Bachelorarbeit

Andreas Löffler

Sensorische und teilautonome Navigationsunterstützung von Telepräsenzsystemen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Löffler

Sensorische und teilautonome Navigationsunterstützung von Telepräsenzsystemen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Lehmann
Zweitgutachter: Prof. Dr. Stephan Pareigis

Eingereicht am: 24. Mai 2016

Andreas Löffler

Thema der Arbeit
Sensorische und teilautonome Navigationsunterstützung von Telepräsenzsystemen

Stichworte
Navigation, Fahrerassistenzsystem, Telepräsenzsystem, Umgebungswahrnehmung, SLAM, Teilautonomie, ROS, Sick Tim310, Microsoft Kinect

Kurzzusammenfassung
Thema der Arbeit ist die Entwicklung einer teilautonomen Navigationsunterstützung für Telepräsenzsysteme. Kernpunkte, welche in der Arbeit untersucht werden, sind zum einen, welche Sensoren sich für die Umgebungswahrnehmung im Indoor-Bereich eignen und zum andren wie an Hand dieser Sensoren in unbekannter Umgebung navigiert werden kann. Hierfür kommen ein Laserscanner der Firma Sick, sowie die Microsoft Kinect zum Einsatz. Es wird auf Basis von ROS der vorhandene Teleoperator so erweitert, dass anschließend eine Navigation zu einem vom Operator gewählten Punkt möglich ist. Es wird über potentielle Störgrößen, sowie in der Sensorkombination aufgetretene Fehler diskutiert.

Andreas Löffler

Title of the paper
Sensory and semi-autonomous navigation assistance System of telepresence robot systems

Keywords
navigation, driver assistance system, telepresence robot system, perception of environment, SLAM, semi-autonomous, ROS, Sick Tim310, Microsoft Kinect

Abstract
Topic of this paper is to develop a semi-autonomous navigation assistance system of telepresence robot systems. Key points of this work are the investigation of sensors used for the perception of the indoor environment and the navigation of these sensors in an unkown environment. Therefore a laserscanner from Sick's company and the Microsoft Kinect will be used. The existing teleoperator, based on ROS, will be extended with the result that a navigation to a user selected point is possible. Potential disturbances and errors caused by the combination of sensors will be discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Durch den technischen Fortschritt ist es heutzutage möglich, über ein Telepräsenzsystem an einem entfernten Ort präsent zu sein und dort mit anderen Menschen oder Objekten zu interagieren. Die Navigation an solchen Orten gestaltet sich allerdings teilweise als schwierig, da sich der Benutzer des Telepräsenzsystems die gesamte entfernte Umgebung einprägen muss. Eine Unterstützung für den Benutzer in Form einer Karte der schon besuchten Positionen würde die Navigation in der Ferne um einiges erleichtern. Eine Erweiterung der Unterstützung wäre eine Navigation zu einem Punkt im Sichtbereich oder zu einem Punkt, der zu einem früheren Zeitpunkt im Sichtbereich gewesen ist. Eine solche Navigationsunterstützung kann auf autonomer oder teilautonomer Basis erfolgen. Der Komfort des Benutzers wird erhöht. Außerdem ist es ihm auf diesem Weg möglich gleichzeitig weitere Tätigkeiten auszuführen, wie z.B. der Interaktion mit seinem Umfeld.

1.1. Motivation

Aus vorhergehenden Arbeiten (siehe [87] und [33]) stand das Telepräsenzsystem MoRy-A zur Verfügung, das über ein Head-Mounted-Display (HMD)[1] und ein Head-Tracking-System[2] gesteuert werden kann. Für das Telepräsenzsystem wird in einer parallel geschriebenen Arbeit ein „Leitstand für mobile Systeme mit Virtual Reality“(siehe [35]) entwickelt. Problematisch ist jedoch, den Roboter in gewissen Bereichen zu steuern bzw. sich in unbekannten Bereichen zurecht zu finden. Des Weiteren gibt es für den Benutzer keine Hilfe bei der Kollisionsvermeidung. Über die HMD muss versucht werden, Entfernungen und Engpässe richtig abzuschätzen, um nicht mit Hindernissen in der entfernten Umgebung zu kollidieren. Es stand somit der Gedanke im Raum, das System um Funktionen zu erweitern, die dem Benutzer in dieser Hinsicht mehr Komfort liefern und gleichzeitig die Sicherheit des Roboters erhöhen.

[1] Ein Head-Mounted-Display wird auf dem Kopf getragen und soll dem Träger vermitteln, sich in einer gezeigten Szene zu befinden.

[2] Ein Head-Tracking-System erfasst die Position und die Bewegung des Kopfes

1.2. Problemstellung und Anforderung

Ziel dieser Arbeit ist es, das Problem der teilautonomen Navigation in unbekannter Umgebung für den Benutzer eines Telepräsenzsystems zu erleichtern. Das Telepräsenzsystem der Hochschule für Angewandte Wissenschaften (HAW) Hamburg soll dahingehend erweitert werden, dass dem Benutzer der Nachteil der unzureichenden Wahrnehmung der Umgebung in der Ferne genommen wird. Hierfür soll eine Navigationsunterstützung auf der vom Benutzer gesteuerten Plattform (im Folgenden „mobile Einheit“oder „Roboter“genannt) entwickelt werden. Die Plattform des Telepräsenzsystems MoRy-A ist um entsprechende Sensoren zu erweitern, die für eine verbesserte Wahrnehmung geeignet sind. Zu untersuchen gilt, ob die Sensoren miteinander vereint werden können. Da die teilautonome Navigation durch eine unbekannte Umgebung einen großen Bereich an Anforderungen mit sich bringt, werden die Anforderungen für diese Arbeit auf die Örtlichkeiten der HAW Hamburg beschränkt.
Die mobile Einheit wird von einem Benutzer über eine Bedienerstation gesteuert. Der Roboter generiert, während er fährt bzw. gefahren wird, eine Karte und lokalisiert sich in dieser. Der Benutzer kann sich über eine geeignete Schnittstelle die Karte anzeigen lassen und einen Punkt aus dieser Karte wählen. Die Navigationsunterstützung berechnet einen möglichen Pfad zu diesem Punkt. Nach der Freigabe vom Benutzer für eine autonome Fahrt, übernimmt der Roboter die Steuerung der Bewegung nach links oder rechts und die Kollisionsvermeidung. Der Benutzer bestimmt die Geschwindigkeit, mit der gefahren wird. Am Zielpunkt angekommen, bleibt die mobile Einheit selbstständig stehen. Liegt der vom Bediener gewählte Punkt auf einem Hindernis, soll der Roboter bei Erkennung des Hindernisses vor diesem stehen bleiben. Hindernisse können alle möglichen Gegenstände sein, die in einer Büro ähnlichen Umgebung wieder zu finden sind. Typische Beispiele hierfür sind Tische und Stühle, Schränke, sowie Türen und Menschen. Der Roboter soll sich in einer ebenen Umgebung bewegen, ohne Stockwerke bzw. Treppen überwinden zu müssen. Nur frei zugängliche Örtlichkeiten sollen erreicht werden. Geschlossene oder versperrte Wege sollen nicht geöffnet oder überwunden werden. Kommt es zu Notsituationen an der mobilen Einheit, soll diese über eine Noteinrichtung an der Einheit selbst, angehalten werden können. Die Einheit soll zusätzlich stehen bleiben, wenn Teile der Hardware zur Erkennung der Umgebung ausfallen, um ein „*Blindfahren*“zu vermeiden.

2. Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen vermittelt, um das in 1.2 genannte Problem zu lösen. Es wird ein kurzer Einblick in die relevanten Teilgebiete gegeben. Anschließend wird auf das Hauptproblem Navigation eingegangen. Hierbei soll geklärt werden, was Navigation im Allgemeinen bedeutet, sowie wie in unbekannter Umgebung ein Ziel erreicht werden kann.

2.1. Telepräsenzsystem

Der Einsatz eines Telepräsenzsystems ermöglicht es, sich in einer Umgebung nicht physisch aufzuhalten und dennoch mit dieser zu interagieren [41][77]. Das Wort Telepräsenzsystem bedeutet, zerlegt in seine Einzelteile *Tele*, *Präsenz* und *System* nichts anderes, als durch ein System in der Ferne (*Tele*) anwesend (*Präsenz*) zu sein [15]. Der übliche Aufbau eines solchen Systems lässt sich in vier Teile unterteilen:

1. Teleoperator
2. Operator
3. Bedienerstation
4. Kommunikationskanal

Der Teleoperator ist das ausführende technische System [63]. Üblicherweise ist ein Teleoperator, der nach den Anforderungen aus 1.2 arbeitet, folgendermaßen aufgebaut. Die Basis für den Aufbau ist eine mobile Einheit, die elektronisch angesteuert werden kann. An der mobilen Einheit sind Sensoren montiert, die es ermöglichen, die Umgebung wahrzunehmen oder mit ihr zu interagieren. Die verschiedenen Sensortypen und deren Einsatzgebiete werden in 2.4 beschrieben. Um Steuersignale empfangen und verarbeiten zu können, ist auf der Basis ein Computer angebracht, der über den Kommunikationskanal mit der Bedienerstation verbunden ist.

Als Operator wird der agierende Mensch (Benutzer) bezeichnet [2][8][17]. Er bildet den zentralen Bestandteil des Teleoperationssystems. Der Teleoperator wird von dem Operator gesteuert und dient als Repräsentationsobjekt in der Ferne.

Die Bedienerstation gibt dem Operator das Gefühl der Telepräsenz und dient außerdem zur Steuerung des Teleoperators [23]. Sie ist fest an einem Ort aufgestellt. Grundlegend besteht die Bedienerstation aus einem herkömmlichen Computer, Eingabegeräten, wie Tastatur oder Gamepad, Lautsprechern und einer Virtual Reality (VR)-Brille. Eine VR-Brille ist ein HMD, das üblicherweise mit einem Head-Tracking-System verknüpft ist. VR-Brille und Lautsprecher vermitteln dem Operator die Fernsinne der visuellen und auditiven Wahrnehmung. Für die Nahsinne, wie Tastsinn, kann über ein Eingabegerät haptisches Feedback gesendet werden.

Der Kommunikationskanal verbindet Teleoperator und Operator [63]. Meistens sind Bedienerstation und Teleoperator über eine kabellose Verbindungsform, wie zum Beispiel Wireless LAN, verbunden. Die Bedienerstation ist mit ihren Komponenten fest verkabelt.

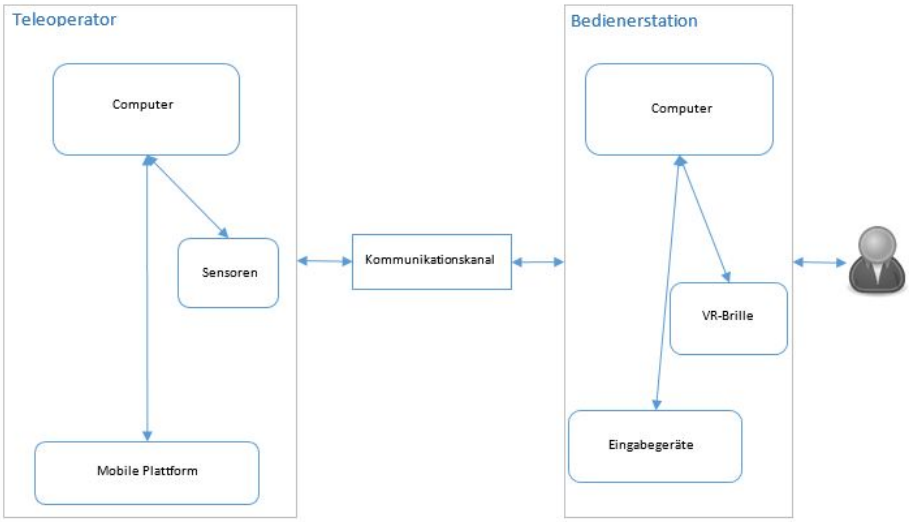

Abbildung 2.1.: Allgemeiner Systemaufbau eines Telepräsenzsystems.

Die Einsatzmöglichkeiten eines Telepräsenzsystems sind vielfältig. Zum einen kann ein Telepräsenzsystem in Bereichen eingesetzt werden, in denen es von Vorteil ist, unbemannt vorzugehen [80]. Als Beispiel wären hier aus dem militärischen Bereich die Minenräumung

oder die Aufklärung von gefährdeten Orten zu nennen. Zum anderen kann ein Einsatz eines Telepräsenzsystems im Arbeitsumfeld erfolgen [80]. Im medizinischen Bereich könnten Ärzte über den Teleoperator mit Patienten, die unter besonderer Beobachtung stehen, in Kontakt treten [84][10][80]. In Firmen können Konferenzen trotz örtlicher Abwesenheit von Personal durchgeführt werden[80][10]. Abbildung 2.2 zeigt einen der heute gängigen Teleoperatoren für

Abbildung 2.2.: Teleoperator Double 2 [13].

das Arbeitsumfeld. Der Double 2 ist aufgrund fehlender Aktoren sehr eingeschränkt und kann daher nur in einem Tätigkeitsfeld eingesetzt werden, indem es lediglich der Kommunikation bedarf. Das Double 2 Full Set mit allem Zubehör, was für die Telepräsenz benötigt wird, ist ab einem Preis von umgerechnet 2630 Euro erhältlich [14].

2.2. Fahrerassistenzsysteme

Im Folgenden wird der Begriff Fahrerassistenzsystem erläutert und beschrieben, wie ein Benutzer eines Telepräsenzsystems von einem solchen System unterstützt werden kann.
Ein Fahrerassistenzsystem ist ein System, das dem Fahrer bzw. dem Benutzer die Ausführung einer Aktion erleichtern soll [12]. Es besteht aus Hard- und Softwarekomponenten. Typischerweise werden Fahrerassistenzsysteme in der Automobilbranche eingesetzt, um dort die Sicherheit sowohl für den Fahrer als auch für die Umwelt zu erhöhen [12]. Ein klassisches Beispiel hierfür ist der Notbremsassistent. Der Assistent überwacht während der Fahrt permanent mit Hilfe von Sensoren die Umgebung vor dem Fahrzeug [3]. Bei Situationen, die von dem Assistenten als Notsituationen eingestuft werden, wie z.B. zu schnelles Auffahren auf ein Stauende, wird eine Notbremsung eingeleitet, die den Fahrer vor einem Unfall schützen

soll. In Bezug auf die Problemstellung dieser Arbeit kann der Benutzer eines Telepräsenzsystems dahingehend unterstützt werden, dass die Wegfindung zu einem von dem Benutzer bestimmten Punkt durch den Assistenten bestimmt wird. Der Benutzer muss lediglich den vorher bestimmten Weg abfahren. Kollisionsvermeidung und Lenkung wird vom Assistenten übernommen. Der Mehrwert eines solchen Assistenten ist zum einen, dass der Benutzer den Weg nicht kennen muss, sondern nur einen Punkt auf einer von dem Teleoperator erstellten Karte angeben muss. Zum anderen kann durch solche Assistenten Zeit gewonnen werden, da Umwege eingespart werden. Es wird also die Effizienz, die Sicherheit der mobilen Einheit durch Kollisionsvermeidung und der Komfort für den Benutzer erhört [42].

2.3. Autonomie

Aus den Anforderungen (Kapitel 1.2) geht hervor, dass teilautonom navigiert werden soll. Hierzu ist es notwendig, den Begriff der *Autonomie* in Bezug auf das Handeln der mobilen Einheit genauer zu definieren und zu erläutern. Der Begriff *autonom* bzw. *Autonomie* lässt sich in zwei Teile gliedern: Die mobile Einheit könnte sich entweder voll autonom oder teilautonom durch den Raum bewegen. Von *voll autonomen Handeln* ist dann die Rede, wenn die Verantwortung des Handelns bei der mobilen Einheit liegt. Das bedeutet im Hinblick auf die Anforderungen, dass die mobile Einheit völlig selbstständig über den zu fahrenden Weg sowie die Geschwindigkeit und die Richtung bestimmt [31]. Voraussetzung dafür ist, dass der Benutzer vorher einen Punkt X bestimmt hat, zu dem gefahren werden soll. Hierbei ist zu beachten, dass alle Gefahrenpotentiale vorher erkannt und im Ernstfall umgangen werden müssen. Von *teilautonomen Handeln* wird dagegen gesprochen, wenn die Verantwortung trotz teilweiser autonomer Handlungen bei dem Benutzer liegt [45]. Dieser kann zu jedem Zeitpunkt die volle Kontrolle über die mobile Einheit erlangen. In einem Anwendungsfall bedeutet dies, dass der Benutzer einen Punkt X bestimmt, die mobile Einheit einen Weg zu dem gewählten Punkt bestimmt und anschließend die Bewegung nach links und rechts übernimmt. Der Benutzer hat in diesem Fall die Kontrolle über die Geschwindigkeit. Tritt ein unerwartetes Ereignis ein, kann der Benutzer die seitlichen Bewegungskommandos der Teilautonmie übersteuern und somit die komplette Kontrolle über die mobile Einheit zurück erlangen. Die Verantwortung für die mobile Einheit liegt damit bei dem Benutzer. Ein aktuelles Beispiel aus der Industrie ist der "Freightliner Inspiration Truck"[51]. Der LKW ist mit autonomen Steuerungssystemen ausgerüstet, welche es dem Fahrer ermöglichen, während der Fahrt anderen Dingen nachzugehen und sich nicht mehr auf die Straße konzentrieren zu müssen. Der LKW steuert selbstständig mit Hilfe von Radarsystemen, Stereokamera, Spurhalteassistent und Abstandsregeltempomat

durch den Straßenverkehr. Stößt das System durch Störungen wie schlechte Sicht oder unübersichtliche Streckenführung an seine Grenzen, wird der Fahrer aufgefordert, die Kontrolle zu übernehmen. Aufgrund der Tatsache, dass der Fahrer zu jeder Zeit die Kontrolle über sein Fahrzeug durch Übersteuern oder Ausschalten des Steuerungssystem erlangen kann, ist das System von Mercedes-Benz als teilautonom einzuordnen [71]. Zusammenfassend ist für die zwei Teile des Begriffs *Autonomie* zu sagen, dass es Unterschiede in der Verantwortung und somit in Safety- und Securityanforderung an ein solches System gibt.

2.4. Navigation

Der Begriff der Navigation hat seinen Ursprung in der Schifffahrt. Laut Definition ist *Navigation* die Bestimmung des aktuellen Standorts und Kurses von Schiffen [58]. In der heutigen Zeit wird der Begriff jedoch nicht mehr nur in der Schifffahrt verwendet, sondern ist in allen Bereichen wiederzufinden, in denen Objekte von Motoren durch den Raum bewegt werden. In jedem dieser Bereiche sind die Grundfragen der Navigation jedoch gleich. Es ist immer die Frage nach *Wo befindet sich das sich bewegende Objekt?*, bezogen auf diese Arbeit lautet die Frage *Wo befindet sich die mobile Einheit?*, *Wo ist der Zielort zu dem sich bewegt werden soll?* und *Wie soll der Zielort erreicht werden?* [82][46][44]. Das *Wie* beinhaltet die Möglichkeiten, bestimmte Voraussetzungen zu erfüllen, wie z.B. den energieeffizientesten oder schnellsten Weg zu wählen. Navigation besteht aus der Wegplanung und der eigentlichen Fahrt [16]. Die Wegplanung beinhaltet die Selbstlokalisierung und die Planung eines möglichen Weges unter gegebenen Voraussetzungen zu einem bestimmten Zielort. Die Fahrt hingegen beinhaltet das Abfahren der zuvor berechneten Strecke unter Einhaltung der Safteyanforderungen des zugrunde liegenden Systems und der physikalischen Möglichkeiten, wie z.B. maximale Kurvengeschwindigkeit, Höhen oder Breiten. Eine große Rolle bei der Wegplanung und der eigentlichen Fahrt spielt die Umgebung. Diese kann in unbekannte und bekannte Umgebung typisiert werden [16]. Die Typen werden jeweils in statisch und dynamisch unterteilt. Der Unterschied besteht darin, dass sich in der statischen Umgebung „Nur der Zustand des Roboters innerhalb der Umgebung ändert (...)“[82] und in der dynamischen Umgebung Objekte ihre Lage oder ihren Zustand verändern können [82]. Für jeden Typ kann baumartig weiter unterteilt werden in *Indoor* und *Outdoor*, gefolgt von strukturierter oder unstrukturierter Umgebung. Strukturierte Umgebung zeichnet sich durch einfache, meist rechtwinklige Strukturen aus, wie z.B. Tische, Türen oder Stühle. Als unstrukturierte Umgebung werden hingegen komplexe Strukturen wie z.B. Sträucher bezeichnet [16].
Damit die mobile Einheit in der Umgebung navigieren und Objekte wahrnehmen kann, kön-

nen verschiedene Sensoren eingesetzt werden. Für diese Arbeit sind lediglich Sensoren für den Indoor-Bereich (siehe 1.2) von Nutzen. Daher werden im Folgenden für heute typische Sensorarten, die im Indoor-Bereich verwendet werden, aufgeführt. Die Sensorarten werden hinsichtlich Einsatzwecks und ihrer Vor- und Nachteile kurz beschrieben. Anschließend wird ein Verfahren vorgestellt, mit dem es möglich ist, eine Selbstlokalisierung und gleichzeitiges Kartieren einer unbekannten Umgebung durchzuführen. Auf Basis dieses Verfahren werden dann Wegfindungsalgorithmen vorgestellt, die es ermöglichen, die kürzesten Wege in einer erstellten Karte zu finden.

2.4.1. Laserscanner

Ein Laserscanner ist ein auf Laserstrahlung basierter Sensor, mit dem es möglich ist, Entfernungen zu messen und Oberflächen zu detektieren [43][50][85][59]. Ein Scanvorgang läuft im Allgemeinen erklärt folgendermaßen ab:
Es wird ein Laserstrahl entsendet. Der Strahl trifft in seiner Umgebung auf ein Objekt bzw. auf eine Oberfläche und wird reflektiert. Die reflektierte Strahlung wird von einer Empfangsoptik aufgenommen und ausgewertet. Das Impulsmessverfahren und das Phasenvergleichsmessverfahren stellen für die Auswertung zwei verschiedene Messverfahren dar, nach denen auch die Sensoren klassifiziert werden[43]. Das Impulsmessverfahren beruht auf dem Aussenden von Lichtimpulsen, die auf einer Trägerwelle aufmodelliert sind. Gemessen wird die Zeit zwischen dem Entsenden und dem Empfangen der Impulse[43]. Bei dem Phasenvergleichsmessverfahren hingegen werden mehrere Lichtwellen unterschiedlicher Wellenlänge ausgesendet. Die Lichtwellen werden miteinander verglichen. Über die Phasenverschiebung kann die zurückgelegte Wegstrecke ermittelt werden [43]. Die gemessenen Umgebungsdaten werden üblicherweise in m zurückgegeben. Der Sichtwinkel von Industrie-Laserscannern ist je nach Preisklasse zwischen 1 ° - 270 ° frei einstellbar und hat eine Auflösung zwischen 0.25 ° - 1.0 °. Hier liegen auch die Stärken des Sensors. Der Laserscanner ist eine präzise und effiziente Möglichkeit, die Umgebung zu vermessen. Schwierigkeiten hingegen sind durchsichtige Oberflächen bzw. Oberflächen, die nicht, oder nur minimal reflektieren. Diese können von einem Laserscanner nicht erkannt werden. Ein weiterer negativer Punkt sind die hohen Anschaffungskosten eines Laserscanners. Diese liegen oftmals bei mehreren tausend Euro pro Sensor.

2.4.2. Infrarotsensoren

Infrarot (IR)-Sensoren werden in der Robotik meist zur Objektdetektion genutzt [59][24]. Das Prinzip von Infrarotsensoren ist ähnlich dem des Laserscanners. Es wird ein Infrarotsignal

ausgesendet und die Reflexion des Signals aufgenommen und ausgewertet. Um das ausgesendete Infrarotsignal von anderen Infrarotquellen unterscheiden zu können, wird das Signal meist mit einer niedrigen Frequenz entsendet[59]. Nachteile eines IR-Sensors sind, dass nicht mit Sicherheit gesagt werden kann, dass sich bei ausbleibender Reflexion kein Objekt vor dem Sensor befindet. Des Weiteren können weder durchsichtige noch schwarze Objekte detektiert werden, da hier ebenfalls die Reflexion ausbleibt oder zu schwach ist [59].

2.4.3. Bildsensoren

Mit einem Bildsensor wird ein Bild der momentanen Situation aufgenommen. Er liefert die benötigten Daten, um ein Bild so darstellen zu können, wie es die menschlichen Augen zur gleichen Zeit wahrnehmen. Die heutige Technik der Bildsensoren ist die Complementary metal-oxide-semiconductor (CMOS) Technik. Da für diese Arbeit die genaue Funktionsweise der CMOS-Technik irrelevant ist, kann sie bei Bedarf unter [26] nachgelesen werden. Das Paper beschreibt die Anforderungen für ein CMOS Bild und die historische Entwicklung von Charge-coupled device (CCD) zu CMOS. Über eine Bildverarbeitung können aus den vom Sensor erhaltenen Bilddaten Aussagen bezüglich Objekten vor dem Sensor getroffen werden [64]. Es ist über eine Bildverarbeitung möglich, Objekte zu erkennen und zu identifizieren [64]. Über die Identifikation von Objekten kann anschließend dorthin navigiert werden, falls dies gefordert wäre. Ein Nachteil der Bildverarbeitung ist die hohe Rechenintensität, die Performanceeinbußen mit sich bringt.

2.4.4. Sonar

Sound navigation and ranging[1] bildet das Akronym Sonar. Sonar ist eine Messtechnik zur Bestimmung von Entfernungen zu Objekten im Raum [32][16]. Sie beruht auf der Messung von Signallaufzeiten. Es wird ein kurzer, für den Menschen nicht wahrnehmbarer, Schallimpuls (Ultraschall) ausgestoßen, der von Objekten im Raum reflektiert wird. Über einen Empfänger wird der zurück geworfene Schall aufgenommen. Die Zeit, die der Schall braucht, um vom Sender über das Objekt zum Empfänger zu gelangen und die Schallgeschwindigkeit ermöglichen es, die Entfernung zu einem Objekt im Raum zu bestimmen. Im Bezug auf die Wahrnehmung von Objekten in der Umgebung stellt der Sonar-Sensor eine günstige und einfach zu verwendende Möglichkeit dar. Nachteil des Sensors ist jedoch, dass aufgrund von Überlagerung der Schallwellen die gemessene Entfernung sehr unsicher ist und somit eine genaue Bestimmung der Richtung, in der sich das Objekt befindet, schwer möglich ist [55][16][27]. Des Weiteren

[1] Schallnavigation und Entfernungsmessung

ist die Form des Objekts, das erkannt werden soll, von großer Bedeutung. Es kann passieren, dass der Schall in einem ungünstigen Winkel auf das Objekt trifft (siehe Abbildung 2.3) und somit die Reflexion den Empfänger nicht erreichen kann[27].

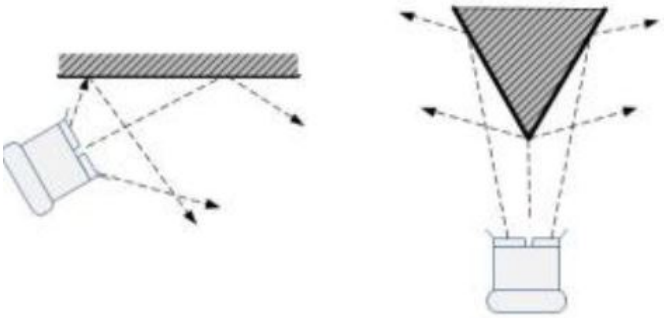

Abbildung 2.3.: Negativbeispiele für Sonar-Sensoren [27].

2.4.5. Kontaktsensoren

„*In Zeiten, wo der Alltag immer hektischer wird und man kaum Zeit für sich selbst findet, ist es eine Wohltat, wenn man nicht auch noch alles im Haushalt tun muss*“. Diese Aussage ist die Quintessenz aus der heutzutage gezeigten Saugroboterwerbung [36]. Saugroboter sollen den Alltag der Menschen erleichtern, indem sie den Wohnraum von Staub und losem Dreck befreien [36]. Hierzu fahren sie autonom und meist in Abwesenheit der Besitzer durch die Räume der Wohnung. Um sich im autonomen Betrieb in unbekannten Terrain zurecht zu finden, sind Saugroboter neben Algorithmen für das effiziente Abfahren eines Raumes auch mit Kontaktsensoren ausgestattet. Kontaktsensoren stellen den physikalischen Kontakt mit einem Objekt fest [59]. Einfach beschrieben, wird nur eine Zustandsveränderung gemessen. Meist kommen sogenannte *Bumper* zum Einsatz. Diese Kontaktsensoren funktionieren wie ein Taster. Dabei gibt es die Varianten, die entweder bei Kontakt direkt einen Stromkreis unterbrechen, oder über ein Durchbrechen einer Lichtschranke innerhalb des Sensors ein Signal auslösen [59]. Eine weitere Art von denkbaren Kontaktsensoren sind diverse biegsame Trägermaterialien, die mit Dehnungsmessstreifen ausgestattet sind [59]. Ein Dehnungsmessstreifen ist mit einer dünnen Widerstandsschicht beschichtet, die ihren Widerstand je nach Dehnung oder Beugung verändert und so Rückschlüsse auf ein Hindernis zulässt. Nachteil dieser kostengünstigen Sensoren zur Hinderniserkennung ist, dass das Signal des Erkennens eines Hindernisses erst am Hindernis selbst geschieht. Es ist somit nicht möglich, mit schnelleren Geschwindigkeiten

zu agieren, wenn der Roboter nicht mit entsprechender Pufferzone ausgestattet ist. Es könnte also zu Schäden am Roboter führen. Zusätzlich könnten bewegliche Hindernisse umgestoßen und dabei ebenfalls zerstört werden.

2.4.6. Odometrie

Eine der am meisten genutzten Techniken, um die Position eines Roboters herauszufinden, ist die Odometrie [16]. Mit Encodern, die an den Rädern des Roboters befestigt sind, wird die Drehbewegung jedes Rades gemessen. Es können dadurch Aussagen getroffen werden, wie weit das Rad den Roboter in eine bestimmte Richtung bewegt hat. Die Odometrie kann für primitive Navigation verwendet werden. Diese Form der Navigation wird als Koppelnavigation (engl. Dead Reckoning) bezeichnet [16]. Sie berechnet aus der alten bekannten Position des Roboters und der Ticks, die bei jeder vollen Radumdrehung an den Encoder gegeben werden, die neue Position. Vorteil dieser Methode ist, dass „(...) Odometrie-Sensoren vollkommen unabhängig (...)“[16] sind und eine schnelle und einfache Positionsschätzung für einen Roboter zu lassen. Ein großer Nachteil hingegen ist, dass sich Messfehler immer weiter aufsummieren, wenn diese nicht korrigiert werden [16][6]. Fehler entstehen beispielsweise durch Schlupf[2] oder durch Abweichungen von gemessenen Kenngrößen der Realität. Der Fehler und die damit immer weiter abweichende Position des Roboters kann jedoch durch Einbeziehung mehrerer anderer Sensoren korrigiert werden. Beispielsweise kann ein Laserscanner oder ein Bildsensor markante Objekte (z.B. einen Tisch) in der Umgebung identifizieren und seine Entfernung zu diesem Objekt mit in die Odometriedaten einfließen lassen (siehe 2.5).

2.4.7. Künstliche Markierungen

Künstliche Markierungen werden in die Umgebung eingebracht, um die Orientierung und somit die Navigation für einen Roboter zu vereinfachen [88][81]. Es gibt passive und aktive Markierungen. Passive Markierungen sind dem Roboter teilweise bekannte Muster, die an Wänden oder am Boden angebracht werden. Dies können zum Beispiel bestimmte Quick-Response (QR)-Codes sein, die während der Fahrt gescannt und ausgewertet werden. Eine andere Möglichkeit, die auch bei Einsteigermodellen der Robotik wieder zu finden ist, sind Linien. Diese werden auf dem Boden aufgeklebt und können mit Hilfe von Sensoren abgefahren werden. Eine weitere passive Markierung, die dem Roboter nicht bekannt sein muss, stellt der Reflektor dar [72]. Die Reflektoren werden an der Wand angebracht. Über Lichtsensoren kann der Roboter zuvor ausgestrahltes Licht erkennen und somit seine Position bestimmen. Außer-

[2] Schlupf - Als Schlupf bezeichnet man das Durchdrehen eines oder mehrere Räder

dem ist der Roboter in der Lage, bestimmte Befehle auszuführen, wie z.B. nach dem dritten erkannten Reflektor links abzubiegen. Ein Vorteil der passiven künstlichen Markierung ist, dass sie nicht mit Energie versorgt werden muss. Aktive Markierungen hingegen müssen mit Energie versorgt werden, bieten aber zeitgleich auch mehr Vorteile. Aktive Markierungen sind Infrarotbaken, Transponder oder induktive Begrenzungsschleifen [72][88]. Infrarotbaken senden ein Infrarotlicht, das vom Roboter empfangen und ausgewertet werden kann. Dabei können mehrere Baken unterschiedliche Trägerfrequenzen besitzen, die die Bestimmung der Position im Raum vereinfachen. Die Position kann außerdem über Transponder bestimmt werden [16]. Transponder können einzeln abgefragt werden. Dazu wird ein kurzer Ultraschallimpuls von dem Roboter erzeugt. Sobald ein Transponder den Impuls registriert, antwortet er mit einem oder mehreren Lichtblitzen. Über die verstrichene Zeit und die Empfangsrichtung kann die Position eines Roboters bestimmt werden [88]. Vorteile gegenüber den passiven Markierungen ist die Genauigkeit, bei der Bestimmung der Position und die Flexibilität des Roboters. Transponder können z.B. aus jeder beliebigen Richtung angesprochen werden. QR-Codes, die an der Wand angebracht werden, müssen dagegen erst gefunden werden. Aufgeklebte Linien auf dem Boden lassen nur eine bestimmte Route zu, die gefahren werden kann. Durch Infrarotbaken oder Transponder kann der Raum besser ausgenutzt werden.

2.4.8. SLAM

Simultane Lokalisierung und Kartierung (SLAM, Simultaneous Localization and Mapping) bildet die zentrale Einheit für die Navigation in unbekannter Umgebung. Hier wird das Problem der gleichzeitigen Kartierung der Umgebung und das Zurechtfinden in dieser gelöst [79]. SLAM ist kein Algorithmus, sondern viel mehr ein bestimmtes Konzept nach dem vorgegangen wird, um die genannte Problemstellung zu lösen [66]. Für das Konzept gibt es wiederum mehrere Verfahren, von denen im späteren Verlauf dieses Unterkapitels zwei vorgestellt werden. Die Grundvoraussetzung aller Verfahren bilden jedoch die Odometrie (2.4.6) und die Landmarken [66][79]. Landmarken sind markante Objekte in der Umgebung, die einfach wieder gefunden werden können. Sie dienen als Anhaltspunkt für die Selbstlokalisierung und können mit Hilfe der Daten von Entfernungsmessgeräten, wie zum Beispiel einem Laserscanner (2.4.1), bestimmt werden. Kriterien für geeignete Landmarken sind: einfache Wiedererkennbarkeit, Unterscheidbarkeit, häufig in der Umgebung zu finden und stationär [66]. Die Extraktion von Landmarken aus Sensordaten hängt von der Art der Landmarken und den verwendeten Sensoren ab. Hier kann zum Beispiel ein Laserscanner verwendet werden, der die Messdaten aus der horizontalen Ebene liefert, in der sich ein markanter Punkt befindet. Landmarken im Indoor-Bereich sind beispielsweise Tisch(-beine) oder Türrahmen. Um den markanten Punkt

zu extrahieren, wird in den Messdaten nach sprunghaften Veränderungen gesucht. Wird eine solche Veränderung gefunden, wird dieser Punkt aus den Messdaten als Landmarke definiert [66].
Wie oben beschrieben, soll mit dem Konzept SLAM eine Karte von der Umgebung erstellt werden, in der sich der Roboter gleichzeitig auch lokalisieren soll. Die Kartierung erfolgt zu Anfang recht einfach. Es wird aus den Messdaten des Laserscanners eine Karte produziert. Diese soll simultan zur Bewegung des Roboters erweitert werden. Dazu ist es notwendig, dass sich der Roboter innerhalb der Karte jederzeit wiederfinden kann, um die Karte entsprechend zu erweitern. Zum einen werden dafür Odometriedaten verwendet und zum anderen Landmarken. Probleme bei der Lokalisierung sind jedoch die Daten-Assoziation, also das Erkennen und Unterscheiden von alten und neuen Landmarken, sowie Fehler, die über ungenaue Messungen der Entfernungssensoren oder die Odometrie entstehen können [66]. Odometriefehler können beispielsweise Drift oder Schlupf sein. Mit Drift ist hierbei das Rutschen des Roboters über den Untergrund bei sich nicht drehenden Rädern gemeint. Schlupf ist das Durchdrehen der Räder oder eines Rades. Beide Fehler führen dazu, dass der Roboter fehlerhafte Größen erhält, mit denen keine genaue Positionsbestimmung durchgeführt werden kann. Aufgrund dieser Unsicherheiten wird bei SLAM mit Wahrscheinlichkeitsrechnung und -verteilungen gearbeitet. Es werden im Folgenden zwei Verfahren für SLAM vorgestellt, mit denen es möglich ist, die Position innerhalb einer gerade erstellten Karte möglichst genau zu schätzen und Fehler zu minimieren. Die Verfahren werden nur in ihrer Grundfunktionalität erklärt, da es für den späteren Verlauf dieser Arbeit nicht notwendig ist, die genaue mathematische Formulierung zu kennen. Der Vollständigkeit halber können diese jedoch unter [11] (Kapitel 3.2 und 3.3) nachgelesen werden.

EKF

Der Extended Kalman Filter ist eines der mit am weit verbreitetsten Verfahren, um im Bezug auf SLAM die Position des Roboters zu schätzen [79]. Dazu werden die Odometriedaten und die in 2.4.8 beschriebenen Landmarken benötigt. Die Abbildung 2.4 zeigt den Ablauf des SLAM-Prozesses mit dem EKF.

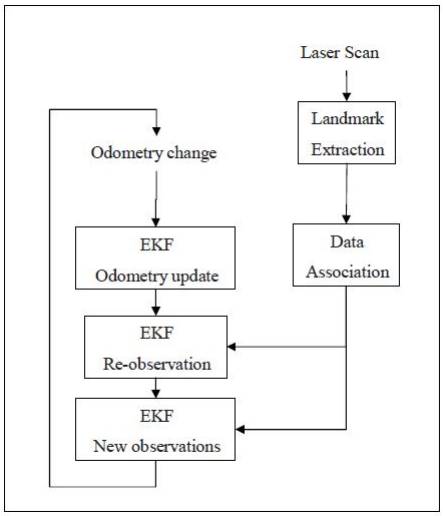

Abbildung 2.4.: SLAM-Prozess mit dem EKF [65].

Als erstes wird an Hand der alten Position und der veränderten Odometriedaten die Position des Roboters geschätzt. Im nächsten Schritt wird die geschätzte Position durch wieder erkannte Landmarken korrigiert. Die Landmarken werden im Bereich der *Data Association* in bereits erkannte oder neu hinzugekommene Landmarken eingestuft. Dies geschieht unter Einbeziehung von Matrizen, in denen unter anderem die aktuelle Roboterposition, sowie Kovarianzen berechnet und in jedem neuen Schritt aktualisiert werden. Genaueres zu den Matrizen kann unter [66] (Kapitel 11) nachgelesen werden. Im dritten und letzten Schritt werden neu erkannte Landmarken hinzugefügt und somit die Karte erweitert, in der sich der Roboter bewegt. Im nächsten Schritt beginnt der Prozess wieder von vorne [66].

Rao-Blackwellized Partikelfilter

Der Rao-Blackwellized (RBPF) ist eine Kombination aus dem oben genannten EKF und dem Partikelfilter. Vereinfacht funktioniert ein Partikelfilter folgendermaßen:
Es erfolgt eine Initialisierung der Roboterposition innerhalb der aktuellen Karte. Dabei werden alle möglichen Positionen des Roboters in eine Liste eingetragen. Mögliche Positionen (Partikel) sind die Orte, an denen sich der Roboter aufgrund seiner Messdaten aktuell befinden könnte [79]. Durch Steuerbefehle wird geschätzt, wie die Roboterposition im nächsten Schritt aussehen müsste. Die einzelnen geschätzten Partikel werden gewichtet. Nach Ausführung der Steuerbefehle wird ein sogenanntes *Resampling* durchgeführt, das Positionen, die durch

neue Messdaten unwahrscheinlicher sind, aus der Liste entfernt und neue hinzufügt [86]. Der Partikel mit der höchsten Gewichtung wird als aktuelle Roboterposition angenommen.
Das Ziel hinter der Kombination aus Partikelfilter und EKF ist, die hohe Anzahl an aufkommenden möglichen Positionen zu minimieren, um somit den Rechenaufwand geringer zu halten. Dies geschieht, indem der EKF bei der Schätzung für die nächst möglichen Roboterpositionen jedes Partikels verwendet wird. So können Landmarken mit Hilfe des EKFs wiedererkannt werden und der Partikelfilter muss nur die durch Landmarken identifizierten Partikel gewichten. Nach der Gewichtung der unterschiedlichen Partikel wird wieder ein *Resampling* durchgeführt [86].

2.4.9. Wegplanung

Bislang wurde erläutert, wie durch Sensoren die Umwelt wahrgenommen und der Roboter sich selbst in dieser lokalisieren kann. Die Grundvoraussetzungen für eine erfolgreiche Navigation sind somit geschaffen. Um aber von Punkt A nach Punkt B innerhalb einer Karte und somit auch in der reellen Welt zu gelangen, steht noch aus. Es muss die Frage „*Wie soll ein Zielort erreicht werden?*“geklärt werden. Im Folgenden werden Kartenmodelle zur Wegplanung sowie zwei verschiedene Wegplanungsalgorithmen, die in der Implementierung verwendet werden können, vorgestellt.

Karten

Ein Roboter benötigt für die Navigation mindestens zwei Karten. Eine globale Karte, die dem kompletten Weltmodell entspricht und eine lokale Karte, die den aktuellen Sichtbereich des Roboters repräsentiert [5][21][22]. Der Roboter bildet hierbei den Ursprung der lokalen Karte. Um sich innerhalb der globalen Karte zu lokalisieren, müssen beide Karten vereint werden. Hierfür muss die lokale Karte so transformiert werden, dass sie auf den entsprechenden Ausschnitt der globalen Karte passt [21][22]. Für lokale und globale Karten können unterschiedliche Modelle verwendet werden, die im Folgenden vorgestellt und auf ihre Einsatzmöglichkeit hin untersucht werden.

Geometrisches Modell

Bei einem geometrischen Modell werden die Positionen der Umgebungsmerkmale in ein Koordinatensystem eingetragen [22][5][76]. Die Positionen können direkt aus den Messdaten der Sensoren für die Entfernungsmessung übertragen werden. In der vorliegenden Arbeit wird die *Occupancy Grid* Karte vorgestellt. Das Occupancy Grid ist eine Rasterdarstellung

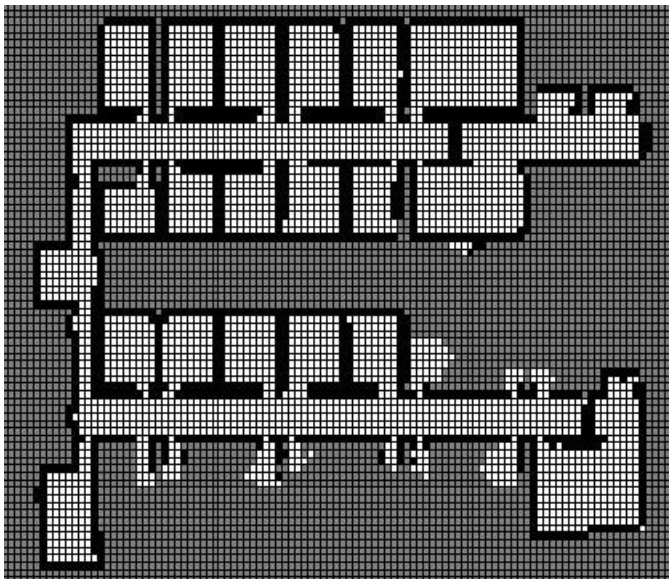

Abbildung 2.5.: Beispiel einer Occupancy Grid Karte [38].

einer geometrischen Karte [5]. Die Felder der Karte (Abbildung 2.5) haben eine fest definierte Größe [22][5]. Jedem Feld kann ein Wert zwischen 0 und 255 zugewiesen werden. Dieser Wert kategorisiert das Feld nach *befahrbar*, *unbekannt* oder *belegt*. Der Wert 255 wird meistens dem Zustand *belegt* zugeordnet. Geometrische Kartenmodelle eignen sich sehr gut für lokale Karten, da die einfache Übertragung der Messdaten in ein Koordinatensystem keinen hohen Rechenaufwand benötigt [5][76].

Topologisches Modell

Das topologische Weltmodell stellt die Umwelt in einem Graphen dar, der aus Knoten und Kanten besteht [5]. Die Knoten bilden den Repräsentant für identifizierte Landmarken. Die Kanten sagen aus, dass von einem Knoten zu einem anderen Knoten gelangt werden kann. Der Roboter wäre in der Lage von Landmarke A zu Landmarke B fahren. Das topologische Modell muss nicht maßstabgetreu sein [5][76]. Es dient lediglich der Darstellung von „Nachbarschaftsbeziehungen“[5]. Ein typisches Beispiel für ein topologisches Weltmodell ist ein Liniennetz eines Eisenbahnunternehmens oder eines Busunternehmens (Abbildung 2.6) [5]. Das topologische Modell eignet sich insbesondere für die globale Karte, da in dieser üblicherweise die Bestimmung des groben Wegplans durchgeführt wird. Hierfür ist es hilfreich, eine abstrahierte Sicht auf die Umgebung zu haben, da das Ziel ist, einen Weg von Punkt A nach

B zu finden [76]. Hier würden viele verschiedene Umgebungsdetails einen verlangsamenden Faktor bilden [5].

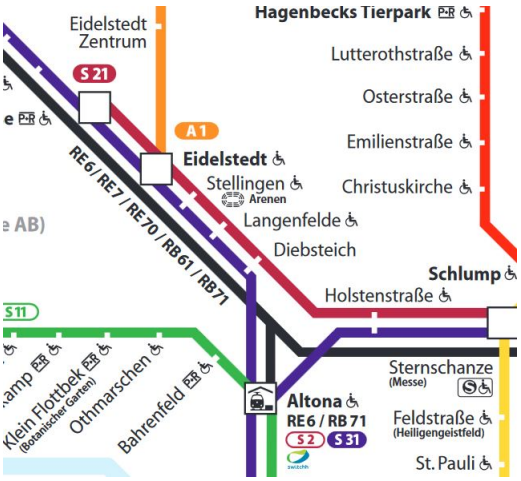

Abbildung 2.6.: Liniennetzplan des HVV Hamburg [34].

Dijkstra Algorithmus

Der Dijkstra-Algorithmus löst das Problem des kürzesten Pfads [75]. Er basiert auf einem kantengewichteten Graphen. Die Knoten des Graphen stellen Orte dar, zu denen gelangt werden kann. Die Kanten sind mit positiven Gewichtungen (Kosten) belegt. Der Dijkstra-Algorithmus errechnet für jeden Knoten im Graphen den kostengünstigsten Weg von einem Startpunkt S aus. Es wird wie folgt vorgegangen:

1. Weise allen Knoten die Attribute *besucht*, *Entfernung zu S* und *Vorgänger* zu.
2. Initialisiere für alle Knoten außer S, dass sie nicht *besucht* sind, die *Entfernung zu S* gleich unendlich ist und der *Vorgänger* undefiniert ist.
3. Für den Startknoten S setzte die Entfernung zu S auf null und den Vorgänger auf sich selbst.
4. Wiederhole folgenden Vorgang, bis alle Knoten als *besucht* markiert sind:
   4.1. Wähle den Knoten, der die geringsten Kosten (Kantengewicht) hat und markiere diesen als *besucht*.

4.2. Errechne für alle Nachbarn des gerade markierten Knotens die eigene *Entfernung zu S* plus das Kantengewicht zu dem Nachbarknoten:

4.2.1. Wenn die Summe geringer ist als die im Nachbarn aktuell gespeicherte *Entfernung zu S*, dann setzte die Summe als *Entfernung zu S* und setze den aktuellen Knoten als *Vorgänger* des Nachbarknotens.

Der Dijkstra-Algorithmus ist eine einfache Möglichkeit, um im späteren Verlauf dieser Arbeit einen Weg zu einem Punkt X in einer vom Roboter erstellten Karte zu finden. Wie oben erläutert, muss die Karte allerdings auf Basis eines Graphen sein.

A* Algorithmus

Der A*-Algorithmus (gelesen *A Star*) ist eng verwandt mit dem Dijkstra-Algorithmus. Der Unterschied zwischen den Algorithmen ist, dass mit Dijkstra der kürzeste Weg vom Startknoten zu jedem Knoten innerhalb des Graphen errechnet werden kann. Beim A* Algorithmus wird hingegen nur der kürzeste Weg von S zu einem Zielpunkt Z errechnet. Die Kosten des Knotens werden hier nicht wie bei Dijkstra allein über die Entfernung zum Startknoten S berechnet, sondern bilden sich aus der Entfernung zu S und einer Heuristik, die die Entfernung zu dem Zielpunkt Z schätzt [83][78]. Dadurch arbeitet der Algorithmus „(...) zielgerichteter als der Dijkstra-Algorithmus“[83]. Knoten des A* haben ebenfalls die Attribute *Vorgänger* und *Kosten*. Der Algorithmus arbeitet mit zwei getrennten Listen, die Knoten speichern. Die *open list* nimmt die Nachbarknoten auf und sortiert sie nach Berechnung der Kosten in aufsteigender Reihenfolge. Untersucht wird aus der *open list* immer der Knoten mit den geringsten Kosten. Nach der Untersuchung eines Knotens wird dieser in die *closed list* verschoben. In dieser Liste stehen alle Knoten, für die ein optimaler Weg bestimmt wurde [83]. Der Algorithmus wird solange ausgeführt, solange der Zielkonten Z nicht der *open list* entnommen wurde oder die Liste nicht leer ist. Die *closed list* wird später verwendet, um den optimalen Weg zu rekonstruieren. Eine kurze Beschreibung des Ablaufs soll das Verfahren verdeutlichen:

1. Solange Z nicht gefunden und die *open list* nicht leer ist
2. Nehme den ersten Knoten aus der Liste:
   2.1. Prüfe für jeden Nachbarn des entnommen Knotens, ob er in der *closed list*, in der *open list* oder in gar keiner Liste enthalten ist.
      2.1.1. Der Nachbarknoten ist in der *closed list* und die neu errechneten Kosten sind geringer als die zu einem früheren Zeitpunkt berechneten Kosten. Der Nach-

barknoten wird aus der *closed list* entfernt und mit den neu berechneten Kosten in die *open list* aufgenommen

2.1.2. Der Nachbarknoten ist in der *open list* enthalten und die neu errechneten Kosten sind geringer als die zu einem früheren Zeitpunkt berechneten Kosten. Die Kosten des Nachbarknotens werden aktualisiert und er bleibt in der *open list.*

2.1.3. Der Nachbarknoten ist in keiner der Listen enthalten. Es werden die Kosten aus den Kosten bis zum Startpunkt S plus die geschätzten Kosten bis zum Zielpunkt Z errechnet und dem Knoten zugewiesen. Anschließend wird er in die *open list* aufgenommen

3. Füge den Knoten der *closed list* hinzu.

Wurde der Zielknoten Z bearbeitet, ist der Algorithmus erfolgreich gewesen und der Weg kann über die *closed list* rekonstruiert werden. Gibt es keinen Knoten mehr in der *open list*, kann kein Weg zum Zielknoten Z bestimmt werden [83].

2.5. Sensorfusion und Sensorintegration

Der Einsatz von mehreren unterschiedlichen Sensoren bringt verschiedene Vorteile. Durch mehrere Sensoren kann ein größerer Sichtbereich wahrgenommen werden, wodurch die Wahrnehmung des Roboters erhöht wird. Ein einfaches Beispiel ist ein Laserscanner und ein Bildsensor. Durch die Hinzunahme des Bildsensors können nun auch Farben für detektierte Objekte erfasst werden. Des Weiteren bietet der Einsatz von mehreren Sensoren eine höhere Zuverlässigkeit und Genauigkeit, da über mehrere Sensoren die gleiche Umgebung wahrgenommen werden kann und somit Fehler verringert werden können. Der Einsatz verschiedener Sensoren wird in der Literatur oft als Sensorintegration und Sensorfusion bezeichnet [30] [16]. Unter Sensorfusion wird die Verknüpfung der einzelnen Sensordatenströme zu einem Einzigen verstanden. Aus einem fusionierten Sensordatenstrom können spezifische Informationen gewonnen werden, die von Relevanz für ein Robotersystem sein können [30] [16]. Beispielsweise kann ein Sensor, der für die Erkennung des Fernbereichs geeignet ist durch einen Sensor erweitert werden, der nur für Erkennung des Nahbereichs geeignet ist. Durch die Fusion der Datenströme entsteht ein Datenstrom, der den Nah- sowie Fernbereich abdeckt. Es wirkt, als wäre nur ein Sensor im Einsatz.
Sensorintegration bedeutet aus jedem einzelnen Sensor Informationen zu erhalten, die aufeinander aufbauen oder sich gegenseitig beeinflussen, um eine gemeinsame Aufgabe zu lösen

[30] [16]. Ein klassisches Beispiel wäre das Greifen eines Roboterarms nach einem Gegenstand. Ein Sensor erkennt, dass sich ein Gegenstand in greifbarer Nähe befindet. Durch Steuersignale wird der Arm so positioniert, dass der Gegenstand gegriffen werden kann. Ob der Greifvorgang erfolgreich war, wird über einen Kontaktsensor ermittelt. Die Integration beider Sensoren in den Hauptvorgang „Greifen“ führt dazu, dass der Vorgang erfolgreich abgeschlossen werden kann. Mit nur einem Sensor kann diese Aufgabe nicht bewerkstelligt werden.

3. Entwicklungsprozess

Das nachfolgende Kapitel erläutert die aufeinander aufbauenden Entwicklungsschritte zur Lösung der Problemstellung. Es sollte das bestehende Telepräsenzsystem MoRy-A um eine teilautonome Navigationsunterstützung erweitert werden. Hierfür wurde als erstes der bestehende Teleoperator genauer beschrieben (3.1). Im nächsten Schritt wurde untersucht, welche der an der Hochschule für Angewandte Wissenschaften Hamburg zur Verfügung stehenden Sensoren dem Roboter die bestmögliche Sicht auf die Umgebung bringen können (3.2). Somit wurde die Grundlage für die Kartierung und Lokalisierung geschaffen. Diese wird im Abschnitt 3.3 näher erläutert. Als nächster Entwicklungsschritt wurden auf Basis der aus den Daten errechneten Karte eine Navigation zu einem Punkt sowie die Kollisionsvermeidung implementiert. Die verwendete Software ermöglicht es dem Benutzer des Telepräsenzsystems, den Teleoperator autonom steuern zu lassen. Um für zukünftige Arbeiten am Telepräsenzsystem die Funktion des autonomen Fahrens einfach zu ermöglichen, wurde auf eine modulare Softwarearchitektur gesetzt, die in Abschnitt 3.5 beschrieben wird. Zur Lösung des teilautonomen Fahrens wurde ein Modul implementiert, das in 3.6 erläutert wird. Anschließend wird auf die Inbetriebnahme des Teleoperators, sowie die erzielten Fahrergebnisse eingegangen.

3.1. Telepräsenzsystem MoRy-A als Basis

Das Telepräsenzsystem MoRy-A besteht aus den in 2.1 beschriebenen Komponenten Teleoperator, Operator, Bedienerstation und Kommunikationskanal. Eine grafische Übersicht über die Komponenten bietet die Abbildung 3.1.

Die Bedienerstation besteht aus einem stationären Computer, einem DualShock Controller als Eingabegerät sowie einer Oculus Rift[1]. Die Steuerung des Teleoperators erfolgt über den DualShock-Controller und die Oculus Rift. Als Grundlage für die Verarbeitung von Steuerbefehlen wird die Software Robot Operating System (ROS) verwendet, die im späteren Verlauf dieses Abschnitts vorgestellt wird. Die genaue Steuerung innerhalb der Bedienerstation ist

[1] Oculus Rift - Die Oculus Rift ist ein HMD inklusive Trackingsystem für die Kopfposition.

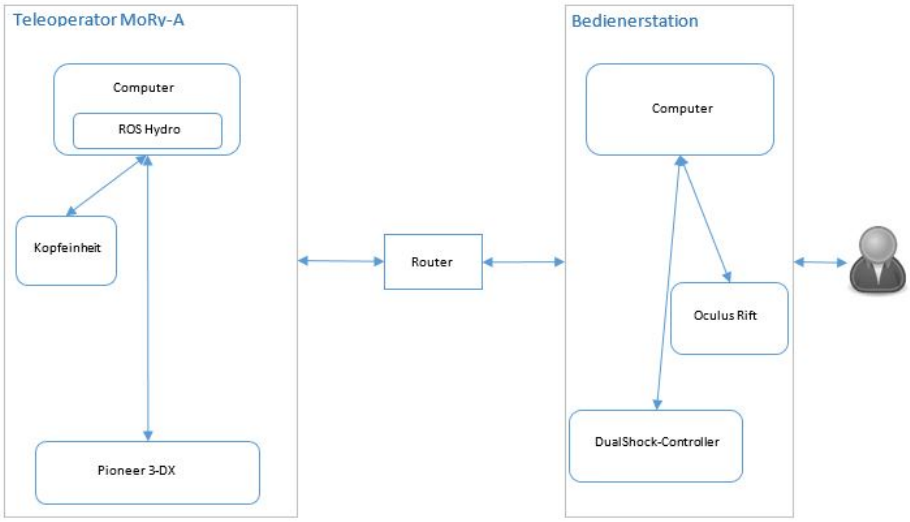

Abbildung 3.1.: Basiskomponenten des Telepräsenzsystems MoRy-A.

für die in dieser Arbeit behandelte Problemstellung irrelevant und kann in der vorhergegangen Arbeit *„Entwicklung einer Basisplattform für Telepräsenzsysteme“* von Hendrik Wiese [87] nachgelesen werden. Parallel zu dieser Arbeit wird die Bedienerstation in der Arbeit *„Leitstand für mobile Systeme mit Virtual Reality“* erweitert. Die genaue Steuerung nach der Erweiterung der Bedienerstation kann der Arbeit von Moritz Heindorf [35] entnommen werden. Die Kommunikation zwischen Bedienerstation und Teleoperator erfolgt über einen herkömmlichen Wireless LAN-Router (Kommunikationskanal).

Der Teleoperator ist die mobile Einheit des Telepräsenzsystems, mit der sich der Benutzer durch die Umgebung bewegt. Die mobile Einheit besteht aus der Plattform Pioneer 3-DX, einem Computer der Firma Intel (Typkennzeichnung: Intel NUC D54250WYKH)), sowie einer Kopfeinheit, die in vorherigen Arbeiten ([87] und [33]) entwickelt und optimiert wurde. Die Abmessungen des Teleoperators sind wie folgt: Höhe (mit Kopfeinheit) h=1.25 m, kreisförmige Plattform mit Radius r=0.25 m. Im Folgenden werden die Plattform und die Software, die für den bisherigen Gebrauch installiert wurde, erläutert. Die Kopfeinheit wird nicht näher erläutert, da sie für diese Arbeit nicht von Relevanz ist, aber der Vollständigkeit halber mit aufgelistet ist.

3.1.1. Pioneer 3-DX (P3-DX)

Der Pioneer 3-DX (Abbildung 3.2) ist die Basisplattform des Teleoperators (mobile Einheit). Das Gehäuse besteht aus 1,6 mm starken Aluminium. Das Gewicht beträgt 9 kg und es können maximal 17 kg hinzugeladen werden, um mit einem Gesamtgewicht von 26 kg manövrierfähig zu bleiben. Der P3-DX besitzt drei Räder. Das linke und rechte Rad werden über ein Differentialfahrwerk angesteuert. Das dritte Rad dient lediglich zur Stabilisierung der Plattform und kann nicht direkt angesteuert werden. Es sind also die Bewegungen vor, zurück, links, rechts und Rotation auf der Stelle möglich. Über die mit Luft befüllten, einzeln ansteuerbaren Räder kann die Plattform Stufen bis 2.5 cm, Spalten im Boden bis 5 cm und eine Steigung von 25% überwinden. Es sind Geschwindigkeiten von maximal 1.2 m/s in der Vorwärts- und Rückwärtsbewegung mit einer Drehgeschwindigkeit von 300 $^\circ/s$ möglich. Das Ansteuern erfolgt über eine RS232-Schnittstelle. Über seine acht Ultraschall-Distanzsensoren kann der P3-DX, bei entsprechend vorhandener Software, Hindernisse erkennen. Die Stromversorgung wird über drei 12 V 7,2 Ah Bleiakkumulatoren sichergestellt. Die Akkumulatoren können während des Betriebs getauscht werden, solange ein Akkumulator mit der Plattform verbunden bleibt. Die maximale Laufzeit laut des Pioneer 3-DX beträgt bei voll aufgeladenen Akkumulatoren acht bis zehn Stunden [1]. Der P3-DX wird über das ROS (3.1.2) Package *ROSARIA* gesteuert. Die Spezifikation des Packages sind dem Anhang C.1 zu entnehmen. Für die Plattform wird der Treiber *ARIA* von Adept MobileRobots [54] verwendet.

Abbildung 3.2.: Adapt MobileRobots Pioneer 3-DX [9].

3.1.2. ROS - Robot Operating System

Um mit der beschriebenen Plattform kommunizieren zu können, ist auf dieser der oben erwähnte Intel NUC verbaut. Ubuntu 12.04 LTS bildet das Betriebssystem des Computers. Zusätzlich ist

das *„Meta-Betriebssystem“* [68] ROS installiert. ROS bringt Funktionen, wie *„Hardwareabstraktion, Gerätetreiber, Utilityfunktionen, Interprozesskommunikation und Paketmanagment“* [68] mit sich. Die Grundlage in ROS bilden *Messages*, die entweder selber definiert werden, oder aus bereits vordefinierten ROS *Messages* gewählt werden können. Jede *Message* beinhaltet unterschiedliche Informationen, wie zum Beispiel einfache Datentypen, oder auch andere *Messages*. Die *Messages* werden entweder über *Services* oder *Topics* zwischen sogenannten ROS-*Nodes* kommuniziert. *Topics* und *Services* haben eigenständige Namen. ROS-*Nodes* werden Programme genannt, die wie gewöhnliche Programme funktionieren, also Berechnungen o.Ä. durchführen, aber ROS als Kommunikationsmittel verwenden. *Nodes* sind in sogenannten *Packages* enthalten. Ein *Package* kann verschiedene *Nodes* enthalten, die unterschiedliche Aufgaben ausführen. Mehrere *Packages* werden zu einem sogenannten *Stack* zusammen gefasst.
Über *Services* können synchrone Nachrichten verschickt werden. Dazu fragt ein *Node* über einen *Service*, der von einem anderen *Node* angeboten wird, einen bestimmten Wert ab und wartet auf dessen Antwort. Bei *Topics* hingegen gibt es ROS-*Nodes*, die als *Publisher* und als *Subscriber* fungieren. *Publisher* veröffentlichen selbstständig *Messages* bzw. Werte auf der *Topic* ohne Nachfrage eines anderen *Nodes*. Ein *Subscriber* kann sich über den Namen einer *Topic* auf dieser registrieren. Wird von einem *Publisher* eine *Message* auf der *Topic* veröffentlicht, wird im *Subscriber* eine Funktion aufgerufen, die vorher bestimmt wurde. Diese Funktionen heißen *Callbackfunktionen*.
Um ROS in Betrieb nehmen zu können, muss eine Version von ROS installiert sein. Auf dem verbauten Intel NUC ist die Version ROS Hydro Medusa [69] installiert. Um die Funktionen von ROS nutzen zu können, muss immer als erstes der Befehl *roscore* auf dem ausführenden System (hier der Intel NUC) ausgeführt werden. Dieser Befehl startet den *Master-Node*. Der *Master* stellt einen Namens- und Registrationsdienst, bei dem die einzelnen *Topics* und *Services* registriert werden. Ohne das Registrieren beim *Master* können *Topics* und *Services* nicht von den *Nodes* kontaktiert werden. Es wäre somit keine Kommunikation möglich. Des Weiteren startet der Befehl *roscore* einen Parameter-Server, der Parameter von einzelnen *Nodes* verwaltet und speichert. Als drittes wird ein *Node* gestartet der *rosout* heißt. Über diesen *Node* werden alle Fehler, Warnung oder Debuginformationen geloggt.

3.2. Verwendete Sensoren für die Umgebungswahrnehmung

Im Kapitel 2.4 wurde beschrieben, mit Hilfe welcher Sensoren die Umgebung im Indoor-Bereich erkannt werden kann. An der HAW Hamburg standen eine Kinect Xbox 360, ein Laserscanner der Firma Sick, sowie das am Pioneer 3-DX vorhandene Sonar zur Verfügung. Für die

Abbildung 3.3.: 2D-Laserscanner Sick TIM310 [74].

Umgebungserkennung wurde die Kinect der Spielekonsole Xbox 360 und der Laserscanner Tim310 der Firma Sick verwendet. Es wurde sich aufgrund der mangelnden Richtungsbestimmung von Objekten gegen das Sonar entschieden. Des Weiteren können sich in büroähnlichen Indoorumgebungen Gegenstände befinden, die, wie in Abbildung 2.3 gezeigt, ungünstige Kanten aufweisen. Tür-, Stuhl- oder Tischkanten könnten eventuell nicht erkannt werden. Die Kombination aus Laserscanner und Kinect wurde von Vorteil gegenüber dem Sonar bzw. Kombinationen mit dem Sonar befunden, da die einzelnen Sensoren weniger fehleranfällig gegenüber den Nachteilen des Sonars sind. Im Folgenden werden die Sensoren inklusive verwendeter Software sowie die Erweiterung des Teleoperators (3.1) beschrieben.

3.2.1. Sick TIM310

Der Sick TIM310 ist ein 2D-Laserscanner. Die Stärke der Laserstrahlung ist als ungefährlich eingestuft (Laserklasse 1). Der Laserscanner hat einen radialen Sichtbereich und einen Öffnungswinkel von 270 °. Objekte können in einem Bereich von 0.05 m bis 4 m mit einer Auflösung von 1 ° detektiert werden. Ein Objekt kann bei 0% Remission[2] in einer Entfernung von 4 m ab einer physikalischen Mindestgröße von 170 mm detektiert werden. Die Scanfrequenz liegt bei 15 Scans pro Sekunde (15 Hz). Die Versorgungsspannung des Sick TIM310 liegt zwischen 10 - 28 Volt DC. Angeschlossen wird der TIM310 über das Anschlussmodul CDB730-001 [73], welches den Laserscanner mit einer externen 0,8 A Sicherung schützt. Das Modul ist auf dem Pioneer 3-DX (3.1.1) verbaut und wird über die drei Akkumulatoren gespeist. Konfigurationen für den Messbereich sowie die Übermittlung der Messwerte erfolgen über eine USB-Schnittstelle. Diese ist mit einer USB-Schnittstelle des verbauten Intel NUC verbunden. Auf dem Intel NUC wird das ROS-Package *sick_tim* verwendet. Das Package stellt sowohl die Treiber, als auch entsprechende *Nodes* und *Topics*, die für die Inbetriebnahme des Laserscanners

[2] Als Remission bezeichnet man das von einem Körper zurückgeworfene ungerichtete Licht [4]

benötigt werden. Für diese Arbeit wurde der Node *sick_tim310s01* mit der mitgelieferten default Konfiguration verwendet (siehe Anhang C.2). Montiert wurde der Laserscanner in einer dafür vorgesehen Halterung auf der Plattform 3-DX mit der Scanrichtung nach vorne. Der Abbildung 3.5 ist die Position auf der Plattform zu entnehmen. Die Höhe des Scanners beträgt 0.42 m über dem Boden. Objekte in der Umgebung müssen daher eine Mindesthöhe von 0.42 m aufweisen. Problematisch wird dies jedoch erst, wenn die mobile Einheit sich in engen Räumlichkeiten bewegt, in denen die Objekte zu dicht für die Erkennung durch die Kinect (siehe unten) sind.

3.2.2. Microsoft Kinect

Die Microsoft Kinect, die erstmals am 1. Juni 2009 unter dem Projektnamen *Project Natal* vorgestellt wurde, ist eine Steuereinheit für Multimediageräte [52]. Ausgestattet mit einem Infrarot-Emitter, einer Infrarot-(CMOS) und RGB-Kamera, sowie einem Mikrofon-Array zur Detektion von Geräuschen, ist es der Microsoft Kinect möglich, die Umgebung zu detektieren. Die Anordnung der einzelnen Hardwarekomponenten sind der Abbildung 3.4 zu entnehmen. Mit dem in der Abbildung (3.4) gezeigten Motor, ist es möglich, den Kopf der Steuereinheit um

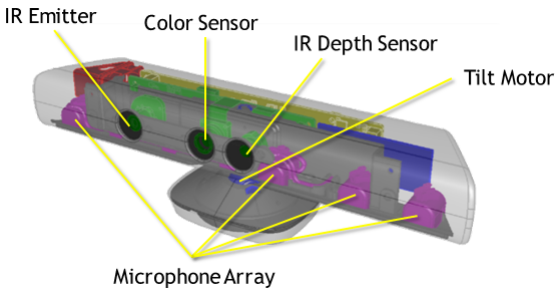

Abbildung 3.4.: Microsoft Kinect Hardwarekomponenten [56].

$\pm 27°$ zu neigen. Der Sichtbereich der Kameras beträgt vertikal 43 ° und horizontal 57 °. Es wird in dieser Arbeit nur auf den Infrarot-Emitter und die Infrarot-Kamera eingegangen, da nur diese zur Bestimmung der Entfernung von Objekten verwendet werden. Es wird im Folgenden ein grober Einblick in die Funktionsweise der Entfernungsmessung der Kinect gegeben.
Durch den IR-Emitter wird ein Punktmuster in den Raum projiziert. Die Punkte des Musters sind in einer bestimmten und bekannten Form angeordnet. Fällt das projizierte Punktmuster auf ein Objekt im Raum, ändert sich der Abstand der einzelnen Punkte des Musters. Das projizierte und ggf. veränderte Muster wird über die IR-Kamera mit 30 Frames per Second (FPS)[3]

[3]Gibt die Anzahl der Bilder an, die pro Sekunde aufgenommen werden.

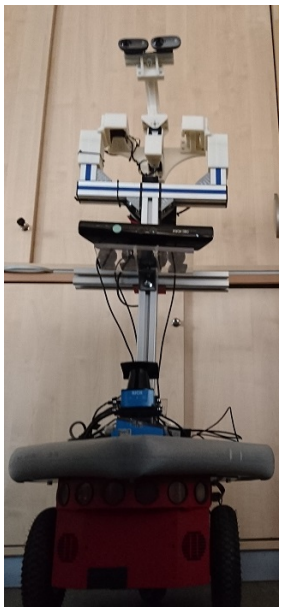

Abbildung 3.5.: Telepräsenzsystem MoRy-A.

aufgenommen. Die Auflösung der IR-Kamera beträgt 640 x 480 Bildpunkte [57]. Durch die Änderung der angeordneten Punkte kann die Entfernung von Objekten in einer Entfernung von 0.4 m bis 8 m bestimmt werden. Die Fehleranfälligkeit steigt jedoch bei zu dichten oder zu weit entfernten Objekten. Daher können Distanzen zu Objekten nur in einer Entfernung von 1 m bis 3 m genau detektiert werden [40]. Die Kinect wird durch ein Netzteil mit 12 Volt und maximal 1.1 Ampere gespeist und ist per USB-Schnittstelle mit dem Intel NUC verbunden. Es werden die ROS-Packages *openni_launch* und *openni_camera* (Anhang C.3) verwendet. *openni_camera* stellt hierbei den Treiber für die Kinect auf dem Linuxsystem dar. Über das Package *openni_launch* ist es möglich, Zugriff auf den benötigten IR-Datenstrom zu erlangen. Montiert ist die Kinect in einer Höhe von 0.75 m über dem Boden mit der Scanrichtung nach vorne. Abbildung 3.5 ist die Position am Teleoperator zu entnehmen. Die Höhe von 0.75 m ergibt sich aus folgender Problemstellung:
Die mobile Einheit bewegt sich in einer Umgebung, in der Objekte, wie z.B. Tische vorkommen. Der oben erwähnte Laserscanner würde bei einem Scan der Umgebung nur die Tischbeine des Tisches erkennen. Aus dem Datenstrom kann nicht erkannt werden, dass sich in einer gewissen Höhe eine Tischplatte befindet.
Der Teleoperator hat, wie in 3.1 beschrieben, eine Höhe von 1.25 m. Das bedeutet, dass die Durchfahrtshöhe von Objekten größer als 1.25 m sein muss, um den Roboter nicht zu gefährden.

Es wurde zur Sicherheit ein Puffer von 0.25 m auf die aktuelle Höhe des Roboters addiert. Daraus ergibt sich eine Gesamthöhe von 1.50 m. Die Montagehöhe der Kinect muss bei der Hälfte der Gesamthöhe liegen, da der vertikale Sichtbereich ein gleichschenkliges Dreieck bildet und die Distanz zu Objekten, die in 1.5 m Höhe erkannt werden sollen, möglichst gering sein soll. Die kürzeste Distanz in der ein Objekt dieser Höhe erkannt wird, beträgt 1.90 m. Abbildung 3.6 und die unten stehende Berechnung sollen dies verdeutlichen.

$$b = a/\tan(\alpha) = 0.75m/\tan(21.5)^\circ \approx \underline{\underline{1.90m}} \tag{3.1}$$

Mit:

$$a = 1.5m/2 = 0.75m \quad \text{und} \quad \alpha = 43^\circ/2 = 21.5^\circ \tag{3.2}$$

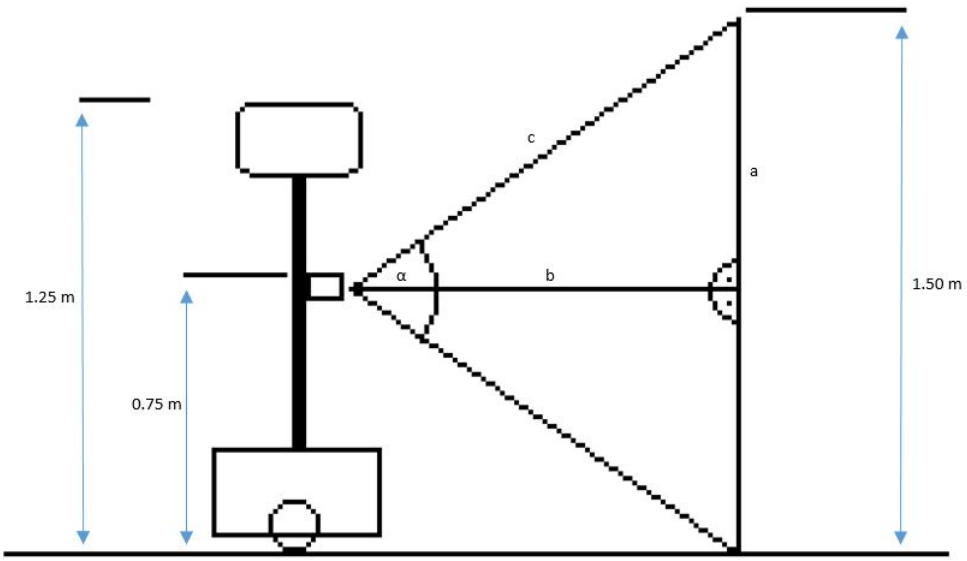

Abbildung 3.6.: Berechnung der minimalen Entfernung von Objekten mit 1.5 m Höhe.

3.3. SLAM als Grundvoraussetzung

Dieser Abschnitt beschreibt, wie die Umgebung von der mobilen Einheit erfasst, kartiert und sich darin lokalisiert werden kann. Beide Sensoren sollten zu einem Datenstrom zusammengefasst werden, um eine Karte zu generieren, die alle Objekte der Umgebung aufzeigt, die für den Roboter relevant werden können (also alle Objekte mit einer Höhe von 0.01 m bis 1.50 m). Es müssen für eine Fusionierung der Datenströme der Kinect, sowie die des Lasers die gleichen Informationen vorliegen, um diese sinnvoll miteinander zu verknüpfen. Der Laserscanner liefert Daten in Form einer *sensor_msgs/LaserScan*-Nachricht. Die Nachricht enthält unter anderem ein *float32 ranges*-Array, in dem alle gemessenen Distanzen in m gespeichert sind. Des Weiteren werden Start- und Endblickwinkel in rad angegeben. Aus diesen beiden Größen lässt sich der Gesamtblickwinkel des Laserscanners bestimmen. Die Kinect hingegen verwendet mehrere Nachrichtentypen zur Kommunikation ihrer Daten. Über das 640 x 480 (Spalten x Reihen) Pixel große Tiefenbild [57], das im Nachrichtenformat *sensor_msgs/Image* vorliegt, werden die gemessenen Werte der Kinect kommuniziert. Um die Sensordaten zu fusionieren, müssen wie oben beschrieben, die Datenströme in gleicher Form vorliegen. Um den Datenstrom der Kinect in den des Laserscanner zu fusionieren, stand die Frage im Raum, wie eine *sensor_msgs/Image*-Nachricht in eine *sensor_msgs/LaserScan*-Nachricht fusioniert werden kann, ohne die Information der Höhe, die im Tiefenbild der Kinect enthalten ist, zu verlieren. Der Lösungsansatz, der im Rahmen dieser Arbeit verwendet wurde, basiert auf der Idee, die einzelnen Spalten des Tiefenbildes zu durchlaufen und in jeder Spalte den Wert herauszusuchen, welcher am niedrigsten ist. Der niedrigste Wert entspricht dem kleinsten Abstand zur Kinect. Zur Konvertierung des Tiefenbildes zu einer *Laserscan*-Nachricht wird das ROS-Package *depthimage_to_laserscan* verwendet (Anhang C.4). Für die Konvertierung wurden die Parameter: *scan_height = 400*, *range_min = 0.5* und *range_max = 2.0* gesetzt. Die *scan_height* repräsentiert die Höhe des Bildes, das zur Konvertierung genutzt wurde. Hier wurde der Wert von ursprünglich 640 auf 400 herunter gesetzt, da bei höher gesetzten Werten der Untergrund auf dem der Roboter fährt, als geringste Distanz gemessen wurde und somit nicht die Objekte in der Entfernung gemessen werden konnten. Der Parameter *range_min* und *range_max* repräsentieren die Distanz innerhalb der Objekte erkannt werden sollen. Die Werte 0.5 m als Untergrenze und 2.0 m als Obergrenze wurden experimentell durch Fahrversuche bestimmt. Innerhalb dieser Distanz werden die besten Ergebnisse erzielt. Es lagen nach der Konvertierung zwei Nachrichten des Typ *LaserScan* zu Grunde, die fusionieren werden sollten. Der Laserscanner liefert bei einem Sichtwinkel von 270 $^\circ$ für jeden Grad einen Wert (270 Messwerte). Die Kinect hat bei einem horizontalen Sichtwinkel von 57 $^\circ$ 640 Werte, die erfasst werden. Diese

640 Werte mussten so in die Daten des Laserscanners fusioniert werden, dass sie hinterher der Realität entsprachen. Die Scans mussten also so übereinander gelegt werden, dass die 57 ° der Kinect dieselben 57 ° des Laserscanners widerspiegelten. Wie auch im Paper [90] wurde das *IRA Laser Tools* ROS-Package benutzt, um die beiden Laserscans zu verknüpfen. Das Resultat der Verknüpfung war eine funktionierende Fusion zweier Nachrichten desselben Typs.
Der nächste Schritt zur Kartierung der Umgebung und der Lokalisation der mobilen Einheit, war die Erstellung einer Umgebungskarte. Das ROS-Package *gmapping* (Anhang C.5) bringt einen kompletten SLAM mit sich. Das Package setzt sich aus Kartierung und Lokalisierung zusammen. Für die Kartierung wird *slam_gmapping* verwendet und für die Lokalisierung ein Rao-Blackwellized Partikelfilter (2.4.8), welcher ebenfalls im Node *slam_gmapping* enthalten ist. Das Package *gmapping* benötigt zum Ausführen von SLAM einige Voraussetzungen:
Die zu verarbeitenden Nachrichten müssen im *sensor_msgs/LaserScan*-Format vorliegen, da *gmapping* laserbasiert läuft. Es muss zusätzlich eine Odometrie Quelle (3.1.1) vorhanden sein, die für die Lokalisierung neben den Entfernungsdaten Informationen über die gefahrene Strecke liefert. Die einzelnen Sensoren müssen so miteinander kommunizieren können, dass ROS Aussagen über Entfernungen und Positionen der einzelnen Sensoren zueinander treffen kann. Somit können die von den Sensoren erhaltenen Daten kombiniert und verarbeitet werden. ROS muss als Beispiel bekannt sein, dass der Laserscanner 0.1 m in X-Richtung (vorne) und 0.2 m in Z-Richtung (Höhe) über dem Robotermittelpunkt sitzt. Unter diesen Voraussetzungen ist es möglich aus den Sensordaten Aussagen bezüglich der Entfernung zu erhalten. Hierfür wird ein sogenannter Transform-Baum (tf-Baum, tf-tree) aufgespannt. Mit einem tf-tree ist es möglich, den Roboter zu beschreiben. In ROS wird für jeden Sensor, für den Roboter selbst, und für die globale Welt ein sogenanntes *Frame* erstellt, in dem sich bewegt wird. Ein *Frame* ist ein Koordinatensystem. Erkennt der Laserscanner ein Objekt in seiner Umgebung, trägt er diesen Punkt in sein Koordinatensystem ein. Möchte ROS Aussagen zur Entfernung des Robotermittelpunkts zu dem eben erkannten Punkt treffen, muss bekannt sein, wie der Laserscanner in Relation zum Mittelpunkt des Roboters angebracht ist. Diese Beschreibung der Beziehung zueinander beziehungsweise die Transformation von einem Frame in ein anderes wird mit Hilfe des *tf* berechnet. Die genaue Funktionsweise von *tf* kann dem Paper [25] entnommen werden. Der Aufbau des *tf-Trees* dieser Arbeit ist im Anhang A dargestellt.
Nach Erfüllung der Voraussetzungen kann *gmapping* gestartet werden. Dies erfolgt durch folgende Befehlskette:

1. *roscore* - Zum Starten von ROS
2. Starten von *ROSARIA*

3. Starten der Sensoren Sick Tim310 und Kinect (mit Umwandlung in 2D-Laserscan)
4. Verknüpfung der Sensordatenströme
5. Transformationen über *tf* erstellen
6. *rosrun*[4] *gmapping slam_gmapping scan:=multi_scan*

Das Erstellen der Transformation unter Punkt 5 erfolgt über einen *static_transform_publisher*. Der *static_transform_publisher* verbindet in diesem Fall den Robotermittelpunkt (*base_link* genannt) mit den verwendeten Sensoren. Zum besseren Verständnis wird der Transform von *base_link* zum Laserscanner erklärt:

rosrun tf static_transform_publisher 0.28 0.0 0.11 0.0 0.0 0.0 /base_link /laser 100

Dieser Aufruf sorgt dafür, dass alle 100 ms die Transformation von *base_link* zum *laser* über *tf* veröffentlicht wird. In diesem Beispiel ist der Laserscanner vom Robotermittelpunkt um 0.28 m in x-Richtung und 0.11 m in z-Richtung verschoben. Argumente des *static_transform_publisher* sind folgendermaßen zu verstehen:

static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms

Das Argument *multi_scan* bei Punkt 6 gibt die Topic an, auf die der fusionierte Datenstrom geschrieben wird. Abbildung 3.7 zeigt das Ergebnis der Kartierung mit dem fusionierten Datenstrom. Als Vergleich wird in Abbildung 3.8 das Ergebnis an Hand des Datenstroms des Laserscanners gezeigt. Auf den Abbildungen ist ein Teil des Gebäudes der HAW Hamburg abgebildet. Es handelt sich bei der Karte um einen Ausschnitt des Flurs, sowie einen angrenzenden Raum eines Stockwerkes. Es ist deutlich zu sehen, dass die entstandene Karte des Laserscanners genauer und feiner ist. Kanten werden präziser erkannt und nicht als doppelt oder ungenau dargestellt. Des Weiteren fällt auf, dass der Flur sowie der Raumausschnitt in der Abbildung 3.8 mehrere Krümmungen aufweist. Die Krümmungen, sowie die Ungenauigkeiten sind auf die Microsoft Kinect zurückzuführen.
Des Weiteren fiel während der Fahrt auf, dass Objekte, die durch die Kinect erkannt wurden und aus dem Sichtbereich dieser fielen, von dem Laserscanner als freie Fläche erkannt und gelöscht wurden. In Kombination von Laserscanner und Kinect ist die Karte aufgrund der hohen Ungenauigkeit unbrauchbar. Die Ergebnisse, sowie die Ungenauigkeit der Kinect werden durch die Paper [90] und [60] bestätigt.

[4] rosrun - Startet ein Node Aufruf:*rosrun Packagename Nodename Argumente*

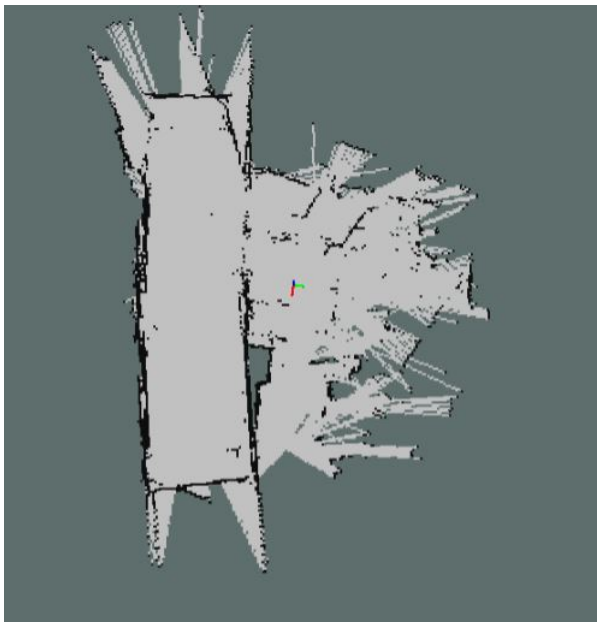

Abbildung 3.7.: SLAM mit dem fusionierten Datenstrom.

Aufgrund der Ergebnisse wird für die Kartierung und Lokalisierung nur der Laserscanner eingesetzt. Eine Datenfusion ist mit den gegebenen Sensoren nicht möglich. Die Kinect wird im späteren Verlauf der Arbeit unter anderem für die Kollisionsvermeidung verwendet. An dieser Stelle ist somit weder eine Sensorfusion noch Sensorintegration erfolgt. Die neue Befehlskette zur Inbetriebnahme des SLAM der ROS-Package *gmapping* sieht wie folgt aus:

1. *roscore* - Zum Starten von ROS
2. Starten von *ROSARIA*
3. Starten der Sensoren Sick Tim310
4. Transformationen über *tf* erstellen
5. *rosrun gmapping slam_gmapping scan:=scan*[5]

SLAM wird auch bei nicht teilautonomer Fahrt die Umgebung aufzeichnen, um bei Bedarf in jede mögliche, bereits bekannte Stelle der Karte zu navigieren.

[5] scan - *scan* ist die vom Laserscanner verwendete Topic

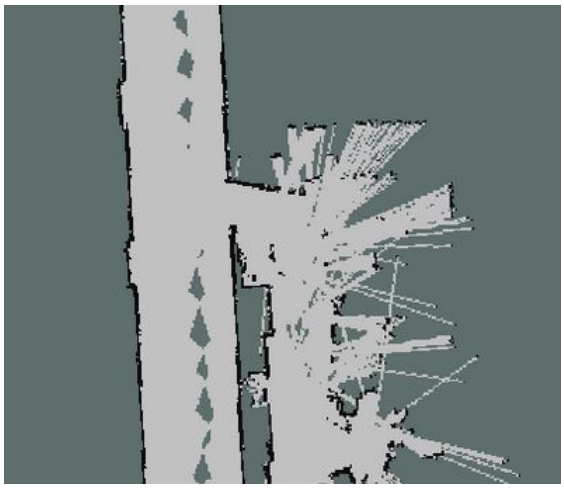

Abbildung 3.8.: SLAM mit dem Datenstrom des Sick Tim310.

3.4. Navigation Stack

Nachdem mittels *gmapping* SLAM erfolgreich durchgeführt werden kann, fehlten zur Navigation noch die Kollisionsvermeidung und die Wegplanung zu einem gewählten Punkt. In Kapitel 2.4.9 wurde bereits auf die Wegplanung eingegangen und die *Occupancy Grid* Karte vorgestellt. Die Werte (Kosten), die in einer solchen Karte (costmap) eingetragen werden können, sollen als Grundlage für die Wegfindung und die Kollisionsvermeidung dienen. Aufgrund der erstellten costmap, der erstellten Karte durch SLAM und den Sensordatenströmen soll zu einem gewählten Punkt navigiert werden können. Hierfür wird der *Navigation Stack* [49] eingesetzt, der ein Zusammenschluss mehrerer ROS-Packages ist, die für die Navigation geeignet sind. Die verwendeten ROS-Packages werden im Folgenden genauer auf ihre Funktionsweise, die für diese Arbeit benötigten Parameter, sowie den Einsatzzweck erläutert.

3.4.1. costmap_2d

Die Grundlage für die Navigation bildet die costmap, die durch das Package *costmap_2d* erstellt wird. Dieses erstellt auf Grundlage der Sensordatenströme des Laserscanners und der Kinect eine *occupancy grid* und belegt diese, bei Erkennung von Hindernissen, mit Kosten. Die costmap ist in eine globale und eine lokale Karte unterteilt [18]. Für beide Karten gelten die Parameter:

- *obstacle_range: 2.5* - Maximale Distanz in m, in der Hindernisse in die Costmap eingetragen werden.

- *raytrace_range: 3.0* - Maximale Distanz in m zur Überprüfung, ob ein erkanntes Hindernis noch existiert. Falls nicht, wird es gelöscht.
- *robot_radius: 0.25* - Gibt den Radius des Roboters in m an. Es kann wahlweise auch ein genauer *footprint* spezifiziert werden, falls die Roboterplattform nicht kreisförmig ist.
- *inflation_radius: 0.35* - Dieser Parameter ist für die Kollisionsvermeidung zuständig. Es wird der Radius für den Abstand zu Hindernissen in m angegeben. Der Robotermittelpunkt wird einen Abstand von 0.35 m zu Hindernissen halten. Der *inflation_radius* muss größer als der *robot_radius* sein, damit Kollisionen vermieden werden.
- *observation_sources: laser_scan_sensor_tim laser_scan_sensor_kinect* - Gibt die Sensoren an, die für die Hinderniserkennung genutzt werden sollen. Die Sensoren müssen genauer spezifiziert werden. Dies geschieht durch die Definition der Variablen, die hinter dem Doppelpunkt aufgezählt sind.
  - *laser_scan_sensor_tim: {sensor_frame: laser, data_type: LaserScan, topic: scan, marking: true, clearing: true, inf_is_valid: true}*
  - *laser_scan_sensor_kinect: {sensor_frame: outframe, data_type: LaserScan, topic: scan_kinect, marking: true, clearing: true, inf_is_valid: true}*

    Der Variablenname *laser_scan_sensor_xxx:* beinhaltet über die Definition in den geschweiften Klammern, welcher Sensor benutzt werden soll.
    - *Sensor_frame* - Gibt das Frame im *tf-Tree* an.
    - *data_type* - Gibt den Nachrichtentyp an.
    - *topic* - Gibt den Namen der Topic des Sensors an.
    - *marking* - Bedeutet, dass der Sensor Hindernisse in die Karte eintragen darf.
    - *clearing* - Bedeutet, dass der Sensor Hindernisse aus der Karte entfernen darf.
    - *inf_is_valid: true* - Werte, die in der Laserscan-Nachricht undefiniert sind, da nichts erkannt worden ist, werden auf die maximale Distanz des entsprechenden Sensoren gesetzt.

Für die globale Karte gelten die Parameter:

- *global_frame: /map* - Gibt das Frame an, welches die reelle Umgebung widerspiegelt, in der sich der Roboter bewegt.

- *robot_base_frame: /base_link* - Gibt das Frame des Robotermittelpunkts an.
- *updatefrequency: 5.0* - Gibt die Updatefrequenz der globalen Karte in Hz an.
- *publish_frequency: 1.0* - Gibt die Frequenz der globalen Karte in Hz an, mit der die Karte für eine Visualisierung veröffentlicht werden soll.
- *static_map: true* - Gibt an, dass es sich um eine statische Karte handelt.

Für die lokale Karte gelten die Parameter:

- *global_frame: /odom* - Gibt das Frame an, welches für die lokale Karte den Ursprung bildet.
- *robot_base_frame*, *update_frequency* und *publish_frequency* haben die gleichen Parameter wie in der globalen Karte.
- *static_map: false* und *rolling_windows: true* - Gibt an, dass es sich nicht um eine statische Karte handelt. Bewegt sich der Roboter, so wird die Karte entsprechend mit verschoben. Der Roboter bildet immer den Ursprung der Karte.
- *width: 4.0*, *height: 4.0* und *resolution: 0.05* - Geben die Auflösung der Karte in m an.

Abbildung 3.9 zeigt eine Karte, die mit Hilfe von *costmap_2d* erstellt wurde. In der Abbildung stellen die schwarzen Linien und Punkte Hindernisse dar, die durch den Laserscanner und die Kinect erkannt wurden. Um jedes Hindernis wird ein Kreis mit dem angegebenen Radius gezogen (Parameter: *inflation_radius*), der als Kollisionsschutz dient. In der Navigation wird der Mittelpunkt des Roboters später die Begrenzungen meiden und nur in dem Bereich fahren, welcher nicht durch einen Kollisionsschutz bedeckt ist. Die roten Punkte innerhalb der Abbildung stellen Hindernisse dar, die durch die Sensoren erkannt werden.

3.4.2. global_planner

Der *global_planner* berechnet einen Weg zu einem gewählten Punkt X auf Grundlage der globalen costmap [47]. Der Punkt X wird vom Benutzer gewählt. Als Algorithmen können der Dijkstra- oder der A* Algorithmus (2.4.9) genutzt werden. Für diese Arbeit wurde der Dijkstra-Algorithmus verwendet. Der errechnete Pfad wird in Form einer *nav_msgs/Path* Nachricht auf der Topic */plan* veröffentlicht. Die Nachricht *nav_msgs/Path* enthält ein Array mit verschiedenen Positionen, welche aus Punkt und Ausrichtung des Roboters bestehen. Der Pfad wird also über verschiedene Positionen beschrieben und kann so abgefahren werden.

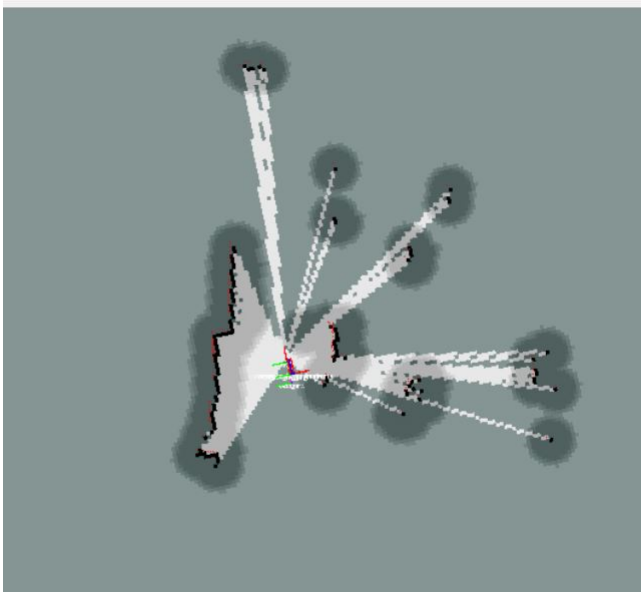

Abbildung 3.9.: Startposition des Teleoperators aus Sicht der costmap.

3.4.3. base_local_planner

Das Package *base_local_planner* erstellt aus dem berechneten globalen Pfad und der lokalen costmap Kommandos für die mobile Einheit. Dafür stellt es einen Controller bereit, der die lokale costmap als Basis nimmt und dort den globalen Pfad hineinprojiziert. Es wird für jeden Punkt des Pfades eine mögliche Bahnkurve errechnet, die aus einer bestimmten Geschwindigkeit und einer Rotation besteht. Für die Bahnkurven wird intern simuliert, ob diese zu einer Kollision mit einem auf der costmap markierten Hindernis führen würden. Falls ja, wird eine neue Bahnkurvenberechnung durchgeführt. Bei einer geeigneten Bahnkurve werden die Geschwindigkeit und Rotation in Form einer *geometry_msgs/Twist*-Nachricht auf der Topic *cmd_vel* veröffentlicht [19]. Auf dieser Topic „lauscht“ *ROSARIA* und setzt die dort veröffentlichte Nachrichten in Bewegung des Roboters um. Es werden folgende Parameter verwendet:

- *controller_frequency: 10.0* - Gibt die Frequenz in Hz an, mit der der Controller die *geometry_msgs/Twist*-Nachrichten veröffentlicht.
- *max_vel_x: 0.0* - Gibt die maximale Geschwindigkeit in m/s an. Es wurde hier 0 m/s gewählt, da dieser Wert von dem Benutzer bestimmt werden soll.

- *min_vel_x: 0.0* - Gibt die kleinste Geschwindigkeit an, mit der der Roboter fahren soll. Hier wurde ebenfalls 0 m/s gewählt, da dieser Wert kleiner gleich dem Wert *max_vel_x* sein muss.
- *max_rotational_vel: 0.5* - Gibt die maximale Rotationsgeschwindigkeit des Roboters in rad/s an.
- *min_in_place_rotational_vel: 0.4* - Gibt die kleinste Rotationsgeschwindigkeit des Roboters in rad/s an.
- *acc_lim_theta: 3.2* - Gibt die obere Grenze der Beschleunigung der Rotation in m/s^2.
- *acc_lim_x: 2.5* - Gibt die maximale Beschleunigung in x-Richtung in m/s^2 an.
- *acc_lim_y: 2.5* - Gibt die maximale Beschleunigung in y-Richtung in m/s^2 an.
- *xy_goal_tolerance: 0.5* - Gibt die erlaubte Abweichung in x- und y-Richtung des Roboters vom Ziel an.
- *yaw_goal_tolerance: 6.29* - Gibt die erlaubte Abweichung in rad der Rotation des Roboters im Ziel an. 6.29 rad entsprechen ca. 360 °.
- *holonomic_robot: false* - Gibt an, ob die Geschwindigkeiten für eine holonome[6] oder eine nicht holonome Plattform generiert werden sollen.
- *sim_time: 2.0* - Gibt die Zeit in s an, die für die interne Simulation der Trajektorien (Bahnkurven) benötigt werden darf.
- *meter_scoring: true* - Gewährt Robustheit gegenüber Veränderung der Auflösung der costmap, da Parameter der Veränderung angepasst werden.

3.4.4. move_base

Bis jetzt wurden drei verschiedene Packages erläutert, die drei unterschiedliche Funktionen mitbringen, die in Kombination miteinander eine funktionierende Navigation liefern. Um die Packages bzw. die darin enthaltenen Nodes miteinander zu verknüpfen, gibt es das *move_base* Package. Die einzelnen Nodes werden so miteinander vernetzt, dass navigiert werden kann. *Move_Base* implementiert unter anderem Akionen, mit denen einfache Befehle, wie z.B. "Fahre zu Punkt X", ausgeführt werden können. Hierfür wird der *actionlib*-Stack verwendet. Die

[6] Holonom - Holonom bedeutet vereinfacht auf Roboter bezogen, dass jeder Punkt in einem 2D-Raum ohne Rotationsbewegung angesteuert werden kann [61].

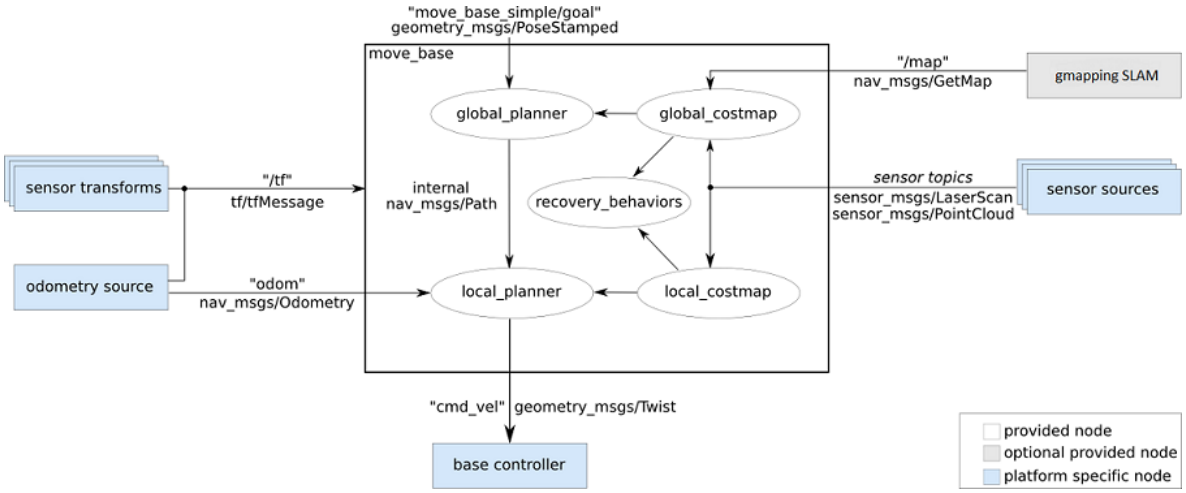

Abbildung 3.10.: Komponenten der Navigation. Originalabbildung:[48]

genaue Funktionsweise des *actionlib*-Stacks kann unter [20] nachgelesen werden. Für diese Arbeit wurde *MoveBaseClient* verwendet, hinter dem die Implementierung des *actionlib* steckt [48].

Abbildung 3.10 gibt einen Überblick über alle Komponenten der Navigation und deren Verknüpfung. Die zentrale Einheit zur Navigation bildet *move_base*. Über die durch SLAM erstellte Karte (grau eingezeichnet) werden mit Hilfe der Sensoren (*sensor sources*, blau eingezeichnet) die beiden costmaps erstellt. An Hand der Odometriedaten (*odometry source*, blau eingezeichnet) kann der *base_local_planner* (*local_planner*) den vom *global_planner* erstellten Pfad für die lokale costmap simulieren und entsprechende Steuerbefehle an den *base_controller* (blau eingezeichnet) senden. Die *sensor_transforms* (blau eingezeichnet), die über *tf* veröffentlicht werden, sind für die Transformationen aller Sensoren in die entsprechenden Frames (Koordinatensysteme) notwendig. Akzeptierte Eingaben zur Navigation mit dem *move_base*-Package bzw. *Naviagtion Stack* sind Nachrichten in Form von *geometry_msgs/Twist* oder per Action-Client. Der Action-Client ist eine der oben erwähnten Implementierung des *actionlib*-Stacks. Wird der Teleoperator in eine Situation gebracht, in der der *Navigation-Stack* entscheidet, dass die mobile Einheit sich festgefahren hat, wird auf die *recovery_behaviors* zurückgegriffen. Festgefahren ist die mobile Einheit, wenn ein zuvor berechneter Weg zur Laufzeit nicht abgefahren werden kann, da dynamische Hindernisse den Weg versperren oder auftretende Messefehler den Weg als zu eng kommunizieren. Um aus einer solchen Situation zu gelangen, wird folgendes Prozedere angewendet:

1. Rotation auf der Stelle, um ggf. einen Ausweg zu finden.
2. Löschen der nicht sichtbaren Hindernisse auf der costmap und erneute Rotation auf der Stelle. Es wird eine Neuaufnahme der Umgebung durchgeführt.
3. Scheitern der vorherigen Schritte, liefert eine interne Fehlermeldung, dass der Zielpunkt nicht erreicht werden kann. Der Benutzer wird über die Topic *rosout* informiert, dass die Navigation zu dem gewählten Punkt X abgebrochen wurde.

3.5. Modulare-Softwarearchitektur

In den vorherigen Abschnitten wurde beschrieben, aus welchen Teilen die komplette Navigation besteht. Es wäre nun möglich, mit der mobilen Einheit autonom zu navigieren. Die Problembestellung besagt jedoch, dass ein teilautomes System entwickelt werden soll. Die im Folgenden vorgestellte Softwarearchitektur soll die Möglichkeit bieten, im zukünftigen Verlauf der Entwicklung des Telepräsenzsystems MoRy-A autonom fahren zu können. Es wurde in der Architektur darauf geachtet, dass der Benutzer entweder komplett auf sich gestellt und mit voller Kontrolle die mobile Einheit bewegen kann, oder durch Zuhilfenahme des Fahrerassistenzsystems. Die Abbildung 3.11 stellt den modularen Aufbau der Softwarearchitektur dar. Für das Fahrerassistenzsystem bildet der *Navigation Stack* die Hauptkomponente. Alle

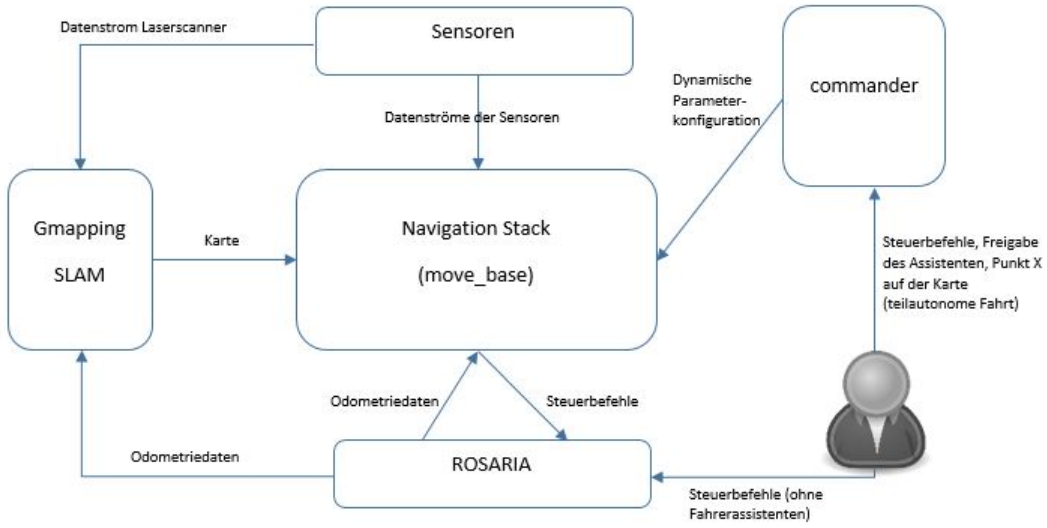

Abbildung 3.11.: Modulare Softwarearchitektur.

Module, die mit der Hauptkomponente verbunden sind, können ausgetauscht werden, solange das ausgetauschte Modul die entsprechende Schnittstelle bedienen kann. In dem Modul der Sensoren wäre es zum Beispiel möglich, andere Sensoren, als die in der jetzigen Konfiguration verwendeten, einzusetzen. Es könnten beispielsweise auch andere Packages als das SLAM *gmapping* verwendet werden. Außerdem wäre es denkbar, mit bekannten Karten zu arbeiten. Der Grundriss eines Gebäudes oder Stockwerks könnte dem *Navigation Stack* über einen *Map Server* [7] zur Verfügung gestellt werden. Mit Hilfe von *amcl*[7] wäre es ROS möglich sich in der vorhandenen Karte zu lokalisieren.
Wie in Abbildung 3.11 zu sehen ist, kann der Benutzer über zwei Wege Steuerbefehle an die mobile Einheit senden. Es besteht die Möglichkeit, direkt und ohne Unterstützung des Fahrerassistenzsystem die Plattform über *ROSARIA* anzusteuern, oder über das Modul *commander* das Fahrerassistenzsystem mit einzubeziehen. Die Teilautonomie ist im Package *commander* gekapselt, das im Abschnitt 3.6 näher erläutert wird. Wird das Package aus dem modularen Aufbau entfernt, könnte der Roboter voll autonom fahren. Es müssten lediglich die Parameter der maximalen Geschwindigkeit vor der autonomen Fahrt gesetzt werden (siehe Parameter *max_vel_x* in 3.4.3). Außerdem müsste dem Roboter durch den Benutzer ein Punkt X mitgeteilt werden, zu dem autonom gefahren werden soll. Grenzen des vorliegenden Systems sind im Bereich der Sicherheit des Roboters (Safety) aufzuzeigen. Möchte der Benutzer ohne Unterstützung fahren, ist zurzeit keine Kollisionserkennung bzw. Kollisionsvermeidung möglich. Ein möglicher Lösungsansatz wäre, die costmap des bei nicht teilautonomer Fahrt im Hintergrund laufenden *Navigation Stack* abzugreifen, um eine Anwendung zu implementieren, die den Benutzer vor Kollisionen warnt. Der Entwurf einer Kollisionswarnung für nicht teilautonomes Fahren war nicht Teil der Fragestellung dieser Arbeit, kann aber bei Bedarf einfach implementiert werden und als Modul der Architektur hinzugefügt werden. Der Benutzer muss, bevor teilautonom navigiert werden soll, der mobilen Einheit die entsprechende Berechtigung geben. Gerät die mobile Einheit in eine unerwartete Situation, kann der Benutzer die Berechtigung sofort entziehen und die Plattform direkt ansteuern. Kommt es zu einer Gefahrensituation, die nicht durch das System erkannt wird, kann manuell ein *Reset* der Plattform durch Drücken eines Tasters an der Plattform durchgeführt werden. Der *Reset* bewirkt, dass der Roboter stehen bleibt. Anschließend müssen allerdings alle in der Architektur gezeigten Module neu gestartet werden. Ein möglicher Lösungsansatz wäre eine Routine einzubauen, die automatisch die Module neu startet.
Die Sicherheit des Systems gegenüber mutwilliger Beschädigung oder Diebstahl ist nur in geringer Form vorhanden. Die Sensoren sind fest mit der mobilen Einheit verschraubt. Somit ist

[7] amcl - amcl ist ein Package, welches zur Lokalisierung in bekannten Karten dient. [29]

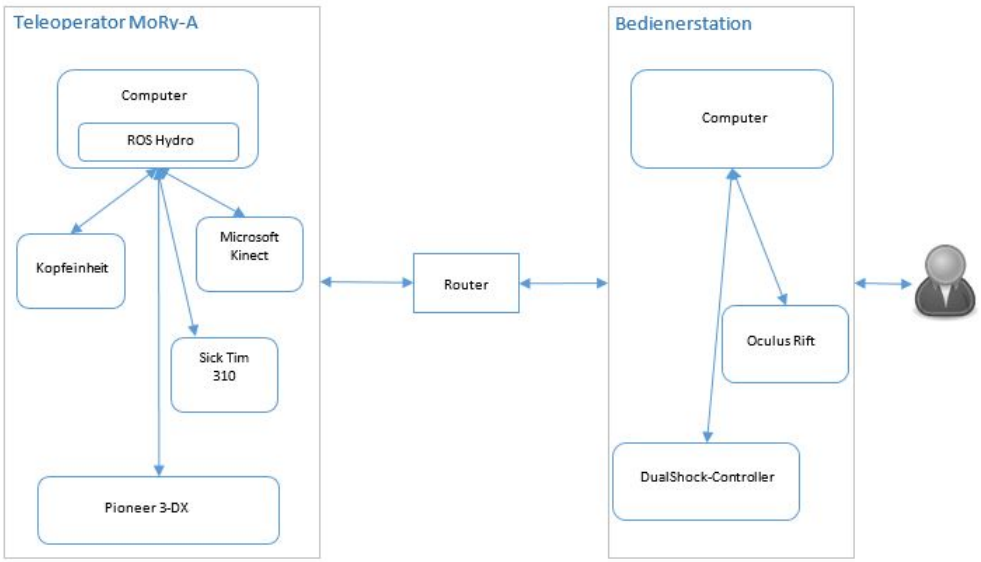

Abbildung 3.12.: Übersicht der Systemkomponenten des Telepräsenzsystems.

ein einfacher Diebstahl zumindest nicht ohne Werkzeug möglich. Im Bereich der Software gibt es kaum Sicherheitsmechanismen. Wird sich mit dem entsprechenden *rosmaster* verbunden, können auf die vorliegenden *Topics* eigen produzierte Nachrichten veröffentlicht werden. Das System kann somit auch von fremder Hand übersteuert werden. Bis jetzt gibt es noch keinen Sicherheitsmechanismus gegen solch ein Fremdeinwirken.
Im Anhang B befindet sich eine Übersicht über die verwendeten Topics und wie sie miteinander verbunden sind. Der Graph wurde nach Umsetzung des Entwicklungsprozesses von ROS generiert. Die Topics von *costmap_2d*, *base_local_planner* und *global_planner* sind auf dieser Übersicht nicht ersichtlich, da sie durch *move_base* miteinander verknüpft werden und daher unter der entsprechenden Topic von *move_base* wiederzufinden sind. Der Vollständigkeit halber wird in Abbildung 3.12 die Ergänzung zur Abbildung 3.1 gezeigt, die den Basisaufbau des Teleoperators beschreibt. Die Abbildung wurde um die beiden verwendeten Sensoren erweitert und stellt nun alle Komponenten des Systems dar.

3.6. commander - Modul der Teilautonomie

Im letzten Abschnitt wurde die Softwarearchitektur des Teleoperators beschrieben. Es wurde beschrieben, dass durch den modularen Aufbau eine autonome Fahrt möglich wäre, wenn das Modul des *commanders* entfernt werden würde. In diesem Abschnitt wird das Modul näher erläutert. *Commander* bildet sowohl den Modulnamen in der Architektur, als auch den Packagename der entwickelten ROS-Nodes. In dem Package sind zwei unterschiedliche Nodes vorhanden. Der erste Node (*pointnavigator*) ist für die Erstellung einer Nachricht zuständig, die, um zu einem gewählten Punkt zu navigieren, an *move_base* gesendet wird. Dazu registriert sich der *pointnavigator* auf der Topic *clicked_point* und wartet, bis dort Nachrichten veröffentlicht werden. Die Topic *clicked_point* muss durch einen anderen Node bereitgestellt werden. Wird über ein Visualisierungstool, wie zum Beispiel *Rviz*[8], ein Punkt auf der Karte angewählt, wird eine Nachricht erzeugt und auf der von *Rviz* bereitgestellten Topic *clicked_point* veröffentlicht. Der *pointnavigator* empfängt die Nachricht und verarbeitet sie, indem der Punkt aus der Nachricht in eine *MoveBaseGoal*-Nachricht verpackt und über einen Action-Client an *move_base* gesendet wird. Der Action-Client, sowie die *MoveBaseGoal*-Nachricht sind in *move_base* über die in 3.4.4 angesprochene *actionlib* implementiert. Mit Erhalt der *MoveBaseGoal*-Nachricht beginnt der *global_planner* einen möglichen Weg zu dem gewünschten Punkt zu errechnen.
Der zweite Node ist mit dem Modulnamen, sowie dem Packagenamen namentlich identisch. Der Node (commander) beinhaltet die Funktionsweisen der Überwachungssensoren sowie der teilautonomen Fahrt. Die Überwachung der Sensoren funktioniert auf ROS-Ebene. Die Kinect, die über das Package *depthimage_to_laserscan* einen Laserscan auf die Topic *scan_kinect* veröffentlicht, wird dadurch überwacht, dass die Topic zyklisch nach neuen Nachrichten abgefragt wird. Erhält die Topic keine neuen Nachrichten, bedeutet das, dass entweder die Konvertierung vom *sensor_msgs/Image* zu *LaserScan* nicht mehr funktioniert, oder die Kinect keine Daten mehr liefert. Da bei Eintreten einer der genannten Fälle keine Höhe mehr erkannt werden kann, entsteht somit eine potentielle Gefahr für den Teleoperator. Daher wird dieser durch Abschalten des Fahrerassistenten angehalten. Durch das Ausschalten des Assistenten wird nach Ausfall eines oder mehrerer Sensoren gewährleistet, dass diese bei erneuter teilautonomer Fahrt wieder einsatzbereit sind. Das Starten des Assistenten ist bei einem Defekt eines Sensors nicht möglich. Der Laserscanner ist in einer ähnlichen Weise gesichert. Dieser gibt bei einem Verbindungsverlust eine Fehlermeldung auf der Konsole aus. Dies wurde sich zu Nutze gemacht, indem an der Stelle im Sourcecode eine ROS-Nachricht mit einem Boolean generiert wird, an der die Fehlernachricht auf die Konsole geschrieben wird. Die generierte Nachricht wird auf

[8] Rviz - Rviz ist von ROS mitgebrachtes Tool zur Visualisierung.

der Topic *ctrl/sensorsalive* veröffentlicht. Enthält die Nachricht den Wert *true*, wird ebenfalls der Fahrerassistent abgeschaltet und die mobile Einheit angehalten.
Der Aspekt der Teilautonomie wird über den *commander*-Node realisiert. Der Hintergrund ist die Vorstellung, dass der Teleoperator zu einem späteren Zeitpunkt autonom fahren wird. Um dies zu erreichen, darf nicht in den natürlichen Ablauf des *Navigation Stacks* eingegriffen werden. Um ungewollte Nebeneffekte zu vermeiden, sollten keine zusätzlichen Nachrichten mit Steuerbefehlen auf die vom *Navigation-Stack* verwendeten Topics veröffentlicht werden. Daher werden über den *Commander* zwei Topics bereitgestellt, die vom Benutzer verwendet werden können, um dort die Steuersignale, sowie die Freigabe für eine teilautonome Fahrt zu veröffentlichen. Über die Topic *ctrl/auth* empfängt der *commander* Nachrichten, die einen Boolean enthalten. Ist der Boolean gleich *true* wird die teilautonome Fahrt erlaubt und die Steuerbefehle verarbeitet, die über *ctrl/vel* empfangen werden. Aus der Nachricht, die die Steuerbefehle enthält, wird die Geschwindigkeit in der Bewegung nach vorne extrahiert. Anschließend wird dynamisch der Parameter *max_vel_x* des *base_local_planner* auf die extrahierte Geschwindigkeit gesetzt. Dies hat zur Folge, dass der Teleoperator die Geschwindigkeit fährt, die der Benutzer steuert. Es kann beim teilautonomen Fahren allerdings zu Situationen kommen, in denen die vom Operator gewünschte Geschwindigkeit nicht erreicht oder gehalten werden kann, wie zum Beispiel bei der Fahrt durch enge Kurven. In diesen Situationen regelt der *base_local_planner* die Geschwindigkeit herunter. Dies ist ein weiterer Aspekt der Sicherheit (Saftey), da der Benutzer unter Verwendung des Fahrerassistenzsystems den Roboter nicht in Geschwindigkeiten bringen kann, mit dem die steuernde Software nicht ausführbar ist. Es ist mit dem *commander* also möglich, eine gewünschte Maximalgeschwindigkeit zu setzen, die gefahren wird, wenn es die Situation zulässt.

3.7. Inbetriebnahme des Fahrerassistenten

Um den entwickelten Fahrerassistenten in Betrieb zu nehmen, müssen folgende Voraussetzungen geschaffen werden:

- Der zum Telepräsenzsystem zugehörige Wireless LAN Router ist gestartet und das Netzwerk MoRy-A steht zur Verfügung.
- Der Intel NUC und die Kinect sind an eine Stromquelle (vorzugsweise eine Kabeltrommel) angeschlossen (der NUC kann über die Akkumulatoren betrieben werden, für die Kinect fehlt eine solche Vorrichtung).
- Die Akkumulatoren sind aufgeladen in der Plattform angeschlossen.

- Der Intel NUC ist gestartet und es wurde zwischen der Bedienerstation und dem Intel NUC eine Verbindung mit dem Befehl *ssh tpr@192.168.1.100* hergestellt (Passwort: tpr).
- Die Plattform Pioneer 3-DX ist eingeschaltet.
- Der Laserscanner ist mit den Akkumulatoren verbunden.
- Kinect und Laserscanner sind per USB an den NUC angeschlossen (über USB-Hub)

Der *roscore* wird gestartet und anschließend in einem neuen Terminal (Verbindungsherstellung über ssh) der Befehl *roslaunch pioneer_2dnav pioneer_bringup.launch* aufgerufen. *Roslaunch* ist ein Tool, um mehrere Nodes aus gleichen oder verschiedenen Packages zu starten. Dazu wird ein *.launch*-File geschrieben, das die gewünschten Nodes und deren Packagenamen enthält. Die genaue Verfahrensweise zum Erstellen einer *.launch*-File kann unter [70] nachgelesen werden. Das *.launch*-File startet alle Nodes, die der Fahrerassistent für die Navigationsunterstützung benötigt. Hierzu zählen:

- *ROSARIA*
- Der Laserscanner Sick TIM310 (Node: *sick_tim310s01*)
- Die Microsoft Kinect
- *Depthimage_to_laserscan*
- *Static_transform_publisher* für die Transformationen von der Kinect zu *base_link* (Robotermittelpunkt), Laserscanner zu *base_link* und Laserscan der Kinect zu *base_link*
- *slam_gmapping* zur Kartierung und Lokalisierung
- *commander* des Packages *commander*

In einer weiteren Konsole (ssh) wird die Navigationsunterstützung mit dem Befehl *roslaunch pioneer_2dnav move_base_isr_map.launch* gestartet. Das *.launch*-File startet die Nodes:

- *pointnavigator* aus dem Package *commander*
- *move_base* mit den Parametern von *costmap_2d*, *base_local_planner* und *global_planner*.

Die *.launch*-Files sind Teil des Workspace, welcher im Anhang D genauer beschrieben wird. Der Fahrerassistent ist nun bereit, den Benutzer zu unterstützen. Um zu einem gewählten Punkt X zu kommen, muss der Bediener über ein geeignetes Visualisierungstool einen Punkt aus der

Karte wählen und diesen auf der Topic *clicked_point* (3.6) veröffentlichen. Ein geeignetes Tool hierfür wäre *Rviz*, da es sowohl die Möglichkeit der Visualisierung bietet, als auch die, einen Punkt aus der angezeigten Karte zu wählen. Der Fahrerassistent wurde bei der Entwicklung mit *Rviz* getestet. Andere Visualisierungstools für ROS wurden nicht eingesetzt und getestet. Um *Rviz* verwenden zu können, muss sich dieses mit dem *rosmaster* auf dem Intel NUC verbinden. Hierfür muss eine Konfiguration vorgenommen werden, bevor der *roscore* auf dem NUC gestartet wird.

1. Verbinden mit dem Intel NUC per ssh
2. Im Terminal werden folgende Befehle ausgeführt:
   a) *export ROS_MASTER_URI=http://192.168.1.100:11311*
   b) *export ROS_IP=192.168.1.100* (IP des Teleoperators)
3. Im Terminal werden folgende Befehle auf dem lokalen Computer (Bedienerstation) ausgeführt:
   a) *export ROS_MASTER_URI=http;//192.168.1.100:11311*
   b) *export ROS_IP=192.168.1.105* (IP der Bedienerstation. Diese kann variieren und muss nicht der angegeben entsprechen)

Anschließend kann nach dem Start des *roscore* und der *.launch*-Files *Rviz* auf der Bedienerstation aufgerufen werden. Es können die Daten der Navigationsunterstützung visualisiert werden. Eine geeignete Konfiguration für *Rviz* befindet auf dem Intel NUC unter */home/t-pr/.rviz/navstack.rviz*.

3.8. Fahrergebnisse

In diesem Abschnitt werden die erzielten Fahrergebnisse, sowie die Probleme, die aufgetreten sind, näher erläutern und diskutiert. Unter optimalen Voraussetzungen funktioniert die Navigationsunterstützung wie gewünscht. Es wird auf kürzestem Weg zu einem gewählten Punkt navigiert. Die Geschwindigkeit wird von dem Benutzer bestimmt und Kollisionen werden vermieden. Optimale Voraussetzungen sind gegeben, wenn keine Hindernisse während der Fahrt im Bereich des errechneten Pfades auftreten. Des Weiteren gibt es keine Störfaktoren für die Sensoren, wie z.B. Sonnenlichteinstrahlung oder andere IR-Quellen. Sonnenlichteinstrahlung bewirkt, dass die Kinect nahezu „blind“ wird, da der IR-Sensor der Kinect nicht zwischen Sonnenlicht und eigener IR-Strahlung unterscheiden kann. Durchsichtige Objekte

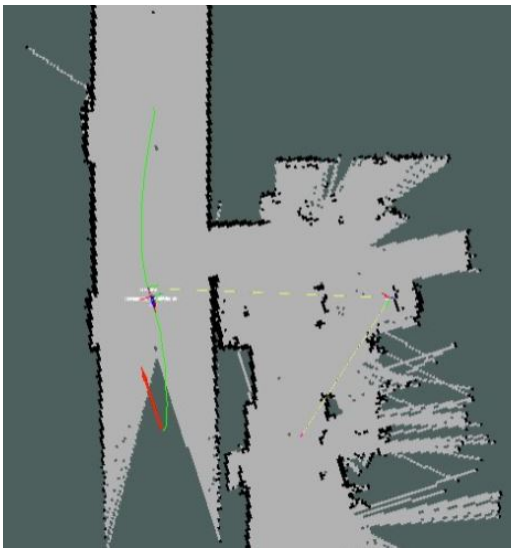

Abbildung 3.13.: Abfahren der berechneten Wegstrecke.

könnten vom Laserscanner nicht erkannt werden und sind eine weitere Störgröße. Da diese Objekte nicht erkannt werden, kann ein Pfad errechnet werden, der durch diese Objekte führt. Diese werden erst erkannt werden, wenn sie in den Sichtbereich der Kinect geraten. Ohne Störfaktoren wird folgendes Ergebnis erzielt:
Es wird durch den Bediener die teilautonome Fahrt freigegeben und ein Punkt X auf der angezeigten Karte bestimmt. Anschließend gibt der Bediener über sein Eingabegerät die nötigen Steuerbefehle für die Bewegung nach vorne. Die mobile Einheit bestimmt einen Weg, der anschließend unter Berücksichtigung von Hindernissen abgefahren wird. Ist die mobile Einheit am Zielpunkt angekommen, bleibt sie stehen und wartet entweder auf weitere Befehle oder auf Übernahme der Kontrolle durch den Benutzer. Abbildung 3.13 zeigt die Navigation zu einem von dem Benutzer gewählten Punkt X. Die grüne Linie stellt den durch die Wegplanung errechneten Pfad dar, der von der mobilen Einheit abgefahren wird. In der gezeigten Abbildung ist zu erkennen, dass an einigen Stellen ein Bereich hinter der Wand fälschlicherweise erkannt wurde. Dies geschieht durch fehlerhafte Messungen des Laserscanners, welche aber im Verlauf der Bewegung durch Aktualisierungen der Karte korrigiert werden.
In Abbildung 3.14 ist die Korrektur sichtbar. Die Bereiche, die durch Fehlmessungen verursacht wurden, sind nicht mehr sichtbar. In dieser Abbildung ist wieder der errechnete Weg eingezeichnet. Hier zeigt sich, dass der errechnete Pfad ohne ersichtlichen Grund einen Bogen schlägt. Dies ist durch Lichtverhältnisse und Reflexion durch die Deckenbeleuchtung des Flurs

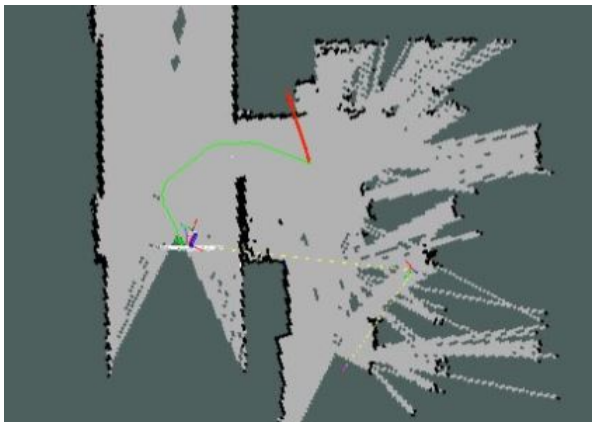

Abbildung 3.14.: Wegberechnung durch Reflexion beeinflusst.

zu erklären. Die Sensoren nehmen hier fälschlicherweise Reflexion wahr und belegen die costmap an dieser Stelle mit hohen Kosten, was einem Hindernis entspricht. Daher ist der effizienteste Weg nach dem *global_planner*, die Störung zu umfahren. Es entsteht eine nicht optimale Route, die aber nicht die Funktion des Fahrerassistenten außer Kraft setzt.
In Abbildung 3.15 hingegen ist ein Fehlerfall dargestellt, der den Fahrerassistenten unbrauch-

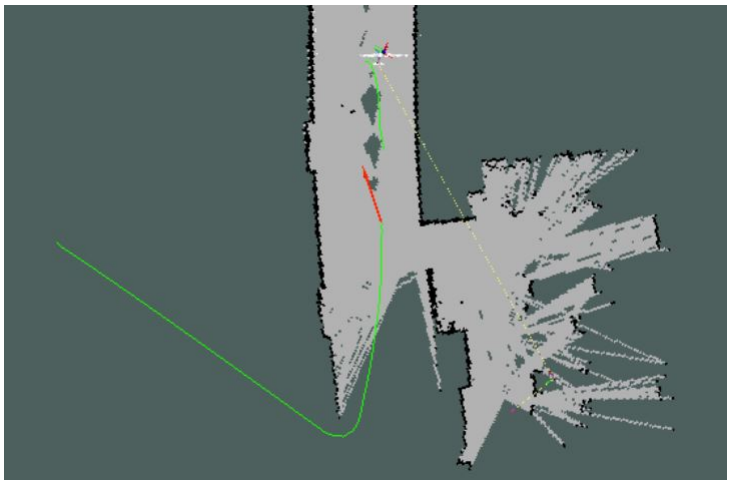

Abbildung 3.15.: Fehlerfall, ausgelöst durch falsche Messdaten von Kinect und Laserscanner.

bar macht. Durch falsche Werte der Odometrie, die durch Schlupf erzeugt werden, sowie durch Lichtverhältnisse, die falsche Messerwerte ergeben, verliert das SLAM die Orientierung. Es war zu beobachten, dass der Roboter innerhalb der Karte von dem Pfad immer weiter abwich

und der Pfad durch unrealistische Wege immer wieder neu berechnet und dargestellt wurde. Die Navigationsunterstützung musste neu gestartet werden. Es ist festzuhalten, dass bei fehlerhafter Ausführung von SLAM der Fahrerassistent unbrauchbar wird und übersteuert werden muss. Kleinere Fehler, die vereinzelt auftraten, konnten durch die Software korrigiert werden. Beispiele hierfür waren unter anderem Schlupf der Räder oder Reflexionen bzw. Sonneneinstrahlung. Solange nur einer der beiden Entfernungsmesssensoren von den Störfaktoren betroffen war, gab es immer eine Möglichkeit, zum gewählten Punkt zu gelangen. Der Schlupf wurde durch Landmarken (2.4.8) korrigiert und die Reflexion, sowie Sonneneinstrahlung wurden durch Abwenden bzw. die Fortsetzung der Fahrt nach Verlassen des Störfeldes durch das Aktualisieren der Karte korrigiert.
Bei Ausfall eines Sensors, zum Beispiel durch das Ziehen der Verbindung zum Intel NUC, blieb die mobile Einheit stehen und war nur noch vom Benutzer direkt steuerbar.

4. Fazit und Ausblick

Ziel dieser Arbeit war es, eine Navigationsunterstützung für das Telepräsenzsystem MoRy-A zu entwickeln. Dabei lag der Schwerpunkt auf der sensorischen Erfassung der Umwelt und der Kombination der Sensoren. Es sollte zu einem Punkt X navigiert werden, der zuvor vom Benutzer auf einer im Laufe der Fahrt generierten Karte gewählt wurde. Zudem wurden die Aspekte der Sicherheit der mobilen Einheit im Bezug auf Safety und Security untersucht.
Es wurden verschiedene Sensortypen auf ihre Eigenschaften und ihren Einsatzzweck im Rahmen dieser Arbeit untersucht. Ein Laserscanner der Firma Sick sowie die Microsoft Kinect (Version 1, entwickelt für die Spielekonsole XBOX 360) standen an der HAW Hamburg zur Verfügung. Das an der Plattform Pioneer 3-DX (3.1.1) verbaute Sonar wurde aufgrund von negativen Eigenschaften nicht verwendet (siehe 2.4.4 und 3.2). Die für die Navigationsunterstützung relevanten Softwarekomponenten wurden in einer modularen Systemarchitektur angeordnet, mit der es möglich ist, die mobile Einheit um Sensoren oder Algorithmen zu erweitern oder gegebenenfalls dem technischen Stand anzupassen. Die Basis innerhalb der Systemarchitektur ist ROS (3.1.2). Über ROS wird die Kommunikation zwischen den einzelnen Modulen abgewickelt. Die einzelnen Softwaremodule, die eingesetzt wurden, gliedern sich in folgende Punkte:

- Ansprechen der Sensoren und Verpacken der Informationen in ROS-Format.
- Kartierung und Lokalisierung der Umgebung.
- Erstellen einer Karte, die befahrbare und nicht befahrbare Bereiche mittels Kosten definiert.
- Anhand der Karte mögliche Routen errechnen.
- Erzeugen der Steuersignale für die Plattform anhand der errechneten Route.
- Verknüpfung der Softwarekomponenten.
- Auswerten der Steuersignale des Benutzers, dynamische Rekonfiguration der Geschwindigkeit (Teilautonmie), Sensorüberwachung und Schnittstelle für Zielpunkt Eingabe.

Durch die verwendeten Softwarekomponenten und Sensoren konnte eine Navigationsunterstützung erreicht werden. Der Benutzer wird nach Aktivierung des teilautonomen Betriebes bei der Steuerung der mobilen Plattform in der Bewegung nach links oder rechts und in der Kollisionsvermeidung unterstützt. Die Aspekte der Sicherheit wurden im Bereich Safety durch eine Kollisionsvermeidung, eine Noteinrichtung in Form eines Reset-Buttons an der mobilen Einheit und eine Freigabeeinrichtung für teilautonomes Fahren auf Softwareebene gelöst. Der Aspekt der Security wurde durch festes Verschrauben der abnehmbaren Sensoren gelöst. Ein Schutz auf Softwareebene ist nicht vorhanden. Als Erweiterung könnte hier in zukünftigen Arbeiten ein Mechanismus zur Authentifizierung zum Schutz vor unbefugtem Eingreifen in die Kommunikation entwickelt werden. Eine weitere Möglichkeit wäre, die gesamte Kommunikation innerhalb eines Virtual Private Network (VPN) abzuwickeln. Mit der Navigationsunterstützung konnten Ergebnisse einer fehlerfreien Fahrt zu einem vom Benutzer ausgewählten Punkt X erzielt werden. Sobald Störgrößen mit eingezogen wurden, war die Navigationsunterstützung nur teilweise nutzbar. Lieferte ein Sensor für kurze Zeit fehlerhafte Messwerte, konnte trotzdem weiter navigiert werden. Allerdings waren Umwege und nicht die direkte Ansteuerung des Punktes die Folge. Waren beide Sensoren betroffen, war die Navigationsunterstützung unbrauchbar (siehe 3.8). In weiteren Arbeiten an dem Telepräsenzsystem könnten die Aspekte der Fehlerminimierung im Vordergrund stehen. Die Microsoft Kinect sollte durch ihren Nachfolger, der für die Spielekonsole Xbox One ([89]) entwickelt wurde, ersetzt werden. Der Nachfolger der Kinect hat den Vorteil, dass das Infrarotmuster, das zur Entfernungsmessung genutzt wird, mit einer bestimmten Frequenz projiziert wird. Dadurch können Fehler, die durch andere Infrarotquellen erzeugt werden, eliminiert bzw. reduziert werden. Bei der momentan verbauten Kinect wird der Tiefensensor durch andere Infrarotquellen (z.B. Sonnenlicht) gestört. Die Kinect ist teilweise „blind“, da nicht zwischen eigener IR-Strahlung und fremder unterschieden werden kann. Es könnte in Zukunft auch auf die Multisensorlösung verzichtet werden. Durch geeignete Montage eines schwenkbaren 2D-Laserscanners oder eines einfachen 3D-Laserscanners an der mobilen Einheit, könnte ein Sensor eingespart werden. Es müsste untersucht werden, ob mit nur einem Sensor dieselben Möglichkeiten wie momentan oder auch mehr Möglichkeiten bestehen würden. Eine weitere Möglichkeit wäre das Identifizieren von Objekten, zu denen dann navigiert werden könnte. Für die mobile Einheit sollte ein Energiekonzept entwickelt werden und die restlichen Sensoren, die bis jetzt noch nicht über die Akkumulatoren laufen, an diese angeschlossen werden. Auf Softwareebene könnte sowohl das Betriebssystem, als auch die ROS-Version auf den aktuellen Stand gebracht werden. Aktueller Stand der möglichen Software bei der Verfassung dieser Arbeit ist die Linux Distribution Ubuntu 16.04 LTS und die ROS-Version Jade Turtle. Eine zukünftige Version von ROS wird im Mai 2016 unter dem

Versionsnamen Kinetic Kame erwartet. Mit den neuen Versionen stünde eine größere Auswahl von bereits existierenden Packages zur Verfügung. Des Weiteren könnte eine Zukunftsvision des Telepräsenzsystems sein, Punkte autonom anzusteuern oder autonom einen Bereich zu erkunden, um dem Benutzer anschließend einen Überblick über die gesamte Umgebung zu geben. Bei autonomer Erkundung könnte die Umgebung in sogenannte *Points of interest (POI)* klassifiziert werden. *POI*s könnten dem Benutzer für ihn interessante Punkte bilden, wie zum Beispiel Ladestationen für den Teleoperator oder andere besondere Räumlichkeiten. Weitere Möglichkeiten MoRy-A zu erweitern, wäre die Montage einer oder mehrerer Vorrichtungen zum Greifen und Betätigen von Knöpfen, um z.B. mit dem Fahrstuhl zu fahren. Hierbei wäre jedoch zum einen auf die Gewichtsverteilung und maximale Zuladung des Teleoperators zu achten. Außerdem müsste die Software so konzipiert werden, dass im Falle des Fahrens mit dem Fahrstuhl die einzelnen Stockwerke unterschieden werden könnten und die Karten nicht ineinander geschrieben werden würden.

Literaturverzeichnis

[1] 2016 adept mobilerobots. – URL www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx. – Zugriffsdatum: 25.02.2016

[2] Acker, Andrea: *Anwendungspotential von Telepräsenz- und Teleaktionssystemen für die Präzisionsmontage*. Januar 2011. – Zugriffsdatum: 03.04.2016

[3] Bäker, B.: *Moderne Elektronik im Kraftfahrzeug*. expert-Verlag, 2007 (Fachbuch Bd. 2). – URL https://books.google.de/books?id=EqQUeHHrlzsC. – S. 172 ff.. – ISBN 9783816926689

[4] Bet: Februar 2016. – URL https://www.bet.de/lexikon/remission/. – Zugriffsdatum: 26.02.2016

[5] Blanck, Ilona B.: *Kartenerstellung und Navigation zur Positionsbestimmung autonomer Fahrzeuge*. Juli 2008. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2008/blanck/bericht.pdf. – Zugriffsdatum: 18.04.2016

[6] Borenstein, Johann ; Feng, Liqiang ; Everett, HR: *Navigating mobile robots: systems and techniques*. AK Peters, Ltd., 1996

[7] Brian Gerkey, Tony P.: *map_server*. – URL http://wiki.ros.org/map_server. – Zugriffsdatum: 01.05.2016

[8] Clarke-Griebsch, S.: *Overcoming Network Delays in Telepresence Systems with Prediction and Inertia*. Herbert Utz Verlag, 2006. – URL https://books.google.de/books?id=fCYYNAAACAAJ

[9] cyberbotics: *Pioneer3-DX*. Februar 2016. – URL https://www.cyberbotics.com/guide/png/pioneer3dx_real.png. – Zugriffsdatum: 25.02.2016

[10] Desai, M. ; Tsui, K. M. ; Yanco, H. A. ; Uhlik, C.: Essential features of telepresence robots. In: *2011 IEEE Conference on Technologies for Practical Robot Applications*, April 2011, S. 15–20. – ISSN 2325-0526

[11] Doert, Colin: *Simultane Lokalisierung und Kartenerstellung eines autonomen Roboters in einer dynamischen Umgebung bei Anwesenheit von Personen.* Januar 2009. – URL http://patrec.cs.tu-dortmund.de/pubs/theses/da_doert.pdf. – Zugriffsdatum: 18.04.2016

[12] Dorrer, C.: *Effizienzbestimmung von Fahrweisen und Fahrerassistenz zur Reduzierung des Kraftstoffverbrauchs unter Nutzung telematischer Informationen.* expert-Verlag, 2004 (Schriftenreihe des Instituts für Verbrennungsmotoren und Kraftfahrwesen der Universität Stuttgart). – URL https://books.google.de/books?id=sUrv1_ImvZsC. – S. 18, 135. – ISBN 9783816923848

[13] DoubleRobotics: *Double 2.* – URL http://www.doublerobotics.com/. – Zugriffsdatum: 03.04.2016

[14] DoubleRobotics: *Double 2.* – URL http://www.doublerobotics.com/pricing.html. – Zugriffsdatum: 03.04.2016

[15] Draper, John V. ; Kaber, David B. ; Usher, John M.: Telepresence. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 40 (1998), Nr. 3, S. 354–375. – URL http://hfs.sagepub.com/content/40/3/354.abstract

[16] Dresssler, Jörg: *Navigation mobiler Systeme in Indoor-Umgebungen*, Fachhochschule Brandenburg, Diplomarbeit, August 2001. – Zugriffsdatum: 01.03.2016

[17] Ehrenstrasser, M.: *Sensoreinsatz in der telepräsenten Mikromontage.* Utz, 2008 (Forschungsberichte IWB). – URL https://books.google.de/books?id=-6zPuX13BiEC. – ISBN 9783831607433

[18] Eitan Marder-Eppstein, Dave H.: *costmap_2d.* – URL http://wiki.ros.org/costmap_2d. – Zugriffsdatum: 13.05.2016

[19] Eitan Marder-Eppstein, Eric P.: *base_local_planner.* – URL http://wiki.ros.org/base_local_planner?distro=hydro. – Zugriffsdatum: 13.05.2016

[20] Eitan Marder-Eppstein, Vijay P.: *Actionlib.* – URL http://wiki.ros.org/actionlib. – Zugriffsdatum: 01.05.2016

[21] Elfes, A.: Sonar-based real-world mapping and navigation. In: *IEEE Journal on Robotics and Automation* 3 (1987), June, Nr. 3, S. 249–265. – ISSN 0882-4967

[22] Elfes, A.: Using occupancy grids for mobile robot perception and navigation. In: *Computer* 22 (1989), June, Nr. 6, S. 46–57. – ISSN 0018-9162

[23] Ellis, S.: *Pictorial Communication In Real And Virtual Environments*. Taylor & Francis, 1991. – URL https://books.google.de/books?id=8c6PClwCymcC. – S. 233 ff.. – ISBN 9780748400089

[24] Flynn, A.M.: Combining Sonar and Infrared Sensors for Mobile Robot Navigation. In: *The International Journal of Robotics Research* 7 (1988), Nr. 6, S. 5–14. – URL http://ijr.sagepub.com/content/7/6/5.abstract

[25] Foote, Tully: tf: The transform library. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, April 2013 (Open-Source Software workshop), S. 1–6. – ISSN 2325-0526

[26] Fossum, Eric R.: *CMOS Image Sensors: Electronic Camera-On-A-Chip*. IEEE. Oktober 1997. – Zugriffsdatum: 28.03.2016

[27] generationrobots: *Ultraschallsensoren für Kollisionsvermeidung*. – URL http://www.generationrobots.com/de/content/65-ultraschallsensoren-f%C3%BCr-kollisionvermeidung. – Zugriffsdatum: 23.03.2016

[28] Gerkey, Brian: *gmapping*. – URL http://wiki.ros.org/gmapping. – Zugriffsdatum: 12.05.2016

[29] Gerkey, Brian P.: *amcl*. – Zugriffsdatum: 01.05.2016

[30] Görke, W. ; Rininsland, H. ; Syrbe, M.: *Information als Produktionsfaktor: 22. GI-Jahrestagung Karlsruhe, 28. September bis 2. Oktober 1992*. Springer Berlin Heidelberg, 2013 (Informatik aktuell). – URL https://books.google.de/books?id=j130BgAAQBAJ. – S.354 f.. – ISBN 9783642778100

[31] Grote, G.: *Autonomie und Kontrolle: zur Gestaltung automatisierter und risikoreicher Systeme*. vdf, Hochschulverl. an der ETH Zürich, 1997 (Mensch, Technik, Organisation). – URL https://books.google.de/books?id=0R8wn_kn4_kC. – S. 7 ff.. – ISBN 9783728123886

[32] Gutmann, Jens-Ste en: Vergleich von Algorithmen zur Selbstlokalisierung eines mobilen Roboters. In: *Master's thesis, Universit at Ulm* (1996)

[33] Harmsen, Lars: *Bewegungslatenzkompensation für HMD von Telepräsenzrobotersystemen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2014

[34] hbf.de hamburger: *Liniennetz Hamburg.* – URL `http://hamburger-hbf.de/liniennetz_hamburg.html`. – Zugriffsdatum: 19.04.2016

[35] Heindorf, Moritz: *Leitstand für mobile Systeme mit Virtual Reality.* Mai 2016

[36] irobot.de: *Ihr Partner für ein saubereres Zuhause.* – URL `http://www.irobot.de/`. – Zugriffsdatum: 26.04.2016

[37] Jochen Sprickerhof, Martin G.: *sick_tim.* – URL `http://wiki.ros.org/sick_tim`. – Zugriffsdatum: 12.05.2016

[38] Juha Roening, Tapio S.: *Intelligent Systems Group (ISG).* 2008. – URL `http://www.infotech.oulu.fi/Annual/2008/isg.html`. – Zugriffsdatum: 18.04.2016

[39] Juric-Kavelj, Srecko: *ROSARIA.* – URL `http://wiki.ros.org/ROSARIA`. – Zugriffsdatum: 25.04.2016

[40] Khoshelham, Kourosh ; Elberink, Sander O.: Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. In: *Sensors* 12 (2012), Nr. 2, S. 1437. – URL `http://www.mdpi.com/1424-8220/12/2/1437`. – ISSN 1424-8220

[41] Krause, Carsten ; Strunz, Ulrich: Vision system for teleoperating mobile robots. In: *Forschung im Ingenieurwesen* 63 (1997), Nr. 1, S. 18–26. – URL `http://dx.doi.org/10.1007/PL00010747`. – ISSN 1434-0860

[42] Küçükay, Ferit ; Bergholz, Janine: *Driver assistant systems.* 2004. – URL `http://www.icatconf.com/icat2000-2006/tr/2004/matbaa/papers/9.Ferit%20Kucukay%20-%20Driver%20Assistant%20Systems.pdf`. – Zugriffsdatum: 19.05.2016

[43] Laserscanning-Europe: *Funktionsweise eines Laserscanners.* – URL `http://www.laserscanning-europe.com/de/glossar/funktionsweise-eines-laserscanners`. – Zugriffsdatum 19.03.2016

[44] Leonard, J. J. ; Durrant-Whyte, H. F.: Mobile robot localization by tracking geometric beacons. In: *IEEE Transactions on Robotics and Automation* 7 (1991), Jun, Nr. 3, S. 376–382. – ISSN 1042-296X

[45] Levi, P. ; Bräunl, T. ; Oswald, N.: *Autonome Mobile Systeme 1997: 13. Fachgespräch, Stuttgart, 6.–7. Oktober 1997*. Springer Berlin Heidelberg, 2013 (Informatik aktuell). – URL https://books.google.de/books?id=BSamBgAAQBAJ. – S. 149. – ISBN 9783642609046

[46] Levi, P. ; Schanz, M.: *Autonome Mobile Systeme 2001: 17. Fachgespräch Stuttgart, 11./12. Oktober 2001*. Springer Berlin Heidelberg, 2013 (Informatik aktuell). – URL https://books.google.de/books?id=Et3zBQAAQBAJ. – S. 11 f.. – ISBN 9783642567872

[47] Lu, David: *global_planner*. – URL http://wiki.ros.org/global_planner?distro=hydro. – Zugriffsdatum: 13.05.2016

[48] Marder-Eppstein, Eitan: *move_base*. – URL http://wiki.ros.org/move_base. – Zugriffsdatum: 01.05.2016

[49] Marder-Eppstein, Eitan: *navigation*. – URL http://wiki.ros.org/navigation. – Zugriffsdatum: 13.05.2016

[50] Mendes, A. ; Bento, L. C. ; Nunes, U.: Multi-target detection and tracking with a laser scanner. In: *Intelligent Vehicles Symposium, 2004 IEEE*, June 2004, S. 796–801

[51] Mercedes-Benz: *Autonom durch Nevada - Freightliner Inspiration Truck*. 2016. – URL https://www.daimler.com/innovation/autonomes-fahren/freightliner-inspiration-truck.html. – Zugriffsdatum: 11.03.2016

[52] Microsoft: *Kinect for Xbox360*. – URL http://research.microsoft.com/en-us/um/redmond/events/fs2010/presentations/Kudo_Fitzgibbons_Kinect_for_Xbox360_071210_FacSummit.pdf

[53] Mihelich, Patrick: *openni_launch*. – URL http://wiki.ros.org/openni_launch. – Zugriffsdatum: 13.05.2016

[54] MobileRobots, Adeopt: *ARIA Driver*. – URL http://robots.mobilerobots.com/wiki/ARIA. – Zugriffsdatum: 25.04.2016

[55] Mojaev, Alexander ; Zell, Andreas: Sonardaten-Integration für autonome mobile Roboter. In: *Mustererkennung 1998*. Springer, 1998, S. 556–563

[56] msdn.microsoft.com. – URL https://i-msdn.sec.s-msft.com/dynimg/IC584396.png. – Zugriffsdatum: 27.02.2016

[57] MSDN.MICROSOFT.COM: *Depth Stream.* – URL https://msdn.microsoft.com/en-us/library/jj131028.aspx. – Zugriffsdatum: 20.05.2016

[58] NAV. – URL http://www.wissen.de/lexikon/navigation. – Zugriffsdatum: 14.03.2016

[59] NEHMZOW, Ulrich: *Mobile Robotik.* Springer, 2002. – URL http://download.springer.com/static/pdf/812/bok%253A978-3-642-55942-6.pdf?originUrl=http%3A%2F%2Flink.springer.com%2Fbook%2F10.1007%2F978-3-642-55942-6&token2=exp=1458486354~acl=%2Fstatic%2Fpdf%2F812%2Fbok%25253A978-3-642-55942-6.pdf%3ForiginUrl%3Dhttp%253A%252F%252Flink.springer.com%252Fbook%252F10.1007%252F978-3-642-55942-6*~hmac=115c203d6acb036849cd6a6ea815b2d616d65ca949a8df5bdea93d1a5a7bf95c. – S. 27 ff.

[60] OLIVER, Ayrton ; KANG, Steven ; WÜNSCHE, Burkhard C. ; MACDONALD, Bruce: Using the Kinect As a Navigation Sensor for Mobile Robotics. In: *Proceedings of the 27th Conference on Image and Vision Computing New Zealand.* New York, NY, USA : ACM, 2012 (IVCNZ '12), S. 509–514. – URL http://doi.acm.org/10.1145/2425836.2425932. – ISBN 978-1-4503-1473-2

[61] OUBBATI, Mohamed: *Einführung in die Robotik - Kinematik.* 2012. November 2012. – URL https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.130/Mitarbeiter/oubbati/RobotikWS1113/Folien/Kinematik.pdf. – Zugriffsdatum: 30.04.2016

[62] PATRICK MIHELICH, Radu Bogdan R.: *openni_camera.* – URL http://wiki.ros.org/openni_camera. – Zugriffsdatum: 13.05.2016

[63] PETZOLD, B.: *Entwicklung eines Operatorarbeitsplatzes für die telepräsente Mikromontage.* Utz, 2008 (Forschungsberichte IWB). – URL https://books.google.de/books?id=KC2HoTF7DkcC. – ISBN 9783831608058

[64] RAKPRAYOON, P. ; RUCHANURUCKS, M. ; COUNDOUL, A.: Kinect-based obstacle detection for manipulator. In: *System Integration (SII), 2011 IEEE/SICE International Symposium on*, Dec 2011, S. 68–73

[65] RIISGAARD, SÃ¸ren ; BLAS, Morten R.. – URL http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-sprin

g-2005/projects/1aslam_blas_repo.pdf. – Figure 1 Overvie of the SLAM process (S.10); Zugriffsdatum: 19.03.2016

[66] Riisgaard, SÃ¸ren ; Blas, Morten R.. – URL http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf. – Zugriffsdatum: 19.03.2016

[67] Rockey, Chad: *depthimage_to_laserscan.* – URL http://wiki.ros.org/depthimage_to_laserscan. – Zugriffsdatum: 12.05.2016

[68] ROS.org: *EinfÃ¼hrung ROS.* – URL http://wiki.ros.org/de/ROS/Introduction. – Zugriffsdatum: 25.04.2016

[69] ROS.org: *ROS Hydro Medusa.* – URL http://wiki.ros.org/hydro. – Zugriffsdatum: 25.04.2016

[70] ROS.org: *Roslaunch tips for large projects.* – URL http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects. – Zugriffsdatum: 04.05.2016

[71] Rügheimer, Hannes: *Daimler Inspiration Truck: Der erste autonome Truck fährt in Nevada.* Mai 2015. – URL http://intelligente-welt.de/daimler-inspiration-truck-der-erste-autonome-truck-faehrt-in-nevada. – Zugriffsdatum: 11.03.2016

[72] Rupp, Torsten: *Absolute Lokalisation mobiler Roboter durch Codierungen mit Landmarken.* dissertation. de, 2001

[73] SICK: *Anschlussmodul CDB730-001.* 2016. – URL https://www.sick.com/media/pdf/9/49/049/dataSheet_CDB730-001_1055981_de.pdf. – Zugriffsdatum: 26.02.2016

[74] SICK: *SICK TIM3xx.* Februar 2016. – URL https://www.sick.com/media/ZOOM/2/32/732/IM0041732.png. – Zugriffsdatum: 26.02.2016

[75] Skiena, S: Dijkstra's algorithm. In: *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley* (1990), S. 225–227

[76] Stenzel, Roland: *Steuerungsarchitekturen für autonome mobile Roboter*, Bibliothek der RWTH Aachen, Dissertation, 2002

[77] Steuer, Jonathan: Defining Virtual Reality: Dimensions Determining Telepresence. In: *Journal of Communication* 42 (1992), Nr. 4, S. 73–93. – URL http://dx.doi.org/10.1111/j.1460-2466.1992.tb00812.x. – ISSN 1460-2466

[78] theory.stanford.edu: *Introduction to A**. – URL http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. – Zugriffsdatum: 22.04.2016

[79] Thrun, Sebastian ; Leonard, John J.: *Springer Handbook of Robotics*. Kap. Simultaneous Localization and Mapping, S. 871–889. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – URL http://dx.doi.org/10.1007/978-3-540-30301-5_38. – Kapitel 37. – ISBN 978-3-540-30301-5

[80] Tsui, K. M. ; Desai, M. ; Yanco, H. A. ; Uhlik, C.: Exploring use cases for telepresence robots. In: *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2011, S. 11–18. – ISSN 2167-2121

[81] Tsumura, T. ; Okubo, H. ; Komatsu, N.: A 3-D position and attitude measurement system using laser scanners and corner cubes. In: *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on* Bd. 1, Jul 1993, S. 604–611 vol.1

[82] TU-Chemnitz: *Navigation für autonome mobile Roboter - Grundlagen*. – URL https://www.tu-chemnitz.de/informatik/KI/edu/robotik/ws2012/robotik_6_1.pdf. – Zugriffsdatum: 18.03.2016

[83] Waldschmidt, Heiko: *Vergleich von Pathfinding-Algorithmen*, Universität Kassel, Diplomarbeit, Oktober 2008. – URL https://www.uni-kassel.de/eecs/fileadmin/datas/fb16/Fachgebiete/PLM/Dokumente/Master_Bachelor_Diplom/masterarbeit.pdf. – Zugriffsdatum: 22.04.2016

[84] Wang, Y. ; Laby, K.P. ; Jordan, C.S. ; Butner, S.E. ; Southard, J.: *Medical tele-robotic system*. August 2 2005. – URL https://www.google.com/patents/US6925357. – US Patent 6,925,357

[85] Wender, S. ; Dietmayer, K.: 3D vehicle detection using a laser scanner and a video camera. In: *IET Intelligent Transport Systems* 2 (2008), June, Nr. 2, S. 105–112. – ISSN 1751-956X

[86] Werner, Sebastian: *VARIABILITÄTSMODELLIERUNG IN KARTOGRAPHIERUNGS- UND LOKALISIERUNGSVERFAHREN*. Juli 2014. – Zugriffsdatum: 18.04.2016

[87] Wiese, Hendrik: *Entwicklung einer Basisplattform für Telepräsenzsysteme*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, April 2014

[88] wissen/Navigation rn: *Navigation*. – URL `http://rn-wissen.de/wiki/index.php?title=Navigation`. – Zugriffsdatum: 28.03.2016

[89] www.xbox.com: *Kinect für Xbox One*. – URL `http://www.xbox.com/de-DE/xbox-one/accessories/kinect-for-xbox-one`. – Zugriffsdatum: 06.05.2016

[90] Yeon, Ahmad Shakaff A. ; Kamarudin, Kamarulzaman ; Visvanathan, Retnam ; Mamduh, Syed M. ; Kamarudin, Latifah M. ; Zakaria, Ammar ; Shakaff, Ali Yeon M.: Feasibility Advancednalysis of 2D-SLAM using combination of Kinect and Laser-Scanner. In: *Jurnal Teknologi* 76 (2015), oct, Nr. 12. – URL `http://dx.doi.org/10.11113/jt.v76.5858`

Abkürzungsverzeichnis

API	Appilcation-Programming-Interface
CMOS	Complementary metal-oxide-semiconductor
CCD	Charge-coupled device
FPS	Frames per Second
HAW	Hochschule für Angewandte Wissenschaften
HMD	Head-Mounted-Display
IR	Infrarot
QR	Quick-Response
ROS	Robot Operating System
VPN	Virtual Private Network
VR	Virtual Reality

A. TF-Tree

Der TF-Tree zeigt die Transforms, welche zwischen den einzelnen verwendeten Softwareteilen verwendet wurden. Die Wurzel des TF-Baums bildet das Frame bzw. Koordinatensystem *map*. *Map* ist das globale Koordinatensystem, in dem sich bewegt wird. Alle abgehenden Transforms können in das global-Frame gerechnet werden. Genauere Infos zu *tf* und dem daraus resultierenden Baum kann unter [25] nachgelesen werden.

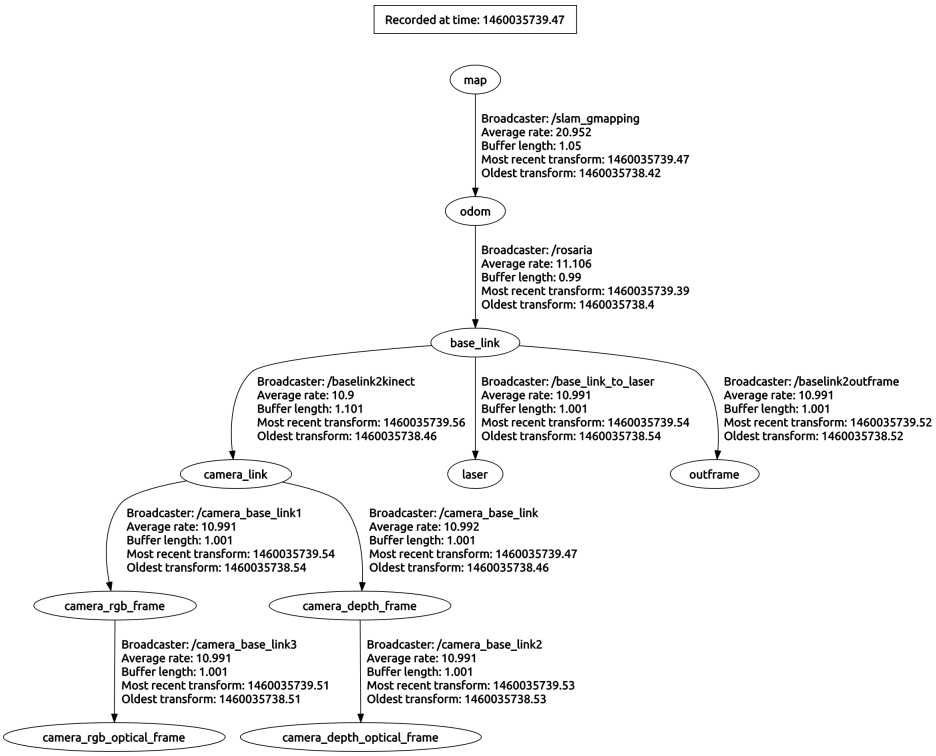

Abbildung A.1.: TF-Tree des gesamten Systems (Orginalgröße siehe Anhang E).

B. RosGraph

Der hier gezeigte ROS-Graph beinhaltet alle verwendeten Topics der Arbeit. Aufgrund des Umfangs kann das Bild nur in kleiner Ausführung dargestellt werden. Das Bild ist in Originalgröße auf der beigelegten CD zu finden (siehe Anhang E).

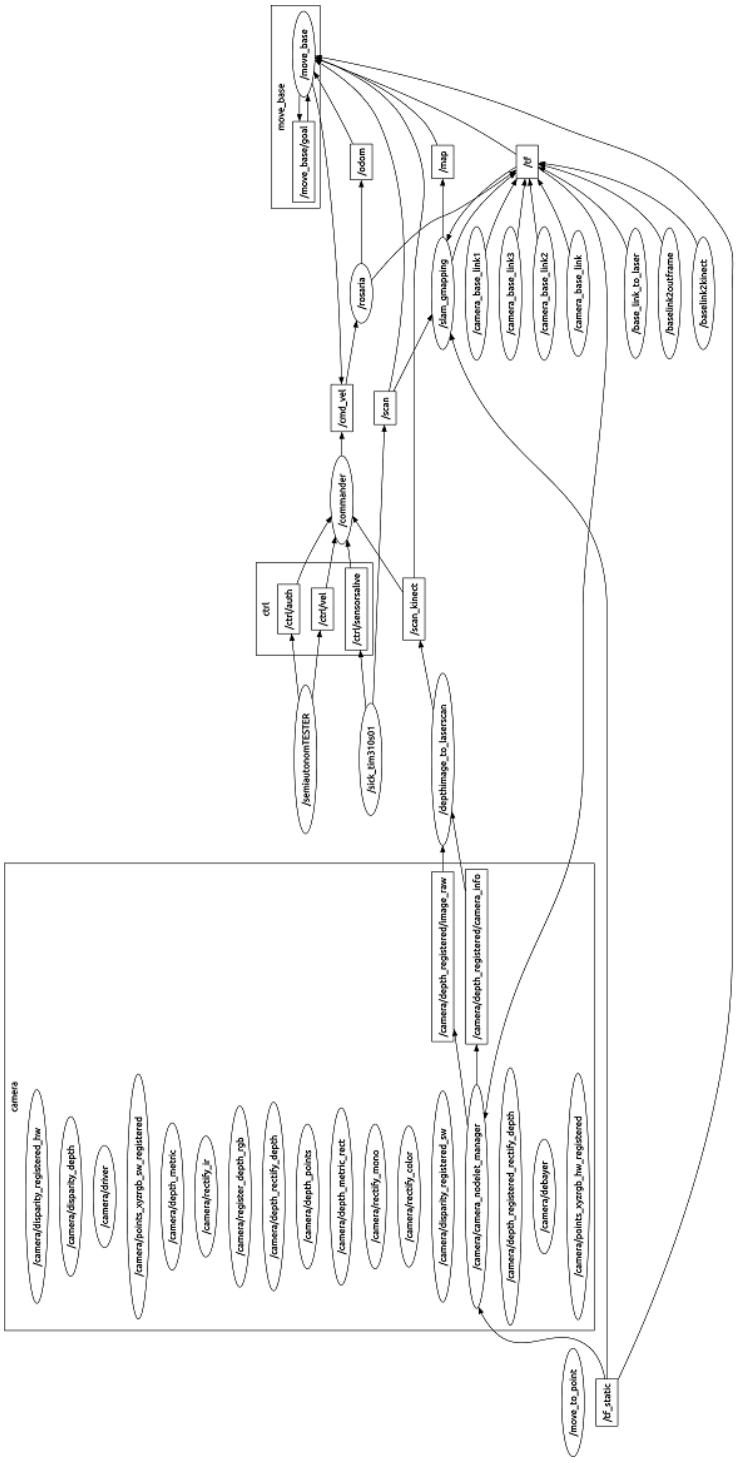

Abbildung B.1.: Übersicht der verwendeten Topics.

C. ROS-Packages

Es werden im Folgenden ROS-Packages, die zur Lösung der Problemstellung beigetragen haben, mit ihren dazugehörigen Topics, Services, Argumenten und Parametern erläutert. Es werden nur benötigte Topics, Services oder Parameter, die explizit gesetzt wurden, aufgelistet. Nicht genannte Schnittstellen oder Parameter sind dem Appilcation-Programming-Interface (API) jeden Packages zu entnehmen. Alle hier aufgelisteten Angaben sind dem ROS-Wiki für die ROS Version *Hydro Medusa* entnommen.

C.1. ROSARIA

ROSARIA (API: [39]) stellt ein ROS-Interface für die Plattform Pioneer 3-DX. Dabei kann das Package Informationen über Geschwindigkeit, Beschleunigung, Sonar, Batteriestatus und Motorstatus liefern.

C.1.1. Subscribed Topics

- „*/cmd_vel*“(geometry_msgs/Twist): empfängt neue Geschwindigkeiten, welche im *ARIA*-Treiber gesetzt werden.

C.1.2. Published Topics

- „*pose*“(nav_msgs/Odometry): veröffentlicht Odometriedaten der Plattform mit einer Frequenz von 10 Hz.

C.1.3. Paramter

- „*port*“(/dev/ttyUSB0): gibt den USB-Port an, welcher verwendet wird, um über einen RS232/USB Adapter mit dem Pioneer 3-DX zu kommunizieren.

C.2. sick_tim

sick_tim wurde zur Ansteuerung des Laserscanners verwendet. In dem Package (API: [37]) sind mehrere Nodes für mehrere Laserscanner enthalten. Es wurde der Node *sick_tim310s01* verwendet.

C.2.1. Published Topics

- „*scan*“(sensor_msgs/LaserScan): veröffentlicht die Laserscans.

C.2.2. Parameter

- „*min_ang*“(-1.04 * pi): Die erste Hälfte des Sichtwinkels angegeben in rad. Bildet zusammen mit „*max_ang*“den Gesamtblickwinkel.
- „*max_ang*“(1.04 * pi): Die zweite Hälfte des Sichtwinkels in rad.
- „*intensity*“(False): Intensitäten der Messerwerte.
- „*skip*“(0): Scans die zwischen zwei Scans übersprungen werden sollen.
- „*frame_id*“(„laser“): Name des *tf*-Frames.

C.3. openni

Über die beiden unten genannten Packages ist es möglich, die Microsoft Kinect anzusteuern und die gemessenen Daten auszulesen.

C.3.1. openni_camera

Der in diesem Package (API: [62]) enthaltene Node stellt den Treiber für die Mircosoft Kinect dar. Über diesen Node ist es möglich, an die benötigten Datenströme zu gelangen. Die hierfür benötigte Topic ist im folgenden aufgelistet.

C.3.2. Topics

- „*depth_registered/image_raw*“(sensor_msgs/Image): Liefert die Rohdaten des Tiefenbildes der Kinect.

C.3.3. openni_launch

Über dieses Packages (API: [53]) ist es möglich, den oben genannten Node des *openni_camera* Packages zu starten. Das *.launch*-File wurde von ROS übernommen und folgender Parameter wurde für den Node angepasst:

- „*depth_registration*“(True): Der Wert wurde von *False* auf *True* gesetzt. Der Parameter bewirkt, dass nur Punkte gemessen werden, die von der RGB-Kamera der Kinect ebenfalls erkannt werden.

C.4. depthimage_to_laserscan

Das Package *depthimage_to_laserscan* (API: [67] wandelt ein Tiefenbild in eine Nachricht vom Typ *sensor_msgs/LaserScan* um.

C.4.1. Subscribed Topics

- „*image*“(sensor_msgs/Image): Liest das Bild der von openni-Topic */camera/depth/image_raw* ein.

C.4.2. Published Topics

- „*scan*“(sensor_msgs/LaserScan): Veröffentlicht das umgewandelte Bild als Laserscan.

C.4.3. Parameter

Die Parameter „*scan_height*“, „*range_min*“und „*range_max*“sind dem Abschnitt 3.3 zu entnehmen.

- „*output_frame*“(„outframe“): Gibt den TF-Frame-Name an.

C.5. gmapping

Das *gmapping*-Package (API: [28]) enthält den verwendeten Node *slam_gmapping*, welcher es ermöglicht SLAM zu betreiben.

C.5.1. Subscribed Topics

- „*tf*“(tf/tfMessage): Benötigte Transformationen zwischen Sensoren, Plattform und Odometrie.

- „*scan*“(sensor_msgs/LaserScan): Laserscans zum Erstellen der Karte.

C.5.2. Published Topics

- „*map*“(nav_msgs/OccupancyGrid): Stellt die generierte Karte zur Verfügung.

C.5.3. Services

- „*dynamic_map*“(nav_msgs/GetMap): Der Service kann in Anspruch genommen werden, um die Karte zu erhalten.

C.5.4. Parameter

- „*map_update_interval*“(2.0): Gibt die Zeit in s an, mit der die Karte aktualisiert werden soll.
- „*maxRange*“(4.0): Gibt die maximale Entfernung in m an, die von dem Sensor, welcher für die Kartenerstellung genutzt wird, gemessen werden kann.

C.5.5. TF-Anforderungen

Es wird die Transformation vom Sensor zur Plattform (Basis) und die Transformation von der Basis zur Odometrie benötigt. Der TF-Baum ist dem Anhang A zu entnehmen.

D. Workspace

Der Workspace ist auf dem der Arbeit beigelegtem Datenträger wiederzufinden. Die genaue Speicherort auf dem Datenträger ist dem Anhang E zu entnehmen. Der Workspace enthält drei verschiedene Ordner:

- build
- devel
- src

Der Ordner *src* enthält Unterordner, welche den Packagenamen der Packages entsprechen. In den einzelnen Unterordnern sind mehrere Unterordner, wovon einer *src* heißt. In diesem Ordner befindet sich der Sourcecode der einzelnen Packages. Ein Beispiel zum Auffinden der Sourcecode-Dateien des Nodes *sick_tim310s01* im Package *sick_tim* stellt folgender Pfad dar:

$catkin\_ws/src/sick\_tim/src/sick\_tim310s01.cpp$

E. Datenträger

Der Datenträger ist dieser Arbeit beigefügt. Auf ihm befinden sich:

- Thesis.pdf - Das PDF enthält die hier geschriebene Bachelorarbeit.
- Ordner Thesis - Enthält den LaTeX- Code dieser Arbeit, sowie die Unterordner:
  - Bilder - enthält die in der Arbeit verwendeten Abbildungen.
  - logo - enthält die HAW-Logos.
  - Quellen - enthält die Quellen, die in dieser Arbeit verwendet wurden.
- Ordner Anhang - enthält den Workspace im *.zip*-Format, den Workspace als Unterordner und die Bilder aus dem Anhang. Die Bilder sind wieder in einem Unterordner zusammengefasst.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. Mai 2016 Andreas Löffler