, um den Abstand zu Objekten messen zu können. Da ein kommerzielles LIDAR-System sehr kostspielig ist, wurde auf den Kauf eines solchen Systems verzichtet und ein Prototyp erstellt, der ebenfalls ein Entfernungsbild erstellt. Dieser ist in Abbildung 4.2 zu sehen. Der entwickelte Prototyp hat die besondere Eigenschaft, dass er in der X-Achse und Y-Achse entkoppelt[4] vom Roboter gelagert ist. Auf einer unebenen Fläche wirkt sich, bei einem fest verbauten Sensor, jede Bewegung des Roboters nachteilig auf das Sensorbild aus. Damit der Sensor nicht ins Trudeln kommt, wurde ein bürstenloser Elektromotor unterhalb des Sensors angebracht. Dieser hat die Aufgabe eine Schwungmasse in Rotation zu bringen und durch das Ausnutzen des Gesetzes zur Drehimpulserhaltung[5] zu stabilisieren. Als Abstandssensor wurde eine Fusion aus IR-Abstandssensor und Ultraschallsensor gewählt.

4.2.1. verwendete Bauteile

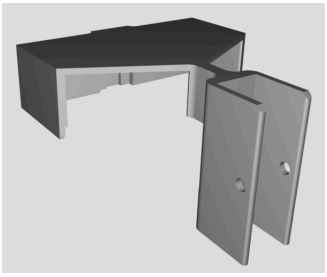
In Abbildung 4.3 ist das Explosionsbild des Sensorsystems zu sehen. Nicht dabei ist die Aufhängung sowie die elektronischen Bauteile. Die Platinenhalterung, der Schrittmotorring und der Schrittmotor sind fest miteinander verbunden und bilden zusammen mit dem Topcase den rotierenden Teil des Systems. Der Rotor des Schrittmotors ist fest in die Rotoraufnahme eingepresst und bildet so den Kontakt zum restlichen starren Systemabschnitt. In die Motorfassung ist der Brushlessmotor eingepasst, an dem die Schwungmasse aus Stahl befestigt ist. Der Schutzdeckel ist zusammen mit der Motorhalterung und der Rotoraufnahme mit zwei Bolzen befestigt. Der Schrittmotor wurde aus einem alten HP-Drucker ausgebaut und besitzt eine Schrittauflösung von 7,5°. Der Brushlessmotor wurde aus einem alten Barcodescanner extrahiert und hat

[4] wie eine Kardanische Aufhängung jedoch nur in zwei Achsen

[5] Besagt, dass ein Körper die Geschwindigkeit und Richtung seines Drehimpulses beibehalten will wobei die Richtung dabei senkrecht auf der Drehachse liegt.



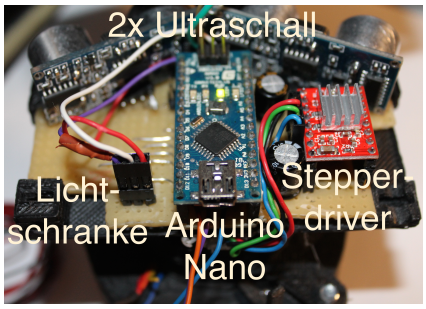
(a) a) Platinenhalterung b) Schrittmotorring c) Schrittmotor d) Rotoraufnahme mit Aufhängungsbohrung e) Brushless Motor f) Motorfassung g) Schwungmasse h) Schutzdeckel



(b) Topcase mit Halterung für den IR-Sensor

Abbildung 4.3.: Explosionsbild LIDAR

eine Betriebsspannung von 5V bei einer Drehzahl von 5000 min^{-1}. In Abbildung 4.4(a) ist die Platine



(a) Sensorplatine

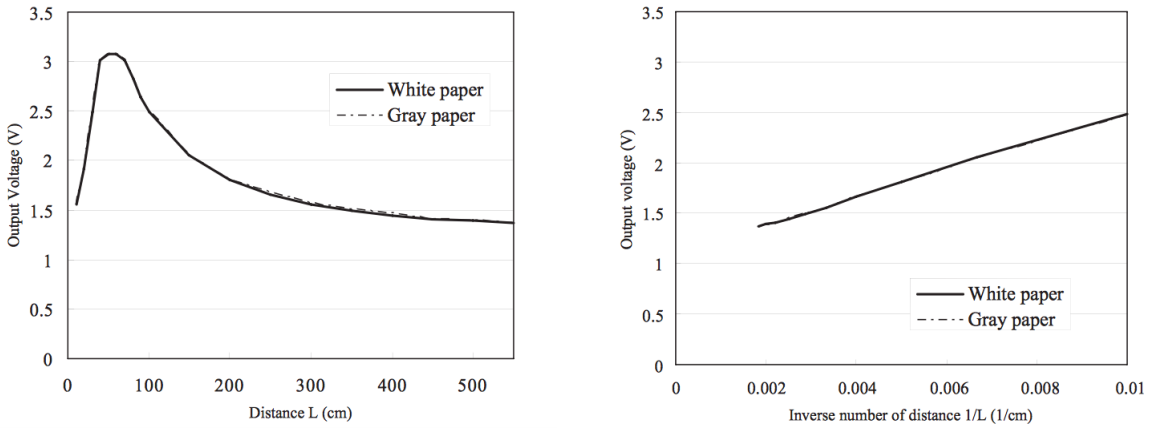
(b) Sharp Infrarot Sensor GP2Y0A710K0F

Abbildung 4.4.: Sensoren und Platine vom LIDAR

zu sehen, auf der ein Arduino Nano Mikrocontroller verbaut ist. Die Hauptaufgaben des Mikrocontrollers bestehen im wesentlichen aus der Ansteuerung des Schrittmotors, dem ersten Auswerten der Sensordaten und der seriellen Kommunikation. Die Ansteuerung des Schrittmotors erfolgt über den Schrittmotortreiber. Dieser ist fest auf 1/16 Mikroschritte eingestellt. Somit macht der Sensor bei jedem Schritt eine horizontale Richtungsänderung von 0,46875°. Eine Gabellichtschranke, die unterhalb der Platinenhalterung angebracht ist, dient dazu, nach jedem Scan zu der selben Startposition zurückzukehren. Die zwei Ultraschallsensoren weisen einen Versatz zur 0° Linie von +/- 22° auf. Sie erkennen Hindernisse, die sich näher als 1m befinden. Obwohl sie laut Datenblatt eine Reichweite von 7m haben, wirken sich Bodenunebenheiten (Gras) aufgrund des großen Öffnungswinkel von 15° schon früh aus.

4.2.2. Sharp Gleichung

Um von dem gemessenen analogen Signal auf die resultierende Distanz zu kommen, muss eine Gleichung aufgestellt werden. Anfangs wurde die Gleichung anhand der gegebenen Kurven, siehe dazu die Kurven in Abbildung 4.5, aus dem Datenblatt erstellt, die bei Tests aber unbefriedigende Ergebnisse lieferte. Die Abweichung im Nahbereich (<2m) lag bei etwa +20cm und bei Entfernungen von >3m waren diese sogar deutlich darüber. Es wurde daher eine Kurve anhand von der tatsächlichen Entfernung gegenüber der



(a) Volt - Distanz Kurve (b) Volt - 1/Distanz Kurve

Abbildung 4.5.: Volt/Distanz-Kurve aus dem Datenblatt Sharp [2006] des GP2Y0A710K0F

gemessene Eingangsspannung angelegt.

Der Messaufbau

Für die Messung wurde ein **Arduino Nano** als Mikrocontroller benutzt, der die gemessene Spannung auf dem seriellen Port ausgegeben hat. Als **Referenzspannung** wurde die 3.3V des Nanos benutzt, um so eine höhere Auflösung gegenüber der normalen 5V zu bekommen. Da der Sharp Sensor laut Datenblatt bei einem Sendepuls ca. 330mA benötigt muss dieser extern mit 5V versorgt werden. Zusätzlich wird ein 220 μF Kondensator zum Glätten der Versorgungsspannung des Sharp's eingesetzt, was für ein stabileren Betrieb sorgen sollte. Als Reflektor diente ein weißes DIN A4 Blatt, welches auf einer Sperrholzplatte fixiert wurde

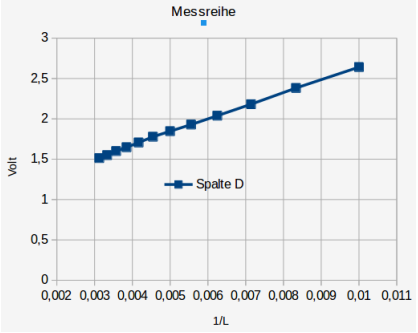
Die Messung

Für das Sammeln der Daten wurde ein kleines Kalibrierungsprogramm geschrieben, welches aus 10 Messungen den Median errechnet. Gestartet wird eine Messung mit der Eingabe von 'n'. Abgespeichert wird ein Messwert nur, wenn 'o' in die Konsole eingegeben wurde nachdem eine Messergebnis zu sehen war. Die Eingabe von 'r' bewirkt das Wiederholen dieser Messung und die einzustellende Weite wird wiederholt angezeigt. Nach dem Abspeichern eines Messwertes kann die momentan gespeicherte Tabelle mit 't' ausgegeben werden. Die Messintervalle gehen dabei beginned mit 80cm in 20cm-Schritten weiter. Das Ergebnis, einer kompletten Messung über die Distanz von 0,8m - 3,2m ist in Tabelle 4.6(a) zu sehen. Diese

Daten wurden dann in ein Liniendiagramm 4.6(b) überführt. Ein Vergleich mit der Kurve 4.5(b) zeigte, dass die Grenzwerte nicht übereinstimmten.

Distanz	Volt
80	2,94573
100	2,64070
120	2,38183
140	2,18129
160	2,03883
180	1,92894
200	1,84637
220	1,7778
240	1,706532
260	1,64771
280	1,60003
300	1,5499
320	1,513241

(a) Tabelle mit Messdaten des IR-Sensors



(b) Liniendiagramm anhand der Messdaten aus Tabelle 4.6(a)

Abbildung 4.6.: Verlauf der gemessenen Spannung gegenüber der Distanz

Gleichung aufstellen

Aufgrund der Linearisierung in 4.6 mit $1/L$ kann eine Funktion der Form $y = mx + b$ gefunden werden. Dazu werden die zwei Punkte $p_1(0,01|2,64070)$, $p_2(0,003125|1,513241)$ aus der Tabelle 4.6(a) gewählt und in ein lineares Gleichungssystem überführt.

$$I: \quad 1,513241 = 0,003125m + b$$
$$II: \quad 2,64070 = 0,01m + b$$

Um m zu ermitteln wird das Subtraktionsverfahren angewendet, $II - I$

$$2,64070 - 1,513241 = 0,01m - 0,003125m \quad +b - b$$

$$\Leftrightarrow \quad 1,127459 = 0,006875m \quad | * \tfrac{1}{0,006875}$$

$$\Leftrightarrow \quad m = 163,994\overline{036}$$

Fehlt somit nur noch b um die komplette Funktionsgleichung $f(x)$ zu bekommen. Dazu muss lediglich m in I oder II eingesetzt und nach b aufgelöst werden.

$$2,64070 = 0,01 * 163,994\overline{036} + b \quad | - (0,01 * 163,994\overline{036})$$

$$\Leftrightarrow \quad 2,64070 - (0,01 * 163,994\overline{036}) = b$$

$$b = 1,0007596\overline{3}$$

Somit haben wir die Funktion für y gefunden

$$f(x) = 163,994036 * x + 1,00075963 \tag{4.1}$$

Um jedoch anhand der gemessenen analogen Spannung auf die Entfernung schließen zu können muss die Gleichung 4.1 nach x umgestellt werden und invertiert werden.

$$\begin{aligned}
& y && = && 163,994036 * x^{-1} + 1,00075963 && | -1,00075963 \\
\Leftrightarrow\ & y - 1,00075963 && = && 163,994036 * x^{-1} && | * \tfrac{1}{163,994036} \\
\Leftrightarrow\ & \tfrac{y-1,000759}{163,994036} && = && x^{-1} && | Kehrwertbilden \\
\Leftrightarrow\ & \tfrac{163,994036}{y-1,000759} && = && x &&
\end{aligned}$$

Die daraus resultiert die Funktionsgleichung berechnet anhand der gemessenen Eingangsspannung die Entfernung:

$$f(y) = \frac{163,994036}{y - 1,000759} \tag{4.2}$$

4.2.3. Implementierungen

IR-Sensor

Im vorherigen Abschnitt wurde die Herleitung der Funktionsgleichung gezeigt, mit der die Entfernung berechnet wird. Listing 4.1 zeigt nun die Verwendung im Code.

Listing 4.1: Funktion zum Berechnen der Entfernung des IR-Sensors

```
float readSensorDistance()
{
    // Spannung am Analogen Referenz Pin messen
    float aref = analogRead(AREF_PIN);
    // 3.46Volt entspricht der tatsaechlich am 3.3V Pin anliegende
        Spannung
    // Spannungsschwankungen erkennen
    aref = 3.46 * aref / 1024;
    // Wert des analogen Pin messen und in Volt umrechnen
    float voltage = 1 / (aref * analogRead(IR_INPUT_PIN) / 1024);
    // Oberer Schnitt, denn ab 3V kann keine Entfernung berechnet werden
    if(voltage > 3.0) return 0;

    //Rueckgabe der Entfernung in cm
    return 163.99403636 / (voltage - 1.000759);
}
```

Ultraschall

Für das Abfragen der Ultraschallsensoren wurde die NewPing[6]-Bibliothek verwendet. Sie ermöglicht eine einfache Abfrage der Sensoren und bietet die Angabe einer maximalen Distanz.

Schrittmotor

Für die Ansteuerung des Schrittmotors wurde ebenfalls auf eine vorhandene Bibliothek zurückgegriffen. Die StepperDriver[7]-Bibliothek unterstützt das Ansteuern eines Schrittmotors über eine Treiberplatine. Die Bibliothek bietet die Möglichkeit, dass mehr als nur ein Schritt angesteuert wird. Das hat aber zur Folge, dass während der Ausführung der Bewegung nicht aus der Methode zurückgekehrt wird und folglich keine Messungen durchgeführt werden können. Aus diesem Grund macht der Schrittmotor nur einen Schritt während eines Schleifendurchlaufes. Es musste aber darauf geachtet werden, dass die Pulspausen am Step-Pin zeitlich mit der eingestellten Geschwindigkeit übereinstimmen. Dazu wurde aus der Bibliothek die folgende Zeile benutzt, um die richtige Pulspause zu errechnen, mit der diese arbeitet.

```
// microseconds = (micros in 1 Minute) / 48(Steps pro Revolution)/ 16
    / Speed in (Rounds per Minute)
step_pulse = 60 * 1000000L / MOTOR_STEPS / MICROSTEPS / RUN_SPEED;
```

Um jeden Scan von ein und dem selben Punkt zu starten wird der Schrittmotor solange nach rechts gefahren, bis die Gabellichtschranke schließt.

Das Zusammenspiel

Der Ablauf eines Scans im Microcontroller soll im folgenden Pseudocode vereinfacht dargestellt werden.

Listing 4.2: LIDAR Loop auf dem Nano

```
Algorithm LIDAR(MAX, MIN, Ringbuffer[]):
   forever do:
      // fahre zur Startposition
      stepper_homing() = MIN
      // solange maximale Gradzahl nicht erreicht ist
      for position = MIN to MAX do:
         // fahre einen Step weiter weiter (7.5Grad /16)
         stepper.move(1)
         // messe solange nicht der naechste Schritt gefahren werden
            darf
         while not time to do next Step:
            ir = readSensorDistance()
            left_us = ping_cm()
            right_us = ping_cm()
         end while
         // fuege Messwerte in einen Buffer
         Ringbuffer ← ir, left_us, right_us
```

[6] https://bitbucket.org/teckel12/arduino-new-ping/downloads
[7] https://github.com/laurb9/StepperDriver

```
            // sende Buffer, wenn er abgefragt wird
18          if send buffer
               Serial ← len(Ringbuffer), position, Ringbuffer
20       end for
         // solange der Buffer nicht leer ist warte
22       while Ringbuffer not empty
            // zu lange warten => alles aus
24          if connection timeout
               sleep()
26          // sende Buffer, wenn er abgefragt wird
            if send buffer
28             Serial ← len(Ringbuffer), MAX, Ringbuffer
```

Das Gegenstück zum Microcontroller befindet sich auf dem Raspberry PI und ist in Python geschrieben. Dieser ruft bei dem Sensor einen Datensatz ab und wartet (prozessblockierend) auf diesen. Auch hier soll ein vereinfachter Pseudocode den groben Ablauf erklären. Die verwendeten Filter werden im nächsten Abschnitt erklärt.

Listing 4.3: LIDAR Loop auf dem Raspberry PI

```
1    Algorithm LIDAR_PI():
        while ros is running:
3          // rufe naechste Werte ab
           Serial  ← request
5          // warte auf Daten
           response ← Serial
7          // wenn Header nicht stimmt starte neu ansonsten in sync
           if len(response.header) not ok
9             reset
           // solange noch Daten vorhanden sind fuege die Distanzen hinzu
11         for response.data > 0
              position, ir, us_left, us_right ← response.data
13            add_range_set(ir, us_left, us_right, position)
           // Fertig?
15         if position is MAX
              filter_ranges()
17            publish Laserscan
```

4.2.4. Die Filter

Die Methoden *add_range_set* und *filter_ranges* aus Listing 4.3 machen aus den Rohdaten den fertigen Laserscan. Diese Daten, die vom Sensor empfangen wurden, sind mit einigen Fehlern behaftet und gefiltert werden müssen. Die folgende Tabelle repräsentiert einen einzelnen kompletten Scan und ist unterteilt in drei Sektoren. Anhand dieser Sektoren werden die eingesetzten Filter erklärt.

	Sektor 3			Sektor 2					Sektor 1		
Grad	110	...	88	<88	...	0	...	>-88	-88	...	-110
IR	val	...	val	val	...	val	...	val	val	...	val
US_left	val	...	val	val	...	val	...	val	nan	...	nan
US_right	nan	...	nan	val	...	val	...	val	val	...	val

Sektor 1: Durch den Offset von +22° des linken Ultraschallsensors gegenüber dem IR-Sensor werden für diesen Bereich keine Messwerte erhalten.

Sektor 2: Werte von allen Sensoren liegen vor.

Sektor 3: Dieser Sektor ist ähnlich zu dem ersten, wobei der rechte Ultraschallsensor davon betroffen ist.

Der Vorfilter

Der hier als Vorfilter bezeichnete Filter hat die Aufgaben, die einkommende Entfernungsdaten mithilfe der zugehörigen Position den richtigen Sektoren zu zuordnen und zu speichern. Sobald vom rechten Ultraschallsensor Werte empfangen werden, werden diese mit den bereits gespeicherten Daten vergleichen und abgespeichert. Verglichen wird mit der Methode, die in Listing 4.4 zu sehen ist. Der erste Abfrage ist darauf zurückzuführen, dass für die gemessenen Spannungen theoretisch zwei verschiedene Distanzen möglich sind, wie in Abbildung 4.5(a)zu sehen ist. Wenn der Parameter b ein Distanz von < 0.4m hat, wird die IR-Entfernung ignoriert. Ist Parameter b in einer ϵ Umgebung von a, so wird eine Gewichtsfunktion angewandt. Die Werte von ϵ und der Gewichtung von a können jeweils in der Paketkonfigurationsdatei angepasst werden.

Listing 4.4: Funktion zum Berechnen der Entfernung des IR-Sensors

```
def best_or_mean(self, a=0.0, b=0.0):
        """
        a is weighted range value and b any other range value to compare
        use the best value of the two or build the mean if both are in
        an epsilon area
        :param a: IR range; first range value weight set in config
        :param b: US Range; second range value weight is 1 - weight for a
        :return: returns the best value or weighted mean of both
        """

        if 0 < b < 0.4:
            return b
        if a == 0 == b:
            return 0
        if a == 0:
            return b
        if b == 0:
            return a

        if float(a) + self.eps > b > float(a) - self.eps:  # b between
            epsilon and return weighted mean
```

```
21              return float(a) * self.weight_for_a + float(b) * (1 - self.
                    weight_for_a)
            else:
23              return a
```

Nachdem ein Scan komplett ist, wird er für das Publizieren in das ROS-System vorbereitet. Dazu durchläuft das Array mit den gefilterten Entfernungen noch zwei Stationen. Einen Nahbereichsfilter, denn die Ultraschallsensoren neigen dazu, Hindernisse breiter zu machen, als sie sind und das Interpolieren von fehlenden Zwischenwerten.

Der Nahbereichsfilter

Dieser Filter räumt den Nahbereich auf und basiert auf der Annahme, dass sich die Entfernung innerhalb von $\pm 1°$ nicht unterscheidet (ausgenommen von 0 m). Die Arbeitsweise ist relativ einfach: Wird ein Wert y < 0.8 von einem Wert größer 0.8 umgeben so ist der neue Wert für y = 0. Dies ist noch einmal in Abbildung 4.7 dargestellt und zeigt links einen ungefilterten Nahbereich und rechts das Ergebnis nach der Filterung. Warum 0 gewählt wird, geht aus dem nächsten Abschnitt hervor.

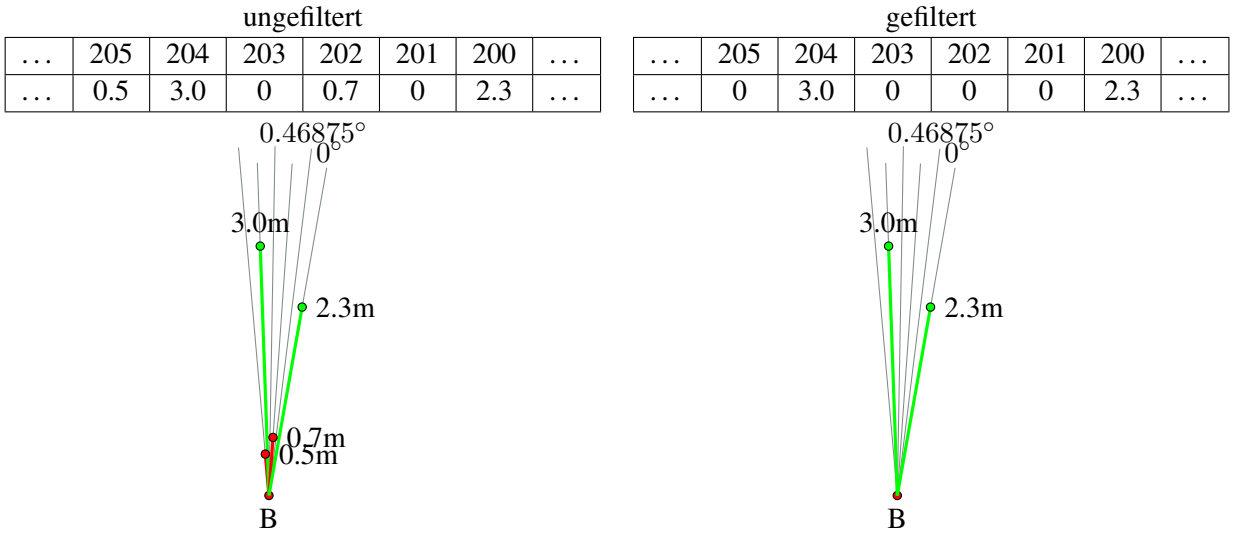


Abbildung 4.7.: Veranschaulichung der Arbeitsweise des Vorfilters

Das Interpolieren von fehlenden Werten

Dieses Verfahren ist etwas aufwendiger, jedoch notwendig um das System mit einer höheren Geschwindigkeit betreiben zu können und um ein besseres Sensorbild zu bekommen. Anhand der nachstehenden Grafik, welche vom vorherigen Beispiel 4.7 schon bekannt ist, wird das Interpolieren von 0 Werten erläutert. Wenn zwischen zwei Werten nur ein Wert fehlt, kann der Mittelwert dieser beiden Werte gebildet werden. Andernfalls wird die nachfolgende Arbeitsweise durchgeführt.
Das Interpolieren von Zwischenwerten erfolgt durch das Zeichnen einer Gerade $A|C$. Die Punkte A und C sind reguläre Entfernungen und bilden unter Hinzunahme des Punktes B jeweils die Geraden $B|C$ und

$B|A$. Alle drei Geraden bilden im Allgemeinen ein unregelmäßiges Dreieck. In Abbildung 4.8 ist dieses Dreieck durch die grünen Geraden zusammen mit der blauen Gerade visualisiert. Daraus folgt, dass die Seiten $a = 3.0m$ und $c = 2.3m$ lang sind. Weiteres Wissen kann aus der Anordnung der Werte geschlossen werden. Jeder Eintrag in der Entfernungsliste repräsentiert genau eine Entfernungsmessung vom Sensor.
Da der Schrittmotor in jeder Iteration genau $0.46875°$ weiter fährt, kann anhand der Indexdifferenz der beiden Einträge der Winkel β berrechnet werden. Der Wert von β ist somit $4 * 0.46875° = 1,875°$. Somit stehen drei Werte zur Berechnung aller fehlenden Werte des Dreiecks zur Verfügung. Gesucht sind alle Schnittpunkte der grau eingezeichneten Geraden mit der Seite b (blau). Als Nächstes wird die Seite b berechnet. Unter heranziehen des Kosinussatzes $b^2 = a^2 + c^2 - 2ac * \cos\beta$, der nach dem Wurzel ziehen folgende Form hat:

$$b = \sqrt{a^2 + c^2 - 2ac * \cos\beta} \tag{4.3}$$

und Einsetzen der gegebenen Werte wird für b eine Länge von $\approx 0.70526m$ erhalten. (man beachte: der Wert für β ist in Grad angegeben) Nun fehlen die Werte der restlichen Winkel α und γ. Es genügt den Wert für einen der beiden Winkel zu berechnen und bereits bekannt sind die Werte für β, b, c. Unter Anwendung des effizienteren Sinussatz $\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$, der nach γ' aufgelöst wird, ergibt folgende Form:

$$\gamma' = asin(\frac{\sin(\beta) * c}{b}) \tag{4.4}$$

Für γ' errechnet sich ein Wert von $\approx 8°$. Aufgrund des Verlaufs der Sinusfunktion muss eine Fallunterscheidung gemacht werden:

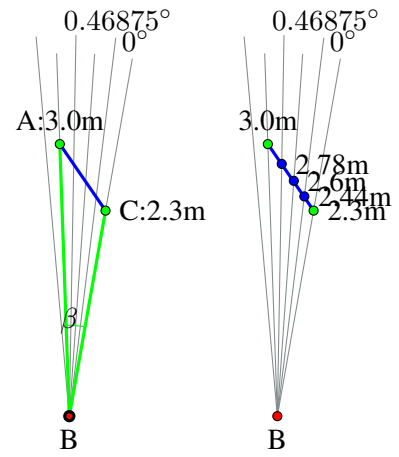
- **Fall I:** $c < a \rightarrow \gamma = \gamma'$
- **Fall II:** $c > a \rightarrow \gamma = 180 - \gamma'$

Somit ist unter Anwendung von Fall II $\gamma \approx 172°$.
Die fehlenden Zwischenwerte c^i, wobei $i \in [1, \ldots, 3]$, werden mit den bereits ermittelten Parametern berechnet. Konstant für alle Zwischenwerte sind a und γ', jedoch β^i mit $0.46875 * i$ verschieden. Mithilfe der Tatsache, dass die Winkelsumme aller Winkel in einem Dreieck $180°$ ergeben, werden die jeweiligen Winkel α^i berechnet.
Die Berechnung:
für **i = 1**

$$\begin{aligned} \beta^1 &= 0.46875° * 1 &&= 0.46875° \\ \alpha^1 &= 180° - \gamma - \beta^1 \\ &= 180° - 172° - 0.46875° \\ &= 7.53125° \\ c^1 &= \tfrac{sin(\gamma) * a}{sin(\alpha)} \\ &\approx 2.44225m \end{aligned} \tag{4.5}$$

für **i = 2**

$$c^2 \approx 2.60344m \tag{4.6}$$



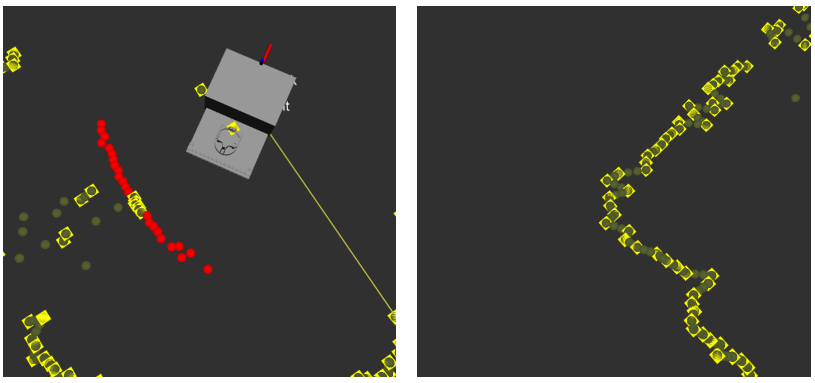
(a) Eingang Interpolierungsstufe (b) Alle Zwischenwerte aufgefüllt

Abbildung 4.8.: Interpolieren von Zwischenwerten

für **i = 2**

$$c^3 \approx 2.7876m \tag{4.7}$$

In Abbildung 4.9(b) wird das Ergebnis visualisiert. Das Interpolieren wird allerdings nur angewendet, wenn die beiden Punkte A und C maximal zehn Felder weit auseinanderliegen. Das entspricht einem Winkel β von $4.6875°$ und bei maximaler Entfernung von 5m ergibt das eine Länge von $b = 0,4m$.
Für die gegebenen Daten veranschaulicht Abbildung 4.9 das reale Sensorbild.



(a) rote Kreise = vor Filterung, gelbe Vierecke = vor dem Interpolieren, grüne Kreise = fertiger Laserscan (b) gelbe Vierecke = vor dem Interpolieren, grüne Kreise = fertiger Laserscan

Abbildung 4.9.: Filtern und Interpolieren aufgenommen in rviz

4.3. Arduino Mega Robotersteuerung

Die Firmware für den Arduino Mega ist mit dem ROS-System fest verbunden und stellt durch die Benutzung des *rosserial*-Pakets eine eigenständige Node dar. Die Grundaufgaben des μ-Controller wurden in

Abschnitt 4.1.2 bereits dargelegt und sollen nun bezogen auf das Gesamtsystem vertieft werden.
In Lübke [2014] wurde schon auf die Vor- und Nachteile des Antriebes mit Schrittmotoren eingegangen. Im späteren Verlauf dieser Arbeit hat sich zusätzlich noch als Nachteil herausgestellt, dass die Schrittmotorensteuerung eine kontinuierliche und gleichmäßige Versorgung mit Schrittbefehlen benötigt, um eine flüssige Fahrt zu gewährleisten. Die Auswirkungen dieser Problematik kam erst gegen Ende der Testphase zum Vorschein, als alle zum Betrieb notwendigen Nodes[8] auf dem Raspberry geladen wurden. Die unregelmäßige Ansteuerung der Motoren, verursacht durch eine sehr hohe Prozessorauslastung, ließ eine ordnungsgemäße Fahrt nicht zu. Aus diesem Grund wurden alle Interaktionen mit der Hardware auf den schon erwähnten μ-Controller ausgelagert. Die wichtigsten Teile der Implementierung werden im Folgenden erläutert.

Finate State Machine

In Abbildung 4.10 ist die Finite State Machine (FSM) des Arduino als Statechart zu sehen. Sie besteht aus sieben Zuständen, wobei der Roboter jeweils nur in den Zuständen **MOW** und **RUN** die Motoren ansteuert. Wird der Kontakt des Bumpers ausgelöst, erfolgt eine Transition in den Zustand **Blocked**. Nicht modelliert wurde hier, dass dort nur eine Invertierung des letzten Schrittbefehls akzeptiert wird, um den Roboter zurückzubewegen. Sobald der Bumper wieder öffnet, kann im RUN Zustand fortgefahren werden. Der **ERROR** Zustand kann nur durch ein Reset Event verlassen werden, unter der Voraussetzung, dass der e_switch_state nicht mehr **HIGH** ist.
Eine Besonderheit liegt im **SLEEP**-Zustand. Dieser kann nur erreicht werden, wenn keine Verbindung mehr zum ROS-Master besteht. Falls dieser also unerwartet ausfällt, ist durch diesen Zustand sichergestellt, dass der Roboter nicht unkontrolliert weiter fährt.

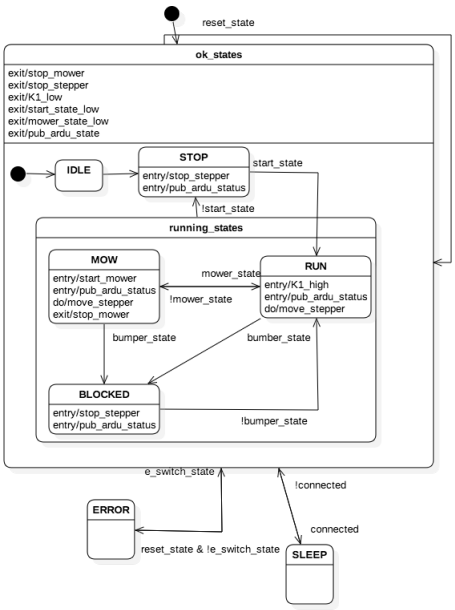


Abbildung 4.10.: Finate State Machine des Arduino Mega Steuerungsboards

[8] siehe Anhang B

Schrittmotorensteuerung

Wie eingangs erwähnt, gestaltete sich die Ansteuerung der Schrittmotoren aus zeitkritischer Sicht etwas schwierig. Im Arduino wurde die Ansteuerung der Schrittmotoren mit Timer-Interrupts gelöst. Arduino bietet für den Mega2560 insgesamt drei Timer, die jeweils verschiedene Pins des μ-Controllers zugeordnet sind. Die TimerOne-Bibliothek nutzt die Pins: 11,12,13 und TimerThree die Pins: 2,3,5. Die Ansteuerung der Schrittmotoren erfolgt über Pulsweitenmodulation (PWM) aus dem Grund, dass für das Ausführen eines Schrittes die Steuerplatine eine steigende Flanke benötigt. Ein digital moduliertes Signal besteht aus dem Wechsel zischen HIGH und LOW am Pinausgang. Die benutzten Timer-Bibliotheken stellen die *pwm(pin, duty, period)*-Methode zur Verfügung, mit der am unterstützten Pin ein PWM-Signal erzeugt wird. Der Duty-Cycle[9] liegt bei 50%, also Pulspause und Puls sind zeitlich gleich lang. Um auch die Rotationsgeschwindigkeit der Schrittmotoren regulieren zu können, muss die Periodendauer[10] des PWM-Signals anpassbar sein.

Einheiten umrechnen

Das ROS-System benutzt **Twist Messages**[11], um Motoren zu steuern und die Geschwindigkeit besitzt die Einheit rad/s. Es muss eine Gleichung aufgestellt werden, die für rad/s die Periodendauer berechnet. Im folgenden ist die Herleitung dieser Gleichung beschrieben.
Als konstante Werte liegen die Schritte S, die ein Schrittmotor für eine Umdrehung benötigt, die Microschritte M, die an der Schrittmotorsteuerung eingestellt ist und das Übersetzungsverhältnis δ des Getriebes vor. Daraus kann die Gesamtzahl der benötigten Schritte für eine Umdrehung U berechnet werden.

$$U = S * M * \delta \tag{4.8}$$

Variabel ist somit nur die Rotationsgeschwindigkeit ω. Für einen Vollwinkel (360°) werden per Definition $2\pi rad$ benötigt und das Verhältnis zwischen dem Vollwinkel und ω ergibt somit, wie lange für eine Umdrehung benötigt wird.

$$\text{U}= \frac{2*\pi*rad}{\frac{\omega*rad}{s}} = \frac{2*\pi*rad*s}{\omega*rad} = \frac{2*\pi*s}{\omega} \tag{4.9}$$

Durch das Einsetzen von Gleichung 4.8 in Gleichung 4.9 resultiert die Periodendauer p in Sekunden. Anschließend wird 1 Sekunde durch 10^6 Microsekunden ersetzt.

$$p = \frac{2 * \pi * s}{\omega} * \frac{1}{U} = \frac{2 * \pi * s}{\omega} * \frac{1}{S * M * \delta} = \frac{2 * \pi * s}{\omega * S * M * \delta} = \frac{2 * \pi * 10^6 \mu s}{\omega * S * M * \delta} \tag{4.10}$$

Motoren ansteuern

Die Periodendauer wird mit der erarbeiteten Formel 4.10 anhand der Zielgeschwindigkeit errechnet. Wenn der Roboter sich in einem der **running_states** befindet, wird *pwm(pin, duty, period)*-Methode aufgerufen und damit der Timer gestartet. Anhand des Vorzeichens der Zielgeschwindigkeit kann die Richtung abgeleitet werden, in der sich ein Schrittmotor drehen soll. Diese Richtung wird durch den DIR-Pin der

[9] Verhältnis zwischen Pulspause und Puls
[10] Die Frequenz, wie schnell sich Pulspause und puls abwechseln
[11] siehe Dokumentation: http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

Schrittmotorsteuerung mitgeteilt. Da der Roboter über eine Differenzialantrieb verfügt, drehen sich die Motoren entgegengesetzt um die selbe Bewegung auszuüben.

Encoder

Aufgrund der Tatsache, dass Schrittmotoren permanent angesteuert werden müssen, um sich zu bewegen, kann auf einen Encoder verzichtet werden. Durch das Ändern der Ansteuerungsart der Schrittmotoren war es herkömmlicher Weise nicht mehr möglich, die Schritte zu zählen. Dies konnte aber durch die Verwendung einer Interrupt service routine (ISR)-Funktion, die mithilfe der *attachInterrupt(function)*-Methode dem Timer zugeordnet wurde, gelöst werden. Die ISR-Funktion zählt dabei eine Countervariable, abhängig von der Richtung in der der Motor angesteuert wird hoch bzw. runter. Dabei ist wieder darauf zu achten, dass der Roboter einen Differenzialantrieb besitzt. Anhand des Counters kann so errechnet werden, wie weit sich ein Rad gedreht hat.

Ultraschall Frontbar

Die Ultraschallsensoren werden nicht mehr über die I^2C-Schnittstelle angesteuert. Das Datenblatt RobotElectronics des SRF02-Ultraschallsensors beschreibt unter anderem die Möglichkeit, den Sensor über einen seriellen BUS anzusteuern. Das Besondere dabei ist, dass ein Autofeedback (Pushing) existiert. Das bedeutet, sobald der Sensor die Entfernung berechnet hat, schickt er diese auf den BUS. Der Vorteil von Pushing gegenüber Polling besteht in diesem Fall darin, dass der Sensor nicht mehr zyklisch abgefragt werden muss. Es genügt nun, zu schauen, ob Daten auf dem BUS geschrieben wurden. Es muss allerdings sichergestellt sein, dass nur ein BUS-Teilnehmer das Schreibrecht für den BUS besitzt. Weil ein gleichzeitiger Betrieb der US-Sensoren ein gegenseitiges Stören zur Folge hat, werden diese sequenziell angesteuert und damit ist nur ein Sensor schreibberechtigt.

Status

Um das restliche System über den aktuellen Status von Knöpfen, Tastern, Kontakten und dem Roboterzustand mitzuteilen wird, eine Statusnachricht verschickt.

4.4. Paketbeschreibung und Implementierung

In diesem Teil des Kapitels werden die implementierten Pakete vorgestellt. Jedes dieser Pakete trägt seinen Teil zur Erfüllung der Aufgabenstellung bei. Angefangen mit dem *three_sd_lidar*-Paket Dieses beinhaltet die Firmware des LIDAR-Systems sowie Testdateien und die Driver-Node. Das Nächste ist das *rosshaun_hw_iface*-Paket, welches die Firmware des Arduino Mega enthält sowie die gleichnamige Schnittstellen-Node. Bei dem *rosshaun_common*-Paket handelt es sich um ein Metapaket, welches weitere Pakete enthält, die zum Betreiben erforderlich sind.

- rosshaun_bringup enthält die Hauptkonfigurationsdateien, Launchdateien und Karten
- rosshaun_description enthält das Robotermodel als urdf-datei und die benötigten STL-Dateien
- rosshaun_msgs enthält alle selbst erstellten Massages und Services

- rosshaun_navigation beinhaltet die Navigations-Konfigurationsdateien, die Launchdateien der externen Pakete aus dem Nav-Stack
- rosshaun_teleop enthält alle wesentlichen Dateien, um den Roboter manuell steuern zu können

Die einzelnen Pakete sind nachfolgend einmal genauer beschrieben.

Three SD LIDAR Paket

Die Firmware und Testdateien des Systems sind im Ordner src hinterlegt und können mithilfe der Arduino-IDE problemlos auf den Microcontroller geladen werden. Im Ordner nodes befindet sich der Driver, der eine Verbindung zum LIDAR-System aufbaut und die im Abschnitt 4.2 beschriebene Filterung und Interpolierung durchführt. Dem ROS-System wird der Laserscan vor und nach jeder Stufe mitgeteilt. Für die eigentliche Aufgabe dient jedoch nur der resultierende Laserscan.

Rosshaun Hardware Interface Paket

Die Firmware für den Arduino Mega ist im Ordner Firmware zu finden und kann ebenfalls über die Arduino-IDE auf den $\mu-$Controller geladen werden. Neben der Firmware beinhaltet das Paket noch den C^{++}-Sourcecode für die Robot-Klasse, die von der *rosshaun_hardware_iface_node* instanziiert wird. Die Roboterklasse erweitert wiederum die Klasse RobotHW aus dem Fremd-Paket *ros_control*. In der *rosshaun_hardware_iface_node* werden die Status-, Ultraschall-Roh- und Encoder-Feedback-Daten aufbereitet und dem restlichen System unter dem jeweiligen Topic zur verfügung gestellt. Zusätzlich hat die Node noch die Aufgabe, dem Arduino die Ansteuerungspunkte der Schrittmotoren mitzuteilen.
Es wurde unter anderem noch ein Feature implementiert, welches eine Änderung des internen Zustands von Knöpfen, Taster und Relays auf dem Roboter ermöglicht. Diese Änderungen nimmt ebenfalls die rosshaun_hardware_iface_node, via eines Service-Requests vor. Damit die Möglichkeit besteht, dass mehrere Remote-Geräte diese Service nutzen können, wurde eine weitere Node in dem Paket implementiert. Die *state_mux*-Node hat die Aufgabe, anhand von einer prioritätenbasierten Topicliste den Topic mit der höchsten Priorität den Vorzug zu lassen. Jedes Topic kann dabei mit einem Timeout versehen werden, um niederpriorisierte Geräte nicht verhungern zu lassen. Die Priorisierung wird in einer Konfigurationsdatei geschrieben, die die Node vom Parameterserver einliest. Nachfolgend ist die momentan benutzte Konfiguration zu sehen. Die *states* geben darüber Auskunft, auf welche Zustände das jeweilige Topic Zugriff haben möchte.

Rosshaun Bringup Paket

Der Zweck dieses Paketes besteht darin, alle Startdateien (Launch-Dateien), Karten und Aufzeichnungen gebündelt in einem Paket zu haben. In dem Ordner /textitlaunch befinden sich die für den Start und Betrieb notwendigen Launch-Dateien. Die *rosshaun.launch*-Datei beinhaltet das Laden sämtlicher Nodes die im Laufe dieser Arbeit getestet wurden. Durch die Übergabe von Argumenten, die standardmäßig auf *false* sind, kann jede Node einzeln geladen werden. So kann der Aufbau bzw. die Zusammensetzung des kompletten Systems mit nur einer Änderung variiert werden. Ein Beispiel ist die manuelle Steuerung.

Listing 4.5: Manuelle Steuerung aus der rosshaun.launch

```
1          <!--MANUEL CONTROL-->
    <arg name="keyboard" default="false"/>
3   <arg name="joystick" default="false"/>
    <arg name="joystick_dev" default="/dev/input/js0"/>
5   <arg name="twist_mux" default="false"/>
    <arg name="yocs_mux" default="false"/>
7
    <!--/////////// cmd mux INPUT \\\\\\\\\\\-->
9
    <group if="$(arg yocs_mux)">
11      <include file="$(find rosshaun_teleop)/launch/yocs_mux.launch"/>
        <group if="$(arg joystick)">
13          <include file="$(find rosshaun_teleop)/launch/joy_teleop.
                launch">
            </include>
15      </group>
        <group if="$(arg keyboard)">
17          <include file="$(find rosshaun_teleop)/launch/keyboard_teleop
                .launch"/>
        </group>
19  </group>
```

Neben den bereits erwähnten Launch-Dateien sind in dem Ordner */config* Konfigurationsdateien, die mithilfe der Launch-Datei in den Parameterserver geladen werden. Zusätzlich sind während dieser Arbeit Bash-Dateien entstanden, die die Arbeit etwas erleichtert haben.

- die **setup.sh** richtet für den PS3-Controller einen Deamon Prozess ein, der automatisch startet; kopiert die Namensregeln der Seriellen Geräte in /etc/udef; kopiert die ros-Arduino-Bibliotheken in den libraries-Ordner.

- die **update.sh** wurde genutzt, um die Software auf dem Roboter aus den Git-Repositories zu aktualisieren; die Roboterbeschreibung zu parsen und den Workspace zu kompilieren

Launcher-Node

Diese Node hat die wichtige Aufgabe, im laufenden Betrieb Launch-Dateien zu laden und auszutauschen. Sie wurde geschrieben um wichtige Ressourcen zu sparen und die Benutzerfreundlichkeit zu erhöhen. Das Anlegen von Karten und das Mähen des Rasen laufen nicht zur selben Zeit ab und aus diesem Grund ist keine Node auf die Ressourcen der anderen angewiesen. Wie im Abschnitt 4.4 zu sehen, besteht die Möglichkeit bequem zwischen den Modi *Karte anlegen* und *Mähen* zu wechseln. Dies wird durch die flexible Nutzbarkeit der *rosshaun.launch* begünstigt.

Rosshaun Description Paket

Die Roboterbeschreibung ist ein wichtiger Bestandteil, um das Model in einem Visualisierungstool darstellen zu können. Die *robot_state_publisher*-Node übernimmt dabei sämtliche Transformationen, die durch die Roboterbeschreibung modelliert werden. Die Joint-Werte werden dabei als Eingabe gelesen und als Ausgabe wird die 3D-Position des jeweiligen Links publiziert (siehe Transformationsbaum in Abbildung C.2). Diese Transformationen werden unter anderem dazu benutzt, um die Position im Model zu bestimmen, aus der eingehende Entfernungsdaten stammen.

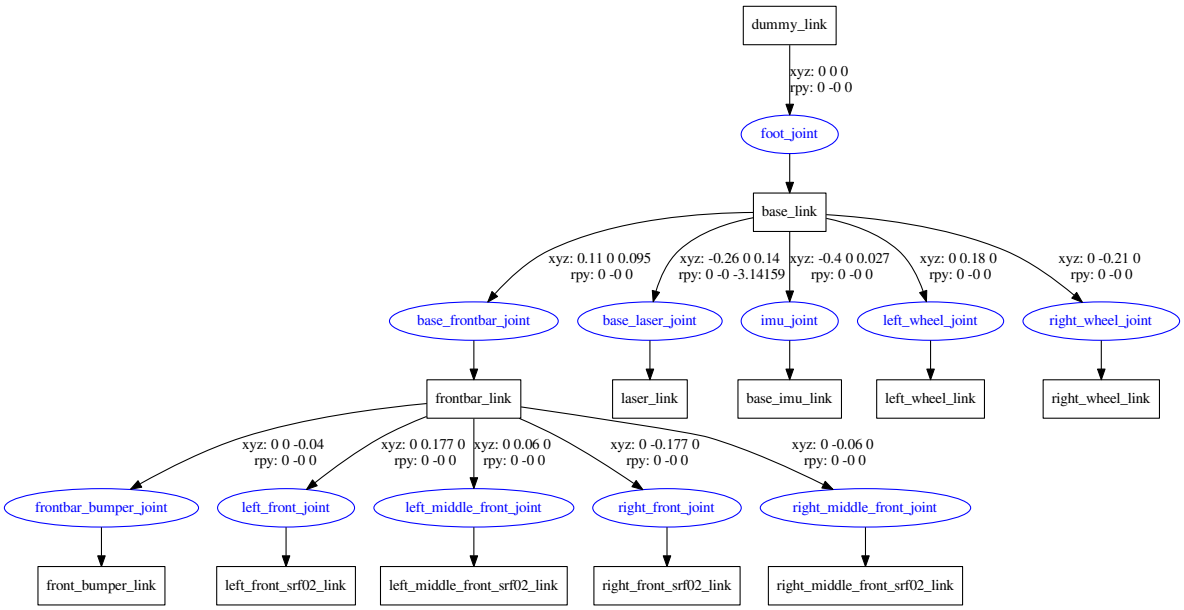


Abbildung 4.11.: Die URDF Beschreibung von Rosshaun als Baumdiagramm

Rosshaun Teleop Paket

Dieses Paket wurde erstellt, um die Nodes zusammenzufassen, die eine manuelle Steuerung ermöglichen. Eine manuelle Steuerung ist bei dieser Arbeit notwendig, um zwei Typen von Karten anzulegen. Die erste Karte ist die, die von Gmapping (siehe Anhang B.6) angelegt wird. Die zweite ist die Perimeter-Karte und wird anhand von gespeicherten Punkten erzeugt. Mehr über dasAnlegen von Karten ist im Abschnitt 4.5 zu lesen.

Eine Steuerung des Roboters ist über die Tastatur oder einen PS3-Controller möglich. Im Anfangsstadium dieser Arbeit wurde überwiegend die Bedienung über die Tastatur genutzt. Diese wurde jedoch mit Implementierung der *PS3-Node* aufgrund der einfacheren Steuerung abgelöst. Ein Wechsel ist, wie in Listing 4.5 zu sehen, jederzeit möglich. Es stellt sich nun aber die Frage welche Befehle zum Roboter geschickt werden sollen, wenn mehrere Nodes die Möglichkeit besitzen den Roboter zu bewegen oder Anweisungen zu schicken.

Die Antwort auf diese Frage wird durch das Multiplexen von mehreren Quellen beantwortet. Ein Mul-

tiplexer wurde bereits in Abschnitt 4.4 vorgestellt und ein weiterer Multiplexer ist dafür zuständig die Bewegungskommandos (Twist-Message) zu kanalisierien. Dies ist in Abbildung 4.12 visuell dargestellt.

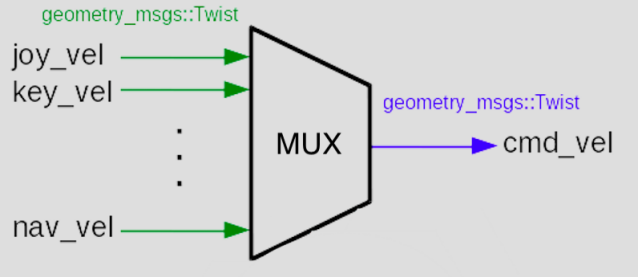


Abbildung 4.12.: Multiplexer Input/Output

Beim zweiten Multiplexer konnte auf ein schon bestehendes Paket zurückgegriffen werden. Anfangs wurde das Paket *Twist_Mux* benutzt, welches jedoch auf dem Raspberry PI nicht gestartet werden konnte. Als Alternative bot sich das *yocs_cmd_vel_mux*-Paket mit der selben Funktion an.
Durch Abbildung A.5 wird die Integration dieses Pakets in das System verdeutlicht.

ps3_2_cmd-Node

Um die Kommandos des Playstation 3-Controllers empfangen zu können, muss dieser mit dem Raspberry Pi verbunden werden. Diese Verbindung wird über Bluetooth hergestellt und muss zuvor eingerichtet werden. Für das Einrichten des Controllers wurde das Packet *ps3joy*[12] genutzt und anhand des dazugehörigen Tutorials[13] konfiguriert. Dadurch, dass die *ps3joy*-Node auf die Hardware des Raspberry Pi zugreift, wird diese mit Rootrechten gestartet. Wie schon im Abschnitt 4.4 erwähnt wurde, wird diese Node als Deamon[14] automatisch beim Start des Betriebssystems ausgeführt. Die von der *ps3joy*-Node publizierte *Joy*-Message[15] wird von der *ps3_2_cmd*-Node ausgewertet. Die Belegung ist in Abbildung 4.13 (a) und (b) zu sehen.

yocs_cmd_vel_mux-Node

Für diese Node wird, wie schon bei dem State-Multiplexer, eine Konfigurationsdatei benötigt, die die eingehenden Topics sowie deren Priorität beinhaltet. (siehe Listing 4.6)

Listing 4.6: Konfigurationsdatei für die yocs_cmd_vel_mux-Node

```
subscribers:
  - name:        "Navigation stack"
    topic:       "/nav_vel"
    timeout:     0.5
    priority:    1
    short_desc:  "Navigation stack controller"
```

[12]Link zum Paket http://wiki.ros.org/ps3joy
[13]Tutorial zum Einrichten des Controllers http://wiki.ros.org/ps3joy/Tutorials/PairingJoystickAndBluetoothDongle
[14]Link zum Tutorial http://wiki.ros.org/ps3joy/Tutorials/Running%20ps3joy.py%20as%20daemon
[15]Doku http://docs.ros.org/jade/api/sensor_msgs/html/msg/Joy.html

(a) 0-reset, 1-Steuerhebel, 3-starten/stop, 12-E-Stop, 14-Position speichern, 16-Koppeln

(b) 8-Karten speichern, 9-Mähmotor an, 10-Karten anlegen, 11-Mähen

Abbildung 4.13.: PS3 Controller Button Belegung

```
7   - name:         "Onboard joystick"
      topic:        "joy_vel"
9     timeout:      0.5
      priority:     10
11  - name:         "Keyboard operation"
      topic:        "key_vel"
13    timeout:      0.5
      priority:     7
15 publisher:       "cmd_vel"
```

4.5. Karte anlegen

In diesem Kapitel wurden bisher alle essentiell wichtigen Pakete vorgestellt, die eine manuelle und autonome Steuerung ermöglichen. Diese autonome Steuerung wäre allerdings orientierungslos. Um die Aufgabenstellung aus Kapitel 1 zu erfüllen wird eine Karte der Umgebung benötigt. Diese Karte dient dem Roboter während der Kartenerstellung (siehe Abschnitt 3.1) und des Mähvorgangs(siehe Abschnitt 4.6) zur Lokalisierung. Zum Erstellen dieser Karte wurde auf das bereits implementierte Paket *Gmapping* zurückgegriffen, welches schon im Abschnitt B.6 erklärt wurde. Um dieses Paket in das System einzubinden, wurde eine Launch-Datei erstellt, die alle einstellbaren Parameter enthält, um Gmapping zu konfigurieren. Neben der Karte wird zusätzlich eine Perimeterkarte angelegt, die dem Roboter während des Mähvorgangs die Zone vorgibt, in der gemäht werden soll. Die Erstellung dieser Karten wird nachfolgend erklärt.

Mappingmodus

Das Starten der *initial.launch*-Datei muss von einem Remote-PC augeführt werden und ist Voraussetzung, um in den *Mapping*-Modus zu gelangen (siehe dazu Anlage A). Mithilfe des PS3-Controllers wird die Erstfahrt durchgeführt und muss dazu mit dem Roboter verbunden sein. Die Kopplungs-Taster (16 siehe Abbildung 4.13) des Controllers sorgen für die Herstellung der Verbindung. Mit anschließendem Drücken

auf Taste L1(10) wird die *mapping.launch*-Datei über die *launcher*-Node geladen. Nach erfolgreichem Laden sind jetzt alle für das Kartographieren erforderlichen Nodes bei dem Master angemeldet. Auf dem Remote-PC kann durch das Ausführen der *pc.launch*-Datei das Kartographieren in rviz verfolgt werden. Es wurden bereits einige Routenvariationen für das Kartieren der Umgebung getestet. Diese Tests sind im Kapitel 5 zu finden. Die Fahrtabfolge, die das beste Ergebnis geliefert hat, wird an dieser Stelle näher erläutert.

äußere Grenzen

Nachdem das System soweit vorbereitet wurde, kann die Fahrt beginnen. Um den Mähroboter zu bewegen, wird der linke Stick (1) benutzt. Um die Eckpunkte der Perimetergrenze festzulegen, wird die Taste X (14) benötigt. Als erstes werden die äußeren Grenzen abgefahren. Dazu wird in einem Abstand von ca. 1m entlang dieser gefahren. Werden Eckpunkte erreicht, muss mit dem Roboter bis zu dem Punkt gefahren werden, der einen Eckpunkt darstellen soll. Ist dieser Punkt erreicht, wird mit der X-Taste bestätigt und die *field_mapper*-Node speichert das Koordinatenpaar {x,y} ab. Ist die gesamte äußere Grenze abgefahren, kann in rviz überprüft werden, ob die Karte noch Lücken aufweist.

Lücken schließen

Nicht kartographierte Punkte werden durch parallele Fahrten entlang einer markanten Grenze geschlossen. So hat der Scanmatcher von Gmapping (siehe Abschnitt B.6) die Möglichkeit, Ähnlichkeiten zwischen eingehenden Scans und der bisher angelegten Karte zu finden.

Speichern der Karten

Wenn die erstellte Karte ein sauberes geschlossenes Bild aufweist, wie in Abbildung 4.14 zu sehen, muss diese durch das Drücken der L2-Taste am Controller gespeichert werden. Ebenfalls werden die aufgezeich-

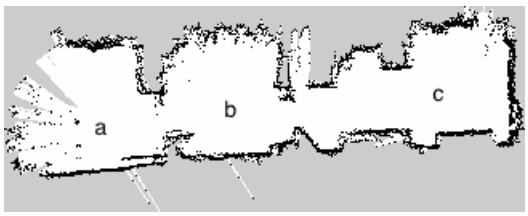


Abbildung 4.14.: Beispielsaubere Umgebungskarte a = Küche, b = Spielzimmer, c = Wohnzimmer

neten Grenzpunkte in eine Konfigurationsdatei exportiert. Gleichnamige bestehende Karten werden dabei überschrieben. Damit eine Karte nicht unabsichtlich überschrieben werden kann, können diese nur im *Mapping*-Modus gespeichert werden. Ist die erstellte Karte nicht sauber aufgezeichnet worden, wie in Abbildung 4.15 zu sehen, muss das Kartographieren erneut durchgeführt werden. Gründe für das unsaubere Aufzeichnen können eine zu schnelle Fahrt oder Drehung sein.

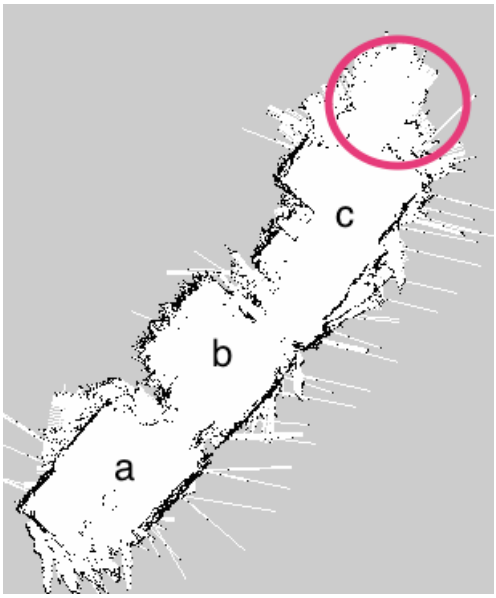


Abbildung 4.15.: Beispiel unsaubere Umgebungskarte a = Küche, b = Spielzimmer, c = Wohnzimmer

4.6. Der autonome Mähvorgang

Dieses Thema war eines der schwierigsten in dieser Bachelorarbeit und birgt das Potenzial für mehrere Arbeiten. Das *au_automow_common*-Paket wurde aus Komplexitätsgründen und zur Erfüllung der Aufgabenstellung adaptiert und an die aktuelle ROS-Version angepasst. Nähere Details über das Paket und die Anpassungen sind im Anhang B.7 beschrieben.

Der autonome Mähvorgang kann nur durchgeführt werden, wenn die Umgebungskarte und die Perimeterkarte existieren. Diese sollten wie in Abschnitt 4.5 beschrieben, angelegt und gespeichert werden. Existieren diese beiden Karten für das aktuelle Einsatzgebiet, kann durch das Drücken der R1-Taste(11) am Controller der *Mowing*-Modus gestartet werden. Wie schon zuvor startet auch jetzt die *Launcher*-Node die entsprechende Launch-Datei. In diesem Fall ist es die *mowing.launch*-Datei. Falls direkt aus dem Mapping-Modus in den Mowing-Modus geschaltet wird, werden alle Nodes beendet, die für das Kartographieren benötigt wurden. Dies spart wertvolle Ressourcen, die für die Lokalisierung, Planung und Navigation zwingend erforderlich sind.

Laden der Karten

Nach Aktivierung des Mowing-Modus wird die *map_server*-Node mit dem Pfad zur Umgebungskarte geladen. Diese publiziert nun die Karte mit der Frame-id[16] */map*. Auf diese Frame-id wird der restliche TF-Tree eingehängt. Neben der Umgebungskarte wird zusätzlich noch die Perimeterkarte eingebunden. Dies wird mit der *field_publisher*-Node erzeugt. Die drei erzeugten Grenzen sind in Abbildung 4.16 zu sehen.

[16] siehe Anhang B.4

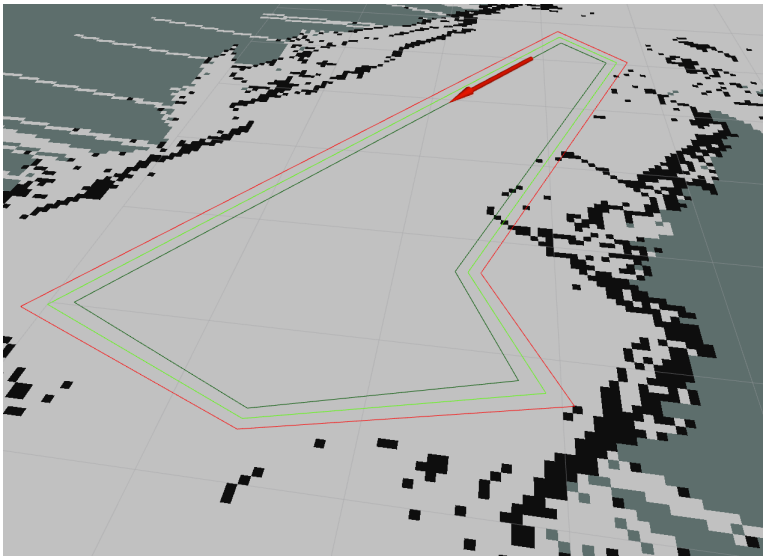


Abbildung 4.16.: Zonen in rviz: hellgrün=sichere Zone, dunkelgrün=Mähzone, rot=Begrenzung

Linienfahrt planen

Sobald die Polygone berechnet und veröffentlicht wurden, beginnt die *path_planner*-Node mit der Planung der Linienfahrt innerhalb der sicheren Zone. Diese Planung basiert ausschließlich auf der Grundlage der Perimetergrenze. Hindernisse, die sich innerhalb der Planungszone befinden, finden keine Berücksichtigung in der Planung. Diese Hindernisse werden erst durch Move Base erkannt und umfahren. Der entsprechende Plan für die Zonen aus Abbildung 4.16 nachfolgend zu sehen.

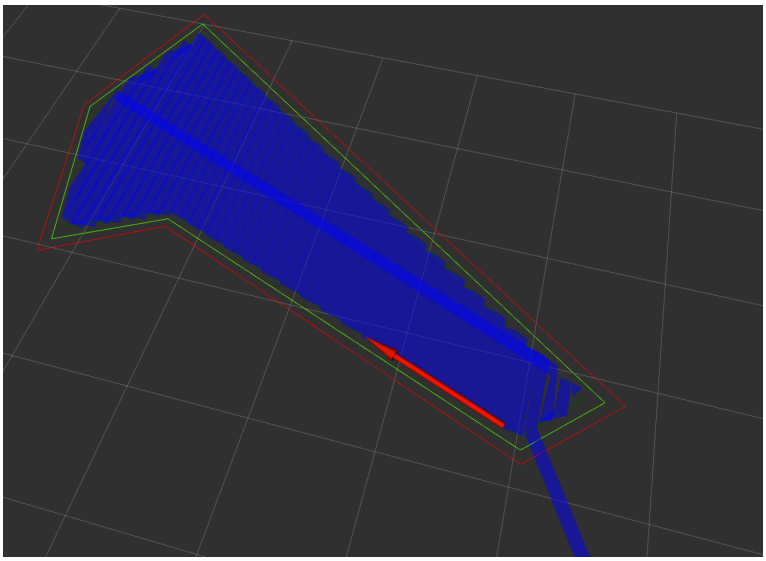


Abbildung 4.17.: Der Plan für die Linienfahrt wobei die Breite des Pfades dem Durchmesser des Mähtellers

Anfahren von Wegpunkten

Sobald der Plan für das Mähen erstellt wurde, kann dieser in rviz visualisiert werden. Dieser Plan wird von der Planner-Node durch das Versenden von Wegpunkten an Move Base abgearbeitet. Move Base plant auf den Grundlagen von local und global Costmap sowie den eingestellten Parametern, den Weg zum erhaltenen Wegpunkt. Ist der Weg zu einem Wegpunkt berechnet, versendet Move Base *Twist*-Messages. Diese sorgen dafür, dass der Weg abgefahren wird. Move Base überwacht daraufhin die Position und Ausrichtung des Roboters und stellt Nachberechnungen an bzw. ändert den Inhalt der Twist-Message. Ein Ziel gilt als erreicht, sobald der Roboter sich in einer bestimmten Epsilon-Umgebung des Ziels befindet. Dies soll ein osszillieren um einen Punkt verhindern. Nach erreichen eines Ziels wird der Planer darüber in Kenntnis gesetzt und versendet den nächsten Wegpunkt. Diese Schleife wird solange fortgeführt, bis der Roboter alle Ziele erreicht hat oder stehen bleibt.

Der Mähvorgang

Der Mähmotor wird mithilfe der *cutter_control*-Node angesteuert, die darauf achtet, ob sich der Mähmotor innerhalb der *Mähzone* befindet. Diese Zone wird ebenfalls von der *field_publisher*-Node berechnet und zur Verfügung gestellt.

Sicherheit

Durch die implementierten Sicherheitsstufen *state_mux*-Node und dem Arduino Steuerungsboard, ist gewährleistet, dass über die PS3-Fernbedienung der Mähmotor abgestellt sowie die Steuerung übernommen werden kann. Zusätzlich wird direkt beim Schließen des Stoßfängers der Mähmotor abgestellt. Dies deckt der Zustandsübergang (Abbildung 4.10) von **MOW** in **BLOCKED** ab.

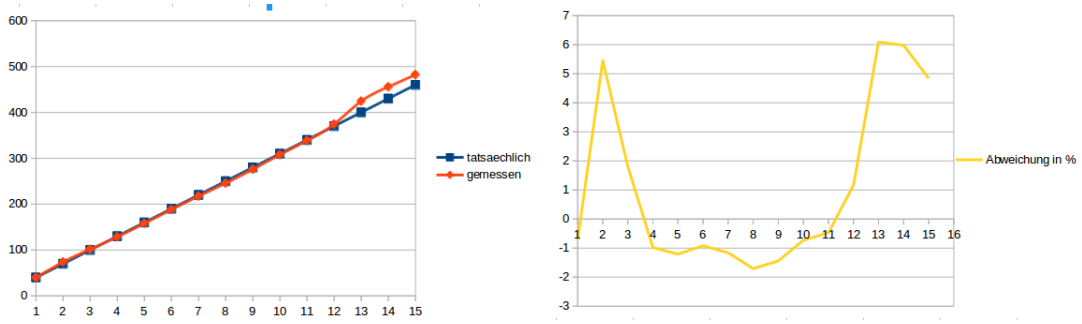
5. Evaluation

In diesem Kapitel gilt es nun zu zeigen, wie gut sich das in dieser Arbeit entwickelte System eignet, um die in Kapitel 2 entworfene Aufgabenstellung zu erfüllen.
Als Testumgebung für kleinere Fahrten und Aufzeichnungen wurde Erdgeschoss meines Hauses benutzt und Testfahrten im Aussenbereich, wurden in einem Garten zur durchgeführt.
Der erste Testfall betrifft das entwickelte LIDAR-System und soll die Qualität des Scans in Abhängigkeit zur Entfernung analysieren. Die darauf folgenden Tests sollen Aufschluss darüber geben, mit welcher Fahrweise die besten Ergebnisse bei der Kartierung der Umgebung gemacht werden können. Diese Tests wurden im Innen sowie im Aussenbereich durchgeführt. Der Finale Test soll schließlich aufzeigen, welche Lokalisierungsprobleme, die in Abschnitt 3.2 behandelt wurden, gelöst werden können.

5.1. Lidar

Der Test des Entfernungssensors wurde mit dem fertigen System durchgeführt. Dabei wurden die Qualität und Genauigkeit der Entfernungsmessungen genauer überprüft. Als Testumgebung wurde ein Innenraum gewählt, um ein unverfälschtes Testergebnis zu erhalten. Der Sensor wurde dabei drei unterschiedlichen Tests unterzogen, die nachfolgend dokumentiert sind.

5.1.1. Genauigkeitstest



(a) Graphen der tatsächlichen und gemessenen Entfernung

(b) Fehlerabweichung in %

Abbildung 5.1.: Testresultat Genauigkeitstest

Die Genauigkeit wurde getestet, damit eine Aussage darüber getroffen werden kann, bis zu welcher Entfernung dieser Sensor genutzt werden kann. Dabei wurde die tatsächliche Entfernung zu einem Hindernis mit der gemessenen Entfernung verglichen. In der Datenreihe, die erhoben wurde, bestand ein Messwert aus zehn aufeinanderfolgenden Entfernungen in einem Scan (Median) und diese Messungen wurden schrittweise mit Abständen von 30 cm durchgeführt. Aus den Abbildungen in 5.3 geht hervor, dass in einer Entfernung von ca. 80 cm der Fehler kurzzeitig auf 6% ansteigt. Der Grund ist, dass bei dieser Entfernung

der Infrarotsensor in die Berechnung mit aufgenommen wird. Dieser liefert in diesem Bereich eine kleine Abweichung. Wird der Verlauf der beiden Graphen bis auf 480 cm verfolgt, kann festgestellt werden, dass diese fast korrilieren. Der Fehler lag in einem Bereich von -1% und -2% wie aus Abbildung 5.1(b) entnommen werden kann. Dies stellt einen akzeptablen Wert dar. Anzumerken ist jedoch, dass ab einer Entfernung von 380 cm der Fehler sprungartig auf 6% ansteigt. Der Ursprung dieses Problems liegt in der Geschwindigkeit, in der sich der Sensor dreht. Aufeinanderfolgende Entfernungen weisen starke Schwankungen, wie in Abbildung 5.1.1 auf. Um dieses Problem zu lösen könnte ein weiterer Filter entwickelt werden, der diese Schwankungen glättet.

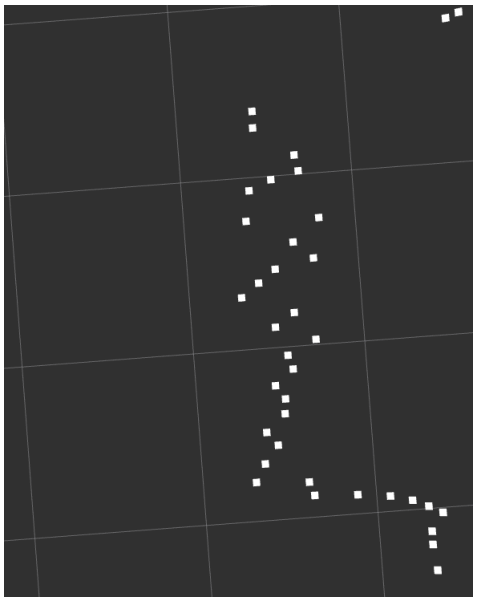
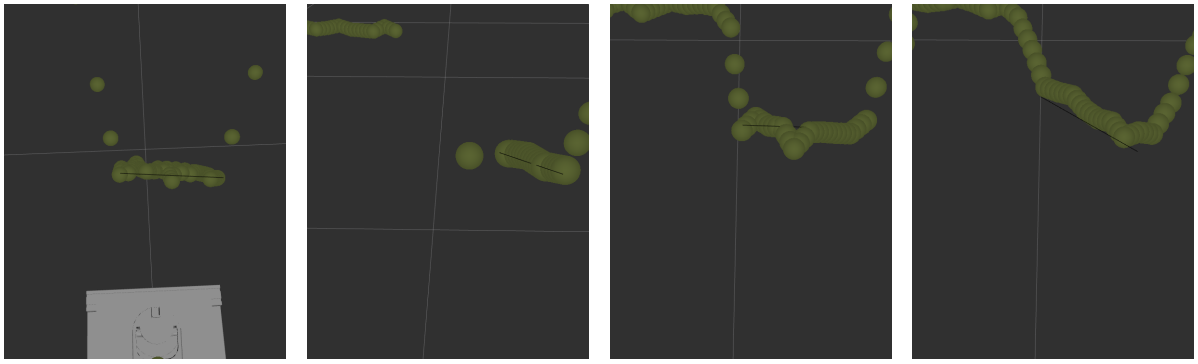


Abbildung 5.2.: nicht interpoliertes Entfernungsbild einer Glatten Wand in 380 cm Entfernung

5.1.2. Reflektionswinkel-Test

Dieser Test soll Aufschluß darüber geben, ob der Winkel, mit dem ein Objekt zum Sensor steht, Auswirkung auf die Erkennung hat. Dabei wurde dieser Test in zwei Entfernungen durchgeführt. Im Nahbereich, um die Ultraschallsensoren zu testen und in einer Entfernung von 130 cm für den Infrarot-Sensor. Als Reflexionsobjekt wurde ein handelsüblicher Aktenordner benutzt, der eine Breite von 28 cm aufweist. In Abbildung 5.3(a) ist zu sehen, dass die Ultraschallsensoren den Ordner vollständig erkennen. Die in rviz gemessene Breite des Ordners betrug ca. 32 cm. Eine Drehung des Ordners um 30° hatte zur Folge, dass dieser nicht mehr komplett wahrgenommen wurde und so eine Lücke im Bild entstand. Dies ist in Abbildung 5.3(a) zu sehen. Dieser Test wurde für den Infrarotsensor wiederholt und ist in den Abbildungen 5.3(d) und 5.3(c) zu sehen. Der Infrarotsensor hat in beiden Ausrichtungen des Ordners diesen gut erkennen können. Die Messung der Breite im Sensorbild ergab jeweils ca. 26 cm. Um das Problem im Nahbereich zu beseitigen könnten die Ultraschallsensoren durch weitere Infrarotsensoren ausgetauscht werden.

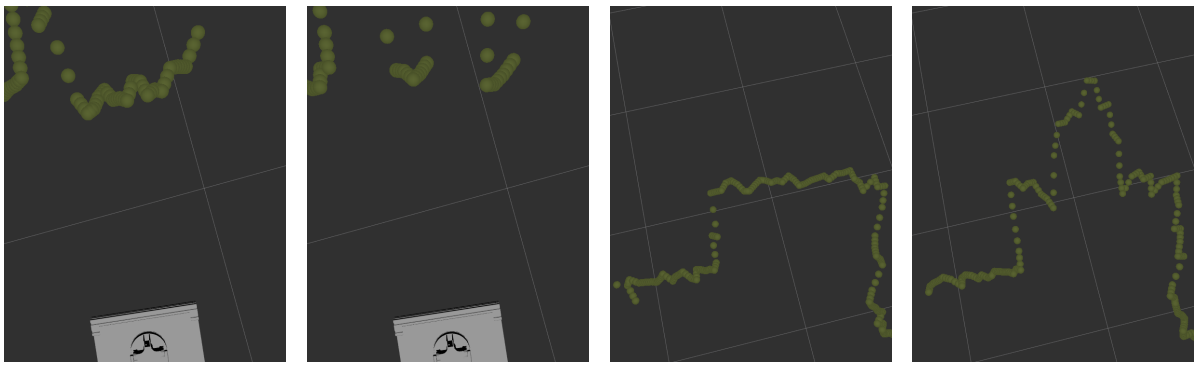


(a) Ordner parallel zum Sensor in 60cm Entfernung (b) Ordner um 30° verschoben in 60cm Entfernung (c) Ordner parallel zum Sensor in 130cm Entfernung (d) Ordner um 30° verschoben in 130cm Entfernung

Abbildung 5.3.: Testresultat Reflektionswinkel

5.1.3. Reflektionsmaterial-Test

Der letzte Test, der für diesen Sensor durchgeführt wurde, ist ein Test, der das Verhalten auf unterschiedliche Materialien prüfen soll. Es wurden vier Materialen ausgewählt, die jeweils unterschiedliche Strukturen aufwiesen. Diese Materialien waren eine Decke, bekleidete Beine, ein raues Sofa und ein Spiegel. Der Spiegel soll eine weitere Limitierung aufzeigen. Die in 5.4 zu sehenden Abbildungen zeigen das Erwartete



(a) Decke gewellt (b) Hosenbeine (c) Sofa rau (d) Spiegel vor dem Sofa

Abbildung 5.4.: Entfernungsbilder beim Materialtest

Ergebnis. Bis auf Abbildung 5.4(d) wurden alle getesteten Oberflächen gut erkannt. Der Spiegel zeigt die Einschränkung des Infrarotsensors, gegenüber stark reflektierenden Gegenständen auf.

5.2. Testfahrten

In diesem Abschnitt werden unterschiedliche Fahrweisen getestet und das jeweilig entstandene Kartenmaterial verglichen. Aus dem Vergleich soll eine Fahrweise gewonnen werden, die das bestmögliche Ergebnis dieses Systems liefert. Dabei wurden eine schnelle (100 Umdrehungen pro Minute (UpM)) und eine langsame (30 UpM) gradlinige Fahrweise getestet. Eine zusätzliche Fahrt sollte darüber Aufschluss geben, ob die Ausrichtung des Entfernungssensors eine bessere Karte erzeugen würde. Dazu wurde die Kartierung beim Rückwärtsfahren durchgeführt.

Vergleich Fahrweise

In 5.5 sind die drei Karten zu sehen, die in diesem Test entstanden sind. Ein optischer Vergleich zeigt, dass bei schneller Fahrt die Karte 5.5(a) nicht vollständig gezeichnet wurde (rote Kreise). Ein weiterer Unterschied ist im Sitzbereich (grünes Viereck) zu sehen, der nicht so sauber gezeichnet wurde, wie in der zweiten und dritten Karte 5.5(b) und 5.5(c). Am aufschlussreichsten ist allerdings die zusätzliche fahrt, die rückwärts durchgeführt wurde. An der erstellten Karte ist zu erkennen, dass die gezeichneten Grenzen gegenüber den anderen beiden sehr satt gezeichnet wurden. Das bedeutet, dass die einzelnen Scans gut aufeinander gepasst haben.

(a) Kartendarstellung bei schneller Fahrt von Punkt a nach b, ohne Wendung

(b) Kartendarstellung bei langsamer Fahrt von Punkt a nach b, ohne Wendung

(c) Kartendarstellung bei langsamer Fahrt (rückwärts) von Punkt a nach b, ohne Wendung

Abbildung 5.5.: Kartenvergleich zwischen langsamer und schneller Fahrt ohne Wendung

Die nächsten drei Karten, aus Abbildung 5.6, sind aus der Fortsetzung der ersten Fahrt entstanden. Dabei wurde eine 180-Grad-Wendung vollzogen und zurück zum Ausgangspunkt gefahren. In den beiden ersten Karten ist zu erkennen, dass ein Versatz (rote Kreise) stattgefunden hat. Die dritte Karte weist diesen Fehler jedoch nicht auf und bestätigt damit, dass der Sensor am besten nach vorne ausgerichtet sein sollte. Zusätzlich ist noch zu erkennen, dass im Wohnbereich durch die Wand gezeichnet wurde (blaue Kreise). Dieser Fehler ist darauf zurückzuführen, dass der Entfernungssensor ca. zwei Sekunden für einen Scan

benötigt und bei Drehungen wird dieser Scan verzogen. Dieser Fehler könnte durch das Zerteilen des

(a) Kartendarstellung bei schneller Fahrt von Punkt a nach b, mit Wendung und schneller Fahrt zurück zu Punkt b

(b) Kartendarstellung bei langsamer Fahrt von Punkt a nach b, mit Wendung und langsamer Fahrt zurück zu Punkt a

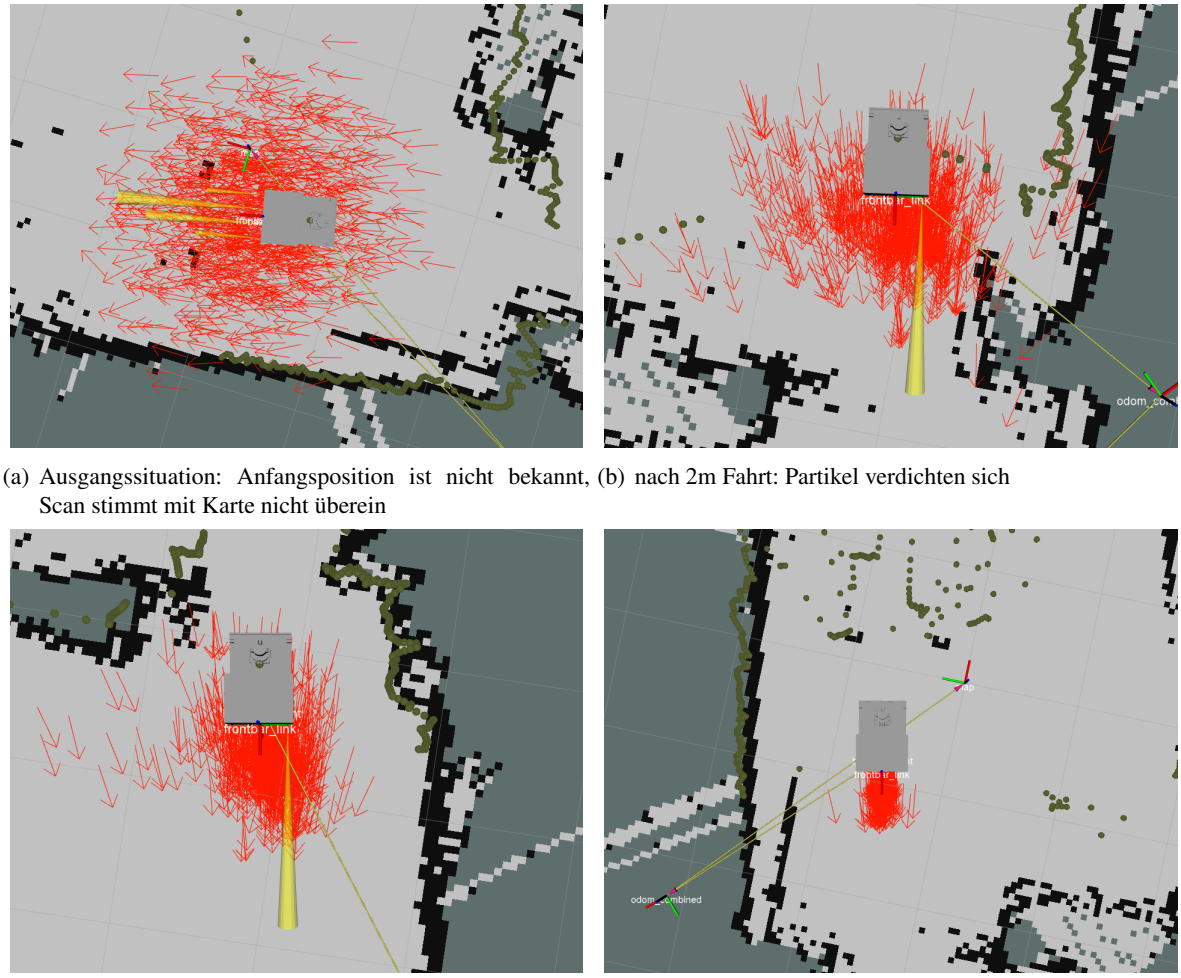
(c) Kartendarstellung bei langsamer Fahrt (rückwärts) von Punkt a nach b, mit Wendung und langsamer Fahrt (rückwärts) zurück zu Punkt a

Abbildung 5.6.: Kartenvergleich zwischen langsamer und schneller Fahrt mit Wendung

ganzen Scans in kleinere Sectoren behoben werden. Das würde eine Erhöhung der Frequenz bewirken, in der die Teilscans zur Kartographierung empfangen werden. Eine andere Möglichkeit wäre das Ignorieren von Scans während einer Drehung. Die saubersten Grenzen wurden allerdings in allen Tests durch das langsame Fahrverhalten erzeugt, da für den selben Weg mehr Scans in die Kartierung eingeflossen sind. Die Schlussfolgerung aus diesem Test ist, dass das beste Kartenmaterial entsteht, wenn langsam rückwärts gefahren wird.

5.3. Test Lokalisierung

In diesem Test wird demonstriert, welche der bekannten Lokalisierungsprobleme mit diesem System gelöst werden können. Anhand der erstellten Karten aus dem vorherigen Test ist zu erkennen, dass das lokale Lokalisierungsproblem gelöst wurde. Diese Karten würden ohne eine Positionsverfolgung nicht entstehen können. Das Globale Lokalisierungsproblem soll durch die Anwendung des AMCL-Algorithmus[1] auf der Karte, die in 5.6(c) erstellt wurde, getestet werden.



(a) Ausgangssituation: Anfangsposition ist nicht bekannt, Scan stimmt mit Karte nicht überein

(b) nach 2m Fahrt: Partikel verdichten sich

(c) nach 4m Fahrt: Scan passt zur Karte, Position gefunden

(d) zurück am Ausgangspungt: exakte Position wurde bestimmt

Abbildung 5.7.: Globales Lokalisierungsproblem Test in rviz mit Partikelwolke

Anhand der Abbildungen, die in 5.7 zu sehen sind, kann unter der Voraussetzung, dass langsam gefahren wird gesagt werden, dass das globale Lokalisierungsproblem ebenfalls gelöst werden konnte. Um zu testen, ob das Kidnapped Robot-Problem ebenfalls gelöst werden konnte, wurde der Roboter von der Position in 5.7(d) (Küche) zwei Zimmer weiter in das Wohnzimmer gebracht und auch noch um 180° gedreht. Nach 10 min Fahrt wurde der Test abgebrochen und als nicht bestanden gewertet. Das dieser Test nicht bestanden

[1] siehe Abschnitt 3.4.3

wurde, konnte nicht auf die benutzten Pakete zurückgeführt werden, sondern auf die geringe Frequenz des Entfernungssensors.

5.4. IMU Test

Da es Probleme gab, die Daten des IMU-Boards mit den Odometriedaten zu fusionieren, wurden die Daten des Sensors mit rqt (siehe Anhang B.1) als Plotgrafik dargestellt. Dieser Test hat ergeben, dass in nicht regelmäßigen Abständen die Daten ungewöhnliche Ausschläge aufweisen. Diese Anomalie ist in Abbildung 5.8 zu sehen. Aufgrund dieses Fehlers, dessen Ursache nicht gefunden werden konnte, ist das IMU-Board zwar vollständig in das System integriert, wird aber nicht genutzt.

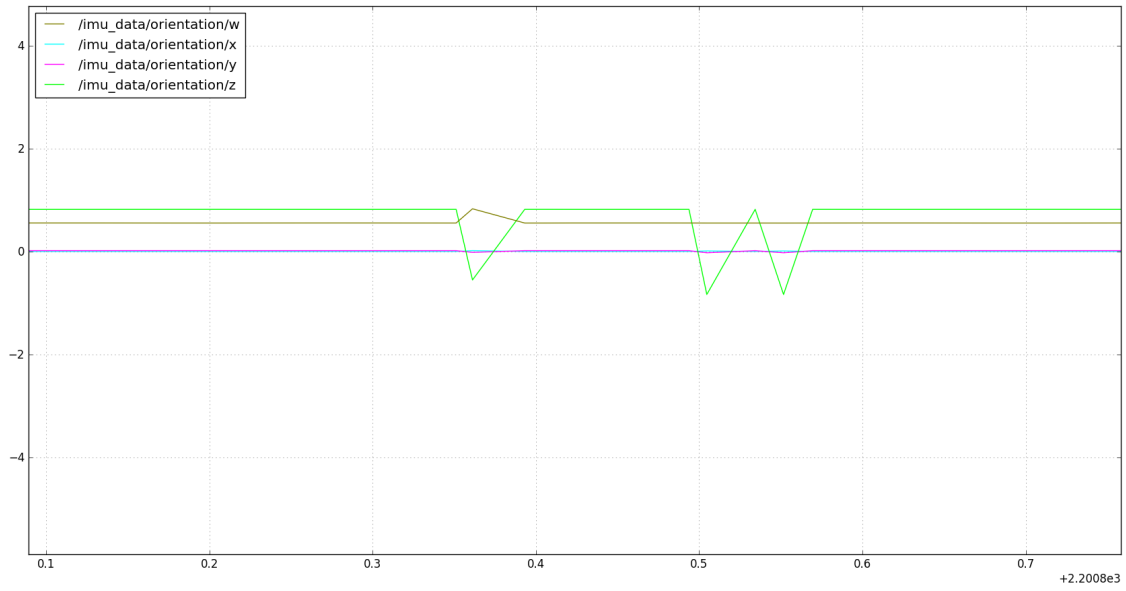


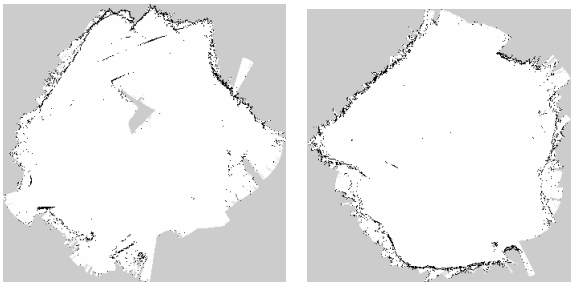
Abbildung 5.8.: Plot des IMU-Fehlers

5.5. Testfahrten im Außenbereich

Auf der Grundlage der vorangegangenen Tests sollte mit diesen Tests ein Bewegungsmuster gefunden werden, dass das beste Kartenmaterial im Außenbereich liefert. Die vorherrschenden Witterungsverhältnisse ließen nicht viele Tests in diesem Bereich zu. Einige wenige Tage konnten aber genutzt werden um Tests durchzuführen. Nachfolgend werden die Ergebnisse begründet dargelegt.

Kartierung der Umgebung

Für die Kartierung der Umgebung wurden zwei verschiedene Bewegungsmuster getestet. Die erste war das Abfahren der Grenzen und Auffüllen von Zwischenräumen und das zweite Muster bestand darin parallele Bahnen in einem Abstand von 2 m zu fahren. Beide Muster führten aber zu kein befriedigendes Ergebnis, wie in Abbildung 5.9 zu sehen ist.



(a) Bewegungsmuster: zuerst Grenzen dann der Innenraum

(b) Bewegungsmuster: parallele Bahnen

Abbildung 5.9.: Testfahrt im Außenbereich

Die Gründe für dieses falsche Kartenmaterial, ist die Tatsache, dass der Odometriefehler bei Kurven zu groß ist, als dass er von Gmapping kompensiert werden kann. Erschwerend kommt hinzu, dass das verwendete LIDAR-System eine geringe Reichweite besitzt und somit zu wenige Landmarken in das Modell einfließen. Innere Landmarken wurden gänzlich, durch ein überzeichnen, aus der Karte entfernt. Während der Testfahrten wurde jedoch beobachtet, dass das erste Bewegungsmuster am vielversprechensten aussah, was darauf zurückzuführen ist, dass weniger Kurven gefahren wurden.

6. Zusammenfassung und Ausblick

In der aktuellen Konfiguration von Hard- und Software ist der Mähroboter in der Lage eine Karte von seiner Umgebung anzulegen. Zusätzlich wird eine zweite Karte angelegt, die das zu mähende Gebiet vorgiebt. Diese Karten können gespeichert und geladen und dadurch zur Lokalisierung genutzt werden.
Nachfolgend werden die positiven Aspekte dieser Arbeit benannt und anschließend die Probleme erläutert, die aufgetreten sind. Zukünftige mögliche Arbeiten werden zum Schluss dieses Kapitel genannt.

6.1. Positive Elemente dieser Arbeit

Mit dieser Arbeit ist ein komplexes und dank des Robot Operating Systems, gut erweiterbares, strukturiertes, modernes und lauffähiges Software-Paket in nur sehr kurzer Zeit entstanden. Dank diesem Framework konnte der Fokus auf die Problemstellung dieser Arbeit gelegt werden. Um das System jedoch einsetzen zu können, gab es sehr viel zu lernen und viele frustrierende Stunden aber auch Glücksmomente, wenn ein Teil des Systems das tat, was es sollte. Mit den vielen zur Verfügung stehenden Paketen konnte, bezogen auf das Einsatzgebiet und der verwendeten Hardware, eine Auswahl getroffen werden, die am besten zu dem Roboter passte. Auch wenn nicht alle Ziele erreicht wurden konnte jedoch verifiziert werden, aus welchen Gründen diese nicht erreicht wurden. Der Entwurf eines eigenen Sensorsystems war eine große Herausforderung. Als die erste Version dieses Sensors nicht die erwarteten Ergebnisse liefern konnte, war das nach zweiwöchiger Entwicklungsphase ein riesiger Rückschlag. Daraufhin wurde eine neue Version entworfen, mit der die aufgetretenen Fehler behoben wurden. Dieses System ist dann auch zum Einsatz gekommen. Das Steuern über einen PS3-Controller erleichterte das Fahren des Rasenmähroboters im Rahmen dieser Arbeit.

6.2. Probleme und Lösungstheorie

Probleme gab es während und sind auch noch nach Abschluss dieser Arbeit vorhanden. Durch den Eigenbau eines Entfernungsscanners war es möglich, eine akzeptable Karte zu zeichnen. Jedoch war dieses mit einigen Einschränkungen verbunden, die leider erst während der Testphase aufgetaucht sind. Die vorherrschenden Witterungsverhältnisse verhinderten sorgte nicht dafür, dass jeden Tag im freien getestet werden konnte. Einige trockene Tage wurden jedoch genutzt um Tests im Freien durchführen. An manchen Tagen, wenn die Luftfeuchtigkeit sehr hoch war, lieferte das Sensorsystem allerdings sehr konfuse Daten. Schuld daran sind die Ultraschallsensoren, denn die kleinen Wassertropfen in der Luft reflektieren die Ultraschallwellen. Das sorgt dafür, dass kein brauchbares Sensorbild entsteht und somit weder das Anlegen von Karten, noch das Mähen funktionierten. Eine weitere Limitierung ist die Geschwindigkeit, in die das System Messwerte liefert. Aus diesem Grund bringt nur eine sehr langsame Fahrt brauchbares Kartenmaterial. Die geringe Reichweite ist auch nicht von Vorteil, wenn ein größerer Garten kartographiert werden soll. Um die Entfernung und die Frequenz des Sensors zu erhöhen kann auf den LIDAR-Sensor *LIDAR-lite-V3* zurückgegriffen werden. Dieser ist seit Januar 2017 wieder auf dem Markt erhältlich und hat eine Reichweite von ca. 40m bei einer Frequenz von max. 500Hz. Dieser Sensor würde alle genannten Limitierungen beheben und zudem den Einsatz von Hector-SLAM[1] ermöglichen. Dies wäre effizienter als Gmapping.

[1] Link zum ROS-Paket: http://wiki.ros.org/hector_slam

Ein weiteres Problem verbirgt sich hinter dem eingebauten IMU-Sensor. Dieser springt bei eingeschalteten Roboter um die z-Achse. Dieses Phänomen ist kein Drift sondern wird durch eine elektrische Quelle verursacht, die nicht genau bestimmt werden konnte. Am PC angeschlossen sind die gelieferten Werte in Ordnung. Aus diesem Grund konnten diese Daten nicht zur Fusion mit den Odometriedaten herangezogen werden.
Der Antrieb des Roboters inklusive der Relays stellen weitere Probleme dar, die nicht mehr behoben werden konnten. Der Antrieb sorgt dafür, dass der Roboter ziemlich laut ist und die Batterien sehr schnell leer sind. Es wurde viel Zeit damit verbracht, die Schrittmotoren auf ein gutes Verhältnis einzustellen. Im Haus kann zum Beispiel ohne Problem mit einer Stromabsenkung von 100% gefahren werden. Auf einer Rasenfläche kann der Strom nur maximal 50% gesenkt werden, was den Stromverbrauch selbst bei Nichtfahren in die Höhe treibt. Aus diesem Grund ist es ratsam, den Antrieb durch Gleichspannungsmotoren mit eingebautem Encoder und Getriebe auszutauschen. Das Auswechseln der Motoren hat zudem den Vorteil, dass die Relays entfernt werden können. Die Relays sorgen beim Schalten für Spannungsspitzen, was manche digitalen Eingangspins als logische 1 wahrnehmen. Durch Änderungen der Abfrage im Code konnten so ungewollte Zustandsänderungen verhindert werden, für das System sind diese Spitzen aber hinderlich. Die Antriebsräder benötigen ein besseres Profil um im nassen Gras weniger Schlupf zu erzeugen.

6.3. Ausblick

Das in dieser Arbeit entwickelte Softwarepaket besitzt bereits einen hohen Grad an Funktionalität, lässt aber viel Potential für weitere Arbeiten. Neben den schon genannten Lösungsvorschlägen im vorherigen Abschnitt gibt es noch eine Reihe von möglichen Arbeiten, die in dieses System passen.

Als mögliche Verbessungen ist hier eine kurze Liste verfasst:

- Nutzung von OpenCV zur Bilderkennung für Hindernisse im Gras
- Nutzung von X-Box Kinect Sensor für das Erstellen einer 3D- Karte zum Erkennen von Hindernissen(z.b. Unterführungen)
- Erweiterung des Sensornetzwerks für eine Rundumsicht (Infrarot und bumper)
- Entwicklung eines Planungsalgorithmus, der Hindernisse in der Karte mit einbezieht
- Überwachung der Energieversorgung und Entwurf einer Basisstation
- Einbeziehen der Coverage-Map in die Motorsteuerung des Mähmotors
- Webseite zur Statusüberwachung und Fernsteuerung
- App für mobile Endgeräte

Alle diese Themen können unter ROS gelöst werden.
Das Weiterführen dieser Arbeit, ist mit einem hohen Arbeitseinsatz und Zeitaufwand verbunden. Wie bereits erwähnt, ist das Robot Operating System sehr Komplex. Notwendige Skills zur Verbesserung des

Systems sind u.a das Beherschen von C++ und/oder Python, elektrotechnische Grundkenntnisse sowie Erfahrungen mit Mikrocontrollern.
Für ROS existiert auch eine Java- und Java-Script-API die allerdings nicht in dieser Arbeit benutzt wurden und über die daher keine Erfahrungswerte geliefert werden können.

Literaturverzeichnis

[Anderson u. a. 2002] ANDERSON, C. ; DOMINGOS, P. ; KOLLER, Daphne ; PFEFFER, Avi: Relational Markov models and their application to adaptive Web navigation. In: *Knowledge Discovery and Data Mining* (2002)

[Asada u. a. 2016] ASADA, Minoru ; BALCH, Tucker ; BONARINI, Andrea ; BREDENFELD, Ansgar ; GUTMANN, Steffen: Middle Size Robot League Rules and Regulations for 2017. In: *RoboCup* (2016), December

[Coleman u. a. 2013] COLEMAN, Dave ; TSOUROUKDISSIAN, Adolfo R. ; MAGYAR, Bence ; FERNANDEZ, Enrique: *ros control*. http://wiki.ros.org/ros_control. Version: 2013. – Eingesehen am 15.02.2017

[Elfers 2008] ELFERS, Carsten: *Aktionsvorhersage durch relationale Hidden Markov Modelle auf der Basis einer qualitativen raumzeitlichen Repräsentation*. Universität Bremen, March 2008. – Diplomarbeit

[Firedman u. a. 1999] FIREDMAN, Nir ; GETOOR, Lise ; WELD, D. ; SCHULZ, Dirk: Learning Probabilistic Relational Models. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (1999)

[Fox u. a. 1999] FOX, Dieter ; BURGARD, Wolfram ; THRUN, Sebastian: Markov Localization for Mobile Robots in Dynamic Environments. In: *J. Artif. Intell. Res. (JAIR)* 11 (1999), S. 391–427

[Grisett u. a. 2005] GRISETT, Giorgio ; STACHNISS, Cyrill ; BURGARD, Wolfram: Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In: *IEEE International conference on robotics and automation* (2005)

[Grzonka 2006] GRZONKA, Slawomir: *Untersuchungen zur Genauigkeit von SLAM-Verfahren mit Partikel-Filtern*. ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG FAKULTÄT FÜR ANGEWANDTE WISSENSCHAFTEN, 2006. – Diplomarbeit

[Gutmann 1999] GUTMANN, Jens S.: Robuste Navigation autonomer mobiler Systeme. (1999)

[Jäger 2014] JÄGER, Martin: *Hier mäht der Robo-Gärtner*. http://www.chip.de/artikel/Rasenmaehroboter-Test_71119704.html. Version: 2014. – Eingesehen am 20.10.2016

[Lübke 2014] LÜBKE, Michael: *Konstruktion und Aufbau eines autonomen, fehlertoleranten Rasenmäherroboters*. Department für Informatik Systemsoftware und verteilte Systeme, 2014. – Diplomarbeit

[Mürmann 2014] MÜRMANN, Michael: *Wahrscheinlichkeitstheorie und Stochastische Prozesse*. Springer Spektrum, 2014. – ISBN 978–3–642–38160–7

[Rehm u. Röhr 2016] REHM, Jenna A. ; RÖHR, Ulrike: Mähroboter und natürliche Gartenchemie- Produkte liegen im Trend. In: *GfK* (2016). – Pressemitteilung vom 02.09.2016

[Robomow] ROBOMOW: *mc800 Produktseite*. http://robomow.com/de-DE/shop/mowers-de-de/mowers/mc800/. – Eingesehen am 15.02.2017

[RobotElectronics] ROBOTELECTRONICS: SRF02 Ultrasonic range finder Technical Specification / Robot Electronics. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf. – Forschungsbericht

[Sanghai u. a. 2003] SANGHAI, Sumit ; DOMINGOS, Pedro ; WELD, Daniel: Dynamic Probabilistic Relational Models. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2003 (IJCAI'03), 992–997

[Sharp 2006] SHARP: GP2Y0A710K0F Distance Measuring Sensor Unit Measuring distance: 100 to 550 cm Analog output type / Sharp. 2006. – Forschungsbericht

[Sparkfun] SPARKFUN: *Razor-AHRS*. https://github.com/Razor-AHRS/razor-9dof-ahrs/wiki/Tutorial. – Eingesehen am 10.02.2017

[Thrun 2001] THRUN, S.: Is Robotics Going Statistics? The Field of Probabilistic Robotics. In: *Communications of the ACM* (2001), March

[Thrun u. a. 2005] THRUN, Sebastian ; BURGARD, Wolfram ; FOX, Dieter: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. – ISBN 0262201623

[Thrun u. a. 2001] THRUN, Sebastian ; FOX, Dieter ; BURGARD, Wolfram ; DELLAERT, Frank: Robust Monte Carlo localization for mobile robots. In: *Artificial Intelligence* 128 (2001), Nr. 1-2, 99–141. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.8488

[Wirtschaftslexikon 2016] WIRTSCHAFTSLEXIKON, Gabler: *Stichwort: Smart Home*. http://wirtschaftslexikon.gabler.de/Archiv/-2046533094/smart-home-v2.html. Version: 2016. – Eingesehen am 21.10.2016

[Woodman 2007] WOODMAN, Oliver J.: An introduction to inertial navigation / University of Cambridge, Computer Laboratory. Version: August 2007. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf. 2007 (UCAM-CL-TR-696). – Forschungsbericht

[Worx] WORX: *Worx Landroid Produktseite*. https://www.worxlandroid.com/de-DE/products/landroid-m-1000-wg791e.1. – Eingesehen am 15.02.2017

Abkürzungsverzeichnis

Anhang

A. Zusätzliche Dokumentation

A.1. Das ROS Grundsystem

Bei dem Raspberry Pi 3 wird das Betriebssystem auf eine μSD-Karte aufgesetzt und zur Auswahl standen 2 Betriebssysteme, unter denen auch ROS einsetzbar war. Das hauseigene Raspbian Jessie und Ubuntu Mate 16.04. Getestet wurden beide Betriebssysteme in Bezug auf die Kompatibilität mit ROS. Die Installation von ROS unter Raspbian war eine Herausforderung, was eine eventuelle Neuinstallation sehr erschweren würde ganz im Gegenteil zu Ubuntu. Dort war die Installation und Konfiguration des ROS-Grundsystems in einem Bruchteil der benötigten Zeit, wie unter Raspbian Jessie, abgeschlossen. Zur Gegenüberstellung der Benötigten Zeit, die gebraucht wurde zum einlesen, installieren und konfigurieren.

Raspbian Jessie	Ubuntu Mate 16.04
ca. 2 Tage	3h

Es ist also keine Überraschung, dass die Wahl des Betriebssystems auf Ubuntu viel.

A.1.1. SD-Karte vorbereiten und Ubuntu aufsetzen

Es wird empfohlen eine Klasse 10[1] μSD-Karte mit min 8GB Speicher zu nutzen und dieser Empfehlung wurde auch nachgekommen. Die benutzte μSD-Karte fasst 32GB und ist somit mehr als ausreichen für dieses Projekt. In Listing A.1 sind die Anweisungen[2] gelistet, die benötigt werden, um das Image von Ubuntu Mate(1.1GB) auf die SD-Karte zu schreiben. Um den Namen der SD-Karte herauszufinden muss einfach *lsblk* in die Konsole eingegeben werden. Der Name der SD-Karte sollte dann dort gelistet werden.

Listing A.1: Ubuntu Image auf die SD-Karte schreiben

```
sudo apt-get install gddrescue xz-utils
unxz ubuntu-mate-16.04-desktop-armhf-raspberry-pi.img.xz
sudo ddrescue -D --force ubuntu-mate-16.04-desktop-armhf-raspberry-pi.
    img /dev/sdx
```

Beim ersten Start des Raspberry Pi erscheint(falls am Monitor angeschlossen) ein Willkommensfenster. Dort kann die SD-Karte auf die volle Kapazität erweitert werden.
Am besten jetzt auch schon gleich die Netzwerkverbindung herstellen. Falls auf X11 verzichtet wurde (ohne GUI) kann ganz einfach über das Terminal eine automatische Verbindung mit einem beliebigen Netzwerk eingerichtet werden. Dazu folgende Anweisungen ausführen:

```
sudo apt-get install wireless-tools wpasupplicant
sudo nano /etc/network/interfaces
```

Dort dann folgende Zeilen einfügen:

```
allow-hotplug wlan0
auto wlan0
iface wlan0 inet dhcp
wpa-ssid Name des Netzwerks
wpa-psk Passwort
```

[1] https://de.wikipedia.org/wiki/SD-Karte
[2] Anleitung von der Ubuntu Internetseite https://ubuntu-mate.org/raspberry-pi/ Hier kann auch das Image geladen werden

Nun nach guter alter Windowsart einen Neustart machen.

```
sudo reboot
```

A.1.2. SSH einrichten

Um sich bequem über ssh auf den Roboter anmelden zu können, muss auf dem Raspberry eine Netzwerkverbindung eingerichtet sein. Ich gehe bei der folgenden Anweisung davon aus, dass das Betriebssystem des Remote-Clients ebenfalls eine Linux-Distribution ist.
Remote Client Setup: Als erstes vergeben wir dem Server einen Namen in der hosts-datei. Ist für die Nutzung von ssh zwar nicht Notwendig, wird aber im späteren Verlauf eine Rollen spielen. Dazu einfach folgende Zeile hinzufügen und speichern.

```
<IP> <Name>
192.168.0.1 router #als Beispiel
```

Öffnen der Datei

```
sudo nano /etc/hosts
```

Dann wird ein ssh-Schlüssel mit *ssh-keygen* erzeugt und eine Konfigurationsdatei mit *nano /.ssh/config* erstellt. Diese Datei muss folgenden Inhalt besitzen:

```
Host  router# Name des Servers
Hostname  router# IP oder wie im Beispiel aus der host-datei router
User admin# Username
PubKeyAuthentication yes
IdentityFile id_rsa # Pfad wo der Schluessel liegt
```

Dann noch den Schlüssel mit dem Server Teilen. Das geht mit

```
ssh-copy -i ~/.ssh/id_rsa hostname
```

Schließlich noch das Passwort zum angegebenen Usernamen eingeben und fortan kann sich mit der Eingabe von *ssh router* am Server angemeldet werden.
Schließlich noch X11 abschalten, da momentan kein Monitor auf dem Roboter angebracht ist. Dies spart auch Ressourcen, auf die wir angewiesen sind.

Listing A.2: X11 Abschalten

```
graphical disable
```

A.2. erweiterte Packetdokumentation

Dies ist eine zusätzliche Beschreibung der einzelnen Packet und stellt eine Übersicht zur Verfügung. Dies Übersicht stellt Beziehungen wischen Publisher und Subscriber dar und beinhaltet das jeweilige Message-Format eines Topics.

A.2.1. Three sd LIDAR

publish

Topic: laser_link, raw_laser_scan, filter_laser_scan
Massageformat: LaserScan[3]

Node-topic Graph beim Karte anlegen

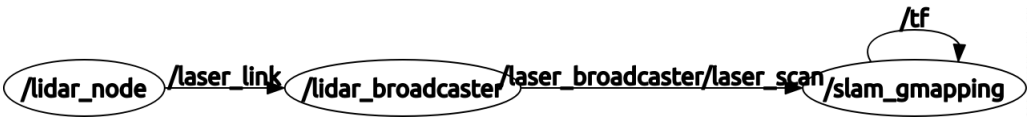


Abbildung A.1.: Lidar - Gmapping Nodegraph

A.2.2. Rosshaun Hardware Interface

Listing A.3: Konfiguration State Mux

```
topics:
  -
    name     : ps3
    states   :
        reset : true
        start : true
        mow   : true
        estop : true
    timeout : 0.5
    priority: 50
  -
    name     : cutter_controller
    states   :
        reset : false
        start : false
        mow   : true
        estop : false
    timeout : 1.5
    priority: 10
```

state_mux-Node

	Topic	Massageformat
publish	change_state	ChangeState
subscribe	hardware/reset	Empty[4]
	"name"/state	Bool[5]

[3]Message Definition: http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

Abbildung A.2.: State Multiplexer Node-Topic Graph

rosserial-Node Arduino Mega

	Topic	Massageformat
publish	srf_raw	SrfRaw
	ardu_status	Status
	stepper_feedback	StepperFeedback
subscribe	stepper_setpoint	StepperSetpoint
service	mower_state	MowerState

rosshaun_hw_iface-Node

	Topic	Massageformat
publish	bumper_range	Range[6]
	hardware/reset	Empty
	stepper_setpoint	StepperSetpoint
	srf02/right_middle_front	Range
	srf02/left_middle_front	Range
	srf02/right_front	Range
	srf02/left_front	Range
	mobil_base_controller/odom	Odometry[7]
subscribe	stepper_feedback	StepperFeedback
	srf_raw	SrfRaw
	change_state	ChangeState
	ardu_status	Status
	mobil_base_controller/cmd_vel	Twist[8]
service	mower_state	MowerState

Node-Topic Graph

A.2.3. rosshaun bringup

launcher Node

	Topic	Massageformat
subscribe	launch_state	Launch

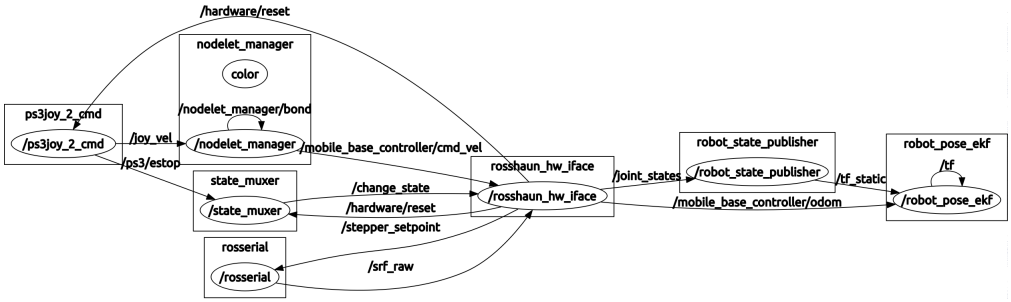


Abbildung A.3.: state_mux, rosshaun_hw_ifave und ArduinoMega als rosserial-Node mit Topics

A.2.4. Teleop

ps3_to_cmd

	Topic	Massageformat
publish	joy_vel	Twist
	ps3/reset	Bool
	ps3/start	Bool
	ps3/estop	Bool
	ps3/mow	Bool
	launch_state	Launch
	save_pose	Empty
subscribe	hardware_reset	Empty
	joy	Joy

Twist Mux

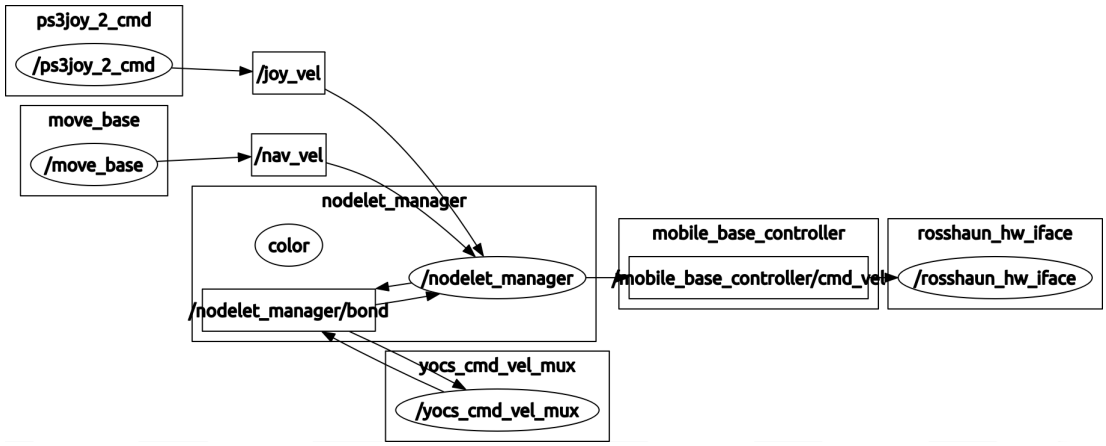


Abbildung A.4.: Node-Topic Graph vom Twist Multiplexer

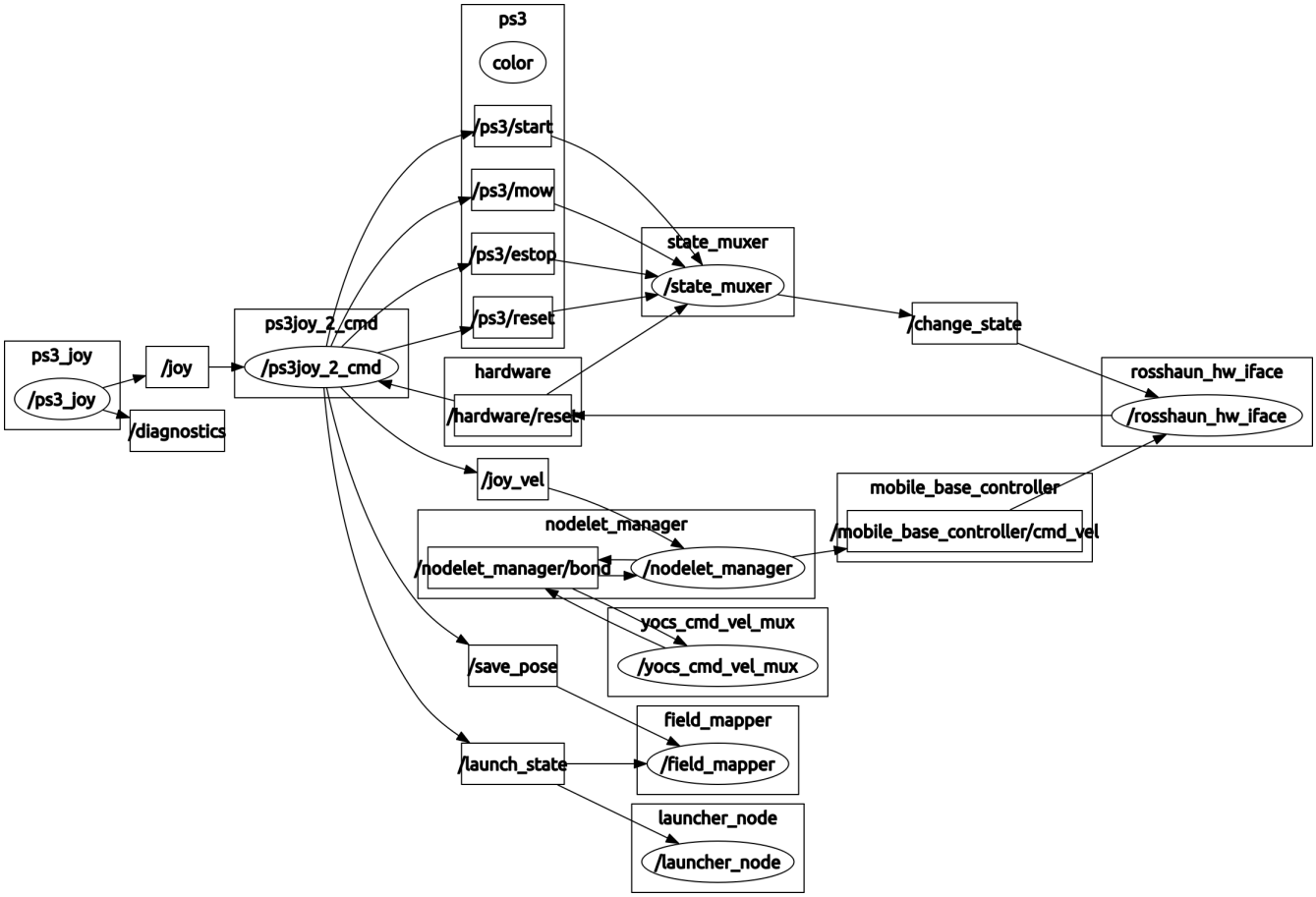


Abbildung A.5.: Node-Topic Graph des gesamten *rosshaun_teleop*-Pakets

B. Robot Operating System

Abbildung B.1.: Robot Operating System Logo

In diesem Anhang soll dem Leser ein kleines Handbuch zur Verfügung stellen, um einen ersten Eindruck über das Framework zu erhalten. Neben Begriffserläuterungen, die im Abschnitt B.1 aufgeführt sind, werden in diesem Anhang auch wesentliche ROS-Pakete vorgestellt, die in dieser Arbeit benutzt wurden. Dieser Anhang deckt bei weitem nicht das ganze Spektrum der Möglichkeiten ab, welche dieses Framework bietet. Was auch den Rahmen dieser Arbeit sprengen würde.

Als nächstes räume ich die Frage aus dem Weg, was das Robot Operation System eigentlich ist. Anders, als der Name vielleicht vermuten lässt, ist es kein Betriebssystem, sondern ein sehr komfortables Framework, dessen Gestalltungs-Ideologie auf die Entwicklung von wiederverwendbarem Code liegt.

Entstanden ist dieses Framework 2007 am Stanford Artificial Intelligence Laboratory und wurde seit 2008 von dem privat finanzierten Robotikforschungsinstitut Willow Garage weiterentwickelt und seit 2014 liegt die Verantwortung für ROS bei der Open Source Robotics Foundation.

ROS ist eine Sammlung von Werkzeugen, Bibliotheken und Entwicklungs- Designrichtlinien mit dem Ziel, welches das Erstellen komplexer und robuster Verhaltensmuster von Robotern erleichtern soll. In meinen Recherchen, bezüglich dieser Arbeit, war ich von der Menge an Möglichkeiten, die dieses Framework dem Anwender bietet, begeistert.

Die erste Anlaufstelle, um an mehr Informationen zu gelangen, ist die Wiki-Seite von ROS.org und bei Problemen steht dem Benutzer das sehr umfangreich und gut gepflegte Forum zur Verfügung. Eine Vielzahl

von Fehlern, die auftauchen können, haben andere auch schon gehabt. Zudem sind in den meisten Paketen zusätzliche Übungen und Beispiele enthalten, die den ersten Gebrauch erleichtern. Um jedoch ein komplett neues System, für den eigenen Roboter zu erstellen, haben mir die Einblicke in bereits entwickelte Systeme weitergeholfen.

B.1. Hintergrund

Dieser Abschnitt hat die Aufgabe auf ein Vokabular zu erstellen, dass in dieser Arbeit benutzt wird. Die Begriffe sind aus dem Englischen übernommen und beibehalten worden, aus dem einfachen Grund, dass eine Übersetzung mancher Begriffe zu fehlinterpretationen führen könnte. ROS besteht aus drei Stufen von Konzepten, der *Dateisystem*-, der *Berechnungs*- und der *Anwenderebene*. Die ersten zwei Ebenen werden nachfolgend erläutert und bilden somit das Vokabular.
Die **Dateisystemebene** umfasst auf der Festplatte befindliche ROS-Ressourcen.

- Der **Workspace**
  ist ein beliebiger Ordner im Homeverzeichnis der auf das Namenspräfix _ws endet. Folgende Kommandosequenz sollte hierzu im Terminal ausgeführt werden um den Arbeitsplatz zu erstellen und zu initialisieren.

  Listing B.1: listing

  ```
  1       mkdir -p ~/workspace_ws/src
          cd ~/workspace_ws/src
  3       catkin_init_workspace
  ```

  Ich empfehle zusätzlich, um den Workspace bei jeder Sitzung ROS zugänglich zu machen, dass folgende Zeile in die .bashrc-Datei an das Ende eingefügt wird

  ```
  1       echo 'source ~/wokspace_ws/devel/setup.bash' >> ~/.bashrc
  ```

- Die **Packages**
  sind die Schlüsseleinheit zum Organisieren von Software in ROS. Grundsätzlich sollte ein Package genügend Funktionalität besitzen, um nützlich zu sein jedoch nicht zuviel, um unüberschaubar zu werden. Eine Paketstruktur kann mit dem nachfolgenden Befehl

  ```
  1       catkin_create_pkg <Name> <Dependencies>
  ```

  in einem catkin[1] Workspace erstellt werden. Eine typische nach den ROS-Richtlinien ausgelegte Paketstruktur sieht je nach Inhalt wie folgt aus.

- **Metapackages**
  bestehen aus einer Sammlung von Packages.

[1] Catkin ist eine Sammlung von cmake macros und Pythoncode

- **Package Manifests**
  Die Manifestdatei, deren Name package.xml lautet beinhaltet sämtliche Metadaten des Paketes. Der Inhalt und Aufbau dieser Datei ist in der REP-0127[2] niedergeschrieben.

- **Messages**(msg) Typen
  beschreiben die Datenstruktur von Nachrichten, die über ROS zum Informationsaustausch genutzt werden und sind im Ordner /msg des jeweiligen Paketes zu speichern.

- **Services**(srv) Typen
  beschreiben eine Request- und Response-Datenstruktur für Services und werden im Ordner /srv gespeichert.

Die **Berechnungsebene** umfasst das Peer-to-Peer[3]-Netzwerk von ROS-Prozessen, die zusammen die Informationen verarbeiten. Zu diesen Prozessen gehören Nodes, Master, Parameter Server, messages, services topics und bags, die das Netz auf jeweils unterschiedlicher Weise mit Daten versorgen.

- **Master**
  Der Master-Prozess ist ein Unique[4]-Prozess, der mit dem Befehl *roscore* gestartet wird. Dieser Prozess dient anderen Prozessen als Einstiegspunkt in das ROS-System und ist für die Registrierung und der Namensvergabe dieser zuständig.

- **Nodes**
  Jede Node ist ein eigenständiger Prozess. Alle Nodes in einem System zusammen bilden einen Graphen und die Kommunikation zwischen diesen Nodes erfolgt über topics, services und dem Parameter Server.

- **Parameter Server**
  erlaubt das Zentrale speichern von Daten bezogen auf einen Schlüssel und macht diese Paare in einem Namensraum zugänglich und ist Teil des Master Prozesses.

- **messages**
  Nodes kommunizieren untereinander, durch das austauschen von messages

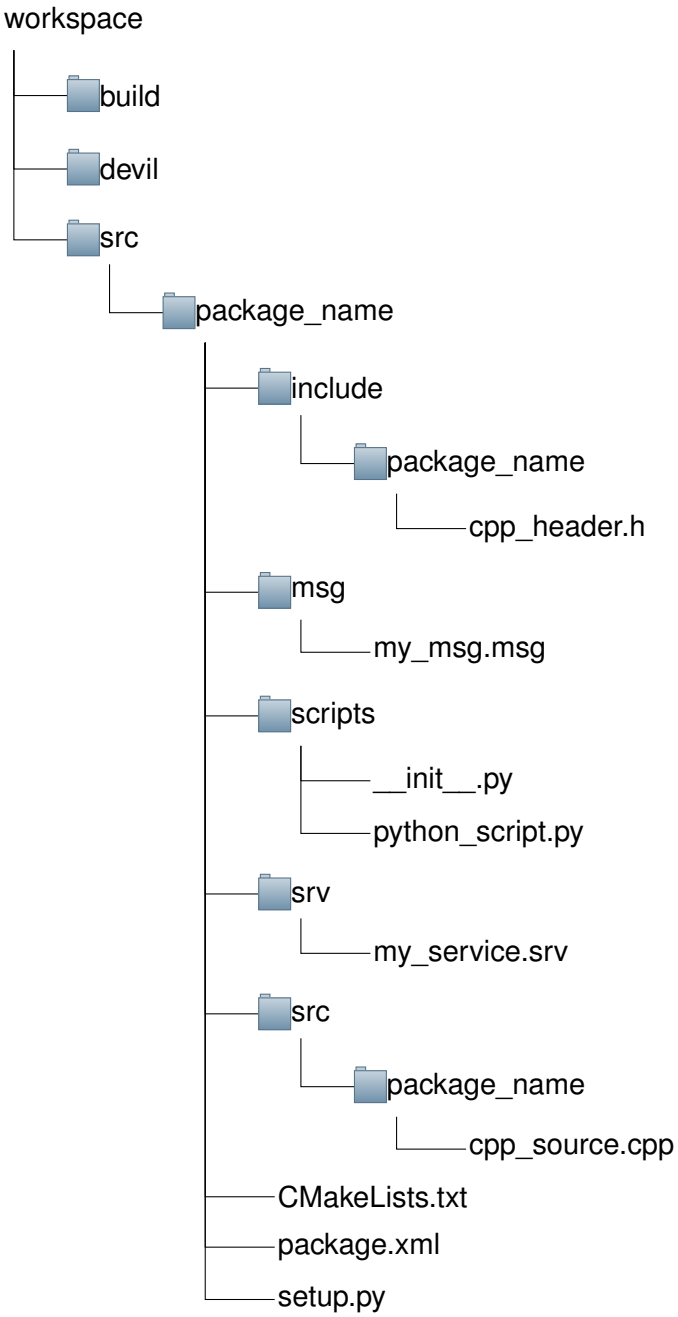
- **services**
  Für Request/Response- Interaktionen sind Services vorgesehen. Sie sind definiert durch ein Paar aus Nachrichten.

- **topics**
  Messages werden über das Verteilersystem mit publisher/subscriber-Semantik versendet. Das bedeutet, eine Node sendet eine Nachricht auf ein bestimmtes Topic und wiederum andere Nodes , welche sich für den Inhalt dieser Nachricht interessieren, können das Topic abonnieren(subscribe).

- **bags**
  ist ein Format um gesendete Nachrichten zu speichern und später auch wieder abzuspielen.

[2] zu finden unter: http://www.ros.org/reps/rep-0127.html
[3] Kommunikation unter Gleichen
[4] einzigartig

Ein Workspace könnte ungefähr wie die nachfolgende Struktur aussehen.



B.2. Werkzeuge

B.2.1. Command-line Tool

Das wichtigste Werkzeug, im Umgang mit ROS ist die Kommandozeile. In Tabelle B.1 zeigt eine Auswahl der wichtigsten Kommandozeilenbefehle, bezüglich dieser Arbeit. Eine deutlich umfangreichere Auflistung ist im Wiki von Ros.org zu finden.

Dateisystem	Bedeutung	Benutzung
roscd	Paketverzeichnis wechseln	`$ roscd [package[/subdir]]`
roswtf	Anzeigen von Fehler	`$ roswtf or roswtf [file]`
catkin_create_pkg	Paket erstellen	`$ catkin_create_pkg [package_name] [depend1]..[dependN]`
catkin_make	kompiliert den Workspace	`$ catkin_make`
rqt_dep	zeigt Paketstruktur und Abhängigkeiten	`$ rqt_dep [options]`
Start & Launch		
roscore	Kernprozess starten	`$ roscore`
rosrun	starten einer Node	`$ rosrun package_name executable_name`
roslaunch	startet Kernprozess (wenn nötig), lokale oder remote Nodes und setzt Parameter	`$ roslaunch package_name launch_name.launch`
Laufzeit		
rostopic	anzeigen von Topic-Informationen	`$ rostopic list, $ rostopic echo /topic_name`
rosnode	anzeigen von Node-Informationen	`$ rosnode list, $ rosnode info node_name`
Anzeige, Graphiken		
rqt	starten der rqt Oberfäche	`$ rqt`
view_frames	erzeugt eine Grafik der aktuellen Frames im System	`$ view_frames`

Tabelle B.1.: Auswahl der am meist benutzten Befehle

B.3. Navigation Stack

Der Navigation Stack ist das wichtigste Metapaket, wenn es um die automatisierte Bewegung eines Roboter geht. Die Funktion dieses zusammengefassten Pakete besteht darin,

- die Lokalisierung in einer vorhandenen Karte durchzuführen
- eine lokale und globale Costmap zu erstellen
- Wege zu Zielkoordinaten zu planen
- Steuerbefehle senden, um den erstellten Plan abzuarbeiten

Der Reihe nach werden nun die Hauptpakete erläutert.

B.3.1. move base

Um den Navigation Stack zu starten, konfigurieren mit mit ihm zu interagieren stellt Move Base ein Interface zur Verfügung. Move Base besteht im wesentlichen aus den 5 Teilen *global_planer, local_planner, global_costmap, local_costmap und recovery_behaviors*, wie in Abbildung B.2 zu sehen ist.

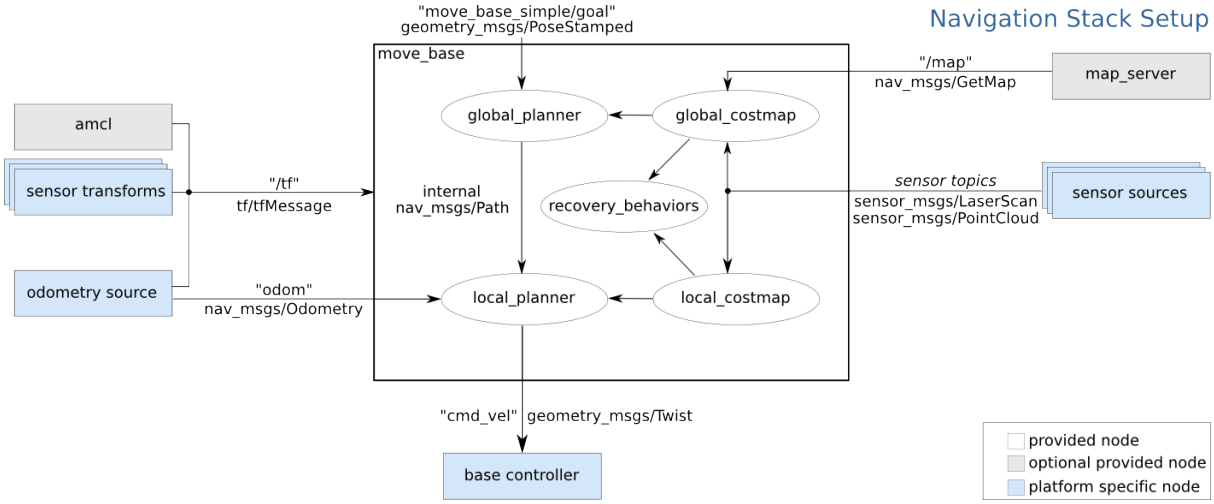


Abbildung B.2.: Move Base Übersicht, adaptiert von ros.org

Die **Global Coastmap** ist eine statische Karte und wird anhand der Hindernisse in geladenen Karte und den eingehenden Sensordaten berechnet und dient dem **Global Planner** als Grundlade zum planen von Wegen zu einem gegebenen Ziel. Hat der Global Planner eine Route gefunden, so wird diese an den **Local Planner** weitergeleitet. Dieser benötigt zusätzlich noch die dynamische **Local Coastmap**, um den Steuerbefehle zu berechnen, die dynamische Hindernisse mit einbeziehen. Durch rückführende Odometriedaten werden in die weitere Erzeugung von Steuerbefehlen mit einbezogen. Sollte der Roboter einmal nicht weiter kommen und keinen Pfad von seiner aktuellen Position hinaus finden, dann wird **Recovery Behaviors** zurückgegriffen und versucht durch Sequenzen von Fahrmanöver die Costmaps zu aktualisieren, um so einen Weg hinaus zu finden. Abbildung B.3 soll den Ablauf dieser Sequenzen verdeutlicht.

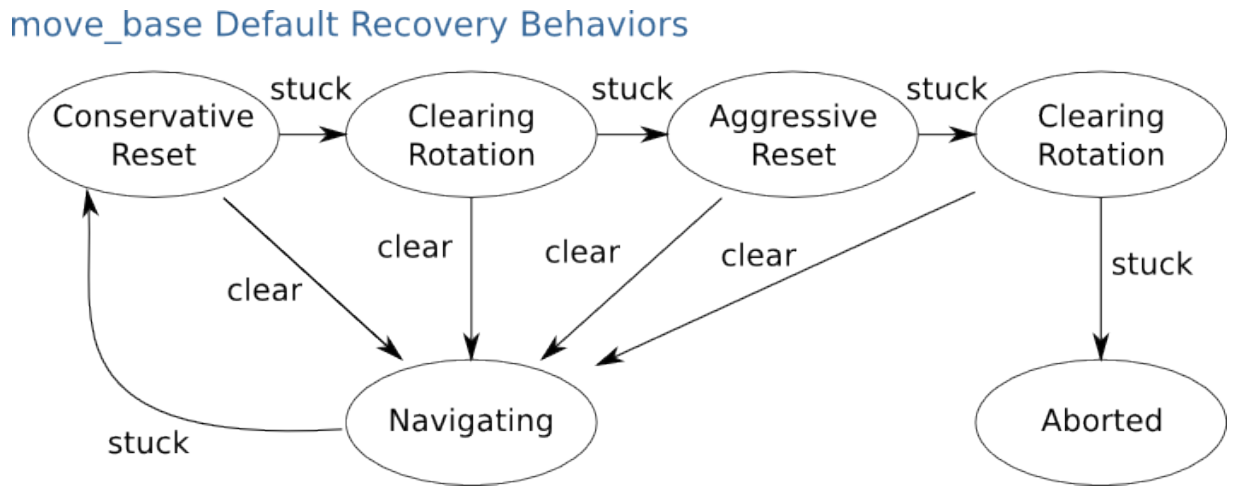


Abbildung B.3.: Move Base Wiederherstellungsverhalten bei einem Feststecken, adaptiert von ros.org

B.3.2. Robot Pose EKF

Dieses Paket ist zwar nicht zwingend für die Navigation erforderlich, da es aber von Bedeutung für diese Arbeit ist und es Teil des Navigation Stack Metapackage ist,wird dessen Aufgabe an dieser Stelle erläutert. Dieses Paket schätzt anhand von verschiedenen Sensordaten die Position des Roboters. Als Sensordaten werden standardmäßig Odometrie, IMU und Visual-Odometrie akzeptiert. Es besteht aber die Möglichkeit, dass andere Sensordaten mit eingebunden werden können. Aus dem vorherigen Kapitel im Abschnitt 3.5 hatte wir uns schon einmal mit der Odometrie und der Ungenauigkeit der Messdaten beschäftigt. Um dieser Messungenauigkeit entgegenzuwirken wird durch die Verwendung eines Erweiterten Kalman Filters auf einem 6D Model (3d Position und 3d Orientierung) benutzt. Es ist das Ziel, die Messfehler des einen Sensors durch den anderen auszugleichen. Was dem Paket lediglich noch fehlt, ist das mit Einbeziehen der Beschleunigungsdaten von der IMU.

B.4. Transformation tf

Ohne Transformationen würde nichts gehen. In einem auf Koordinaten aufbauenden Umgebung ist die effizienteste Methode die Matrixtransformation und Vektorberechnung. Da jedes Glied in einem Roboter, welches frei beweglich ist, seine Position relativ zu anderen Teilen und zur Welt verändert, würden viele dieser Berechnungen im Code stehen. Das TF Packet, übernimmt einen Großteil dieser Rechnungen.

B.5. ROS Control

Dieses Packet soll eine Abstraktionsebene zur Roboterhardware schaffen und hat die Aufgabe, mit dieser zu kommunizieren. Mit Hilfe der Daten, die von der Hardware gelesen wurden und dem Erhalt eines Ziels(Position, Geschwindigkeit u.a) soll der eingesetzte Controller Befehle generieren und an die Hardware senden. In Abbildung B.4 ist die Arbeitsweise eines Cotrollers zu sehen. Der Controller Manager übernimmt dabei die Verwaltet aller Controller und macht diese leichter austauschbar.
Es stehen eine Reihe von Controllern zur Verfügung. Der *diff_drive_controller* ist einer von ihnen und

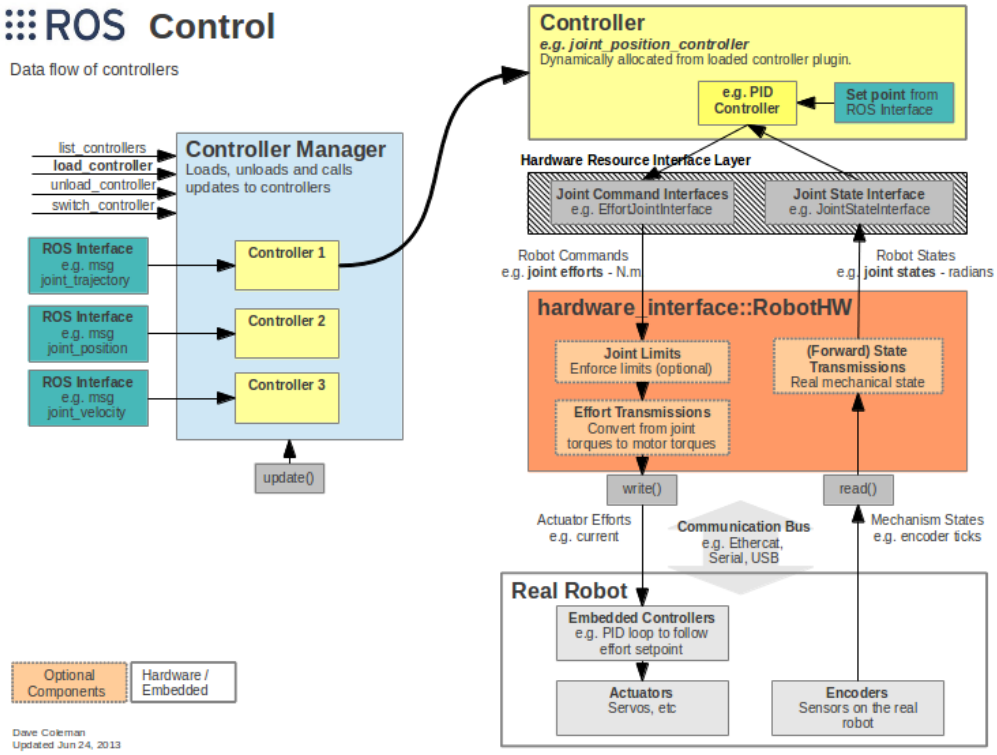


Abbildung B.4.: Ros Control Datenfluss-Diagramm, adaptiert von Coleman u. a. [2013]

wurde für den Einsatz in Robotern mit Differenzialantrieb konzipiert. Der Controller übernimmt dabei das Ansteuern der beiden Antriebsräder. Um die dafür nötigen Befehle zu berechnen benötigt der Controller als Eingabe eine *Twist*-Message, die Informationen von den Encodern und die Beschreibung des Antriebs. Zu dieser Beschreibung gehören der Achsenabstand, der Raddurchmesser, die Grenzen und die Beschleunigungswerte. Darüber hinaus bietet der Controller die Möglichkeit, aus den ohnehin schon gewonnenen Informationen der Hartware, das generieren der Odometriedaten.

B.6. Gmapping

Dieses Paket bietet die Nutzung des SLAM-Algorithmus gmapping von OpenSlam[5] an. Auf der Grundlage der Entfernungsdaten eines Laserscanners und den Odometriedaten erstellt dieser Algorithmus eine 2D-Umgebungskarte. Diese Karte hat eine Gitterstruktur wovon jede Zelle einen Belegungszustand vorweist. Gmapping beruht dabei auf den Rao-Blackwellized Partikelfilter[6].

[5]Link http://openslam.org/gmapping.html
[6]Link zum Paper http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf

B.7. Auburn Automow

Bei meinen Recherchen bin ich auf ein Projekt der Universität Auburn, Alabama in den Vereinigten Staaten gestoßen. Mehrere Teammitglieder[7] haben dort 2009 einen Mähroboter entworfen. Dabei kann dieser Roboter auf ein breit ausgelegte Sensorlandschaft zurückgreifen, um autonom Arbeiten zu können. Interessant für meine Arbeit war, wie die Planungslösung für diesen Roboter implementiert wurde. Für eine Nutzung in meiner Arbeit mussten Änderungen in der Packetstruktur und der Implementierung vorgenommen werden. Ein Grund für diese Änderungen bestand darin, dass die Erstellung dieser Pakete für eine ältere ROS-Version gedacht war. Unterschiede zur aktuellen ROS-Version bestanden in dem benutzten Build-System, in der Packetstruktur und in der Packetbeschreibung. Ein weiterer Grund war, dass die interessanten Packet nicht generisch waren und viele Abhägigkeiten zu anderen Paketen besaßen.
Nachfolgend werden die genutzten Packete aus dem *au_automow_common*-Paket genannt und der Aufgabe erklärt.

B.7.1. Automow Maps

Diese Paket umfasst die *field_publisher*-Node, die anhand von Koordinaten 3 Ploygone berechnet. Dieses Koordinaten wüssen im Parameterserver geladen sein. Die 3 Polygone stehen jeweils für ein bestimmtes Gebiet (safety, boundry und cutarea) und der Abstand zu den Gebieten kann ebenfalls über Parameter geändert werden.

B.7.2. Automow Planning

Dieses Paket beinhaltet 3 Nodes. Die *cutter_control*-Node, die anhand der Cutarea den Mähmotor an bzw. abschaltet. Eine weitere ist die *cutting_coverage*-Node, die eine Karte erstellt, in der zu sehen ist wo bereits gemäht wurde. Diese Karte wird erstellt, indem die Koordinaten gezeichnet werden, bei denen der Mähmotor an war. Die letzte ist die *path_planner*-Node, die schließlich den Lienienfahrt durch die Cutarea plant.

[7] Meschienenbau Studenten: Billy Bivins, Casey Still, Chris Warren, Chris Holland, Chris Khamken, Zach Lamb, Terry Miller, Corey Farmer und Jeremy Garner
Elektro und Computer Studenten: Michael Carroll, Chris Davis, Stephen Haddock, Stephan Henning, Sean Hyde, Mike Payne, Caleb Perry
Softwaretechnik Studenten: John Harrison und William Woodall

C. Grafiken

C.1. externe Grafiken

C.1.1. Mähroboter 2014 Vergleich

Mähroboter

	VIKING MI 632	ROBOMOW RC306	WORX WG794E	BOSCH INDEGO
Rang	1	2	3	4
Preis (ca.)	2.350 €	1.300 €	950 €	1.200 €
Gesamtwertung	100	86	75	67
Leistung	100	98	57	37
Ergonomie	100	64	82	79
Ausstattung	100	95	87	88
TECHNISCHE DATEN				
Abmessungen (LxBxH)	73 x 50 x 27 cm	60 x 40 x 26 cm	54 x 37 x 26 cm	70 x 50 x28 cm
Gewicht	12 kg	10,5 kg	8,5 kg	11 kg
Mähprinzip	Zufällig	Zufällig	Zufällig	Planmäßig
Maximale Mähfläche	3.000 m²	600 m²	1.000 m²	1.000 m²
Schnittbreite/Höhe min/max	30 cm/2 cm/6 cm	28 cm/1,5 cm/6 cm	18 cm/2 cm/6 cm	26 cm/2 cm/6 cm
Randabstand	27 cm	28 cm	35 cm	35 cm
Steigung	35°	20°	20°	20°
Mähzeiten pro Tag/Arbeitstage	3/7	1/7	1/7	2/7
Regensensor/Handbetrieb	■/■	■/□	■/□	■/□
Messer/Wechsel	Balkenmesser/ohne Werkzeug	Sternmesser/Spezialwerkzeug	Klingen/Schraubendreher	Klingen/Schraubendreher
Diebstahlschutz/Passwort	■/■	■/■	■/■	■/■
LABORMESSUNGEN				
Mittl. Geschwindigkeit	0,3 m/s	0,5 m/s	0,3 m/s	0,5 m/s
Mähzeit/Akkuladung	207 min	85 min	81 min	65 min
Mähleistung	303 m2/h	299 m2/h	172 m2/h	114 m2/h
Leistungsaufnahme Laden/Mähen/Bereitschaft	52 W/4,5 W/ 1,5 W	53 W/9 W/ 2 W	52 W/3 W/ 3 W	52 W/ 12,5 W/ 1 W
Ladedauer	216 min	65 min	68 min	66 min
Schallpegel/Eindruck	70 dB/flüsternd	77 dB/hörbar	72 dB/flüsternd	82 dB/brummend

SPITZENKLASSE (100–90,0) OBERKLASSE (89,9–75,0) MITTELKLASSE (74,9–45,0) NICHT EMPFEHLENSWERT (44,9–0)
ALLE WERTUNGEN IN PUNKTEN (MAX. 100) | ■ JA □ NEIN

Abbildung C.1.: Tabelle Mähroboter 2014 von Jäger [2014]

C.2. Transform

C.2.1. TF-Tree während der Kartierung

C.2.2. TF-Tree während im Mähmodus

C.3. Übersicht Node-Topic Graph

Recorded at time: 1487413533.0
map
Broadcaster: /slam_gmapping
Average rate: 20.1
Buffer length: 10.0
Most recent transform: 1487413532.99
Oldest transform: 1487413522.99
odom_combined
Broadcaster: /robot_pose_ekf
Average rate: 26.353
Buffer length: 9.98
Most recent transform: 1487413532.89
Oldest transform: 1487413522.91
base_footprint
Broadcaster: /base_footprint_2_base_link
Average rate: 10.076
Buffer length: 9.924
Most recent transform: 1487413533.02
Oldest transform: 1487413523.1
dummy_link
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.43
Oldest transform: 1487413523.45
base_link
Broadcaster: /robot_state_publisher
Average rate: 31.426
Buffer length: 9.96
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 31.426
Buffer length: 9.96
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 31.426
Buffer length: 9.96
Most recent transform: 1487413532.89
Oldest transform: 1487413522.93
right_wheel_link
base_imu_link
left_wheel_link
laser_link
Broadcaster: /laser_link_frame_2_laser_fake_link
Average rate: 10.079
Buffer length: 9.922
Most recent transform: 1487413533.02
Oldest transform: 1487413523.1
laser_fake_link
frontbar_link
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
Broadcaster: /robot_state_publisher
Average rate: 50.099
Buffer length: 9.98
Most recent transform: 1487413533.45
Oldest transform: 1487413523.45
right_front_srf02_link
left_middle_front_srf02_link
front_bumper_link
left_front_srf02_link
right_middle_front_srf02_link

Recorded at time: 1487413688.65
map
Broadcaster: /amcl
Average rate: 0.377
Buffer length: 7.955
Most recent transform: 1487413689.03
Oldest transform: 1487413681.08
odom_combined
Broadcaster: /robot_pose_ekf
Average rate: 26.001
Buffer length: 10.0
Most recent transform: 1487413688.55
Oldest transform: 1487413678.55
base_footprint
Broadcaster: /base_footprint_2_base_link
Average rate: 10.07
Buffer length: 9.93
Most recent transform: 1487413688.4
Oldest transform: 1487413678.71
dummy_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413689.11
Oldest transform: 1487413679.11
base_link
Broadcaster: /base_link_2_cutter
Average rate: 10.075
Buffer length: 9.925
Most recent transform: 1487413688.62
Oldest transform: 1487413678.7
cutter
Broadcaster: /robot_state_publisher
Average rate: 31.001
Buffer length: 10.0
Most recent transform: 1487413688.61
Oldest transform: 1487413678.61
right_wheel_link
Broadcaster: /robot_state_publisher
Average rate: 31.001
Buffer length: 10.0
Most recent transform: 1487413688.61
Oldest transform: 1487413678.61
left_wheel_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.61
Oldest transform: 1487413679.11
base_imu_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.61
Oldest transform: 1487413679.11
laser_link
Broadcaster: /laser_link_frame_2_laser_fake_link
Average rate: 10.072
Buffer length: 9.929
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
laser_fake_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
left_middle_front_srf02_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
front_bumper_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
frontbar_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
right_front_srf02_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
right_middle_front_srf02_link
Broadcaster: /robot_state_publisher
Average rate: 50.103
Buffer length: 9.999
Most recent transform: 1487413688.67
Oldest transform: 1487413679.11
left_front_srf02_link

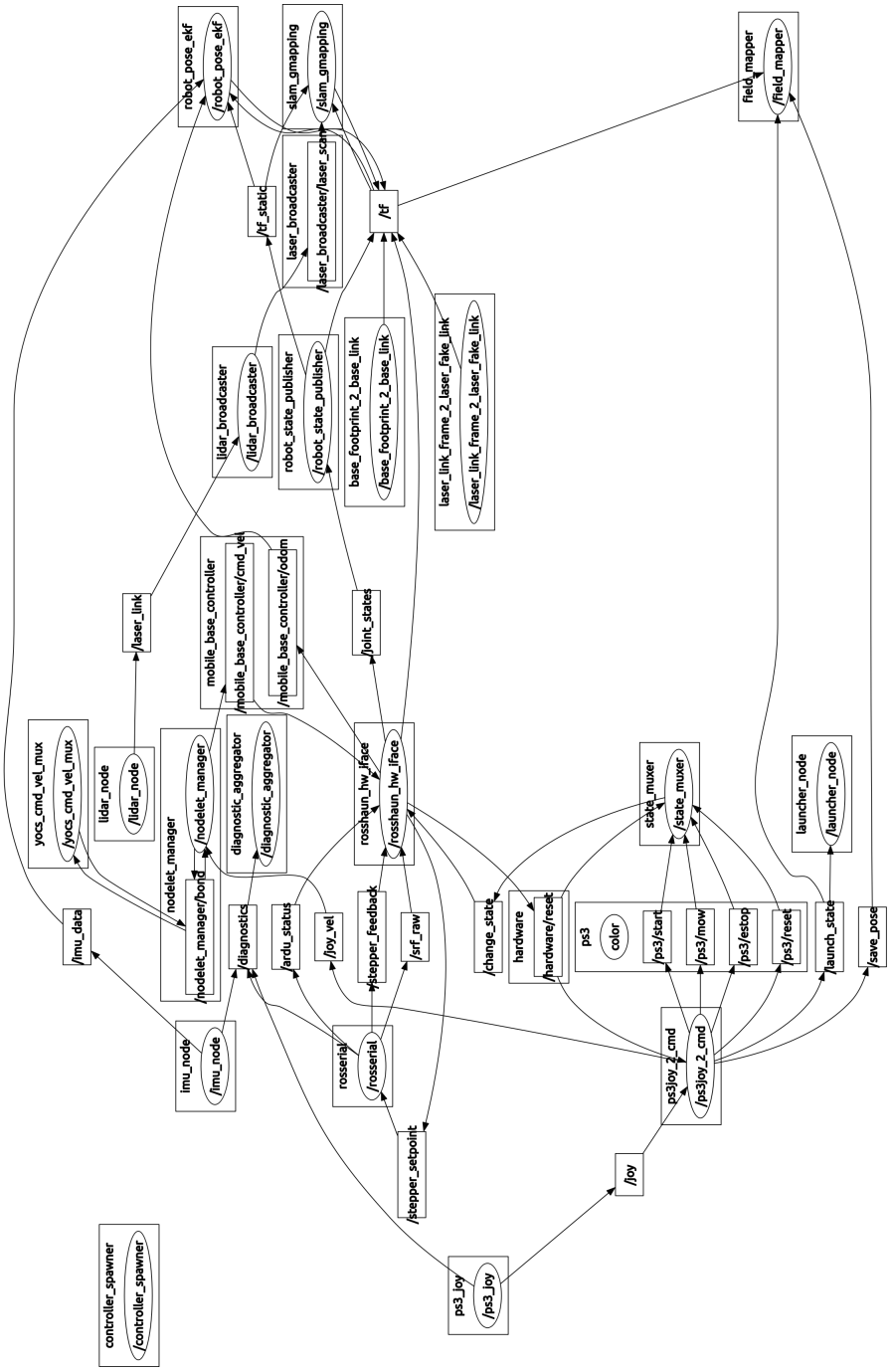


Abbildung C.4.: Node-Topic Graph während der Kartierung

D. CD