OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

BACHELORARBEIT

Andreas Hilmer

Indoor 2D-SLAM Basierte Pfadplanung auf einem Mobilen Roboter

Indoor 2D-SLAM Based Path Planning on a Mobile Robot

Fakultät:	Informatik und Mathematik
Studiengang:	Technische Informatik
Abgabefrist:	31.08.2021
Betreuer/Prüfer:	Prof. Dr. Alexander Metzner

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.

2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

_____

Ort, Datum und Unterschrift

Student:	Andreas Hilmer
Matrikelnummer:	3146934
Bearbeitungsdauer:	01.04.2021 – 31.08.2021
Zweitprüfer:	Prof. Dr. Daniel Münch

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AMCL	Adaptive Monte Carlo Localization
EKF	Extended Kalman Filter
PWM	Pulse Width Modulation
ROS	Robot Operating System
Rviz	ROS Visualization Tool
SLAM	Simultaneous Localization and Mapping
TOF	Time of Flight
USART	Universal Synchronous and Asynchronous Receiver-Aransmitter
URDF	Unified Robot Description Format
XML	Extensible Markup Language (Dateiformat)
YAML	Yet Another Markup Language (Dateiformat)

1 Einleitung

Roboter werden in vielen Industriebereichen schon seit langem standardmäßig zur Automatisierung von Herstellungsprozessen eingesetzt. Auch für die Luft- und Raumfahrt ist der Einsatz von mobilen Robotern essenziell. Bei den eingesetzten Robotern wird zwischen zwei Arten unterschieden. Zum einen Roboter mit einer festen Basis wie Schweißroboter, Montageroboter und Lackierroboter. Zum anderen gibt es mobile Roboter, welche sich frei in ihrer Umgebung Bewegen können. Mobile Roboter können je nach Art auf festem Untergrund, in Wasser oder in der Luft operieren (1 S. xiii). Sie werden u.a. als Sortierroboter und Kommissionierungsroboter in der Logistik (2) oder wie der *Mars Curiosity Rover* als Erkundungsroboter in der Raumfahrt eingesetzt. (3). Auch im militärischen Bereich werden heute schon mobile Roboter, z.B. in Form von Drohnen eingesetzt. Insbesondere dort ist es notwendig auch ethische Aspekte miteinzubeziehen. Es ist denkbar, dass in Zukunft mobile Roboter auch vermehrt als Such- und Rettungsroboter, z.B. nach Naturkatastrophen oder Bränden, eingesetzt werden. Nachdem in den letzten Jahren die Rechenleistung stark angestiegen und gleichzeitig der Preis von Hardware und Sensoren zur Umgebungserfassung gefallen ist, hat sich das Einsatzgebiet von mobilen Robotern in den letzten Jahren auch auf den persönlichen und häuslichen Bereich ausgeweitet. So sind z.B. Staubsauger-Roboter, Rasenmäher-Roboter und Quadrocopter schon für wenige hundert Euro erhältlich und so konnten im Jahr 2019 schon mehr als 23,2 Millionen solcher Geräte verkauft werden (4). Es ist zudem davon auszugehen, dass sich der Trend hinzu Autos und Lastkraftwagen, die sich Teilautonom oder sogar Vollautonom fortbewegen, fortsetzen wird. Die Bereiche in denen mobile Roboter eingesetzt werden sind vielfältig und es kommen laufend neue Einsatzgebiete dazu. So werden auch in Zukunft mobile Roboter immer mehr an Bedeutung gewinnen.

1.1 Problemstellung

Eine der größten Herausforderungen von mobilen Robotersystemen und autonomen Fahrzeugen ist es, sich zu orientieren. Sich zu orientieren bedeutet zu wissen wie die Umgebung aussieht, wo man sich gerade befindet und wo man schon überall war. Mithilfe dieser Informationen ist es Robotern möglich sich gezielt in einer unbekannten Umgebung zu bewegen, Hindernisse früh genug zu erkennen und damit Kollisionen zu vermeiden. Zur Erfassung der benötigten Informationen, um Aussagen über die räumliche Lage eines Roboters zu treffen, werden sowohl interne, als auch externe Sensoren verwendet. Interne Sensoren geben dem Roboter Auskunft über seinen

eigenen Zustand, wie die Motordrehzahl oder dem Ladezustand der Energieversorgung. Mit externen Sensoren hingegen erfasst ein Roboter Informationen, die Auskunft über seine Umgebung geben. Dazu zählen u.a. GPS-, Lidar-, Ultraschallsensoren oder Kameras. Bei der Auswahl von Sensoren für die jeweiligen Roboter wird nach Einsatzfeldern unterschieden. So kommen zur Lokalisierung außerhalb von geschlossenen Räumen oft GPS-Sensoren zum Einsatz. Innerhalb von Gebäuden und/oder wenn eine akkuratere Positionsbestimmung - insbesondere relativ zu Objekten in der näheren Umgebung - notwendig ist, kommen häufig Lidar-Sensoren zum Einsatz. Methoden zur Lösung des Problems der Orientierung von mobilen Robotern in unbekannten Umgebungen, d.h. es liegen keine Karte und keine Positionsdaten vor, werden als Simultaneous Localization and Mapping (SLAM) Verfahren bezeichnet (5).
Diese Bachelorarbeit beschäftigt sich ausführlich mit der 2D Kartographierung, Positionsbestimmung und Navigation eines mobilen Roboters in Gebäuden, wie es beispielsweise bei Staubsaugerrobotern oder Lagerrobotern der Fall ist.

1.2 Zielsetzung

In dieser Bachelorarbeit wird das Ziel verfolgt, von Grund auf einen mobilen Roboter zu entwerfen, der sich in einer unbekannten Umgebung (indoor) orientieren und navigieren kann. Hierzu soll dieser gleichzeitig eine 2D-Kartographierung und Lokalisierung durchführen. Dabei soll sich der Roboter autonom in der Umgebung fortbewegen und alle zugänglichen Bereiche kartographieren. Es soll dem Benutzer möglich sein den Roboter über eine visuelle Karte mit Positionsdaten des Roboters - welche über SLAM-Verfahren erstellt wurde und kontinuierlich aktualisiert wird - zu einem gewünschten Punkt auf der Karte zu navigieren. Der Roboter soll dann den kürzesten Weg zum Zielpunkt finden. Um das zu erreichen werden zunächst jene Grundlagen der mobilen Robotik herausgearbeitet, die zur Realisierung eines solchen Roboters nötig sind. Dazu gehört der Bereich der Kinematik, Umgebungserfassung und SLAM. Des Weiteren wird eine Analyse von geeigneten Sensoren durchgeführt, welche in Kombination in der Lage sein sollen, die benötigten Odometrie- und Entfernungsdaten für die eingesetzten SLAM- und Pfadfindungsalgorithmen zu liefern. Die Software des Robotersystem wird auf Basis des Robot Operating Systems (ROS) (6) erstellt. Die Auswahl von geeigneten Sensoren und Algorithmen im Hinblick auf vorhandene Standardpakete in ROS getroffen. Zunächst wird mit Hilfe von Gazebo (7) eine Simulation des Roboters erstellt. Dabei werden auch die Pakete zur Ausführung der Mapping -und Navigationsaufgaben ausgewählt. Mithilfe von Gazebo und den geeigneten ROS Paketen werden dann Karten in einer virtuellen Testumgebung erstellt und im ROS Visualization Tool (Rviz) (8) dargestellt. Auf den erfolgreich erstellten Karten wird nun die Navigation des Roboters virtuell getestet. Nachdem der Roboter in der Simulation die gewünschten Ergebnisse liefert, steht fest, welche

Sensoren im Roboter verbaut werden müssen. Nun kann das Fahrzeuggestell des Roboters mit dem CAD Programm Autodesk Inventor designed, die benötigten Teile anschließend mit einem 3D-Drucker ausgedruckt und der Roboter montiert werden. Es folgt die Implementierung der Treiberschichten für die verwendeten Sensoren. Um dem ROS Ökosystem nun die Kommunikation mit dem Mikrocontroller zu ermöglichen, wird ein Hardware Interface implementiert. Es werden dann alle in der Simulation durchgeführten Tests, bezüglich Navigation und Kartenerstellung, auch auf dem realen Roboter durchgeführt. Im letzten Schritt werden die Mapping und Navigationsdaten ausgewertet und mit am Markt vorhandenen Robotersystemen verglichen, welche die gleichen Funktionen unterstützen.

2 Grundlagen Mobile Robotik

Im folgenden Kapitel möchte ich auf die Grundlagen der mobilen Robotik eingehen, welche besonders wichtig bei Design, Analyse, Steuerung und Programmierung des von mir beabsichtigten Roboters sind.

2.1 Kinematik

Das Themenfeld der Kinematik in der mobilen Robotik behandelt die Bewegung von Körpern in einem Robotersystem ohne Berücksichtigung der Kräfte, welche die Bewegung verursacht haben. Sie gehört damit zu den grundlegendsten Bestandteilen bei der Planung eines Robotersystems. (9 S. 11)

2.1.1 Pose und Bewegungsbeschreibung

Pose

In der Robotik gibt es verschiedene Arten die Position und Orientierung eines Körpers darzustellen. Da sich der in dieser Arbeit behandelte Roboter nur auf einer planen Ebene bewegt, wird sich darauf beschränkt seine Pose (siehe *Abbildung 1*) nur mit drei Variablen darzustellen. So kann die Pose eines solchen Roboters durch folgenden Vektor beschrieben werden:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

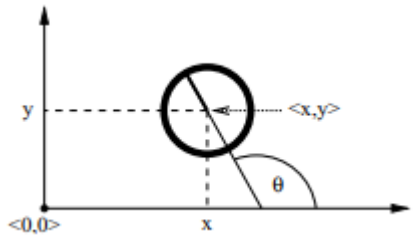

Abbildung 1. Roboterpose in einem globalen Bezugssystem

Quelle: (5 S. 118-119)

Hierbei beschreiben x, y die Position und θ die Orientierung des Roboters relativ zu einem externen Bezugssystem.

Bewegungsbeschreibung

Die Bewegung eines Roboters der sich auf einer planen Fläche bewegt kann durch zwei Geschwindigkeiten definiert werden. Zum einen die Translationsgeschwindigkeit v und zum anderen die Rotationsgeschwindigkeit ω. Es ergibt sich: $u = \begin{pmatrix} v \\ \omega \end{pmatrix}$

Viele mobile Roboter bieten Schnittstellen, welche Kommandos der Form u akzeptieren und den Roboter entsprechend steuern. Mobile Roboter mit Ackermann Lenkung und solche mit Differentialantrieb gehören zu den Fahrzeugen die häufig auf diese Weise gesteuert werden. (5 S. 121) Da ich mich bei meinem Roboter aus Gründen der Wendigkeit für einen Differentialantrieb entschieden habe, werde ich im Folgenden Abschnitt diese Antriebsart genauer betrachten.

2.1.2 Differentialantrieb

Bei mobilen Robotern mit Differentialantrieb werden zwei Räder die auf der gleichen Achse liegen unabhängig voneinander angesteuert. Dies ermöglicht dem Roboter sich auf einer Stelle im Kreis zu drehen und macht ihn daher besonders wendig.

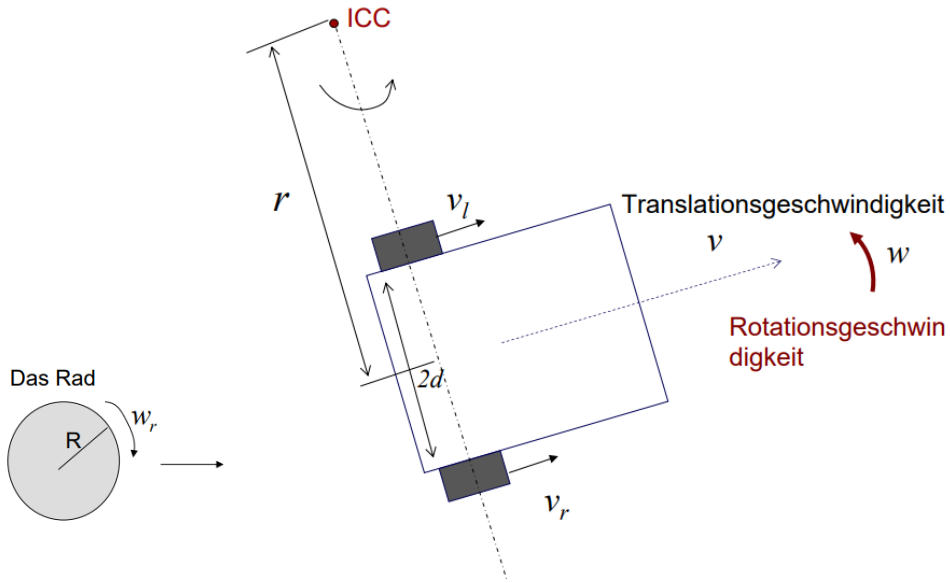

Abbildung 2: Mobiler Roboter die Differentialantrieb

Bildquelle: (10)

Die Translationsgeschwindigkeit v und Rotationsgeschwindigkeit ω können mittels v_l und v_r folgendermaßen berechnet werden:

$$v = \frac{v_l + v_r}{2}$$

$$\omega = \frac{v_r - v_l}{2d}$$

Daraus folgt zur Berechnung von v_l und v_r aus den im vorherigen Absatz beschrieben Steuerungskommando $\begin{pmatrix} v \\ \omega \end{pmatrix}$:

$$\begin{pmatrix} v_r \\ v_l \end{pmatrix} = \begin{pmatrix} 1 & d \\ 1 & -d \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Nun ergibt sich aus dem Radius R des Rades und Rotationsgeschwindigkeit w_r des rechten Rades v_r:

$$v_{r=\omega_r R}$$

Aus diesem Zusammenhang kann wiederum die direkte Berechnung der Rotationsgeschwindigkeiten w_r, w_l aus dem Steuerungskommando $\begin{pmatrix} v \\ \omega \end{pmatrix}$ erfolgen:

$$\begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{d}{R} \\ \frac{1}{R} & \frac{-d}{R} \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Mithilfe von $\begin{pmatrix}\omega_r\\ \omega_l\end{pmatrix}$ können die beiden Antriebsmotoren direkt angesteuert werden.
Der Radius r der Kreisbahn auf der sich der Roboter bewegt ergibt sich durch:
$r = d\frac{v_l+v_r}{v_r-v_l}$ (10)

2.1.3 Odometrie

Nachdem im vorherigen Abschnitt beschrieben wurde wie die Pose von Robotern definiert und wie diese mithilfe von Steuerungskommandos verändert werden kann, soll es nun darum gehen wie die Pose des Roboters bei fortlaufender Bewegung ständig neu bestimmt werden kann. So versteht man unter Odometrie Methoden zur Schätzung der Position, Orientierung und Geschwindigkeit eines mobilen Roboters relativ zu dessen Startposition. Um Odometriedaten zu erhalten werden zum Antrieb von mobilen Robotern oft Motoren mit Inkrementaldrehgeber verwendet. (5 S. 139-140). Im Folgenden soll die Bestimmung der Odometriedaten in Bezug auf Fahrzeuge mit Differentialantrieb und Inkrementaldrehgeber erläutert werden.

Ein Inkrementaldrehgeber sendet mit einer Frequenz $f = \left(\frac{1}{t_n - t_{n-1}}\right)^{-1}$ pro Radumdrehung eine bestimmte Anzahl an Pulse P_{ppr} (Pulse pro Rotation). Die gesendeten Pulse werden nun über die Zeit integriert. Anhand der gezählten Pulse p_{t_n} zum Zeitpunkt t_n und der gezählten Pulse $p_{t_{n-1}}$ zum Zeitpunkt t_{n-1} kann die zurückgelegte Strecke Δs im Zeitraum $\Delta t = t_n - t_{n-1}$, bei einem gegebenen Radradius R folgendermaßen berechnet werden:

$$\Delta s = \frac{p_{t_n} - p_{t_{n-1}}}{P_{ppr}} 2\pi$$

Mittels $\Delta s_l, \Delta s_r$ kann nun $v_l(t), v_r(t)$ (Notation aus *Abbildung 2*) bestimmt werden:

$$v_l(t) = \frac{\Delta s_l}{\Delta t} \qquad v_r(t) = \frac{\Delta s_r}{\Delta t}$$

Mittels dieser Informationen kann die Pose des Roboters $\begin{pmatrix}x(t)\\ y(t)\\ \theta(t)\end{pmatrix}$ bestimmt werden:

$$x(t) = \int_0^t v(\tau)\cos\big(\theta(\tau)\big)d\tau$$
$$y(t) = \int_0^t v(\tau)\sin\big(\theta(\tau)\big)d\tau$$
$$\theta(t) = \int_0^t \omega(\tau)d\tau$$

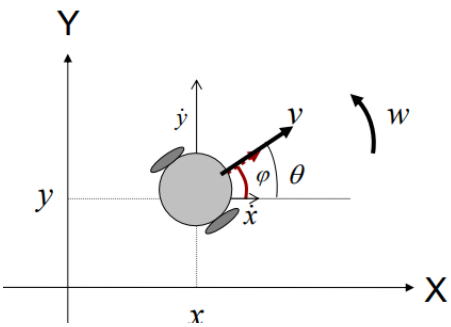

Abbildung 3: Berechnung der Pose eines mobilen Roboters mit Differentialantrieb

Quelle: (10)

Da die Odometriedaten, welche von einem Inkrementaldrehgeber stammen, mit der Zeit einem immer größer werdenden Fehler unterliegen, können diese meist nicht zur globalen Bestimmung der Pose benutzt werden. Oft wird zur Bestimmung der Pose ein Laser-Scan-Matching Verfahren angewandt. Der Nachteil bei dieser Art von Verfahren liegt darin, dass die Pose sich bei einem Miss-Matching der Laserscans sehr sprunghaft ändern kann. Da der Fehler bei Inkrementaldrehgebern zwar beliebig groß, aber dennoch stetig anwächst, ist es sinnvoll eine Kombination aus Inkrementaldrehgeber und Laser-Scan-Matching zu verwenden. Dies ist auch bei dem Roboter im Kontext dieser Arbeit der Fall. So kann bei dem Laser-Scan-Matching Verfahren anhand der Odometriedaten des Inkrementaldrehgebers besser abgeschätzt werden, ob die berechnete Pose im Bereich des Möglichen liegt und somit sprunghafte Änderungen der Pose verhindert werden.

2.2 Entfernungsmessung zur Umgebungserfassung

Durch Erfassung von Odometriedaten ist es möglich die Pose des Roboters relativ zum Koordinatenursprung des Odometrie-Bezugssystems zu bestimmen. Der Roboter hat so allerdings noch kein Verständnis von seiner Umwelt und es kann keine Aussage zur Roboterpose relativ zu möglichen Hindernissen gemacht werden. Deshalb ist es nötig den Roboter mit Sensoren zur Entfernungsmessung auszustatten. Die so erhaltenen Entfernungsmessungsdaten stellen sowohl die Grundlage für die Kartographierung und Lokalisierung, als auch für die Navigation des Roboters da.

2.2.1 Methoden zur Entfernungsmessung

Zunächst möchte ich auf die mathematischen Grundlagen zur Entfernungsmessung eingehen. Mithilfe dieser Grundlagen kann aus den von Lasern erfassten Daten die Entfernung zu Objekten, relativ zum Koordinatenursprung des Laser-Bezugssystems berechnet werden.

Time of Flight Entfernungsmessung
Zur Entfernungsmessung mit der Time of Flight (TOF) Methode wird ein sogenanntes Trigger-Signal vom Sensor ausgesandt. Trift dieses Signal auf ein Objekt, wird es reflektiert und kehrt zurück zum Absender. Nun wird gemessen wie lange das Signal unterwegs war. Mithilfe der gemessenen Zeit t und der Signalgeschwindigkeit v kann die zurückgelegte Strecke s und analog dazu die Distanz D, zwischen Sender und Empfänger bestimmt werden: (11 S. 9)

$$s = vt$$

$$D = v\frac{t}{2}$$

Der große Vorteil von TOF besteht also darin, dass aus den Sensordaten sehr leicht die Distanz zum Entfernten Objekt bestimmt werden kann, ohne irgendwelche komplexen Analysen oder Berechnungen durchführen zu müssen. Ein Nachteil ist,

speziell bei akustischen TOF Systemen, die variablen Ausbreitungsgeschwindigkeiten des Signals. Bei TOF Systemen bei denen Lasersignale verwendet werden, besteht die Schwierigkeit darin den Ankunftszeitpunkt des reflektierten Signals akkurat festzustellen. (12 S. 95-96)

Phasenvergleichsverfahren

Beim Phasenvergleichsverfahren wird durch die Messung der Phasenverschiebung die Distanz bestimmt. Sie ist somit eine besondere Form der TOF Entfernungsmessung. Bei dem Verfahren wird zunächst eine ununterbrochene Welle ausgesandt. Dabei wird die Amplitude des Trägersignals mithilfe von sinusförmigen Signalen mit unterschiedlichen Frequenzen so angepasst, dass sich ein sogenanntes Messignal ergibt. Das reflektierte Signal wird nun mit dem vorher gesandten Signal bezüglich der Phasenverschiebung $\Delta\varphi$ verglichen. Anhand der Wellenlänge λ des angepassten Signals und dessen Frequenz f kann nun die Distanz D zum entfernten Objekt berechnet werden:

$$D = \frac{\Delta\varphi\lambda}{4\pi} = \frac{\Delta\varphi v}{4\pi f}$$

Systeme die das Phasenvergleichsverfahren anwenden verwenden zwei Frequenzen zur Anpassung. Eine hohe Frequenz für eine hohe Auflösung, sowie eine niedrige Frequenz zur Messung großer Distanzen. (11 S. 10)

Triangulation

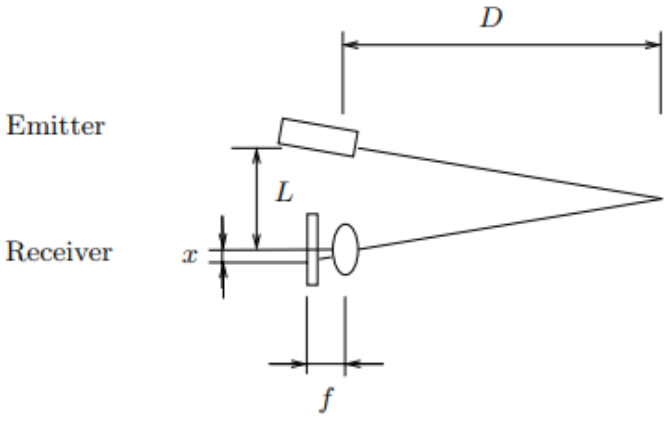

Abbildung 4: Funktionsprinzip der Triangulation

Die Triangulation ist ein weiteres Verfahren zur Entfernungsmessung. Anders als die TOF Verfahren setzt es nicht auf Zeitmessung, sondern macht sich geometrische Zusammenhänge zunutze. So werden der Sender und Emitter mit Abstand L am Sensor angebracht. Der Emitter sendet ein Signal, welches nachdem es von einem Objekt reflektiert wurde vom Empfänger, mithilfe einer Lochkamera, mit einer

Abweichung x erkannt wird. Anhand dieser Abweichung ist es möglich die Distanz zum reflektierenden Objekt zu berechnen: (11 S. 11)

$$D = f\frac{L}{x}$$

2.2.2 Sensoren zur Entfernungsmessung

Nachdem im letzten Kapitel geklärt wurde mit welchen Methoden Entfernungsmessungen durchgeführt werden können, möchte ich nun genauer auf die Sensoren eingehen, welche die erforderlichen Daten zur Anwendung der Methoden liefern. Dabei Beschränke ich mich auf die Sensoren, welche auch bei dem Roboter im Kontext dieser Arbeit verwendet werden.

Laserscanner

Abbildung 5: RPLIDAR A1 360° Laser Scanner
Quelle: (13)

Wie der Name vermuten lässt, sind Laserscanner-Systeme zur Entfernungsmessung, die Laserlicht nutzen, um Distanzen zu messen. Es wird eine Scanmethode benutzt die es erlaubt Oberflächen zu Digitalisieren. Um dies zu ermöglichen ist es nötig, dass sich die Laser rotieren oder bewegen lassen. Die Einsatzbereiche solcher Scanner sind u.a. Hindernisserkennung, Lokalisierung und Kartenerstellung. Es gibt Laserscanner sowohl mit TOF und Phasenvergleichsverfahren, als auch mit dem Triangulationsverfahren. Aufgrund der hohen Geschwindigkeit von Licht sind Laserscanner die TOF nutzen im Vergleich zu solchen die Triangulation nutzen sehr teuer. (11 S. 9-12) Ein 360° 2D-Laserscanner (siehe *Abbildung 5*) ist schon im Preisbereich um die 100 Euro erhältlich. Folgendes sind entscheidende Faktoren, die bei der Auswahl eines 2D Laserscanners berücksichtigt werden sollten:

Entfernungsbereich: In welchem Distanzbereich kann der Scanner Messungen vornehmen.

Winkelsichtbereich: In welchem Bereich von 0°-360° können Messungen vorgenommen werden

Winkelauflösung: Mit welcher Winkelauflösung haben die Scans. Beträgt der Winkelsichtbereich beispielsweise 180° es werden in dem Bereich 360 Messungen durchgeführt, beträgt die Winkelauflösung 0,5°

Scanfrequenz: Mit welcher Frequenz wird der Winkelsichtbereich gescannt.

Der Vorteil bei Laser-Scannern liegt darin, dass die Auswertung der gelieferten Daten zur Entfernungsberechnung keine hohe Rechenleistung benötigt. Außerdem kann eine sehr akkurate Entfernungsmessung gewährleistet werden.

Ultraschall-Sensor

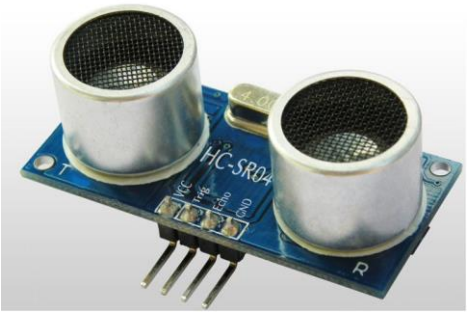

Abbildung 6: Ultraschall-Sensor HC-SR04
Quelle: (14)

Bei Ultraschallsensoren kann, im Gegensatz zu Laser-Scannern, die TOF Messung sehr kostengünstig umgesetzt werden. Dies ist darin begründet, dass die Schallgeschwindigkeit mit $343\,\frac{m}{s}$ um den Faktor 10^6 langsamer als die Lichtgeschwindigkeit mit $300\ x\ 10^9\,\frac{m}{s}$ ist. So ist der Sensor HC-SR-04 (siehe *Abbildung 6*) schon für unter drei Euro erhältlich. Ultraschallsensoren können zur Kollisionsvermeidung, Lokalisierung, Kartenerstellung und als Bewegungsdetektoren verwendet werden. Nachdem 2D 360° Lasersensoren heutzutage relativ günstig sind, werden Ultraschallsensoren alleine sehr viel weniger für Zwecke wie Lokalisierung und Kartenerstellung genutzt. Bei dem Roboter, um den es in dieser Arbeit geht, kommen drei Ultraschallsensoren zur Hindernisserkennung in Bereichen zum Einsatz, die der Lasersensor nicht erfassen kann. Ein großer Nachteil von Ultraschallsensoren ist, dass die Schallgeschwindigkeit je nach Temperatur, Luftdruck und Luftfeuchtigkeit variiert. Dies macht es schwierig die Genauigkeit eines solchen Sensors zu bestimmen. (9 S. 753-756)

2.3 Simultaneous Localization and Mapping

Der folgende Abschnitt soll eine grundlegende Einführung in das Thema Simultaneous Localization and Mapping (SLAM) darstellen. Es sollen zunächst die mathematischen Grundlagen auf denen SLAM Algorithmen basieren dargelegt werden, um darauf aufbauend den Extended Kalman Filter genauer zu Untersuchen. SLAM beschäftigt sich mit dem Problem eine räumliche Karte einer unbekannten Umgebung zu erstellen, während gleichzeitig die Pose des Roboters innerhalb der Karte bestimmt und laufend aktualisiert werden soll. Die Schwierigkeit besteht darin, dass sowohl Pfad und Karte unbekannt sind. Das SLAM Problem kann als eines der wichtigsten Probleme, wenn es um die Konstruktion von vollautonomen Robotern geht, angesehen werden. Die Lösung des SLAM Problems ist also von enormer praktischer Relevanz. Nachdem die Forschung auf diesem Gebiet in den letzten Jahren große Fortschritte erzielen konnte, ist es mittlerweile in einer statischen, strukturierten und größenbegrenzten Umgebung möglich eine sehr akkurate Kartographierung durchzuführen. Die Kartographierung von unstrukturierten, dynamischen und offenen Umgebungen, stellt jedoch weiterhin ein großes offenes Forschungsproblem dar. (9 S. 1153-1154). Das im Rahmen dieser Bachelorarbeit konstruierte Robotersystem beschränkt sich auf das Mapping von Innenräumen und fällt damit in die einfachere Kategorie.

2.3.1 Mathematische Grundlagen

Zunächst wird - analog zu 2.1.1 - die Roboterpose zum Zeitpunkt t als x_t festgelegt. Nun sei eine Sequenz solcher Posen, auch genannt Pfad, gegeben als:

$$X_T = \{x_0, x_1, x_2, \dots, x_T\}$$

Die initiale Pose x_0 dient oft als Ausgangspunkt für Estimation Algorithmen.
Des Weiteren wird festgelegt, dass u_t jene Odometriedaten beschreibt, welche die Translations- und Rotationsbewegung zwischen dem Zeitpunkt $t-1$ und dem Zeitpunkt t charakterisieren. Dann kann die relative Bewegung des Roboters durch folgende Sequenz ausgedrückt werden:

$$U_T = \{u_1, u_2, u_3, \dots, u_T\}$$

Theoretisch wäre es nun möglich, ausgehend von der initialen Pose x_0, mittels U_T alle Posen bis zum Zeitpunkt T zu berechnen. Jedoch unterliegen Odometriedaten immer einem gewissen Rauschen, welches dazu führt, dass das Ergebnis der Integration des Weges, zwangläufig von der Realität abweicht.
Natürlich erfasst der Roboter auch Objekte in seiner Umgebung. Sei m also die tatsächliche Karte der Umgebung. In dieser können sich nun Landmarken, Objekte, verschiedene Oberflächen usw. befinden, deren Position von m beschrieben wird.

Es wird zudem davon ausgegangen, dass die Umgebung mit Landmarken ausgestattet ist. Die Sensoren des Roboters erfassen nun Informationen bezüglich der Objekte in m und der Roboterpose. Es wird angenommen, dass zu jedem Zeitpunkt genau eine Messung vorgenommen wird. Daraus resultiert folgende Sequenz:

$$Z_T = \{z_1, z_2, z_3, \dots, z_T\}$$

In *Abbildung 7* wird das SLAM Problem durch einen Graphen dargestellt. So folgt aus den Odometriedaten u_t eine Pose x_t. In dieser Pose wird dann eine Messung der Umgebung u_t durchgeführt. Die Kanten geben kausale Beziehungen zwischen Knoten an. Die Informationen der gefärbten Knoten kann der Roboter jederzeit abrufen. Die Aufgabe von SLAM Verfahren ist es nun, aus den zur Verfügung stehenden Informationen, eine Karte zu erstellen, in der auch die Pose des Roboters ständig aktualisiert wird.

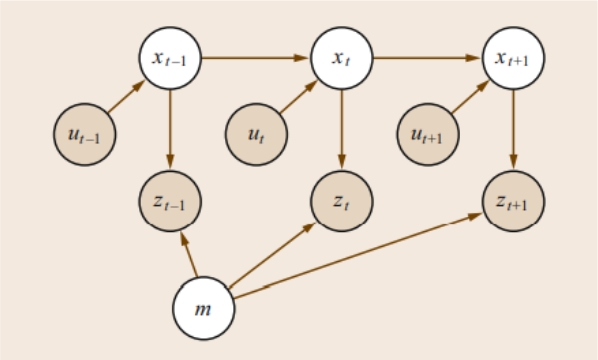

Abbildung 7: Probabilistisches Graphisches Modell des SLAM Problems
Quelle: (9 S. 1205)

Es gibt zwei Hauptkategorien in die das SLAM Problem unterteilt wird. Zum einen das das sogenannte Full Slam Problem. Beim Full SLAM wird anhand der Sequenzen von Odometriemessdaten U_T und Umgebungsmessungen Z_T, der gesamte zurückgelegte Pfad des Roboters X_T, zusammen mit der Karte m geschätzt:

$$p(X_{T,m} | Z_T, U_T)$$

Algorithmen für das Full SLAM Problem verarbeiten also zum Zeitpunkt T alle vorher gesammelten Daten, um eine Schätzung durchzuführen.
Die andere große Kategorie nennt sich Online SLAM Problem. Beim Online SLAM wird nur eine Abschätzung bezüglich der momentanen Roboterpose x_t und nicht des gesamten Pfades durchgeführt. Es ergibt sich:

$$p(x_{t,m} | Z_t, U_t)$$

Algorithmen zur Lösung des Online SLAM Problems werden oft als Filter bezeichnet.

Um das SLAM Problem lösen zu können müssen noch zwei weitere mathematische Modelle, in Bezug auf den Roboter und seine Umgebung eingeführt werden. Eines zur Zuweisung von Odometriedaten u_t zu den Roboterposen x_{t-1} und x_t. Zudem eines für die Zuweisung von Umgebungsmessungen z_t zur Umgebung m und der Roboterpose x_t.
Mathematische Modelle werden bei SLAM Verfahren für gewöhnlich als Wahrscheinlichkeitsverteilungen angesehen. So beschreibt das sogenannte *Motion Model* $p(x_{t,}|\, x_{t-1}, u_t)$ die Wahrscheinlichkeitsverteilung von x_t unter der Bedingung, dass die Roboterpose x_{t-1} bekannt ist und die Odometriedaten u_t erfasst wurden. Analog dazu beschreibt das *Observation Model* $p(z_{t,}|\, x_t, m)$ die Wahrscheinlichkeitsverteilung für die Messung z_t, unter der Annahme die Messung wurde von der bekannten Pose x_t durchgeführt. Es ist jedoch bekannt, dass beim SLAM Problem weder Pose des Roboters noch Kartendaten zur Verfügung stehen. Mithilfe des Satzes von Bayes kann dafür Abhilfe geschafft werden. (9 S. 1153-1155)

2.3.2 Landmarken

Für gewöhnlich wird bei SLAM Verfahren davon ausgegangen, dass sich in der zu observierenden Umgebung Landmarken befinden. Landmarken sind Umgebungsmerkmale die einen leichten Wiedererkennungswert besitzen. Vom Roboter können diese Merkmale genutzt werden, um sich in einer Umgebung zu Lokalisieren. Konkret könnten diese Landmarken z.B. Türrahmen, Ecken in Räumen und Tischfüße sein. Um die Lokalisierung zu ermöglichen, müssen solche Landmarken allerdings zunächst identifiziert werden. Das zu schaffen ist allerdings eine der größten Herausforderungen von SLAM Verfahren. (9 S. 1155)

2.3.3 Extended Kalman Filter SLAM

Die drei Standard-SLAM-Verfahren sind Extended Kalman Filter (EKF) SLAM, SLAM basierend auf Partikelfilter und Graph SLAM. Viele andere SLAM Verfahren basieren auf diesen drei Ansätzen. Im Folgenden werde ich das EKF SLAM Verfahren genauer untersuchen. EKF SLAM war historisch betrachtet eines der ersten SLAM Verfahren und hatte großen Einfluss auf viele der später entwickelten Methoden. Obwohl EKF SLAM heute, aufgrund seiner schlechten Laufzeiteigenschaften, nicht mehr sehr häufig eingesetzt wird, eignet sich das Verfahren trotzdem sehr gut, um ein Grundverständnis für SLAM zu erlangen.
Beim EKF SLAM Verfahren wird ein Zustandsvektor benutzt, um die Pose des Roboters und die Merkmale der Umgebung mit dazugehöriger Schätzfehler-Kovarianzmatrix, welche die Unsicherheiten dieser Näherungen repräsentiert, abzuschätzen. Während der Roboter sich in der Umgebung fortbewegt und Messungen durchführt, wird der Zustandsvektor und die Kovarianzmatrix unter Verwendung eines Kalman Filters aktualisiert.

Der EKF Algorithmus stellt die Näherungen des Roboters als mehrdimensionale Normalverteilung dar:

$$p(x_t, m \mid Z\_t, U\_t) = \mathcal{N}(\mu_t, \Sigma_t)$$

Der mehrdimensionale Vektor μ_t enthält die beste Näherung bezüglich der Pose x_t des Roboters und der Umgebungsmerkmale. Die Matrix Σ_t beschreibt die Kovarianz der Abschätzung des Fehlers, in der Annäherung μ_t.

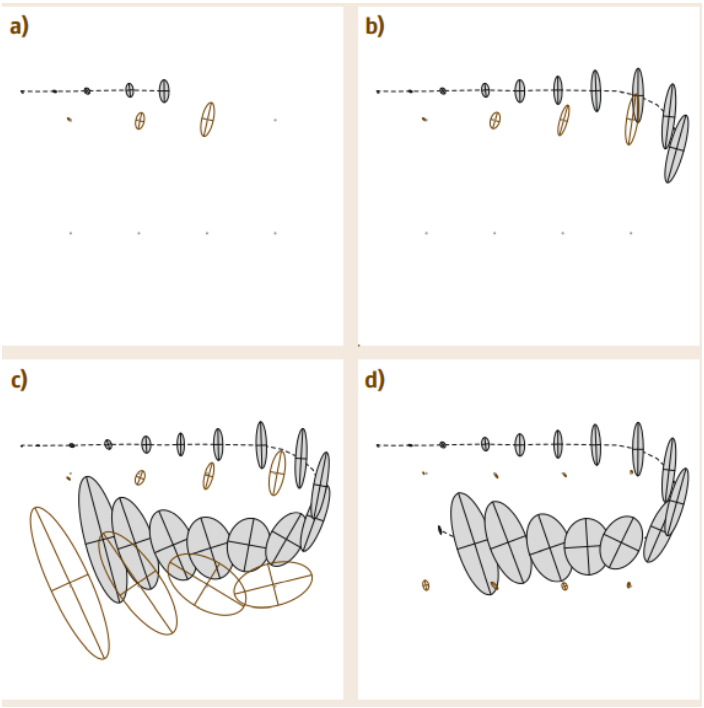

Abbildung 8: Beispiel EKF SLAM
Quelle: (9 S. 1158)

In *Abbildung 8* wird das EKF SLAM Verfahren anhand eines einfachen Beispiels veranschaulicht. Der Pfad des Roboters ist dabei als Strichlinie dargestellt. Die gräulichen Ellipsen stehen für die Abschätzung der Roboterpose. Die acht kleinen Punkte stehen für unterscheidbare Landmarken. Die weißen Ellipsen stellen die abgeschätzte Position der Landmarken dar. Sobald der Roboter damit beginnt, sich von seiner initialen Pose fortzubewegen, wächst die Unsicherheit bezüglich seiner Pose an. Der Roboter erfasst zudem naheliegende Landmarken. Nun werden Pose und Umgebungsmerkmale mit einer Unsicherheit in die Karte eingetragen, welche sich aus der statischen Unsicherheit der Umgebungsmessung und der ansteigenden Unsicherheit der Roboterpose zusammensetzt. Wie in **(a-c)** zu sehen, steigt zunächst sowohl die Unsicherheit der Roboterpose, also auch die Unsicherheit der Landmarkenposition an. In **(d)** trifft der Roboter plötzlich auf eine Landmarke die er gleich zu Beginn schon observiert hatte. Der Roboter erkennt die Landmarke wieder und somit reduzieren sich die Unsicherheiten schlagartig. Dies macht deutlich wie wichtig die Landmarkenidentifikation für eine akkurate Kartographierung und

Lokalisierung ist. (9 S. 1178-1158) Wird ein vorher schon gemappter Bereich wiedererkannt, wird dies als Loop-Closure bezeichnet. (15 S. 1). Mit der Zeit kann der Roboter so einen immer größeren Teil der Umgebung erkunden und dabei die Unsicherheit, bezüglich Roboterpose und Umgebungsmerkmale, möglichst geringhalten.

3 ROS - Robot Operating System

In letzten Kapiteln wurden die mathematischen und physikalischen Grundlagen für die Ausarbeitung des Robotersystems behandelt. Nun möchte ich auf die Roboter-Middleware eingehen, die zur Erstellung der Software für den Roboter verwendet wird: Das Robot Operating System (ROS). Anders als der Name vermuten lässt, ist ROS kein vollwertiges Betriebssystem. Es ist lediglich ein Meta-Betriebssystem, dass auf Unix basierten Betriebssystemen installiert werden kann. Eine Portierung auf andere Betriebssysteme ist möglich und wurde auch schon experimentell für Windows umgesetzt. Im Kontext dieser Arbeit wird mit Ubuntu 20.04 und ROS Noetic gearbeitet. ROS bietet betriebssystemähnliche Services und Infrastruktur, wie Hardware-Abstraction, Gerätetreiber, Implementierung von häufig genutzten Funktionalitäten, Nachrichtenaustausch zwischen Prozessen und Paketmanagement. Der Grundgedanke bei dem Open Source Projekt ROS ist es also, nicht bei jedem Roboter das Rad neu erfinden zu müssen. In diesem Sinne soll bei diesem Roboter ROS für die Kommunikation zwischen den verschiedenen Komponenten des Systems genutzt werden. Zudem sollen für die Kartographierung, Lokalisierung und Navigation des Roboters, wenn möglich, schon vorhandene ROS Standardpakete verwendet werden. Es folgt nun ein Überblick der wichtigsten Komponenten und Funktionalitäten von ROS.

3.1 Grundlagen ROS

Pakete
Softwarekomponenten in ROS sind als Pakete strukturiert. Diese Pakete können Nodes, 3rd-Party-Libraries, Configuration-Files oder alle anderen spezifischen Bestandteile, die zur Funktion der in dem Paket gebündelten Software nötig sind, enthalten. Sowohl die vom ROS Framework angebotene Software, als auch selbst erstellte Software wird als Pakete bereitgestellt. Dies soll die Wiederverwendbarkeit erleichtern. (16)

Nodes
Nodes sind lediglich ausführbare Dateien, die einen eindeutigen Namen haben und den ROS Client nutzen, um mit anderen Knoten zu kommunizieren. Die Knoten können sich in verschiedenen Namespaces befinden. (17)

roscore
Der roscore ist eine Ansammlung von Nodes die als Mindestvoraussetzung für ein laufendes ROS Netzwerk gelten. So startet der roscore den ROS **Master** und **rosout**. Erst wenn der roscore gestartet wurde, ist eine Kommunikation zwischen Nodes möglichen. (18)

Master

Der ROS Master bietet einen Namens- und Registrierungsservice für Nodes und alle andren Komponenten des ROS Ökosystems. Der Master stellt außerdem einen **Parameterserver** zur Verfügung.

Topics
Topics sind Kommunikationskanäle mit einem eindeutigen Namen. Über diese Kanäle können Nodes **Nachrichten** austauschen. Die Struktur dieser Nachrichten muss eindeutig festgelegt sein. Es können entweder von ROS bereitgestellte Standardnachrichtentypen verwendet werden oder eigene Nachrichtentypen definiert werden. (19)

Publisher - Subscriber Pattern

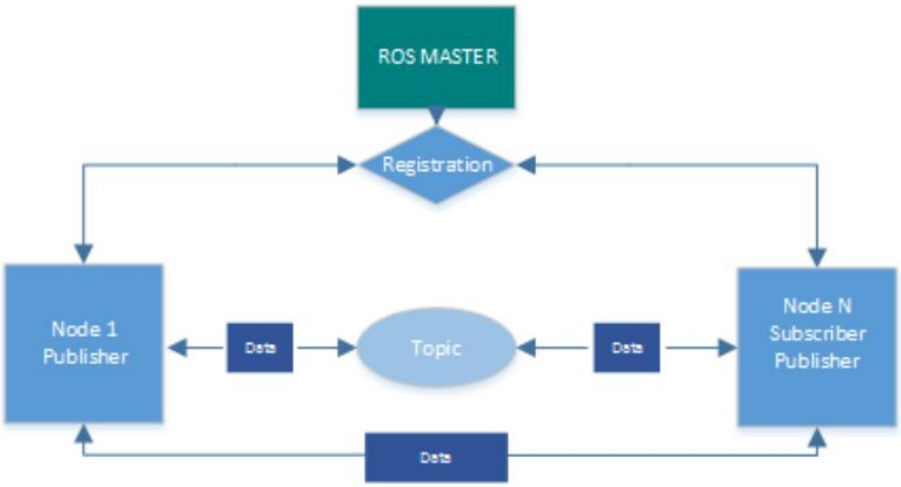

Abbildung 9: ROS Publisher-Subscriber Pattern
Quelle: (20)

Zum Nachrichtenaustausch in ROS wird auf das Publisher-Subscriber Pattern gesetzt (siehe Abbildung 9). Nachdem der ROS Master gestartet wurde, können sich Nodes bei diesem registrieren. Dabei wird festgehalten auf welche Topics ein Knoten Nachrichten publishen will und von welchen Topics er Nachrichten empfangen will. Es können beliebig viele Knoten auf ein Topic subscriben. Es können auch beliebig viele Knoten Nachrichten auf ein Topic publishen. Ob dies jedoch sinnvoll ist, muss je nach Einsatzgebiet entschieden werden. Ein richtig konfiguriertes Netzwerk vorausgesetzt, können Nodes auf verschiedenen Rechnern in einem Netzwerk laufen und untereinander kommunizieren. Unter Verwendung von VPN-Tunneln ist dies auch über entfernte Netzwerke möglich. (21 S. 31-33)

Parameter Server
Der Parameter Server ist ein geteiltes Dictionary auf das von allen Nodes in einem ROS Netzwerk zugegriffen werden kann. Die Einträge sind wie Knoten in Namespaces unterteilt. (22)

roslaunch
Bei roslaunch handelt es sich um ein Tool für das komfortable launchen von mehreren Nodes. Dazu lädt roslaunch ein oder mehrere XML-Dateien in denen u.a. spezifiziert ist, welche Parameter gesetzt werden sollen, welche Configuration-Files geladen

werden sollen und welche Nodes gestartet werden sollen. Configuration-Files werden in ROS meist im YAML Dateiformat abgespeichert.

URDF
URDF steht für Universal Robot Description Format und hat den Zweck die physischen Merkmale eines Roboters zu beschreiben. Diese Dateien werden genutzt, um über einen Parser, ROS Informationen über Aussehen und physikalische Eigenschaften eines Roboters zur Verfügung zu stellen. In URDF beschreibt ein **Link** ein Objekt (z.B. Fahrzeugrahmen, Reifen, Sensor) und ein **Joint** einen Punkt an dem zwei Objekte miteinander verbunden sind. Die Komponenten eines Roboters können so in einer Baumstruktur dargestellt werden. Die Namen von Links werden auch zur Identifikation des Bezugssystems, in dem sich das Objekt befindet genutzt (23)

TF
TF (Transform) ist ein ROS Paket mit der Funktion die Transformation zwischen Bezugssystemen durchzuführen. Da ein in URDF beschriebener Roboter die relative Position aller Links zueinander beschreibt, können diese Informationen von TF genutzt werden, um Transformationen durchzuführen. TF behält zudem über die gesamte Laufzeit einen Überblick über die aktuelle Position und Ausrichtung von allen Links. Auf all dieses Informationen kann von jedem Knoten, über eine Subscription des TF Topics, zugegriffen werden. TF ist somit Basis für viele Kernaufgaben der mobilen Robotik. So müssen beispielsweise die von einem Sensor erfassten Entfernungsmessdaten für die Kollisionsvermeidung in die Bezugssysteme anderer Links des Roboters transformiert werden (24)

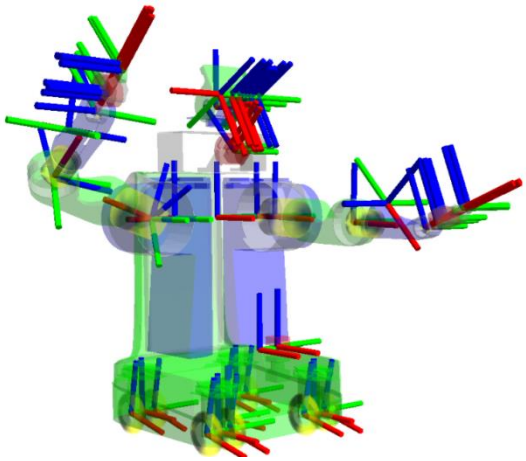

Abbildung 10: Visualisiertes URDF Modell eines Roboters und den aktuellen TF Frames
Quelle: (24)

Rviz
Rviz ist ein 3D Visualisierungstool für ROS. Es dient dazu, das Robotermodell, Sensormessdaten, Karten TF-Daten usw. zu visualisieren. Zudem können über die Rviz GUI auch Nachrichten gepublished werden. So ist es u.a. möglich per Mausklick Zielpunkte zu publishen, zu denen der Roboter navigieren soll.

3.2 ROS Navigation Stack

Auch für die Navigation von mobilen Robotern stellt ROS eine Reihe von Schnittstellen zur Verfügung. Mithilfe des ROS Navigation Stack (siehe Abbildung 11), kann die Navigation eines Roboters, durch die Integrierung von Mapping, Lokalisierung und Pfadplanung erreicht werden. Der Navigation Stack nutzt Odometriedaten, Sensordaten, Kartendaten und Zielposition um Steuerungskommandos zu berechnen, die dann an den Roboter gesandt werden und diesen ans Ziel führen sollen. Zunächst möchte ich genauer auf die Schnittstellen des Navigation Stack eingehen. Danach werde ich die internen Abläufe des Navigation Stack erläutern. (21 S. 75)

3.2.1 Schnittstellen zum Navigation Stack

Beim Roboter, der im Rahmen dieser Arbeit erstellt wurde, stammen die Informationen ausfolgenden Quellen und werden an die Schnittstellen des Navigation Stack übergeben (siehe zur Einordnung der erwähnten Nodes Anhang D_A1):

Topic: **/tf** - Message: **tf/tfMessage**
Die Message wird aus einer Kombination aller transformierter Sensor-, Odometrie- und Lokalisierungsdaten erstellt und u.a. vom */robot_state_publisher* Knoten gepublished.

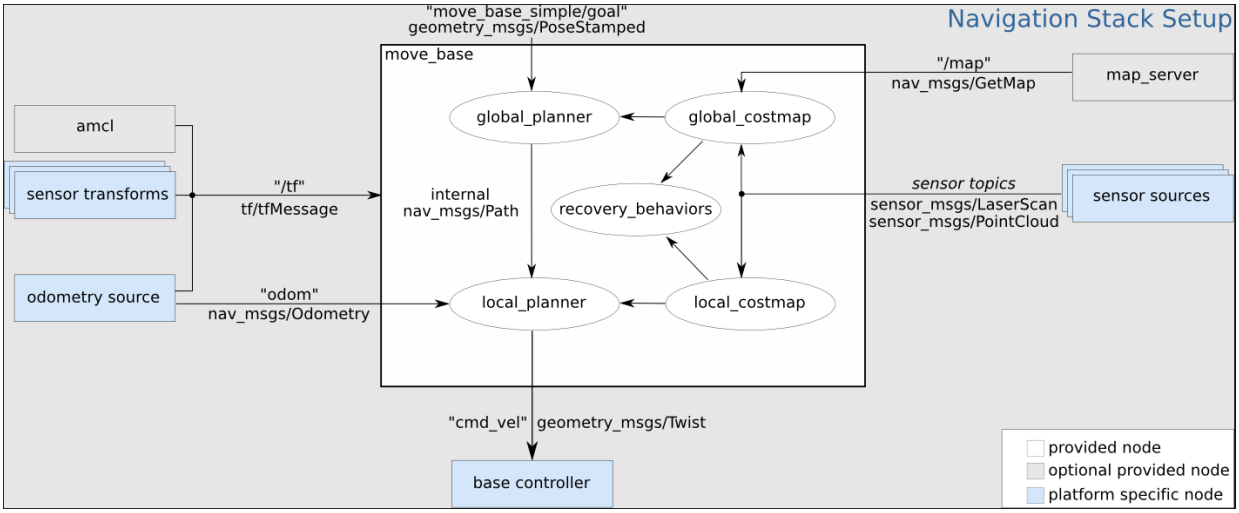

Abbildung 11: ROS Navigation Stack Setup
Quelle: (25)

Topic: **/odom** - Message: **nav_msgs/Odometry**
Die Quelle für die Odometriedaten sind die Wheel-Encoder der DC-Motoren des Roboters. In der Simulation werden die Daten von dem Simulationsprogramm Gazebo bereitgestellt. Gepublished werden die Messages vom ROS */diffbot_hardware_interface Knoten.*

Topic: **/map** - Message: **/nav_msgs/GetMap**
Die Karte kann entweder statisch zur Verfügung gestellt werden und stammt so vom map_server oder laufend durch ein SLAM Verfahren aktualisiert werden und stammt so z.B. vom */slam_gmapping* Knoten. Es ist auch möglich den Navigation-Stack ganz ohne Karte zu nutzen.

Topic: **/scan** - Message: **/sensor_msgs/LaserScan**
Die Laserscandaten werden in echt von einem Laserscanner und in der Simulation von Gazebo bereitgestellt. Beim realen Roboter steht ein ROS Treiber für den verwendeten Laserscanner zur Verfügung. In diesem Fall published der */rplidarNode* Knoten die Messages.

Topic: **/move_base_simple/goal** - Message: **/geometry_msgs/PoseStamped**
Mithilfe von Rviz kann ein Navigationsziel gepublished werden. Dies könnte natürlich auch von jedem anderen Knoten gemacht werden.

Anhand all dieser Informationen erstellt der Navigation-Stack dann eine **/geometry_msgs/twist** Nachricht und published sie an das **/cmd_vel** Topic. Dieses Kommando kommt, nachdem es die Schichten des diff_drive_controller und des Roboter-Hardware-Interface durchlaufen hat, letztendlich an den DC Motoren an und steuert so gezielt den Roboter.

3.2.2 Pfadplanung des Navigation Stack

Ich möchte nun darlegen, wie die am Ende des letzten Abschnitts erwähnte **/geometry_msgs/twist** Nachricht zustande kommt. Die Pfadplanung wird unter Verwendung des *move_base* Pakets durchgeführt und ist in globale (*global_path_planner*) und lokale (local_path_planner) Pfadplanung unterteilt. Der globale Pfadplaner sucht nach dem kürzesten Weg zum Zielpunkt. Der der lokale Pfadplaner versucht - immer unter Berücksichtigung von aktuellen Sensormessdaten – Steuerungskommandos zu erstellen, die den Roboter dem globalen Pfad folgen lassen und dabei Hindernissen ausgewichen wird. Informationen zu Hindernissen stehen durch die *global_costmap und local_costmap* zur Verfügung. Während die *global_costmap* anhand der Map aus dem */map* Topic erstellt werden kann, wird die *local_costmap* basierend auf aktuellen Sensormessdaten ermittelt. (21 S. 76)

3.3 ROS Control

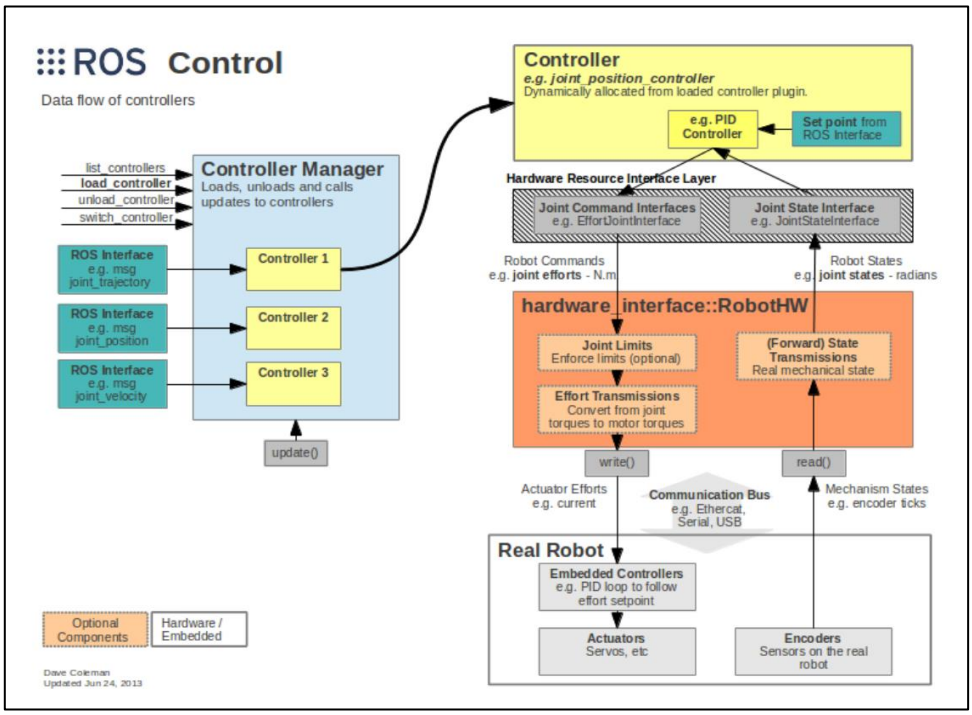

Abbildung 12: Übersicht ROS Control
Quelle: (26)

Es kann eine anspruchsvolle Aufgabe sein Roboter Regelungs- und Steuerungssoftware zu implementieren. Das Ziel von ROS Control (siehe Abbildung 12) ist es Roboter Hardware dem ROS Ökosystem möglichst unkompliziert

bereitstellen zu können und zugleich Regelungs- und Steuerungssoftware auf die gleiche Art wiederverwendbar zu machen, wie es ROS schon in höheren Schichten, wie beispielsweise mit dem Navigation Stack macht. Hierzu wird ein Hardware-Abstraction-Layer eingeführt. Diese Schicht kann durch Verwendung der *hardware_interface::RobotHW* Klasse implementiert werden. Dieser neue Layer bietet sowohl eine Schnittstelle zur Roboter Hardware, als auch zum Controller Layer. Ein Controller in ROS ist ein Plugin für den Controller Manager, welches das Controller Interface Implementiert und üblicherweise über ein sogenanntes *JointCommandInterface* einem Aktuator die Stellgröße übergibt. Daten von Sensoren erhält der Controller für gewöhnlich über ein *JointStateInterface.* Es sei erwähnt, dass Controller in ROS nicht immer die Art von Controllern sein müssen, die aus der Steuerungs- und Regelungstechnik bekannt sind. Der *JointStateController* beispielsweise steuert nichts, sondern published lediglich Positions- und Geschwindigkeitsdaten eines Gelenks. (21 S. 687)

4 Simulation mit Gazebo

Da nun alle theoretischen Grundlagen geklärt sind, kann in den praktischen Teil übergegangen werden. Zunächst wurde eine Simulation des Robotersystems mit Gazebo erstellt. Gazebo ist eine Roboter-Simulationssoftware die durch Plugins sehr gut geeignet ist, Simulationen von Robotern zu erstellen, deren Softwarebasis ROS ist. Zuerst eine Simulation zu erstellen ist sinnvoll, da so getestet werden kann, ob Antriebsart, Sensoren und ROS Pakete für das Robotersystem so gewählt wurden, dass es seine Aufgaben zuverlässig ausführen kann. In meinem Fall ist das ein mobiler Roboter mit folgenden Eigenschaften, Komponenten und ROS Paketen:

- Antriebsart: Differentialantrieb
- 2D 360° Lasersensor mit 5Hz Scanfrequenz und 12m Reichweite
- Ultraschallsensor mit 5Hz Scanfrequenz und 4m Reichweite
- Controller: *diff_drive_controller, joint_state_controller, robot_state_controller*
- Pakete für SLAM: *gmapping, cartographer_ros*
- Global Planner Paket: *navfn/NavfnROS*
- Local Planner Paket: dwa_local_planner/DWAPlannerROS
- Paket für Lokalisierung: *amcl*

Es wurden noch andere ROS Standardpakete verwendet die jedoch für den Kontext des Problems nicht so relevant sind.

Für die Simulation musste nun ein URDF Model des Roboters (siehe Anhang D_A2) erstellt werden. Darin sind alle nötigen geometrischen Eigenschaften, physikalischen Eigenschaften, Sensoren und Controller beschrieben, die für eine Simulation nötig sind. Unter Verwendung des *ros_gazebo* Pakets und dem darin enthaltenen Knoten *diffbot_spawn konnte* das erstellte Modell nun in Gazebo geladen und auch in Rviz visualisiert werden:

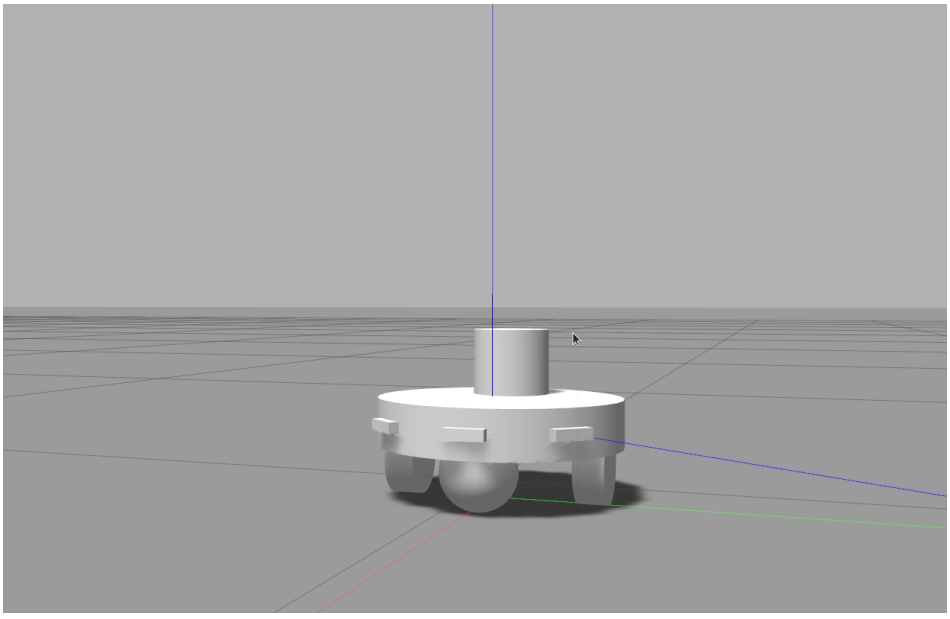

Abbildung 13: Modell des Roboters in Gazibo

Der Transformation-Tree kann im digitalen Anhang **D_A3** betrachtet werden. Bei der Größe des Roboters wurde darauf geachtet, dass der Durchmesser des Zylinderförmigen Fahrgestells unter 220mm beträgt. Dies war nötig, weil dies die maximale Druckgröße des verwendeten 3D-Druckers ist. Man kann sehen, dass in der Simulation die Sensoren funktionieren und die Entfernungsmessdaten korrekt in Rviz dargestellt werden.

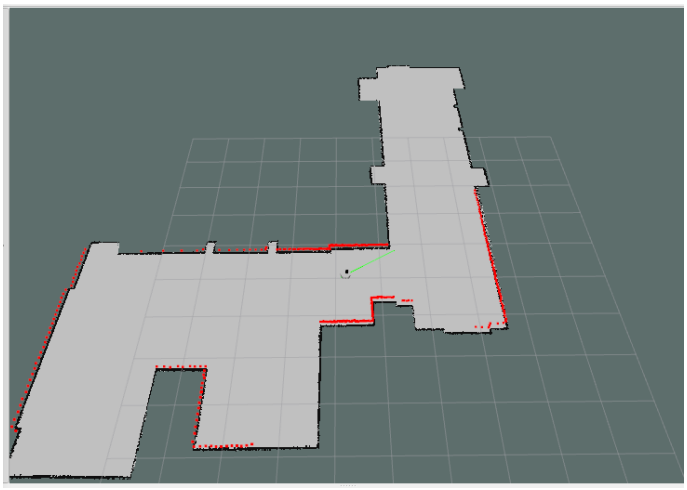

Abbildung 14: Visualisierung in Rviz

Jetzt wurden alle notwendigen ROS Pakete für SLAM und Navigation installiert und einige Launch-Files definiert Dem Roboter können nun über Rviz Zielpunkte zugwiesen werden zu denen der Roboter navigiert und währenddessen SLAM durchführt. Es ist auch möglich den Roboter über die Tastatur zu steuern. Dazu wird das *teleop_twist_keyboard* Paket von ROS verwendet. Das Ergebnis sieht folgendermaßen aus.

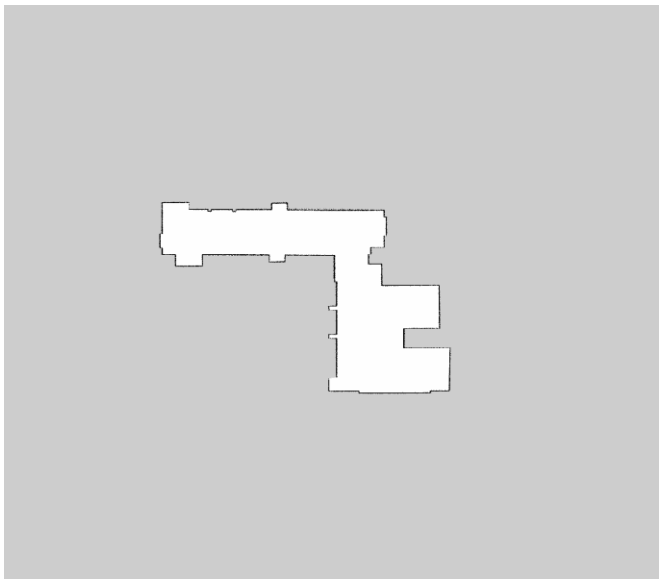

Abbildung 15: Erstellte Karte mit Gmapping

5 Setup Realer Roboter

5.1 Roboter Design

Da die Auswahl von Antriebsart, Sensoren und Algorithmen durch die gut funktionierende Simulation bestätigt wurde, möchte ich nun auf das Design des echten Roboters eingehen.

Systemarchitektur
Zunächst möchte ich mit Abbildung 5 einen Überblick zur System-Architektur bereitstellen. Dort sind die Aufgaben der einzelnen Komponenten, sowie der Informationsfluss eingetragen.

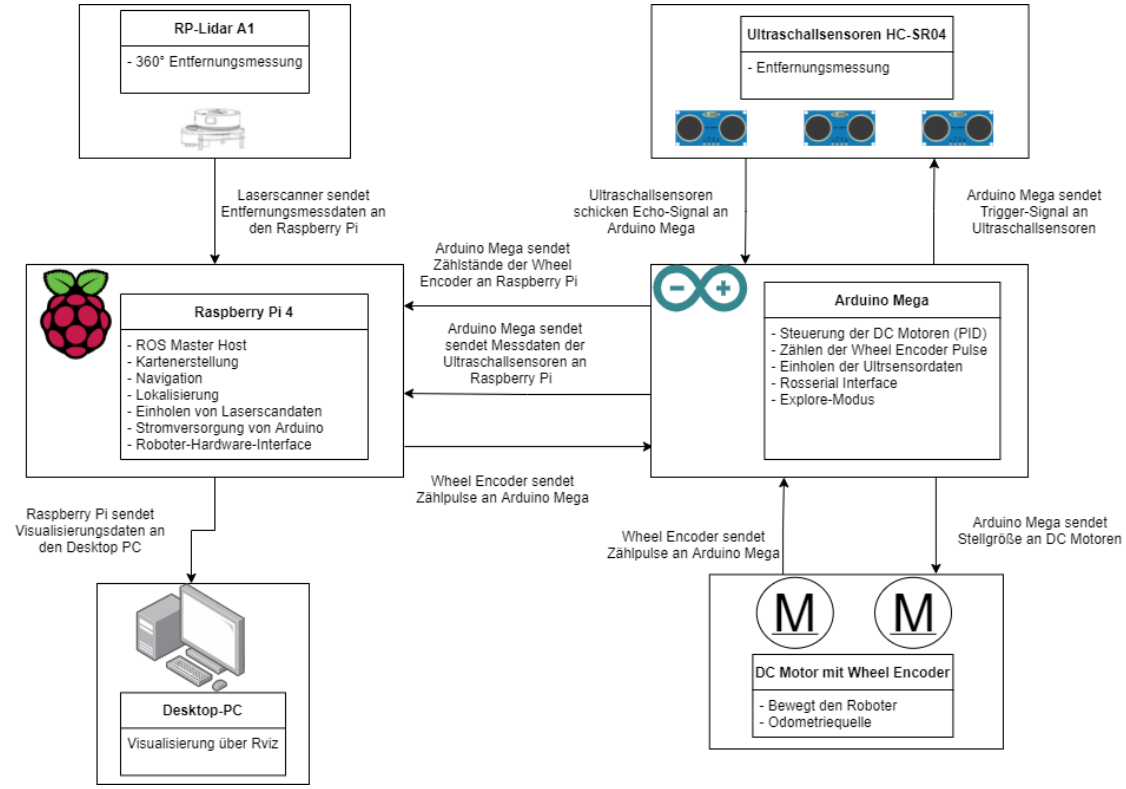

Abbildung 16:Systemarchtektur

Wie man sieht werden die rechenintensiveren Aufgaben wie Mapping und Pfadplanung auf dem Raspberry Pi durchgeführt. Diese nicht auf den Desktop PC auszulagern hat den Vorteil, dass der Roboter so theoretisch autark alle Aufgaben durchführen kann und keine Verbindung zu einem externen Datenverarbeitungssystem vorhanden sein muss. Der Desktop-PC wird nur als Visualisierungsplattform mittels Rviz genutzt. Während der Lasersensor seine Daten direkt, über einen von ROS bereitgestellten Treiber, im ROS Netzwerk verfügbar machen kann, müssen die Daten des Wheel -Encoders und der Ultraschallsensoren einen Umweg über den Arduino Mega machen.

Verwendete Hardware
Hier eine vollständige Liste der verwendeten Hardwarekomponenten

- Raspberry Pi4 B (8GB) als internes Datenverarbeitungssystem
- Arduino Mega (ATMEGA 2560) als internes Datenverarbeitungssystem
- PC (i7 7700, 16GB RAM, GTX980) als externes Datenverarbeitungssystem
- RpLidar 2D 360° Lidar Sensor, 5,5Hz Scanfrequenz, 12m Reichweite
- 14.4V Li-Ionen Akku (5200mAh) zur Energieversorgung der internen Systeme
- Step-Down-Converter: 5V Raspberry Pi 4 und Arduino. 12V für DC Motor.
- SR-HC04 Ultraschallsensoren zur Kollisionsvermeidung
- 2x 12V DC Getriebemotor – max. 60rpm
- Wheel Encoder der 1079 Pulse pro Rotation zählt
- TB6612FNG Motor-Driver

Konstruktion, Herstellung und Zusammenbau des Roboters
Unter Verwendung des 3D CAD Programms Autodesk Inventor Professional 2021 habe ich anhand der groben Vorgaben des URDF Modells das Gehäuse des mobilen Roboters konstruiert. Die 3D Modelle können im digitalen Anhang **D_A4** gefunden werden. Dabei habe ich ein Augenmerk daraufgelegt, wie die Konstruktion aussehen muss, damit alle Komponenten auf dem engen Raum untergebracht werden können. Dazu habe ich auch Dummy-Modelle der Hardwarekomponenten in der Baugruppe platziert. Ein weiterer wichtiger Aspekt war es die schweren Bauteile möglichst über dem Vorderrad zu positionieren, um ein umkippen des Roboters zu verhindern. Deshalb wurde die Batterie möglichst weit vorne platziert. Zudem sei zu erwähnen, dass die Achse der Räder so positioniert, dass ihr Mittelpunkt genau die Symmetrieachse des zylinderförmigen Fahrgestells schneidet. Somit ist eine Drehung auf einer Stelle möglich, ohne dass sich der Platzbedarf des Roboters im Raum ändert. Im folgenden Schritt wurden die Gehäusebauteile mit einem Ender-3-Pro 3D Drucker unter Verwendung von PLA Filament ausgedruckt. Dann erfolge die Montage. Zudem wurde noch ein Custom-Shield für den Arduino Mega mittels einer Lochrasterplatine Platine erstellt. Mithilfe von JST-Verbindungsstücken können Sensoren und DC-Motoren komfortabel angeschlossen werden und Wackelkontakten wie bei einfachen Jumpern kann vorgebeugt werden. Hier nun das Ergebnis der erstellten 3D Baugruppe und des echten zusammengebauten Roboters:

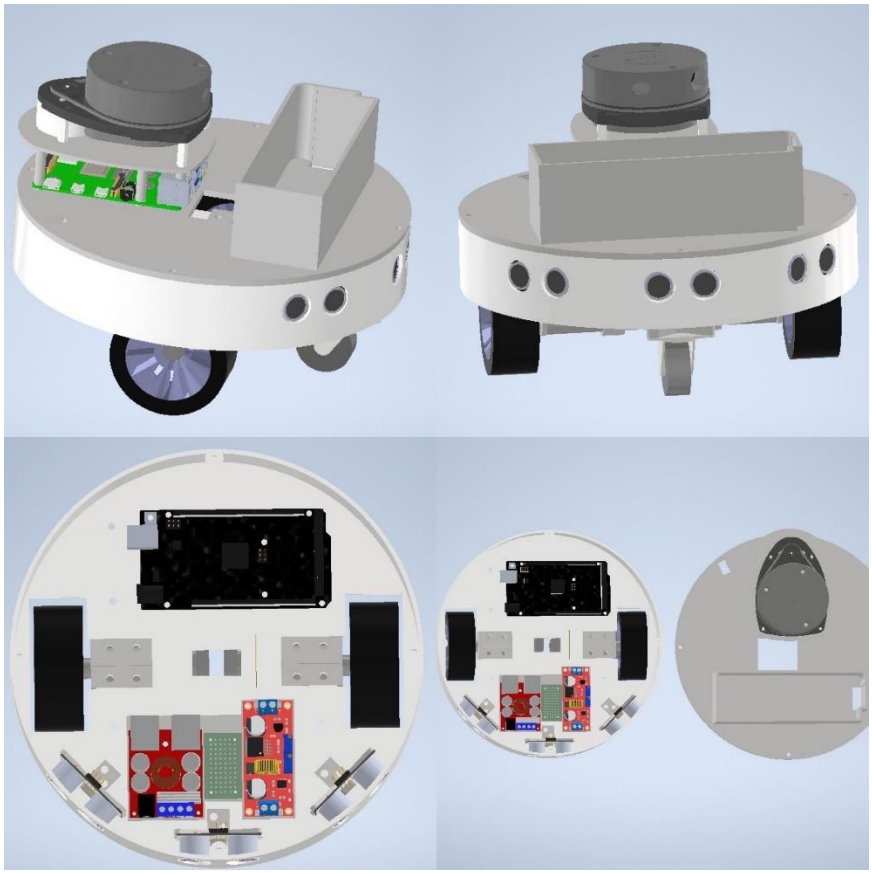

Abbildung 17: 3D Modell des Roboters

Abbildung 18: Echter Roboter

5.2 Software Implementierung

5.2.1 Implementierung Roboter Hardware-Interface Node

Das der Roboter nun samt seiner Hardwarekomponenten zusammengebaut war, konnte mit der Implementierung der Software fortgesetzt werden. Um nun eine Kommunikation zwischen ROS Control / ROS Navigation Stack und dem ATMEGA 2560 inklusive seiner Sensoren und DC Motoren zu ermöglichen ist es nötig ein Hardware Interface zu implementieren. Diese Schnittstelle zum realen Roboter ersetzt die Schnittstelle, welche in der Simulation von Gazebo und den dazugehörigen Gazebo ROS Plugins bereitgestellt wurde. Auf diese Weise ist es möglich alle Pakete und deren Konfiguration aus der Simulation ohne Änderungen beim Setup mit dem realen Roboter wiederzuverwenden. Die Voraussetzung dafür ist natürlich, dass der in URDF beschriebene Roboter mit dem realen Roboter übereinstimmt. Die Mächtigkeit von ROS wird klar, wenn man betrachtet, in welchem Kontext meines SLAM- und Navigationsroboters sich die Softwarebestandteile befinden, die ich selbst implementieren muss. Es müssen lediglich die untersten beiden Schichten implementiert werden, die in Abbildung 19 blau markiert sind. Das ist zum einen das Roboter-Hardware-Interface auf dem Raspberry-Pi und als Gegenstück dazu die Software auf dem Mikrocontroller inkl. Treiberschichten, PI-Regler und RosSerial-Interface. Alle anderen Softwarekomponenten zur Realisierung des Robotersystems werden von ROS bereitgestellt.

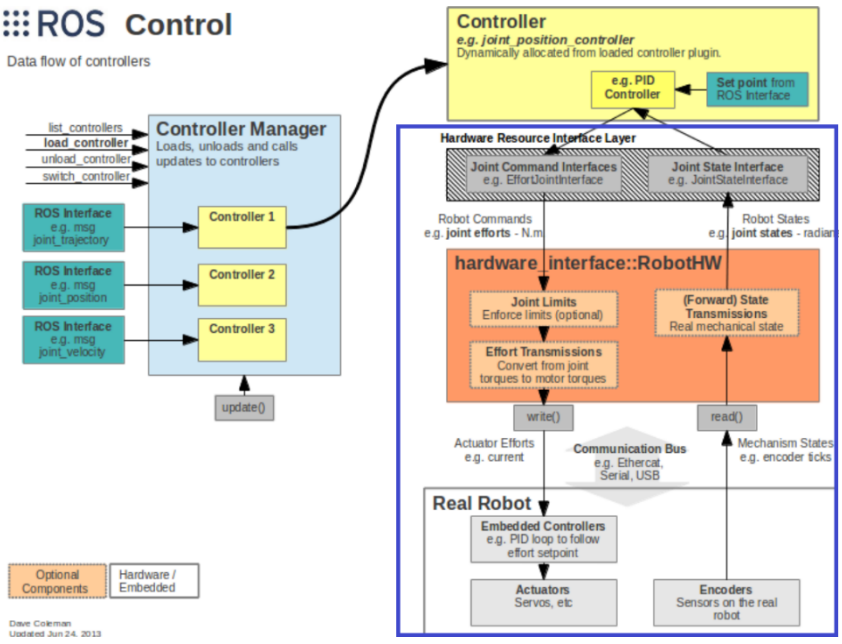

Abbildung 19: Zu implementierende Softwarekomponenten
Quelle: (26)
(Bild abgeändert)

Um die Struktur und Funktion der Robot-Hardware-Interface Node darstellen zu können, habe ich ein Flow-Chart Diagramm erstellt. Dieses ist in Abbildung 20 dargestellt.

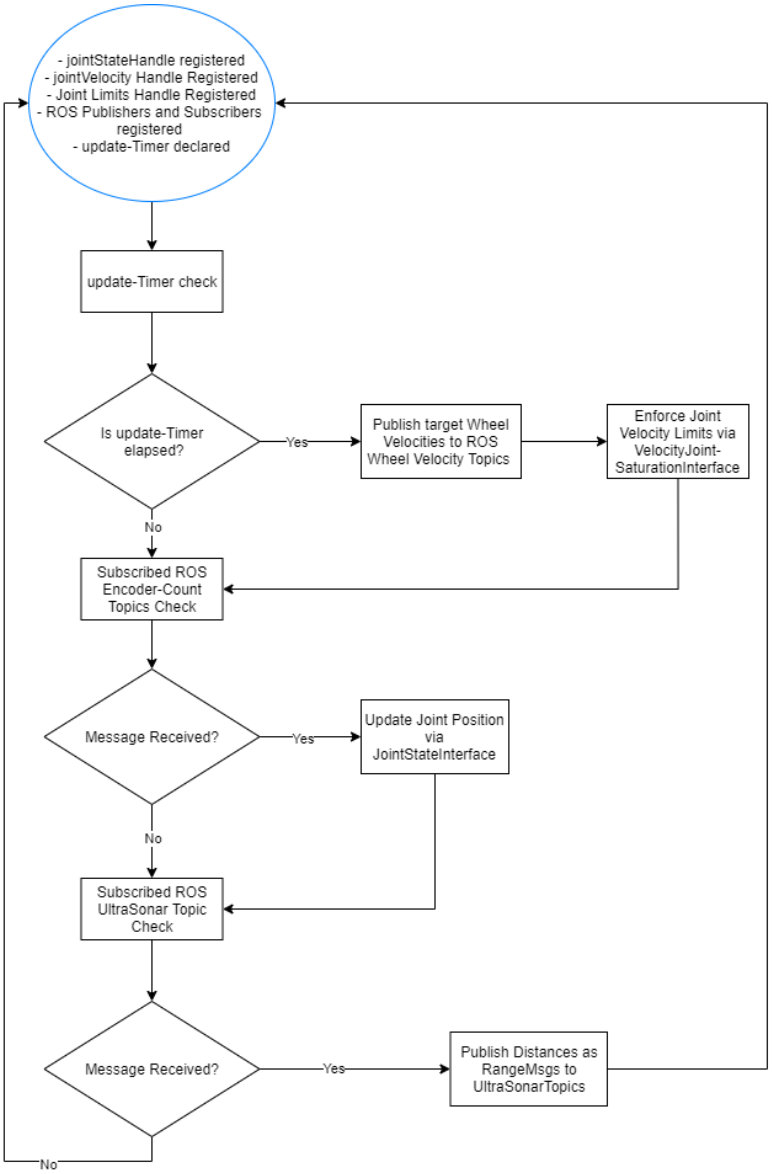

Abbildung 20: Flow-Chart des Robot-Hardware-Interfaces

5.2.2 Implementierung Serielle Schnittstelle

In diesem Schritt wurden zunächst die Treiber für die verwendeten DC-Motoren und Ultraschallsensoren implementiert, um dann letztendlich eine serielle Schnittstelle zum Mikrocontroller zur Verfügung zu stellen.

DC-Motor Treiberschicht

Der DC Motor Treiber bietet folgende Schnittstelle, um sowohl Drehzahldaten auszulesen, als auch um die gewünschte Drehzahl des Motors festzulegen:

```
void startMotor(); // Get Motor ready to accept Target Speeds

void stopMotor(); // stop the Motor

void setTargetPPM(int32_t _targetPpm); // Set Target Pulses Per Minute

int32_t getPPM(); // Get Current Pulses Per Minute
```

Zur Regelung der Motordrehzahl wurde ein PI-Regler implementiert. Die Performance des Reglers kann anhand von Abbildung 21 betrachtet werden.

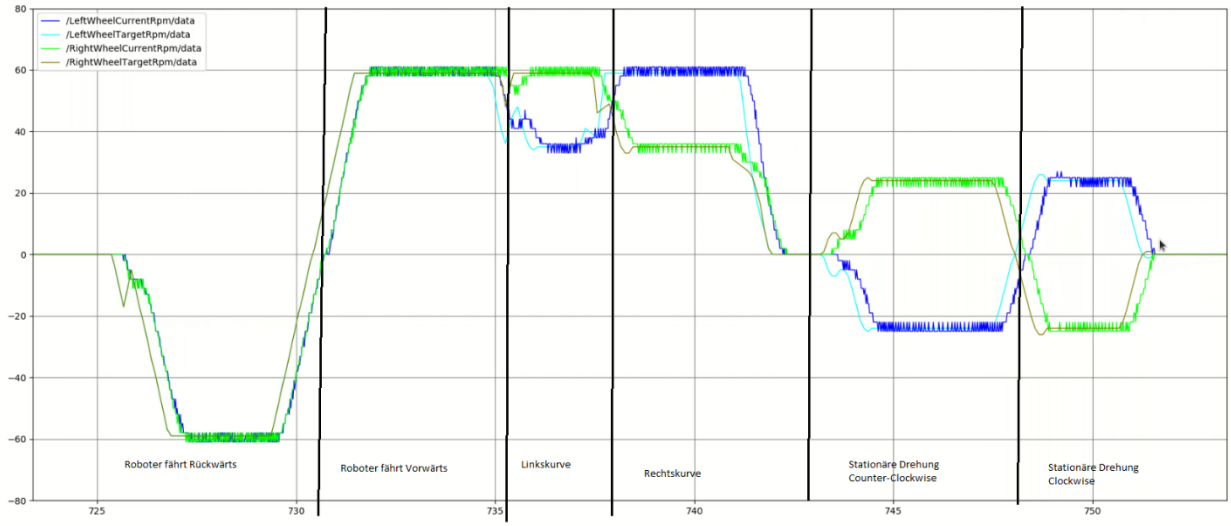

Abbildung 21: Performance PI-Regler

Die grünlichen Kurven stellen Soll- und Ist-Drehmoment des rechten DC-Motors dar und die bläulichen Kurven stellen die des linken Motors dar. Es ist zu erkennen, dass Der PI Regler die Zieldrehzahl sehr reaktiv erreicht. Im ersten Abschnitt sind die Drehzahlen beider Motoren negativ und so fährt der Roboter Rückwärts. Es ergeben sich folgende weitere Fahrverhalten:

Drehzahl links: stark positiv, Drehzahl rechts: stark positiv: -→Roboter fährt Geradeaus

Drehzahl links: leicht positiv, Drehzahl rechts: stark positiv: -→ Linkskurve

Drehzahl links: stark positiv, Drehzahl rechts: leicht positiv: -→ Rechtskurve

Drehzahl links: leicht negativ, Drehzahl rechts: leicht positiv:-→ Stationäre Drehung gegen den Uhrzeigersinn.

Drehzahl links: leicht positiv, Drehzahl rechts: leicht negativ:-→ Stationäre Drehung im Uhrzeigersinn

Ultraschallsensor Treiberschicht

Nachdem ich mit der Performance des PI Reglers zufrieden war habe ich einen Treiber für die Ultraschallsensoren implementiert. Dieser bietet zum Erzeugen und Auslesen der Entfernungsmessdaten folgende Schnittstelle:

```
void pullTrigger(); // Send Trigger Signal. According to Datasheet for 10 us
void stopTriggering(); // Stop the trigger signal
int16_t getDistance(); // Get Distance in cm
```

ROS Serial Interface

Zum einen müssen nun die über die Treiber abgerufenen Daten bezüglich Encoder-Zählstand und Entfernungsmessung der Ultraschallsensoren für den Roboter-Hardware-Interface Knoten verfügbar gemacht werden. Zum anderen ist es nötig Steuerungskommandos vom Hardware-Interface Knoten zu empfangen, um damit die DC Motoren und letztendlich den Roboter steuern zu können. Wie dies gemacht wird ist dem Programmierer überlassen und wird von ROS nicht vorgegeben. Ich habe mich für eine Kommunikation über USART entschieden. ROS stellt hier ein Framework zur Verfügung, um direkt von Mikrocontrollern über USART auf ROS Topics publishen und subscriben kann. Dazu läuft auf dem Raspberry Pi eine sogenannte Bridge-Node, welche als Gateway zwischen dem ROS Ökosystem und dem Mikrocontroller agiert. Die Funktionalität der auf dem Arduino Mega implementierten Software kann durch folgendes Flow-Chart Diagramm sehr gut beschrieben werden.

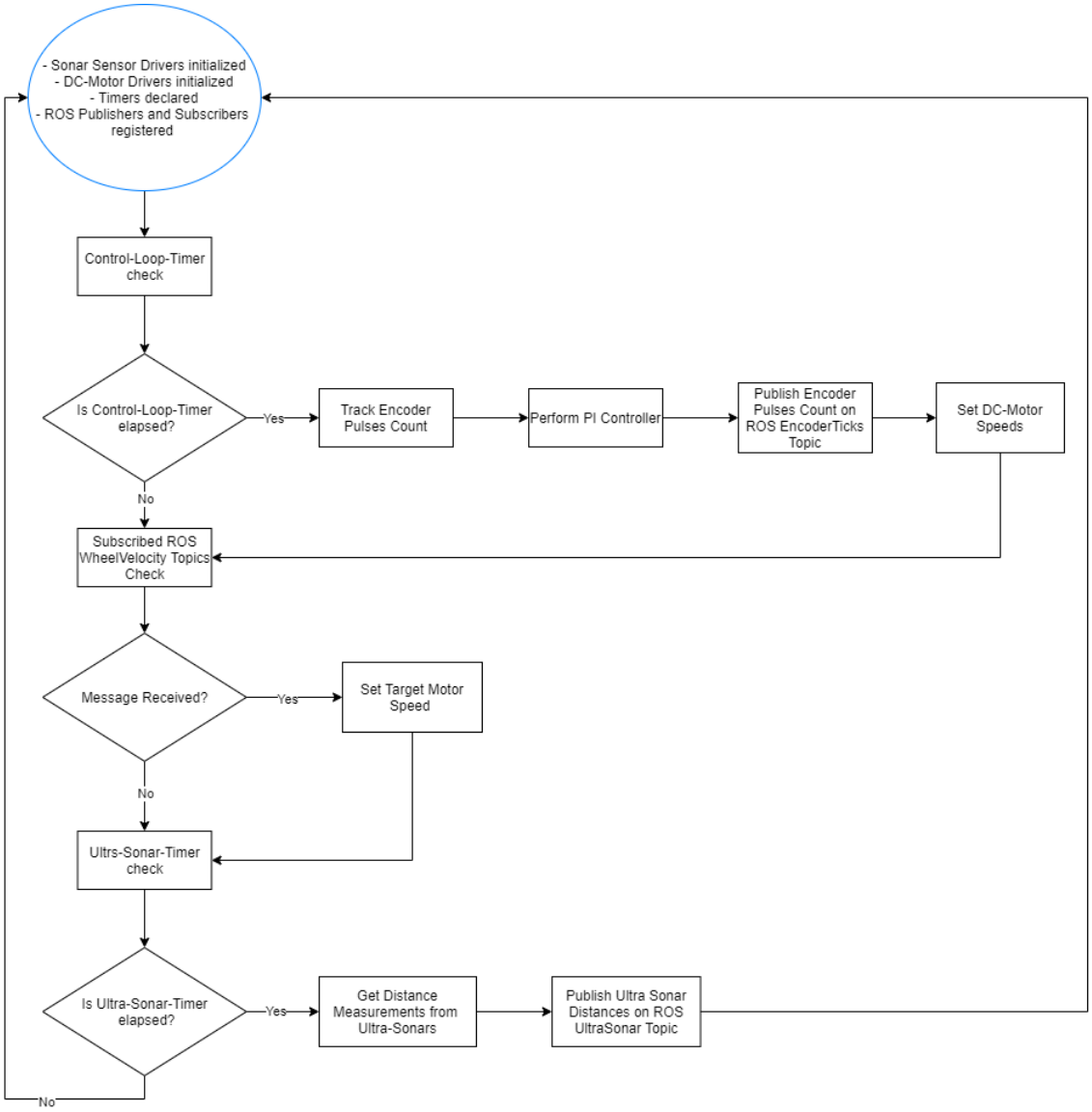

Abbildung 22: Flow Chart Arduino Mega Software

6 Ergebnisse

Nachdem der Roboter nun voll Einsatzbereit war, habe ich die SLAM und Navigationsfunktion des Roboters getestet. Es ist zu erkennen, dass die Kartenqualität etwas schlechter als in der Simulation ist. Dies liegt daran, dass in realen Bedingungen Störfaktoren, wie unterschiedliche Wandoberflächen oder Ungenauigkeiten bei der Konfiguration von z.B. gemessenem Radabstand und Raddurchmesser einen negativen Einfluss auf den Kartenerstellungsprozess haben. Die Ergebnisse können anhand der folgenden Bilder von erstellten Karten eingesehen werden und in Vergleich zur Karte eines Xiaomi Staubsaugerroboters gesetzt werden. Die navigation hat bei beiden verwendeten SLAM Verfahren sehr gut funktioniert. Es wurde immer der gleiche Bereich kartographiert:

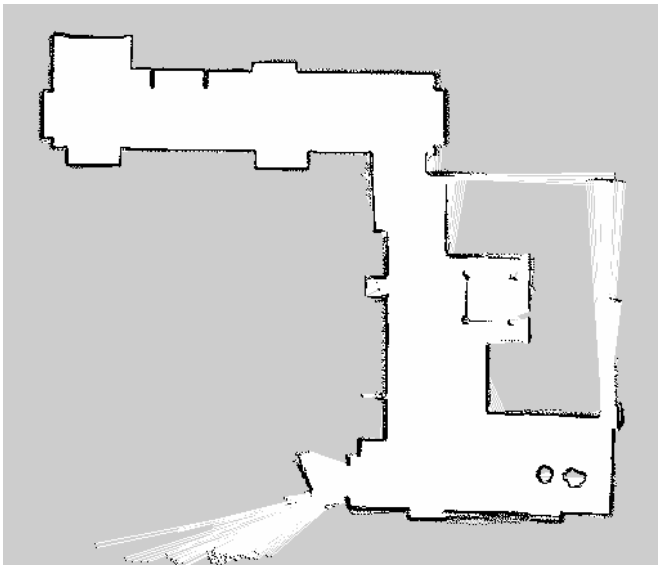

Abbildung 23: Map erstellt mit gmapping

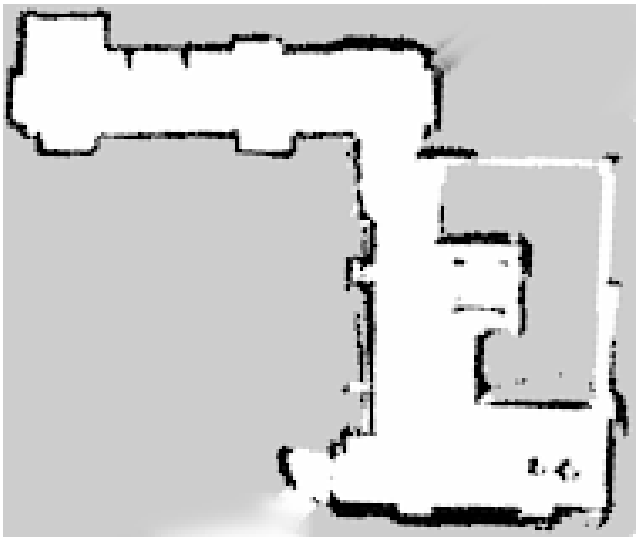

Abbildung 24: Map erstellt mit Google Cartographer

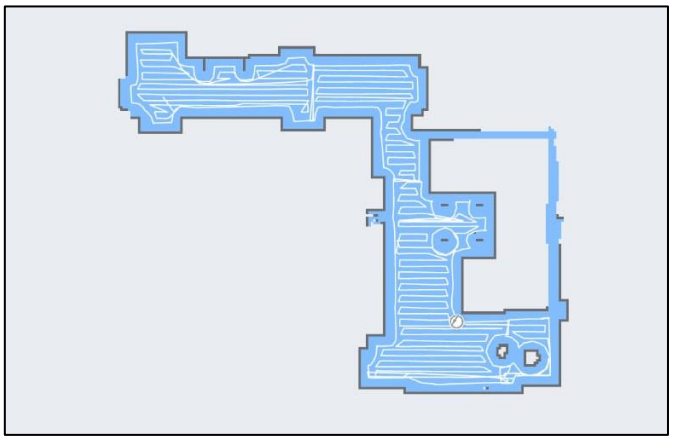

Abbildung 25: Map erstellt mit Xiaomi Staubsaugerroboter

Es ist leicht zu erkennen, dass die Karte von Google Cartographer am schlechtesten abschneidet. Dahingegen ist die Qualität der anderen beiden Karte schon ähnlich gut. Der Unterschied ist, dass bei dem Xiaomi Roboter Punkte der Kartenumrisse zu Linien verbunden werden und so alles sauberer aussieht. Durch Image-Processing könnte das gleiche wohl auch mit der Gmapping Karte erreicht werden.

Im Folgenden möchte ich noch ein Bild präsentieren, dass die Navigation des Roboters zeigt, nachdem die Karte abgespeichert wurde und dann statisch über den Map Server bereitgestellt wurde. Nun findet kein Mapping mehr statt. Die Lokalisierung des Roboters wird anhand des *amcl* Pakets durchgeführt. Hierbei haben sich einige Probleme ergeben, die auf dem folgenden Bild zu sehen sind. Es wird nicht geschafft den Roboter korrekt zu lokalisieren. In Rot sind die Laserscans eingezeichnet. Wäre die Lokalisierung erfolgreich würden diese direkt über den Kanten der Karte liegen. So kann natürlich auch keine Zielsichere Navigation durchgeführt werden.

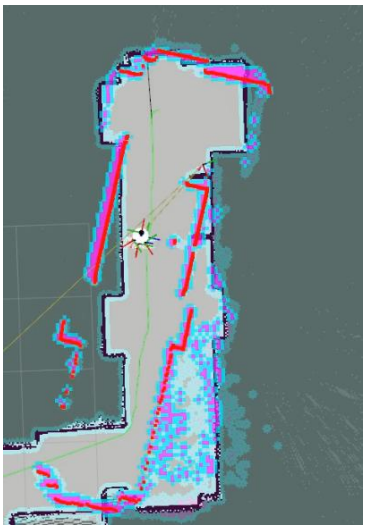

Abbildung 26: Lokalisierung via amcl

Videos von Mapping und Navigation können im Anhang **D_A5** gefunden werden.

7 Fazit

In dieser Arbeit ist es gelungen die nötigen Grundlagen der Robotik herauszuarbeiten, die als Basis für das Design eines mobilen Roboters, mit der Aufgabe der Kartenerstellung und Pfadplanung, gelten. Zudem wurden eine Einführung in das Robot Operating System geboten und dabei erläutert, wie nützlich es bei der Entwicklung von Robotersystemen sein kann. Mit diesem Wissen wurde dann erfolgreich eine Simulation des Robotersystems erstellt. Aufbauend auf die gute Simulation konnte der mobile Roboter dann sehr effizient in echt realisiert werden. Hierzu wurden zunächst alle Gehäuseteile in 3D konstruiert und mit einem 3D-Drucker ausgedruckt. Nachdem Zusammenbau des Roboters wurde die Implementierung der Treiberschichten vorgenommen und eine Schnittstelle zu ROS hergestellt. Diese Schnittstelle hat die Schnittstelle ersetzt, welche vorher von Gazebo virtuell bereitgestellt wurde. Der echte Roboter war nach wenigen Anpassungen in der Lage dazu alle Aufgaben durchzuführen, die in der Zielsetzung festgelegt wurden. Er verhielt sich sehr ähnlich wie der Roboter in der Simulation. Diese beiden Tatsachen bestätigen wie hilfreich es ist erst eine Simulation zu erstellen, bevor man den echten Roboter baut. Die Qualität der erstellten Karten ist ähnlich zu denen des verglichenen Staubsaugers. Bei der Navigation, insbesondere auf Basis von statischen Karten, gibt es noch Verbesserungspotenzial. Auch die Lokalisierung in statischen Karten mittels des verwendeten *amcl* ROS Pakets kann nicht mit der des Vergleichsgeräts mithalten. Womöglich könnte der Einsatz eines Scanners mit höherer Scanfrequenz das Ergebnis verbessern.

Aufbauend auf den jetzigen Roboter gibt es viele Themen die behandelt werden könnten. So könnten beispielsweise selbst SLAM Verfahren, Methoden zur Lokalisierung und Pfadplanungsalgorithmen implementiert oder bestehender Code verbessert werden.

8 Literaturverzeichnis

1. **Cook, Gerald.** *Mobile Robots - Navigation, Control and Remote Sensing.* 2011.

2. **mecalux.de. [Online] [Zitat vom: 10. Juli 2021.] https://www.mecalux.de/blog/autonome-mobile-roboter.**

3. mars.nasa.gov. [Online] [Zitat vom: 7. Juli 2021.] https://mars.nasa.gov/msl/home/.

4. computerwelt.at. [Online] [Zitat vom: 08. 07 2021.] https://computerwelt.at/news/service-roboter-boom-verkaufszahlen-steigen-weltweit-um-32-prozent/.

5. Thrun, Sebastian. ***Probabalistic Robotics.*** **s.l. : The MIT Press, 2005.**

6. ROS. [Online] [Zitat vom: 28. Februar 2021.] https://www.ros.org/.

7. Gazebosim. [Online] [Zitat vom: 28. Februar 2021.] http://gazebosim.org/.

8. ROS. [Online] [Zitat vom: 28. Februar 2021.] http://wiki.ros.org/rviz.

9. Siciliano, Bruno und Oussama, Khatib. ***Springer Handbook of Robotics.*** **s.l. : Springer, 2016.**

10. Oubbati, Mohamws. uni-ulm.de. [Online] [Zitat vom: 08. 07 2021.] https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.130/Mitarbeiter/oubbati/RobotikWS1113/Folien/Differentialantrieb.pdf.

11. ***3D Robotic Mapping - The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom.*** **Nüchter, Andreas. [Hrsg.] Springer-Verlag. 2009.**

12. Borenstein, J., Everett, H. R. und Feng, L. ***Where am I? Sensors and Methods for Mobile Robot Positioning.*** **[Hrsg.] The University of Michigan. 1996.**

13. slamtec.com. [Online] [Zitat vom: 10. August 2021.] http://www.slamtec.com/en/lidar/a1.

14. mikrocontroller-elektronik.de. [Online] [Zitat vom: 08. August 2021.] https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc-sr04/.

15. ***Loop closure detection for visual SLAM using PCANet features.*** **Yifan, Xia, et al. 2016. International Joint Conference on Neural Networks.**

16. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/Packages.

17. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

18. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/roscore.

19. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics.

20. *Human-robot interaction using colored LEDs and custom made hand-held device: a service robot design example.* Madunić, Ivan, et al. Split, Croatia : s.n., 2018. Conference: 3rd International Conference on Smart and Sustainable Technologies.

21. Kouba, Anis. *Robot Operating System - The Complete Reference (Volume 1).* s.l. : Springer, 2016.

22. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/Parameter%20Server.

23. ros.org. [Online] [Zitat vom: 01. August 2021.] http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file.

24. ros.org. [Online] [Zitat vom: 02. August 2021.] http://wiki.ros.org/tf.

25. ros.org. [Online] [Zitat vom: 03. August 2021.] http://wiki.ros.org/navigation/Tutorials/RobotSetup.

26. ros.org. [Online] [Zitat vom: 03. August 2021.] http://wiki.ros.org/ros_control.

27. *AMOS: comparison of scan matching approaches for self-localizationin indoor environments.* Gutmann, Steffen und Schlegel, Christian. [Hrsg.] IEEE Computer Society Press. 1996.

28. Andrade-Cetto, Juan und Sanfelio, Alberto. *Environment Learning for Indoor Mobile Robots.* s.l. : Springer Verlag, 2006.

Inhalt Digitaler Anhang (USB-Stick)

Auf dem USB-Stick befinden sich die digitale Version dieser Arbeit, der gesamte programmierte Source Code, alle von mir erstellten ROS Pakete und einige Videos die den Roboter in Aktion zeigen. Im Folgenden noch eine Liste der explizit erwähnten digitalen Anhänge.

D_A1:
Beschreibung:
Graph-Übersicht von Topics und Nodes
Speicherort:
USB:\Bachelorarbeit_Andreas_Hilmer\Digitaler_Anhang\Bilder\D_A1_rosgraph_topics_and_nodes.png

D_A2:
Beschreibung:
URDF Model des Roboters
Speicherort:
USB:\Bachelorarbeit_Andreas_Hilmer\Digitaler_Anhang\My_Diffbot_ROS_Packages\diffbot_description\urdf\diffbot.urdf

D_A3:
Beschreibung:
Transformation-Tree des Roboters
Speicherort:
USB:\Bachelorarbeit_Andreas_Hilmer\Digitaler_Anhang\Bilder\D_A3_tf_tree.png

D_A4:
Beschreibung:
3D Modelle Roboter
Speicherort: USB:\Bachelorarbeit_Andreas_Hilmer\Digitaler_Anhang\CAD

D_A5:
Beschreibung:
Videos SLAM
Speicherort: USB:\Bachelorarbeit_Andreas_Hilmer\Digitaler_Anhang\Videos