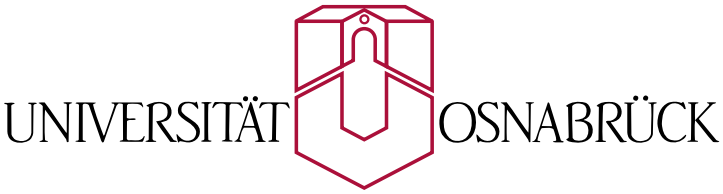


Institut für Informatik
AG Wissensbasierte Systeme

Multisensorfusion zur semantisch gestützten Navigation eines autonomen Assistenzroboters

Dissertation

zur Erlangung des Doktorgrades (Dr. rer. nat.)
des Fachbereichs Mathematik/Informatik
der Universität Osnabrück

vorgelegt von

Stefan Stiene

19. März 2009

Erstgutachter:	Prof. Dr. Joachim Hertzberg
Zweitgutachter:	Dr. techn. Norbert Elkmann
Drittgutachter:	Prof. Dr.-Ing. Werner Brockmann

Zusammenfassung

Ein alltagstauglicher autonomer Assistenzroboter in einem gemeinsamen Arbeitsumfeld mit dem Menschen erfordert, dass der Roboter sämtliche Hindernisse in seiner Umgebung wahrnimmt und diesen sicher ausweicht. Stand der Technik ist jedoch, dass meist nur 2D-Sensordaten zur Navigation herangezogen werden. Oder es werden 3D-Verfahren verwendet, die ausschließlich mit einer speziellen Sensorkonfiguration arbeiten. Diese Arbeit untersucht im Rahmen des LiSA-Projekts wie 3D-Sensordaten effizient und flexibel zur sicheren Navigation eines autonomen Assistenzsystems eingesetzt werden können.

Dazu wird in dieser Arbeit mit der Virtual Range Scans (VRS)-Methode ein Verfahren zur Hindernisvermeidung entwickelt, das beliebige Konfigurationen von Abstandssensoren in den Hindernisvermeidungsprozess integriert. Das Verfahren nutzt klassische Verfahren zur 2D-Hindernisvermeidung, um 3D-Hindernisvermeidung zu realisieren. Dadurch wird das VRS-Verfahren unabhängig von der Hindernisvermeidungsstrategie und kann flexibel bestehende Verfahren wiederverwenden. Neben der Hindernisvermeidung wird gezeigt, wie die reichere Umgebungsinformation, die in 3D-Sensordaten vorhanden ist, zur robusteren Selbstlokalisierung des Roboters genutzt werden kann. Hier wird ein effizientes Verfahren vorgestellt, das Abstandssensordaten mit 3D-Umgebungsmodellen vergleicht. Ferner wird ein Verfahren vorgestellt, das Semantik im Umfeld des Roboters verankert und sie zur Navigation des Roboters nutzt.

Abstract

An autonomous mobile robot suitable for everyday use in a life science environment shared with humans has to ensure safe navigation which includes detection and avoidance of all obstacles in the robot surrounding. However state of the art is, that mostly 2D sensor configurations are used. If a 3D sensor configuration is used the applied obstacle avoidance strategy is bound to this specific hardware. This thesis is about flexible, reusable and efficient integration of 3D sensor data into mobile robot control algorithms independent of the used hardware.

This thesis presents a new method for seamless fusion of arbitrary range sensors for mobile robot obstacle avoidance. This method, named Virtual Range Scans (VRS), is able to deal with arbitrary sensor configurations (2D and 3D) and it is independent of the underlying obstacle avoidance strategy and hardware. This makes it a very flexible approach that can reuse existing 2D obstacle avoidance algorithms for 3D obstacle avoidance. Besides obstacle avoidance an efficient method for comparison of 3D sensor data with 3D environment models is presented. This BSP-tree based method can be flexibly plugged into several mobile robot localization strategies. Furthermore, this thesis shows how to anchor semantics in the robot's working environment and how to use this semantics for safe robot navigation.

Danksagung

Zunächst möchte ich Herrn Prof. Joachim Hertzberg danken, der mir einen Einblick in die Welt der Robotik gegeben hat. Er hat mich während der gesamten Arbeit durch konstruktive Diskussion und hilfreiche Beiträge unterstützt. Seine Freundlichkeit und stets offene Tür haben für eine angenehme Arbeitsatmosphäre gesorgt.

Ebenfalls danke ich meinem Zweitgutachter Dr. Norbert Elkmann und meinem Drittgutachter Prof. Dr.-Ing. Werner Brockmann.

Des Weiteren möchte ich Dr. Andreas Nüchter, Dipl.-Inform. Kai Lingemann, Thomas Wiemann M.Sc. und Christopher Lörken M.Sc. für die gemeinsame Forschung im Bereich der autonomen mobilen Robotik danken. Ihre Ideen und Anregungen haben mir während dieser Arbeit sehr geholfen.

Ein weiterer Dank gilt ebenfalls allen studentischen Hilfskräften, die im Rahmen des LiSA-Projekts mitgewirkt haben.

Ferner möchte ich den Projektpartnern im LiSA-Projekt danken. Insbesondere Erik Schulenburg und Angelika Girstel haben durch konstruktive Diskussion dieser Arbeit geholfen.

Nicht zuletzt möchte ich meiner Frau Annika danken, die mich in jeder Hinsicht unterstützt hat, so dass ich mich in den letzten drei Jahren voll auf diese Arbeit konzentrieren konnte. Neben Annika möchte ich Monika Stiene und Bärbel Unland danken, die den Text mit viel Geduld Korrektur gelesen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
BSP	Binary Space Partioning
EKF	Extended-Kalman-Filter
KF	Kalman-Filter
KURT	Kanal-Untersuchungs-Roboter-Testplattform
LiSA	Life Science Assistent
ND	Nearness Diagram
MCL	Monte Carlo Lokalization
ML	Markow Lokalisierung
RANSAC	Random Sample Consensus
SIR	Sampling Importance Resampling
SPS	Speicherprogrammierbare Steuerung
USARSim	Urban Search And Rescue Simulation
VFH	Vector Field Histogram
VRS	Virtual Range Scans

Kapitel 1

Einleitung

Der Wunsch nach autonomen für den Menschen arbeitenden Assistenzsystemen bildet seit vielen Jahren die Grundlage für eine intensive Forschung und Entwicklung auf diesem Gebiet. Einsatzgebiete dieser Systeme reichen von fahrerlosen Transportsystemen in der Industrie über den mobilen Roboter als Haushaltshilfe, den Rettungsroboter in Katastrophengebieten, bis zu dem mobilen Assistenzsystem in Umgebungen, die für den Menschen unzugänglich, da gesundheitsgefährdend sind. Durch Science-Fiction-Filme wird vermittelt, dass alle diese Roboter bereits existieren und Roboter in der Lage sind, sich sicher im Raum von A nach B zu bewegen, ihre Umwelt wahrzunehmen, zu verstehen und mit dieser kommunizieren zu können. Ein weiterer Grund dafür, dass die Roboter in der allgemeinen Meinung wesentlich weiter entwickelt sind, als es tatsächlich der Fall ist, liegt darin, dass in der autonomen mobilen Robotik Probleme gelöst werden müssen, die für den Menschen alltäglich sind und häufig unbewusst gelöst werden. Beispielsweise gehören zu diesen Problemen die Selbstlokalisierung des Menschen, die Objektwahrnehmung und -klassifizierung sowie die Handlungsplanung. Die Realität sieht jedoch anders aus. So beschränkt sich der Einsatz mobiler Roboter in der Industrie auf fahrerlose Transportsysteme. Diese sind zwar mobil, aber nicht autonom, da sie häufig über Leitdrähte oder Transponder an eine definierte Spur gebunden sind. Der Grund für diese Tatsache ist das Gefährdungspotential, das von einem solchen System ausgeht. Es muss sichergestellt werden, dass ein autonomes System Menschen und andere Hindernisse im gemeinsamen Arbeitsbereich wahrnimmt und diesen sicher ausweicht. Stand der Technik ist jedoch, dass solche Systeme nicht über eine Wahrnehmung ihres 3D-Umfelds verfügen, sondern ihre Umgebung über Abstandssensoren in einer zweidimensionalen Ebene wahrnehmen. Ferner sind sich die Systeme in keinster Weise bewusst, in welchem semantischen Zusammenhang sie zu ihrer Umgebung stehen.

Hier setzt das im Rahmen des „Assistenzroboter in Laboren von Life-Science-Unternehmen“ (LiSA)-Projekts entwickelte System an. Das Ziel des Projekts ist es, ein alltagstaugliches, autonomes Assistenzsystem zu entwickeln, welches im Bereich der Life-Science-Forschung Transportaufträge übernimmt. Das LiSA-Projekt ist ein Verbundprojekt, daher sind Komponenten des Roboters und seiner Steuerung von anderen Forschungsinstituten und Firmen

entwickelt[1]. So wird der Manipulator des Roboters vom Fraunhofer IFF entwickelt, die Plattform wird von der Firma Götting hergestellt und die Multimodale Interaktion wird von den Firmen Sympalog und Project Syntropy realisiert. In dieser Arbeit wird die Sensorkonfiguration der Plattform des Roboters konzipiert und es werden die Algorithmen zur autonomen Navigation entwickelt.

Der LiSA-Roboter soll wirklich autonom sein, was bedeutet, dass in der Umgebung keine Umbaumaßnahmen (Transponder, Leitdrähte, Reflektoren,...) durchgeführt werden müssen und der Roboter anhand eines Umgebungsmodells den zu fahrenden Weg frei bestimmt. Diese Tatsache im Zusammenhang mit der Alltagstauglichkeit und der sicheren Mensch-Roboter-Interaktion im gemeinsamen Arbeitsumfeld bedeutet, dass der Roboter seine Umgebung in allen drei Dimensionen wahrnehmen und Hindernissen sicher ausweichen muss.

In dieser Arbeit wird der LiSA-Roboter und dessen Sensorkonfiguration zur 3D-Umgebungserfassung vorgestellt. Diese erlaubt es, Menschen und andere Hindernisse im gesamten dreidimensionalen Umfeld des Roboters wahrzunehmen. Es wird gezeigt, wie diese 3D-Sensorkonfiguration in Kombination mit einem 3D-Umgebungsmodell zur Selbstlokalisierung genutzt werden kann. Ferner wird ein neuartiges Verfahren vorgestellt, das eine sichere Navigation ermöglicht, die allen Hindernissen im dreidimensionalen Arbeitsraum des Roboters ausweicht. Des Weiteren wird gezeigt, wie die Navigation des Roboters durch Verankerung von Semantik im 3D-Umgebungsmodell gestützt werden kann.

Der LiSA-Roboter bildet somit einen Schritt in die Richtung des autonomen, alltagstauglichen, flexibel einsetzbaren Assistenzroboters. Anwendungsgebiete dieser Systeme sind zahlreich und beschränken sich nicht auf einfache Hol- und Bringdienste. Da die in dieser Arbeit entwickelten Verfahren unabhängig von der Hardware des LiSA-Roboters sind, bilden sie einen Beitrag zur Lösung aller oben angesprochenen Anwendungsszenarien autonomer mobiler Roboter.

1.1 Aufbau der Arbeit

Kapitel 1 gibt eine Einleitung zum Thema der Arbeit. Es wird der Aufbau der Arbeit erläutert und es werden die wesentlichen wissenschaftlichen Beiträge der Arbeit herausgestellt.

Kapitel 2 stellt das LiSA-Projekt vor. Hier wird auf das Arbeitsumfeld des Roboters sowie auf den Roboter selbst eingegangen. Es wird auf die einzelnen Komponenten des Roboters eingegangen und es wird erläutert welche Anforderungen das LiSA-Szenario an die Navigation des Roboters stellt.

[1]Dieses Forschungs- und Entwicklungsprojekt wird mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF) innerhalb des Rahmenkonzeptes "Forschung für die Produktion von Morgen" (Förderkennzeichen 02PB2170 bis 02PB2177) gefördert und vom Projektträger Forschungszentrum Karlsruhe, Bereich Produktion und Fertigungstechnologien (PTKA-PFT), betreut.

Kapitel 3 beschreibt den Stand der Technik. Hier werden alternative Assistenzsysteme vorgestellt und mit dem in dieser Arbeit entwickeltem LiSA-Roboter verglichen. Ferner werden Verfahren zur 2D/3D-Lokalisierung und Hindernisvermeidung beschrieben.

Kapitel 4 erläutert das Konzept zur Navigation des LiSA-Roboters. Neben der Zielsetzung der Arbeit wird das in dieser Arbeit ermittelte kinematische Modell des Roboters und die verschiedenen Softwarekomponenten zur Navigation des LiSA-Roboters vorgestellt.

Kapitel 5 beschreibt die a priori Aufnahme der Arbeitsumgebung des LiSA-Roboters. Da dieser seine Umgebung nicht autonom exploriert, benötigt er ein Umgebungsmodell zur Lokalisierung und Pfadplanung. Ferner wird das Modell zur Mensch-Roboter-Interaktion eingesetzt. Kapitel 5 legt den gesamten Prozess von der 3D-Aufnahme der Umgebung bis zur automatischen Geometrieextraktion und manuellen Erstellung der 3D-Navigationskarte dar.

Kapitel 6 stellt die beiden wesentlichen wissenschaftlichen Schwerpunkte dieser Arbeit heraus. Es wird auf die Verwendung von 3D-Sensorkonfigurationen in Kombination mit 3D-Umgebungsmodellen zur Roboterlokalisierung eingegangen. Ferner wird mit dem VRS-Konzept ein Verfahren vorgestellt, das eine nahtlose Integration verschiedener Abstandssensoren in den Hindernisvermeidungsprozess ermöglicht. Am Beispiel der 3D-Sensorkonfiguration des LiSA-Roboters wird gezeigt, wie dieses Konzept 3D-Hindernisvermeidung mit Hilfe klassischer zweidimensionaler Hindernisvermeidungsverfahren realisiert.

Kapitel 7 gibt einen Überblick über das in dieser Arbeit verwendete Zonenkonzept zur Verhaltenssteuerung des Roboters. Dieses wird verwendet, um Semantik in der Umgebung des LiSA-Roboters zu verankern. Es wird erläutert, wie der LiSA-Roboter mit Hilfe dieser Zonen spezielle Fahrmanöver zur sicheren Türdurchfahrt, bzw. im Bereich einer Arbeitsstation realisiert. Ferner werden im Rahmen dieser Arbeit eingesetzte Verfahren zur Bewegungsdetektion und Bewegungsverfolgung beschrieben, da diese ebenfalls zur Klasse der semantischen Umgebungswahrnehmung gehören.

Kapitel 8 fasst die Arbeit zusammen, zeigt Grenzen der Verfahren und offene Probleme auf. Des Weiteren wird ein Ausblick auf weiterführende Arbeiten gegeben.

1.2 Wissenschaftlicher Beitrag

In dieser Arbeit wird die Konzeption und Navigation der Plattform eines autonomen Assistenzsystems realisiert. Der Sicherheitsaspekt in einer gemeinsamen Arbeitsumgebung von Mensch und Roboter steht dabei im Vordergrund. Es wird eine neuartige Sensorkonfiguration vorgestellt, die es dem LiSA-Roboter ermöglicht, seine Umgebung in allen drei Dimensionen wahrzunehmen. Auf Grundlage dieser Sensorkonfiguration werden im Rahmen dieser Arbeit entwickelte Verfahren zur Integration von 3D-Daten in den Lokalisierungs- und Hindernisvermeidungsprozessen demonstriert.

Lokalisierung: Diese Arbeit zeigt, wie ein effizientes Verfahren der Computergrafik zur Schnittpunktsberechnung zwischen einem 3D-Strahl und 3D-Polygonmodell zur Lokalisierung eines autonomen mobilen Roboters eingesetzt werden kann.

Hindernisvermeidung: Ein weiterer wissenschaftlicher Beitrag dieser Arbeit liegt im Bereich der 3D-Hindernisvermeidung. Obwohl bereits Verfahren zur 3D-Hindernisvermeidung autonomer Roboter existieren, sind diese im Allgemeinen spezielle Lösungen, die auf eine bestimmte Sensorkonfiguration abgestimmt sind. In dieser Arbeit wird ein Verfahren vorgestellt, das beliebige Abstandssensoren nahtlos in den Hindernisvermeidungsprozess integriert. Innovativ an diesem Ansatz ist, dass ein virtueller zweidimensionaler Datensatz verwendet wird, so dass klassische zweidimensionale Verfahren eingesetzt werden können, um eine Hindernisvermeidung in allen drei Dimensionen zu realisieren.

1.2.1 Vorab publizierte Ergebnisse

[1] In [1] wurden auf dem *„Third International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED '06)“* Ergebnisse zu Arbeiten mit der Simulationsumgebung USARSim veröffentlicht (vgl. Kapitel 4.3.3).

[81] In [81] wurden auf dem *„Workshop on Semantic Information in Robotics auf der IEEE International Conference Robotics and Automation (ICRA '07)“* Ergebnisse zur semantischen Korrektur von Sensordaten zum 6D-Scanmatching vorgestellt. Diese werden zur 3D-Kartierung der Arbeitsumgebung des LiSA-Roboters eingesetzt (vgl. Kapitel 5).

[31] In dem Buch Towards Affordance-Based Robot Control wurde anhand des in dieser Arbeit entwickeltem VRS-Konzepts auf unterschiedliche Repräsentationen befahrbarer Bereiche eingegangen und es wird betrachtet, in wieweit diese Repräsentationen bei der Entwicklung von Kontrollstrukturen für autonome mobile Roboter helfen.

[25, 73] In [25] sowie in [73] wurde die Gesamtkonzeption des LiSA-Roboters auf dem *„International Workshop on Safety, Security, and Rescue Robotics (SSRR '07)“* bzw. der *„30th Annual German Conference on AI (KI '07)“* vorgestellt.

[79, 80] in [79] und [80] wurde das in Kapitel 6.2.2 vorgestellte VRS-Konzept auf den „Fachgespräche Autonome Mobile Systeme (AMS '07)“ veröffentlicht bzw. ist für das Jahr 2009 eingereicht.

Kapitel 2

Assistenzroboter in Laboren von Life-Science-Unternehmen

Im Rahmen des „Assistenzroboter in Laboren von Life-Science-Unternehmen“ (LiSA)-Projekts wird ein alltagstauglicher Assistenzroboter für den Laboreinsatz in Life-Science-Unternehmen entwickelt, gebaut und erprobt. Der Roboter wird Transportaufgaben und Bestückung der Geräte vornehmen. Ein Schwerpunkt dieses Projekts liegt in der Alltagstauglichkeit des Assistenzsystems. Dies beinhaltet eine sichere Navigation in hoch dynamischen unmodifizierten Umgebungen, sowie intuitive multimodale Interaktion zwischen Mensch und Assistenzsystem.

2.1 Das Arbeitsumfeld des LiSA-Roboters

Der LiSA-Roboter arbeitet in einem bekannten Umfeld. Es wird von einer statischen Gebäudegeometrie ausgegangen, die vor dem Einsatz des LiSA-Roboters vermessen wird (vgl. Kapitel 5). Obwohl das Arbeitsumfeld eine Laborumgebung ist, handelt es sich um eine hoch dynamische Umgebung. Der LiSA-Roboter arbeitet gemeinsam mit dem Laborpersonal im selben Arbeitsbereich. Es existieren dynamische Hindernisse, wie Drehstühle, Mülleimer und vergleichbare Gegenstände. Das Standardszenario des LiSA-Roboters sind Transportaufträge zwischen zwei Arbeitsstationen in unterschiedlichen Räumen des Labors. Einer dieser Räume ist in Abbildung 2.1 zu sehen. Es wird deutlich, dass die Enge der Umgebung Wendemanöver unmöglich macht, da der Roboter mit einer senkrechten Orientierung zu der Arbeitsstation den Arbeitsbereich für das Personal blockieren und somit einen potentiellen Fluchtweg versperren würde. Dies erfordert ein spezielles Fahrverhalten des Roboters in diesen Bereichen, das in Kapitel 7.1 diskutiert wird. In den ausgewählten Szenarien existieren die folgenden drei Arbeitsstationen.

Abbildung 2.1: Arbeitsumgebung des LiSA-Roboters

(a) **(b)** **(c)**

Abbildung 2.2: Arbeitsstationen des LiSA-Roboters: (a) Tisch-Übergabe zur Übergabe von Multischalen an den Roboter (b) Floreszenz-Reader (c) Inkubator mit automatischer Übergabe

Fluoreszenz-Reader: Der Fluoreszenz-Reader ist in Abbildung 2.2b dargestellt. Er besitzt eine automatische Schleuse (dunkel blau), die vom LiSA-Roboter gesteuert werden kann.

Tisch-Übergabe: Die Tisch-Übergabe ist in Abbildung 2.2a zu sehen. Sie dient dazu, Multischalen, die für einen Transportauftrag vorgesehen sind, für den Roboter zugänglich zu machen. Es handelt sich um eine DIN A4- bis DIN A3-Fläche auf einer Tischoberfläche, deren Ränder optische Marker besitzen, um die bildbasierte Objekterkennung zu erleichtern. Die Tisch-Übergabe erfordert von der Navigation eine präzise Zielanfahrt, da mindestens drei Marker in den Kameradaten vorhanden sein müssen. Die bauartbedingte Position der Kamera und deren Öffnungswinkel erfordern eine Anfahrt der Zielpose mit einer Toleranz von 8 cm und einem Orientierungsfehler von maximal 5°.

Inkubator: Der Inkubator (Abbildung 2.2c) besitzt eine automatische Schleuse, die vom LiSA-Roboter gesteuert werden kann.

2.2 Der LiSA-Roboter

2.2.1 Die Plattform

Das in Abschnitt 2.1 beschriebene Arbeitsumfeld legt Randbedingungen für die Baugröße und Kinematik des LiSA-Roboters fest. So gibt die Türbreite von 75 cm eine maximale Roboterbreite von 60 cm vor. Des Weiteren geben die Arbeitsstationen zusammen mit dem maximalen Arbeitsraum des Manipulators eine minimale Höhe der Plattform von 80 cm vor. Das Gewicht und der Schwerpunkt des ausgefahrenen Manipulators geben eine Abschätzung der Mindestmasse und des Schwerpunkts der Plattform. Die Masse wiederum beeinflusst die möglichen Antriebe, da diese leistungsstark genug sein müssen, um den Roboter zu bewegen. Leistungsstarke Motoren besitzen wiederum einen großen Energiebedarf. Da eine Betriebszeit von etwa 4 Stunden ohne Ladevorgang erreicht werden soll, wird hierzu eine hohe Batterieleistung benötigt, welche wiederum das Gewicht des Roboters erhöht. Ferner fordert die enge Arbeitsumgebung eine omnidirektionale Kinematik des LiSA-Roboters. Da hierfür zwei Motoren pro Rad benötigt werden, ist der Platzbedarf relativ groß. Ebenfalls ist an die Räder die Anforderung gestellt, möglichst weit am Rand des Roboters zu liegen, um die Standfestigkeit des Roboters auch bei voll ausgefahrenem Manipulator zu gewährleisten. Neben den Rädern bilden die Laptops zur Steuerung des Roboters einen entscheidenden Faktor, was Energie- und Platzbedarf betrifft. Hier sind drei Laptops auf dem Roboter vorhanden. Dies ist notwendig, um eine parallele Entwicklung des Roboters durch die Projektpartner zu vereinfachen.

Diese Randbedingungen berücksichtigend, wurde der in Abbildung 2.3 dargestellte LiSA-Roboter entwickelt und aufgebaut. Der Roboter hat eine Gesamtmasse von etwa 200 kg. Die Plattform hat die Maße (60 cm × 80 cm × 80 cm). Inklusive Manipulator ist der Roboter 1,9 m hoch.

2.2.2 Sensorkonfiguration

Der LiSA-Roboter hat eine neuartige Sensorkonfiguration mit insgesamt sechs Laserscannern. So werden zwei SICK S300-Laserscanner an gegenüberliegenden Ecken des Roboters 10 cm über dem Boden und horizontal zu diesem ausgerichtet montiert. Diese Laserscanner haben eine Reichweite von $30m$ und können ein Blickfeld von 270° mit einer Auflösung von 0.5° abtasten. Neben diesen Laserscannern wird an jeder Seite des Roboters unten mittig ein Hokuyo URG-04LX-Laserscanner angebracht. Sie haben eine Reichweite von etwa 4 m und eine Auflösung von 0.36°. Diese Laserscanner sind so ausgerichtet, dass ihre Scanebenen mit dem Roboter einen Winkel von 40° einschließen, so dass die vier Scanebenen einen Trichter um den Roboter bilden (siehe Abbildung 2.4). Neben den optischen Sensoren hat der Roboter Encodersensoren, die Daten über die Radwinkel und Radumdrehungen liefern. Bekanntlich ist die Energieaufnahme ein potentielles Problem für einen Dauerbetrieb mobiler Roboter. Dieses Problem ist hier durch Verwendung neuer, relativ energieeffizienter Laserscanner-

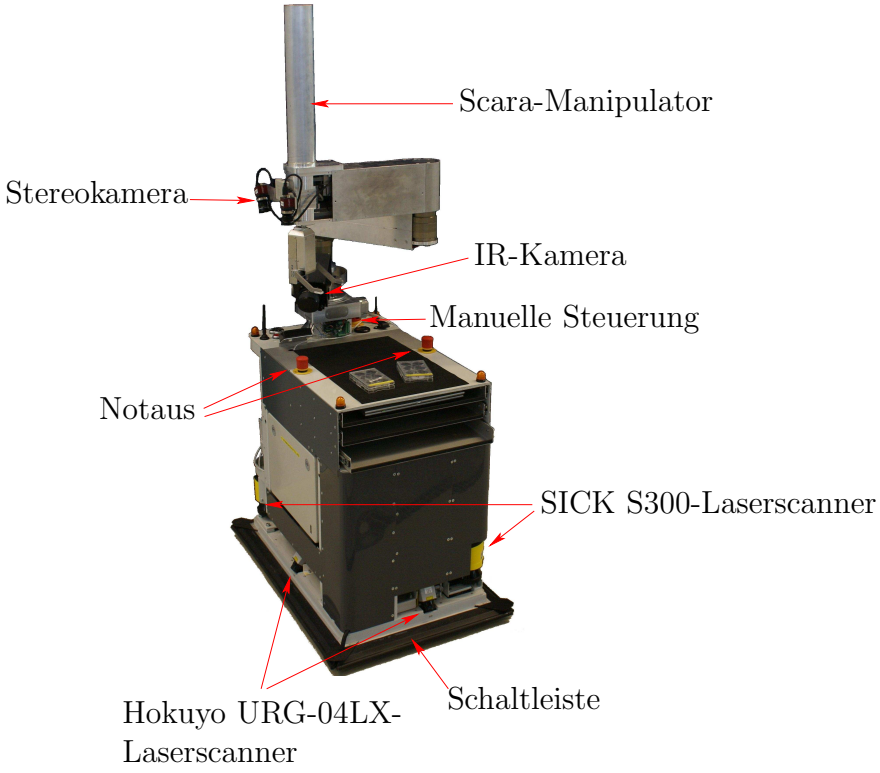
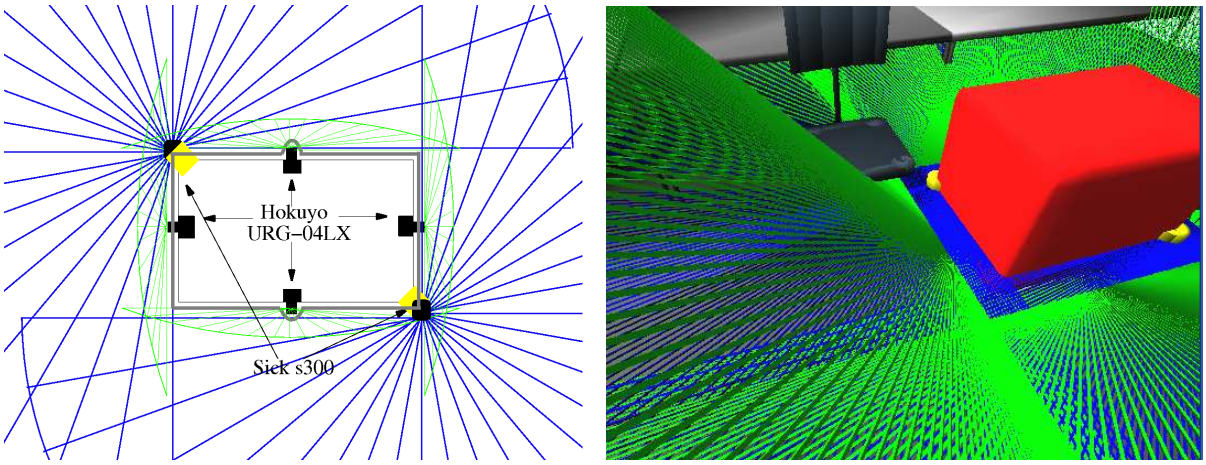


Abbildung 2.3: Der LiSA-Roboter besteht aus einer omnidirektionalen Plattform mit einem Scara-Manipulatoraufbau. Der Manipulator hat eine IR-Kamera zur sicheren Identifizierung von Händen im Arbeitsfeld des Manipulators. Ferner ist ein Stereokamerasystem zur Identifizierung der 6D-Pose der zu manipulierenden Proben angebracht. Die Plattform ist mit zwei SICK S300 Laserscannern ausgestattet, die einen 360° Blickfeld in einer horizontalen Ebene in einer Höhe von etwa 10 cm ermöglichen. An jeder Seite des Roboters ist jeweils ein Hokuyo URG-04LX-Laserscanner montiert, die zusammen einen Trichter um den Roboter bilden und zur 3D-Hindernisvermeidung eingesetzt werden [98].



(a) (b)

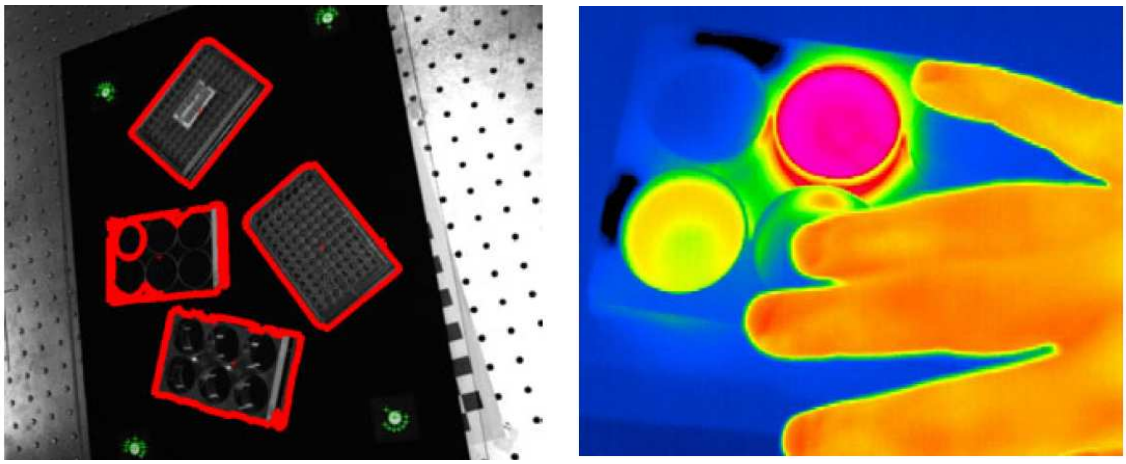
Abbildung 2.4: Sensorkonfiguration des LiSA-Roboters: (a) Aufsicht auf die 3D-Sensorkonfiguration aus sechs Laserscannern. (b) Visualisierung der Laserscannerebenen in der Simulationsumgebung USARSim

Modelle entschärft. Ein SICK S300-Laserscanner benötigt 8 W, ein Hokuyo-Laserscanner 2,5 W. Die Leistungsaufnahme der kompletten Laserscanner-Konfiguration beträgt also nur 26 W. Zum Vergleich: Ein Scanner der bislang auf mobilen Robotern üblicherweise eingesetzt wurde, besitzt eine Leistungsaufnahme von 20 W (SICK LMS200).

2.2.3 Der Manipulator

Obwohl der Manipulator des LiSA-Roboters unabhängig von dieser Arbeit am Fraunhofer IFF entwickelt wird, stellt dieser Abschnitt diesen kurz vor, da er gewisse Voraussetzungen an die Navigation stellt und Rahmenbedingungen für diese vorgibt.

Zu diesen Rahmenbedingungen gehört zunächst einmal, dass der Manipulator am Ende der Längsseite des Roboters angebracht ist (vgl. Abbildung 2.3). Dies gibt der symmetrisch konzipierten Plattform des LiSA-Roboters eine Vorzugsrichtung. Obwohl Experimente zeigen, dass das im Vergleich zur Plattform geringe Gewicht des Manipulators das Fahrverhalten des Roboters nicht wahrnehmbar beeinflusst, erzeugt die Montage am Rand des Roboters jedoch für den Menschen eine psychologische Vorzugsrichtung. So wird eine Fahrt in Richtung des Manipulators als Rückwärtsfahrt wahrgenommen. Um für den Menschen im Arbeitsumfeld des LiSA-Roboters die Vorhersage der Bewegung des Roboters zu erleichtern, wird diese Vorzugsrichtung ebenfalls in die Navigation aufgenommen. Neben der psychologischen Komponente spricht noch ein Sicherheitsaspekt für die Vorzugsrichtung. So kollidieren Hindernisse in der Höhe des Manipulators bei einer Fahrt in Vorzugsrichtung später mit dem Manipulator, so dass die Zeit größer ist, um diese Hindernisse zu bemerken bzw. abzubremsen. Ferner besteht die Möglichkeit, dass bei über die Umgebungsgeometrie herausragenden Hindernissen der Roboter gar nicht so dicht an die Hindernisse heranfahren kann, so dass



(a) (b)

Abbildung 2.5: Bildverarbeitung zur sicheren Manipulation des LiSA-Roboters: (a) Objekt-Detektion in Kameradaten des Manipulators. (b) Detektion einer Hand in den Daten des IR-Kameramoduls. [25]

der Manipulator mit diesen kollidiert. Neben der Vorzugsrichtung stellt die Bauart des Manipulators und des verwendeten Kamerasystems zur Identifizierung der zu manipulierenden Objekte eine Voraussetzung an die Präzision der Navigation. So wird die IR-Kamera des Roboters verwendet, um die Position der Übergabestation zu identifizieren und diese in das Manipulatorkoordinatensystem zu transformieren. Abbildung 2.5 zeigt exemplarische Daten der IR-Kamera. In (a) sind die identifizierten Marker der Übergabestation (grün) und die zu manipulierenden Objekte (rot) dargestellt. In (b) ist das Wärmebild der IR-Kamera dargestellt, das eine sichere Identifizierung von Händen im Arbeitsbereich des Manipulators erlaubt. Voraussetzung für die Identifizierung der Übergabestation ist, dass mindestens drei Marker der Übergabestation in den Kameradaten vorhanden sein müssen. Dies erfordert von der Navigation, dass sie die Pose vor der Übergabestation mit einer maximalen Ungenauigkeit von nur ? cm anfahren muss. Ferner fordert die begrenzte Reichweite des Manipulators, dass der Roboter parallel und möglichst nah zum Tisch steht, auf dem sich die Arbeitsstation befindet. Damit wird erreicht, dass der Manipulator einen maximalen Arbeitsbereich auf der Arbeitsstation besitzt. Dieses parallele Anfahrtmanöver ist ein Grund für die Notwendigkeit des in dieser Arbeit verwendeten Zonenmodells zur semantisch gestützten Navigation.

2.2.4 Mensch-Roboter-Interaktion

Obwohl die multimodale Interaktion des LiSA-Roboters ebenso wie der Manipulator unabhängig von dieser Arbeit entwickelt wird, ist diese hier kurz beschrieben, da die in dieser Arbeit mitentwickelte Kommunikationsstruktur (vgl. Abschnitt 4.3.1) von der Multimodalen Interaktion des LiSA-Roboters mit dem Menschen abhängig ist. Ferner wird das in dieser Arbeit erzeugte 3D-Umgebungsmodell der Arbeitsumgebung des LiSA-Roboters zur Visualisierung in der Mensch-Roboter-Interaktion eingesetzt.

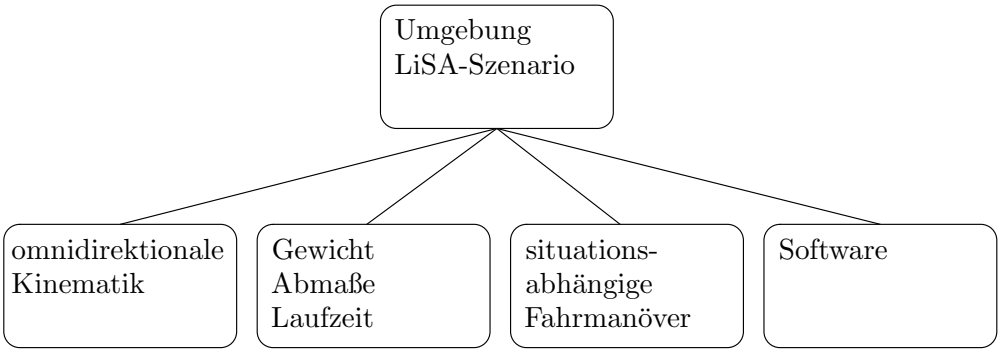


Abbildung 2.6: Anforderungen, die aus der Arbeitsumgebung und den Rahmenbedingungen des LiSA-Projekts folgen (Erläuterung siehe Text).

Die Interaktion zwischen dem LiSA-Roboter und dem Mitarbeiter, der diesem Aufträge erteilt, ist multimodal. Das bedeutet, dass der Benutzer mit dem Roboter mit Hilfe eines Headsets über die gesprochene Sprache als auch mit Hilfe eines Touchpads über grafische Eingaben interagieren kann. Die Sprachverarbeitung wird von der Firma Sympalog realisiert und erlaubt natürlichsprachliche Interaktion. Anhand der Sprachverarbeitung wird die grafische Darstellung des Touchpads gesteuert und deren Eingaben werden parallel zur Sprachverarbeitung ausgewertet. Somit ist eine kombinierte Auftragserstellung über gleichzeitiges Bedienen der GUI in Kombination mit Spracheingaben möglich.

2.3 Anforderungen

Die Anforderungen an den LiSA-Roboter und die Kontrollalgorithmen zur Navigation des Roboters können auf drei Grundanforderungen zurückgeführt werden. Die erste Gruppe von Anforderungen begründet sich auf projektspezifische Rahmenbedingungen wie die Einsatzumgebung des LiSA-Roboters, die von den Projektpartnern eingesetzte Soft- und Hardware, usw. Abbildung 2.6 zeigt, welche Anforderungen aus diesen Rahmenbedingungen folgen. So führt, wie bereits in Abschnitt 2.1 beschrieben, die enge Umgebung dazu, dass der LiSA-Roboter einen omnidirektionalen Antrieb benötigt. Ferner geben die von den Projektpartnern eingesetzte Hardware sowie die Umgebung und die gewünschte Laufzeit Anforderungen an die Größe, die Abmaße und das Gewicht des Roboters (vgl. Abschnitt 2.2.1). Neben der omnidirektionalen Kinematik fordert die Enge der Arbeitsumgebung ebenfalls, dass der Roboter an spezielle Situationen angepasste Fahrmanöver durchführt. Dies wird in dieser Arbeit durch das in Kapitel 7 beschriebene Zonenenkonzept realisiert, welches Semantik in der Umgebung des Roboters verankert. Da die Projektpartner bereits bestehende Software in das Projekt mit einbringen, muss eine Verbindungssoftware für die unterschiedlichen Komponenten geschrieben werden. Als weitere Software wird in dieser Arbeit eine Simulationsumgebung benötigt, da der reale Roboter im Lauf des Projekts neu konzipiert wurde und erst im letzten Drittel zur Verfügung stand.

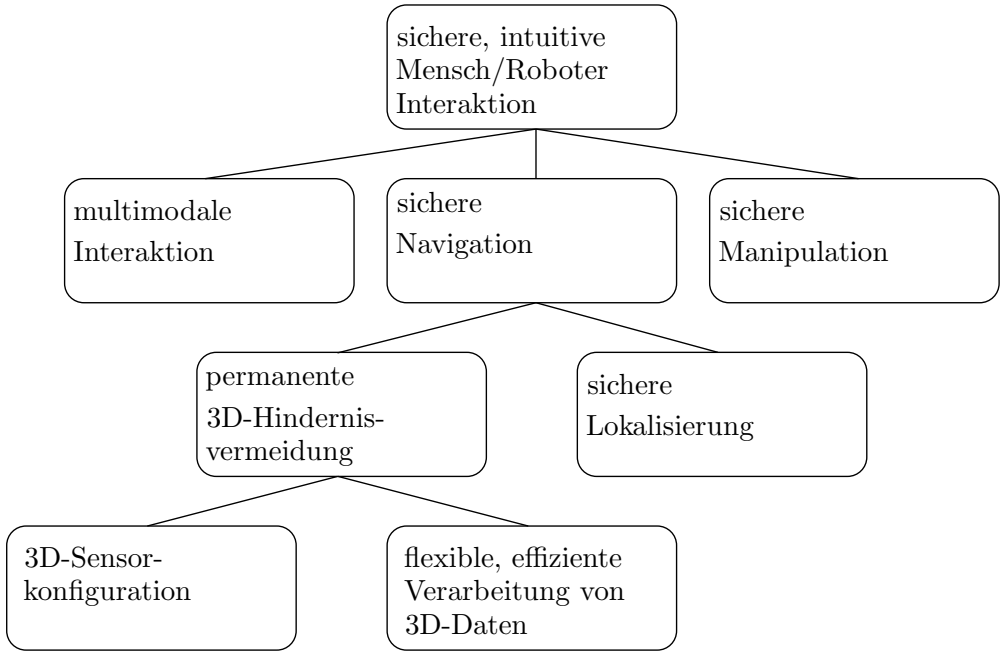


Abbildung 2.7: Durch die Grundanforderung der sicheren, intuitiven Mensch/Roboter Interaktion hervorgerufene Anforderungen an das LiSA-System (Erläuterung siehe Text).

Eine weitere Gruppe von Anforderungen folgt aus der Grundanforderung des LiSA-Projekts, einen sicheren, alltagstauglichen Roboter zu entwickeln. Da der LiSA-Roboter zur Klasse der autonomen mobilen Manipulatoren gehört, setzt dies eine sichere Manipulation und eine sichere Navigation voraus (vgl. Abbildung 2.7). Die Alltagstauglichkeit stellt die Anforderung einer intuitiven, einfachen Bedienung des Systems. Da in dieser Arbeit die Navigation des LiSA-Roboters bearbeitet wird, werden hier weitere Anforderungen, die aus der sicheren Navigation folgen, erläutert. So fordert diese eine sichere Lokalisierung, da ein Roboter, der seine Pose in Bezug zu einem Umgebungsmodell nicht sicher bestimmen kann, nicht in der Lage ist, Ziele innerhalb dieser Umgebung anzufahren. Neben der Lokalisierung ist ein entscheidender Punkt bei der sicheren Navigation eine permanente Hindernisvermeidung. Diese Hindernisvermeidung muss alle drei Dimensionen berücksichtigen, um eine Kollision des Roboters mit seinem Umfeld auszuschließen. Dies setzt zum einen eine Sensorkonfiguration voraus, die permanent die gesamte umschließende Hülle des Roboters abdeckt. Zum anderen wird ein Verfahren benötigt, das effizient mit diesen 3D-Daten eine sichere Hindernisvermeidung realisiert.

Die dritte Grundanforderung an das LiSA-System ist Flexibilität. Wie Abbildung 2.8 zeigt, bedeutet dies für den Bereich der Navigation, dass die Sensorkonfiguration leicht erweiterbar bzw. einfach an sich ändernde Umgebungssituationen anpassbar ist. Dies stellt an die oben beschriebene Hindernisvermeidung zusätzlich die Anforderung, flexibel verschiedene Sensortypen in die Hindernisvermeidung integrieren zu können. Damit der Roboter leicht in unterschiedlichen Umgebungen einsetzbar ist, darf die Lokalisierung nicht auf Marker angewiesen sein. Ferner muss das Umgebungsmodell effizient erstellt werden können. Dies

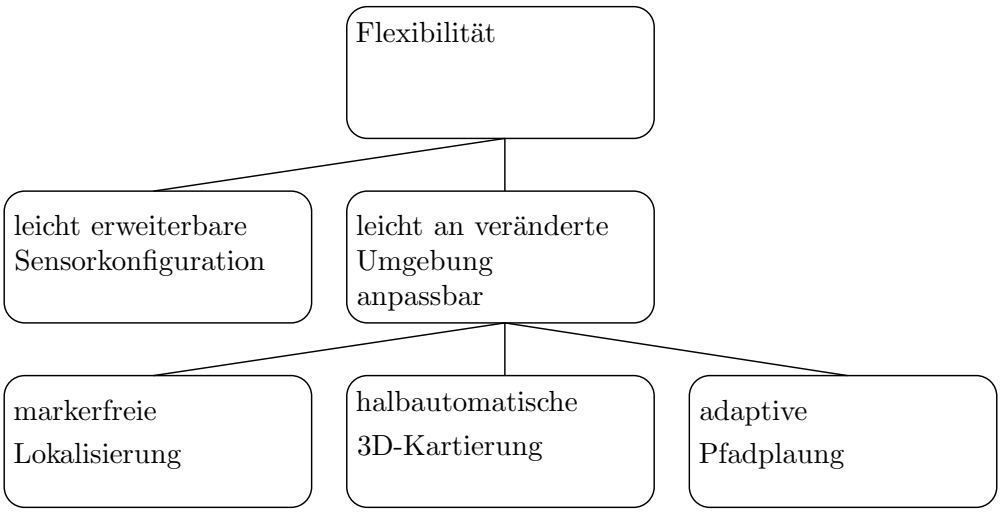


Abbildung 2.8: Durch die Grundanforderung der Flexibilität hervorgerufene Anforderungen an die Navigation des LiSA-Systems (Erläuterung siehe Text).

erfordert zumindest eine halbautomatische 3D-Kartierung. An die Navigation ist darüber hinaus die Anforderung gestellt, dass die Pfadplanung adaptiv veränderte Umgebungssituationen berücksichtigt.

Kapitel 3

Stand der Technik

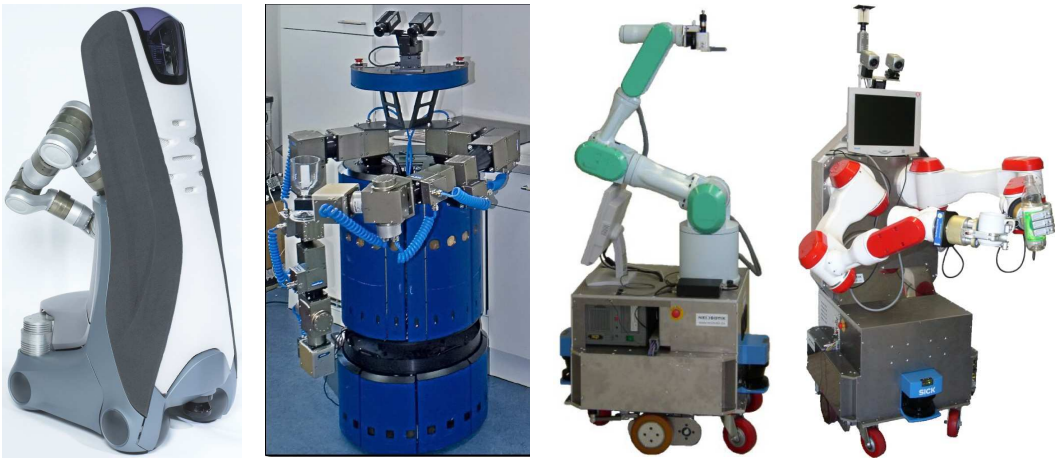
Da im Rahmen dieser Arbeit durch die Entwicklung der autonomen Navigation eines Assistenzsystems viele Teilgebiete der Robotik tangiert werden, teilt sich dieses Kapitel zum Stand der Technik ebenso in verschiedene Bereiche auf. Zunächst wird auf bestehende Roboterassistenzsysteme eingegangen und deren Unterschied zu dem in dieser Arbeit entwickelten LiSA-Roboter herausgestellt. Anschließend wird der Stand der Technik im Bereich der zwei- bzw. dreidimensionalen Lokalisierung und Hindernisvermeidung beschrieben, worin der Kern des wissenschaftlichen Beitrags dieser Arbeit liegt.

3.1 Roboterassistenzsysteme

Im Folgenden werden vier Roboterassistenzsysteme vorgestellt, die zur Klasse der mobilen Manipulation gehören. Es existieren viele vergleichbare Plattformen, so dass die hier vorgestellte Auswahl nur exemplarisch den Stand der Technik verdeutlichen soll.

Care-O-Bot 3: Der Care-O-Bot 3 ist die neueste Generation, der am Fraunhofer IPA entwickelten Serviceroboter. Er ist mit einem omnidirektionalen Antrieb mit vier unabhängig angesteuerten Rädern ausgestattet [14]. Um seine Umgebung dreidimensional zu erfassen, verfügt er über zwei SICK S300-Laserscanner, Ultraschallsensoren, ein Stereokamerasystem und eine 3D-TOF-Kamera.

AssistiveKitchen: Im Rahmen des Projekts „AssistiveKitchen“ wurde an der Technischen Universität München ein Roboterassistenzsystem entwickelt, das in Abbildung 3.1b zu sehen ist. Dieses basiert auf der Plattform B21 der Firma RWI. Diese ist mit Ringen aus Ultraschall- und Infrarotsensoren zur 3D-Umgebungswahrnehmung ausgestattet. Ferner besitzt sie einen in etwa 30 cm Höhe horizontal angebrachten Laserscanner. Des Weiteren besitzt der Roboter ein schwenkbares Stereokamerasystem zur 3D-Umgebungserfassung. Die verwendeten Kontrollalgorithmen basieren ebenso wie die des LiSA-Roboters auf dem Player-Framework (vgl. Kapitel 4.3.2)[67].



(a) (b) (c) (d)

Abbildung 3.1: (a) Der Care-O-Bot 3 ist die neueste Generation der am Fraunhofer IPA entwickelten Serviceroboter [99]. (b) Im „Assistive Kitchen“-Projekt an der Technischen Universität München entwickelter Roboter, der auf der B21 Plattform basiert [67]. (c) Sowohl der im Life-Science-Bereich eingesetzte Assistenzroboter der Universität Bielefeld [71], als auch der an der Universität Hamburg entwickelte Assistenzroboter TASER (d) verwenden die kommerziell erhältliche Plattform MP-L655 der Firma Neobotix [100].

Roboterisierung der Probenentnahme und des Probenmanagements bei der Kultivierung tierischer Zellen im Pilotmaßstab: In diesem Projekt wird ein Roboter entwickelt, der wie der LiSA-Roboter dazu dient, Proben in einer Life-Science-Umgebung zu transportieren und Geräte zu bestücken. Die Plattform wird im Gegensatz zum LiSA-Projekt nicht während der Projektlaufzeit entwickelt, sondern es wird die Neobotix MP-L655 Plattform verwendet. Diese hat einen Differentialantrieb und als einzige Sensoren zur Umgebungserfassung zwei SICK LMS200-Laserscanner, die vorne bzw. hinten an der Plattform angebracht sind. Somit hat der Roboter eine Rundumsicht in einer horizontalen Ebene mit Laserschatten an den Seiten des Roboters. Der Roboter verwendet einen Extended-Kalman-Filter basierten Ansatz zur Lokalisierung. Dieser basiert auf künstlichen Landmarken, so dass der Roboter im Gegensatz zum LiSA-Roboter nicht unabhängig von Umgebungsmarkierungen ist. Ferner beschränkt sich die Hindernisvermeidung auf ein „Warten“ bis das Hindernis sich wieder entfernt hat [71].

TASER: Das an der Universität Hamburg entwickelte Assistenzsystem TASER verwendet ebenfalls die Neobotix Plattform MP-L655. Zusätzlich ist es mit einem Stereokamerasystem und einer omnidirektionalen Kamera ausgestattet. Das Gesamtgewicht des Roboters TASER beträgt 200 kg und er hat eine Betriebszeit von sieben Stunden. Diese Werte sind mit denen des LiSA-Roboter vergleichbar. Die Steuerungsalgorithmen für TASER wurden größtenteils von den im oben beschriebenen Projekt „Roboterisierung der Probenentnahme und des Probenmanagements bei der Kultivierung tierischer Zellen im Pilotmaßstab“ entwickelten Algorithmen übernommen [3].

3.2 Navigation autonomer Assistenzsysteme

3.2.1 Lokalisierung autonomer mobiler Roboter

In diesem Abschnitt werden verschiedene Verfahren vorgestellt, die eine Lokalisierung eines autonomen mobilen Roboters realisieren. Grundsätzlich kann zwischen drei solchen Verfahren unterschieden werden. Die erste Gruppe von Verfahren bestimmt die Roboterpose inkrementell über den Vergleich von aufeinanderfolgenden Sensordaten. Diese Verfahren benötigen somit kein a priori bekanntes Umgebungsmodell und bestimmen ihre Pose immer relativ zur Startpose des Roboters, der den Ursprung des globalen Koordinatensystems bildet. Das a priori bekannte Umgebungsmodell wird von der zweiten großen Gruppe von Verfahren benötigt, da diese die aktuellen Sensordaten mit dem Umgebungsmodell vergleichen und darüber die Pose des Roboters im Umgebungsmodell bestimmen. Diese Art der Lokalisierung ermöglicht es im Gegensatz zu den anderen Verfahren, von einer initial unbekannten Roboterpose auszugehen und den Roboter somit global innerhalb des Umgebungsmodells zu lokalisieren. Die dritte Gruppe von Verfahren löst das sogenannte SLAM-Problem. Dabei wird parallel zur inkrementellen Lokalisierung ohne Umgebungsmodell anhand der Sensordaten ein Umgebungsmodell erstellt, und dieses wird in den Lokalisierungsprozess zurückgeführt. Dies führt dazu, dass Lokalisierung und Erstellung des Umgebungsmodells parallel ablaufen und sich gegenseitig beeinflussen.

Ein weiteres Unterscheidungskriterium für Lokalisierungsverfahren bildet die Dimensionalität der Roboterpose. Viele Verfahren zur Lokalisierung autonomer mobiler Roboter rechnen in einer 2D-Ebene, bestimmen also die Pose des Roboters nur mit den drei Freiheitsgraden (x, y, Θ). Dies hat zum einen den Grund, dass viele Anwendungsszenarien für mobile Roboter eine ebene Grundfläche besitzen. Zum anderen sind viele Sensorkonfigurationen mobiler Roboter so ausgelegt, dass sie ihre Umwelt über Abstandssensoren ausschließlich in einer zweidimensionalen Ebene erfassen. Ferner existieren Verfahren, die von einer 3D-Sensorkonfiguration des Roboters ausgehen. Diese Verfahren nutzen entweder den höheren Informationsgehalt der 3D-Umgebungsinformation, um eine robustere Lokalisierung mit drei Freiheitsgraden zu erreichen oder berechnen sogar die Pose des Roboters mit allen sechs Freiheitsgraden.

Roboterlokalisierung ohne Umgebungsmodell

Diese Verfahren werden auch „pose tracking“ genannt, da sie von der initialen Roboterpose ausgehend diese während der Fahrt des Roboters verfolgen. Dies kann über reine Odometrie, mit Hilfe von Sensordaten oder durch eine Kombination von Sensordaten und Odometrie realisiert werden. Bei Verfahren, die Sensordaten verwenden, wird versucht, identische Merkmale in verschiedenen Sensordatensätzen zu identifizieren und anhand dieser den Poseversatz zwischen den Aufnahmen der beiden Sensordatensätze zu bestimmen. Als Sensoren werden häufig Abstandssensoren oder Kameras verwendet. So wird in [44] ein Verfahren verwendet, das zwei Laserscannerdatensätze zur Deckung bringt und anhand der errechneten

Transformation den Poseversatz des Roboters bestimmt. Hier existieren eine Reihe ähnlicher Verfahren, die sich darin unterscheiden, wie die Korrespondenzen in den Laserscannerdaten gefunden werden [29].

Roboterlokalisierung mit a priori Umgebungsmodell

Diese Form der Lokalisierung entspricht der des LiSA-Szenarios. Neben den probabilistischen Lokalisierungsverfahren, die in dieser Arbeit verwendet werden, existieren weitere Ansätze, die hier exemplarisch beschrieben sind. Es sei angemerkt, dass die in dieser Arbeit beschriebenen Verfahren sich durch unterschiedliche Anwendung teilweise auch zur Lokalisierung ohne Umgebungsmodell bzw. zur SLAM basierten Lokalisierung eines autonomen mobilen Roboters eignen.

Die verwendeten Umgebungsmodelle sind häufig zweidimensionale Raster- oder Linienkarten der Umgebung. Aber auch künstliche Landmarken, die einfach in den Sensordaten zu identifizieren sind, werden verwendet. Ferner existieren Ansätze, die topologische Karten verwenden.

In [28] wird ein Laserscanner in Kombination mit einer zweidimensionalen Linienkarte der Umgebung verwendet, um den Roboter in seiner Umgebung zu lokalisieren. In die Laserscannerdaten werden Liniensegmente gefittet. Alle Liniensegmente eines Laserscans werden mit der Linienkarte verglichen und dadurch mögliche Posen des Roboters bestimmt. Diese werden nach einer Plausibilitätsanalyse als Eingebedaten eines Kalman-Filters verwendet (vgl. Kapitel 6.1.2).

Ein weiters Verfahren, das auf der Extraktion von Liniensegmenten beruht, wird in [37] vorgestellt. Hier werden jedoch Kameradaten verwendet, um den Roboter zu lokalisieren. Ferner wird ein 3D-Polygonmodell der Umgebung verwendet. Zunächst werden mit einer Hough-Transformation basierten Methode Liniensegmente in den Kamerabildern identifiziert. Anschließend wird das Bild der geschätzten Kamerapose im 3D-Modell gerendert, und in diesem Bild werden ebenfalls Liniensegmente identifiziert. Durch den Vergleich der Liniensegmente der realen Kamera mit den Liniensegmenten der virtuellen Kamera wird die virtuelle Pose der Kamera und somit die Roboterpose im 3D-Umgebungsmodell aktualisiert. Das Verfahren zeigt eine Performanz, die mit der Monte-Carlo-Lokalisierung (vgl. Kapitel 6.1.4) in Kombination mit Laserscannerdaten vergleichbar ist.

Ferner werden Kameradaten häufig in Kombination mit topologischen Karten zur Roboterlokalisierung eingesetzt. Diese bilden einen Graph von möglichen Zuständen bzw. Orten, an denen der Roboter sich befindet, und deren Relation zueinander. Zu den einzelnen Zuständen des Graphen werden Informationen gespeichert, die dessen Identifizierung ermöglichen. In [45] sind diese Informationen natürliche Landmarken wie Türen, Lampen,..., die mit einer Kamera identifiziert werden. In [87] werden Farbhistogramme einer omnidirektionalen Kamera verwendet, um die einzelnen Orte zu beschreiben. In [62] wird die Principal Component Analysis für denselben Zweck eingesetzt.

Es existieren ebenfalls Ansätze, die die zeitliche Veränderung der Karte berücksichtigen. Solche Veränderungen können z.B. Türen sein, die entweder geöffnet oder geschlossen sind. Das Problem dieser Veränderungen ist, dass die Sensordaten nicht mit dem Umgebungsmodell übereinstimmt. In [78] werden verschiede Konfigurationen der Umgebung identifiziert und für jede Konfiguration wird eine eigene 2D-Rasterkarte der Umgebung angelegt, die den relevanten Bereich der jeweiligen Konfiguration beinhaltet. Die zu den Konfigurationen gehörenden Bereiche der globalen Karte werden iterativ durch die 2D-Rasterkartenausschnitte ersetzt. Es wird für jede Konfiguration getestet, wie die Sensordaten mit der globalen Karte übereinstimmen. In [5] wird zusätzlich noch der zeitliche Verlauf der Konfigurationen berücksichtigt und es werden 2D-Rasterkarten für unterschiedliche Bereiche der globalen Karte und unterschiedliche Dauer der Veränderung erzeugt.

Roboterlokalisierung durch SLAM

Im Folgenden werden verschiedene SLAM-Verfahren beschrieben, die eine 3D-Sensorkonfiguration voraussetzen und den Roboter mit allen sechs Freiheitsgraden lokalisieren. Diese Verfahren erzeugen parallel zur Lokalisierung eine 3D-Repräsentation der Umgebung und eignen sich somit unbekannte Umgebungen zu kartieren. In dieser Arbeit wird ein SLAM-Verfahren eingesetzt, um ein 3D-Modell der Arbeitsumgebung zeitlich vor dem Einsatz des LiSA-Roboters zu erstellen.

In [93] wird die Roboterumgebung mit einem 3D-Laserscanner (vgl. Kapitel 5.1) dreidimensional erfasst. In die Punktwolke der Umgebung werden mit einer RANSAC basierten Methode (vgl. Anhang B.2) ebene Fragmente identifiziert. Diese dienen als Eingabedaten für ein Extended-Kalman-Filter basiertes SLAM-Verfahren. Da die 3D-Erfassung der Umgebung nicht kontinuierlich ist, kann der Roboter seine globale Pose nur bei jeder Aufnahme eines 3D-Scans korrigieren. Dasselbe Problem besteht in dem in [58] verwendeten Verfahren, da hier eine vergleichbare Hardware zur 3D-Umgebungsaufnahme verwendet wird. Im Gegensatz zu dem oben beschriebenen Verfahren werden aus der Punktwolke der Umgebung keine Merkmale wie ebene Fragmente extrahiert, sondern es wird direkt mit den Punkten der Punktwolke gerechnet. Mit Hilfe eines „Iterative Closest Points“-Ansatzes wird bestimmt, welche Transformation zwei nacheinander aufgenommene 3D-Scans am besten zur Deckung bringt. Über diese Transformation erhält man den Poseversatz zwischen den Aufnahmeorten und kann anhand dessen die globale Roboterpose bestimmen. Neben der fehlenden kontinuierlichen 6D-Lokalisierung setzten beide Verfahren voraus, dass nacheinander aufgenommene Scans überlappende Bereiche besitzen müssen, damit über die extrahierten Ebenenfragmente bzw. Punktbeziehungen eine Lagebestimmung der Scans zueinander möglich ist.

In [76] wird der Roboter anhand von rotations- und skalierungsinvarianten SIFT-Merkmalen lokalisiert. Diese werden aus Kamerabildern extrahiert, während der Roboter fährt. Durch die Identifizierung von gleichen SIFT-Merkmalen in unterschiedlichen Kameradatensätzen lassen sich die SIFT-Merkmale im dreidimensionalen Raum lokalisieren und der Roboter kann bei mehreren gefundenen Merkmalen seine Pose in allen sechs Dimensionen relativ zu den gefundenen Merkmalen bestimmen.

Ferner existieren Verfahren, die den Roboter kontinuierlich mit Hilfe von SLAM mit drei Freiheitsgraden lokalisieren, obwohl die Sensorkonfiguration eine kontinuierliche Erfassung der Umgebung in 3D ermöglicht. In [95] wird die kontinuierliche 3D-Umgebungserfassung über einen permanent rotierenden 2D-Laserscanner erreicht. Analog zu dem in dieser Arbeit verwendeten Verfahren werden die Daten der Laserscanner in die 2D-Ebene, in der der Roboter sich bewegt, projiziert. Die Lokalisierung wird anhand dieser Daten realisiert. Neben dem Vorteil der geringeren Rechenkomplexität durch die Verwendung von 2D-Daten besitzt das in [95] entwickelte System den Vorteil, dass der Laserscanner vertikal angebracht ist. Die Herangehensweise, immer den entferntesten Punkt eines Scans zu projizieren, führt zu einer robusten Lokalisierung, da diese Punkte im Allgemeinen statische Punkte wie die Gebäudegeometrie repräsentieren.

Ferner existieren sehr viele probabilistische SLAM-Ansätze, die von einem horizontal angebrachten Laserscanner ausgehen und eine Lokalisierung mit drei Freiheitsgraden realisieren. So wird beim GMapping-Verfahren [27], DP-SLAM-Verfahren [19] sowie beim GridSLAM-Verfahren [30] ein Partikelfilter basierter Ansatz verwendet, der Rasterkarten der Umgebung erzeugt und den Roboter in diesen mit drei Freiheitsgraden lokalisiert.

3.2.2 Hindernisvermeidungsalgorithmen

In dieser Arbeit wird mit dem „Virtual Range Scans“ (VRS)-Konzept ein Verfahren entwickelt, das Sensordaten verschiedener 3D-Sensoren durch Projektion in einen virtuellen 2D-Datensatz kombiniert. Dadurch ist es in der Lage, klassische Hindernisvermeidungsalgorithmen, die eine zweidimensionale Sensorkonfiguration voraussetzen, zur 3D-Hindernisvermeidung einzusetzen. Dies wird in dieser Arbeit mit der „Nearness Diagram“- und der „Vector Field Histogram“-Methode demonstriert. Im Folgenden werden zunächst exemplarisch weitere, nicht in dieser Arbeit verwendete, klassische zweidimensionale Ansätze vorgestellt, die ebenfalls mit dem VRS-Konzept kombiniert werden können. Anschließend wird auf alternative Verfahren eingegangen, die eine 3D-Hindernisvermeidung realisieren.

Verfahren zur 2D-Hindernisvermeidung

Alle zweidimensionalen Hindernisvermeidungsverfahren gehen von einer Sensorkonfiguration aus, die Abstandswerte in einer zweidimensionalen Ebene zur Roboterumgebung liefert.

Dynamic Window Approach: Der Dynamic Window Approach (DWA) wurde 1997 von Fox et. al. vorgestellt [23]. Er berücksichtigt im Gegensatz zu vielen anderen Verfahren die Dynamik des Roboters und somit die Nebenbedingungen, die für die optimale Transversal- bzw. Rotationsgeschwindigkeit bei einem gegebenen kinematischen Zustand des Roboters gelten. Dazu sucht es im Geschwindigkeitsraum nach optimalen Geschwindigkeitskombinationen (Translation und Rotation), die innerhalb eines kurzen Zeitintervalls erreichbar sind (Dynamic Window). Bewertet werden die Geschwindigkeitskombinationen durch eine Funktion, die die Zielrichtung, die Lage der Hinder-

nisse und die aktuelle Geschwindigkeitsrichtung berücksichtigt. Dies wurde zunächst in einem zweischrittigen Verfahren erreicht, bei dem erst die optimale Geschwindigkeit und anschließend die dazu passende Rotationsgeschwindigkeit ermittelt wurde. Später wurde die Idee von dem parallel, aber unabhängig entwickelten „Curvature Velocity"-Verfahren übernommen, das dieselbe Strategie zur Bestimmung der Translations- bzw. Rotationsgeschwindigkeit verwendet, jedoch mit einer Funktion, die parallel die optimale Kombination beider Geschwindigkeiten bestimmt [77].

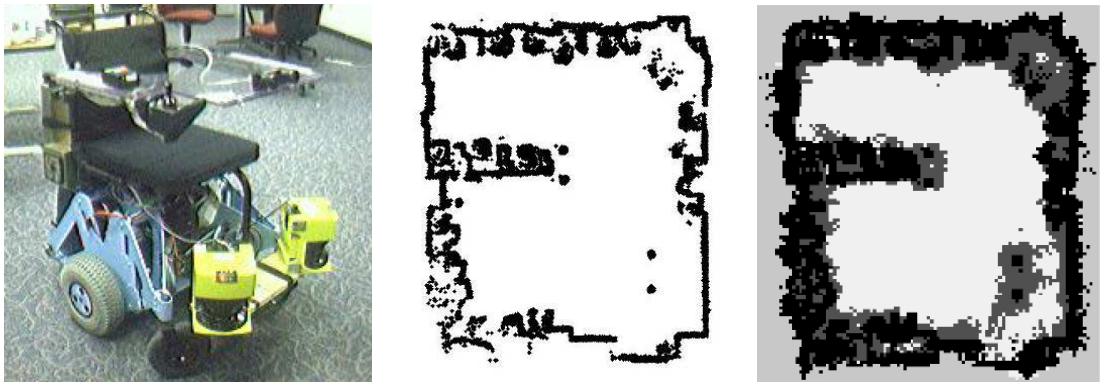
Potential Feld Ansatz: Der von Khatib 1985 vorgestellte Ansatz erzeugt ein künstliches Potentialfeld anhand der Roboterumgebung [35]. Aus diesem Potentialfeld werden anhand der aktuellen Roboterpose zwei Kräfte extrahiert. Die erste Kraft zieht den Roboter in Richtung des Ziels, die zweite drückt den Roboter von Hindernissen weg. Die Überlagerung der beiden Kräfte ergibt den Bewegungsvektor für die jeweilige Situation des Roboters.

Elastic Band Ansatz: Der Elastic Band Ansatz wurde 1993 von Quinlan und Khatib entwickelt [64]. Er ist eine Kombination aus globaler und lokaler Bewegungsplanung, die ausschließlich die Kinematik des Roboters berücksichtigt. Bei gegebener Umgebungskarte wird zunächst von einem globalen Planer ein initialer Pfad geplant. Dieser wird an dynamischen Hindernissen deformiert, so dass er sicher um die Hindernisse herumführt, ohne den globalen Pfad zum Ziel zu verlieren.

Obstacle Restriction Method: Die „Obstacle Restriction Method" (ORM) wurde 2005 von Minguez vorgestellt [47]. ORM ist ein zweistufiges Verfahren, das zunächst anhand der aktuellen Sensordaten Zwischenziele berechnet, die anhand der aktuell sichtbaren Hindernisse einen optimalen Weg zum Ziel erzeugen. Anschließend wird anhand des gewählten Zwischenziels und dessen Entfernung zu den vorhandenen Hindernissen die optimale Bewegungsrichtung und Geschwindigkeit bestimmt. Dieses vorausschauende Fahrverhalten lässt den Roboter nicht in „U"-förmige Hindernisse, sogenannte „Trap"-Situationen, hineinfahren. Es zeigt in engen Umgebungen eine vergleichbare Performanz mit dem in dieser Arbeit verwendeten „Nearness Diagram"(ND)-Verfahren, führt aber im freien Raum zu einer gleichmäßigeren Bewegung des Roboters. Obwohl ORM also Vorteile gegenüber dem ND-Verfahren besitzt, wird in dieser Arbeit das ND-Verfahren verwendet, da bereits eine Implementation in dem eingesetzten Softwareframework besteht.

Verfahren zur 3D-Hindernisvermeidung

In diesem Abschnitt werden Verfahren erläutert, die eine Hindernisvermeidung für verschiedene Sensorkonfigurationen zur 3D-Umgebungserfassung realisieren. Im Folgenden werden zunächst Verfahren beschrieben, die im 3D-Raum rechnen und Bewegungen mit sechs Freiheitsgraden zulassen. Anschließend wird auf Verfahren eingegangen, die, analog zu dem in dieser Arbeit entwickeltem VRS Konzept, 3D-Daten zur Hindernisvermeidung mit drei Freiheitsgraden verwenden.



(a) **(b)** **(c)**

Abbildung 3.2: 3D-Umgebungswahrnehmung durch Kombination von Laserscannern mit einem Stereokamerasystem: (a) der autonome Rollstuhl ist mit zwei Laserscannern und einem Stereokamerasystem ausgestattet. (b) Mit Hilfe der Laserscanner des Rollstuhls erzeugte 2D-Rasterkarte. (c) Über die Rasterkarte wird die 3D-Information des Stereokamerasystems gelegt [55].

Obstacle Restriction Method: Vikenmark und Minguez stellen in [89] eine Erweiterung des bereits beschriebenen ORM-Verfahrens vor. Dieses lässt Bewegungen in allen sechs Freiheitsgraden zu, um den dreidimensional erfassten Hindernissen auszuweichen. Die Funktionalität des Verfahrens wurde bis jetzt leider nur in der Simulation erfolgreich getestet, so dass nicht geklärt ist, ob die erhöhte Rechenkomplexität durch Verwenden aller drei Dimensionen in einem realen Szenario sichere Hindernisvermeidung erlaubt.

3D-Dodger In [70] wird ein Verfahren vorgestellt, das eine 3D-Rasterkarte verwendet, um Hindernisvermeidung im dreidimensionalen Raum zu gewährleisten. Das Verfahren wird mit einem autonomem Helikopter getestet. Dieser ist mit einem Laserscanner ausgestattet, der während des Flugs eine 3D-Punktwolke der Umgebung erzeugt. Zur Hindernisvermeidung werden die Punkte in eine 3D-Rasterkarte eingetragen. Anhand der aktuellen Roboterpose werden für einen pyramidenförmigen Ausschnitt in Flugrichtung innerhalb der Rasterkarte die dichtesten Hindernisse bestimmt. Anhand dieser Hindernisse wird der 2D-Hindernisvermeidungsalgorithmus Dodger [38] für die horizontale und vertikale Richtung des pyramidenförmigen Ausschnitts berechnet. Der relevante Bereich der Rasterkarte wird anhand der aktuellen Fluggeschwindigkeit begrenzt, um das Verfahren rechnerisch kontrollieren zu können.

Optischer Fluss Optischer Fluss ist eine Technik, die auf Kameradaten beruht. Sie wird häufig verwendet, um über den Vergleich von Helligkeitsgradienten aufeinanderfolgender Bilder Bewegungen in Kameradaten zu identifizieren [8]. Beyeler et. al stellen 2007 ein Verfahren vor, das über den Optischen Fluss eine 3D-Hindernisvermeidung eines nur 10 g schweren Ultraleichtfliegers realisiert [4]. In [56] wird der Optische Fluss in einem biologisch motivierten Ansatz als Eingabe eines Neuronalen Netzes verwendet, welches 3D-Hindernisvermeidung realisiert. Im Gegensatz zu dem Ansatz von Beyeler [4] ist das Verfahren jedoch ausschließlich in der Simulation getestet.

(a) **(b)** **(c)**

Abbildung 3.3: 3D-Umgebungswahrnehmung des autonomen Roboters Kurt3D. (a) Der Roboter ist mit einem 3D-Laserscanner ausgestattet. (b) Obwohl die Dosen unterhalb der Laserscannerebene liegen (bei horizontaler Ausrichtung), sind die Daten durch permanentes Nicken des Laserscanners in der Karte vorhanden und können zur Hindernisvermeidung verwendet werden (c) [32]

Neben den oben beschriebenen Hindernisvermeidungsverfahren, die im 3D-Raum rechnen und Bewegungen mit sechs Freiheitsgraden zulassen, existiert eine weitere Gruppe von Verfahren, die eine 2D-Projektion der 3D-Daten zur 3D-Hindernisvermeidung verwenden. Diese Verfahren lassen jedoch nur eine Bewegung mit drei Freiheitsgraden zu. Im Folgenden werden verschiedene Systeme mit unterschiedlichen Sensorkonfigurationen zur 3D-Umgebungserfassung vorgestellt und auf deren Vorgehensweise zur 3D-Hindernisvermeidung eingegangen.

Ein häufig eingesetzter Sensor zur 3D-Umgebungswahrnehmung ist ein Stereokamerasystem. Dieses besteht aus zwei Kameras, die unter einer festen Vergenz dieselbe Szene betrachten. Über den bekannten Abstand der Kameras und die Vergenz lassen sich Objekte in den Kamerabildern über Triangulation in den dreidimensionalen Raum abbilden. In [55] wird ein mit horizontalen Laserscannern und einem Stereokamerasystem ausgestatteter Rollstuhl vorgestellt (vgl. Abbildung 3.2). Zunächst wird mit Hilfe eines SLAM-Verfahrens und der beiden Laserscanner eine 2D-Karte der Umgebung erstellt. Diese ist in Teilabbildung (b) zu sehen. Hier wird die Notwendigkeit der 3D-Umgebungserfassung deutlich, da unten rechts in der Karte ausschließlich zwei Tischbeine, jedoch nicht die für den Roboter gefährliche Tischplatte, zu sehen sind. In einem weiteren SLAM-Verfahren wird mit Hilfe des Stereokamerasystems eine 3D-Punktwolke der Umgebung erzeugt. Diese wird auf die Höhe der mit den Laserscannern erstellten 2D-Rasterkarte projiziert und mit dieser überlagert. Diese Karte ist in Teilabbildung 3.2c zu sehen und enthält alle Hindernisse im 3D-Raum, mit denen der Roboter zusammenstoßen kann. Der Aufnahmeprozess der Umgebung findet im Vorfeld des Robotereinsatzes statt. Bei der Aufnahme muss sichergestellt sein, dass keine Dynamik in der Szene vorhanden ist. Somit wird die 3D-Repräsentation der Umgebung im Gegensatz zu dem in dieser Arbeit vorgestellten Verfahren nicht kontinuierlich aktualisiert und in den Hindernisvermeidungsprozess integriert. Das in [55] vorgestellte Verfahren ist also eine globale Pfadplanung unter Berücksichtigung der 3D-Geometrie in Kombination mit einer 2D-Hindernisvermeidung durch die Laserscanner.

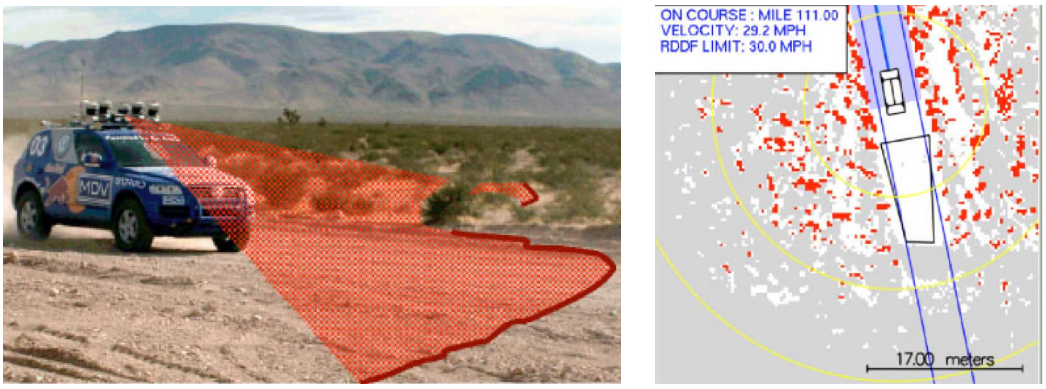
(a) **(b)** **(c)**

Abbildung 3.4: (a) Autonomer Roboter mit permanent rotierendem vertikalen Laserscanner. In einer Szene (b) werden die dichtesten Punkte pro Scan (blau) zur Hindernisvermeidung und diejenigen mit der größten Entfernung zur Lokalisierung verwendet (rot) (c). Die schwarzen Punkte bestimmen die Aufnahmepositionen der 3D-Scans [95].

Die im Folgenden vorgestellten Verfahren verwenden anstelle der Stereokamera unterschiedliche Konfigurationen von Laserscannern zur 3D-Umgebungswahrnehmung. Um mit einem 2D-Laserscanner eine 3D-Umgebungswahrnehmung zu realisieren, wird der Scanner in [32, 69] mit einer horizontalen bzw. vertikalen Nickvorrichtung ausgestattet (vgl. Kapitel 5.1). In [95] wird der Scanner vertikal angebracht und permanent rotiert. In [85] ist der Laserscanner schräg am Roboter angebracht und die 3D-Information wird analog zur LiSA-Sensorkonfiguration über die Eigenbewegung des Roboters erfasst.

In [32] wird die 3D-Umgebungswahrnehmung des in Abbildung 3.3a dargestellten autonomen mobilen Roboter Kurt3D durch einen 3D-Laserscanner realisiert. In Teilabbildung 3.3b ist ein Parcours zu sehen, bei dem es mit horizontal ausgerichtetem Laserscanner zu einer Kollision mit den roten Blechdosen kommen würde, da sich diese unterhalb der Laserscannerebene befinden. Aus diesem Grund wird der Laserscanner permanent um seine horizontale Achse genickt. Die durch die Nickbewegung erhaltenen 3D-Punkte werden in eine 2D-Rasterkarte projiziert, so dass diese die Blechdosen enthält (Teilabbildung 3.3c). Diese Karte ist im Gegensatz zu dem in dieser Arbeit verwendeten Verfahren nicht global, sondern rechnet im lokalen Roboterkoordinatensystem. Daraus folgt, dass die gesamte Karte kontinuierlich anhand der Roboterbewegung aktualisiert werden muss. Die Berücksichtigung dynamischer Hindernisse erfolgt durch einen Alterungsprozess auf der Rasterkarte. Über eine Analyse des Freiraums in der Karte wird diese in den Hindernisvermeidungsprozess integriert.

Das in [95] beschriebene Verfahren verwendet den Vorteil des permanent rotierenden vertikalen Laserscanners (vgl. Abbildung 3.4a). Da ein einzelner Scan einen vertikalen Schnitt der Roboterumgebung liefert, enthält dieser die Daten aller Hindernisse des entsprechenden Winkels. Zur Hindernisvermeidung wird nun ein virtueller 2D-Scan erstellt, indem alle Punkte eines vertikalen Scans in die 2D-Ebene projiziert werden und der Punkt mit dem geringsten Abstand den Entfernungswert für den entsprechenden Winkel im virtuellen, horizontalen 2D-

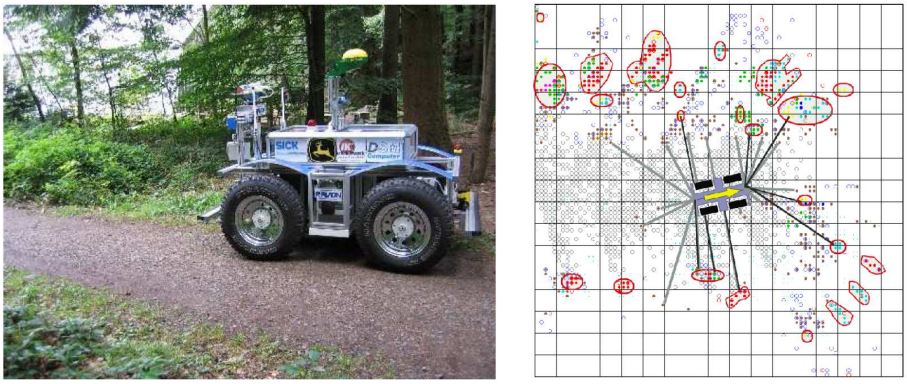


(a) **(b)**

Abbildung 3.5: Hindernisvermeidung des autonomen Fahrzeugs Stanley: (a) Das Fahrzeug ist mit fünf fest angebrachten Laserscannern ausgestattet, die vom Dach des Fahrzeugs unter verschiedenen Winkeln schräg nach unten auf den vor dem Fahrzeug liegenden Bereich gerichtet sind. (b) In der von den Laserscannern während der Fahrt erzeugten Punktwolke (rot) werden Hindernisse identifiziert und ein maximales Rechteck mit befahrbarem Bereich wird vor dem Fahrzeug bestimmt [85].

Scan liefert. In Teilabbildung (b) ist eine Szene abgebildet, die mit dem Roboter abgefahren wird. Die Punkte, die während dieser Fahrt den virtuellen Scans zugeordnet werden, sind in Teilabbildung (c) blau dargestellt. Das Verfahren ist im Gegensatz zu dem in dieser Arbeit vorgestellten VRS-Konzept sehr stark auf die verwendete Hardware abgestimmt. Diese erlaubt im Gegensatz zur LiSA-Sensorkonfiguration keine permanente Erfassung der gesamten 3D-Umgebung. So benötigt der Scanner $2,4$ Sekunden für eine 360°-Drehung. Diese Zeit ist jedoch auflösungsabhängig und kann durch die Verwendung von zwei parallel rotierenden Scannern halbiert werden. So ist eine bis zu einer bestimmten Geschwindigkeit sichere Hindernisvermeidung möglich. Diese Sensorkonfiguration ist jedoch auf dem LiSA-Roboter nicht verwendbar, da dieser durch seine Maße und den verwendeten Manipulator zu einer großen Selbstverdeckung der Laserstrahlen führen würde.

In [85] ist „Stanley“ beschrieben, das autonome Fahrzeug, welches 2005 die DARPA Grand Challange gewonnen hat. Wie in Abbildung 3.5a zu sehen ist, besitzt Stanley eine Sensorkonfiguration von fünf fest auf dem Dach des Fahrzeugs angebrachten Laserscannern. Die Laserscanner sind unter verschiedenen Winkeln schräg nach unten auf den vor dem Fahrzeug liegenden Bereich gerichtet. Über die Eigenbewegung des Fahrzeugs erzeugen diese Laserscanner eine 3D-Punktwolke der Umgebung. Eine wahrscheinlichkeitsbasierte Methode wird verwendet, um Höhensprünge in der Punktwolke zu identifizieren. Diese Höhensprünge sind nicht befahrbare positive oder negative Hindernisse und werden in eine 2D-Rasterkarte eingtragen. Um die optimale Trajektorie unter Berücksichtigung der Hindernisse zu finden, wird in die 2D-Rasterkarte ein Viereck gefittet, das ausschließlich navigierbare Rasterzellen enthält. Teilabbildung (b) ist eine Aufsicht auf die während der Fahrt erhaltene Punktwolke und dem aus dieser als befahrbar klassifizierten Viereck. Nach einer Verifikation des maximal befahrbaren Vierecks durch Kameradaten wird unter Berücksichtigung des aktuellen kine-



(a) **(b)**

Abbildung 3.6: 3D-Umgebungswahrnehmung des autonomen Roboters RAVON: (a) Der Roboter ist mit einem vertikalen Laserscanner ausgestattet, der permanent um die vertikale Achse genickt wird. (b) In der aus der 3D-Punktwolke erzeugten 2D-Rasterkarte wird ein virtueller Abstandsscan generiert [69].

matischen Zustands des Fahrzeugs eine optimale Trajektorie bestimmt. Die Hindernisvermeidung betrachtet keine Dynamik der Szene, da es für die Solofahrt in der Wüste konzipiert ist.

Der in Abbildung 3.6a gezeigte autonome mobile Roboter RAVON ist mit einem um die vertikale Achse nickenden Laserscanner ausgestattet [69]. In der von diesem 3D-Laserscanner erhaltenen Punktwolke werden zunächst Häufungspunkte identifiziert. Die einzelnen Häufungspunkte werden anhand ihrer Höhe klassifiziert. Ferner wird bestimmt, ob es sich um solide Hindernisse (Steine,...) oder flexible Hindernisse (Vegetation, Gräser,...) handelt. Ebenfalls wird die Steigung und Höhenlage des Bodens bestimmt. Diese Eigenschaften werden in Rastern einer globalen 2D-Rasterkarte, die eine Auflösung von 4,5 cm besitzt, gespeichert. Analog zu dem bereits 2007 in [79] vorgestellten Verfahren wird die 2D-Karte anschließend verwendet, um ausgehend von der aktuellen Roboterpose einen virtuellen 2D-Scan der Umgebung zu erzeugen. Dieser Scan kann die Eigenschaften der Raster berücksichtigen, so dass z.B. ein virtueller Scan mit ausschließlich soliden Hindernissen erzeugt wird. Teilabbildung 3.6b zeigt den virtuellen Sensor (graue Linien), der zu der Situation aus Teilabbildung 3.6a gehört. Die 2D-Rasterkarte arbeitet, ebenso wie beim bereits in [79] vorgestelltem VRS-Konzept, als Hindernisgedächtnis und ermöglicht die Integration beliebiger Abstandssensoren in den Hindernisvermeidungsprozess. Im Gegensatz zu dem in [69] vorgestellten Verfahren berücksichtigt das VRS-Konzept (vgl. Kapitel 6.2.2) zusätzlich Dynamik im 3D-Umfeld des Roboters.

Kapitel 4

Konzept eines semantisch gestützten Navigationssystems

4.1 Zielsetzung

Aus dem Stand der Technik ist ersichtlich, dass viele Robotersysteme 2D-Sensorkonfigurationen haben. 3D-Sensordaten werden nur selten zur Hindernisvermeidung und Lokalisierung eingesetzt. Die Systeme, die 3D-Sensordaten verwenden, nutzen dazu Verfahren, die speziell auf die vorhandene Sensorkonfiguration abgestimmt sind. Dies macht diese Verfahren unflexibel und für andere Systeme nicht wiederverwendbar. Da auf der anderen Seite 3D-Hindernisvermeidung eine notwendige Anforderung an die sichere Navigation autonomer Assistenzroboter ist, besteht eine Diskrepanz zwischen dieser Anforderung und dem Stand der Technik. Diese Lücke wird im Rahmen dieser Arbeit geschlossen.

Zielsetzung ist es, 3D-Sensordaten in den Hindernisvermeidungs- und Lokalisierungsprozess eines autonomen mobilen Roboters zu integrieren. Diese Integration muss flexibel sein, so dass beliebige Konfigurationen von Abstandssensoren verwendet werden können. Ein entscheidender Punkt ist ebenfalls, dass bestehende zweidimensionale Hindernisvermeidungsstrategien wiederverwendet und ohne Aufwand ausgetauscht werden können. Ferner zeigt der Stand der Technik, dass keine Sensorkonfigurationen eingesetzt werden, die permanent die gesamte Hülle des Roboters abdecken. Dies macht es notwendig, dass sich das Hindernisvermeidungsverfahren Hindernisse merkt, die in den Sensordaten auftauchen. Problematisch bei dem Einsatz von Hindernisgedächtnissen ist es, dass gleichzeitig dynamische Objekte identifiziert werden müssen. In [85, 32] werden bereits Hindenisgedächtnisse eingesetzt. Jedoch wird die Dynamik in der Umgebung des Roboters ignoriert, so dass diese das Hindernisgedächtnis füllen und den Roboter blockieren würde. Neben der oben beschriebenen Flexibilität, soll das in dieser Arbeit entwickelte Verfahren sowohl ein Hindernisgedächtnis besitzen, als auch die Dynamik in der Umgebung des Roboters berücksichtigen.

Im Bereich der Lokalisierung zeigt der Stand der Technik, dass 3D-Sensordaten nur durch

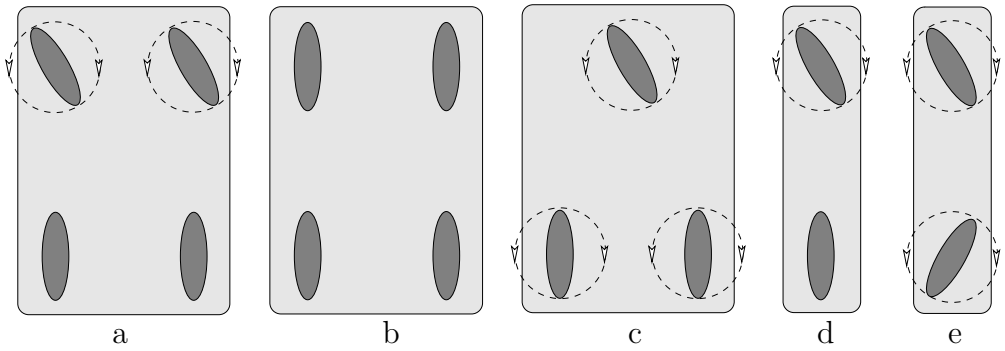


Abbildung 4.1: Kinematische Modelle: a) Ackermann b) Differential c) Omnidirektional d) Zweirad e) Zweirad mit zwei Rotations-Freiheitsgraden

Projektion auf eine 2D-Ebene in den Lokalisierungsprozess integiert werden [95, 32]. Oder es werden Verfahren eingesetzt, die zwar 3D-Sensordaten verwenden jedoch aufgrund der dadurch entstehenden hohen Rechenkomplexität keine permanente Lokalisierung erreichen und die Roboterpose nur in größeren Abständen korrigieren können. In dieser Arbeit wird ein Verfahren entwickelt, das ohne Projektion der 3D-Sensordaten arbeitet. Es soll vor allem darauf eingegangen werden, wie es möglich ist, die Rechenkomplexität, die beim Arbeiten mit 3D-Daten entsteht, zu reduzieren, um einen effizienten Vergleich zwischen einem 3D-Modell und 3D-Sensordaten zu realisieren. Darüber hinaus soll die Lokalisierung ebenfalls unabhängig von der Konfiguration der Abstandssensoren sein, so dass der Stand der Technik erweitert wird und die Beschränkung auf horizontale Abstandssensoren, die durch die Verwendung von 2D-Umgebungsmodellen besteht, aufgehoben wird. Ein weiterer wichtiger Aspekt bei der Integration von 3D-Daten in die Steuerung autonomer mobiler Roboter ist die einfache Erstellung eines 3D-Umgebungsmodells, da dieses für Verwendung von 3D-Daten zur Lokalisierung Voraussetzung ist. In dieser Arbeit wird geprüft, in wie weit diese Geometrieabbildung der Umgebung und die Erstellung einer Simulationsumgebung aus diesen Daten automatisiert werden kann.

Diese Arbeiten werden im Rahmen des LiSA-Projekts entwickelt und getestet. Dadurch erweitert sich die Zielsetzung dieser Arbeit insofern, dass die für die Navigation des LiSA-Roboters notwendigen Anforderungen (vgl. Abschnitt 2.3) ebenfalls gelöst werden müssen. Zum einen muss die Kinematik des omnidirektionalen Antriebs des LiSA-Roboters bestimmt werden. Die Lösung dieses Problems ist im folgendem Abschnitt beschrieben.

4.2 Kinematik des LiSA-Roboters

4.2.1 Kinematisches Modell

Das kinematische Modell definiert anhand charakteristischer Kenngrößen eines Systems ein mathematisches Rahmenwerk, um das kinematische Verhalten des Systems auf mess- bzw.

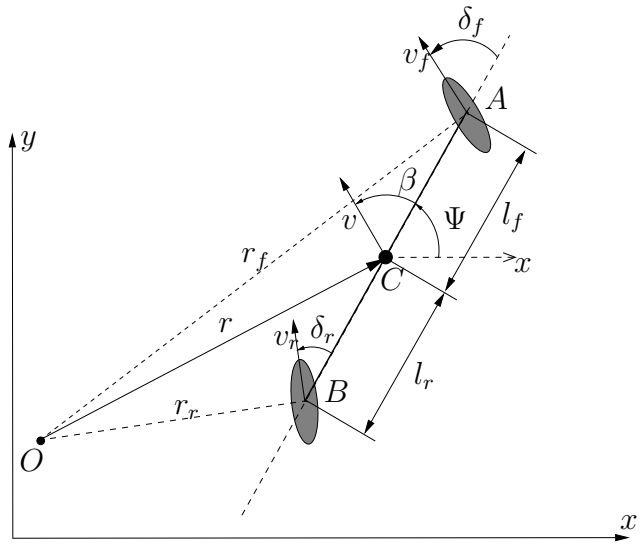


Abbildung 4.2: Schematische Darstellung des kinematischen Zweiradmodells mit zwei Rotations-Freiheitsgraden.

regelbare Größen zurückzuführen. Dieses erlaubt ein exaktes Steuern des Systems durch die regelbaren Größen (inverses kinematisches Modell) bzw. eine Bestimmung des aktuellen Zustands des Systems durch die messbaren Größen (kinematisches Modell). Der Begriff kinematisches Modell wird häufig im Zusammenhang der Manipulatorsteuerung verwendet. Der wesentliche Unterschied zwischen dem kinematischen Modell eines Manipulators und dem eines mobilen Roboters ist der, dass sich die aktuelle Pose des Manipulators direkt über die Regel- und Messgrößen bestimmen lässt, wohingegen das kinematische Modell eines mobilen Roboters dessen aktuelle Pose nicht direkt aus den Sensordaten bestimmen kann. Dies liegt daran, dass sich ein mobiler Roboter frei in seiner Umgebung bewegen kann, so dass das kinematische Modell nur eine von der Zeit abhängige Posedifferenz im lokalen Roboter-Koordinatensystem liefert. Die Bestimmung der aktuellen Pose im globalen Koordinatensystem kann nur durch die Integration dieser Posedifferenzen über die Zeit erreicht werden. Dadurch sind die Abweichungen der Posebestimmung bei mobilen Robotern größer. Dies führt dazu, dass die durch das kinematische Modell erhaltene Pose häufig nur als Schätzung für ein Lokalisierungsverfahren verwendet wird, welches weitere Sensoren verwendet, um die aktuelle Pose des Roboters zu bestimmen (vgl. 6.1).

Bei mobilen radgetriebenen Systemen wird anhand der Geometrie des Fahrzeugs sowie der Lage und Anzahl der angetriebenen bzw. passiven Räder zwischen verschiedenen kinematischen Modellen unterschieden. In Abbildung 4.1 ist eine Auswahl von verschiedenen kinematischen Modellen zu sehen. Im Folgenden wird das kinematische und inverse kinematische "Zweiradmodells mit zwei Rotations-Freiheitsgraden" (vgl. Abbildung 4.1 e) hergeleitet, da sich die Kinematik des LiSA-Roboters, wie in Abschnitt 4.2.3 erläutert wird, auf dieses zurückführen lässt.

Wie in Abbildung 4.2 zu sehen ist, werden folgende Variablen zur Beschreibung des Modells

verwendet:

v_f Geschwindigkeit des Vorderrades A

v_r Geschwindigkeit des Hinterrades B

δ_f Auslenkungswinkel des Vorderrades A in Bezug zur Hauptachse AB

δ_r Auslenkungswinkel des Hinterrades B in Bezug zur Hauptachse AB

C Referenzpunkt des Roboters

l_f Abstand zwischen Vorderrad A und Referenzpunkt C

l_r Abstand zwischen Hinterrad B und Referenzpunkt C

v Geschwindigkeit des Roboters im Referenzpunkt C

Ψ Gierwinkel: Winkel zwischen der x-Achse und der Hauptachse AB

β Schwimmwinkel: Winkel zwischen der Hauptachse AB und der Geschwindigkeitsrichtung

r Abstand des Rotationsmittelpunktes O von dem Referenzpunkt C

Von diesen Variablen sind die Radwinkel δ_f und δ_r sowie die Radgeschwindigkeiten v_f und v_r mess- und regelbar, so dass das kinematische Modell die Pose (x_t, y_t, Ψ_t) des Roboters zum Zeitpunkt t auf diese zurückführen muss.

Die Herleitung des kinematischen Modells beruht auf der Annahme, dass auf die Räder des Fahrzeugs keine lateralen Kräfte wirken. Dies ist gleichbedeutend mit der Aussage, dass der Geschwindigkeitsvektor der Räder A und B in Richtung der Radstellungen δ_f bzw. δ_r zeigt. Diese Annahme ist realistisch für das Szenario des LiSA-Roboters, da sich dieser mit einer maximalen Geschwindigkeit von $1\frac{m}{s}$ bewegt. Nach [66] führt erst eine Orientierungsänderung des Fahrzeugs mit einer Geschwindigkeit von mehr als $5\frac{m}{s}$ zum Driften des Fahrzeugs. Neben der Geschwindigkeitsbegrenzung fordert die Annahme, dass die Radgeschwindigkeiten den Radorientierungen angepasst sind, so dass die Motoren nicht gegeneinander arbeiten und es nicht zum Schieben, also einer lateralen Kraft, über ein Rad kommt.

Die in Anhang A.1 beschriebene geometrische Herleitung ergibt für die zeitliche Ableitung der Orientierung:

$$\boxed{\dot{\Psi} = \frac{v\cos(\beta)\left(\tan(\delta_f) - \tan(\delta_r)\right)}{l_f + l_r}}. \tag{4.1}$$

Der in dieser Gleichung nicht direkt messbare Schwimmwinkel β sowie die nicht messbare Geschwindigkeit v müssen zur Berechnung des kinematischen Modells auf die messbaren Radwinkel und Radgeschwindigkeiten zurückgeführt werden. Unter der Annahme, dass es

sich bei dem Fahrzeug um einen starren Körper handelt, gilt:

$$\beta = \arctan\left(\frac{l_f \tan\delta_r + l_r \tan\delta_f}{l_f + l_r}\right) \tag{4.2}$$

$$v = \frac{v_f \cos(\delta_f) + v_r \cos(\delta_r)}{2\cos(\beta)}. \tag{4.3}$$

Die Unabhängigkeit des Schwimmwinkels von den Radgeschwindigkeiten basiert auf der ursprünglichen Annahme des Modells, dass auf die Räder keine lateralen Kräfte wirken.

Um die globale Orientierung zu einem Zeitpunkt t zu bestimmen, wird das Integral über die Zeit gebildet. Hierzu wird in der Praxis die Zeit in Zeitintervalle $\triangle t$ diskretisiert. Durch Multiplikation von $\dot{\Psi}$ mit $\triangle t$ wird die Orientierungsdifferenz für das Zeitintervall bestimmt. Durch Aufsummation der Orientierungsdifferenzen über alle Zeitintervalle bis zum Zeitpunkt t wird die Orientierung zum Zeitpunkt t bestimmt.

$$\Psi_t = \Psi_{t-1} + \dot{\Psi}_t \triangle t$$

Die Bewegungsänderungen in x- und y-Richtung sind über die Projektion des Geschwindigkeitsvektors anhand der aktuellen globalen Orientierung und der Geschwindigkeitsrichtung auf die jeweiligen Achsen zu bestimmen:

$$\boxed{\dot{x} = v\cos(\Psi_t + \beta)} \quad \text{und} \quad \boxed{\dot{y} = v\sin(\Psi_t + \beta)}. \tag{4.4}$$

Analog zur Orientierung wird die absolute Position über die diskretisierte Integration über die Zeit erreicht.

$$x_t = x_{t-1} + \dot{x} \triangle t$$
$$y_t = y_{t-1} + \dot{y} \triangle t$$

4.2.2 Inverses Kinematisches Modell

Das inverse kinematische Modell führt die Variablen zur Steuerung des Fahrzeugs $\dot{\Psi}$, β und v auf die regelbaren Größen δ_f, δ_r, v_f und v_r zurück. Häufig wird auch die absolute Pose (x_t, y_t, Ψ_t) auf regelbare Größen zurückgeführt. In dieser Arbeit wird jedoch eine reaktive Hindernisvermeidung verwendet, die als Ausgabe die optimalen Werte $\dot{\Psi}$, β und v für den aktuellen Zustand des Roboters (x_t, y_t, v_t, Ψ_t) unter Berücksichtigung der Roboterumgebung liefert (vgl. Kapitel 6.2.3). Somit kann der Zwischenschritt, $(\dot{\Psi}, \beta, v)$ auf eine optimale nächste Pose $(x_{t+1}, y_{t+1}, \Psi_{t+1})$ und diese anschließend wieder anhand der aktuellen Pose auf eine optimale Kombination von $(\dot{\Psi}, \beta, v)$ zurückzurechnen, weggelassen werden.

Obwohl die Kinematik des Zweiradmodells mit zwei Freiheitsgraden die Möglichkeit bietet, beliebige Geschwindigkeitsrichtungen mit beliebigen, überlagerten Orientierungsänderungen auszuführen, wird dies in der Praxis häufig nicht ausgeschöpft. Statt dessen wird

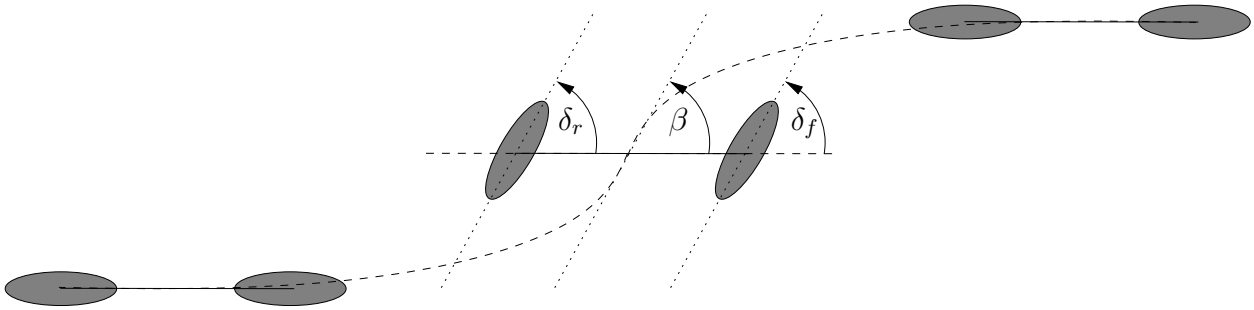


Abbildung 4.3: Schematische Darstellung des Parallel Steering-Manövers. Beide Radwinkel werden immer auf die vorgegebene Bewegungsrichtung β eingestellt.

ausschließlich auf zwei spezielle Fahrverhalten zurückgegriffen, auf das sogenannte "Zero Side-slip"-Manöver und das "Parallel Steering"-Manöver [68]. Im Folgenden werden diese beiden Manöver beschrieben und das dazugehörige inverse kinematische Modell hergeleitet. Anschließend wird das allgemeine Modell hergeleitet, und es wird gezeigt, dass die beiden Manöver Spezialfälle des allgemeinen inversen kinematischen Modells sind.

Wie in in Abbildung 4.3 dargestellt ist, werden beim Parallel Steering-Manöver gleiche Radwinkel für Vorder- und Hinterrad verwendet ($\delta_f = \delta_r$). Damit ergibt sich aus Gleichung 4.2:

$$\beta = \delta_f = \delta_r.$$

Dies bedeutet, dass beim Parallel Steering-Manöver die Bewegungsrichtung immer in die Richtung zeigt, auf die die beiden Radwinkel eingestellt sind. Setzt man diese Beziehung in Gleichung 4.1 ein, folgt $\dot{\Psi} = 0$, so dass ein Parallel Steering-Manöver keine Orientierungsänderung des Fahrzeugs zur Folge hat. Für die Geschwindigkeit des Fahrzeugs im Referenzpunkt folgt aus Gleichung 4.3 $v = v_f = v_r$. Die Geschwindigkeit des Fahrzeugs entspricht also jederzeit derjenigen der einzelnen Radgeschwindigkeiten. Aus der Gleichung folgt ebenfalls, dass die Räder des Fahrzeugs sich immer mit derselben Geschwindigkeit bewegen müssen, anderenfalls würde auf die Räder eine laterale Kraft wirken, und die Grundannahme des Modells wäre verletzt.

Um eine Orientierungsänderung des Fahrzeugs zu erreichen, wird das Zero Side-slip-Manöver durchgeführt. Dieses setzt, wie der Name schon sagt, den Schwimmwinkel auf Null. Da der Schwimmwinkel die Winkeldifferenz zwischen Bewegungsrichtung und Hauptachse des Fahrzeugs beschreibt, führt die resultierende Geschwindigkeitsrichtung bei $\beta = 0$ immer in Richtung der Hauptachse des Fahrzeugs, was zu einem stabilen Fahrverhalten führt.

Die in Anhang A.2 beschriebene Herleitung ergibt für die Radwinkel in Bezug zur Orientierungsänderungsrate:

$$\delta_f = \arctan\left(\frac{\dot{\Psi} L}{2v}\right) \quad \text{und} \quad \delta_r = -\arctan\left(\frac{\dot{\Psi} L}{2v}\right). \tag{4.5}$$

Wobei ($l_f + l_r = L$) der Radabstand des Roboters ist. Für die Radgeschwindigkeiten, die bei einem Manöver mit $\beta = 0$ eingestellt werden müssen, gilt:

$$v_f = \frac{v}{\cos \delta_f} \quad \text{und} \quad v_r = \frac{v}{\cos \delta_r}. \tag{4.6}$$

Die Radwinkel für beliebige Geschwindigkeitsrichtungen und Orientierungsänderungen sind, nach der im Anhang beschriebenen Herleitung, folgenderweise einzustellen:

$$\boxed{\delta_f = \arctan\left(\tan\beta + \frac{\dot{\Psi} L}{2v\cos\beta}\right)} \quad \text{und} \quad \boxed{\delta_r = \arctan\left(\tan\beta - \frac{\dot{\Psi} L}{2v\cos\beta}\right)}. \tag{4.7}$$

Neben den passenden Radwinkeln für eine vorgegebene Richtung mit einer vorgegebenen Geschwindigkeit und einer bestimmten Orientierungsdifferenzrate müssen ebenfalls die passenden Radgeschwindigkeiten bestimmt werden. Wären die Radgeschwindigkeiten nicht an die Radstellungen angepasst, ist es möglich, dass die Motoren der angetriebenen Räder gegeneinander arbeiten, was zu einem schnellen Verschleiß und einer unvorhersagbaren Bewegung des Roboters führen würde.

Die Herleitung ergibt:

$$\boxed{v_f = \frac{\cos(\beta)}{\cos(\delta_f)} v} \quad \text{und} \quad \boxed{v_r = \frac{\cos(\beta)}{\cos(\delta_r)} v}. \tag{4.8}$$

Betrachtet man in Gleichung 4.7 den Grenzwert $\dot{\Psi} = 0$ ergibt sich $\delta_r = \delta_f = \beta$ und somit das Parallel Steering-Manöver. Wird ($\beta = 0$) gesetzt, ergeben sich die in Gleichung 4.5 und 4.6 ermittelten Gleichungen für das Zero Side-slip-Manöver.

Wie aus Gleichung 4.7 ersichtlich ist, gibt es zwei Sonderfälle, die getrennt betrachtet werden müssen. Zum einen sind die Radwinkel bei einer Geschwindigkeit von Null undefiniert. Dies ist der Fall, wenn das Fahrzeug steht bzw. sich auf der Stelle dreht. Der zweite Sonderfall ergibt sich für einen Winkel von $\beta = \frac{\pi}{2}$, da die Tangensfunktion dort eine Pol- und die Kosinusfunktion eine Nullstelle besitzt. Hierzu kommt es, wenn die gewünschte Bewegungsrichtung senkrecht zur Hauptachse des Fahrzeugs steht und der Rotationsmittelpunkt somit in die Hauptachse fällt.

4.2.3 Anpassung des Kinematischen Modells an den LiSA-Roboter

Wie in Abbildung 4.4 zu sehen ist, befinden sich die angetriebenen Räder des LiSA-Roboters auf der Roboterdiagonalen “hinten links” nach “vorne rechts”. Aus diesem Grund können sie nicht zu zwei virtuellen Rädern, die sich auf der Hauptachse des LiSA-Roboters befinden, vereinfacht werden. Vielmehr handelt es sich beim LiSA-Roboter um ein Zweiradmodell mit zwei Freiheitsgraden, dessen Hauptachse gegenüber der Roboterhauptachse um den Winkel

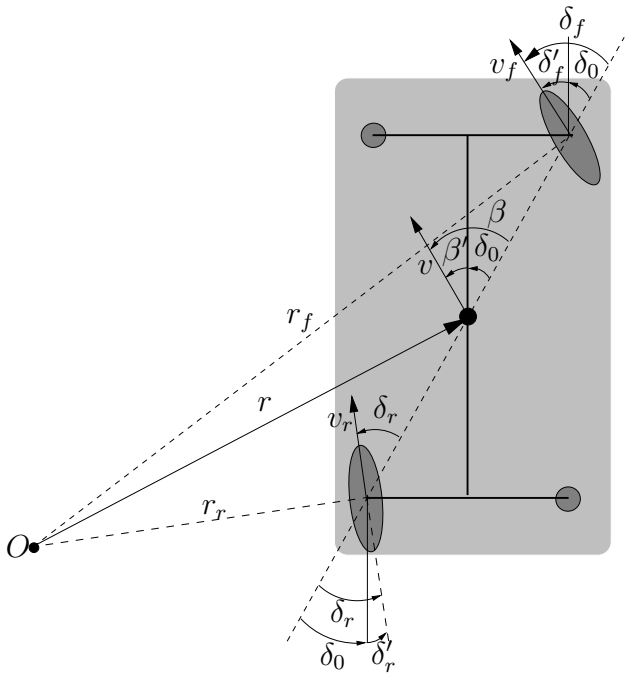


Abbildung 4.4: Anpassung des kinematischen Modells an den LiSA-Roboter. Die Antriebsräder des LiSA-Roboters liegen auf der Diagonalen, so dass in das Zweiradmodell mit zwei Rotations-Freiheitsgraden ein Winkelunterschied von δ_0 integriert werden muss.

δ_0 rotiert ist. Im Folgenden beziehen sich alle mit $'$ versehenen Variablen auf die Hauptachse des Roboters und alle anderen Variablen auf die Diagonale des Roboters.

Zur Anpassung des kinematischen Modells müssen zunächst die gemessenen Radwinkel δ'_r und δ'_f durch Addition von δ_0 auf die Diagonale des Roboters bezogen werden.

$$\delta_f = \delta'_f + \delta_0 \qquad \delta_r = \delta'_r + \delta_0 \tag{4.9}$$

Mit den auf die Diagonale bezogenen Radwinkeln δ_r und δ_f wird die Geschwindigkeitsrichtung β und damit die Orientierungsänderungsrate $\dot{\Psi}$ bestimmt. Über die diskretisierte zeitliche Integration wird die globale Orientierung bestimmt. Diese muss nicht zurücktransformiert werden, da die Integration bei einer Orientierung von Null beginnt und sich somit auf die Hauptachse bezieht und die jeweilige Orientierungsdifferenz unabhängig von der aktuellen Orientierung ist.

Zur Bestimmung der globalen Position (x', y'), muss die mit den auf die Diagonale bezogenen Radwinkeln berechnete Bewegungsrichtung β zurücktransformiert werden. Diese auf die Hauptachse bezogene Bewegungsrichtung $\beta' = \beta - \delta_0$ wird in Gleichung 4.4 eingesetzt, um die globale Positionsänderung des LiSA-Roboters zu bestimmen.

Beim inversen kinematischen Modell wird zunächst die vorgegebene Bewegungsrichtung β' auf die Diagonale bezogen.

$$\beta = \beta' + \delta_0$$

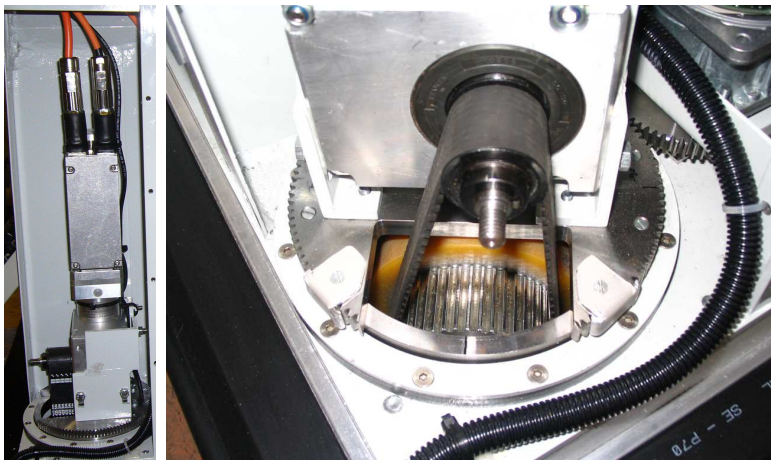


Abbildung 4.5: Aufbau des Radantriebs des Lisa-Roboters.

Mit β und der vorgegebenen Orientierungsänderungsrate $\dot{\Psi}$, die wie oben erläutert nicht transformiert werden muss, lassen sich mit Gleichung 4.7 die Radwinkel δ_r und δ_f bestimmen. Die berechneten Radwinkel werden verwendet, um mit Gleichung 4.8 die Radgeschwindigkeiten v_r und v_f zu bestimmen.

Anschließend werden die berechneten Radwinkel durch Subtraktion von δ_0 auf die Hauptachse des Roboters bezogen:

$$\delta'_f = \delta_f - \delta_0 \quad \text{und} \quad \delta'_r = \delta_r - \delta_0.$$

Eine Transformation der berechneten Radgeschwindigkeiten ist nicht notwendig, da diese nur durch die Stellung der beiden Räder zueinander definiert und unabhängig von der Rotation der Hauptachse sind. Neben den beschriebenen Anpassungen muss beim LiSA-Roboter zusätzlich berücksichtigt werden, dass die Radwinkel nicht frei einstellbar sind. Wie in Abbildung 4.5 zu sehen ist, haben die Antriebszahnräder zur Orientierungsänderung der beiden Räder eine Aussparung, die Platz für die Kette des Antriebsmotors der Räder bieten. Diese Designentscheidung hat den Vorteil, dass keine Schleifkontakte für den Antriebsmoter verwendet werden müssen. Der Nachteil hingegen ist eine Nichtlinearität in der Kinematik des Roboters, da die Räder durch die Aussparung einen maximalen Winkel von $\delta'_{max} = 110°$ bzw. $\delta'_{min} = -110°$ besitzen. Aus Gleichung 4.2 ist ersichtlich, dass dies ebenfalls eine Beschränkung der Bewegungsrichtung β' auf dasselbe Intervall zur Folge hat $[\delta_{min} \leq \beta' \leq \delta_{max}]$. Eine Bewegungsrichtung außerhalb dieses Intervalls lässt sich nur durch eine Invertierung der Radgeschwindigkeiten als auch der Radwinkel erreichen.

Da der LiSA-Roboter in der Lage sein muss, seine Räder auf der Stelle zu drehen, ist aufgrund des hohen Gewichts des Roboters für den Motor zur Orientierungsänderung eine relativ hohe Übersetzung notwendig. Dies hat zur Folge, dass eine Invertierung des Radwinkels etwa zwei Sekunden benötigt. Um innerhalb dieses Zeitintervalls keine ungewollte Positionsänderung zu erzeugen, müssen die Radgeschwindigkeiten während dieser Drehung auf Null zurückgesetzt werden. Daher kann der Übergang von $\beta' \leq \delta_{max} \rightarrow \beta' > \delta_{max}$ bzw. $\beta' \geq \delta_{min} \rightarrow \beta' < \delta_{min}$

nur bei geringen Geschwindigkeiten durchgeführt werden. Dies ist im LiSA-Szenario gegeben, da der Übergang von einer seitlichen in eine nach hinten gerichtete Bewegung nur beim Ausweichen vor einem sehr nahen Hindernis oder beim Ausrichten im Tür- bzw. Zielbereich auftritt (vgl. Kapitel 7.1) und in diesen Situationen mit einer geringen Geschwindigkeit gefahren wird.

4.3 Softwarekomponenten

In diesem Abschnitt werden drei Softwarekomponenten beschrieben, die im Rahmen des LiSA-Projekts verwendet bzw. entwickelt werden. Zunächst wird in 4.3.1 die Software beschrieben, die die Kommunikation zwischen den Modulen zur multimodalen Interaktion mit dem Bediener, sowie den Modulen zur Auftragssteuerung, Navigation und Manipulation realisiert.

Anschließend wird in 4.3.2 mit „Player“ die in dieser Arbeit verwendete Roboterentwicklungsumgebung beschrieben, innerhalb der die Module zur sicheren Navigation des Roboters realisiert sind. Diese beinhalten z.B. Pfadplanung, Hindernisvermeidung und die Kommunikation mit der SPS des Roboters.

Die dritte zur Entwicklung der Navigation eingesetzte Softwarekomponente bildet „USARSim“. Da es sich bei dem LiSA-Roboter um eine komplette Neuentwicklung handelt, war die Hardware erst im letzten Drittel des Projekts verfügbar. Um trotzdem mit der Entwicklung der Navigationsmodule in Player beginnen zu können, wird mit USARSim eine Simulationsumgebung eingesetzt. In Abschnitt 4.3.3 wird beschrieben, wie die Simulationsumgebung aufgebaut ist und wie die Kinematik des Roboters in dieser Simulationsumgebung realisiert wird. Anschließend wird in Abschnitt 4.3.4 beschrieben, wie die Simulationsumgebung USARSim mit Player verbunden wird.

4.3.1 Kommunikationsstruktur der LiSA-Module

Im Rahmen des LiSA-Projekts wird eine eigenständige Kommunikationsstruktur entwickelt. Diese stellt die Verbindung der von den jeweiligen Projektpartnern entwickelten Software dar. So verbindet sie die Software zur multimodalen Anwenderinteraktion mit dem Auftragsscheduling. Ebenfalls werden die Software zur Ansteuerung des Manipulators sowie die Software zur Bildverarbeitung und die im nächsten Abschnitt beschriebene Software zur Navigation miteinander verbunden. Diese Kommunikationssoftware wurde gemeinsam mit den Projektpartnern entwickelt. Der innerhalb dieser Arbeit geleistete Beitrag ist die Konzeption der Software. Ferner wurde das Modul, welches für die Navigation zuständig ist, entwickelt. Die restlichen Komponenten wurden gemeinsam mit den Projektpartnern erarbeitet.

Die Anforderung, bereits bestehende Software sowie im Laufe des Projekts entwickelte Software zu verbinden, macht es notwendig, dass die Kommunikationssoftware mit den Be-

triebssystemen Windows Vista (benötigt von der grafischen Benutzeroberfläche) und einem Unix-basierten Betriebssystem (benötigt von der Navigation) kompatibel ist. An die Software ist des Weiteren die Anforderung gestellt, dass die einzelnen Module prinzipiell auf unterschiedlichen Rechnern laufen können müssen. Diese Anforderung ist notwendig, da die rechenzeitkritischen Systeme wie Navigation, Manipulation und Bildverarbeitung auf unterschiedlichen Rechnern untergebracht sind, um sich gegenseitig nicht zu beeinflussen. Abbildung 4.6 zeigt die Verteilung der verschiedenen Rechner im LiSA-Szenario. Auf dem Roboter befinden sich, wie schon erwähnt, drei Laptops sowie ein WLAN Accesspoint. Über diesen kommuniziert die Software mit den Modulen der Auftragsplanung und Sprachverarbeitung, die auf einem festen PC laufen, der in der Abbildung oben rechts zu sehen ist. Die Sprachverarbeitung kommuniziert mit dem Modul zur grafischen Benutzerinteraktion. Dieses Modul läuft jeweils auf zwei Touchpads, die vom Anwender des Systems frei bewegt werden können. Die zur Sprachverarbeitung notwendigen Headsets werden über Bluetooth angesprochen. Da der LiSA-Roboter autonom ist, könnte er theoretisch während seiner Auftragsausführung den WLAN Bereich verlassen. Dies wird aber aus sicherheitstechnischen Gründen nicht zugelassen, da der Anwender in dem Fall keine Kontrolle mehr über den Roboter hat und Fehlerzustände des Roboters nicht bis zum Anwender gesendet werden können. Der Vorteil, die Sprachverarbeitung auf einen festen PC auszulagern, besteht darin, dass der Anwender gewarnt werden kann, falls der Roboter doch die WLAN Zone verlässt. Ein weiterer Vorteil, die Kommunikationsstruktur so zu realisieren, dass die Module auf beliebigen Rechnern im selben Netzwerk laufen können, liegt darin, dass die parallele Entwicklung der unterschiedlichen Softwarekomponenten der einzelnen Projektpartner wesentlich vereinfacht und somit beschleunigt wird. Diese Vorgehensweise setzt jedoch eine einfache Konfiguration der Verteilung der Module auf die vorhandenen Rechner voraus. Diese Anforderung wird ebenfalls im Integrationsprozess benötigt. Nachdem bestimmt wurde, welche Module wann wieviel Rechenzeit benötigen, können kompatible Module auf einem Rechner zusammengefasst werden. Somit können potentiell Rechner eingespart werden, um die Betriebszeit des LiSA-Roboters zu erhöhen.

Realisiert wird die Software in C/C++ mit Hilfe der Bibliothek „Portable Components in C++“ (POCO). Diese stellt bereits Funktionalitäten zum Parsen von xml-Dokumenten und zur netzwerkbasierten Kommunikation zur Verfügung und ist mit den benötigten Betriebssystemen kompatibel. Da die Module frei auf den Rechnern verteilt werden sollen, wird über xml-Konfigurationsdateien gesteuert, welches Modul auf welchem Rechner gestartet wird. Jedes Modul wird in einem eigenen Thread, unabhängig von den anderen Modulen, die auf dem jeweiligen Rechner laufen, gestartet. Auf jedem Rechner läuft ein sogenannter „Dispatcher“, der eingehende Nachrichten auf die Module verteilt oder ausgehende Nachrichten der Module des eigenen Rechners versendet. Die Struktur der versendeten Nachrichten sind an die Kommunikation der Roboterentwicklungsumgebung „Player“ angelehnt (vgl. 4.3.2). Jede Nachricht hat einen Nachrichtentypen. Dieser spezifiziert die grundsätzliche Eigenschaft der Nachricht, wie z.B. Datennachricht Kommando, Anfrage, Fehler,... Zu jedem Nachrichtentypen existieren verschiedene Subtypen, die die Nachricht näher spezifizieren (z.B. die unterschiedlichen Fehlerzustände). Optional kann jede Nachricht Daten enthalten, die mit dieser verschickt werden können. Diese Daten sowie der Nachrichtentyp, Subtyp, sowie die Id

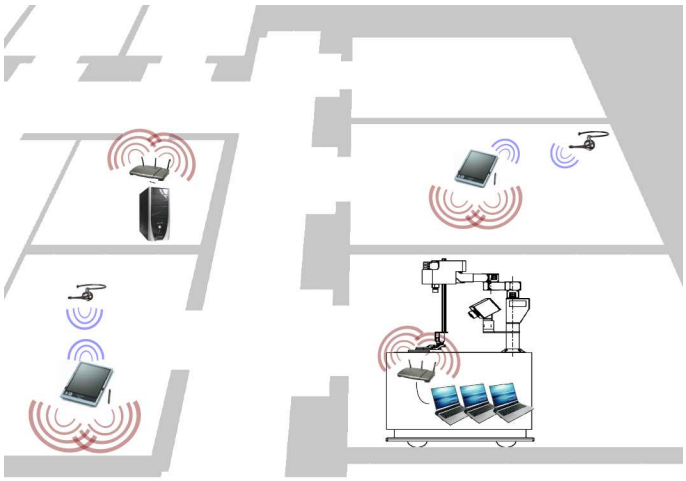


Abbildung 4.6: Schematische PC-Konfiguration im LiSA-Szenario: Auf dem LiSA-Roboter befinden sich drei Laptops zur Navigation, Manipulation und Bildverarbeitung des Roboters. Diese kommunizieren über WLAN (rot) mit einem PC (oben links), auf dem die Sprachverarbeitung läuft. Dieser kommuniziert wiederum über WLAN mit zwei Touchpads, auf denen das Grafische User Interface läuft. Die zur Sprachverarbeitung benötigten Headsets werden über Bluetooth (blau) angesprochen.

des Sender- und Empfängermoduls werden im Dispatcher serialisiert und über das Netzwerk zum Zielcomputer geschickt. Dort werden sie vom Dispatcher des Computers deserialisiert und die erzeugte Nachricht wird an das entsprechende Modul verteilt.

Folgende Module kapseln die Funktionalitäten, die im LiSA-Szenario benötigt werden.

Navigation: Dieses Modul kapselt die gesamte Funktionaliät, die zur Navigation des Roboters notwendig ist. Das Modul ist als Player-Client realisiert (vgl. nächsten Abschnitt). Es nimmt Fahrkommandos entgegen und sendet Navigationsfehler des Roboters (Kollision, kein Pfad zum Ziel, unbekannte Pose) an die anderen Module der Kommunikationsstruktur.

Manipulation: Analog zum Navigations-Modul kapselt das Manipulations-Modul die gesamte Funktionalität zur Steuerung des Manipulators. Es werden verschiedene Greifaufträge angenommen und Fehlerzustände gemeldet. Ferner wird die gesamte Bildverarbeitung zur Objekterkennung und sicheren Manipulation im gemeinsamen Mensch-Roboter-Arbeitsumfeld in diesem Modul gekapselt.

LiSA: Das LiSA-Modul kapselt den Zustand des LiSA-Roboters. Einem LiSA-Modul sind genau ein Navigations- und Manipulations-Modul zugeordnet. Das LiSA-Modul reicht Fahraufträge an das Navigations-Modul bzw. Manipulationsaufträge an das Manipulations-Modul weiter. Des Weiteren überwacht es die strikte Trennung, die es zwischen der Navigation und der Manipulation des Roboters gibt.

LiSA Task Manager: Der LiSA Task Manager (LTM) übernimmt das Auftragsschedu-

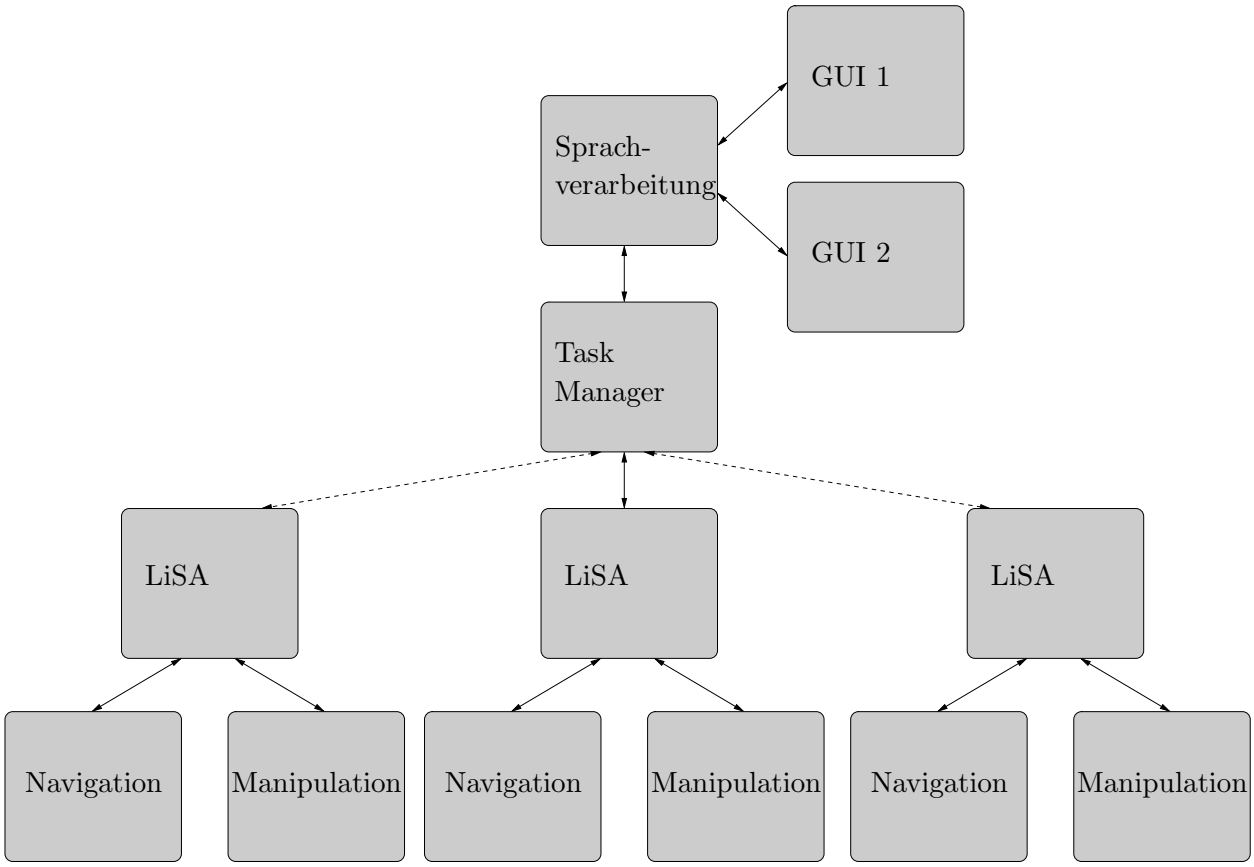


Abbildung 4.7: Kommunikationsstruktur der verschiedenen Module im LiSA-Szenario: Zwischen den einzelnen Modulen besteht eine klare Beziehung, wer mit welchem anderen Modul kommunizieren darf (Pfeile). Es existieren einige Ausnahmen; so werden Bilddaten direkt von der Manipulation zur GUI gesendet, um das Kommunikationsnetz nicht unnötig mit großen Datenmengen zu belasten. Die gestrichelten Pfeile verdeutlichen, dass der Task Manager potentiell Aufgaben an verschiedene LiSA-Roboter verteilt.

ling des Systems. Er nimmt komplexe Aufträge entgegen und unterteilt diese in atomare Aufgaben. Diese Aufgaben, die nacheinander ausgeführt den komplexen Auftrag erfüllen, werden anschließend auf theoretisch beliebig viele LiSA-Roboter verteilt und an die entsprechenden LiSA-Module gesendet.

Sprachverarbeitung: Die gesamte Sprachverarbeitung wird in diesem Modul gekapselt. Es verarbeitet die natürlichsprachliche Eingabe der Anwender und extrahiert die enthaltene Information. Parallel dazu werden potentielle Eingaben der GUI verarbeitet. Anhand dieser Information wird ein komplexer Auftrag erstellt und an das LTM-Modul gesendet.

GUI: Das GUI-Modul kapselt sämtliche Algorithmen zur graphischen Darstellung des aktuellen Status des Systems sowie zur grafischen Eingabe von Aufträgen für den LiSA-Roboter durch den Anwender.

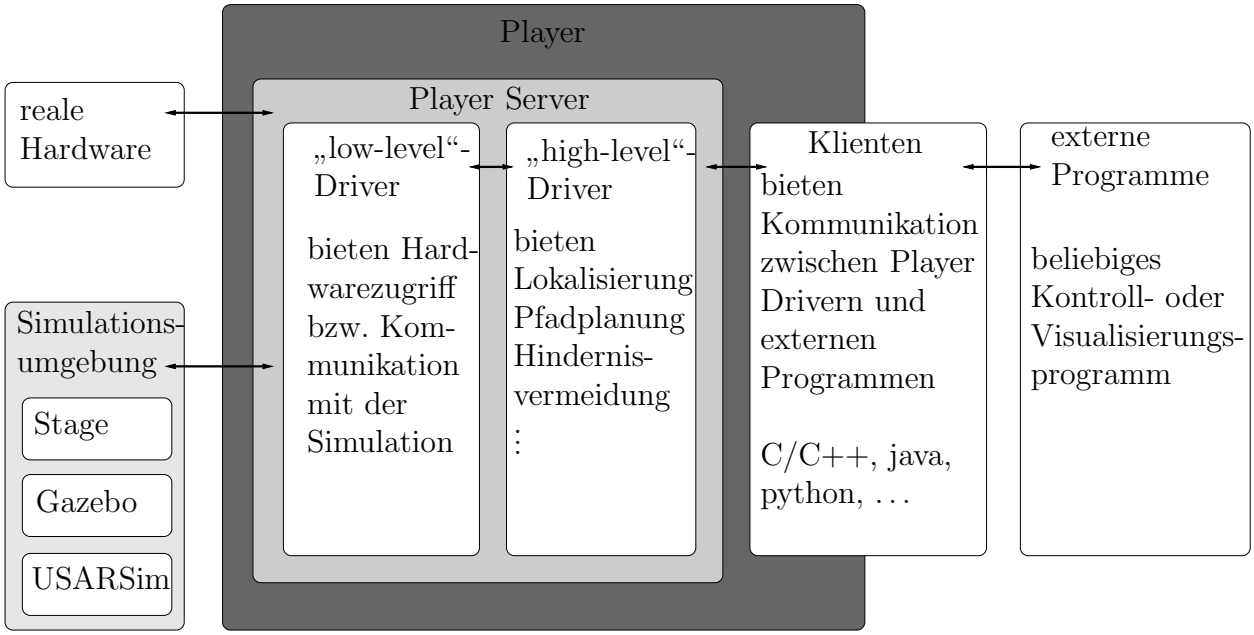


Abbildung 4.8: Struktur der Roboterentwicklungsumgebung Player: Player stellt die Verbindung zwischen einem Kontrollprogramm und der Roboterhardware bzw. einer Simulationsumgebung her. Dabei ermöglicht der standardisierte Zugriff hardwarenaher Playertreiber die Entwicklung komplexer Treiber wie Lokalisierung, Pfadplanung, ... Player bietet ebenfalls eine Client-Bibliothek, die den Zugriff auf alle Treiber von externen Kontrollprogrammen erlaubt.

In Abbildung 4.7 ist dargestellt, welche dieser Module miteinander kommunizieren dürfen. Die Kommunikationswege werden beschränkt, damit sich das Verhalten des Systems leichter analysieren lässt und die Fehlersuche beschleunigt wird.

4.3.2 Player

Player ist eine Entwicklungsumgebung für Roboterkontrollalgorithmen. Als Entwicklungsumgebung für autonome mobile Roboter hat Player die Aufgabe, Software zur Verfügung zu stellen, die es erleichtert, Aufgaben und Problemstellungen im Bereich der Robotik zu lösen. Diese Entwicklungsumgebungen sind notwendig, da das Forschungsgebiet der autonomen mobilen Robotik viele Teilbereiche umfasst und Expertenwissen in diesen Bereichen erfordert, um das Gesamtrobotersystem zu beherrschen. So reicht das Gebiet von der hardwarenahen Reglerprogrammierung bis zum Multiagentensystem zur Roboterschwarmkoordination. Die Aufgabe der Entwicklungsumgebungen ist es, das Expertenwissen in den einzelnen Teilbereichen aufzusammeln und anderen Bereichen unkomplizierten Zugang und Nutzen dieses Wissens zu ermöglichen. Eine Entwicklungsumgebung für autonome mobile Roboter sollte aus diesem Grund sehr flexibel sein und keine Vorgaben an die verwendete Hardware oder das Softwaredesign stellen. Des Weiteren muss sie klare Schnittstellen zu den Teilbereichen definieren.

Es existieren eine Reihe von verschiedenen Roboterentwicklungsumgebungen. Neben kommerziellen, proprietären Entwicklungsumgebungen wie dem „Microsoft Robotics Studio“ gibt es viele auf freier Software basierende Systeme wie CARMEN [52], ORCA [10] oder CLARAty [90]. In dieser Arbeit wird die Entwicklungsumgebung Player verwendet, da sie einen Großteil der verwendeten Hardware des LiSA-Roboters unterstützt und eine gute Anbindung an verschiedene Simulationsumgebungen bietet. [39] gibt einen detaillierten Vergleich von Entwicklungsumgebungen in der Robotik, der ebenfalls zu dem Schluss kommt, dass Player die zur Zeit beste Entwicklungsumgebung ist. Daher ist es nicht verwunderlich, dass Player als freie Software weltweit von mehreren hundert Forschungsgruppen und Unternehmen genutzt und weiterentwickelt wird. Laut [88] stellt Player den de facto Standard der Roboter Middleware dar. Die Entwicklung von Player wurde 1999 von Robotics Research Lab der University of South Carolina (USC) begonnen. Player läuft auf den Betriebssystemen Linux, Solaris, BSD and MAC OSX(Darwin).

Player ist so konzipiert, dass es eine flexible Entwicklung der Kontrollstrukturen erlaubt. So gibt Player für diese keine feste Struktur vor und ist so programmiert, dass es prinzipiell unabhängig von der verwendeten Programmiersprache und vom Betriebssystem ist.

Um dies zu realisieren, ist Player als Socket-basiertes Server/Client Modell implementiert. Die Grundidee ist, dass ein Player Server auf dem Roboter läuft. Dieser Server bietet einen einfachen und klar strukturierten Zugriff auf die Sensoren und Aktoren des Roboters. Kommuniziert wird nachrichtenbasiert über ein TCP/IP Netzwerk [13]. Dies erlaubt es, die Kontrollalgorithmen, die von dem Player Server z.B. Sensordaten erhalten, in einer beliebigen Programmiersprache zu schreiben, die TCP Sockets unterstützen.

Player ist aus den drei folgenden Grundkonzepten aufgebaut:

Interface: Ein Interface spezifiziert, wie auf eine Klasse von Sensoren, Aktoren oder Algorithmen zugegriffen wird. Es werden Anfragen definiert, die an eine Sensor-, Aktor- oder Algorithmenklasse gestellt werden können und welche Daten die Antwort auf eine spezielle Anfrage enthalten. Des Weiteren wird festgelegt, welche Daten ohne Anfrage gesendet werden und wie diese zu interpretieren sind. Die dritte Gruppe von Nachrichten, die in einem Interface spezifiziert wird, sind Kommandos, die an eine Klasse gesendet werden können, welche das Interface implementiert. Durch ein Interface ist sichergestellt, dass die gesendeten Daten verschiedener Klassen, die zur selben Sensor-, Aktor- oder Algorithmenklasse gehören, immer gleich interpretiert werden können. Somit ist es einer anderen Klasse, die mit dieser Klasse kommuniziert, egal, welcher spezielle Sensor, Aktor oder Algorithmus sich hinter der Klasse verbirgt. Dies führt dazu, dass sich Hardware- und Softwarekomponenten einfach austauschen lassen.

Driver: Ein Driver implementiert das oben beschriebene Interface für einen Sensor, Aktor oder Algorithmus. Seine Aufgabe ist es, die spezifischen Eigenschaften z.B. eines Sensors nach außen nicht sichtbar zu machen. Zum Beispiel ist der Treiber für den SICK S300-Laserscanner in der Lage, mit diesem zu kommunizieren, dessen spezifisches Datenformat auszulesen und die Daten konform zum Laser-Interface bereit zu stellen.

Device: Ein Device repräsentiert eine spezielle Instanz eines Drivers. Ein Device besitzt eine IP-Adresse, unter der die spezielle Klasse erreichbar ist. Die gesamte Kommunikation läuft über die Devices. Es ist möglich, mehrere Devices desselben Drivers zu erzeugen. Am Beispiel des LiSA-Roboters wären dies vier Devices des Hokuyo-Laserscanner Drivers. Jedes dieser Devices würde mit einem speziellen Laserscanner kommunizieren.

Durch diese Strukturierung bietet Player einen leichten Einstieg in die Robotersteuerung. In Player existieren bereits viele Driver für Hardware, die in der Robotik zum Einsatz kommen.

Durch den standardisierten Zugriff auf die Hardware, können neben den „low-level"-Treibern, die den Zugriff auf die Hardware realisieren, hardwareunabhängige „high-level"-Driver entwickelt werden. Diese „high-level"-Driver verwenden die Daten der „low-level"-Driver und implementieren basierend auf diesen Daten Algorithmen, die im Rahmen der Robotik auftreten (vgl. Abbildung 4.8). So bietet Player hardwareunabhängige Verfahren zur Lokalisierung, Hindernisvermeidung oder Pfadplannung an. Um eine Standardisierung dieser Algorithmen zu erreichen, werden für die jeweiligen Algorithmenklassen ebenfalls Interfaces definiert. Dies ermöglicht, verschiedene Verfahren der jeweiligen Algorithmenklasse via Konfigurationsdatei auszutauschen. Es ist sogar möglich, unterschiedliche Algorithmen zur Laufzeit auszutauschen. So verwendet der LiSA-Roboter in unterschiedlichen Situationen unterschiedliche Algorithmen zur Hindernisvermeidung, ohne dass es den Drivern, die auf die Hindernisvermeidung aufbauen, bewusst ist (vgl. Kapitel 7.1).

Eine weitere Konsequenz des standardisierten Zugriffs auf Sensoren und Aktoren ist, dass Kontrollalgorithmen, die mit Hilfe einer Simulationsumgebung entwickelt werden, ohne Veränderung auf die reale Hardware anwendbar sind. Player liefert hierzu die 2D-Simulationsumgebung „Stage" und die 3D-Simulationsumgebung „Gazebo". Diese Simulationsumgebungen bieten Driver an, die mit den Sensoren und Aktoren des simulierten Roboters kommunizieren und deren Daten konform zum entsprechenden Interface anbieten. Somit ist es möglich „high-level"-Driver zu entwickeln, die auf diesen Daten basieren. Diese „high-level"-Driver werden auf die reale Hardware überführt, indem die Driver der simulierten Sensoren und Aktoren durch Driver realer Sensoren und Aktoren ersetzt werden, die jeweils dasselbe Interface implementieren. In dieser Arbeit wird die in Kapitel 4.3.3 beschriebene Simulationsumgebung „USARSim" eingesetzt. Die Driver zur Kommunikation mit den simulierten Sensoren und Aktoren in der Simulationsumgebung USARSim werden in Kapitel 4.3.4 beschrieben.

Ein weiterer Grund für die weite Verbreitung der Player Software ist eine vorhandene Implementierung von TCP Sockets in C/C++ und Python zur Kommunikation mit den Drivern des Player Servers. Es existiert für jedes Interface und somit für jeden Driver ein sogenannter „Klient", der mit dem jeweiligen Driver kommuniziert. Dieser Klient bietet die Schnittstelle zwischen Player und einer beliebigen Kontrollstruktur. Neben diesen integrierten Klienten existieren weitere, die von Anwendern geschrieben wurden, um mit dem Player Server in ihrer Kontrollsoftware kommunizieren zu können. Im Moment existieren Klienten in den Programmiersprachen java, Octave, Scheme/Guile, Lisp, Ada und Matlab. Somit ist die Verwendung der Player Module nicht auf das Framework selbst begrenzt, sondern offen für andere Anwendungen und Frameworks [15, 26].

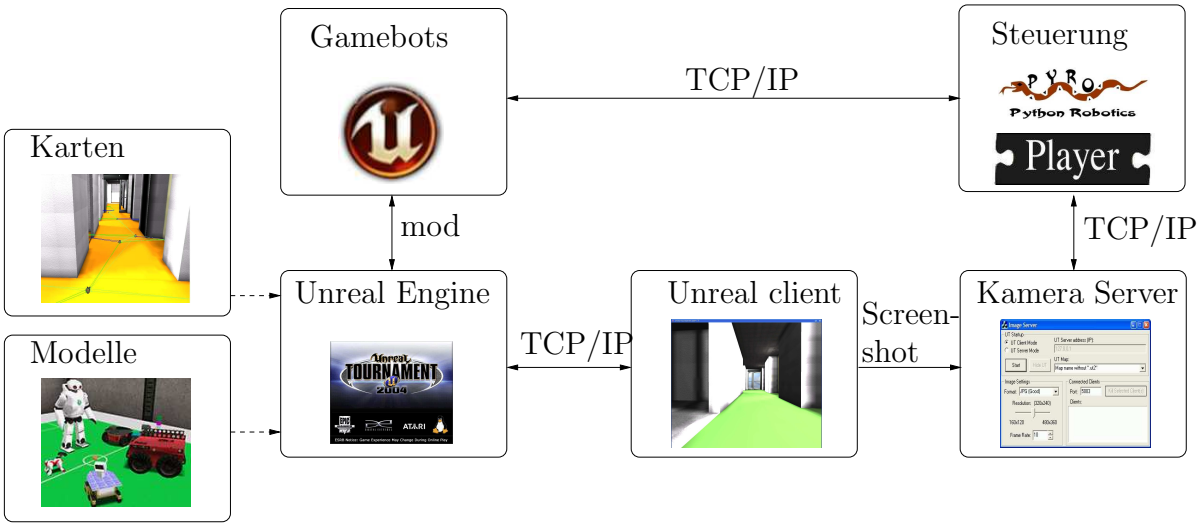


Abbildung 4.9: Schematischer Aufbau der USARSim Kommunikationsstruktur nach [1].

Es bestehen somit zwei Möglichkeiten, Kontrollalgorithmen zu entwickeln. Die erste ist, die oben beschriebenen Klienten-Bibliotheken zu verwenden, um sich zu den verschiedenen Sensoren, Aktoren oder Algorithmen zu verbinden und deren Daten zu nutzen, um die Kontrollstrukturen zu entwickeln (vgl. Abbildung 4.8). Diese Methode wird am häufigsten verwendet, da sie den Vorteil hat, dass der Entwickler frei in der Wahl der Programmiersprache und der Strukturierung seiner Algorithmen ist. Es existieren sogar weitere Frameworks, die diese Schnittstelle verwenden und Player als reine Hardwareabstraktionsschicht nutzen. Der Nachteil dieser Methode ist der, dass die geschriebenen Kontrollalgorithmen im Allgemeinen nicht wieder verwendet werden können. Hier setzt die zweite Alternative an, die vorsieht, direkt im Player Server zu programmieren. Diese Alternative impliziert die Beschränkung, dass die Kontrollalgorithmen in verschiedenen Drivern strukturiert sein müssen, die an spezifische Interfaces gebunden sind. Der entscheidende Vorteil ist jedoch, dass alle Player-Anwender diese Kontrollstrukturen wieder verwenden können, unabhängig davon, ob sie im Player-Server oder klientenbasiert programmieren. Ein weiterer Vorteil ist der, dass Daten, die zwischen Drivern des Player Servers ausgetauscht werden nicht über eine TCP/IP Verbindung gesendet werden müssen. Dies ist im LiSA-System mit drei kommunizierenden Rechnern, einer SPS und sechs angesprochenen Laserscannern ein entscheidender Faktor. In dieser Arbeit wird wegen der genannten Gründe die zweite Variante verwendet.

4.3.3 Simulationsumgebung USARSim

Da der reale LiSA-Roboter erst im letzten Drittel des Projektes zur Verfügung stand, ist eine Simulationsumgebung notwendig. Diese ermöglicht zum einen eine Bestimmung der optimalen Sensorkonfiguration, um die Umgebung des Roboters dreidimensional wahrzunehmen. Da die Versuche mit unterschiedlichen Sensoren zeitlich vor dem Aufbau des Roboters durchführbar sind, konnten die Ergebnisse der Robotersimulation bereits in der Konzeptionsphase des

Roboters genutzt werden. Dies gilt nicht nur für die Sensorkonfiguration, sondern genauso für das kinematische Modell des LiSA-Roboters. Neben der Bestimmung optimaler Hardwarekomponenten bietet eine Simulationsumgebung den weiteren Vorteil, dass die Entwicklung der Kontrollalgorithmen zur kollisionsfreien Navigation des LiSA-Roboters bereits begonnen werden kann, bevor die reale Plattform vorhanden ist.

In dieser Arbeit wird die "Unified System for Automation and Robot Simulation" (USARSim) Simulationsumgebung eingesetzt. Diese baut auf die kommerzielle Physik-Engine des Computerspiels "Unreal Tournament 2004" auf. Den Kern der Simulation bieldet die „Unreal Engine". Diese wird vom Spielentwickler Epic Games als Middleware für zahlreiche Spiele verwendet. Die Unreal Engine bietet neben der exzellenten 3D-Grafik eine Verbindung zur Physik-Engine, die ein realistisches Verhalten der Akteure im Spiel gewährleistet. "Unreal Tournament 2004" verwendet die Unreal Engine in der Version 2.5. In dieser wird die „Karma"-Physik-Engine eingesetzt.

Die Karma-Physik-Engine aus dem Jahr 2002 arbeitet nach dem Starrkörper-Prinzip. Sie geht davon aus, dass die Akteure aus starren Komponenten aufgebaut sind, die durch verschiedene Gelenke verbunden sind. Durch unterschiedliche Gelenke können verschiedene Nebenbedingungen für die Bewegung zweier starrer Körper zueinander bzgl. Geschwindigkeit, Kraft oder Position realisiert werden. Neben der Verbindung zur Physik Engine bietet die Unreal Engine eine objektorientierte Scriptsprache (Unreal Script), um Spieleentwicklern eine Möglichkeit zu bieten, die Spiellogik der Spiele, die auf der Unreal Engine aufbauen, zu verändern. Diese Modifikatoren werden mods (Subkulturen) genannt und tragen zur Beliebtheit der Unreal Engine basierten Spiele bei, da die Anwender eigene Spieleideen realisieren können. So wird die Unreal Engine z.B. in der Künstlichen Intelligenz verwendet, um das autonome Verhalten simulierter Agenten zu programmieren [36].

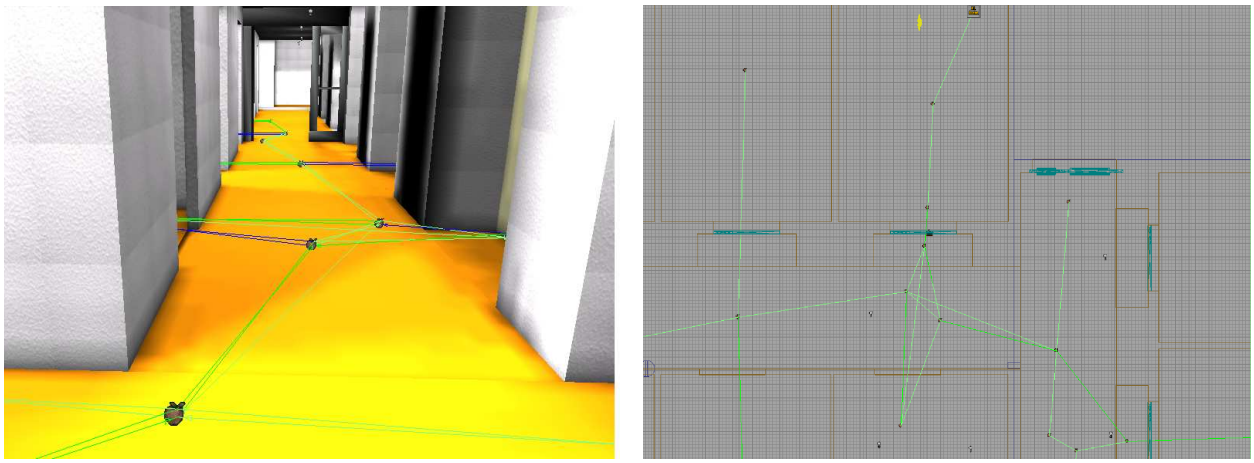
Ein weiterer mod wird in der Simulationsumgebung USARSim verwendet, um mit dem Unreal Server kommunizieren zu können. Da das Kommunikationsprotokoll der Unreal Engine proprietär ist, wurde der „Gamebots"-mod an der University of South California entwickelt [34]. Dieser mod öffnet einen TCP/IP Socket in der Unreal Engine und erlaubt es, Daten über diesen Socket zwischen externen Programmen und der Unreal Engine auszutauschen. USARSim nutzt diese Verbindung, um Kontrollkommandos und Sensordaten zwischen externen Applikationen und dem simulierten Roboter zu verschicken (vgl. Abbildung 4.9).

Der Vorteil einer kommerziellen Spiele Engine wie der Unreal Engine gegenüber der einer Open Source Variante wie z.B. Gazebo liegt darin, dass die Grafik und Physik-Simulation parallel weiterentwickelt wird. So wird USARSim im Moment auf die Unreal Engine 3.0 portiert, die von dem im November 2007 erschienenen Spiel Unreal Tournament 3 verwendet wird. Die Unreal Engine 3.0 unterstützt mit der Ageia PhysX Physik-Engine eine der zur Zeit leistungsstärksten Engines [16]. Ferner ist USARSim auf allen Betriebssystemen verwendbar, die auch die Unreal Engine unterstützen, wie Windows, Linux oder Mac OS. Der Nachteil, den USARSim als proprietäre Software besitzt, ist, dass es sich auf unterster Ebene nicht nachvollziehen lässt, warum die Roboter sich so verhalten, wie es in der Simulation zu sehen ist. Die Simulationsumgebung USARSim kann in vier Gebiete aufgeteilt werden [12].

Simulation der Umgebung: Die virtuelle Umgebung des simulierten Roboters lässt sich mit Hilfe des Level Editors des Spiels erstellen. Hier besteht die Möglichkeit "CAD"-Dateien oder andere 3D-Formate aus anderen 3D-Modellierungsprogrammen wie Maya, 3DS-Max oder Blender zu importieren. Diese Objekte können entweder statisch sein oder passiv dynamisch. Das bedeutet, dass sie sich nicht aktiv bewegen, sich aber durch andere Akteure der Simulation bewegen lassen (z.B. ein Ball, der von einem Akteur angestoßen wird). Neben den statischen und passiv dynamischen Objekten ist es möglich, dynamische Objekte zu erzeugen, die sich aktiv bewegen. Diesen sogenannten „Bots" kann ein Navigationsnetz vorgegeben werden, das sie verwenden können. Des Weiteren ist es möglich, über ein sogenanntes „AI"-Script, das einem Bot zugewiesen wird, sehr komplexe Verhaltensmuster des jeweiligen Bots zu programmieren. Ferner können verschiedene Lichtverhältnisse und spezielle Materialien wie Glas oder Spiegel simuliert werden.

Simulation des Roboters: Es existieren verschiedene Robotertypen, die mit Hilfe von USARSim simuliert werden können: radgetriebene Roboter, Laufroboter, sogar fliegende und Unterwasserroboter können simuliert werden. Die radgetriebenen Roboter werden nach ihrem kinematischen Model in Ackermann oder differential angetriebene Fahrzeuge unterteilt. Um einen Roboter zu simulieren, werden mit Hilfe eines 3D-Modellierungsprogrammes alle Komponenten des Roboters erstellt, die sich gegeneinander bewegen können. Ferner wird die Kollisionshülle für jede Komponente modelliert. Des Weiteren wird über eine Konfigurationsdatei definiert, um welche Klasse von Roboter es sich handelt, welche Masse er besitzt, wie stark die Motoren sind usw. Ferner wird festgelegt, wo sich die einzelnen Komponenten bzgl. der Roboterbasis befinden und welche Nebenbedingungen die Bewegungen der Komponenten zueinander besitzen.

Simulation der Sensorik und Aktorik: Neben dem Roboter selbst wird die Sensorik und Aktorik des Roboters simuliert. Hier gibt es propriozeptive Sensoren wie Batteriestatus, Odometrie oder Inertial Measurement Unit. Diese können direkt aus dem Status des simulierten Roboters bzw. durch den Vergleich mit dem vorherigen Status in der Unreal Engine berechnet werden. Neben diesen Sensoren, die unabhängig von der Umwelt des Roboters sind, existieren verschiedene exterozeptive Sensoren, die die Umwelt erfassen. Hier existieren z.B. Sensoren zur globalen Lokalisierung des Roboters wie GPS, oder Sensoren, die Entfernungsdaten der Umgebung liefern wie Laserscanner, Sonar, ... Die zu diesen Sensoren gehörenden Sensordaten werden durch Unreal Script Klassen erstellt, die von der aktuellen Roboterpose Raytracing Verfahren verwenden. Darüber hinaus lassen sich Kameras und Berührungssensoren simulieren. Die Simulation von Kameradaten nimmt in USARSim eine gesonderte Rolle ein, da diese nicht explizit berechnet werden und über die Gamebotsschnittstelle versendet werden (vgl. Abbildung 4.9). Stattdessen wird die in Unreal Tournament integrierte Visualisierung verwendet, da diese das Rendern der simulierten Umgebung aus verschiedenen Posen bereits ermöglicht. Die zu rendernden Posen werden auf die aktuellen Posen der simulierten Kameradaten gesetzt. Nun ist es möglich, die Kameradaten durch iterative Screenshots des Unreal Tournament Fensters zu erzeugen.



(a) **(b)**

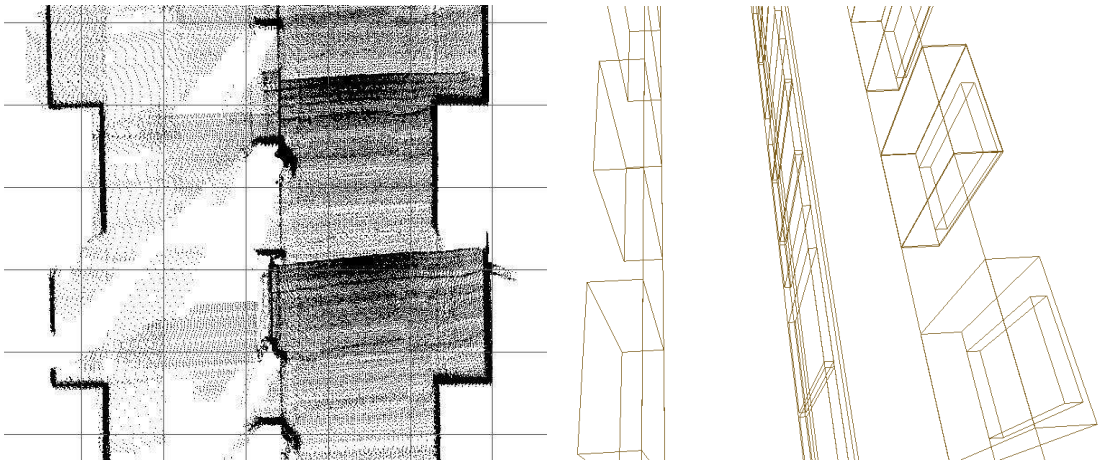
Abbildung 4.10: Navigationsnetzwerk in der simulierten Arbeitsumgebung des LiSA-Roboters: (a) Navigationsnetzwerk zur Simulation von Dynamik im Arbeitsumfeld des LiSA-Roboters. (b) Aufsicht auf einen Ausschnitt der simulierten Arbeitsumgebung und des Navigationsnetzwerkes (hellgrüne Linien).

Verschiedene Eigenschaften der Sensoren sowie deren Pose bzgl. der Roboterbasis werden über die Konfigurationsdatei gesetzt. Es lassen sich z.B. der prozentuale Fehler der Sensordaten, sowie der Sichtbereich und die Auflösung der Sensoren einstellen. Neben den Sensoren werden in USARSim verschiedene Aktoren simuliert, mit denen sich die simulierte Umwelt des Roboters verändern lässt. Hier existieren z.B. Scheinwerfer oder Aktoren, die auf Befehl RFID Marker in der Umgebung befestigen.

Simulation der Kommunikation: In USARSim ist es ferner möglich, die Kommunikation zwischen der Kontrollstation und dem Roboter bzw. die Kommunikation zwischen verschiedenen Robotern zu simulieren. Hierzu wird eine Kommunikationsstation in die simulierte Umgebung des Roboters eingefügt. Anhand der Entfernung des simulierten Roboters zu dieser Station wird entschieden, ob die Nachrichten übertragen werden oder ob die Verbindung unterbrochen wird. Aktuell wird von den USARSim Entwicklern dieses Modell verfeinert, indem durch Ray-Tracing Verfahren die Anzahl der Wände und Möbel bestimmt wird, die zwischen der Kommunikationsstation und einem Roboter bzw. zwischen zwei Robotern liegen und somit die Kommunikation beeinflussen.

Erstellung der simulierten Arbeitsumgebung des LiSA-Roboters

Die simulierte Arbeitsumgebung des LiSA-Roboters wird mit dem Unreal Editor erstellt. Es handelt sich prinzipiell um eine Unreal Tournament Karte, der spezielle Eigenschaften zugeordnet sind. Die Geometrie der Arbeitsumgebung wird mit einem autonomen mobilen Roboter dreidimensional vermessen (vgl. Kapitel 5) und mit einem Marching Cubes-

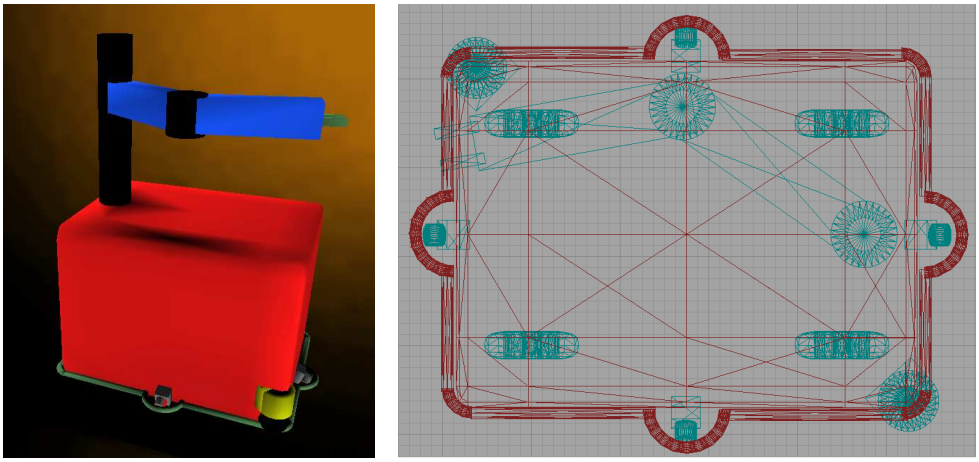


(a) **(b)**

Abbildung 4.11: Simulation der Reflexion von Laserstrahlen an Fensterscheiben in der Arbeitsumgebung des LiSA-Roboters: (a) Aufsicht auf eine dreidimensionale Punktwolke der realen Arbeitsumgebung des LiSA-Roboters, die eine Spiegelung der Flurgeometrie in den Fensterscheiben des Flures zeigt. Die Wandkontur links ist die Spiegelung der Wand rechts in den Scheiben in der Mitte. (b) Simulationsumgebung, die die gespiegelte Flurgeometrie berücksichtigt.

basierten Verfahren aus der erhaltenen Punktwolke extrahiert. Das erhaltene Polygonmodell der Umgebung wird in den Unreal Editor eingefügt und manuell in eine 3D-Karte umgesetzt. Unterstützt wird dies durch eine Grundrisszeichnung der Arbeitsumgebung. Neben der Gebäudegeometrie werden Türen sowie das vorhandene Mobiliar ebenfalls simuliert. Bei der Vermessung des Gebäudes hat sich herausgestellt, dass die Fensterfront des Flures, auf dem sich der LiSA-Roboter bewegen soll, zu einer Spiegelung der Flurgeometrie führt und somit gesondert betrachtet werden muss (vgl. Abbildung 4.11a). Trifft ein Laserstrahl auf diese Glasflächen, wird er von dieser auf die gegenüberliegende Wand reflektiert, anstatt die Glasfläche zu durchdringen. Obwohl die Unreal Engine prinzipiell die Simulation spiegelnder Oberflächen unterstützt, gilt dies nur für optische Effekte. Da der Laserscanner in USARSim über ein Ray-Tracing Verfahren realisiert wird, muss die Gebäudegeometrie an der Glasfläche gespiegelt und ebenfalls in die Karte eingetragen werden. Damit die gespiegelte Gebäudegeometrie nur die Laserscanner beeinflusst, wird sie als unsichtbar deklariert, um nicht in den Kameradaten sichtbar zu sein. Die gespiegelte Wand (Abbildung 4.11b) wird somit für die Laserscanner als tatsächlich vorhandene Wand angenommen, so dass die Reflexion für den Roboter den Normalfall darstellt.

Neben der statischen Umgebung wird die Dynamik berücksichtigt, wie sie im realen Arbeitsumfeld des LiSA-Roboters ebenfalls vorhanden ist. Hierzu werden verschiedene Bots in die Karte eingefügt. Wie in Abbildung 4.10 zu sehen ist, wird für diese Bots ein Navigationsnetzwerk erstellt, das Pfade festlegt, die die Bots im Arbeitsumfeld des LiSA-Roboters verwenden können. Über ein AI-Script wird festgelegt, welcher Bot welche Pfade abläuft. Das Script steuert ebenfalls, dass die einzelnen Bots nicht auf den LiSA-Roboter reagieren, sondern auf ihren vorgegebenen Pfaden verbleiben. Somit ist es die Aufgabe des LiSA-Roboters, den



(a) (b)

Abbildung 4.12: Der simulierte LiSA-Roboter: (a) in einer gerenderten 3D-Ansicht; (b) Aufsicht auf den LiSA-Roboter.

Bots auszuweichen. Ferner werden Unreal Scripte verwendet, um Türen, die vom Navigationsnetzwerk der Bots gekreuzt werden, automatisch zu öffnen und zu schließen, falls ein Bot sich der jeweiligen Tür nähert bzw. sich von dieser entfernt.

Erstellung des simulierten LiSA-Roboters

Der simulierte LiSA-Roboter besteht aus elf unterschiedlichen Komponenten. Diese sind die Roboterbasis inklusive des Manipulators, die vier Räder und die sechs Laserscanner. Die Roboterbasis sowie die Modelle der SICK S300- und Hokuyo-Laserscanner und der Räder werden mit Hilfe des 3D-Modellierungsprogramms Maya erstellt und in den Unreal Editor portiert. Dort werden Kollisionsmodelle für alle Modelle bestimmt und dem Modell hinzugefügt. Anschließend werden in der USARSim Konfigurationsdatei die Posen der Laserscanner und der Räder eingetragen. Da das kinematische Modell des LiSA-Roboters keiner der beiden radgetriebenen Roboterklassen von USARSim entspricht, ist es notwendig, eine neue omnidirektionale Roboterklasse hinzuzufügen. Diese besitzt beliebig viele angetriebene Räder, deren Orientierungen und Geschwindigkeiten unabhängig voneinander einstellbar sind. Zusätzlich können an diese Roboterklasse beliebig viele freilaufende, passive Räder hinzugefügt werden. Welche Räder passiv und welche angetrieben sind, wird ebenfalls über die Konfigurationsdatei gesteuert. Neben den Laserscannern besitzt der simulierte LiSA-Roboter als weitere Sensoren Encoder an den beiden angetriebenen Rädern. Ferner besitzt er einen „Ground Truth“-Sensor, der die aktuelle Roboterpose fehlerfrei angibt. Dieser Sensor wird verwendet, um die Ergebnisse der Lokalisierung (vgl. Kapitel 6.1) mit der tatsächlichen Pose des simulierten LiSA-Roboters vergleichen zu können. Der Manipulator ist beim simulierten LiSA-Roboter starr mit der Roboterbasis verbunden. Da im LiSA-Projekt eine strikte zeitliche Trennung zwischen Manipulation und Navigation des LiSA-Roboters besteht, ist eine Simulation des Manipulators aus Sicht der Navigation nicht notwendig.

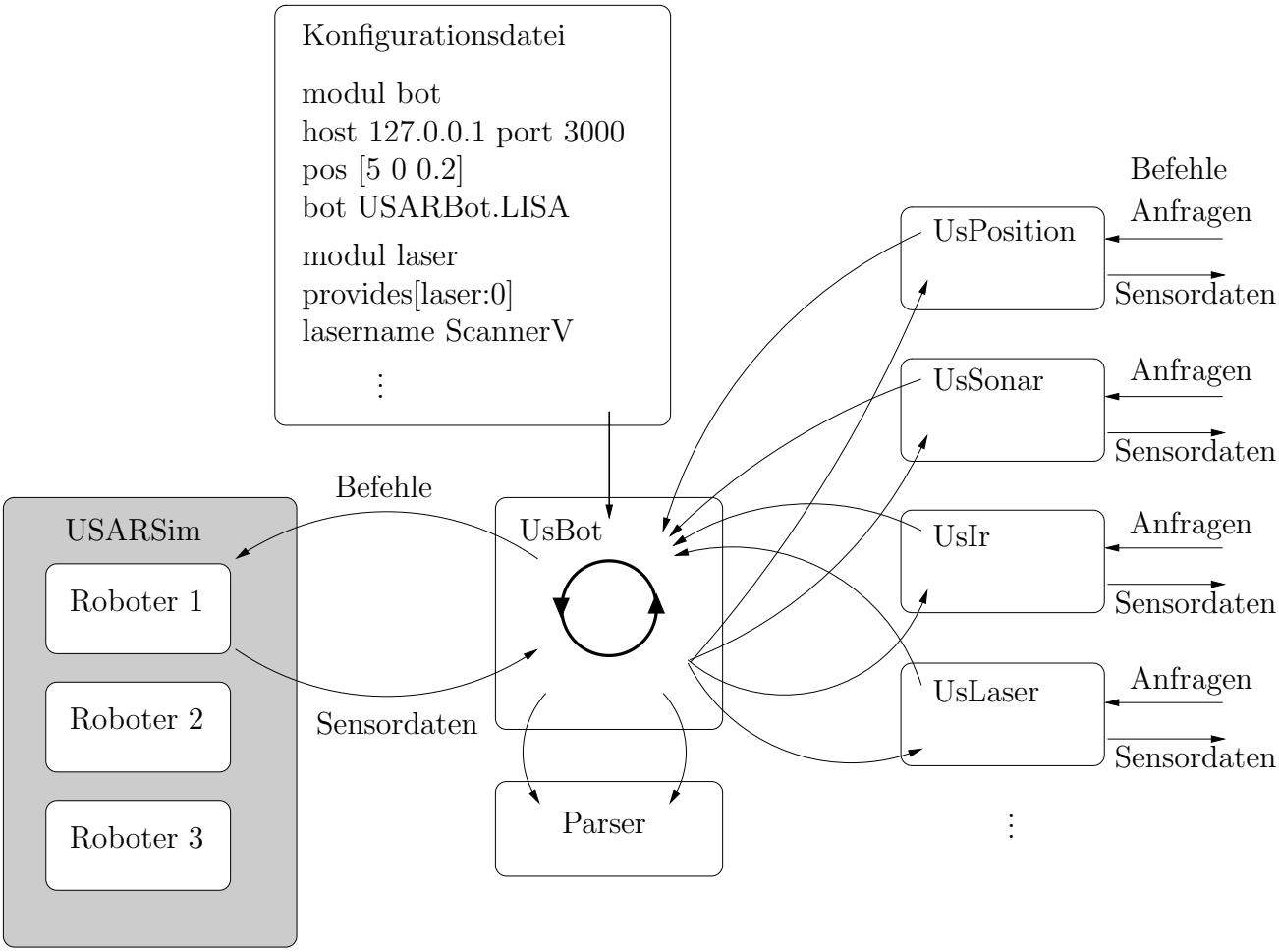


Abbildung 4.13: Schematische Darstellung des Player-USARSim-Interfaces

4.3.4 Player-USARSim-Interface

Um die in Abschnitt 4.3.3 beschriebene Simulationsumgebung USARSim zusammen mit dem in Abschnitt 4.3.2 beschriebenen Framework Player nutzen zu können, muss eine Verbindung zwischen beiden hergestellt werden. Konkret bedeutet das, dass Player in der Lage ist, sich zu verschiedenen Robotern zu verbinden, deren Sensordaten zu empfangen und ihnen Fahrkommandos schicken zu können. Wie in Abbildung 4.13 zu sehen ist, bildet die Klasse *UsBot* die zentrale Klasse des Interfaces. Diese liest die Player Konfigurationsdatei, in der steht, unter welcher IP-Adresse und unter welchem Port sie mit der USARSim Simulation kommunizieren kann. Des Weiteren enthält die Konfigurationsdatei den Robotertypen und die gewünschte Pose innerhalb der Simulation, an der der Roboter in der Initialisierungsphase erzeugt werden soll. Ferner wird angegeben, welche Sensoren des simulierten Roboters im Player Framework zur Verfügung stehen sollen. Für jeden angegebenen Sensor wird ein Playertreiber erstellt, der das dem Sensortypen entsprechende Player Interface implementiert.

Folgender Kommunikationszyklus beschreibt die Reaktion des Player-USARSim-Interfaces auf eine Konfigurations- bzw. Geometrieanfrage eines Playermoduls.

- Es wird eine Konfigurations- oder Geometrieanfrage von einem Device an eines der Devices gestellt, das einen Sensor in USARSim repräsentiert. Diese Anfrage wird in die korrespondierende USARSim Anfrage konvertiert und in den Nachrichtenpuffer der *UsBot*-Klasse eingefügt.
- Die *UsBot*-Klasse prüft in jedem Zyklus, ob Nachrichten vorhanden sind, die über die TCP/IP-Verbindung zum USARSim Server geschickt werden müssen.
- USARSim verarbeitet die Anfrage und schickt die passende Antwort zurück.
- Die Nachricht wird von der *UsBot*-Klasse empfangen und mit Hilfe des Parsers in den entsprechenden Konfigurations- oder Geometriedatentyp der Sensorklasse konvertiert, die die Anfrage ursprünglich erhalten hatte.
- Die Sensorklasse leitet die Antwort an das Device weiter, welches die Anfrage gestellt hat.

Parallel zu diesen Anfragen werden ständig die Sensordaten geparst, die USARSim ohne Anfrage permanent sendet. Anhand des Sensortyps und des Sensornamens, die in dem Sensordatenstring enthalten sind, wird die entsprechende Sensorklasse ermittelt und die Daten werden von dieser im passenden Player-Datenformat an alle unterschriebenen Devices versendet.

Kapitel 5

Dreidimensionale Kartierung der Arbeitsumgebung

Um 3D-Sensordaten in den Lokalisierungsprozess integrieren zu können, wird ein 3D-Modell der Umgebung benötigt. Dieses Kapitel beschreibt, wie die Erstellung eines 3D-Umgebungsmodells automatisiert werden kann. Im Rahmen des LiSA-Projekts wird das Modell neben der Lokalisierung zur globalen Pfadplanung eingesetzt. Ferner wird es als Grundlage zur Erstellung der Simulationsumgebung herangezogen und zur Mensch-Roboter-Interaktion eingesetzt.

Im LiSA-Projekt wird von einer festen Gebäudegeometrie im Roboterumfeld ausgegangen. Die Arbeitsumgebung wird im Vorfeld des Robotereinsatzes dreidimensional vermessen. Dies ermöglicht die Modellierung statischer Hindernisse im gesamten Arbeitsumfeld. Es handelt sich bei dem LiSA-Szenario also nicht um ein „Simultaneous Localization and Mapping“ (SLAM)-Problem, bei dem der Roboter eine unbekannte Umgebung erforschen und sich gleichzeitig in dieser lokalisieren muss, sondern um ein Lokalisierungsproblem, da die Umgebung als a priori bekannt vorausgesetzt wird.

5.1 Dreidimensionale Aufnahme der Umgebung als Punktwolke

Die automatisierte Geometrieextraktion ist ein zweigeteilter Prozess. Zunächst muss die Umgebung vermessen werden, und anschließend muss aus diesen Daten die zugrunde liegende geometrische Struktur extrahiert werden.

Die Umgebung wird in dieser Arbeit mit einem autonomen Roboter vermessen, der mit einem 3D-Laserscanner ausgestattet ist. Dies ermöglicht ihm vollautomatisch eine 3D-Punktwolke der Umgebung zu erzeugen. Hierzu werden 3D-Scans der Umgebung von verschiedenen Standorten aufgenommen und durch Scanmatching in ein gemeinsames Koordinatensystem

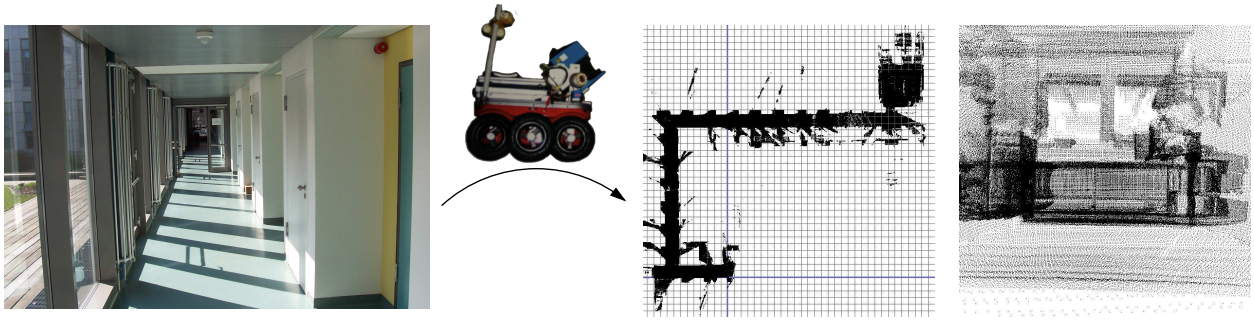


Abbildung 5.1: 3D-Umgebungsaufnahme und Rekonstruktion. Links: Die reale Umgebung wird mit dem autonomen mobilen Roboter Kurt3D (Mitte) vermessen. Rechts: Aufsicht auf die erstellte dreidimensionale Punktwolke der Laborumgebung, wobei ein Kästchen die Seitenlänge $1m$ besitzt. Perspektivische Sicht eines Ausschnitts der Punktwolke.

transformiert (registriert). Ein einzelner Scan wird aufgenommen, indem der 2D-Laserscanner, der als Basis des 3D-Laserscanners dient, mit Hilfe eines Servomotors um die horizontale Achse genickt wird. Über den jeweiligen Nickwinkel lassen sich die Abstandswerte auf Punkte im dreidimensionalen Raum abbilden.

Das Registrieren mehrerer Scans in einem gemeinsamen Koordinatensystem von verschiedenen Aufnahmeorten in derselben Szene geschieht mit Hilfe eines „Iterative Closest Points“ (ICP)-basierten Scanmatching-Verfahrens [83]. Dieses wählt zufällig Punktpaare aus zwei zu kombinierenden Scans aus und bewertet die Lage der Scans zueinander über eine Fehlerfunktion, die diese Punktpaare verwendet. Die Fehlerfunktion betrachtet den euklidischen Abstand der beiden Punkte eines Punktpaares. Je mehr Punktpaare einen geringen Abstand besitzen, desto wahrscheinlicher ist es, dass die Punkte durch dasselbe Objekt hervorgerufen werden, das in überlappenden Bereichen beider Scans sichtbar ist. Die Wahl zufälliger Punktpaare und deren Bewertung wird mit verschiedenen Transformationen in allen sechs Dimensionen des zu matchenden Scans iterativ solange wiederholt, bis die Fehlerfunktion unter einem definierten Schwellwert liegt. Die gefundene 6D-Pose ist die wahrscheinlichste Aufnahmepose des zu matchenden Scans im globalen, gemeinsamen Koordinatensystem. Das Ergebnis des Scanmatching-Verfahrens ist eine gemeinsame Punktwolke aller Scans, die eine konsistente Repräsentation der Umgebung liefert (vgl. Abbildung 5.1).

5.2 Semantisch gestütztes Scanmatching

Da der verwendete 3D-Laserscanner [82] einen Servomotor verwendet, um den 2D-Laserscanner zu rotieren, besitzen die 3D-Punkte eines Scans die Ungenauigkeit der Winkelauflösung des Servomotors. Diese führt dazu, dass der Fußboden und die Decke in den Scans oft nicht eben und parallel zueinander sind. Dadurch resultiert das Scanmatching oft in einer bananenförmig registrierten Punktwolke, wie sie in Abbildung 5.2 rechts oben zu sehen ist.

Um dieses Problem zu lösen, wird in dieser Arbeit a priori angenommen, dass die einzelnen Scans in Gebäuden aufgenommen werden. Die Information über die Geometrie von Gebäuden wird genutzt, um die Ungenauigkeit der Sensordaten auszugleichen. Hier wird im Speziellen die Information verwendet, dass der Boden innerhalb der Gebäude eben ist. Um diese Information auszunutzen, werden zunächst vor der Registrierung der Scans in den einzelnen Scans die Punkte identifiziert, die zum Boden gehören. Dies wird nach dem in [60] vorgestellten Verfahren durchgeführt. Dazu werden durch den Scan von der Scannerposition ausgehend vertikale Schnitte gelegt. In einem Zylinderkoordinatensystem besitzt der Scan die Darstellung $p_{i,j} = (\Phi_i, r_{i,j}, z_{i,j})^T$, wobei Φ_i der Drehwinkel der jeweiligen vertikalen Schnittebene ist und $z_{i,j}$ bzw. $r_{i,j}$ die Koordinaten eines der sich in der vertikalen Ebene i befindlichen Punktes j sind. Der Gradient $\alpha_{i,j}$ zwischen zwei benachbarten Punkten lässt sich mittels folgender Gleichung bestimmen:

$$\alpha_{i,j} = \arctan\left(\frac{z_{i,j} - z_{i,j-1}}{r_{i,j} - r_{i,j-1}}\right) \quad \text{mit} \quad -\frac{1}{2}\pi \leq \alpha_{i,j} < \frac{3}{2}\pi \tag{5.1}$$

Der Vergleich von $\alpha_{i,j}$ mit einem festen Schwellwert τ ermöglicht eine Interpretation des Punktes:

1. $\alpha_{i,j} < \tau$: Punkt j ist ein Bodenpunkt

2. $\tau \leq \alpha_{i,j} \leq \pi - \tau$: Punkt j ist ein Objektpunkt

3. $\pi - \tau < \alpha_{i,j}$: Punkt j ist ein Deckenpunkt

Nach der Klassifizierung der Punkte eines Scans wird in alle Bodenpunkte eine Ebene gefittet. Hierzu wird das im Anhang B.2 beschriebene Verfahren verwendet. Es wird die Transformationsmatrix zwischen der gefundenen Ebene und der globalen x, y-Ebene bestimmt. Diese Matrix wird vor dem Scanmatching auf den gesamten Scan angewendet, so dass alle Scans vor dem Scanmatching in einer gemeinsamen Bodenebene ausgerichtet sind. Damit diese Ausrichtung nicht verloren geht, wird das Scanmatching von allen sechs Dimensionen auf drei Dimensionen reduziert. Die z-Koordinate sowie die Drehungen um die x- und y-Achse werden als fest angenommen.

Abbildung 5.2 zeigt das Ergebnis der beiden unterschiedlichen Herangehensweisen beim Scanmatching. Oben rechts ist die bereits erwähnte bananenförmige registrierte Punktwolke zu sehen, die durch Scanmatching in 6D und ohne Bodenausrichtung entsteht. Darunter ist das Ergebnis des semantisch gestützten Scanmatchings zu sehen, das eine horizontal ausgerichtete registrierte Punktwolke erzeugt. Vergleichsmessungen in der realen Umgebung zeigen im Vergleich mit der registrierten Punktwolke, dass diese nur einen Fehler von etwa 2% besitzt.

Abbildung 5.2: Links: Aufsicht auf die registrierte Punktwolke der Umgebung. Rechts, oben: 6D-Scanmatching ohne semantische Korrektur erzeugt eine bananenförmige Umgebung. Rechts, unten: 3D-Scanmatching mit semantischer Korrektur erzeugt qualitativ korrekte Umgebungsmodelle [81].

5.3 Geometrierekonstruktion aus 3D-Punktwolken

5.3.1 Extraktion von 3D-Modellen

Um den Prozess der 3D-Kartenerstellung aus Punktwolken zu automatisieren, muss im zweiten Schritt die der Punktwolke zu Grunde liegende Geometrie des Gebäudes extrahiert werden.

Im Rahmen dieser Arbeit wurde die 3D-Houghtransformation getestet, die in der Lage ist beliebige parametrisierbare geometrische Primitive an Punktwolken zu fitten [65]. Ferner wurde ein RANSAC-Verfahren [20] getestet, um die Geometrie zu extrahieren. Diese beiden Verfahren besitzen jedoch den Nachteil, dass sie unbegrenzte Ebenen in die Punktwolke fitten und durch eine anschließende Betrachtung aller Punkte, die der jeweiligen Ebene zugeordnet werden, ein begrenztes Polygon erzeugt werden muss. Die beiden Verfahren sind im Anhang beschrieben und werden aufgrund ihrer schlechten Performanz im Vergleich zum im folgenden beschriebenen Verfahren hier nicht näher erläutert.

In dieser Arbeit wird ein kombiniertes Verfahren aus Marching Cubes [43] mit anschließendem Region Growing eingesetzt. Es wurde 2007 von Wiemann entwickelt [94]. Der Marching Cubes Algorithmus unterteilt den 3D-Raum, in dem sich die Punktwolke befindet, in Würfel fester Seitenlänge. Die Auflösung des späteren Modells ist abhängig von der Größe der Würfel. Anschließend werden alle Würfel betrachtet, in denen sich Punkte der 3D-Punktwolke befinden. In alle Punkte eines Würfels wird mit einem „Least Squares“-Verfahren eine Ebene gefittet. Durch die Betrachtung, welcher der acht Ecken des Würfels sich auf welcher Seite der gefitteten Ebenen befindet, wird dem Würfel eins von 16 verschiedenen polygonalen Approximationsmustern zugeordnet. Die Approximationsmuster der einzelnen Würfel bilden in Verbindung mit den Approximationsmustern der jeweils benach-

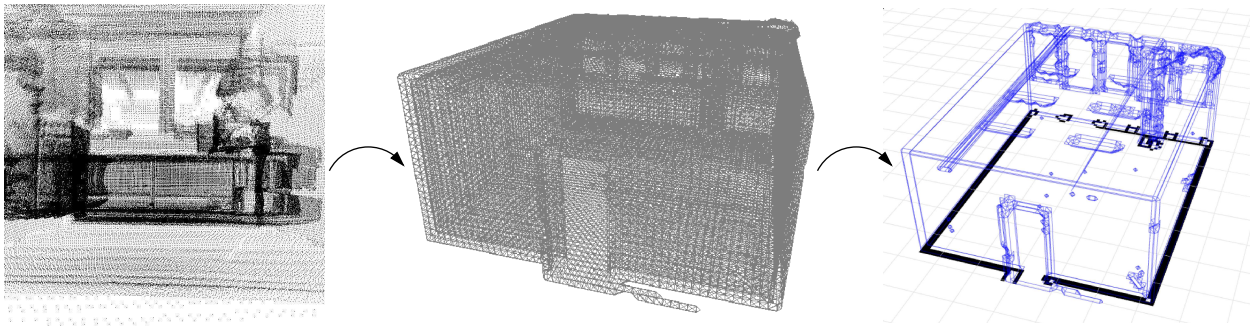


Abbildung 5.3: Marching Cubes basierte Geometrieextraktion aus einer Punktwolke. Das extrahierte Dreiecksnetz (Mitte) wird mit einem Region Growing Algorithmus zu großen Polygonen zusammengefasst (rechts)[94].

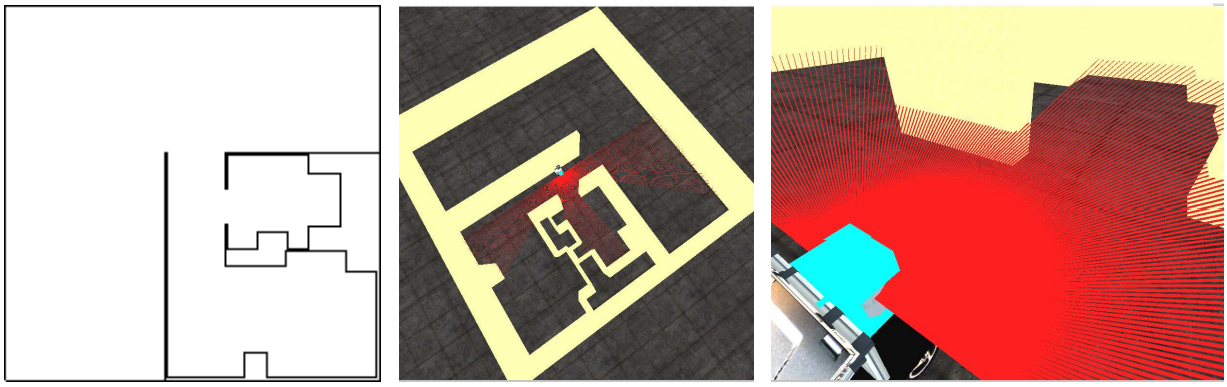
barten Würfel ein geschlossenes Dreiecksnetz, das die Punktwolke als polygonale Struktur der Punktwolke wiedergibt. Es gibt einige Kombinationen von Approximationsmustern, bei denen kein geschlossenes Netz erzeugt wird. Diese sind jedoch einfach zu identifizieren und zu beheben. Da ein einzelnes Approximationsmuster aus bis zu vier Dreiecken besteht, entstehen beim Marching Cubes-Verfahren schnell Dreiecksnetze, die sehr viele Dreiecke haben und somit rechnerisch schlecht handhabbar sind. So besitzt eine Szene von $10\,\mathrm{m} \times 10\,\mathrm{m} \times 3\,\mathrm{m}$ bei einer Würfelkantenlänge von 10 cm 300000 Würfel. Es entsteht also im schlimmsten Fall ein Dreiecksnetz mit $1,2$ Millionen Dreiecken.

Um die Anzahl der Dreiecke im Dreiecksnetz zu reduzieren, wird dieses in [94] optimiert. Es wird ausgenutzt, dass es nur 16 verschiedene Approximationsmuster gibt und bestimmte Kombinationen von benachbarten Approximationsmustern planare Flächen erzeugen. Um diese ebenen Flächen zu finden, wird ein Region Growing Algorithmus auf die Würfel angewendet, der bei den jeweiligen Nachbarwürfeln nach passenden Approximationsmustern sucht. Anschließend werden alle Dreiecke, die innerhalb einer ebenen Fläche liegen, zu einem Polygon vereint. Wie in Abbildung 5.3 dargestellt, führt dies, je nachdem wie viele ebenen Strukturen die Punktwolke enthält, zu einer erheblichen Reduktion der Dreiecke im Netz.

Abbildung 5.3 (rechts) zeigt ebenfalls, dass es möglich ist eine klassische zweidimensionale Rasterkarte (schwarze Linien) mit Hilfe eines Schnitts durch das 3D-Modell zu erstellen. Diese 2D-Karte kann z.B. zur globalen Pfadplanung eingesetzt werden.

5.3.2 Automatisierte Erstellung der Simulationsumgebung

Um eine Simulationsumgebung aus dem automatisch generierten 3D-Modell zu erzeugen, muss dieses manuell nachgearbeitet werden. Dies liegt vor allem daran, dass die 3D-Umgebungsaufnahme die Umgebung nie komplett erfassen kann. So kommt es durch Verdeckung dazu, dass bestimmte Bereiche der Umgebung nicht in der Punktwolke vorhanden sind. Diese Bereiche führen zu Löchern in dem mit Hilfe des Marching Cubes-Verfahrens erstellten 3D-Modell. Wie in Kapitel 6.1.5 beschrieben ist, wird jedoch zur Lokalisierung ein geschlossenes



(a) (b) (c)

Abbildung 5.4: Simulationserstellung aus einer 2D-Rasterkarte: Die Rasterkarte (a) wird automatisch in eine Simulationsumgebung umgewandelt ((b) und (c))

Polygonmodell benötigt.

Aus diesem Grund wird das reduzierte Netz in dieser Arbeit zur Unterstützung der manuellen Erstellung der Simulationsumgebung und eines 3D-Umgebungsmodells eingesetzt (vgl. Abschnitt 4.3.3). Dazu wird das reduzierte 3D-Netz in ein 3D-Modellierungsprogramm geladen, und das 3D-Umgebungsmodell wird manuell an dieses Netz angepasst.

Um automatisiert Simulationsumgebungen erstellen zu können, wird in dieser Arbeit ein alternativer Ansatz verfolgt. Als Eingangsdaten dieses Ansatzes werden 2D-Rasterkarten verwendet. Diese lassen sich zum einen, wie bereits erwähnt, als Schnitt durch das 3D-Modell erzeugen, zum anderen können diese mit klassischen zweidimensionalen SLAM-Verfahren oder manuell mit einem einfachen Grafikprogramm erstellt werden.

Das Verfahren nutzt die Möglichkeit der Simulationsumgebung USARSim, dynamisch zur Laufzeit der Simulation Gegenstände in die simulierte Umgebung einzufügen. Für jedes belegte Raster wird ein Quader, dessen Höhe dynamisch festgelegt werden kann, in die Simulationsumgebung eingefügt. Das Ergebnis dieser Vorgehensweise ist in Abbildung 5.4 dargestellt. Teilabbildung (a) zeigt die 2D-Rasterkarte, die als Vorlage zur Erstellung der Simulationsumgebung dient. Die Teilabbildungen (b) und (c) stellen die daraus erstellte Simulationsumgebung dar.

Da der Ansatz 2D-Rasterkarten als Eingabedaten verwendet, lässt sich die Lokalisierung unter Verwendung von 3D-Daten nicht in dieser Simulationsumgebung testen. Dennoch hat dieser Ansatz interessante Anwendungen. So lässt sich eine Simulationsumgebung parallel zur Umgebungsaufnahme der Umgebung durch den realen Roboter erstellen. Dies kann z.B. für einen „Next Best View“-Planer des realen Roboters genutzt werden, indem die Sensordaten des simulierten Roboters an verschiedenen Posen der Simulationsumgebung ausgewertet werden. Da Player die 2D-Rasterkarten aus Pixelgrafiken lädt, lassen sich mit diesem Ansatz auch sehr schnell Simulationsumgebungen mit einfachen Grafikprogrammen erzeugen. Dies ist z.B. sinnvoll, wenn man testen möchte, wie eine bestimmte Hindernisvermeidungsstrategie

sich in einer bestimmten Situation verhält.

Im nächsten Kapitel wird die Navigation eines autonomen Roboters unter Verwendung von 3D-Sensordaten beschrieben. Hierzu wird im Rahmen der Lokalisierung ein 3D-Umgebungsmodell eingesetzt, das wie oben beschrieben von der Roboterumgebung erstellt wurde. Eine 2D-Schnittkarte des Modells wird zur globalen Pfadplanung eingesetzt, wie sie in Abschnitt 6.3 beschrieben wird. Die Algorithmen zur Navigation des Roboters werden in der Simulationsumgebung getestet, die durch die oben beschriebenen Verfahren leicht an bestimmte Umgebungssituationen angepasst werden kann.

Kapitel 6

Navigation

In diesem Kapitel werden die Themen beschrieben, deren Zusammenspiel die autonome Navigation eines mobilen Roboters in einer bekannten Umgebung ermöglichen. Es gibt drei klassische Forschungsdisziplinen: Lokalisierung, Hindernisvermeidung und Pfadplanung. Die Lokalisierung versucht, anhand von Sensordaten die Pose des Roboters in dem a priori bekannten Umgebungsmodell zu finden. Es ist anschaulich klar, dass ein autonomer Roboter nur dann sinnvoll agieren kann, wenn er weiß, wo er sich befindet. Die zweite Disziplin ist die Hindernisvermeidung. Diese übernimmt die lokale Bewegungsplanung des Roboters, um anhand der Sensordaten Kollisionen mit Objekten im Roboterumfeld zu vermeiden. Da diese Verfahren meistens rein reaktiv und lokal arbeiten ist die Hindernisvermeidung im Allgemeinen unabhängig von der Lokalisierung. Die dritte Disziplin ist die Pfadplanung. Diese übernimmt die globale Bewegungsplanung des Roboters in der bekannten Umgebung. Sie bestimmt anhand des Umgebungsmodells den Pfad, über den der Roboter von Punkt A nach Punkt B gelangt.

In dieser Arbeit wird der Schwerpunkt auf die Bereiche Lokalisierung und Hindernisvermeidung gelegt. Im Bereich der Lokalisierung wird eine Optimierung des am häufigsten verwendeten Lokalisierungsverfahrens vorgestellt, die die Verwendung von 3D-Umgebungsmodellen erlaubt. Im Bereich der Hindernisvermeidung wird ein neuartiges Konzept vorgestellt, das die Fusion beliebiger Abstandssensoren in den Hindernisvermeidungsprozess erlaubt. Dieses Verfahren ermöglicht eine sichere Hindernisvermeidung mit allen Objekten im 3D-Umfeld des Roboters unter Verwendung klassischer Hindernisvermeidungsverfahren, die ausschließlich im zweidimensionalen Raum rechnen.

6.1 Lokalisierung

Lokalisierung beschreibt im Zusammenhang der mobilen Robotik die Fähigkeit des Roboters den Ort zu bestimmen, an dem er sich befindet. Dies bedeutet, dass er seine Pose in einem festen, globalen Koordinatensystem bestimmt.

6.1.1 Markow-Lokalisierung

Probabilistische Lokalisierungsverfahren berücksichtigen die Tatsache, dass Sensormessungen immer mit Rauschen behaftet und Fehlmessungen möglich sind. Dies berücksichtigend, wird die Pose eines Roboters im Gegensatz zu deterministischen Verfahren immer mit einer Unsicherheit behaftet. Die Pose wird als Maximum einer Wahrscheinlichkeitsdichte bestimmt. Grundlage aller im Folgenden beschriebenen probabilistischen Verfahren ist der „Bayesische Filter“. Dieser rechnet mit bedingten Wahrscheinlichkeiten und ermöglicht das Umkehren von Schlussfolgerungen. Bayesische Filter werden immer dann eingesetzt, wenn von zwei bedingten Wahrscheinlichkeiten $p(a|b)$ bzw. $p(b|a)$ eine relative einfach zu bestimmen ist, so dass es möglich ist, anhand dieser die umgekehrte bedingte Wahrscheinlichkeit zu berechnen. Wird der Bayesische Filter auf das Problem der Roboterlokalisierung angewendet spricht man von Markow-Lokalisierung (ML) und die bedingten Wahrscheinlichkeiten sind:

1. $p(\text{Roboterpose}|\text{Messungen}) \equiv p(x_t|d_{t\ldots0})$
2. $p(\text{Messung}|\text{Roboterpose und vorherige Messungen}) \equiv p(d_t|x_t, d_{t-1\ldots0})$

Der erste Fall beschreibt die Wahrscheinlichkeit, dass sich der Roboter bei allen gegebenen Messungen bis zur Zeit t in der Pose x_t befindet. Dieser Fall ist der, der in der Realität auftritt und das Lokalisierungsproblem beschreibt. Diese bedingte Wahrscheinlichkeit ist im Gegensatz zur umgekehrten bedingten Wahrscheinlichkeit relativ schwer zu bestimmen. Die umgekehrte bedingte Wahrscheinlichkeit beschreibt den Fall, dass die Pose x_t des Roboters zur Zeit t gegeben ist und die Wahrscheinlichkeit gesucht wird, mit der eine Messung d_t auftritt. Bei gegebener Repräsentation der Umgebung ist dies relativ einfach zu bestimmen [84]. In der probabilistischen Robotik wird eine Unterscheidung zwischen Messungen o (Laser, Ultraschall,...) und Aktionen a (Fahrkommandos) vorgenommen. Da aber Fahrkommandos nur indirekt über die Odometrie messbar sind, gehören beide Gruppen zu den wahrnehmbaren Messungen d. Durch die Aufteilung der Messungen in diese beiden Gruppen und die Anwendung der Bayesischen Regel ergibt sich folgender Zusammenhang zwischen den beiden bedingten Wahrscheinlichkeiten:

$$p(x_t|a_{t\ldots0}, o_{t\ldots0}) = \frac{p(o_t|x_t, o_{(t-1)\ldots0}, a_{t\ldots0}) \cdot p(x_t|o_{(t-1)\ldots0}, a_{t\ldots0})}{p(o_t|o_{(t-1)\ldots0}, a_{t\ldots0})}$$

Da der Nenner unabhängig von der Pose x_t des Roboters ist, kann dieser durch eine Normalisierungskonstante α ersetzt werden, die sicherstellt, dass die Gesamtwahrscheinlichkeit aller möglichen Roboterposen x_t zu eins summiert.

$$p(x_t|a_{t\ldots0}, o_{t\ldots0}) = \alpha\, p(o_t|x_t, o_{(t-1)\ldots0}, a_{t\ldots0}) \cdot p(x_t|o_{(t-1)\ldots0}, a_{t\ldots0}) \tag{6.1}$$

Unter Verwendung der Markow-Annahme, dass die aktuelle Messung nur vom aktuellen Zustand abhängt und unabhängig von früheren Messungen ist, ergibt sich aus Gleichung 6.1:

$$p(x_t|a_{t\ldots0}, o_{t\ldots0}) = \alpha\, p(o_t|x_t) \cdot p(x_t|o_{(t-1)\ldots0}, a_{t\ldots0})$$

Mit Hilfe des Theorems über die absolute Wahrscheinlichkeit [84], welches folgende Beziehung für die bedingeten Wahrscheinlichkeiten zweier Zustände x, y bei einem kontinuierlichen Zustansraum liefert:

$$p(y) = \int p(y|x) \cdot p(x)dx$$

ergibt sich aus Gleichung 6.1.1:

$$p(x_t|a_{t...0}, o_{t...0}) = \alpha\, p(o_t|x_t) \cdot \int p(x_t|x_{(t-1)}, o_{(t-1)...0}, a_{t...0}) \cdot p(x_{(t-1)}|o_{(t-1)...0}, a_{t...0})dx_{(t-1)}$$

Erneute Anwendung der Markow-Annahme und die Tatsache, dass das Fahrkomando a_t den Zustand x_{t-1} nicht beeinflusst ergibt:

$$\underbrace{p(x_t|a_{t...0}, o_{t...0})}_{aktueller\ Zustand} = \alpha \quad \underbrace{p(o_t|x_t)}_{Sensormodell} \cdot \int \underbrace{p(x_t|x_{(t-1)}, a_t)}_{Bewegungsmodell} \cdot \underbrace{p(x_{t-1}|o_{(t-1)...0}, a_{(t-1)...0})}_{letzter\ Zustand} dx_{(t-1)}. \quad (6.2)$$

Wie in Gleichung 6.2 zu sehen ist, gibt das sogenannte „Bewegungsmodell“ die Wahrscheinlichkeit an, mit der eine Pose anhand eines Fahrkommandos in eine neue Pose überführt wird. Das „Sensormodell“ gibt die Wahrscheinlichkeit an, dass der Roboter in der Pose x_t die Messung o_t erzeugt.

Gleichung 6.2 zeigt, dass zur Berechnung des neuen Zustands aufgrund des kontinuierlichen Zustandsraums unendlich viele Posen des Roboters berücksichtigt werden müssen. Die ML lässt offen, wie die Zustandsraum repräsentiert und das Bewegungs- bzw. Sensormodell implementiert werden.

Im Folgenden werden mit dem „Kalman-Filter“, der „Grid-Lokalisierung“ und der „Monte-Carlo-Lokalisierung“ konkrete Implementierungen bzw. Spezialisierungen der ML vorgestellt.

6.1.2 Kalman-Filter

Der Kalman-Filter ist die am besten untersuchte Implementierung des Bayeschen Filters. Er gehört zur Familie der Gaußfilter und ist einer der ersten rechnerisch handhabbaren Implementierungen des Bayschen Filters [84]. Der Kalman-Filter bestimmt die Wahrscheinlichkeit einer Variable mit Hilfe zweier Variablen: der Kovarianz Σ und dem Erwartungswert μ.

$$p(x_t) = \det(2\pi\Sigma_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_t - \mu)^T \Sigma_t^{-1}(x_t - \mu)\right)$$

Diese Darstellung der Wahrscheinlichkeitsverteilung wird momentenbasierte Repräsentation genannt, da der Erwartungswert und die Kovarianz das erste und zweite Moment der Wahrscheinlichkeitsverteilung sind. Der Kalman-Filter rechnet im kontinuierlichen Raum und der Übergang zwischen den Wahrscheinlichkeiten eines Zustands ist eine lineare Funktion.

$$x_t = A_t x_{t-1} + B_t a_t + \epsilon_t$$

Wobei A_t und B_t Matrizen sind. ϵ_t ist ein Gaußscher Zufallsvektor, dessen Erwartungswert Null und Kovarianz R_t ist. Für das Bewegungsmodell ergibt sich:

$$p(x_t|x_{(t-1)}, a_t) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t a_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t a_t)\right)$$

Das Sensormodell $p(o_t|x_t)$ muss ebenfalls linear in seinen Argumenten sein.

$$o_t = C_t x_t + \delta_t$$

Analog zum Bewegungsmodell ist C_t eine Matrix und δ_t modelliert das Sensorrauschen als multivariate Gaußfunktion, deren Erwartungswert Null und Kovarianz Q_t beträgt. Somit ergibt sich für das Sensormodell beim Kalman-Filter:

$$p(o_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(o_t - C_t x_t)^T Q_t^{-1}(o_t - C_t x_t)\right)$$

Mit diesem Bewegungs- und Sensormodell ist, unter der Annahme, dass der initiale Zustand x_0 normalverteilt ist, sichergestellt, dass die Wahrscheinlichkeit eines Zustands x_t immer durch eine Gaußfunktion dargestellt wird. Da eine Gaußfunktion unimodal ist, hat dies zur Folge, dass der Kalman-Filter immer nur die Wahrscheinlichkeit eines einzigen Zustands über die Zeit bestimmt. Der Kalman-Filter ist nur für Probleme geeignet, die einen linearen Zustandsübergang besitzen. Diese kommen jedoch in der Realität sehr selten vor. Aus diesem Grund wurde mit dem Extended-Kalman-Filter (EKF) dieses Konzept erweitert und es können nicht lineare Funktionen g und h beim Übergang zwischen den Zustandswahrscheinlichkeiten verwendet werden.

$$x_t = g(a_t, x_{t-1}) + \epsilon_t \quad \text{und} \quad o_t = h(x_t) + \delta_t$$

Trotz dieser Erweiterung bleibt der EKF einfach und effizient zu berechnen und bildet trotz der Tatsache, dass es weiterhin nur einen Zustand verfolgt, eines der am häufigsten Verfahren zur Zustandsbestimmung in der Robotik. Das Problem der globalen Lokalisierung bzw. das „kitnapped robot problem“ kann also mit dem EKF nicht gelöst werden. Aus diesem Grund wurden Verfahren entwickelt, die z.B. Summen von Gaußfunktionen zur Zustandsbeschreibung verwenden (Multi-Hypothesis Extended Kalman Filter).

6.1.3 Grid-Lokalisierung

Die Grid-Lokalisierung ist eine konkrete Implementierung der ML. Bei der Grid-Lokalisierung wird die Quantisierung des Zustandsraums durch Verwendung einer Rasterkarte erreicht. Diese Lösung besitzt je nach Auflösung der Rasterkarte eine hohe Rechenkomplexität und einen enormen Speicherbedarf [24]. Dadurch, dass der gesamte Zustandsraum des Roboters quantisiert wird, ist die Grid-Lokalisierung in der Lage das globale sowie das kidnapped robot Lokalisierungsproblem zu lösen. Für jede Zelle der Rasterkarte wird die Wahrscheinlichkeit berechnet, dass sich der Roboter in diesem Zustand befindet. Aus diesem Grund sind nur

sehr kleine bzw. sehr grob aufgelöste Umgebungen des Roboters rechnerisch handhabbar. Ferner ist die Lokalisierung mit dem systematischen Fehler der halben Rasterzellenbreite behaftet.

Eine wesentlich elegantere Methode zur Verfolgung mehrerer Hypothesen ist die im folgenden beschriebene Monte-Carlo-Lokalisierung. Sie ist in der Lage die Wahrscheinlichkeit für mehrere Hypothesen im kontinuierlichem Raum zu bestimmen.

6.1.4 Monte-Carlo-Lokalisierung

Die Monte-Carlo-Lokalisierung (MCL) ist ein 1999 entwickeltes Lokalisierungsverfahren für autonome mobile Roboter [17]. Es ist eine Variante des sogenannten „Partikelfilters“. Partikelfilter werden eingesetzt, um nicht direkt messbare Parameter eines Systems anhand indirekter verrauschter Messungen des Systems zu bestimmen. Hierzu wird anhand eines Modells des zu untersuchenden Systems eine Wahrscheinlichkeitsdichte im Zustandsraum erzeugt und mit dieser der wahrscheinlichste Systemzustand bestimmt. Jeder mögliche Zustand wird durch einen Partikel beschrieben, der eine Gewichtung besitzt, die die Wahrscheinlichkeit des Zustands beschreibt. Im Fall der Roboterlokalisierung beschreiben die Partikel mögliche Posen des Roboters. Die Wahrscheinlichkeit, dass sich der Roboter in der Pose x_t befindet, ist proportional zur Menge aller Partikel:

$$p(x_t|a_{t\ldots 0}, o_{t\ldots 0}) \propto \{x_t^i, w_t^i\}_{i=1,\ldots,n}$$

Die MCL ist ein iterativer Bayesischer Filter und arbeitet nach dem dreistufigen „Sampling Importance Resampling“ (SIR) Verfahren.

1. Sampling: In der sogenannten „Prediction Phase“ werden anhand des Bewegungsmodells alle Partikel $\{x_{t-1}^i, w_{t-1}^i\}$ der letzten Iteration verschoben. Alle Partikel besitzen dieselbe Gewichtung w_{t-1}^i und die Summe aller Partikelgewichte ist eins.
2. Importance: Anhand des Sensormodells $p(o_t|x_t)$ werden die einzelnen Partikelgewichte w_t^i bestimmt. Die Summe aller Partikelgewichtungen summiert nun nicht mehr zu eins.
3. Resampling: Anhand der gewichteten Partikel $\{x_{t-1}^i, w_t^i\}$ werden neue Partikel $\{x_t^i, w_t^i\}$ erzeugt. Dabei werden in der Umgebung von Partikeln mit hoher Gewichtung mehr neue Partikel erzeugt. Partikel mit einem geringen Gewicht werden nicht mit in die neue Menge übernommen. Die neu erzeugten Partikel besitzen alle dieselbe Gewichtung und die Summe aller Partikelgewichte ist eins.

Das iterative Anwenden der drei Schritte führt dazu, dass sich die Partikel unter Berücksichtigung des Bewegungsmodells, der Sensordaten und des Umgebungsmodells an der wahrscheinlichsten Roboterpose konzentrieren. In den folgenden Modellen wird vor allem der „Importance“-Schritt näher betrachtet. In diesem Schritt muss bewertet werden, wie wahrscheinlich von einer bestimmten Pose ein bestimmter Messwert ist $p(o_t|x_t)$. Aus diesem Grund benötigt die MCL ein Umgebungsmodell, in dem die Entfernung zum nächsten Hindernis eines Partikels in Richtung der Sensordaten bestimmt wird. Über die Differenz der ermittelten

Entfernung zu den realen Sensordaten wird die Wahrscheinlichkeit des Partikels gewichtet. Häufig werden als Umgebungsmodell zweidimensionale Rasterkarten verwendet. Diese haben jedoch mehrere Nachteile: zum einen die Beschränkung auf zwei Dimensionen. Dies fordert eine Sensorkonfiguration mit horizontal angebrachten Entfernungssensoren. Ferner ist die Ortsauflösung in einer Rasterkarte nicht beliebig hoch. So ist die ermittelte Roboterpose immer mit der Unsicherheit der halben Auflösung der Rasterkarte behaftet.

6.1.5 Monte-Carlo-Lokalisierung mit einer 3D-Karte

In dieser Arbeit wird die MCL in einer zweidimensionalen Ebene anhand eines dreidimensionalen Modells der Umgebung realisiert. Das Umgebungsmodell wird benötigt, um im Importance-Schritt der MCL die Gewichte der Partikel zu bestimmen. Hierzu wird anhand des Modells bestimmt, wie wahrscheinlich die aktuellen Sensordaten von einem Partikel aus gemessen werden. Der Vorteil eines 3D-Modells liegt darin, dass dieses mehr Information über die modellierte Umgebung enthält als eine zweidimensionale Karte dieser Welt. Der Nachteil dieser Herangehensweise liegt darin, dass die Rechenkomplexität des Sensormodells $p(o_t|x_t)$ bei der Partikelgewichtung zunimmt, da die Schnittpunktbestimmung zwischen Sensordaten und Polygonen des 3D-Modells komplexer ist als die zwischen Sensordaten und einer zweidimensionalen Rasterkarte. Um untersuchen zu können, welche Vorgehensweise das Problem rechnerisch handhabbar macht, wird zunächst betrachtet, was die Rechenkomplexität verursacht. Die im Folgenden diskutierte Optimierung der MCL bezieht sich ausschließlich auf Abstandssensoren. Die MCL wird auch mit anderen Sensoren wie Kameras verwendet; das erfordert jedoch eine andere Herangehensweise zur Bestimmung der Partikelgewichte [97]. Im Folgenden wird anstatt von Sensordaten, die mit der MCL verwendet werden, von Laserstrahlen gesprochen, weil ein Laserscanner ein typischer Abstandssensor ist, der für die MCL verwendet wird. Ferner wird das Problem der Schnittpunktberechnungen bei Laserstrahlen als Sensordaten anschaulicher.

Pro Sekunde muss bei der MCL folgende Anzahl an Schnittpunktberechnungen durchgeführt werden.

$$\begin{aligned}\frac{\text{Schnittpunktberechnungen}}{\text{Sekunde}} = &\text{Aktualisierungsfrequenz} \cdot \text{Anzahl der Laserstrahlen} \cdot \\ &\text{Anzahl der Polygone des Modells} \cdot \text{Anzahl der Partikel}\end{aligned}$$

Mit typischen Werten für die einzelnen Felder ergibt sich:

$$\begin{aligned}\frac{\text{Schnittpunktberechnungen}}{\text{Sekunde}} &= 5 \cdot 180 \cdot 2000 \cdot 2000 \\ &= 3.600.000.000\end{aligned}$$

Die Schnittpunktberechnung stellt somit den kritischen Teil der MCL-Berechnung dar und ist mit diesen Werten nicht durchführbar. Es gibt verschiedene Herangehensweisen, die einzelnen Komponenten des Produktes zu reduzieren. Eine Verringerung der Aktualisierungs-

frequenz ist problematisch, da diese von der Geschwindigkeit des Roboters und der Genauigkeit der Odometriedaten abhängt. Bei einer hohen Robotergeschwindigkeit und einer ungenauen Odometrie wächst der Fehler zwischen Odometrie-Poseschätzung und der realen Pose proportional schneller als bei einer geringeren Robotergeschwindigkeit bzw. einer genaueren Odometrie. Bei einer zu geringen Aktualisierungsfrequenz ist die Distanz zwischen Odometrie-Poseschätzung und realer Pose zu groß, so dass die Aktualisierung der MCL anhand der Laserdaten diesen Fehler nicht ausgleichen kann, was zu einer fehlerhaften Lokalisierung führt.

Der zweite Faktor wird häufig reduziert, so dass nicht alle Werte eines Laserdatensatzes mit dem Umgebungsmodell verglichen werden. Typische Implementierungen verwenden je nach verwendeter Hardware etwa jeden zehnten Laserstrahl. Im folgenden Abschnitt wird ein in dieser Arbeit eingesetztes Verfahren beschrieben, das ebenfalls hier ansetzt. Dieser Ansatz verwendet nicht die Laserdaten direkt, sondern aus den Laserdatensätzen extrahierte Ebenenfragmente. Dadurch werden die zu betrachtenden Daten um eine Größenordnung von 10^2 reduziert.

Der dritte Faktor, die Anzahl der Polygone des 3D-Umgebungsmodells, werden im zweiten in dieser Arbeit eingesetzten Ansatz reduziert. Dies ist möglich, indem die Polygone in einer an das Problem angepassten effizienten Baumstruktur gespeichert werden. Dadurch werden die Polygone, die pro Schnittpunktsberechnung zwischen Laserstrahl und 3D-Modell betrachtet werden müssen, stark reduziert.

Der vierte Faktor, die Anzahl der Partikel, wird mit der sogenannten adaptiven MCL reduziert. Hier wird die Partikelanzahl an die Unsicherheit angepasst, die die aktuelle Lokalisierung besitzt [22]. Wenn die Lokalisierung sicher bzgl. der Roboterpose ist, befinden sich viele Partikel in einem eng begrenzten Bereich. Dadurch entsteht eine Redundanz der Partikel, so dass diese Anzahl reduziert werden kann. Bei einer unsicheren Lokalisierung hingegen kann kein eindeutiger Häufungspunkt von Partikeln identifziert werden. Aus diesem Grund wird die Anzahl der Partikel erhöht, um die globale Pose des Roboters wieder zu finden. Die Partikelanzahl ist der entscheidende Faktor, weshalb in dieser Arbeit weiterhin nur die Roboterpose in einer zweidimensionalen Ebene bestimmt wird. Theoretisch stehen durch das 3D-Umgebungsmodell bei passender 3D-Sensorkonfiguration (wie z.B. der des LiSA-Roboters) die Information über die 6D-Pose des Roboters zur Verfügung. Dies bedeutet, dass die Partikel der MCL ebenfalls sechs Dimensionen besitzen und auch über sechs Dimensionen verteilt werden müssen. Dazu sind so viele Partikel notwendig, dass die MCL nicht mehr handhabbar wird. Ferner wird beim LiSA-Szenario von einer ebenen Grundfläche ausgegangen, so dass eine Lokalisierung mit drei Freiheitsgraden ausreichend ist.

Bei der Verwendung von dreidimensionalen Daten im Zusammenhang mit der MCL wird eine dreidimensionale Karte der Umgebung benötigt. In dieser Arbeit wird das Dateiformat *obj* verwendet. Dieses lässt sich leicht aus anderen 3D-Formaten wie dem Standard zur Gebäudegeometriebeschreibung *CAD* konvertieren. Des Weiteren lässt es sich direkt aus der in dieser Arbeit verwendeten Simulationsumgebung USARSim exportieren (vgl. Kapitel 5).

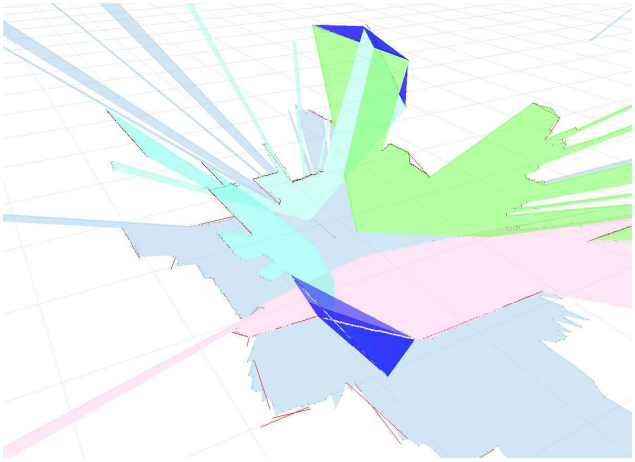


Abbildung 6.1: Extraktion von Ebenenfragmenten (dunkel blau) durch die Bestimmung von sich schneidenden Liniensegmenten (rot). In die Datensätze der vier Hokuyo-Laserscanner (hell blau, türkis, grün und rosa) sowie in VRS-Datensatz der beiden SICK-Laserscanner werden Liniensegmente gefittet. Schneiden sich zwei Liniensegmente von verschiedenen Laserscannern werden die Liniensegmente zu einem Ebenenfragment kombiniert.

Ebenensensor

Der Ebenensensor greift die in [93] vorgestellte Idee auf, als Eingabedaten der MCL Ebenenfragmente zu verwenden und diese mit dem Umgebungsmodell zu vergleichen. In dem dort vorgestellten Verfahren wird ein 3D-Laserscanner, wie er in Kapitel 5.1 beschrieben ist, eingesetzt um eine dreidimensionale Punktwolke der Umgebung zu erhalten. In diese Punktwolke werden Ebenen gefittet. Da die Aufnahme eines einzelnen 3D-Scans eine nicht vernachlässigbare Zeit benötigt, kann dieses Verfahren keine Lokalisierung bei kontinuierlicher Fahrt liefern.

Die kontinuierliche Lokalisierung anhand von Ebenenfragmenten wird in dieser Arbeit durch die permanente 3D-Erfassung der Umgebung durch die Sensorkonfiguration des LiSA-Roboters aus sechs Laserscannern realisiert. Hierzu werden rekursiv Liniensegmente in den Datensatz jedes Laserscanners gefittet. Im Gegensatz zu [57], wo nicht horizontale Liniensegmente direkt zu einem Ebenensegment extrapoliert werden, wird in dieser Arbeit die Sensorkonfiguration des LiSA-Roboters genutzt, um die Ebenenfragmente mit einer größeren Sicherheit zu bestimmen. Hierzu werden die Liniensegmente ausschließlich dann zu Ebenenfragmenten kombiniert, wenn sie sich in einem Punkt schneiden. Das Ebenenfragment ist durch die vier Endpunkte der sich schneidenden Liniensegmente definiert. Es werden jeweils Liniensegmente des rechten bzw. linken Hokuyo-Laserscanners mit denen des vorderen und hinteren Hokuyo-Laserscanners und den horizontalen Sick-Laserscannern verglichen. Der Vergleich zwischen linkem und rechtem Hokuyo-Laserscanner ist unnötig, da diese keinen gemeinsamen Sichtbereich haben und ihre Daten somit keine Linien desselben planaren Objekts enthalten können. Beim Vergleich wird der minimale Abstand jedes Segments zu allen Segmenten der anderen Laserscanner bestimmt. Unterschreitet dieser Abstand einen

festen Schwellwert, wird angenommen, dass die Segmente dasselbe planare Objekt zeigen, und die Liniensegmente werden zu einem Ebenenfragment kombiniert. Die Sendefrequenz des Ebenensensors wird durch die Frequenz eines Laserscanners bestimmt. Der Datensatz des Ebenensensors besteht aus allen Ebenenfragmenten, die seit dem letzten Senden extrahiert wurden. Bei der Gewichtung der Partikel im „Importance“-Schritt der MCL wird für jedes Ebenenfragment des Ebenensensors der Schwerpunkt berechnet. Dieser bestimmt die Richtung im 3D-Umgebungsmodell, unter der eine Ebene gesucht wird, die die vom Ebenensensor gefundene Ebene hervorruft. Aus Partikelpose und Ebenenrichtung wird ein Strahl gebildet und mit dem im folgenden Abschnitt beschriebenen Verfahren bestimmt, welches Polygon der Strahl als erstes schneidet. Um die Wahrscheinlichkeit der Pose zu bestimmen, wird das gefundene Polygon mit dem Ebenenfragment verglichen. Es wird zum einen eine Gaußverteilung über die Differenz zwischen der Entfernung des Ebenenfragments und des gefundenen Polygons zur Pose des Partikels gebildet. Des Weiteren wird eine Gaußverteilung über die Differenz zwischen der Normale des Ebenenfragments und des Polygons gebildet. Die Wahrscheinlichkeit einer Messung ergibt sich als Produkt der beiden Gaußkurven.

Die Gewichtung eines Partikels ergibt sich als Produkt aller Gewichtungen aller Messungen des aktuellen Sensordatensatzes des Ebenensensors.

Neben dem bereits erwähnten Vorteil der Datenreduzierung besitzt der Ebenensensor einen weiteren Vorteil gegenüber der direkten Verwendung von Sensordaten im Zusammenhang der MCL. Dieser liegt darin, dass die extrahierten Ebenenfragmente robust gegenüber Rauschen und Fehlmessungen der Laserscanner sind, da das Ebenenfragment von vielen Messpunkten gestützt wird.

Effiziente Schnittpunktberechnung durch den „Binary Space Partitioning“-Baum

Die Rechenkomplexität der MCL wird entscheidend dadurch beeinflusst, wie effizient der Schnittpunkt eines Laserstrahls mit dem verwendeten 3D-Modell bestimmt werden kann. Da komplexere Umgebungsmodelle häufig aus mehreren Tausend Polygonen bestehen, ist der iterative Test mit jedem Polygon nicht möglich, um das erste Polygon zu finden, das der Laserstrahl schneidet. Aus diesem Grund wird in dieser Arbeit der Binary Space Partitioning (BSP)-Baum als eine Datenstruktur zur Repräsentation der dreidimensionalen Umgebung eingesetzt, die eine effiziente Schnittpunktberechnung ermöglicht [6]. Der Vorteil des BSP-Baums ist, dass zur Schnittpunktberechnung jeweils nur ein kleiner Ausschnitt des 3D-Modells betrachtet werden muss. Neben dem BSP-Baum existieren weitere hierarchische, räumliche Datenstrukturen wie die Spezialisierung des BSP-Baums, der kD-Baum, sowie Octree oder die Bounding Volume Hierarchy. Dabei bilden BSP- und kD-Bäume die performantesten Strukturen. In dieser Arbeit wird der BSP-Baum verwendet, da mit dem Solid Leaf BSP-Baum (vgl. Abschnitt 6.1.5) eine Variante existiert, die an das Problem der Repräsentationen von Gebäudestrukturen angepasst ist. Im Folgenden wird jedoch zunächst die Erzeugung des generellen BSP-Baums beschrieben und danach auf die Spezialisierung des Solid Leaf BSP-Baums eingegangen.

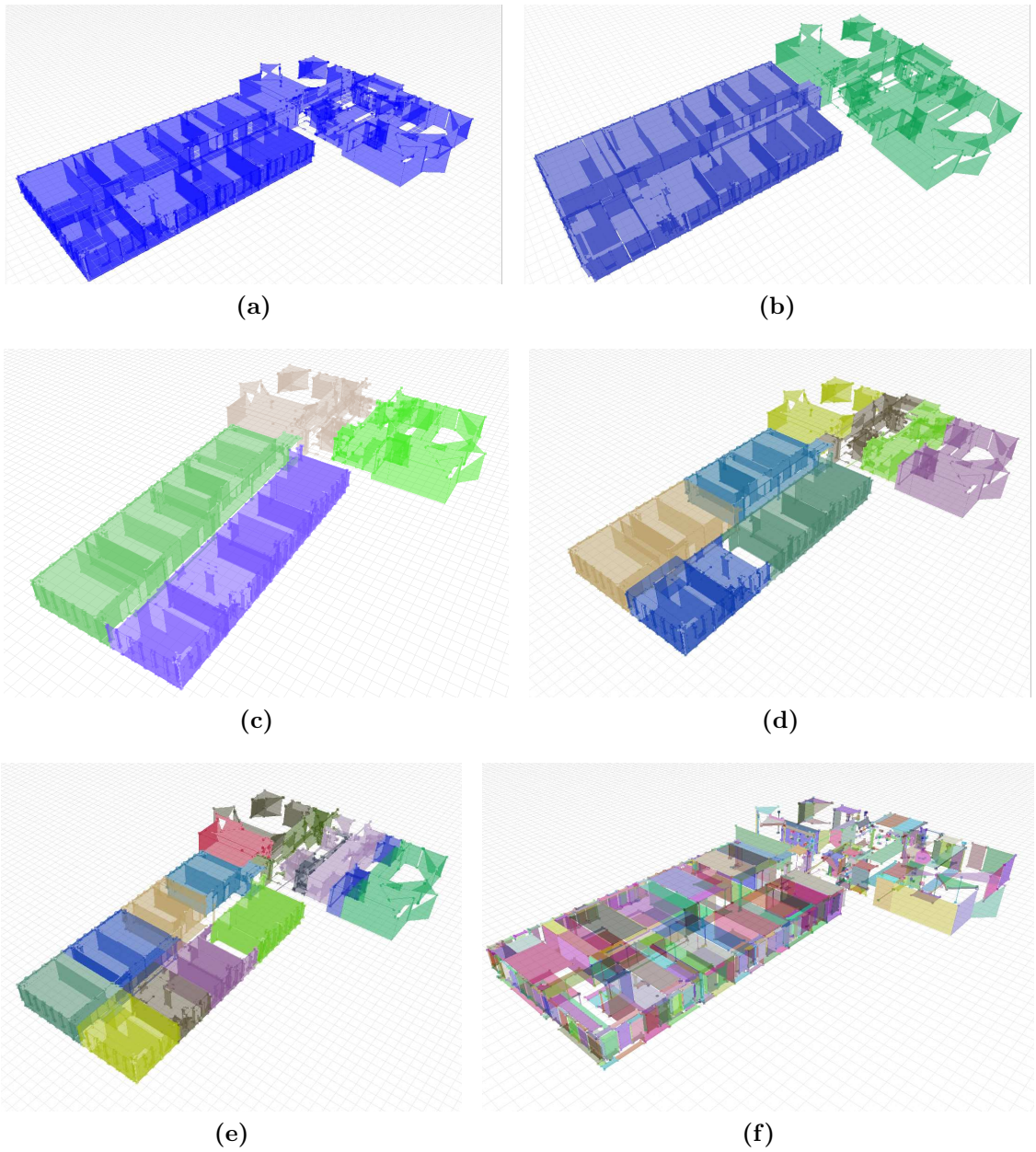


Abbildung 6.2: Darstellung der iterativen Zuordnung der Polygone der dreidimensionalen Umgebungskarte zu den Knoten des BSP-Baums. (a): Dem Wurzelknoten des BSP-Baums werden alle Polygone zugeordnet. (b) bis (e): Die Polygonzuordnung in den Tiefen eins bis vier des BSP-Baums. (f): Die Blätter des BSP-Baums (2110 Polygone).

Die Erzeugung des BSP-Baums beginnt mit dem Wurzelknoten, der alle Polygone des 3D-Modells enthält. Diese Polygone werden anhand einer Trennebene in zwei Kinderknoten aufgeteilt. Die einzelnen BSP-Bäume unterscheiden sich anhand der Wahl der Trennebene. Beim kD-Baum muss diese parallel zu den Achsen des globalen Koordinatensystems des 3D-Modells verlaufen. Beim „polygon aligned BSP"-Baum muss die Trennebene mit der durch eines der Polygone definierten Ebene übereinstimmen. In dieser Arbeit wird aus später erläuterten Gründen der „polygon aligned BSP"-Baum verwendet und im Folgenden mit „BSP"-Baum bezeichnet. Die Normale der Trennebene entspricht ebenfalls der des Polygons, das die Trennebene definiert. Dies ist wichtig, da die Lage der restlichen Polygone des betrachteten Knotens bzgl. der Trennebene bestimmt wird.

Es existieren drei mögliche Fälle für die Lage eines Polygons bzgl. dieser Trennebene. Das Polygon liegt entweder vor bzw. hinter der Trennebene. In dem Fall, dass die Ebene das Polygon schneidet, wird das Polygon in ein vorderes und hinteres Teilpolygon getrennt. Alle Polygone, die vor der Trennebene liegen, werden dem sogenannten „Vorderen Kindknoten" zugeordnet und die restlichen Polygone dementsprechend dem „Hinteren Kindknoten".

Zur Bestimmung der optimalen Trennebene werden zwei Kriterien herangezogen: Balance und Trennfaktor. Die Balance beschreibt die Differenz zwischen der Anzahl der den beiden Kindknoten zugeordneten Polygonen. Sie ist somit optimal, wenn den beiden Kindknoten gleich viele Polygone zugeordnet werden. Der Trennfaktor beschreibt, wie viele Polygone durch die Trennebene in zwei Teile getrennt werden. Er ist umso besser, je weniger Polygone getrennt werden. Die möglichen Trennebenen werden nach folgender Gleichung bewertet:

$$\text{Trennfehler} = k * \text{Trennfaktor} + (1 - k) * \text{Balance}.$$

Der Faktor k ermöglicht eine Gewichtung des Trennfaktors bzw. der Balance. Die richtige Wahl des Gewichtungsfaktors ist entscheidend für die Effizienz der Datenstruktur. Ein unbalancierter Baum führt zu einer größeren Tiefe und somit zu Effizienzverlust. Ein Baum mit einem großen Trennfaktor führt dazu, dass mehr Polygone und somit mehr Knoten erzeugt werden. Das Verhältnis von Trennfaktor und Balance muss also immer an das jeweilige 3D-Modell angepasst werden. Um die optimale Trennebene zu bestimmen, wird iterativ für jedes Polygon des aktuellen Knotens der Trennfehler berechnet. Die optimale Trennebene eines Knotens ist diejenige mit dem geringsten Trennfehler.

Die Erstellung ist also ein iterativer Prozess aus Finden der optimalen Trennebene, Aufteilen der Polygone auf die Kindknoten und Wiederholen des Prozesses für die beiden Kindknoten. Dies wird solange wiederholt, bis die jeweiligen Kindknoten nur noch ein Polygon enthalten. Diese Knoten werden Blätter des Baumes genannt.

Abbildung 6.2 zeigt die iterative Aufteilung der Polygone eines 3D-Modells. In (a) ist der Wurzelknoten zu sehen, dem alle Polygone zugeordnet sind. In (b) sind der vordere (grün) und der hintere Kindknoten (blau) zu sehen. Hier wird der Einfluss der Balance auf die optimale Trennebene deutlich. Da die rechte Seite des 3D-Modells mehr Polygone enthält als die linke, liegt die Trennebene nicht mittig im Modell sondern verschiebt sich nach rechts. (c) - (e) zeigen die Polygonzuordnung im BSP-Baum in der vier, acht bzw. 16 Kindknoten in

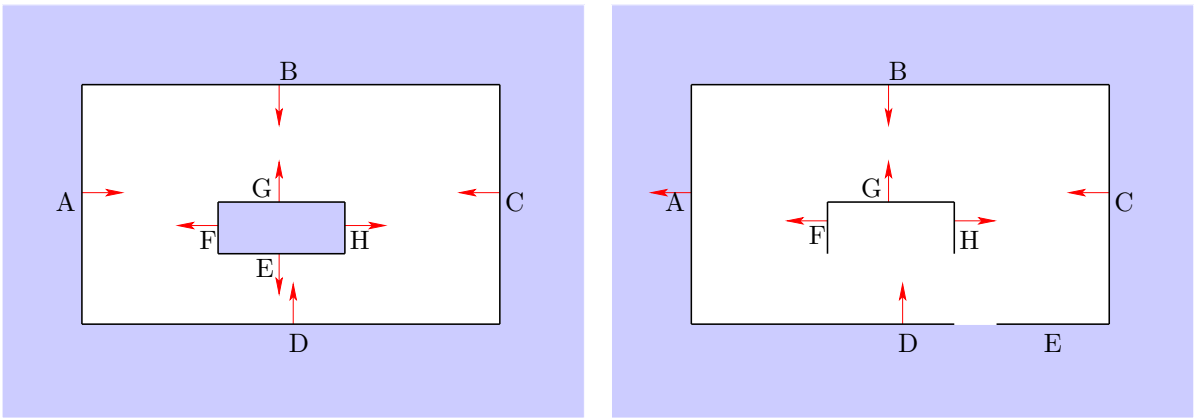


Abbildung 6.3: Vergleich legaler bzw. illegaler Modelle zur Erzeugung eines Solid Leaf BSP-Baums:(a) Eine legale Geometrie, die sich zur Erstellung des Solid Leaf BSP-Baums eignet. Es gibt einen definierten soliden Bereich (blau) sowie einen leeren Bereich (weiß). Die Normalen der einzelnen Wände (rot) zeigen in den freien Bereich. (b) Eine illegale Geometrie, da zwischen den Polygonen D und E eine Verbindung zwischen Außen- und Innenbereich besteht, die Normale des Polygons A in den soliden Raum zeigt und die Rückseiten der Polygone F, G und H direkt sichtbar sind.

den folgenden Tiefenebenen des BSP-Baums. Teilabbildung (f) zeigt alle Blätter des BSP-Baums, die jeweils nur noch ein Polygon enthalten. Es ist zu sehen, dass einige Polygone des 3D-Modells durch die Trennebenen in Teilpolygone aufgeteilt wurden.

Solid Leaf BSP-Baum

In dieser Arbeit wird eine Variante des BSP-Baums, der Solid Leaf BSP-Baum, eingesetzt. Dieser hat an die verwendeten Polygone die Anforderung, dass sie ein solides Gebilde mit definiertem Innen- und Außenbereich bilden. Bei der Arbeitsumgebung im LiSA-Szenario ist dies, wie bei vielen anderen Robotikanwendungen in Gebäuden, gegeben. Beim Solid Leaf BSP-Baum darf das 3D-Modell keine Verbindungen zwischen Innen- und Außenbereich besitzen. Stellt das 3D-Modell ein Gebäude dar, sind diese Verbindungen beispielsweise Fenster, die bei der Erstellung des 3D-Modells durch extra Polygone geschlossen werden müssen. Der Innenbereich, in dem sich der Roboter aufhalten kann, wird im Folgenden als „leerer“ Raum, und der nicht erreichbare Außenbereich als „solider“ Raum bezeichnet. Eine weitere Anforderung betrifft die Polygone des 3D-Modells. Alle Polygone besitzen eine definierte Vorder- bzw. Rückseite. Die Normale des Polygons muss zur Vorderseite zeigen. Die Vorderseite jedes Polygons muss an den leeren Raum und die Rückseite jedes Polygons an den soliden Raum grenzen. Dies ist gleichbedeutend mit der Aussage, dass keine Rückseite eines Polygons direkt aus dem leeren Bereich gesehen werden darf. Abbildung 6.3a zeigt ein Beispiel für eine legale Geometrie. Es gibt einen definierten soliden Bereich (blau) sowie einen leeren Bereich (weiß). Die Normalen der einzelnen Polygone (rot) zeigen in den freien Bereich und keine

Polygonrückseite ist aus dem leeren Bereich zu sehen. In 6.3b ist ein illegales Modell zu sehen, aus dem sich kein Solid Leaf BSP-Baum erzeugen lässt. Dies ist der Fall, da zwischen den Polygonen D und E eine Verbindung zwischen solidem und leerem Raum besteht, die Normale des Polygons A in den soliden Raum zeigt und die Rückseiten der Polygone F, G und H aus dem leeren Raum direkt sichtbar sind.

Die Erstellung des Solid Leaf BSP-Baums verläuft analog zu der bereits beschriebenen Erstellung des BSP-Baums. Die Blätter des Solid Leaf BSP-Baums besitzen die Eigenschaft, dass sie den leeren vom soliden Raum trennen, da die jeweiligen Blätter jeweils nur ein Polygon enthalten. Die Blätter der Solid Leaf BSP-Baums erhalten zwei Kindknoten, die diese Eigenschaft in den Baum integrieren. Der vordere Kindknoten wird als leer bezeichnet, da diese Seite des im Blatt enthaltenen Polygons in den leeren Raum zeigt. Der hintere Kindknoten des Blattes wird dementsprechend als solide markiert. In D.1 wird detailliert beschrieben, wie aus der in Abbildung 6.3 dargestellten legalen Geometrie ein Solid Leaf BSP-Baum erstellt wird.

Im Folgenden wird beschrieben, warum sich der Mehraufwand, der zur Erstellung des 3D-Modells des Solid Leaf BSP-Baums benötigt wird, rentiert. Es wird erläutert, welche Vorteile der Solid Leaf BSP-Baum gegenüber dem generellen BSP-Baum besitzt.

Effizientes Lösen des „Punkt im soliden Raum“-Problems

Das „Punkt im soliden Raum“-Problem beschreibt die Frage, ob ein gegebener Punkt im Innen- oder Außenbereich eines 3D-Modells liegt. Dieses Problem besitzt im Bezug zur MCL einen entscheidenden Einfluss. Da der Roboter sich nur im leeren Raum befinden kann, ist die Wahrscheinlichkeit für ein Partikel, das im soliden Raum liegt, gleich Null. Dadurch wird der Vergleich der Sensordaten mit dem 3D-Modell für dieses Partikel gespart. Des Weiteren findet das Partikel im Resampling-Prozess ebenfalls keine Berücksichtigung.

Die Solid Leaf BSP-Baum Datenstruktur eignet sich, um das „Punkt im soliden Raum“-Problem sehr effizient zu lösen. Hierzu ist lediglich eine einfache Traversierung des Baums notwendig, die beim Wurzelknoten beginnt. In jedem Knoten wird berechnet, ob der betrachtete Punkt vor bzw. hinter der Trennebene des Knotens liegt. Liegt der Punkt vor der Trennebene, wird die Prozedur im vorderen, andernfalls im hinteren Kindknoten wiederholt. Die Traversierung des Baums ist beendet, wenn der Kindknoten eines Blattes des Baums erreicht ist. Ist dieser Knoten als solide markiert, befindet sich der Punkt im soliden Raum, andernfalls bei einem als leer markierten Knoten im leeren Raum. Um zu bestimmen, ob ein Punkt innerhalb eines 3D-Modells liegt, müssen also lediglich $O(\log(N))$ Punkt-Ebenen Vergleiche berechnet werden, wobei N der Anzahl der Polygone des Modells entspricht. Ein einzelner Vergleich lässt sich über die Hessesche Normalform berechnen. Für jeden Punkt $\vec{r}$ der Trennebene gilt:

$$0 = \vec{n}_0 \cdot \vec{r} - d.$$

Wobei $\vec{n}_0$ die Normale der Ebene und d deren Abstand vom Ursprung des Koordinatensys-

tems ist. Für einen beliebigen Punkt $\vec{p}$ ergibt sich der Abstand s durch:

$$s = \vec{n}_0 \cdot \vec{p} - d.$$

Ein einfacher Vergleich mit Null der ermittelten Distanz zur Trennebene gibt die Entscheidung für den nächsten Knoten:

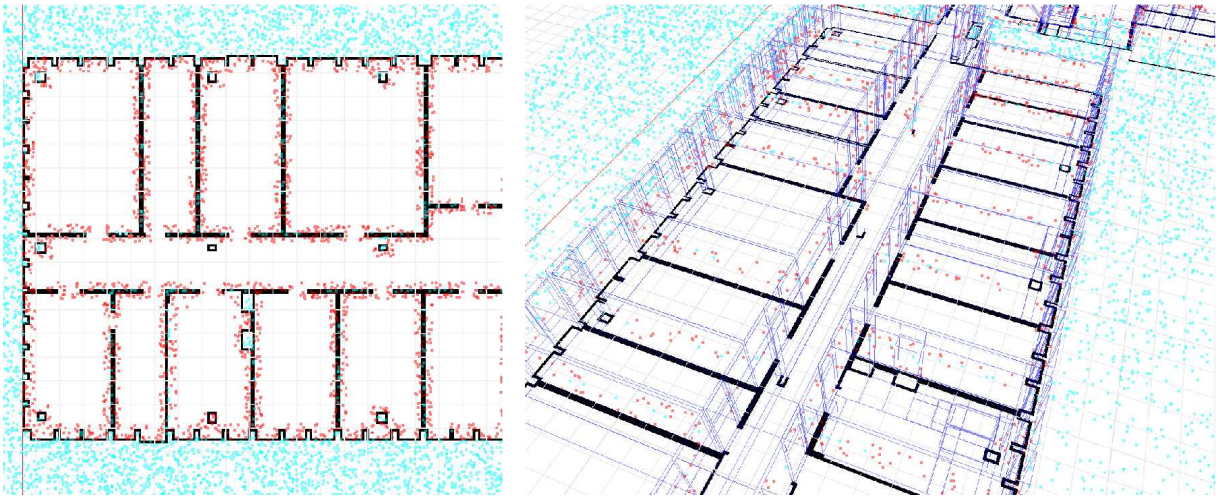
$$s \begin{cases} s < 0\text{: gehe zum hinteren Kindkonten.} \\ s > 0\text{: gehe zum vorderen Kindkonten.} \\ s = 0\text{: traversiere beide Knoten} \end{cases} \tag{6.3}$$

Dieser Vergleich lässt ebenfalls eine leichte Integration der Robotergeometrie zur Bestimmung gültiger Partikelposen zu. Da ein Roboter nicht punktförmig ist, sind nicht alle Punkte im leeren Raum tatsächlich mögliche Roboterpositionen. Die Positionen, die dichter als die halbe Roboterbreite δ an der Umgebungsgeometrie liegen, können ebenfalls ausgeschlossen werden. Dies wird realisiert, indem beim Vergleich der Distanz einer Position mit der Trennebene in Gleichung 6.3 bereits zum hinteren Kindknoten weitergegangen wird, wenn die Distanz negativ oder kleiner als δ ist.

$$s \begin{cases} s > \delta & \text{: gehe zum vorderem Kindkonten.} \\ s < 0 & \text{: gehe zum hinteren Kindkonten.} \\ s \geq 0 \leq \delta & \text{: traversiere beide Knoten} \end{cases}$$

Abbildung 6.4 zeigt das Ergebnis eines Experiments, das das bereits in Abbildung 6.2 dargestellte Umgebungsmodell mit 2110 Polygonen verwendet. In der um fünf Meter erweiterten umschließenden Hülle des Modells werden in einer Höhe von $1,5$ bis $1,6m$ 150000 Partikel zufällig verteilt. Für jedes dieser Partikel wird bestimmt, ob es sich im soliden Raum befindet. Ist dies der Fall, werden sie blau markiert. Ferner wird in dem Experiment eine Roboterbreite von $\delta = 30\,\text{cm}$ angenommen. Alle Partikel, die aufgrund des Vergleichs mit δ als im soliden Raum liegend ermittelt werden, sind rot markiert. Alle Partikel, die im leeren Raum liegen, werden nicht in die Grafik eingezeichnet. Es ist zu sehen, dass alle im Außenbereich des Modells liegenden Partikel als solide erkannt werden. Ferner werden Partikel in abgeschlossenen Bereichen, wie z.B. Pfeilern oder Schränken, als solide erkannt. Darüber hinaus werden Partikel, die direkt auf einer Wand liegen, als solide markiert. Wände werden in einer legalen Solid Leaf BSP-Baum Geometrie durch zwei Polygone erstellt, da die Rückseite eines Polygons aus dem leeren Raum nie direkt gesehen werden darf. Die als solide erkannten Partikel liegen zwischen diesen beiden Polygonen und somit im soliden Raum. Die roten Partikel in Abbildung 6.4 zeigen deutlich, dass der Vergleich mit δ dafür sorgt, dass alle Partikel, die sich im leeren Raum befinden und dichter als δ an einer Wand liegen, ebenfalls als solide markiert werden.

Mit der in dieser Arbeit entwickelten Implementierung benötigt die Klassifizierung von 1.000.000 Partikeln in dem Umgebungsmodell mit 2110 Polygonen auf einem Standard Laptop mit $2 \times 2,0\,\text{GHz}$ und 2 GB RAM nur $0,3$ Sekunden. Dies entspricht einer Klassifizierung von $3,3$ Millionen Partikeln pro Sekunde.



(a) (b)

Abbildung 6.4: Klassifizierung von möglichen Positionen des Roboters im 3D-Modell: (a) Aufsicht auf das 3D-Modell. Zufällig erzeugte Positionen im 3D-Raum werden danach klassifiziert, ob sie im soliden Raum liegen (blau), oder im leeren Raum. Positionen, die im leeren Raum liegen, sich aber zu dicht an dem 3D-Modell befinden, um eine mögliche Roboterposition zu sein, werden identifiziert und sind rot markiert. (b) 3D-Ansicht der klassifizierten Positionen im 3D-Modell (blau)

Raytracing im Solid Leaf BSP-Baum

Das Ziel des Raytracings ist es, das erste Polygon des 3D-Modells zu finden, welches von einem Strahl mit einem definierten Ursprung und einer definierten Richtung geschnitten wird. Hier wird der Vorteil des Solid Leaf BSP-Baums deutlich, da hierzu in der auf dieses Problem optimierten Baumstruktur eine einfache Vorwärtstraversierung des Baums den Schnittpunkt mit dem ersten Polygon liefert.

Der Strahl, der gegen den BSP-Baum getestet wird, besitzt einen Ursprung $\vec{p}$ und eine Richtung $\vec{d}$. Ein Punkt $\vec{r}$ auf dem Strahl lässt sich über die Größe des Parameters i definieren:

$$\vec{r}(i) = \vec{p} + i \cdot \vec{d}.$$

Da über das Lösen des „Punkt im soliden Raum“-Problems sichergestellt ist, dass sich das Partikel, für den das Raytracing durchgeführt wird, innerhalb des 3D-Modells befindet, kann die maximale Länge des Strahls t_{max} initial auf die Größe der Szene gesetzt werden. Dies stellt sicher, dass der Strahl das Modell auf jeden Fall schneidet. In jedem Traversierungsschritt wird die betrachtete Länge des Strahls, wie im Folgenden beschrieben, aktualisiert. Die Traversierung des Solid Leaf BSP-Baums beginnt beim Wurzelknoten. Zunächst wird bestimmt, auf welcher Seite der Trennebene sich der Ursprung des Strahls befindet. Ferner wird die Länge s des Strahls von dessen Ursprung bis zum Schnittpunkt mit der Trennebene bestimmt. Anhand dieser Werte wird nun bestimmt, welche Kindknoten traversiert werden müssen. Es können die in Abbildung 6.5 gezeigten drei Fälle auftreten. Teilabbildung (a) zeigt den Fall, dass der, für den betrachteten Knoten, relevante Bereich des Strahls (blau)

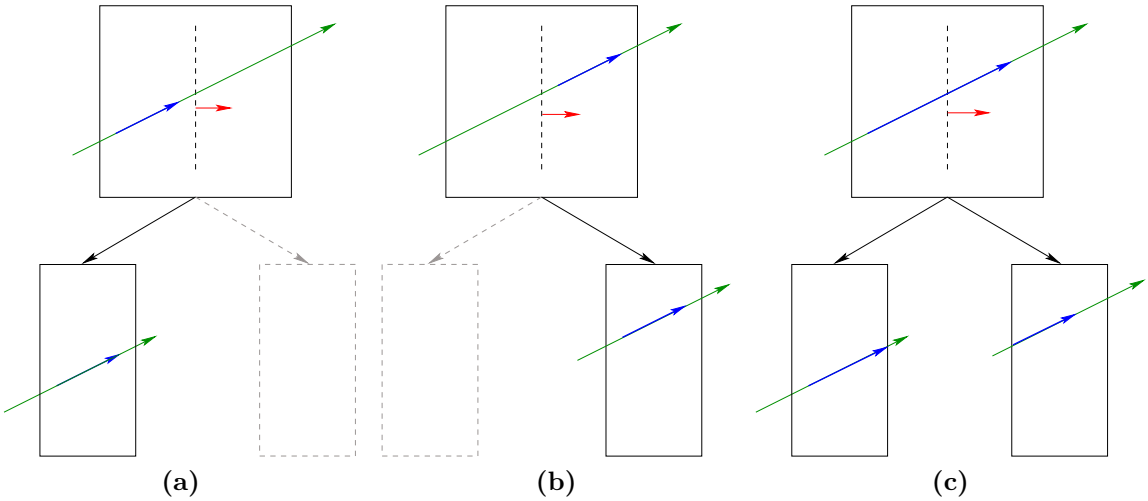


Abbildung 6.5: Die drei möglichen Konstellationen von Strahl-Trennebene beim Raytracing im Solid Leaf BSP-Baum: (b) Der relevante Bereich des Strahls (blau) liegt komplett hinter der Trennebene des Knotens. Bei der Traversierung wird ausschließlich der hintere Kindknoten betrachtet. (a) Der für den betrachteten Knoten relevante Bereich des Strahls (blau) liegt komplett vor der Trennebene des Knotens. Bei der Traversierung wird ausschließlich der vordere Kindknoten betrachtet. (c) Der relevante Bereich des Strahls (blau) schneidet die Trennebene des Knotens. Bei der Traversierung werden der vordere sowie der hintere Kindknoten betrachtet. Die Traversierung wird mit dem Kindknoten fortgesetzt, auf dessen Seite sich der Ursprung des Strahls befindet.

komplett vor der Trennebene des Knotens liegt. Bei der weiteren Traversierung wird ausschließlich der vordere Kindknoten betrachtet. In (b) ist der entgegengesetzte Fall dargestellt, in dem der relevante Bereich des Strahls komplett hinter der Trennebene des Knotens liegt. Hier wird bei der weiteren Traversierung zum hinteren Kindknoten übergegangen. Die dritte mögliche Konstellation zwischen Strahl und Trennebene ist in (c) gezeigt. Hier schneidet der relevante Bereich des Strahls die Trennebene. Um das erste Polygon zu bestimmen, das der Strahl schneidet, müssen also beide Kindknoten betrachtet werden. Hier wird zunächst der Kindknoten, auf dessen Seite sich der Ursprung des Strahls befindet, weiter traversiert, da diese Seite der Trennebene die Polygone enthält, die dichter am Ursprung des Strahls liegen. Wird bei der Traversierung des ersten Kindknotens kein Schnittpunkt mit dem 3D-Modell gefunden, wird der zweite Kindknoten traversiert. Diese drei Traversierungsregeln beachtend wird der Solid Leaf BSP-Baum traversiert, bis ein solider Kindknoten eines Blattes erreicht ist. Der Schnittpunkt des Strahls mit dem entsprechenden Blatt des Baums ist das Ergebnis des Raytracings. Der Traversierungsprozess darf erst bei einem Blatt des Baums beendet werden, da es sonst möglich ist, dass Polygone der Kindknoten den Strahl noch dichter an dessen Ursprung schneiden. In D.2 wird am Beispiel der in Abbildung 6.3 gezeigten Geometrie eine detaillierte Traversierung des Solid Leaf BSP-Baums des Modells dargestellt.

In dieser Arbeit wird eine weitere Optimierung des Raytracing-Verfahrens im Solid Leaf BSP-Baum eingesetzt, das die Anzahl der zu betrachtenden Knoten noch weiter minimiert. Dieser Ansatz versucht den in Abbildung 6.5c gezeigten Fall der Traversierung zu minimieren, da

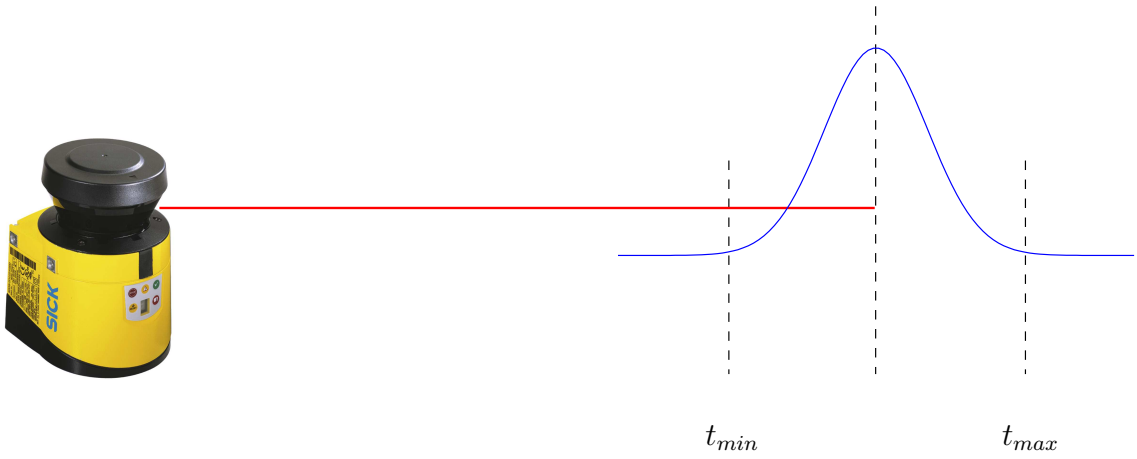


Abbildung 6.6: Begrenzung des relevanten Bereichs eines Strahls beim Raytracing: Die Wahrscheinlichkeit eines Partikels nimmt ab, wenn die Differenz zwischen Sensordaten und im Umgebungsmodell bestimmter Entfernung größer wird. Oft wird dafür eine Gaußkurve (blau) verwendet. Der relevante Bereich eines Strahls $[t_{min}, t_{max}]$ beim Raytracing kann somit in Abbhängigkeit der Varianz der Gaußkurve gesetzt werden.

hier beide Kindknoten traversiert werden müssen. Die Optimierung hat zur Voraussetzung, dass es möglich ist, nicht nur eine maximale Länge des Strahls t_{max} vorzugeben, sondern auch eine minimale Länge t_{min}. Somit wird der relevante Bereich des Strahls initial auf das Intervall zwischen t_{min} und t_{max} begrenzt, so dass nur in diesem Bereich nach Schnittpunkten zwischen dem 3D-Modell und dem Strahl gesucht wird. Eine Verkürzung des relevanten Bereichs hat zur Folge, dass bei der Traversierung die Fälle (a) und (b) in Abbildung 6.5 häufiger und die in (c) dargestellten Fälle seltener auftreten. Somit werden insgesamt weniger Knoten betrachtet und dadurch das Verfahren beschleunigt. Eine Festsetzung von t_{min} bzw. t_{max} ist im Rahmen der MCL möglich. Um die Wahrscheinlichkeit eines Partikels zu bestimmen, werden die beim Raytracing-Verfahren ermittelten Entfernungen mit realen Sensordaten verglichen. Für das Sensormodel des Laserscanners (vgl. Abschnitt 6.1.1) wird eine einfache Gaußverteilung angenommen. Wie in Abbildung 6.6 zu sehen ist, strebt die Wahrscheinlichkeit für ein Partikel ab einer gewissen Differenz zwischen Sensorabstandswert und den im Raytracing-Verfahren ermittelten Wert gegen Null. Die Werte für t_{min} bzw. t_{max} können somit in Abhängigkeit der Varianz der Gaußverteilung gewählt werden.

In Tabelle 6.1 ist ein Vergleich der Laufzeiten für verschiedene Raytracing-Varianten dargestellt. Analog zum Experiment des „Punkt im soliden Raum“-Problems werden in der umschließenden Hülle des in Abbildung 6.2 dargestellten 3D-Modells zufällig Punkte verteilt, die jeweils den Ursprung eines Strahls bilden. Für jeden Strahl wird zusätzlich eine zufällige Orientierung im 3D-Raum bestimmt. Um das Verfahren mit klassischen zweidimensionalen Verfahren vergleichen zu können, wird eine 2D-Rasterkarte als horizontaler Schnitt durch das 3D-Modell erzeugt. In dieser Rasterkarte werden zufällig 2D-Positionen für die Ursprünge der Strahlen bestimmt und jedem Strahl wird zufällig eine Orientierung zwischen 0 und 2π zugewiesen. Für jeden Strahl wird bestimmt, welches Polygon des 3D-Modells bzw. welches

Verfahren	Anzahl der Strahlen	Anzahl der Polygone	t_{min} [m]	t_{max} [m]	Zeit [s]
BSP	100000	2110	$0,0$	$1000,0$	$1,30$
BSP	100000	2110	$d-1,0$	$d+1,0$	$1,11$
2D Bresenham	100000	Rasterkarte 5 cm	$0,0$	$21,0$	$0,97$
2D Bresenham	100000	Rasterkarte 4 cm	$0,0$	$21,0$	$1,14$
2D Bresenham	100000	Rasterkarte 1 cm	$0,0$	$21,0$	$5,45$

Tabelle 6.1: Performanzanalyse des auf dem BSP-Baum basierten Raytracings im 3D-Polygonmodell im Vergleich zum Bresenham basierten Verfahren auf 2D-Rasterkarten.

belegte Raster der 2D-Rasterkarte der Strahl als erstes trifft. Auf der 2D-Rasterkarte wird hierzu ein auf dem Bresenham-Algorithmus basierter Ansatz angewendet [9].

Tabelle 6.1 zeigt, dass die auf dem BSP-Baum basierte Raytracing-Implementation in der Lage ist, Schnittpunkte zwischen 100000 Strahlen und dem 3D-Modell in nur $1,3$ Sekunden zu berechnen. Dieser Ansatz wählt $t_{min} = 0.0$ und $t_{max} = 1000.0$. Dies schränkt den relevanten Bereich des Strahls initial nicht ein. Eine wie oben beschriebene initiale Einschränkung des relevanten Bereichs auf den gemessenen Entfernungswert ergibt einen Performanzgewinn von 15%. Diese Verbesserung nimmt prozentual mit der Anzahl der Polygone zu. Mit der Performanz des Algorithmus lässt sich eine MCL mit einer Aktualisierungsfrequenz von 5 Hz, bei 900 Partikeln und 20 betrachteten Abstandswerten eines Sensordatensatzes realisieren. Diese Werte ermöglichen bei passenden Rahmenbedingungen (geringer Odometriefehler, Übereinstimmung zwischen Umgebung und 3D-Modell, nicht zu viel Dynamik in der Roboterumgebung) eine sichere Lokalisierung.

Neben der Performanz des auf dem BSP-Baum basierten Raytracings zeigt Tabelle 6.1 ebenfalls die Performanz des Bresenham-basierten Raytracings in einer 2D-Rasterkarte. Es ist zu sehen, dass die Performanz stark von der verwendeten Auflösung der Rasterkarte abhängt. Bei einer Auflösung von 4 cm pro Raster entspricht die Performanz des 2D-Verfahrens in etwa dem des auf dem BSP-Baum-basierten Verfahrens. Hier wird ein weiterer Vorteil des BSP-Baum-basierten Ansatzes gegenüber 2D-Rasterkarten deutlich. Der auf dem BSP-Baum basierte Ansatz ist in der Lage, den Schnittpunkt mit dem 3D-Modell exakt zu berechnen, während die Ungenauigkeit bei der Rasterkarte immer der halben Auflösung entspricht. Bei einer Auflösung von 1 cm benötigt der Bresenham-basierte Ansatz bereits die fünffache Zeit des BSP-Baum-basierten Ansatzes. Ein Vorteil des Bresenham-Ansatzes ist der, dass die Performanz nur mit der Auflösung, jedoch nicht mit der Größe der Karte skaliert. Ferner führt eine Karte mit vielen belegten Feldern zu einer besseren Performanz, da der Bresenham-Algorithmus früher ein belegtes Raster findet. Die Performanz des auf dem BSP-Baum-basierten Ansatzes nimmt hingegen mit steigender Anzahl der Polygone logarithmisch ab.

6.1.6 Diskussion zur MCL mit 3D-Umgebungsmodell

Da das Lokalisierungsproblem mit drei Freiheitsgraden in Kombination mit einer 2D-Umgebungsrepräsentation bereits als gelöst angesehen werden kann, stellt sich die Frage, welchen Vorteil ein 3D-Modell der Umgebung liefert. Es ist relativ einfach eine experimentelle Umgebung zu erzeugen, in der 3D-Sensordaten zu einer besseren Lokalisierung als 2D-Sensordaten führen und umgekehrt. Dies liegt daran, in wie weit man die Umgebung von der 2D- bzw. 3D-Umgebungsrepräsentation entfremdet. Aus diesem Grund soll hier auf ein solches Experiment verzichtet werden und statt dessen der generelle Vorteil von 3D-Umgebungskarten erläutert werden.

Der entscheidendste Vorteil von 3D-Umgebungsmodellen ist die Flexibilität bei der Wahl der Sensorkonfiguration des Roboters. So erfordert z.B. eine Sensorkonfiguration von zwei horizontal angebrachten Laserscannern in unterschiedlicher Höhe bei einem zweidimensionalen Ansatz zwei 2D-Umgebungsrepräsentationen in der jeweiligen Höhe. Der dreidimensionalen Ansatz kann hingegen die Sensordaten beider Sensoren mit dem selben Umgebungsmodell vergleichen. Ferner können 3D-Sensoren oder nicht horizontal angebrachte Sensoren bei einer 2D-Umgebungsrepräsentation nur über Projektionstechniken in den Lokalisierungsprozess integriert werden, wohingegen sie beim 3D-Ansatz direkt mit dem Modell verglichen werden können.

Ein weiterer Aspekt ist, dass die 3D-Repräsentation keinen Nachteil gegenüber der 2D-Rasterkartenrepräsentation besitzt, da die Rechenkomplexität vergleichbar ist. So können bestehende 2D-Ansätze ohne Nachteile auf das 3D-Modell portiert werden. Zusätztlich besitzt das 3D-Modell die potentielle Information über die Roboterpose mit allen sechs Freiheitsgraden, auch wenn das in dieser Arbeit nicht ausgenutzt wird, da es im LiSA-Szenario nicht notwendig ist. Obwohl die in dieser Arbeit verwendete MCL mit sechsdimensionalen Partikeln mit derzeitig verfügbarer Hardware nicht rechnerisch handhabbar ist, können effizientere Verfahren wie EKF zur Lokalisierung mit sechs Freiheitsgraden eingesetzt werden.

6.2 Hindernisvermeidung

Hindernisvermeidung beschreibt die zielgerichtete, lokale Bewegungsplanung des Roboters um Kollisionen mit statischen und dynamischen Hindernissen in dessen Umgebung zu vermeiden. Anhand von Sensordaten wird die optimale Bewegungsrichtung und Geschwindigkeit des Roboters ermittelt. Verschiedene Verfahren lassen sich danach beurteilen, ob sie rein geometrisch sind, die Kinematik oder die Dynamik des Systems berücksichtigen. Es gibt Ansätze, die die Hindernisvermeidung als inkrementelle Pfadplanung oder Pfaddeformierung realisieren. Diese Verfahren sind Kombinationen aus lokaler und globaler Pfadplanung, die inkrementell neue Umgebungsinformationen in die globale Pfadplanung integrieren [63, 40, 64]. Dem entgegengesetzt berechnen lokale Verfahren die optimale Kombination aus translatorischer Geschwindigkeit und Rotationsgeschwindigkeit auf Grundlage der aktuellen Sensordaten und einer bekannten Zielpose [7, 48]. Betrachtet das Verfahren zusätzlich den kinematischen Zustand des Systems, wird berücksichtigt, dass bestimmte Bewegungsrichtungen zur Hindernisvermeidung aufgrund der aktuellen Dynamik des Fahrzeugs nicht in Betracht kommen [86]. Die Dynamik betrachtet als weitere Nebenbedingungen die Trägheit und maximale Beschleunigung des Systems, so dass bestimmte Fahrgeschwindigkeiten und Orientierungsänderungen und daraus resultierende Bewegungsrichtungen ausgeschlossen sind, da das Fahrzeug nicht instantan seine Geschwindigkeit ändern kann [23, 77].

6.2.1 Multisensorfusion zur 3D-Hindernisvermeidung

In unstrukturierten, dynamischen Umgebungen reicht es nicht aus, Hindernisvermeidung auf eine horizontale Ebene zu beschränken. So wird die obligatorische Tischkante in der klassischen, zweidimensionalen Hindernisvermeidung nicht erkannt. Des Weiteren bilden Objekte wie die Sitzfläche eines Stuhls oder eine ausgezogene Schublade gefährliche, da für den Roboter in 2D unsichtbare, Hindernisse.

Es gibt grundsätzlich zwei Herangehensweisen, Hindernisvermeidung in allen drei Dimensionen zu realisieren:

Zum einen werden Verfahren verwendet, die im dreidimensionalen Raum rechnen und auch dreidimensionale Bewegungen zulassen [89]. Problematisch bei dieser Klasse von Hindernisvermeidungsalgorithmen ist die Rechenkomplexität, die durch die Verwendung aller drei Dimensionen entsteht.

Da sich die Bewegung eines autonomen mobilen Roboters oft auf die Bewegung in der Ebene beschränken lässt, existieren zum anderen Ansätze, die die dreidimensionale Information der Umgebung in einem zweidimensionalen Datensatz kodieren. Dieser virtuelle Datensatz wird als Eingabe für klassische, zweidimensionale Hindernisvermeidungsalgorithmen verwendet [95, 32]. In dieser Arbeit wird dieser Ansatz gewählt, da der LiSA-Roboter sich nur auf der Fußboden-Ebene bewegt und das Verfahren rechnerisch handhabbar wird.

Voraussetzung für beide Ansätze ist eine Sensorkonfiguration, die die gesamte umschließen-

de Hülle des Roboters abdeckt. Eine Konfiguration verschiedener Sensoren ist notwendig, da ein einzelner Sensor nicht in der Lage ist, die Umgebung des Roboters in allen drei Dimensionen über die gesamte Roboterhülle zu erfassen. Dies wird beim LiSA-Roboter durch eine Sensorkonfiguration von sechs Laserscannern erreicht (vgl. Kapitel 2.2.2). Im Folgenden wird ein Verfahren vorgestellt, das beliebig viele Abstandssensoren in einem virtuellen zweidimensionalen Datensatz fusioniert. Das Verfahren besitzt ein Hindernisgedächtnis und berücksichtigt zusätzlich die Dynamik der Umgebung des Roboters.

6.2.2 Das „Virtual Range Scans“-Konzept

Das „Virtual Range Scans“ (VRS)-Konzept kodiert Daten beliebig vieler, unterschiedlicher Abstandssensoren, die in beliebiger Pose am Roboter angebracht sind in einem virtuellen, zweidimensionalen Datensatz [80]. Zur Bestimmung der Abstandswerte des VRS wird für jeden der realen Abstandssensoren, die in dem VRS fusioniert werden, eine Rotationsmatrix spezifiziert, die die Orientierung des Sensors im Koordinatensystem des Roboters definiert. Zusammen mit der Translationsverschiebung des Sensors zum Ursprung des roboterzentrischen Koordinatensystems lassen sich über diese Matrix die Sensordaten vom Sensorkoordinatensystem auf 3D-Punkte im gemeinsamen Roboterkoordinatensystem abbilden. Dies ist die einzige Voraussetzung, die an die Sensordaten, die mit VRS-Konzept integriert werden, gestellt wird. Somit lassen sich alle Sensoren, deren Daten sich auf Punkte im dreidimensionalen Raum abbilden lassen, nahtlos mit dem VRS integrieren. Um die ermittelten 3D-Punkte in einem zweidimensionalen Datensatz kodieren zu können, werden sie auf die horizontale Ebene projiziert, auf der sich der Roboter bewegt. Diese Ebene wird ausgehend vom Ursprung des roboterzentrischen Koordinatensystems in polare Segmente aufgeteilt, deren Winkel der Auflösung des VRS entspricht. Nun werden für jedes Segment alle Hindernispunkte betrachtet, die in dieses Segment projiziert werden. Der Abstandswert des virtuellen Laserscanners für das polare Segment bestimmt sich durch den minimalen, euklidischen Abstand aller Hindernispunkte vom Ursprung. Es wird der minimale Abstand aller Hindernispunkte verwendet, damit immer das dichteste und somit das Hindernis mit dem größten Gefahrenpotential in den Daten des VRS vorhanden ist. Ein Problem vieler Sensorkonfigurationen bei der Hindernisvermeidung ist, dass Hindernisse oft nur kurzzeitig in den Sensordaten auftauchen, da viele Abstandssensoren wie Laserscanner nur in einer zweidimensionalen Ebene messen. Es existieren nur wenige Sensoren, die permanent Abstandswerte aus einem 3D-Ausschnitt der Umgebung des Roboters liefern (vgl. Kapitel 6.2.4). Abbildung 6.7 zeigt am Beispiel der Sensorkonfiguration des LiSA-Roboters (vgl. Kapitel 2.2.2), dass dieses Problem dafür sorgt, dass es beim VRS zum „Mitwandern“ von Hindernissen wie Tischplatten oder Schubladen kommt. Die Ursache des „Mitwanderns“ liegt darin, dass die Entfernungen der projizierten Hindernispunkte nur von der Höhe des Hindernisses abhängen. Wie in der Abbildung zu sehen ist, ändert sich die Entfernung nicht, wenn die Laserstrahlen des schräg montierten Laserscanners die Tischplatte am Rand oder mittig treffen. Schmale Hindernisse, die nur kurzzeitig in den Hokuyo-Laserdaten auftauchen, werden sogar „vergessen“. Ein möglicher Lösungsansatz ist, sofort auszuweichen, sobald ein Hindernis in den Daten

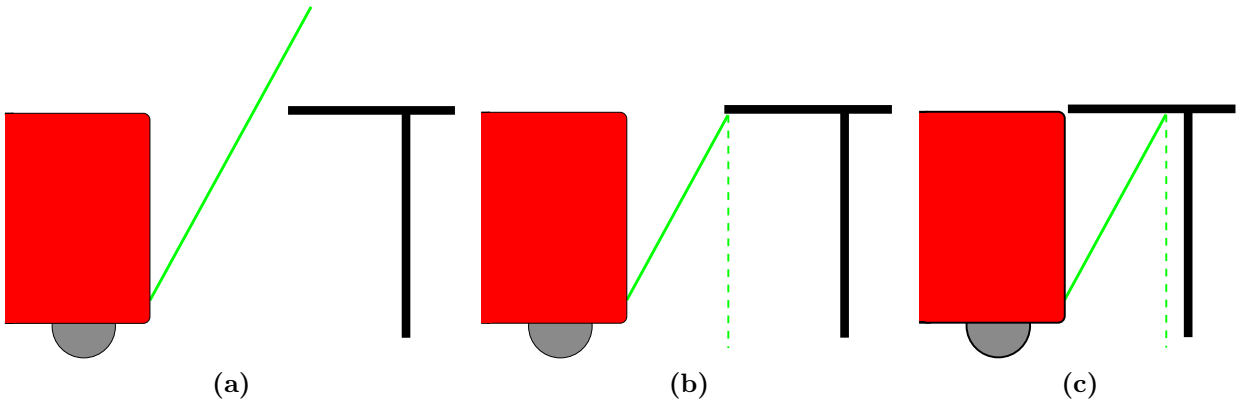


Abbildung 6.7: Problem der mitwandernden Tischplatte im VRS ohne „Gedächtnis“: (a) Der schräge Laserscanner schaut über die Tischplatte. (b) Die Strahlen des Laserscanners treffen die Tischkante und die entsprechenden Hindernispunkte werden auf die Ebene projiziert. (c) Der Roboter steht dichter am Tisch, so dass die Laserstrahlen den Tisch mittiger treffen. Dennoch ändert sich die projizierte Entfernung der Hindernispunkte nicht.

der Hokuyo-Laserscanner auftaucht. Diese Herangehensweise stellt zwei Anforderungen an die Laserdaten. Zum einen müssen die Hindernisse möglichst früh anzeigt werden, damit der Roboter rechtzeitig ausweichen bzw. abbremsen kann. Dies setzt einen relativ großen Winkel zwischen der Scanebene der Hokuyo-Laserscanner und dem Roboter voraus. Die zweite Anforderung verlangt, dass selbst in einem engen Arbeitsumfeld die Hindernisse in den Hokuyo-Laserdaten vorhanden sind. Dazu wird ein relativ steiler Winkel der Laserscanner benötigt und steht somit im Konflikt zur ersten Anforderung. Das Problem ließe sich auf Hardwareebene lösen, indem der Neigungswinkel der Hokuyo-Laserscanner dynamisch an die jeweilige Umgebungssituation und Geschwindigkeit angepasst werden kann. Dieser Ansatz kann jedoch nicht das Problem der schmalen, nur kurzzeitig in den Laserdaten vorhandenen Hindernisse zuverlässig lösen, da selbst sofortiges Ausweichen aufgrund der Trägheit des Roboters dazu führt, dass das Hindernis nicht mehr zu sehen ist und der Roboter seine Fahrt in Richtung des Hindernisses fortsetzt.

Aus diesem Grund wird in dieser Arbeit ein softwarebasierter Ansatz verfolgt. Die Idee ist, VRS mit einem „Gedächtnis“ auszustatten, das sich merkt, wo sich Hindernisse im Arbeitsumfeld des Roboters befinden. Im Fall der mitwandernden Tischplatte bedeutet das, dass VRS sich merkt, wo die Tischplatte begonnen hat und anhand der aktuellen Pose und der projizierten Tischkante die virtuellen Abstandsdaten berechnet. Daraus folgt, dass die Berechnung des VRS von einem lokalen Roboterkoordinatensystem in ein globales Koordinatensystem wechselt. Realisiert wird das Hindernisgedächtnis durch eine zweidimensionale Rasterkarte, in die die projizierten 3D-Hindernispunkte anhand der aktuellen Roboterpose eingetragen werden. Für horizontal am Roboter angebrachte Abstandssensoren bzw. Sensoren mit einer permanenten 3D-Erfassung ist es im Allgemeinen nicht notwendig, sich Hindernisse zu merken, die in diesen Sensordaten auftauchen. Aus diesem Gund können 3D-Punkte entweder direkt oder über die Hinderniskarte in VRS integriert werden.

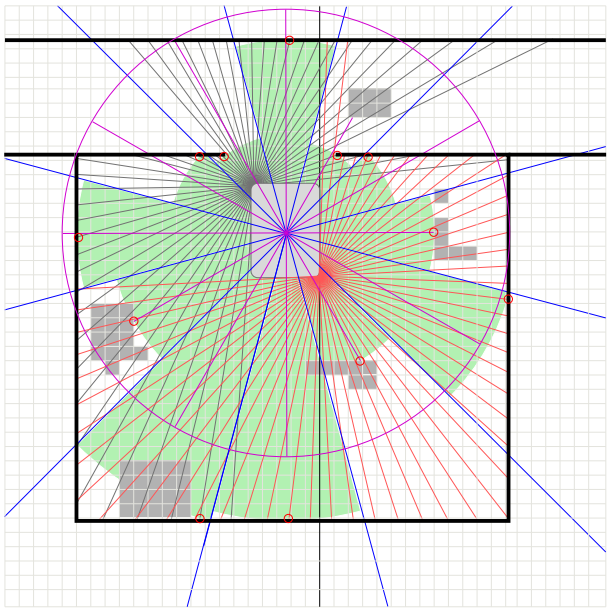


Abbildung 6.8: VRS mit Rasterkarte als Hindernisgedächtnis: Mit Hilfe eines Raytracing-Verfahrens (magenta) wird bestimmt, ob in der Rasterkarte belegte Bereiche (grau) eine geringere Entfernung besitzen als die direkt integrierten Abstandssensoren (rot und schwarz). Ist dies der Fall, wird die dem jeweiligen polaren Segment (blau) zugeordnete Entfernung (grün) auf den durch das Raytracing-Verfahren ermittelten Wert gesetzt.

Um VRS mit integriertem Hindernisgedächtnis zu erstellen, werden zunächst alle Hindernispunkte der direkt integrierten Abstandssensoren auf die polaren Segmente abgebildet. Zusätzlich wird, wie in Abbildung 6.8 zu sehen ist, für jedes polare Segment (blau) ein Raytracing-Verfahren (magenta) auf der Rasterkarte angewendet. Dieses bestimmt mit Hilfe des Bresenham-Algorithmus [9], ausgehend von der aktuellen Roboterpose, die Entfernung des ersten belegten Rasters für das jeweilige polare Segment. Dieses Raytracing-Verfahren wird mit der Frequenz eines der integrierten Sensoren durchgeführt. Dies führt z.B. bei einem Laserscanner mit einer Frequenz von 75 Hz und einer VRS-Auflösung von 1° zu $360 * 75 = 27000$ einzelnen Schnittpunktsberechnungen pro Sekunde. Um das Verfahren zu beschleunigen, wird das Raytracing-Verfahren nur bis zu einer definierten, maximalen Entfernung durchgeführt (magentafarbener Kreis). Der durch das Raytracing ermittelte Abstand wird neben allen direkt integrierten Hindernispunkten ebenfalls zur Bestimmung des Hindernisses mit dem minimalen Abstand zum Roboter herangezogen. Da die Hinderniskarte die Information über Hindernisse im 3D-Umfeld des Roboters besitzt, führt dieses Verfahren dazu, dass in dem zweidimensionalen VRS-Datensatz die 3D-Information über Hindernisse im Umfeld des Roboters enthalten ist. Abbildung 6.9 verdeutlicht den Unterschied zwischen 2D- und 3D-VRS-Umgebungswahrnehmung. In der linken Abbildung werden nur zwei horizontale Abstandssensoren, die in derselben zweidimensionalen Ebene scannen, mit VRS integriert. Da diese Sensoren auf einer Höhe von 10 cm über dem Boden scannen, sind ausschließlich die Wände und Tischbeine im VRS-Datensatz vorhanden. In der rechten Abbil-

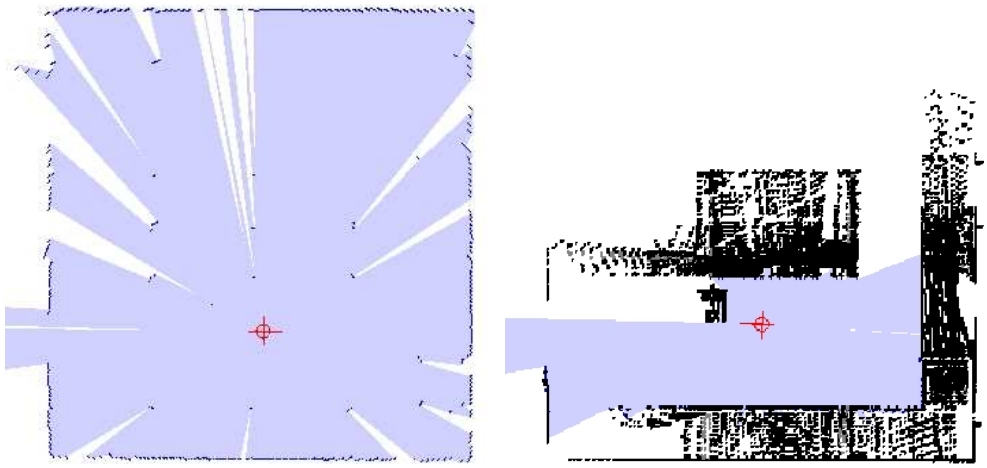


Abbildung 6.9: Unterschied zwischen 2D- und 3D-Umgebungswahrnehmung durch VRS. Der VRS-Datensatz ist die blaue Fläche und markiert den berechneten Freiraum. Links: Der VRS-Datensatz wird ausschließlich durch direkte Integration der horizontalen SICK-Laserscanner der LiSA-Sensorkonfiguration gebildet. Rechts: Der VRS-Datensatz integriert zusätzlich die durch die vier Hokuyo-Laserscanner aufgebaute Hinderniskarte.

dung wird im selben Datensatz neben den beiden direkt integrierten Abstandssensoren eine Hinderniskarte, die von den vier Hokuyo-Laserscannern des LiSA-Roboters aufgebaut wird, integriert. Der VRS-Datensatz enthält alle Hindernisse, wie Tischplatten oder die Sitzfläche eines Stuhls, im 3D-Umfeld des Roboters.

Filtern der 3D-Hindernispunkte

Das Verwenden einer globalen Hinderniskarte führt zu verschiedenen neuen Problemen, die berücksichtigt werden müssen. Das schwerwiegendste Problem ist, wie mit belegten Rastern der Rasterkarte umgegangen wird und wann diese wieder freigegeben werden. Werden belegte Raster für immer als belegt angenommen, führt dies dazu, dass Rasterzellen, die durch Sensorfehlmessungen oder eine fehlerbehaftete Lokalisierung belegt werden, freie Bereiche in der Umgebung des Roboters als blockiert in den Laserdaten auftauchen. Werden belegte Raster hingegen sofort wieder freigegeben, geht die Funktion der Rasterkarte als „Gedächtnis“ verloren.

Um dieses Problem zu lösen, wird versucht die Anzahl der in die Hinderniskarte integrierten 3D-Punkte zu minimieren, indem die Relevanz der Punkte für die Hindernisvermeidung betrachtet wird. Dies wird über verschiedene, im Folgenden beschriebene Filter realisiert.

1. 3D-Hindernispunkte werden nur in die Rasterkarte übernommen, wenn sie eine geringere Entfernung zum Roboter besitzen als die Daten der direkt integrierten Abstandssensoren für das entsprechende polare Segment.

2. Es werden ausschließlich 3D-Hindernispunkte berücksichtigt, deren Entfernung zum Roboter unter einem definierten Schwellwert liegt.

3. Die Höhe eines 3D-Hindernispunktes muss geringer als die Höhe des Roboters sein, um berücksichtigt zu werden.

Der erste Filter ist unbedenklich, da das Hindernis bereits in den direkt integrierten Abstandssensoren zu sehen ist und somit nicht in die Karte übernommen werden muss. Der Filter führt am Beispiel der LiSA-Sensorkonfiguration dazu, dass die Gebäudestruktur nicht in die Hinderniskarte übernommen wird, da die Wände im Allgemeinen in den SICK-Laserdaten sowie in den Hokuyo-Laserdaten dieselbe Entfernung besitzen. Dies hat den Vorteil, dass insbesondere Orientierungsfehler der Lokalisierung nicht so schwerwiegend sind. Würden die Wände mit einer falschen Roboterorientierung als Hindernisse in die Rasterkarte eingetragen werden, würde dies zu einer schlingernden Fahrbewegung des Roboters führen, da dieser versucht, den „schrägen“ Wänden auszuweichen.

Der zweite Filter ist aufgrund der Entfernung der Hindernisse unbedenklich. Er wird verwendet, um den Fehler, den die Unsicherheit der Lokalisierung verursacht, zu minimieren. Dies ist notwendig, da Laserscanner, wie sie in der Robotik eingesetzt werden, eine Reichweite von etwa 30 m besitzen und somit ein Orientierungsfehler von 5° bereits dazu führt, dass ein erkanntes Hindernis in diesem Abstand mit einem Fehler von etwa 2,5 m in die Rasterkarte eingetragen wird. Der Entfernungsschwellwert, ab dem die Hindernispunkte ignoriert werden, kann so gewählt werden, dass die Hindernisse auf jeden Fall noch einmal in den Sensordaten auftauchen müssen, wenn der Roboter näher an das Hindernis heranfährt. Im Fall der LiSA-Sensorkonfiguration ist dieser Schwellwert die Entfernung der projizierten Roboterhöhe in den Ebenen der Hokuyo-Laserscanner. Bei einer Roboterhöhe von 1,9 m und einem eingeschlossenen Winkel zwischen Roboter und der Ebene der Hokuyo-Laserscanner von 40° ergibt sich ein Schwellwert von 1,22 m. Ein Hindernis in einer Höhe, die größer ist als die Höhe des Roboters, kann mit diesem nicht kollidieren. Dies begründet die Ungefährlichkeit des dritten Filters.

Da Experimente mit den Filtern zeigen, dass keiner dieser rein geometrischen Filter in der Lage ist, dynamische Objekte wie einen Menschen im Arbeitsbereich des Roboters herauszufiltern (vgl. Abschnitt 6.2.4), wird ein zusätzlicher semantischer Filter auf die 3D-Hindernisdaten angewendet. Dynamische Objekte sind kritisch für die Hinderniskarte, da z.B. ein um den Roboter herumlaufender Mensch diesen komplett blockieren würde. Theoretisch filtert der erste Filter einige dieser Punkte heraus, da die Beine in den direkt integrierten Sensoren vorhanden sind. Jedoch zeigen die Experimente, dass einzelne Punkte von Oberkörper oder Armen eines laufenden Menschen den Roboter trotzdem noch blockieren. Der semantische Filter bestimmt zusätzlich, welche Hindernispunkte durch dynamische Objekte hervorgerufen werden. Hierzu wird ein in Kapitel 7.2.1 beschriebenes Verfahren eingesetzt, das anhand von Entfernungssprüngen in aufeinander folgenden Datensätzen der Abstandssensoren bestimmt, wo sich dynamische Objekte befinden. Unter Verwendung der aktuellen Roboterpose werden diese Objekte mit einem Kalman-Filter-basierten Algorithmus verfolgt. Das Verfahren weist jedem gefundenen dynamischen Objekt eine Unsicher-

heitsellipse zu. Befinden sich die projizierten Hindernispunkte in dieser Ellipse, werden sie ignoriert. Dieser Filter führt dazu, dass sämtliche durch dynamische Objekte hervorgerufene 3D-Hindernispunkte nicht in die Hinderniskarte eingetragen werden. Dadurch taucht ein dynamisches Hindernis nicht im Hindernisgedächtnis auf und wird ausschließlich durch die aktuellen Sensordaten in den virtuellen Sensordatensatz aufgenommen.

Zusätzlich zu den Filtern wird ein Alterungsprozess auf der Hinderniskarte angewendet. Sobald ein Raster für eine definierte Zeit nicht mehr als belegt gemessen wurde, wird es wieder als unbelegt markiert. Dieses Konzept muss jedoch erweitert werden, da das oben beschriebene Problem der mitwandernden Tischkante wieder auftritt. Der Roboter steht vor der Tischkante und diese wird in die Rasterkarte aufgenommen. Nun fährt er auf diese zu, bis der minimale Abstand erreicht ist, den ein Hindernis besitzen darf. Dort hält er an, und die Tischkante wird durch den Alterungsprozess aus der Rasterkarte entfernt. Dadurch wandert die Tischkante weiter weg und der Roboter fährt dichter an den Tisch heran, bis er mit diesem kollidiert. Um dieses Problem zu lösen, wird der Alterungsprozess nur angewendet, wenn sich die Pose des Roboters um einen definierten Wert geändert hat.

Verwenden des VRS-Konzepts zur Selbstlokalisierung

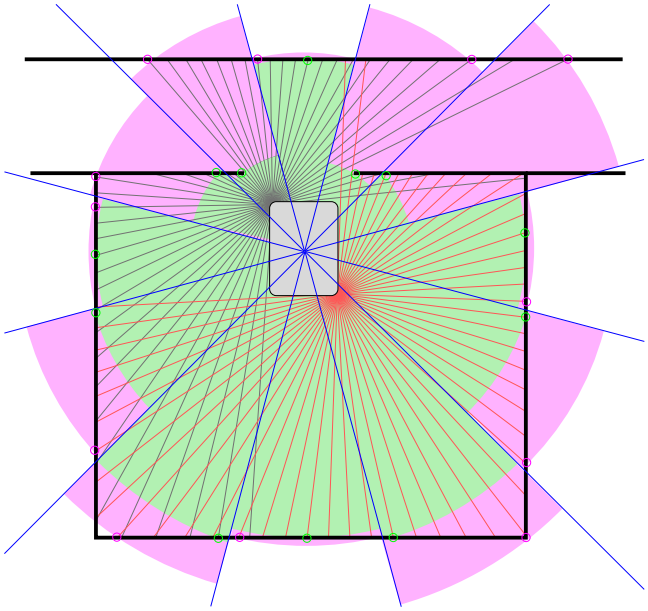


Abbildung 6.10: Verdeckungsproblem beim VRS-Konzept: Laserscannerdaten der beiden SICK-Laserscanner (rot und schwarz). Polare Segmente des VRS (blau). Minimale (grün) und maximale (magenta) Hindernispunkte für das jeweilige Segment.

Zur Lokalisierung eines Roboters, das VRS als Sensor verwendet, wäre es theoretisch sinnvoll, nicht das dichteste, sondern das am weitesten entfernte Hindernis des jeweiligen polaren Segments in den virtuellen Datensatz zu übernehmen. Dies führt dazu, dass z.B. Stuhl-

oder Tischbeine durch die dahinter liegende Wand ersetzt werden und die Gebäudegeometrie somit noch klarer abgebildet wird. Abbildung 6.10 zeigt jedoch, dass hier ein durch die Verwendung mehrerer Abstandssensoren hervorgerufenes Verdeckungsproblem besteht, sobald eine Wand, die z.B. hinter einer Tür liegt, nur in einem Datensatz der beiden Abstandssensoren vorkommt. Dies ist in der Abbildung oben rechts zu sehen. Der vordere Laserscanner (schwarz) sieht bereits die hinter der Tür liegende Wand des Flures, während diese für den hinteren Laserscanner (rot) noch durch die Wand des Raumes verdeckt ist. Würde der VRS-Datensatz nun aus den Sensordaten mit der jeweils maximalen Entfernung (magenta) erstellt werden, würde die Wand des Raumes rechts neben der Tür nicht in den Daten auftauchen, da sie durch die dahinter liegende Wand „überschrieben“ wird. Da die Monte-Carlo-Lokalisierung robust gegenüber zu kurzen Abstandswerten (im Vergleich zur Gebäudegeometrie) ist [84], wie sie durch Stuhlbeine, Menschen usw. entstehen, verwendet VRS auch im Zusammenhang der Lokalisierung immer die minimalen Entfernungen. Beim LiSA-Roboter wird zur Lokalisierung ein VRS-Datensatz verwendet, der ausschließlich die beiden SICK-Laserscanner als Eingabesensoren verwendet (vgl. Abschnitt 6.2.4). Eine zusätzliche Integration der 3D-Umgebungsinformation durch eine Hinderniskarte führt häufig dazu, dass die Gebäudestruktur nicht mehr in den VRS-Daten vorhanden ist (vgl. Abbildung 6.9).

6.2.3 Nearness Diagramm Hindernisvermeidung

In diesem Abschnitt wird das in dieser Arbeit verwendete Hindernisvermeidungsverfahren vorgestellt. Im folgenden Abschnitt 6.2.4 wird experimentell belegt, dass sich dieses Verfahren in Kombination mit dem beschriebenen VRS-Konzept eignet, um 3D-Hindernisvermeidung in unstrukturierten, dynamischen Umgebungen zu gewährleisten.

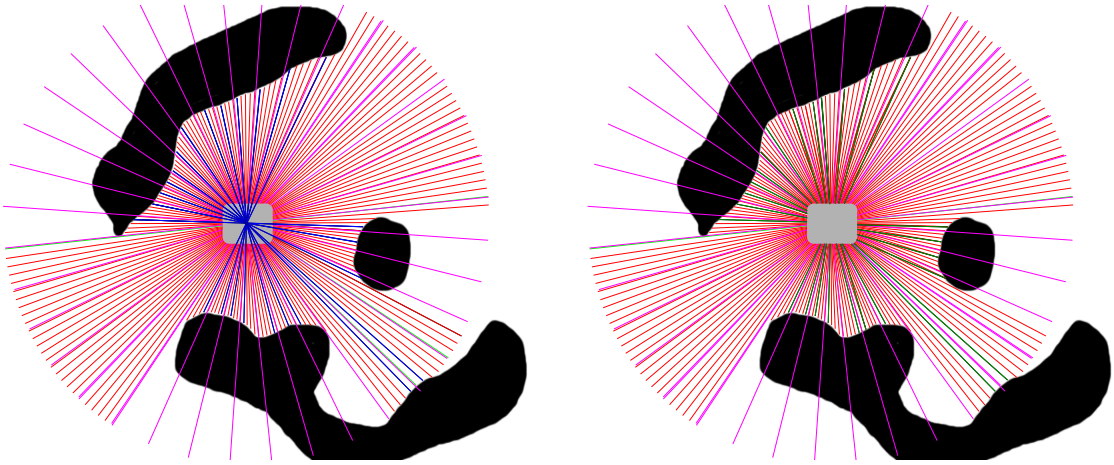
Das Nearness Diagram (ND)-Verfahren gehört zu den klassischen zweidimensionalen Hindernisvermeidungsalgorithmen und wurde 2000 von Minguez und Montano vorgestellt [48]. Es ist ein dreischrittiges Verfahren, das anhand von zweidimensionalen Entfernungsdaten zwei polare Entfernungsdiagramme erstellt. Mit Hilfe dieser Entfernungsdiagramme wird eine Analyse der aktuellen Umgebungssituation erstellt und die Fahrkommandos werden auf diese Situation abstimmt.

In jeder Iteration des ND-Algorithmus werden die folgenden drei Schritte durchgeführt:

1. Bestimmung der optimalen Bewegungsrichtung: Anhand der Sensordaten wird die Umgebung in polare Segmente unterteilt, die als blockiert bzw. befahrbar markiert werden.

2. Analyse der aktuellen Gefahrensituation: Mit Hilfe der ermittelten Segmente sowie einem der Diagramme wird die aktuelle Gefahrensituation ermittelt.

3. Berechnung der Fahrkommandos: Je nach Gefahrensituation werden die Fahrkommandos bestimmt.



(a) (b)

Abbildung 6.11: Links: Das „Central Point Nearness Diagramm“ (PND) bestimmt für jedes polare Segment (magenta) das dichteste Hindernis bzgl. des Roboterschwerpunkts (blau). Rechts: Das „Robot Nearness Diagramm“ (RND) bestimmt für alle polare Segmente (magenta) das dichteste Hindernis bzgl. der umschließenden Hülle des Roboters (grün)

Bestimmung der polaren Diagramme

Zunächst wird die horizontale Ebene, in der die Roboterbewegung stattfindet, in polare Segmente, sogenannte Sektoren, aufgeteilt. Die aktuelle Roboterpose bildet den Ursprung dieser Segmente. Anhand dieser Sektoren werden zwei Diagramme erstellt. Das „Central Point Nearness Diagramm“ (PND) ist exemplarisch in Abbildung 6.11a zu sehen. Es ermittelt für jeden Sektor das nächste Hindernis ausgehend vom Schwerpunkt b des Roboters. Abbildung 6.11b zeigt das „Robot Nearness Diagramm“ (RND). Dieses speichert in jedem Sektor den Abstand des dichtesten Hindernisses bzgl. der umschließenden Hülle des Roboters. Für eine Pose q des Roboters ermitteln sich die Werte der einzelnen Sektoren i für die beiden Diagramme nach folgenden Formeln:

$$PND_i(q,b) = \begin{cases} d_{max} + 2R - \delta_i(p) \mid \delta_i(p) > 0 \\ 0 \qquad\qquad\qquad\quad \mid \delta_i(p) = 0 \end{cases}$$

$$RND_i(q,b) = \begin{cases} d_{max} + E_i - \delta_i(p) \mid \delta_i(p) > 0 \\ 0 \qquad\qquad\qquad\quad \mid \delta_i(p) = 0 \end{cases}$$

Wobei d_{max} die maximale Reichweite des verwendeten Sensors, R der Radius des Roboters und E_i die Abstand zwischen umschließender Hülle und Schwerpunkt des Roboters für das jeweilige Segment ist. $\delta_i(p)$ ist eine funktion, die den minimalen Abstand aller Hindernispunkte des jeweiligen Segments zum Roboterschwerpunkt angibt. $\delta_i(p)$ ist Null, falls das betrachtete Segment keine Hindernispunkte enthält.

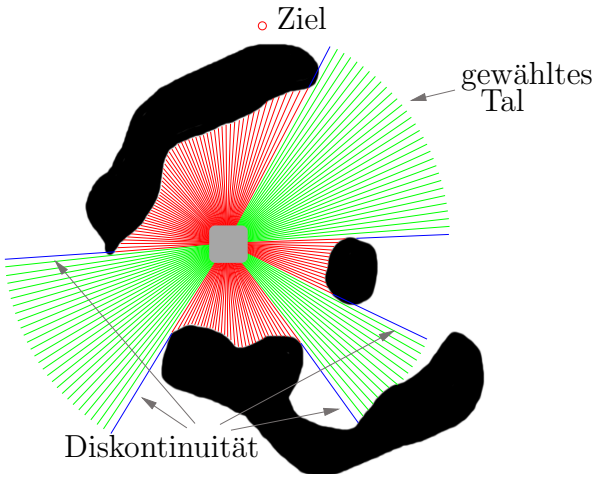


Abbildung 6.12: Aufteilung der Roboterumgebung in befahrbare (grün) und nicht befahrbare Bereiche (rot). Die Bereiche sind jeweils durch eine Diskontinuität begrenzt (blau), die eine potentiell vorhandene Lücke zwischen Hindernissen repräsentiert, die breit genug ist, um vom Roboter durchfahren zu werden.

Ermittlung der befahrbaren Bereiche

Da das PND nicht von der Form des Roboters abhängt, repräsentieren die Werte der einzelnen Sektoren die Topologie der Umgebung. Aus diesem Grund wird es verwendet, um die generelle Richtung des nächsten Fahrbefehls zu bestimmen. Dazu wird zunächst bestimmt, in welchem Sektor sich das anzufahrende Ziel befindet (s_{goal}). Anschließend wird das Diagramm über die Entfernungsdifferenz benachbarter Sektoren und unter Berücksichtigung der Breite des Roboters l in freie Regionen (Täler) und durch Hindernisse blockierte Bereiche aufgeteilt. Eine Diskontinuität liegt vor, wenn die Entfernungsdifferenz benachbarter Sektoren mindestens der Roboterbreite l entspricht. Die Täler werden auf jeder Seite durch eine Diskontinuität des PND begrenzt.

Wie in Abbildung 6.12 zu sehen ist, repräsentiert eine Diskontinuität immer eine zwar nicht direkt sichtbare jedoch potentiell vorhandene Lücke zwischen den Hindernissen, die breit genug ist, damit der Roboter hindurch passt.

Ein Tal wird als breit bezeichnet, wenn die Anzahl der Sektoren zwischen den begrenzenden Diskontinuitäten einen definierten Schwellwert s_{max} überschreitet.

Zur Bestimmung der optimalen Bewegungsrichtung wird ermittelt, welches Tal den Zielsektor s_{goal}enthält. Falls der Zielsektor durch ein Hindernis versperrt ist und nicht zu einem Tal gehört, wird das Tal, das den Sektor mit der minimalen Winkeldifferenz zum Zielsektor enthält, gewählt. Ferner muss das Tal navigierbar sein. Das bedeutet, dass das Tal breit genug ist, um von dem Roboter durchfahren zu werden.

Analyse der Gefahrensituation

Das RND-Diagramm wird verwendet, um die Sicherheit des Roboters zu ermitteln, da es den Abstand der Hindernispunkte zur Hülle des Roboters repräsentiert. Die Sicherheit des Roboters wird zunächst als "Hohe Sicherheit"(HS) bzw. "Niedrige Sicherheit" (LS) eingestuft. Hierzu wird eine Sicherheitszone um den Roboter definiert. Diese Sicherheitszone wird mit dem RND verglichen. Besitzt das RND Werte, die geringer als die Sicherheitszone für den jeweiligen Sektor sind, wird die Situation als LS eingestuft ansonsten als HS. Anhand dieser groben Einstufung und des gewählten Tales wird zwischen den folgenden sechs Sicherheitssituationen des Roboters in unterschieden [50].

HSGV: (High Safety Goal Valley) alle Werte des RND befinden sich oberhalb der Sicherheitszone. Zusätzlich befindet sich der Zielsektor im gewählten Tal.

HSWV: (High Safety Wide Valley) alle Werte des RND befinden sich oberhalb der Sicherheitszone. Der Zielsektor befindet sich nicht im gewähltem Tal und das gewählte Tal ist als breit eingestuft.

HSNV: (High Safety Narrow Valley) alle Werte des RND befinden sich oberhalb der Sicherheitszone. Der Zielsektor befindet sich nicht im gewähltem Tal und das gewählte Tal ist als schmal eingestuft.

LSGV: (Low Safety Goal Valley) die Sicherheitszone ist auf einer Seite des gewählten Tals verletzt und das Ziel befindet sich im gewähltem Tal.

LS1: (Low Safety 1) die Sicherheitszone ist auf einer Seite des gewählten Tals verletzt. Das RND besitzt an dieser Seite also Werte, die unterhalb der Sicherheitszone liegen.

LS2: (Low Safety 2) die Sicherheitszone ist auf beiden Seite des gewählten Tals verletzt.

Bestimmung der Fahrkommandos

Die Fahrkommandos werden der jeweiligen Sicherheitssituation angepasst ermittelt. Es werden die optimalen Wertepaare aus Bewegungsrichtung Θ, translatorischer Geschwindigkeit v und Rotationsgeschwindigkeit ω bestimmt.

Zunächst wird die optimale Bewegungsrichtung bestimmt, die der Roboter in der aktuellen Situation einschlagen kann. Hierzu wird je nach Sicherheitssituation ein optimaler Sektor s_Θ bestimmt. Die optimale Bewegungsrichtung ergibt sich als Winkelhalbierende des gewählten Sektors. Es existieren verschiedene Berechnungsstrategien von s_Θ in Abhängigkeit zur Sicherheitssituation.

HSGV: In der HSGV-Situation befindet sich s_{goal} im gewählten Tal und kann direkt als optimale Bewegungsrichtung gewählt werden.

HSWV: In der HSWV-Situation fährt der Roboter in sicherer Entfernung entlang eines Hindernisses. s_Θ wird so gewählt, dass die Tendenz in Richtung des Ziels geht. Die

Addition bzw. Subtraktion einer Konstanten α zur Bewegungsrichtung stellt sicher, dass der Abstand zum Hindernis konstant gehalten wird und der Roboter sich entlang des Hindernisses bewegt.

HSNV: In der HSNV-Situation ist das aktuelle Tal schmal, und die optimale Bewegungsrichtung wird in die Mitte des Tals gelegt.

LSGV: Bei den Situationen mit niedriger Sicherheit, muss der Winkel und der Abstand der Hindernisse berücksichtigt werden, die die Sicherheitszone verletzen. So ist in der LSGV-Situation die optimale Bewegungsrichtung eine Überlagerung aus der Zielrichtung s_{goal} und einem zusätzlich Term β, der die Bewegungsrichtung so anpasst, dass dem Hindernis ausgewichen wird. Das Verhältnis von s_{goal} zu β wird über den Abstand des Hindernisses gesteuert.

LS1: In der LS1-Situation befindet sich nur auf einer Seite des Roboters ein Hindernis. Die Bewegungsrichtung wird so gewählt, dass dem Hindernis ausgewichen wird und der Roboter in Richtung der Lücke fährt, die die Diskontinuität in Richtung des Zieles erzeugt.

LS2 In der LS2-Situation befindet sich auf beiden Seiten des Roboters ein Hindernis. Als Bewegungsrichtung wird der mittlere Sektor des Tals ausgewählt. Zusätzlich wird ein Korrekturterm addiert bzw. subtrahiert, um den Platz zu den dichtesten Hindernissen auf beiden Seiten zu maximieren.

Neben der Bewegungsrichtung muss ebenfalls die optimale Geschwindigkeit des Roboters bestimmt werden. In einer Situation mit hoher Sicherheit hängt die Geschwindigkeit v linear von der berechneten Bewegungsrichtungsänderung ab. Sie variiert zwischen der maximal erlaubten Geschwindigkeit v_{max} und 0 für eine senkrechte Bewegungsänderung. In einer Situation mit niedriger Sicherheit wird die Geschwindigkeit zusätzlich um das Verhältnis des Abstands des dichtesten Hindernisses zur Größe der Sicherheitszone reduziert.

Die optimale Rotationsgeschwindigkeit ω wird so gewählt, dass der Roboter sich zur gewählten Bewegungsrichtung ausrichtet. Die Größe der Rotationsgeschwindigkeit ist direkt proportional zur Änderung der Bewegungsrichtung.

$$\omega = \omega_{max} \cdot \frac{\Theta}{\frac{\pi}{2}}$$

Für die optimalen Wertepaare (Θ, v, ω) wird bei der ND-Methode eine maximale Bewegungsrichtungsänderung von $\Theta = \frac{\pi}{2}$ erlaubt. Dies lässt sich darauf zurückführen, dass die zur Entwicklung der Methode eingesetzte Hardware nur über eine Sensorkonfiguration mit einem Sichtfeld von 180° verfügt [49]. In Abschnitt 6.2.3 wird beschrieben, welche Änderungen an der ND-Methode eingeführt werden, um sie an die Geometrie und Kinematik des LiSA-Roboters anzupassen.

Eigenschaften der ND-Methode

Die ND-Methode besitzt gegenüber anderen reaktiven Hindernisvermeidungsverfahren entscheidende Vorteile. So besteht nicht die Gefahr, dass der Roboter in „U“-förmigen Hindernissen steckenbleibt. Vor allem gradientenbasierte Methoden haben mit diesen sogenannten „Trap“-Situationen Probleme: Der Roboter fährt in das „U“ hinein und es kommt zu einer oszillierenden Fahrbewegung an der Ausweichschwelle des Hindernisvermeidungsverfahrens. Die ND-Methode hingegen fährt erst gar nicht in „U“-förmige Hindernisse hinein, da diese keine Diskontinuitäten enthalten und somit nicht als eigenständiges Tal bewertet werden. Ein weiterer entscheidender Vorteil ist, dass es nur zwei Parameter gibt, die das Verhalten der Methode beeinflussen: der eine Parameter ist die Größe der Sicherheitszone, der zweite ist der Winkel, ab dem ein Tal als breit klassifiziert wird. Da es sich bei diesen Parametern um geometrische Größen handelt, sind diese leicht an die vorhandene Hardware und das gewünschte Fahrverhalten anpassbar.

Anpassung der ND-Methode an die Kinematik des LiSA-Roboters

Da der LiSA-Roboter durch die Multisensorfusion ein Sichtfeld von 360° besitzt, kann die Begrenzung der maximalen Bewegungsänderung auf $\frac{\pi}{2}$ aufgehoben werden. Da der LiSA-Roboter durch seinen omnidirektionalen Antrieb in der Lage ist, die gewünschte Bewegungsrichtung direkt einzuschlagen und in manchen Situationen nicht gewollt ist, dass der Roboter sich zur Bewegungsrichtung ausrichtet (vgl. Kapitel 7.1.3), darf die Geschwindigkeit für Bewegungsänderungen von $\frac{\pi}{2}$ nicht gegen Null streben, da der Roboter sonst stehen bleiben würde. Um die Geschwindigkeit zu berechnen, wird eine maximale Geschwindigkeit in Richtung der Roboterachse v_{trans} und eine maximale laterale Geschwindigkeit $v_{lateral}$ definiert. Die maximale Geschwindigkeit für eine Bewegungsrichtung Θ ergibt sich nach folgender Gleichung:

$$v_{max}(\Theta) = \sin(\Theta) \cdot v_{lateral} + \cos(\Theta) \cdot v_{trans}.$$

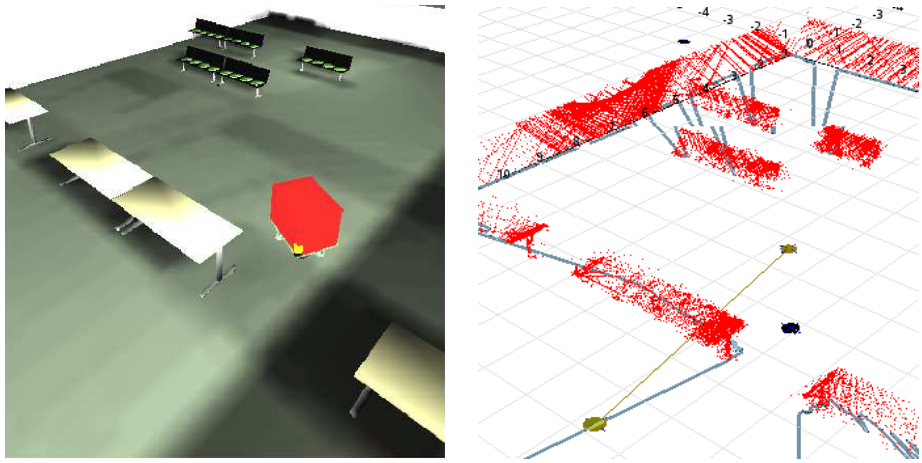
Diese Geschwindigkeit wird in einer Situation mit hoher Sicherheit angenommen. In einer Situation mit geringer Sicherheit wird die Geschwindigkeit proportional zum Verhältnis zwischen der Entfernung des dichtesten Hindernisses d_{obs} und der Größe der Sicherheitszone d_s reduziert.

$$v(\Theta) = v_{max}(\Theta) \cdot \frac{d_{obs}}{d_s}$$

Ferner wird in dieser Arbeit der statische Entfernungsschwellwert zur Einschätzung der Sicherheitslage entfernt. Er wird durch einen adaptiven Schwellwert ersetzt, der von der aktuellen Geschwindigkeit des Roboters abhängt.

$$d_s(v) = d_{s_{min}} + (d_{s_{max}} - d_{s_{min}}) \cdot \frac{v}{v_{max}}$$

Hier bezeichnet $d_{s_{min}}$ den Sicherheitsabstand der unabhängig von der Geschwindigkeit eingehalten werden muss, damit eine Situation als sicher eingestuft wird. $d_{s_{max}}$ bildet die Grenze zwischen hoher und geringer Sicherheit bei maximaler Geschwindigkeit v_{max}.



(a) (b)

Abbildung 6.13: Experimentelle Validierung der Hindernisvermeidung in der Simulationsumgebung: (a) Der LiSA-Roboter in der Simulationsumgebung. (b) Sensordatenvisualisierung: Die Verbindungslinie der beiden Kreise stellt den ursprünglich geplanten Pfad dar. Der schwarze Kreis markiert die aktuelle Roboterposition.

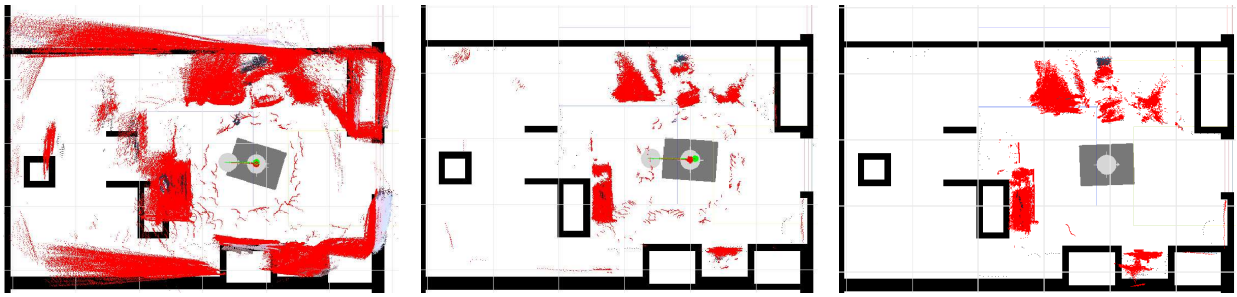
6.2.4 Experimentelle Validierung der Hindernisvermeidung

VRS in der Simulationsumgebung

Abbildung 6.13 zeigt ein Experiment in der Simulationsumgebung USARSim, das VRS mit der Vector-Field-Histogram-Methode kombiniert. Es ist zu sehen, dass die Tische in den Hindernisdaten vorhanden sind und in den VRS-Datensatz übernommen werden. Da der gerade geplante Pfad (vgl. Kapitel 6.3) blockiert ist, weicht der Roboter aus und erreicht die Zielpose durch den freien Bereich zwischen den Tischen. Die Bänke im Hintergrund sind zwar in den Hindernisdaten vorhanden, werden aber wegen der maximalen Entfernung beim Raytracing-Verfahren nicht in den VRS-Datensatz integriert. In diesem Experiment werden alle 3D-Datenpunkte der vier Hokuyo-Laserscanner als Hindernispunkte angenommen und in die lokale Hinderniskarte integriert. Wie das folgende Experiment zum Filtern der Datenpunkte zeigt, ist dies für eine reale Umgebung nicht geeignet.

Untersuchung der VRS-Integrationsfilter

In Abbildung 6.14 sind Aufsichten auf verschieden gefilterte Punktwolken zu sehen, die vom LiSA-Roboter aufgenommen wurden. Es werden die Daten der vier Hokuyo-Laserscanner sowie der PMD-Kamera berücksichtigt. Die jeweiligen Punktwolken werden während einer 360° Drehung in der in Abbildung 6.21a gezeigten Umgebung aufgenommen. Während dieser Drehung ist ein Mensch um den Roboter herumgelaufen. Die Aufnahme der Daten wird für jede Untersuchung (6.14a-(c))wiederholt. In Abbildung 6.14a ist der Fall aus dem Experiment in der Simulationsumgebung zu sehen, bei dem alle 3D-Datenpunkte in die Hinderniskar-



(a) (b) (c)

Abbildung 6.14: Als Hindernis klassifizierte 3D-Punkte in Abhängigkeit des verwendeten Filters: (a) Alle Punkte der Hokuyo-Laserscanner und der PMD-Kamera werden als Hindernispunkte verwendet. (b) Hindernispunkte, deren Entfernung größer gleich den Entfernungen der direkt integrierten Hindernispunkte sind, werden herausgefiltert. (c) Hindernispunkte, die zu dynamischen Objekten gehören oder einen Abstand zum Roboter von mehr als 2 m besitzen, werden herausgefiltert.

te integriert werden. Es ist zu erkennen, dass Orientierungsfehler der Lokalisierung dafür sorgen, dass Hindernispunkte von weiter entfernten Objekten falsch registriert werden. Ferner werden die durch den Menschen hervorgerufenen Hindernispunkte in die Hinderniskarte aufgenommen. Dies sorgt dafür, dass der Roboter komplett blockiert ist.

Um die durch dynamische Objekte sowie die durch die Gebäudegeometrie hervorgerufenen Hindernispunkte aus der Hinderniskarte zu entfernen, werden, wie in Abschnitt 6.2.2 beschrieben, alle 3D-Datenpunkte entfernt, deren Entfernung zum Roboter größer gleich der Entfernung der direkt integrierten Laserscanner ist. Das Ergebnis dieser Filterung ist in 6.14b zu sehen. Der Filter entfernt nahezu alle Hindernispunkte, die durch die Gebäudegeometrie hervorgerufen werden. Es ist zu sehen, dass die Schränke (schwarze „Kästen“ unten rechts) selber nicht zu den Hindernissen hinzugezählt werden, da hier die direkt integrierten Laserscanner und die Abstandssensoren mit Hindernisgedächtnis dieselbe Entfernung liefern. Im Bereich des Regals („Einbuchtung“ unten rechts) sind die Regalbretter jedoch ausschließlich in den Hokuyo-Laserscannern zu sehen. Dies führt dazu, dass die projizierte Entfernung geringer ist als die Entfernung der direkt integrierten SICK-Laserscannerdaten, so dass die Daten der Hokuyo-Laserscanner zu den Hindernispunkten hinzugefügt werden. Die durch den Menschen hervorgerufenen Hindernispunkte werden jedoch zu einem Großteil in die Hinderniskarte übernommen, so dass der Roboter immer noch blockiert wäre. Dies liegt daran, dass in den SICK-Laserscannern nur die Beine des Menschen zu sehen sind, so dass es möglich ist, dass der Oberkörper und die Arme des Menschen Hindernispunkte erzeugen, deren Entfernung geringer als die der SICK-Laserscanner ist.

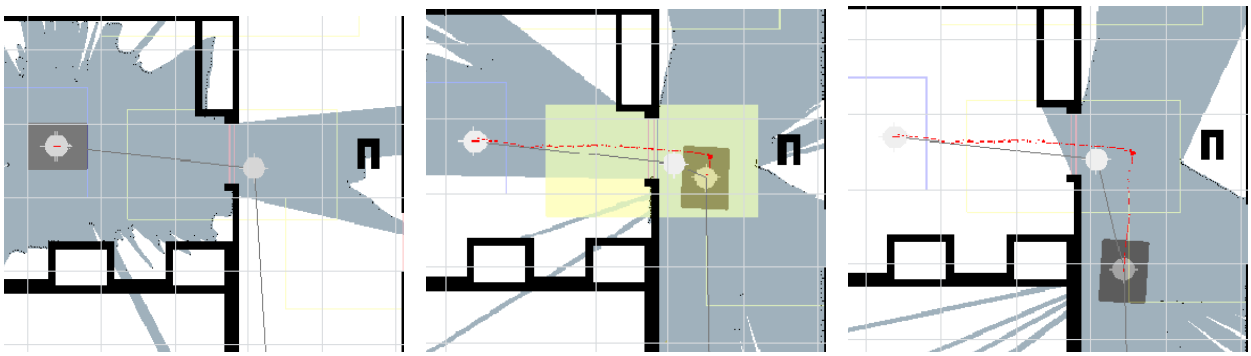
Um die Integration dieser Punkte in die Hinderniskarte zu vermeiden, wird zusätzlich zu den restlichen Filtern mit den in Kapitel 7.2.1 und 7.2.2 beschriebenen Verfahren bestimmt, ob die Punkte im Bereich eines dynamischen Objektes liegen. Das Ergebnis des Filters ist in Abbildung 6.14c zu sehen. Neben der Dynamik werden hier ebenfalls Hindernisse mit einem Abstand von mehr als 2 m oder einer Höhe, die die des Roboters übersteigt, herausgefiltert,

Abbildung 6.15: Experimenteller Aufbau zum Testen der 3D-Hindernisvermeidung durch die Kombination von VRS und klassischer ND-Hindernisvermeidung.

um die aus Lokalisierungsfehlern resultierenden Fehler in der Hinderniskarte zu minimieren. Die Abbildung zeigt, dass sämtliche durch den Menschen hervorgerufenen Hindernispunkte entfernt werden, so dass der Roboter durch den Menschen nicht blockiert wird.

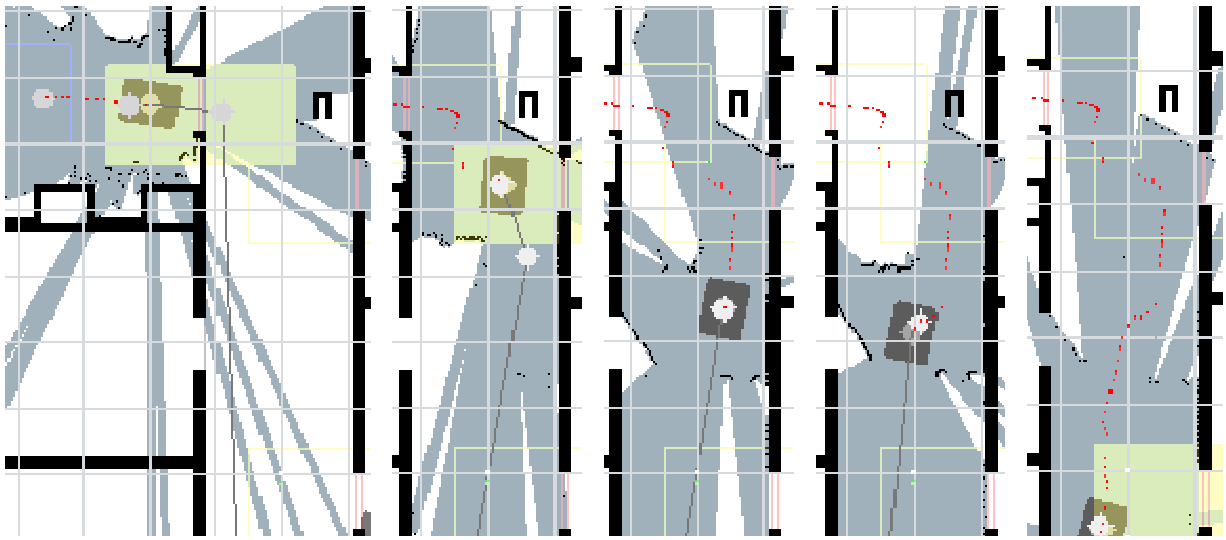
Validierung der 3D-Hindernisvermeidung des LiSA-Roboters

Der Aufbau zu einem weiteren Experiment ist in Abbildung 6.15 dargestellt. Hier wird demonstriert, wie sich das Fahrverhalten des LiSA-Roboters ändert, wenn zur Hindernisvermeidung die 3D-Sensorkonfiguration aus allen sechs Laserscannern bzw. nur die beiden horizontalen SICK-Laserscanner eingesetzt werden. Hierzu werden zwei verschiedene VRS-Datensätze erzeugt und in Kombination mit dem ND-Verfahren verwendet. Der erste VRS verwendet als Eingabesensoren nur die SICK-Laserscanner, die ohne Hindernisgedächtnis integriert werden. Der zweite VRS integriert zusätzlich die Sensordaten der vier Hokuyo-Laserscanner in Kombination mit einem Hindernisgedächtnis. Wichtig bei dem gewählten Aufbau ist, dass die beiden in der Abbildung zu sehenden Stühle nicht über vier Stuhlbeine verfügen und die Stuhlbeine des linken Stuhls sich mittig im Flur befinden. Abbildung 6.16 zeigt die Sensordaten des Roboters beim Abfahren eines Pfades durch den oben beschriebenen Versuchsaufbau. Die Abbildung zeigt den Fall, dass die Hindernisvermeidung mit Hilfe des ersten VRS (nur SICK-Laserscannerdaten) realisiert wird. In 6.16a ist zu sehen, dass der Roboter seinen initialen Pfad geplant hat und mit dem Abfahren beginnt. Die folgende Teilabbildung (b) zeigt, dass der Roboter den Flur erreicht und sich zu diesem ausrichtet (vgl. Kapitel 7.1.1). Von dem Hindernis, das die beiden Stühle bilden, sind aufgrund der zweidimensionalen Sensorkonfiguration ausschließlich die Stuhlbeine zu sehen. Da die Stühle keine vier Stuhlbeine besitzen, ist der freie Bereich in den VRS-Daten auf beiden Seiten des Stuhlbeins des linken Stuhls in etwa gleich groß. Der Roboter fährt in Teilabbildung 6.16c rechts an dem Stuhlbein vorbei, da dies dichter an seinem globalen Pfad liegt. Dies führt dazu, dass er mit den Stühlen kollidiert.



(a) (b) (c)

Abbildung 6.16: Fahrverhalten mit VRS und 2D-Sensorkonfiguration: (a) Der Roboter hat seinen initialen Pfad geplant und beginnt mit dem Abfahren. (b) Es sind nur die Stuhlbeine der Stühle, die den Pfad des Roboters blockieren, in den Laserdaten zu sehen. (c) Der Roboter kollidiert mit den Stühlen.



(a) (b) (c) (d) (e)

Abbildung 6.17: Fahrverhalten mit VRS und 3D-Sensorkonfiguration: (a) Der Roboter hat seinen initialen Pfad geplant und beginnt mit dem Abfahren. (b),(c) und (d) Die gesamte Sitzfläche der Stühle sind in den 3D-Daten des Laserscanners vorhanden und der Roboter weicht diesen aus. (e) Der Roboter hat sein Ziel erreicht.

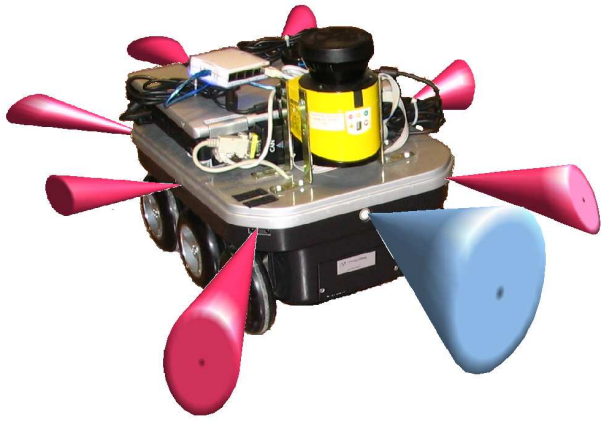


Abbildung 6.18: Die autonome Plattform Kurt2D ist mit einem nach vorne gerichteten Ultraschallsensor (blau) und einem Ring von sieben Infrarotsensoren (rot) ausgestattet. Neben diesen Sensoren besitzt er einen nach vorne gerichteten SICK *S*300-Laserscanner.

In Abbildung 6.17 sind die Sensordaten für das gleiche Experiment gezeigt, nur das zur Hindernisvermeidung die VRS-Daten der 3D-Sensorkonfiguration verwendet werden. Hier ist zu sehen, dass die VRS-Daten nicht nur die Stuhlbeine enthalten, sondern die gesamte Sitzfläche der beiden Stühle als Hindernis erkennt. Da es sich bei den Stühlen um statische Hindernisse handelt, wird wie in Kapitel 6.3.2 beschrieben der Pfad an dieses Hindernis angepasst und führt nun aus Sicht des Roboters links an den Stühlen vorbei. Der LiSA-Roboter ist somit in der Lage, kollisionsfrei in einer dynamischen, dreidimensionalen Arbeitsumgebung zu navigieren.

Kombination verschiedener Sensortypen im VRS

In diesem Abschnitt geht es darum, die Fähigkeit des VRS, verschiedene Sensortypen zu integrieren, experimentell zu belegen. Zunächst werden Infrarot- und Ultraschallsensoren mit einem Laserscanner via VRS kombiniert. Als Träger der Sensoren wird eine weitere autonome mobile Plattform eingesetzt, die ebenfalls verdeutlicht, dass das VRS-Prinzip nicht auf den LiSA-Roboter beschränkt ist. Abbildung 6.18 zeigt den Roboter. Er ist mit einem Ring aus fünf Infrarotsensoren ausgestattet. Zusammen mit einem nach vorne ausgerichteten Ultraschallsensor decken diese Sensoren die Umgebung der Plattform ab. Da Ultraschall- oder Infrarotsensoren im Gegensatz zu Laserscannern einen nicht vernachlässigbaren Öffnungswinkel des ausgesendeten Signals haben, muss zunächst das Problem gelöst werden, die Sensordaten auf 3D-Punkte abzubilden. Da der VRS-Datensatz zur Hindernisvermeidung eingesetzt wird, muss sichergestellt werden, dass das Objekt, welches den Entfernungswert des Ultraschallsensors hervorruft, auf jeden Fall durch einen 3D-Punkt erfasst wird. Aus diesem Grund werden die Ultraschall- und Infrarotsensoren als Kegel modelliert, über deren Grundfläche in der Auflösung des VRS-Datensatzes linear Punkte verteilt werden. Sämtliche Punkte einer Ultraschall- bzw. Infrarotmessung werden in den VRS-Datensatz integriert.

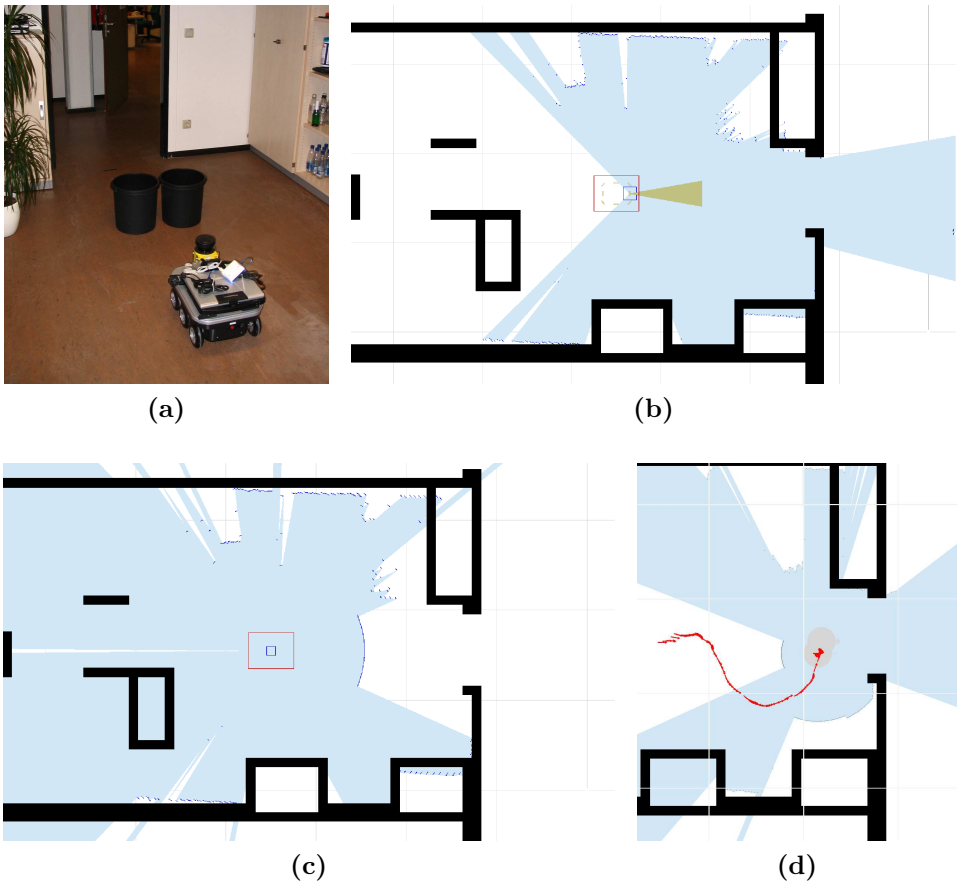


Abbildung 6.19: VRS-Fusion von Ultraschall- und Laserdaten: (a) Experimenteller Aufbau: Der vom Roboter abzufahrende Pfad ist durch zwei vor ihm befindliche Papierkörbe blockiert. (b) Die Höhe der Hindernisse ist geringer als die der Laserscannerebene, so dass diese ausschließlich in den Daten der Ultraschall- und Infrarotsensoren (grün) sichtbar sind. Der VRS-Datensatz fusioniert die Umgebungsinformation aller Sensoren (c) und ermöglicht ein kollisionsfreies Anfahren der Zielpose (d).

Abbildung 6.19 zeigt den Parcours zum Testen des VRS, der Ultraschall- und Infrarotsensoren sowie einen nach vorne gerichteten SICK *S*300-Laserscanner fusioniert. Der Roboter soll eine direkt vor ihm liegende Zielpose in 2 m Entfernung anfahren. Der direkte Weg ist durch zwei Papierkörbe blockiert. Wie in Abbildung (b) zu sehen ist, befinden sich die Papierkörbe unterhalb der Scanebene des SICK Laserscanners (blau). Der Ultraschallsensor erfasst die Hindernisse, so dass diese im VRS-Datensatz vorhanden sind (b). Da die Ultraschall- und Infrarotsensoren die Umgebung des Roboters rundum abdecken und alle Sensoren Hindernisse in etwa derselben Höhe erfassen, können die Sensoren ohne Hindernisgedächtnis in den VRS-Datensatz integriert werden. Abbildung (d) zeigt, dass die ND-basierte Hindernisvermeidung mit VRS als Eingabesensor in der Lage ist, die Hindernisse zu umfahren und kollisionsfrei die Zielpose zu erreichen.

Als weiteren Sensor zur Distanzmessung wird in dieser Arbeit eine 3D-TOF-Kamera ver-

Abbildung 6.20: Photonic Mixer Device (PMD)-Kamera in Kombination mit einer RGB-Kamera: Die PMD-Kamera (unten) leuchtet die Szene aktiv mit einem Feld von Dioden aus und liefert über Laufzeitmessungen eine 3D-Punktwolke der betrachteten Szene. Den einzelnen Punkten können durch die RGB-Kamera (oben) Farbwerte zugewiesen werden.

wendet. Diese in Abbildung 6.20 zu sehende Photonic Mixer Device (PMD)-Kamera ist ein Sensor, der ein dreidimensionales Bild einer Szene mit einer hohen Aktualisierungsfrequenz liefert [41]. Dies wird erreicht, indem mit dem Photomischdetektor (PMD) ein neuartiges auf der CMOS Technologie basiertes Halbleiterelement eingesetzt wird. Dieser Detektor ermöglicht durch Laufzeitmessungen von ausgestrahltem Licht, das nach einer Reflextion an einem Objekt wieder auf die Kamera trifft eine Entfernungsbestimmung des Objekts. Werden mehrere PMD-Detektoren in einer Matrix angebracht, ist es möglich, parallel für mehrere Punkte einer Szene die Entfernung zu bestimmen. Die Kombination der einzelnen Abstandswerte liefert eine 3D-Punktwolke Szene. In dieser Arbeit wird die in Abbildung 6.20 gezeigte Kamera O3D100 der Firma IFM verwendet. Sie besitzt eine Auflösung von 50×64 Pixel und einen maximalen Messbereich von 7,5 m. Die Kamera besitzt eine integrierte Unterdrückung von Hintergrundstrahlung und eine Standardabweichung zwischen 2 mm und 8 mm. Da die Kamera ausschließlich Entfernungs- und Intensitätsdaten liefert, wird sie in dieser Arbeit mit einer RGB-Kamera kombiniert, um den einzelnen 3D-Punkten der Szene Farbwerte zuordnen zu können. Die Farbinformation wird im Folgenden dazu verwendet, um Sensorfehler zu filtern. Die Kamera besitzt also den Vorteil, dass in jedem Datensatz die 3D-Information der Umgebung vorhanden ist und es z.B. nicht zu dem in Abschnitt 6.2.2 beschriebenen Problem der mitwandernden Tischplatten kommen kann. Obwohl die Kamera ein permanenter 3D-Sensor ist, werden die Daten der Kamera mit Hindernisgedächtnis in den VRS-Datensatz integriert. Dies ist der Fall, da die PMD-Kamera durch den relativ kleinen Öffnungswinkel nur einen begrenzten Bereich der Umgebung abdecken kann. Somit sind an die Sensordaten der Kamera hohe Anforderungen bzgl. Genauigkeit und Rauschen gestellt, um Fehleinträge in der Hinderniskarte zu minimieren.

Das in Abbildung 6.21 dargestellte Experiment zeigt jedoch, dass die von der PMD-Kamera erzeugten Daten teilweise stark verrauscht sind. In Teilabbildung (a) und (b) ist die reale

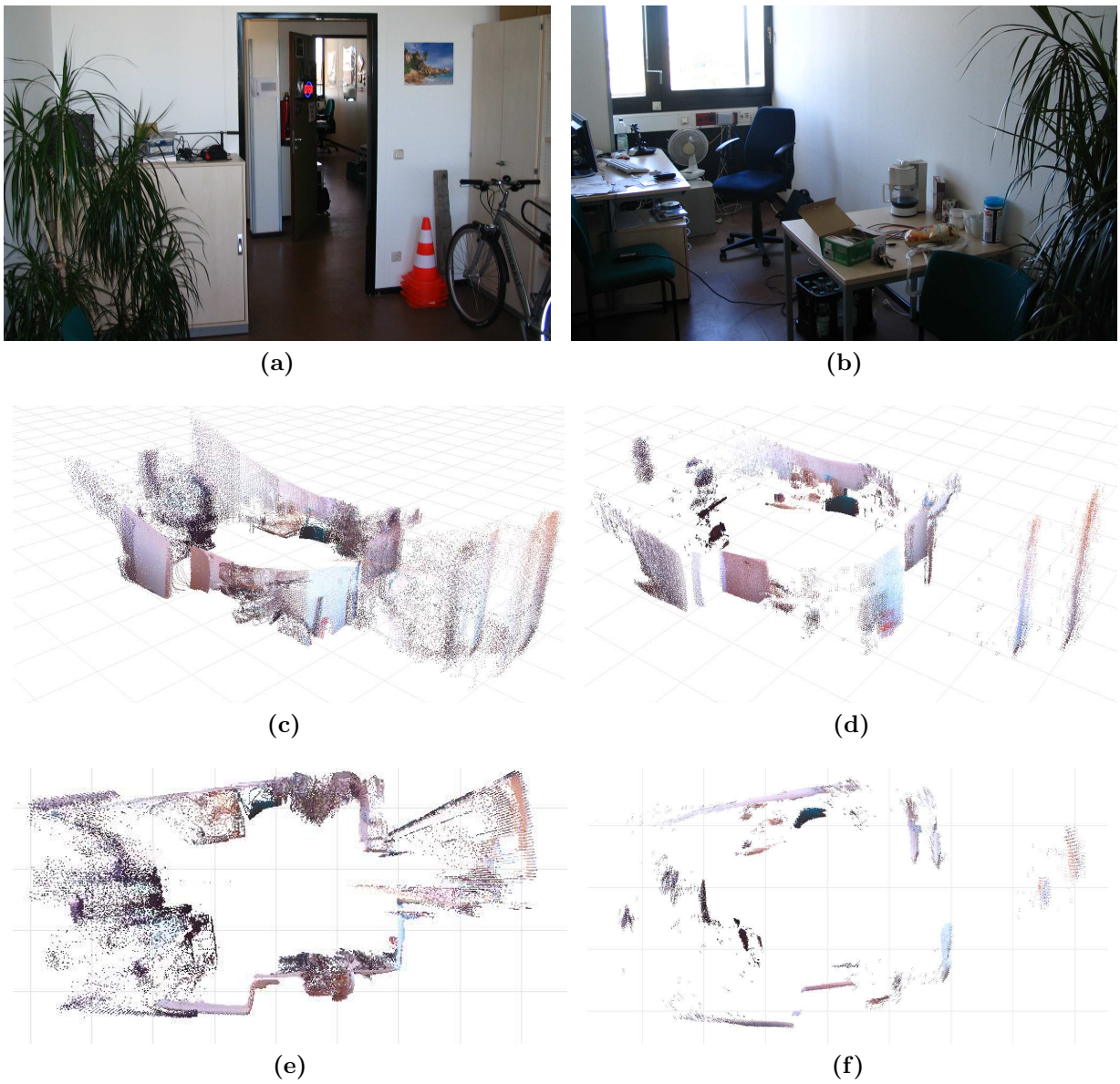


Abbildung 6.21: Reduktion von Fehlmessungen der PMD-Kamera: (a) und (b): Panorama der realen Umgebung. (c) und (e) 3D-Punktwolke von mehreren registrierten PMD-Kameradatensätzen. (d) und (f) 3D-Punktwolke mit herausgefilterten Fehlmessungen.

Büroumgebung zu sehen, in der der PMD-Datensatz aufgenommen wird. In Teilabbildung (c) und (e) sind jeweils registrierte Punktwolken der PMD-Kamera zu sehen, die aus einer 360°-Drehung des Roboters in dieser Umgebung resultieren. Die Abbildungen zeigen, dass der Sensor anfällig gegenüber spiegelnden Oberflächen und Tiefensprüngen ist. Dies wird insbesondere an der Tür und der Fensterfront deutlich [46]. Die Fehlmessungen, die an spiegelnden Oberflächen häufig sehr geringe Abstandswerte annehmen, würden bei der VRS-Integration der PMD-Kamera zu einem regelmäßigen Notstopp des Roboters führen. Ferner würden die durch Tiefensprünge hervorgerufenen Fehlmessungen, wie sie am Beispiel der Tür zu sehen sind, dafür sorgen, dass der abzufahrende Pfad in den VRS-Daten als blockiert erscheint. Der Lösungsansatz, die PMD-Kamera trotz des begrenzten Öffnungswinkels nicht in die lokale Hinderniskarte zu integrieren, würde dieses Problem zwar lösen, kann jedoch den regelmäßigen Not-Stopp des Roboters aufgrund von Reflexionsfehlmessungen nicht verhindern.

Um die PMD-Kamera dennoch zur Hindernisvermeidung nutzen zu können, müssen deren Daten vor der VRS-Integration gefiltert werden. Hierzu werden nach [33] die durch die Kombination der PMD-Kamera mit einer Kamera vorhandenen Farbinformation c sowie die Tiefeninformation d der einzelnen Pixel der PMD-Kamera verwendet, um die Fehlmessungen zu bestimmen und herauszufiltern. Das Verfahren berechnet iterativ für jedes Pixel der PMD-Kamera nach Gleichung 6.4 eine Gewichtung, die die Gaußverteilung über die Tiefen- und Farbdifferenz bildet. Die Gewichtung für ein Pixel bestimmt sich durch die Summe aller Gaußverteilungen zu den Nachbarpixeln, N_i.

$$s_i = \sum_{j \in N_i} s_j \, exp\left(-\frac{1}{\sigma_d}\omega_{ij}(d_i - d_j)^2\right) \quad \text{mit} \quad \omega_{ij} = exp\left(-\frac{1}{\sigma_c}||c_i - c_j||_2^2\right) \tag{6.4}$$

Mit Hilfe der Variabeln σ_d und σ_c lässt sich das Gewichtungsverhältnis von Tiefen- zu Farbinformation steuern. Besitzt ein Pixel i eine Gewichtung s_i, die unter einem definierten Schwellwert liegt, wird der Pixel als Fehlmessung interpretiert und herausgefiltert. Das Ergebnis der Filterung ist in Abbildung 6.21d und (f) zu sehen. Im Vergleich mit der ungefilterten Punktwolke wird deutlich, dass nahezu alle Fehlpixel erfolgreich entfernt werden. Somit lässt sich die PMD-Kamera via VRS integrieren, da die einzige Voraussetzung hierfür erfüllt ist, dass sich die Sensordaten auf Punkte im dreidimensionalen Raum abbilden lassen.

Abbildung 6.22 zeigt das Ergebnis eines Experiments zur Hindernisvermeidung, das einen VRS-Datensatz verwendet, der eine PMD-Kamera und einen Laserscanner fusioniert. Die PMD-Kamera wird mit Hindernisgedächtnis via VRS integriert. In (a) ist der autonome Roboter zu sehen, der mit einem nach vorne ausgerichteten SICK $S300$-Laserscanner und einer ebenfalls nach vorne ausgerichteten PMD-Kamera ausgerichtet ist. Der Ablauf des Experiments gleicht dem vorangegangenen Experiment zur Fusion von Ultraschall- und Laserscannerdaten (vgl. 6.19). Wie in Teilabbildung 6.22b zu sehen ist, sind die Papierkörbe, die den direkten Pfad blockieren, nicht in den Daten des Laserscanners (blau) vorhanden. Die PMD-Kamera erfasst die Hindernisse (rote Punkte), so dass der VRS-Datensatz in Teilabbildung (c) die Information über die vor dem Roboter liegenden Hindernisse enthält. Dies ermöglicht der Hindernisvermeidung um die Papierkörbe ((d) schwarze Kreise) herum zu navigieren (roter Pfad) und das Ziel kollisionsfrei zu erreichen.

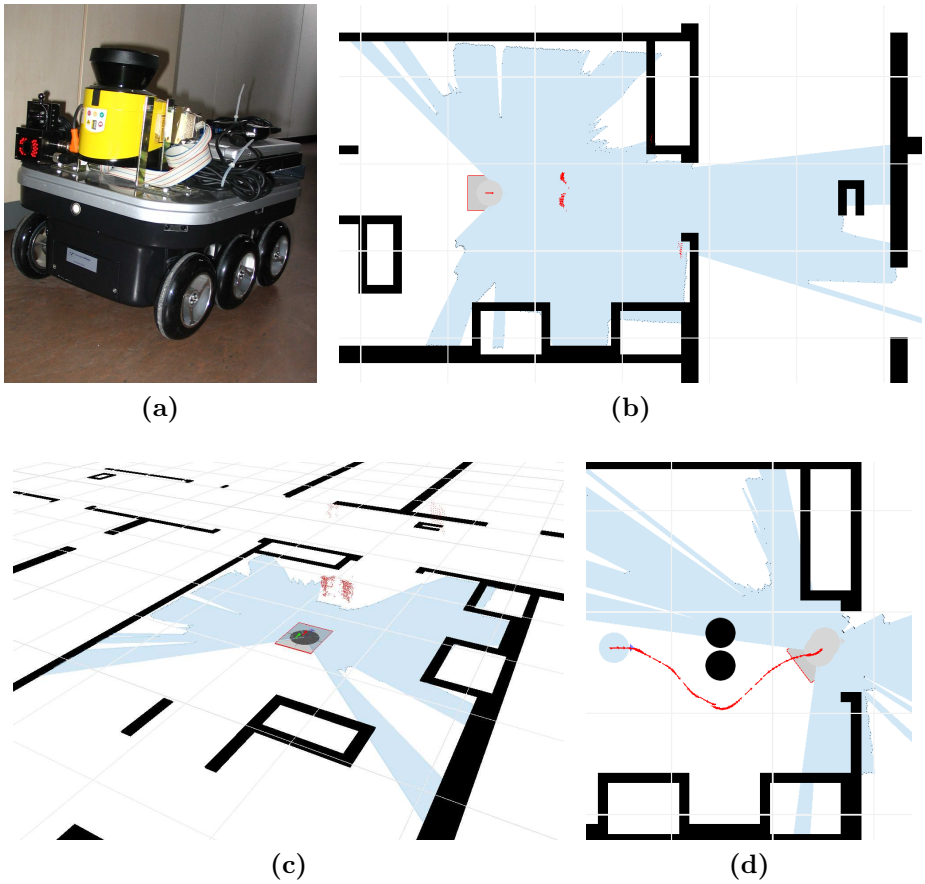


Abbildung 6.22: VRS-Fusion einer Time-Of-Flight 3D-Kamera mit Laserscanner-Daten. Die Umgebung entspricht der in Abbildung 6.19a: (a): Mit Laserscanner und PMD-Kamera ausgestatteter autonomer Roboter. (b): Die Laserscannerebene (blau) liegt über den Papierkörben, so dass diese nur in den PMD-Daten zu sehen sind. (c): Der aus PMD-Kamera und Laserscanner kombinierte VRS-Datensatz enthält die Information über die vor dem Roboter liegenden Hindernisse. (d): Die ND-Hindernisvermeidung navigiert den Roboter anhand der VRS-Daten um die Hindernisse herum.

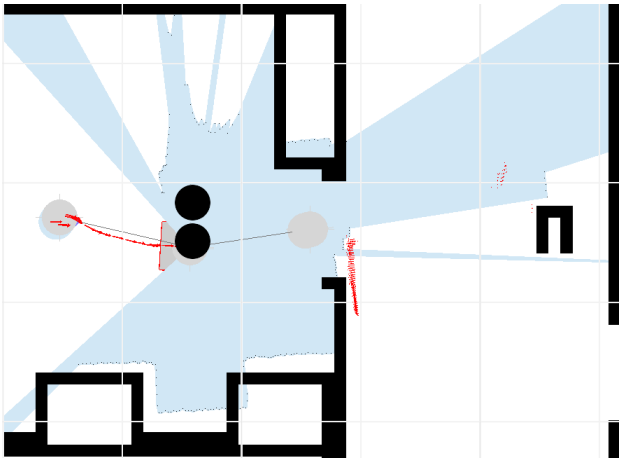


Abbildung 6.23: Einfluss des Hindernisgedächtnises im VRS-Konzept: Ohne Hindernisgedächtnis wird den Hindernissen (schwarze Kreise) nur solange ausgewichen, wie diese im Sichtbereich der PMD-Kamera sind (rote Punkte). Aufgrund des geringen Öffnungswinkels der Kamera führt dies zu einer Kollision des Roboters mit dem Hindernis.

Abbildung 6.23 zeigt ein weiteres Experiment, das den Einfluss des Hindernisgedächtnises verdeutlicht. Es wird untersucht, wie sich eine Variation der Dauer, für die ein Hindernis im Gedächtnis bleibt, auf den Hindernisvermeidungsprozess auswirkt. Das oben beschriebene Experiment zur Fusion einer PMD-Kamera mit einem Laserscanner wird mit derselben Hardware wiederholt. Das Hindernisgedächtnis wird jedoch von 25 Sekunden auf nur eine Sekunde zurückgesetzt. Dies hat zur Folge, dass die Gedächtniswirkung der lokalen Hinderniskarte nahezu aufgehoben ist. Somit wird den Papierkörben nur solange ausgewichen, wie sich diese im begrenzten Sichtbereich der PMD-Kamera befinden. Wie Abbildung 6.23 zeigt, führt dies zu einer Kollision des Roboters mit den Papierkörben, da die PMD-Kamera aufgrund ihres geringen Öffnungswinkels nicht die gesamte Front des Roboters abdeckt und die Scanebene des Laserscanners über den Papierkörben liegt.

Kombination der Playermodule zur 3D Hindernisvermeidung und Lokalisierung

Abbildung 6.24 zeigt, wie 12 Player Devices kombiniert werden, um mit Hilfe der sechs Laserscanner der LiSA-Sensorkonfiguration die Hindernisvermeidung sowie die Lokalisierung zu realisieren. Aus den sechs realen Sensoren werden zwei VRS-Datensätze generiert. Der erste VRS-Datensatz integriert die Daten der beiden SICK-Laserscanner. Da die Laserscanner in derselben horizontalen Ebene etwa 10 cm über dem Boden scannen, enthält dieser VRS-Datensatz ausschließlich Entfernungswerte aus dieser Ebene. Da die Laserstrahlen in dieser Höhe nur die Beine von Stühlen und Tischen treffen, die sich im Arbeitsbereich des Roboters befinden, ist die Wahrscheinlichkeit groß, dass diese Daten die Gebäudegeometrie widerspiegeln. Aus diesem Grund wird dieser Laserscanner in Kombination mit der in Kapitel 6.1.4 beschriebenen Monte Carlo Lokalisierung verwendet (vgl. Abbildung 6.24(AMCL)). Da die SICK-Laserscanner an gegenüberliegenden Ecken des Roboters angebracht sind und

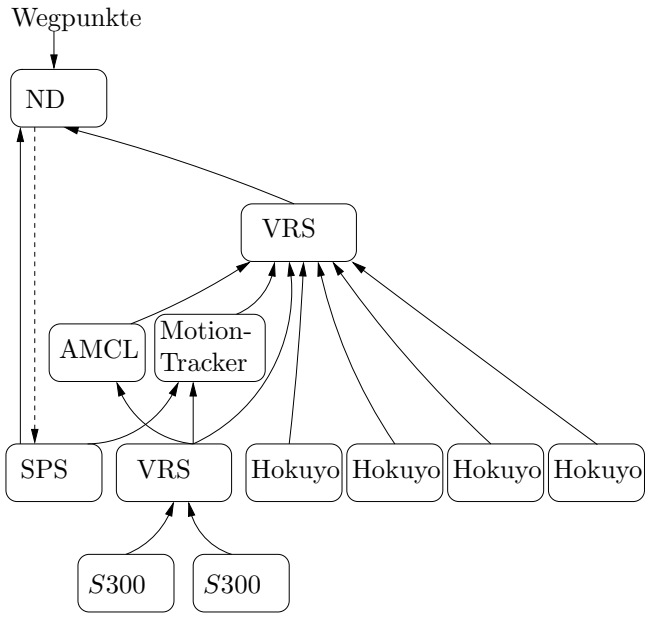
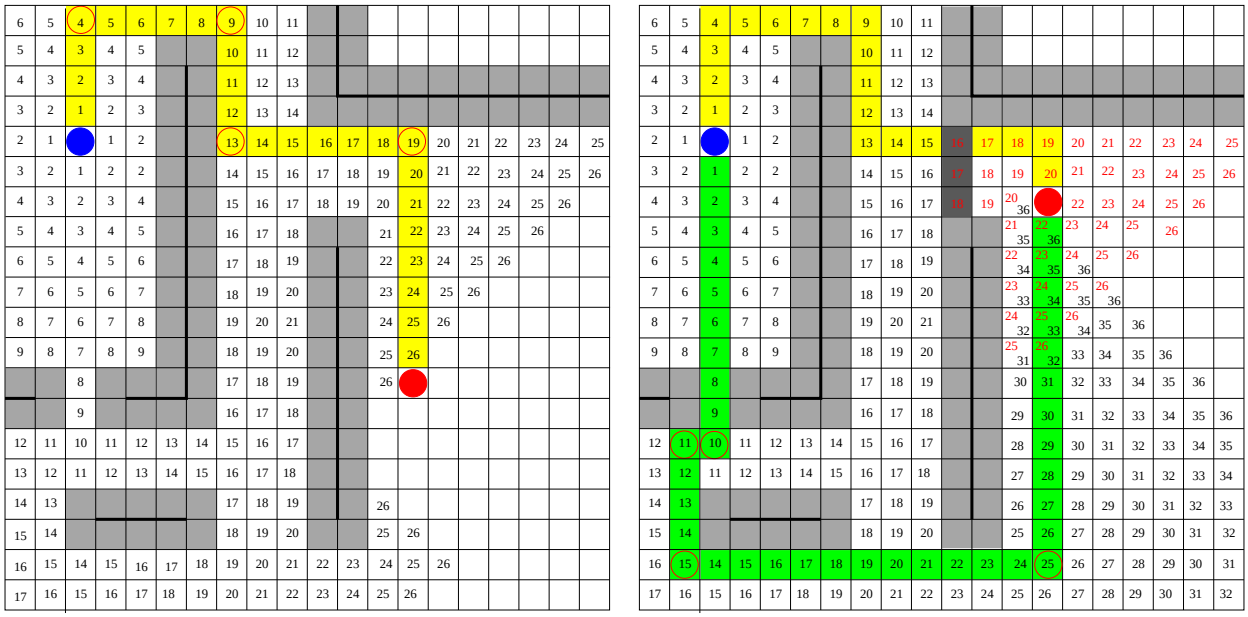


Abbildung 6.24: Zusammenspiel der verschiedenen Player Devices zur Hindernisvermeidung und Lokalisierung. Erklärungen im Text.

jeweils einen Sichtbereich von 270° besitzen, verfügt dieser VRS-Datensatz über ein Sichtfeld von 360°, in dem sogar überlappende Einzeldatensätze der realen Sensoren existieren. Diese Rundumsicht wird ebenfalls beim Motion-Tracker-Device genutzt, indem dieser die VRS-Daten verwendet, um dynamische Objekte im Umfeld des Roboters zu verfolgen.

Der zweite VRS-Datensatz verwendet den ersten VRS-Datensatz und integriert zusätzlich die Sensordaten der vier Hokuyo-Laserscanner in Kombination mit einer Hinderniskarte. Da die Verwendung einer Hinderniskarte eine globale Lokalisierung des Roboters voraussetzt, benötigt der zweite VRS-Datensatz neben den virtuellen und realen Sensoren die Daten der Lokalisierung (AMCL). Ferner werden die Daten des Motion-Tracker-Devices benötigt, um dynamische Hindernisse aus der Hinderniskarte herauszufiltern. Dieser VRS-Datensatz repräsentiert die 3D-Information der Umgebung in einem 2D-Datensatz und dient als Sensor für das ND-Hindernisvermeidungs-Device. Dieses bestimmt anhand der Sensordaten und der Wegpunkte des globalen Pfadplaners die optimale Kombination aus Bewegungsrichtung, Geschwindigkeit und Orientierungsänderung und sendet die Fahrbefehle an das SPS-Device, welches mit der SPS des LiSA-Roboters kommuniziert. Das SPS-Device enthält die in Kapitel 4.2.2 beschriebene inverse Kinematik des LiSA-Roboters und rechnet die Fahrbefehle in Radorientierungen und Geschwindigkeiten um. Diese leitet es an die SPS des Roboters weiter. Neben der inversen Kinematik enthält das SPS-Device auch die Vorwärtskinematik des LiSA-Roboters und berechnet die Schätzung der globalen Roboterpose anhand der Radgeschwindigkeiten und Orientierungen, die es von der SPS bekommt. Diese Schätzung dient als weitere Eingabe zur Lokalisierung mit dem AMCL-Device.



(a) **(b)**

Abbildung 6.25: Pfadplanung mit Hilfe des Wavefront Algorithmus: (a) Mit Hilfe einer sich ausbreitenden Wellenfront in einer Rasterkarte wird über ein Gradientenabstiegsverfahren der optimale Pfad zwischen einer Startposition (roter Kreis) zu einer Zielposition (blauer Kreis) bestimmt. (b) Im Falle einer Blockade werden die rot markierten Raster gelöscht. Anschließend breitet sich die Welle wieder aus, bis sie die Startposition erreicht hat.

6.3 Globale Pfadplanung

Pfadplanung ist die Aufgabe, einen geometrisch, kinematisch und dynamisch fahrbaren Weg von einer Start- in eine Zielpose zu berechnen. Hierzu benötigt sie eine Repräsentation der a priori bekannten Roboterumgebung sowie die Geometrie und Kinematik des Roboters selbst.

6.3.1 Wavefront-Pfadplaner

Der Wavefront-Pfadplaner ist ein zweidimensionaler Pfadplaner und macht, wie viele andere Pfadplanungsalgorithmen auch, folgende Grundannahmen, um das Pfadplanungsproblem zu vereinfachen [42]. Es wird angenommen, dass der Roboter rund ist, so dass die aktuelle Orientierung unwichtig ist und das Problem um eine Dimension reduziert wird. Des Weiteren wird angenommen, dass der Roboter holonom ist, so dass er auf der Stelle drehen und eine beliebige Bewegungsrichtung einschlagen kann.

Der Wavefront Pfadplaner benötigt eine zweidimensionale Karte der Umgebung zur Pfadplanung. Der Freiraum der Rasterkarte repräsentiert den Konfigurationsraum des Roboters. Im Konfigurationsraum sind Hindernisse vergrößert (vgl. Abbildung 6.25 graue Felder), so dass der Roboter als punktförmig betrachtet werden kann. Der Konfigurationsraum wird

ebenfalls dazu genutzt, um die minimale Distanz zu definieren, die ein Pfadpunkt zur Umgebungsgeometrie besitzen darf. Hierzu werden Hindernisse in der Karte nicht nur um die Dimensionen des Roboters, sondern noch zusätzlich um diesen Mindestabstand vergrößert.

Wie in Abbildung 6.25a zu sehen ist, werden iterativ, ausgehend von der vorgegebenen Zielposition des Roboters (blauer Punkt), allen Rastern der Karte, denen noch kein Wert zugewiesen wurde und die eine Vierernachbarschaft zum jeweils betrachteten Raster besitzen, mit einem um Eins inkrementierten Wert des aktuellen Rasters zugewiesen. Diese Wellenausbreitung wird so lange fortgesetzt, bis die Startposition (roter Punkt) erreicht ist. In dem in Abbildung 6.25a gezeigten Beispiel ist dies nach 27 Iterationen der Fall. Ist die Startposition erreicht, beginnt ausgehend von dieser ein Gradientenabstiegsverfahren, das jeweils in Vierernachbarschaft nach dem geringsten zugewiesenen Wert sucht und dieses Raster als nächstes betrachtet. Die auf diese Weise besuchten Raster bilden den Pfad von der Start- zur Zielpose. Besitzen mehrere Raster der Vierernachbarschaft den geringsten Wert, wird das Raster bevorzugt, das zu einem geraden Pfad führt. Der Pfad bildet somit die Verbindung der Start- mit der Zielposition mit der kürzesten Wegstrecke. Des Weiteren ist der Pfad in gerade Segmente maximaler Länge unterteilt. Die Endpunkte der geraden Segmente bilden die Wegpunkte des Pfades.

Es gibt alternative Implementierungen des Algorithmus, die nicht die Vierernachbarschaft, sondern die Achternachbarschaft des jeweiligen Rasters betrachten. Beim Gradientenabstiegsverfahren sind dadurch auch Raster erlaubt, die denselben Wert besitzen, wie das aktuelle Raster. Wird beim Gradientenabstiegsverfahren ebenfalls Achternachbarschaft berücksichtigt, sind damit auch diagonale Bewegungen des Roboters erlaubt. Diese Variante birgt aber die Gefahr, dass ein Hindernis durch eine diagonale Bewegung „geschnitten“ wird.

Abbildung 6.25b verdeutlicht, warum das Wavefront-Verfahren bei der Zielpose mit der Ausbreitung der Wellenfront beginnt. Hier ist der Fall dargestellt, dass der Roboter während des Abfahrens des Pfades anhand seiner Sensorinformationen erkennt, dass der Pfad blockiert ist (dunkelgraue Felder). Nun müssen nicht allen Rastern neue Werte zugewiesen werden. Statt dessen werden ausgehend von den blockierten Rastern alle Raster zurückgesetzt, die einen höheren Wert besitzen (rote Zahlen). Besitzt die aktuelle Position des Roboters nun keinen Wert mehr, beginnt das iterative Zuweisen der Rasterwerte bei dem höchsten noch in der Karte vorhandenen Wert, bis die Roboterposition erreicht ist. Anschließend wird mit einem erneuten Gradientenabstiegsverfahren ein alternativer Pfad zum Ziel verfolgt.

6.3.2 Pfadanpassung an statische Hindernisse

Oft reicht eine lokale Pfadanpassung, wie sie in den klassischen, zweidimensionalen Hindernisvermeidungsmodulen realisiert wird, nicht aus. Blockiert ein Objekt den Pfad des Roboters, ist der alternative Pfad häufig mit rein reaktiver Pfadanpassung nicht zu finden. Da die Hindernisvermeidungsmodule keine globale Repräsentation der Umgebung besitzen, kommt es zu einem oszillierenden Fahrverhalten. Der Roboter weicht iterativ von der Blockade zurück, bis diese nicht mehr zu sehen ist, und fährt anschließend wieder auf diese

zu. Es gibt grundsätzlich zwei Ansätze, dieses Problem zu lösen. Zum einen können die lokalen Hindernisvermeidungsmodule erweitert werden, so dass sie mit einer Art Gedächtnis ausgestattet werden und die Blockade noch berücksichtigt wird, auch wenn sie in den Sensordaten nicht mehr vorhanden ist. Problematisch an diesem Ansatz ist es, den Schwellwert festzulegen, der bestimmt, wie lange die Blockade in der Karte vorgehalten werden soll. Ein zweites Problem ist, dass solche Verfahren immer Lokalisierungsdaten benötigen, um eine globale Repräsentation der Umgebung zu erstellen [49]. Da die Lokalisierung immer mit Fehlern behaftet ist, ist es möglich, dass im Gegensatz zu einem rein reaktiven Verfahren, das nur den aktuellen Datensatz berücksichtigt, der Roboter durch falsch registrierte Daten nicht vorhandenen Hindernissen ausweicht, bzw. von diesen blockiert wird. Ferner ist es möglich, dass dynamische Objekte den Roboter blockieren können, da sie ebenfalls in die Karte eingetragen werden.

Der zweite Ansatz verschiebt die Lösung des Problems zu den globalen Pfadplanungsalgorithmen und lässt die lokalen Hindernisvermeidungsmodule rein reaktiv. Hier besteht das Problem zwischen statischen und dynamischen Hindernissen unterscheiden zu können. In verschiedenen Arbeiten [51, 2] wird hierzu eine Rasterkarte verwendet, in der neben der Wahrscheinlichkeit, dass ein Raster belegt ist, ebenfalls der zeitliche Verlauf der Belegungswahrscheinlichkeiten in einer Baumstruktur gespeichert ist. Anhand des zeitlichen Verlaufs wird zwischen statischen Objekten (Gebäudegeometrie), Objekten mit geringer Dynamik (Türen, Stühle,...) und hochdynamischen Objekten (Menschen) unterschieden. Ein Raster wird dann als statisch klassifiziert, wenn die Belegungswahrscheinlichkeit eine definierte Zeit über einem definierten Schwellwert liegt. Die Bestimmung dieses Schwellwertes ist problematisch. Zum einen muss der Schwellwert relativ gering sein, damit der Roboter nicht zu lange durch statische Hindernisse blockiert ist, bevor er diese in die globale Pfadplanung integriert. Zum anderen führt ein geringer Schwellwert dazu, dass dynamische Hindernisse, die sich kurzzeitig an der selben Position befinden (z.B. ein stehender Mensch) als statische Hindernisse angenommen werden. Um dieses Problem zu lösen, werden in dieser Arbeit mit Hilfe der in Kapitel 7.2.1 und 7.2.2 beschriebenen Verfahren dynamische Objekte in den Abstandsdaten identifiziert und verfolgt. Die zu den dynamischen Objekten gehörenden Sensordaten werden nicht in die zeitliche Rasterkarte integriert. Ebenfalls werden alle Sensordaten, die oberhalb der Bounding Box des Roboters liegen, herausgefiltert, da diese zu Objekten gehören, die den Roboter nicht blockieren können. Die restlichen Sensordaten werden in die zeitliche Rasterkarte aufgenommen. Wird ein Raster eine definierte Zeit als belegt gemessen, wird es als statisch klassifiziert. Alle statischen Raster werden in die a priori bekannte Karte zur globalen Pfadplanung übernommen.

Da der LiSA-Roboter erkennen muss, wann eine Blockade des Pfades nicht mehr besteht, wird ebenfalls bestimmt, wann ein statisches Raster nicht mehr belegt ist. Hier besteht die Möglichkeit ein Raster als frei zu klassifizieren, wenn es für einen definierten Zeitraum als frei gemessen wurde. Diese Methode hat den Nachteil, dass für alle Sensordaten bestimmt werden muss, welche Raster im aktuellen Sichtbereich des Roboters liegen. Dies erfordert eine hohe Rechenleistung. Ferner besteht das Problem, dass bei einer 3D-Sensorkonfiguration die Information gespeichert werden muss, welcher Sensor ein statisches Objekt identifiziert

hat, da die 3D-Umgebungsinformation in einer 2D-Rasterkarte gespeichert wird. Ohne die Information, welcher Sensor das statische Hindernis entdeckt hat, würden z.B. die schrägen Hokuyo-Laserscanner der LiSA-Sensorkonfiguration eine Tischplatte als statisches Hindernis erkennen, während die horizontalen SICK-Laserscanner unter der Tischplatte hindurchschauen und den Bereich als frei markieren. Es müssen somit dieselben Dinge berücksichtigt werden, wie bei der lokalen Hinderniskarte im VRS-Konzept (vgl. Abschnitt 6.2.2).

In dieser Arbeit wird der Ansatz verfolgt, auf der zeitlichen Rasterkarte einen Alterungsprozess laufen zu lassen, so dass Raster, die ein definiertes Zeitintervall nicht als belegt gemessen werden, als frei angenommen werden. Hier besteht die bereits angesprochene Gefahr des oszillierenden Fahrverhaltens des Roboters. Um diese Gefahr zu minimieren, wird das in Kapitel 7.1 beschriebene Zonenkonzept dazu eingesetzt die statischen Hindernisse mit semantischen Informationen zu versehen. Somit lässt sich z.B. feststellen, ob das Objekt, das den Pfad blockiert, eine geschlossene Tür ist. Dies erlaubt es, falls kein alternativer Pfad vorhanden ist, dem Benutzer mitzuteilen, welche Tür verschlossen ist. Ferner wird auf statische Hindernisse, die als Tür markiert sind, kein Alterungsprozess angewendet.

6.3.3 Experimentelle Validierung der Globalen Pfadplanung

In Abbildung 6.26 ist das Ergebnis eines Experiments zur Pfadanpassung an statische Hindernisse zu sehen. Die Zielpose des Roboters wird in einen Raum gesetzt, der über zwei unterschiedliche Türen erreichbar ist. Beide Türen werden geschlossen, so dass der Roboter sein Ziel nicht erreichen kann. In (a) wird der initiale Pfad für den Roboter geplant. Es sind noch keine statischen Hindernisse identifiziert und der Roboter beginnt den Pfad abzufahren. (b) und (c) zeigen den Roboter auf dem Korridor, wie er auf die Tür des Zielraums zu fährt. Bereits hier ist in den Sensordaten zu sehen, dass die Tür verschlossen ist. Sie wird jedoch noch nicht als statisch erkannt, da sie erst für einen definierten Zeitraum sichtbar sein muss. Dieser Schwellwert ist in (d) überschritten. Die Rasterkarte für statische Hindernisse zeigt die Raster der Tür als belegt an. Da sich die Raster in einer Türregion befinden (vgl. Kapitel 7.1), wird das statische Hindernis mit der Marke Tür versehen. Die statischen Hindernisse werden in die globale Karte, die zur Pfadplanung verwendet wird, übertragen, und es wird ein alternativer Pfad geplant, der die zweite Tür des Zielraums verwendet. Teilabbildung 6.26e zeigt den Roboter auf dem alternativen Pfad. Der Roboter hat anhand der Sensordaten die zweite Tür ebenfalls als geschlossen identifiziert. Dies ist bereits bei einer größeren Entfernung als bei der ersten Tür der Fall, da der Roboter aufgrund der Hindernisse des alternativen Pfades nicht so schnell fahren kann und die Tür bereits bei der größeren Entfernung lange genug in den Sensordaten vorhanden ist. Nachdem die zweite Tür ebenfalls in die Pfadplanungskarte aufgenommen worden ist, kann der Pfadplanungsalgorithmus keinen alternativen Pfad zum Ziel mehr finden. Hier zeigt sich, dass es wichtig ist, auf den mit Tür markierten Hindernissen keinen Alterungsprozess laufen zu lassen, da andernfalls die erste Tür wieder aus der Pfadplanungskarte entfernt worden wäre und ein alternativer Pfad durch diese Tür geplant würde. In dem im Experiment gezeigten Fall kann kein alternativer Pfad geplant werden. Dies führt dazu, dass der Benutzer über die Kommunikationssoftware

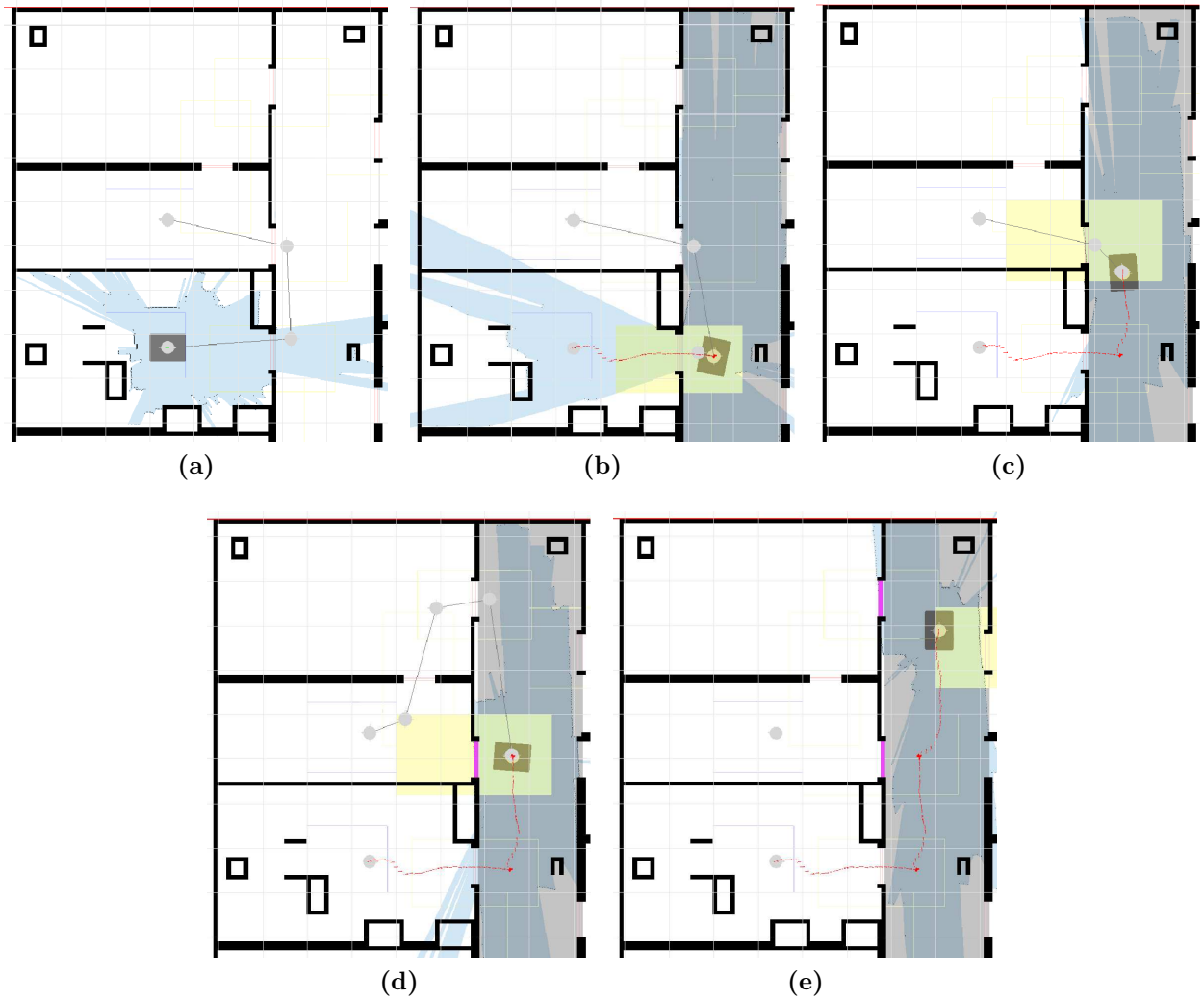


Abbildung 6.26: Pfadanpassung an statische Hindernisse: (a): initialer Pfad ist geplant. (b): Der Roboter hat den Korridor erreicht und richtet sich an diesem aus. (c): Der Roboter hat die zweite Tür erreicht. An den Laserstrahlen ist zu sehen, dass die Tür verschlossen ist. (d). Der LiSA-Roboter hat die Tür als geschlossen erkannt und plant einen alternativen Pfad. (e): Der Roboter hat den alternativen Pfad als ebenfalls blockiert erkannt und informiert den Benutzer.

(vgl. Kapitel 4.3.1) informiert wird. Um die Situation zu lösen, müsste der Benutzer eine der Türen öffnen und dem Roboter dies wiederum durch Quittierung des Fehlers mitteilen. Daraufhin werden die Türen aus der Pfadplanungskarte entfernt, es wird ein neuer Pfad geplant und das Anfahren der Zielpose beginnt erneut.

Kapitel 7

Perzeption

Das LiSA-Szenario stellt Anforderungen an die Navigation des Roboters, die mit den im vorherigen Kapitel beschriebenen Verfahren nicht zu lösen sind. Dies sind z.B. spezielle parallele Anfahrmanöver an Übergabestationen oder der Wechsel zwischen parallelem bzw. tangentialem Fahrverhalten je nach Umgebungssituation. Damit der LiSA-Roboter in der Lage ist, diese Anforderungen zu erfüllen, wird eine rudimentäre Perzeption in das System integriert. Perzeption beschreibt hier die Fähigkeit des Roboters seine Umgebung interpretierend wahrzunehmen und die erhaltene Information in einen semantischen Kontext zu bringen. Es existieren verschiedene Ansätze, um dies für eine strukturierte Umgebung zu realisieren. So wird in [59] die dreidimensionale Gebäudegeometrie aus den Sensordaten extrahiert und mit semantischen Kategorien (Wand, Fussboden, Decke,...) versehen. Ferner werden häufig Kamera- oder Laserscannerdaten verwendet, um Menschen oder andere Objekte in der Umgebung des Roboters zu identifizieren [21]. In [54] wird eine hierarchische Struktur von verschiedenen Kartentypen verwendet. Über eine metrische Karte wird eine Ebene gelegt, die Navigationspunkte der Umgebung besitzt. Darüber wird eine topologische Karte gelegt, die Bereiche und deren Beziegungen zueinander in einem Graph definiert. Über die Topologische Karte wird eine konzeptionelle Schicht gelegt, die eine Klassenhierarchie der Räume und deren Verbindungen sowie von Objekten enthält und in der logisch geschlussfolgert werden kann, welche Aktionen für ein bestimmtes Ziel auszuführen sind.

7.1 Zonenkonzept zur semantisch gestützten Navigation

Im LiSA-Projekt wird ein zonenbasiertes Verfahren verwendet, um dem Roboter zu ermöglichen, die aktuelle Umgebungssituation zu erkennen. Dieses verwendet analog zu dem in [96] beschriebenen Verfahren Polygone, denen verschiedene Kategorien zugeordnet werden, die die Situation beschreiben, wenn sich der Roboter in dem vom Polygon begrenzten Gebiet befindet. Im LiSA-Projekt sind diese Zonen statisch und werden manuell definiert. Dies ver-

schiebt das Problem der Perzeption auf das Problem der Selbstlokalisierung des Roboters. Objekte in der Umgebung des Roboters müssen nicht anhand der Sensordaten identifiziert werden und der semantische Kontext des Roboters muss nicht auf die identifizierten Objekte zurückgeführt werden. Dies führt zu einer starken Vereinfachung des Ausgangsproblems, bringt jedoch auch Nachteile mit sich. Zum einen führt die manuelle Definition der Zonen dazu, dass das System relativ statisch ist und manuell an Umgebungsänderungen angepasst werden muss. Zum anderen ist das Erkennen einer Umgebungssituation immer mit der Ungenauigkeit der Lokalisierung behaftet; vermutet der Roboter durch fehlerhafte Lokalisierung seine Position in einer falschen Zone reagiert er somit auf eine falsche Umgebungssituation. Dies ist besonders gefährlich, wenn die Zonen Gefahrensituationen markieren (z.B. absteigende Treppen), in die der Roboter nicht hineinfahren darf (vgl. 7.1.4). Würde das Objekt, das die Gefahr hervorruft, direkt in den Sensordaten identifiziert, wäre diese Gefahr nicht vorhanden, da selbst bei fehlerhafter Selbstlokalisierung dem Objekt ausgewichen werden kann. Die direkte Identifizierung setzt jedoch eine Sensorkonfiguration voraus, mit deren Hilfe diese Gefährdungspotentiale erfasst werden und des Weiteren sicher identifiziert werden. Die sichere Identifizierung von beliebigen Umgebungssituationen (Korridor, Labor, Türbereich, Übergabebereich,...) anhand von Sensordaten ist jedoch nicht möglich, so dass die Gefahr der fehlerhaften Selbstlokalisierung gering gegenüber der Gefahr einer fehlerhaften Identifizierung der Situation anhand der Sensordaten ist. Zur Verhaltenssteuerung des LiSA-Roboters werden folgende Zonen verwendet:

Türregion: Zonen, in deren Mitte sich eine Tür befindet.

Türrahmen: Zonen, die einen Türrahmen repräsentieren.

Zielregion: Zonen, in denen sich die jeweiligen Zielposen des Roboters befinden.

Raum: Die einzelnen Räume, die zum Arbeitsbereich des LiSA-Roboters gehören werden durch Zonen eindeutig markiert. Dies wird zur Mensch-Roboter-Interaktion verwendet, um dem Menschen verständlich mitteilen zu können, wo sich der Roboter aktuell befindet.

Korridor: In einer Korridor-Region hat der Roboter eine Vorzugsorientierung, die parallel zum Korridor verläuft. Diese Orientierung wird auch bei Ausweichmanövern nicht aufgehoben.

Kurvenregion: In dieser Zonen nimmt der LiSA-Roboter ein Fahrverhalten an, bei dem die Hauptachse des Roboters immer tangential zum aktuellen Pfad verläuft.

Gefahrregion: In Zonen dieser Kategorie sollte der Roboter im normalen Betrieb nicht hineinfahren. Ein sofortiger Notstopp des Roboters erfolgt, falls dies doch der Fall ist.

Neben den verschiedenen Kategorien wird einer Zone ebenfalls eine Gruppe zugeordnet, so dass sich z.B. eine Tür und der Raum, zu dem die Tür gehört, in derselben Gruppe befinden. Dies ermöglicht z.B. Fehlermeldungen wie: „Die Tür zum Raum R502 ist geschlossen und der Pfad kann nicht fortgesetzt werden“.

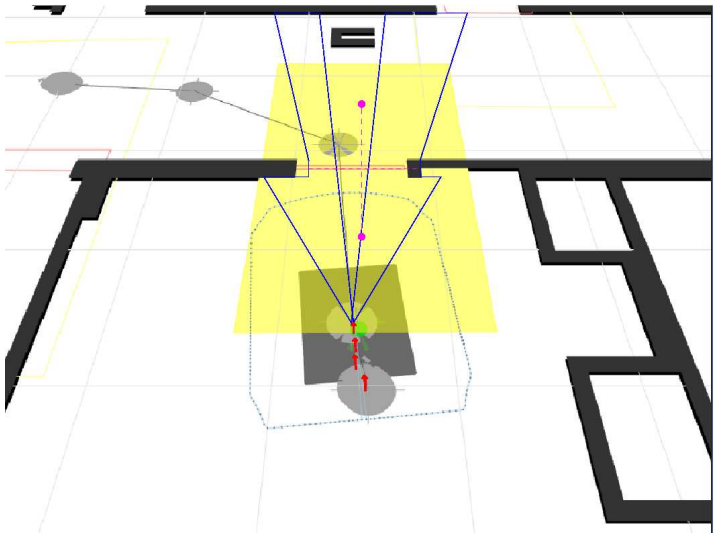
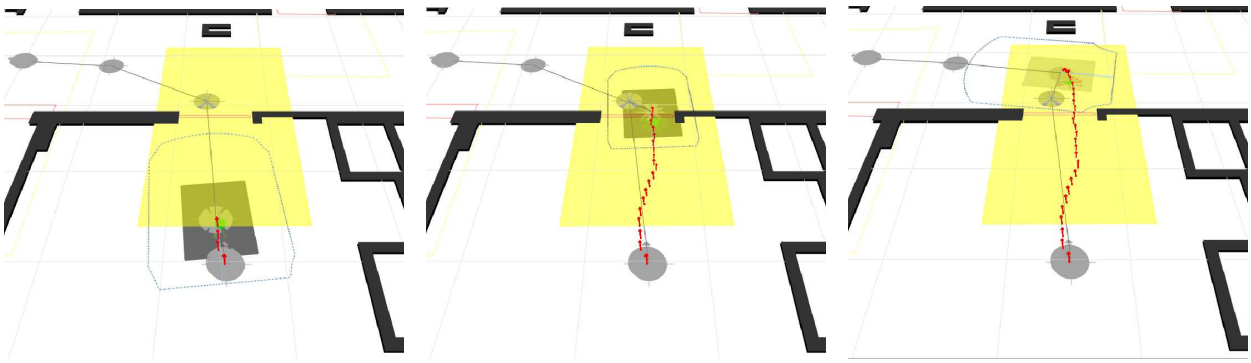


Abbildung 7.1: Türdurchfahrtverhalten des LiSA-Roboters: Es wird iterativ im Bereich der Tür nach Entfernungssprüngen in den Laserscannerdaten gesucht. Anhand dieser werden zwei Zwischenzielposen bestimmt (magenta), die auf einer Geraden liegen (magenta gestrichelt), die senkrecht zur Tür (rot) liegt und diese mittig schneidet.

7.1.1 Türdurchfahrtverhalten des LiSA-Roboters

Ein spezielles Fahrverhalten zur Durchfahrt einer Tür ist notwendig, da der LiSA-Roboter mit einer Breite von 60 cm bei einer Türbreite von 75 cm nur einen Sicherheitsabstand von 7,5 cm zum Türrahmen hat. Untersuchungen mit Hilfe der Simulationsumgebung und dem realen Roboter ergeben, dass dieser minimale Sicherheitsabstand bei einfacher Anwendung der klassischen Hindernisvermeidungsverfahren (VFH und ND) zum Verkanten des LiSA-Roboters in der Tür, bzw. zu einem oszillierendem Fahrverhalten im Türbereich führen kann. Das daraus resultierende Blockieren des Türbereichs ist aus sicherheitstechnischen Gründen nicht zu tolerieren.

Um eine sichere Türdurchfahrt zu gewährleisten, wird der LiSA-Roboter zunächst senkrecht zur Tür ausgerichtet und anschließend sicher durch diese geführt. Hierzu wird jede Tür mit einer Türrahmenregion markiert. Ferner wird ein Bereich von etwa 2 m um die Tür herum als Türregion definiert. Fährt der Roboter in eine dieser Türregionen hinein, wird zunächst geprüft, ob der verbleibende Pfad die zu der Türregion gehörenden Türrahmenregion schneidet. Ist dies der Fall, wird ein Türdurchfahrtverhalten gestartet. Dieses berechnet anhand der aktuellen Roboterpose und der Türrahmenregion die Koordinaten der rechten bzw. linken Seite des Türrahmens. In der Mitte, der durch die Türrahmenkoordinaten definierten Strecke, wird eine Senkrechte definiert. Auf dieser Senkrechten werden zwei Punkte im Abstand von 0,8 m zum Türrahmen bestimmt. Diese Positionen bilden die Ausrichtungsposition des Roboters, die er einnimmt, um sich senkrecht vor der Tür zu positionieren und den Zielpunkt, den der Roboter anfahren soll, nachdem er sich ausgerichtet hat. Die Orientierung, die der Roboter in beiden Positionen hat, entspricht der senkrecht zum Türrahmen projizierten



(a) (b) (c)

Abbildung 7.2: Dynamisches Anpassen der Sicherheitsabstände beim Türdurchfahrtverhalten: (a) Der Roboter beginnt mit dem Türdurchfahrtverhalten und setzt den Sicherheitsabstand (blau) herab, um die Ausrichtungspose erreichen zu können. (b) Der Roboter hat die Ausrichtungspose erreicht und durchfährt die Tür. Dazu werden der Sicherheitsabstand und die Maximalgeschwindigkeit weiter herab gesetzt. (c) Der Roboter hat die Zielpose erreicht. Der Sicherheitsabstand und die Maximalgeschwindigkeit werden auf die Standardwerte zurückgesetzt und die Kontrolle wird an das Pfadabfahrmodul zurückgegeben.

Geraden.

Wird die Ausrichtungspose, wie oben beschrieben, berechnet, wäre sie mit der Ungenauigkeit der Lokalisierung behaftet. Zusätzlich wiese diese Ausrichtungspose den Fehler auf, der potentiell zwischen dem realen Türrahmen und der manuell erstellten Türrahmenregion besteht. Experimente zeigen, dass dieser Fehler im Allgemeinen zu groß ist. Vor allem der Orientierungsfehler der Lokalisierung führt dazu, dass der Zielpunkt der Türdurchfahrt nicht direkt mittig hinter der Tür liegt, sondern seitlich versetzt ist, was zu einer schrägen Türdurchfahrt und einem Verkanten des Roboters führt.

Um dies zu vermeiden, wird das Verfahren dahingehend erweitert, dass die Türöffnung aktiv in den aktuellen Laserscannerdaten gesucht wird. Die berechneten Koordinaten der rechten bzw. linken Seite des Türrahmens im lokalen Roboterkoordinatensystem werden hierbei als Schätzung verwendet, um den zu durchfahrenden Türrahmen von anderen Türrahmen unterscheiden zu können, die potentiell ebenfalls in den Laserscannerdaten vorhanden sind. Wie in Abbildung 7.1 zu sehen ist, werden zunächst die Winkel bestimmt, unter denen der Türrahmen in den Laserscannerdaten zu sehen sein sollte. Anschließend wird in den Laserscannerdaten nach Tiefensprüngen innerhalb eines definierten Bereichs um diese Winkel gesucht. Diese Tiefensprünge definieren die korrigierte rechte bzw. linke Seite des Türrahmens, die anschließend zur Berechnung der Ausrichtungs- und der Zielpose verwendet werden. Um sicherzustellen, dass der Roboter die Ausrichtungspose exakt anfährt, wird das oben beschriebene Verfahren iterativ durchgeführt. Sobald neue Lokalisierungsdaten vorhanden sind, werden diese mit den aktuellen Laserscannerdaten dazu verwendet, die Ausrichtungs- und die Zielpose zu aktualisieren. Dies wird solange wiederholt, bis die Differenz zwischen der Lokalisierungspose und der berechneten Ausrichtungspose unter einem definierten Schwell-

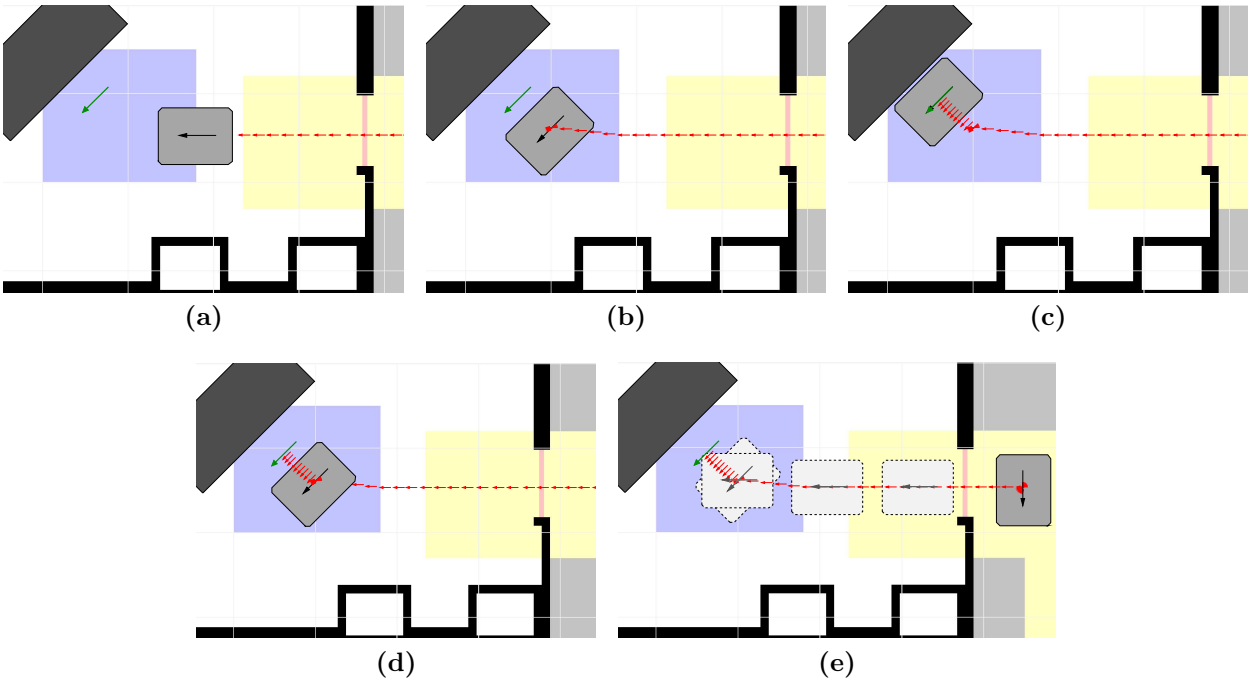


Abbildung 7.3: Fahrverhalten beim Anfahren und Verlassen einer Arbeitsstation: (a) Der Roboter betritt eine Zielzone (blaues Viereck) und beginnt mit einem Zielzonenfahrverhalten, da sich die Zielpose (grüner Pfeil) innerhalb der Zielzone befindet. (b) Der Roboter fährt einen Zwischenzielpunkt an und richtet sich entsprechend der Zielpose aus. (c) Der Roboter fährt die Zielpose parallel an. (d) Nach Beendigung der Manipulation an der Arbeitsstation fährt der Roboter erneut die Zwischenzielpose an und verlässt das Labor in derselben Orientierung, mit der er in das Labor hinein gefahren ist (e).

wert liegt. Nach dem Erreichen der Ausrichtungspose ist der Roboter senkrecht zur Tür ausgerichtet. Durch das Anfahren der direkt vor dem Roboter liegenden Zielpose gelangt der Roboter sicher durch die Tür. Wird die Zielpose erreicht, wird das Türdurchfahrtverhalten beendet und die Kontrolle wieder an das Pfadabfahrmodul übergeben.

Wie in Abbildung 7.2 zu sehen ist, setzt das Türdurchfahrverhalten neben den verschiedenen Zielposen ebenfalls den Sicherheitsabstand und den Ausweichabstand des Roboters. So wird der minimale Abstand für ein Hindernis, dem der Roboter ausweicht, auf 15 cm heruntergesetzt, wenn er die Ausrichtungspose anfährt. Dies ist notwendig, damit er den Ausrichtungspunkt auch erreichen kann und nicht der Wand ausweicht, in der sich die Tür befindet. Nach dem Erreichen der Ausrichtungspose wird der seitliche Sicherheitsabstand auf 2 cm gesetzt, so dass bei einer Türbreite von 75 cm der Roboter 3 cm auf jeder Seite zum Navigieren hat. Ferner wird der Ausweichabstand so eingestellt, dass sich beide Türrahmenseiten im Ausweichbereich befinden. Würde jeweils nur eine Türrahmenseite im Ausweichbereich liegen, käme es beim Durchfahren der Tür zu einer ständigen Ausweichbewegung des Roboters, was die Gefahr einer sich aufschaukelnden Oszillationsbewegung birgt. Hat der Roboter die Tür erfolgreich durchfahren, werden der Ausweich- sowie der Sicherheitsabstand auf die Ausgangswerte zurückgesetzt. Um eine sichere Türdurchfahrt zu gewährleisten, werden die

maximal erlaubten Translations- und Rotationsgeschwindigkeiten ebenfalls vom Türdurchfahrtverhalten an die jeweilige Situation angepasst.

7.1.2 Fahrverhalten des LiSA-Roboters im Bereich einer Arbeitsstation

Ein spezielles Fahrverhalten zum Anfahren einer Arbeits- bzw. Übergabestation ist notwendig, da der befahrbare Bereich begrenzt ist. Zum einen muss der Roboter relativ dicht an der Arbeitsstation stehen, da die Manipulationsreichweite begrenzt ist. Dies schließt eine Orientierungsausrichtung in der Zielposition aus, da der Roboter nicht genug Platz hat, um auf der Stelle zu drehen, so dass sich der Roboter der Arbeitsstation parallel nähern muss. Zum anderen ist der gesamte Bereich in den Laboren, in denen sich die Arbeitsstationen im LiSA-Szenario befinden, so eng, dass eine Rotation auf der Stelle nicht möglich ist (vgl. Kapitel 2.1). Eine Rotation auf der Stelle würde den gesamten Durchgangsbereich blockieren und somit eine Behinderung und Gefährdung des Personals zur Folge haben. Da eine Rotation auf der Stelle nicht möglich ist, muss der LiSA-Roboter innerhalb des Labors immer die selbe Orientierung haben. Der Roboter passt seine Orientierung nur lokal im Bereich der Arbeitsstation an diese an. Insbesondere muss der LiSA-Roboter das Labor mit derselben Orientierung verlassen, mit der er in das Labor hinein gefahren ist.

Um dieses Fahrverhalten zu erreichen, wird um jede Arbeitsstation eine Zielzone definiert. Das spezielle Zielfahrverhalten wird angestoßen, wenn der Roboter in eine solche Zielzone eintritt und der aktuelle Pfad in dieser Zielzone endet. Abbildung 7.3 zeigt die verschiedenen Schritte des Anfahrens der Arbeitsstation. In Teilabbildung (a) tritt der Roboter in die Zielzone ein. Da sich die durch den grünen Pfeil symbolisierte Zielpose innerhalb der Zielzone befindet, wird das Zielfahrverhalten gestartet. Es wird eine Zwischenzielpose berechnet, die dieselbe Orientierung wie die Zielpose besitzt, jedoch weit genug von der Arbeitsstation entfernt ist, um sich parallel zur Zielpose ausrichten zu können. Nachdem sich der Roboter zur Zielpose ausgerichtet hat, fährt er die Arbeitsstation parallel an. Dies wird durch die omnidirektionale Kinematik des LiSA-Roboters ermöglicht. Hat der Roboter die Zielpose erreicht, ist der Navigationsauftrag beendet und der Roboter beginnt mit der Manipulation an der Arbeitsstation (Teilabbildung (c)). Nach Beendigung der Manipulation bekommt der LiSA-Roboter einen neuen Fahrauftrag. Zunächst fährt der Roboter erneut parallel die Zwischenzielposition an, um genügend Platz für eine Orientierungsänderung zu haben. Dort angekommen nimmt er die Orientierung an, die er vor der Anfahrt der Zielpose hatte (Teilabbildung 7.3d). In dieser Orientierung verlässt der LiSA-Roboter das Labor. Das spezielle Fahrverhalten wird beendet, nachdem das Türdurchfahrtverhalten des Roboters abgeschlossen ist. Das Zielzonenverhalten kann nicht mit Beginn des Türzonenverhaltens beendet werden, da das Ausrichten vor der Tür bei einem normalen Türzonenverhalten für eine 180°-Drehung des Roboters sorgen würde. Das Zielzonenverhalten sorgt dafür, dass das Türfahrverhalten den Roboter um eine um π gedrehte Pose ausrichtet.

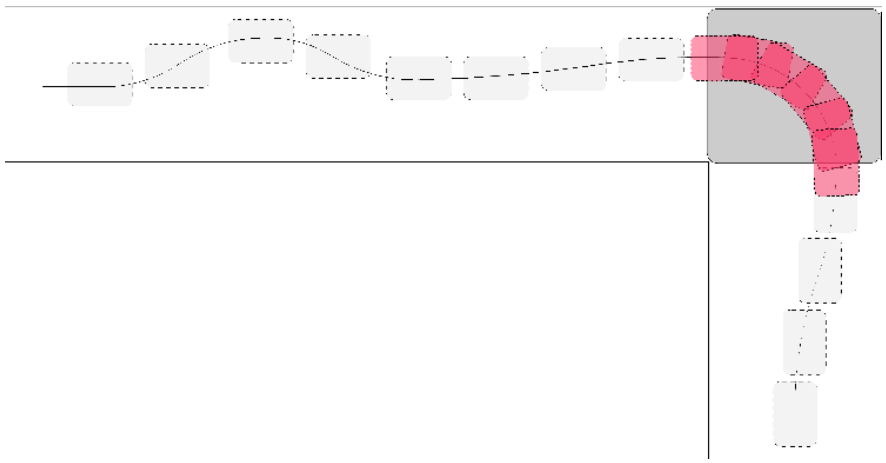


Abbildung 7.4: Der LiSA-Roboter ist immer parallel zum Korridor ausgerichtet, in dem er sich gerade befindet. Damit dies gewährleistet ist, wenn zwei Korridore unter einem Winkel aufeinandertreffen, wird zwischen den Korridoren eine Zone definiert, in der der Roboter seine Orientierung tangential dem Pfad anpasst.

7.1.3 Orientierungskontrolle des Roboters in Korridorbereichen

Wie in Abbildung 7.4 zu sehen ist, fährt der LiSA-Roboter in Korridoren innerhalb der Arbeitsumgebung im parallelen Modus (vgl. Kapitel 4.2.2). Dies ist notwendig, da die Korridore im Anwendungsszenario relativ schmal sind, so dass die Gefahr besteht, dass der LiSA-Roboter den Korridor im Notfall zu stark blockiert, wenn er darin schräg ausgerichtet fährt. Hierzu wird der Bereich, der zu einem Korridor gehört, durch eine entsprechende Zone markiert. Befindet sich der Roboter nach einer Türdurchfahrt (vgl. 7.1.1) in einer Korridorzone, wird der Roboter zunächst in Richtung des Korridors ausgerichtet, bevor die Kontrolle des Roboters wieder dem Pfadabfahrmodul übergeben wird.

Da der globale Pfadplaner nur Zwischenzielpositionen und keine Posen bestimmt, müssen Regionen, in denen der Roboter seine Orientierung ändern soll, speziell markiert werden (siehe Abbildung 7.4 grauer Bereich). Dies ist z.B. der Fall, wenn zwei Korridorzonen im rechten Winkel aufeinandertreffen. Damit der LiSA-Roboter den Korridor, der senkrecht zur aktuellen Pose des Roboters verläuft, nicht quer durchfährt, muss er in der Region, in der die beiden Korridore aufeinandertreffen, eine Orientierung annehmen, die tangential zum Pfad verläuft.

Der LiSA-Roboter besitzt eine hardwarebedingten Orientierungsdrift, die durch nicht exakt gleiche Geschwindigkeiten, bzw. durch eine minimale Winkeldifferenz der beiden Räder entsteht. Diese Drift hat zur Folge, dass der Roboter bei längerer Fahrt im parallelen Modus in einem Korridor seine Ausrichtung zu diesem verliert und schräg zum Verlauf des Korridors fährt. Da diese Fahrweise, wie schon erwähnt, eine Gefährdung darstellt, muss die Drift erkannt und ausgeglichen werden. Hierzu wird die aktuelle Orientierung des Roboters mit der Ausrichtung der Korridorzone verglichen, solange sich der Roboter darin befindet.

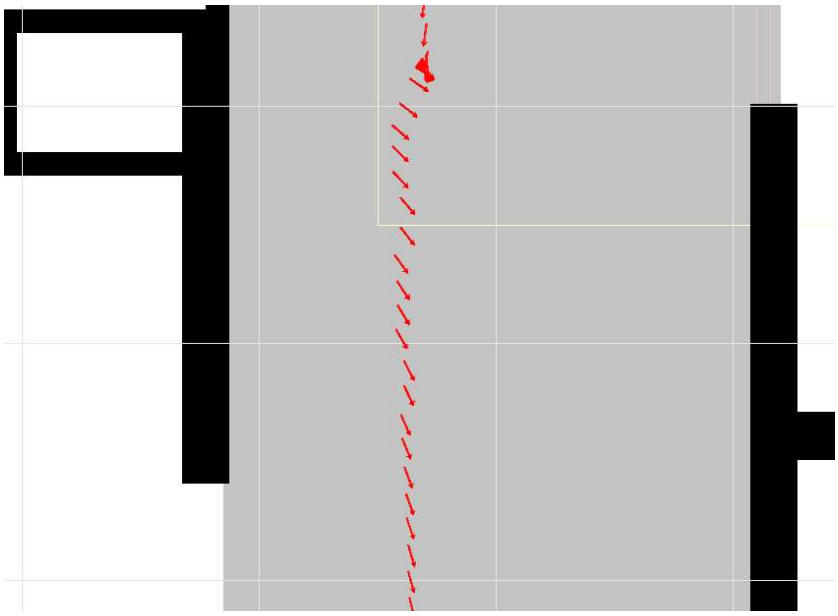


Abbildung 7.5: Anpassung der Orientierung des LiSA-Roboters parallel zum Korridor. Hat der Roboter eine zur Hauptachse des Korridors verschiedene Orientierung (hier manuell erzeugt), wird die Bewegungsrichtung des Roboters mit einer Rotationsbewegung überlagert, die die Winkeldifferenz zwischen Korridor und Roboterorientierung ausgleicht.

Die parallele Bewegung des Roboters im Korridor wird mit einer Rotationsbewegung überlagert, die der Drift entgegenwirkt. Die Geschwindigkeit der Rotationsbewegung ist abhängig von der Winkeldifferenz zwischen Orientierung des Roboters und Orientierung des Korridors. Abbildung 7.5 zeigt ein Experiment, das die Orientierungsanpassung an den Korridor verdeutlicht. Die roten Pfeile repräsentieren Posen auf dem Pfad des Roboters. Um die Orientierungsanpassung zu verdeutlichen, wird die Orientierungsdrift des Roboters durch eine manuelle Rotation des Roboters um 45° simuliert. Es ist zu sehen, dass der Roboter diese Drift ausgleicht, während er seine Bewegung fortsetzt. Auch die Abhängigkeit der Rotationsgeschwindigkeit von der Winkeldifferenz ist zu erkennen. Die Orientierungsänderung des Roboters ist nach der manuellen Ausrichtung am größten und nimmt mit sinkender Winkeldifferenz des Roboters zum Korridor kontinuierlich ab.

7.1.4 Vermeidung von Gefahrenbereichen

Der LiSA-Roboter ist in seinem Arbeitsumfeld einer Reihe von Gefahren ausgesetzt, die sich immer in derselben Region befinden. Zu diesen Objekten gehören z.B. Türen. So stellt eine geschlossene Tür eine potentielle Gefahr dar, da der Roboter einer sich plötzlich öffnenden Tür nicht ausweichen kann. Neben Türen bilden absteigende Treppen eine Gefahr für den LiSA-Roboter, da die verwendete Sensorkonfiguration dafür sorgt, dass der Roboter alle unter der Scanebene der SICK-Laserscanner liegenden Hindernisse nicht wahrnimmt (vgl. Kapitel 2.2.2). Absteigende Treppen sind zwar im eigentlichen Anwendungsszenario

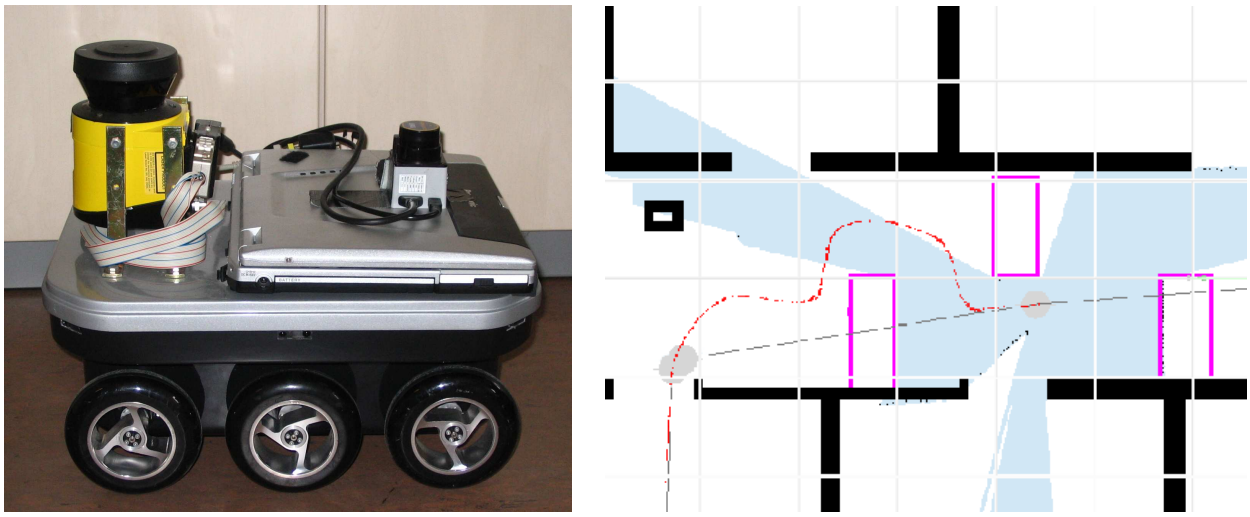


Abbildung 7.6: Integration von Gefahrenbereichen in die Hindernisvermeidung: Links: Der Roboter Kurt2D ausgestattet mit einem SICK *S*300-Laserscanner und einem Hokuyo-Laserscanner. Rechts: Der VRS-Datensatz (blau) enthält die Information der beiden Laserscanner und der Gefahrenbereiche (magenta). Obwohl der globale Pfad (graue Linie) die Gefahrenregion kreuzt, navigiert der Roboter um diese herum (roter Pfad).

ausgeschlossen; dennoch ist es nicht ausgeschlossen, dass sich Treppen oder vergleichbare Hindernisse bei zukünftigen Szenarien im Arbeitsumfeld des Roboters befinden. Aus diesem Grund werden sie in dieser Arbeit ebenfalls berücksichtigt.

Um zu vermeiden, dass ein Roboter in einen als gefährlich eingestuften Bereich hineinfährt, muss er diese Bereiche zunächst identifizieren. Dies kann entweder direkt geschehen, indem in den Sensordaten nach Merkmalen gesucht wird, die den Gefahrenbereich identifizieren. Z.B. können Türen in Kamera- oder Laserscannerdaten erkannt werden. In dieser Arbeit werden die Gefährdungsbereiche manuell durch Zonen mit entsprechendem Label markiert. Problematisch an diesem Ansatz ist, dass das Wahrnehmen der Gefahrenbereiche mit der Unsicherheit der Lokalisierung behaftet ist. Dies schließt bei falscher Selbstlokalisierung des Roboters nicht aus, dass der Roboter z.B. in den Bereich der absteigenden Treppe fährt. Auf der anderen Seite ist es jedoch oft, aufgrund der Sensorkonfiguration oder der Beschaffenheit der Gefahrenbereiche, nicht möglich diese zuverlässig in den Sensordaten zu identifizieren.

Die Gefahrenzonen lassen sich in die Roboterkontrollarchitektur integrieren, indem die Semantik in einem globalen Pfadplaner berücksichtigt wird. Dieser verwendet beim Planen des globalen Pfades die Gefahrenregionen und plant einen Pfad, der diese nicht kreuzt. Das Problem dieses Ansatzes ist, dass eine Integration der Gefahrenbereiche in die globale Pfadplanung nicht garantiert, dass der Roboter, wenn er sich in der Nähe einer Gefahrenregion befindet, nicht in diese hinein fährt. Dies kann z.B. der Fall sein, wenn der globale Pfad blockiert ist und der Roboter ausweichen muss. Da sich die Hindernisvermeidung der Gefahrenregionen nicht bewusst ist, fährt der Roboter potentiell in diese hinein.

Hier setzt der in dieser Arbeit entwickelte Ansatz an. Indem das VRS-Konzept verwendet wird (vgl. Kapitel 6.2.2), werden die Gefahrenbereiche direkt in den Hindernisvermeidungsprozess integriert. Hierzu werden die Gefahrenbereiche in die Rasterkarte eingefügt, die beim VRS-Verfahren als Hindernisgedächtnis dient. Die Gefahrenbereiche werden durch das Ray-Tracing-Verfahren auf der Hinderniskarte parallel zu den andern Sensoren in den VRS-Datensatz integriert. Dadurch sind die Gefahrenbereiche als virtuelle Hindernisse in den VRS-Daten vorhanden. Der Alterungsprozess, der auf der Hinderniskarte läuft, wird auf die Gefahrenbereiche nicht angewendet. Die Gefahrenbereiche werden somit als statisch klassifiziert und automatisch in die globale Pfadplanung integriert (vgl. Kapitel 6.3.2). Nachteilig wirkt sich bei diesem Ansatz aus, dass nur zwei verschiedene Bereiche existieren: Gefahrenbereich und befahrbarer Bereich. Abstufungen zwischen diesen Bereichen lassen sich nicht realisieren. Dies wäre zum Beispiel ein Bereich, der potentiell gefährlich ist, in den der Roboter aber kurzzeitig ausweichen darf.

Abbildung 7.6 zeigt das Ergebnis eines Experiments zur Vermeidung der Gefahrenbereiche. In der linken Abbildung ist der Roboter zu sehen, mit dem dieses Experiment durchgeführt wurde. Dieser ist differential angetrieben und mit einem nach vorne gerichteten SICK-Laserscanner und einem nach hinten ausgerichteten Hokuyo-Laserscanner ausgerüstet. Die rechte Abbildung zeigt den VRS-Datensatz (blau), der die Sensordaten der beiden Laserscanner integriert. Neben den beiden Laserscannern werden die Gefahrenbereiche (magenta) über die Hinderniskarte des VRS integriert und sind somit im VRS-Datensatz als virtuelle Hindernisse vorhanden. Obwohl der globale Pfad (graue Linie) den Gefahrenbereich kreuzt, navigiert das Hindernisvermeidungsmodul, welches den VRS-Datensatz verwendet, den Roboter um dieses herum (roter Pfad). Die Übernahme der Gefahrenbereiche in den globalen Pfadplaner wurde bei diesem Experiment abgeschaltet, um die lokale Pfadanpassung an die Gefahrenbereiche zu verdeutlichen.

7.1.5 Kombination der Playermodule zum zonengesteuerten Verhalten

Um das zonengesteuerte Fahrverhalten zu realisieren, wird ein „ZoneControler“-Device zwischen das Pfadabfahr-Device und das Hindernisvermeidungs-Device („ND“) geschoben. Wie in Abbildung 7.7 gezeigt, erhält dieses vom „ZoneAwareness“-Device die aktuelle Zone, in der sich der Roboter befindet. Der ZoneControler reicht die Pfadpunkte, die er vom Pfadabfahr-Device erhält, im Normalbetrieb weiter an das Hindernisvermeidungs-Device. Im Türdurchfahrtverhalten bzw. Zielanfahrtsverhalten werden die Pfadpunkte wie beschrieben angepasst und an das Hindernisvermeidungs-Device weitergeleitet. Ferner setzt das „ZoneControler“-Device die Maximalgeschwindigkeiten, Fahrmodi und den Ausweich- bzw. Sicherheitsabstand. Das Hindernisvermeidungsdevice integriert die VRS-Daten und ermittelt die optimale Bewegungsrichtung sowie die optimale Kombination aus Translations- und Rotationsgeschwindigkeit. Diese werden an das „Fox“-Device weitergegeben, welches das Kinematische Modell des Roboters enthält und die ermittelten Radorientierungen und Radgeschwindigkeiten an die SPS des Roboters weiterleitet.

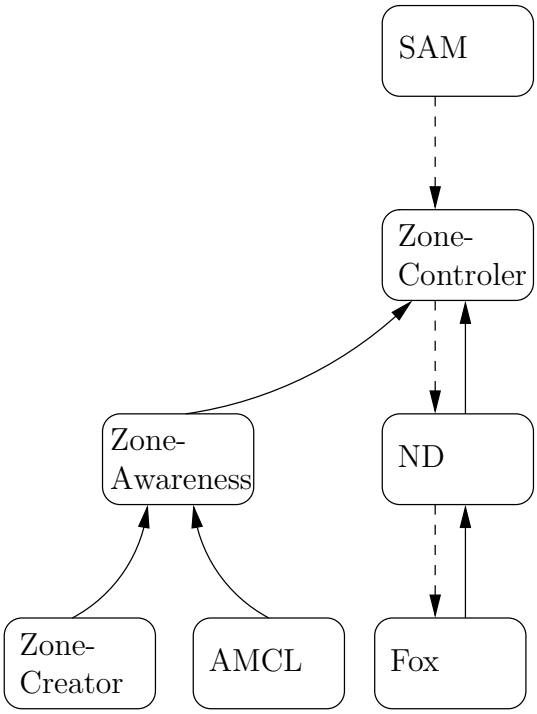


Abbildung 7.7: Kombination der Player-Module zum zonengesteuerten Fahrverhalten des LiSA-Roboters. Erläuterungen siehe Text.

7.2 Bewegungsdetektion und Verfolgung

In einer dynamischen Umgebung ist es wichtig für einen autonomen Roboter, andere dynamische Objekte lokalisieren zu können, da von dynamischen Objekten immer eine potentielle Gefahr ausgeht. Anders herum werden Bewegungen im Roboterumfeld oft durch Menschen hervorgerufen, für die der Roboter eine Gefahr darstellt.

Neben der Erkennung von dynamischen Objekten ist es ebenfalls wichtig, dass der Roboter die einzelnen Objekte verfolgen, bzw. deren zukünftige Bewegung voraussagen kann.

7.2.1 Bewegungsdetektion

Die Bewegungsdetektion bestimmt mittels Sensordaten, ob sich dynamische Objekte im Umfeld des Roboters befinden. Es werden verschiedene Abstandssensoren und Kameras eingesetzt. Bei einem statischen Sensor lässt sich die Bewegung über einfache Differenzbildung aufeinanderfolgender Sensordaten ermitteln. So können Unterschiede in den Abstandswerten eines Laserscanners oder in RGB-Werten einer Kamera nur durch dynamische und nicht durch statische Hindernisse hervorgerufen werden. Schwieriger wird das Problem, wenn der Sensor auf einem mobilen Roboter angebracht ist. Hier muss ermittelt werden, ob die Unterschiede in den Sensordaten durch die Eigenbewegung des Roboters oder durch dynamische

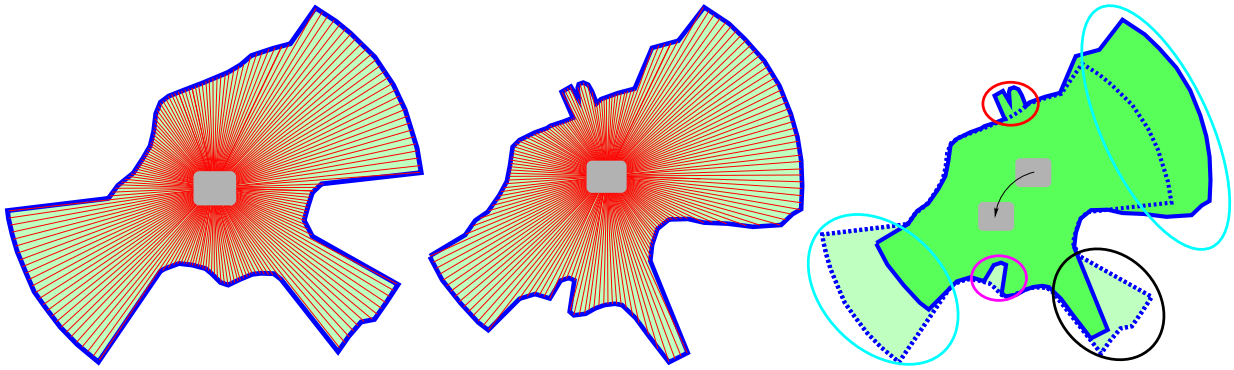


Abbildung 7.8: Schematische Darstellung der Bewegungsdetektion: (a) Anhand der Laserdaten (rot) wird die freie Fläche im Roboterumfeld bestimmt (grün). (b) Die freie Fläche wird für eine weitere Roboterpose bestimmt. (c) Die freien Flächen werden anhand der Odometrie in ein gemeinsames Koordinatensystem transformiert (schwarzer Pfeil). Der Vergleich der freien Flächen ergibt potentielle Bewegungen im Roboterumfeld (Ellipsen).

Objekte im Umfeld des Roboters hervorgerufen werden. Um das Problem zu lösen, muss die Eigenbewegung des Roboters herausgerechnet werden, indem die Sensordaten anhand der aktuellen Roboterpose in ein globales, stationäres Koordinatensystem transformiert werden. Aus diesem Grund ist eine sichere Selbstlokalisierung entscheidend für die Performanz der Bewegungsdetektion. Da Lokalisierungsverfahren häufig rechenintensiv sind und sie die Roboterpose daher nicht kontinuierlich aktualisieren können (vgl. Kapitel 6.1), wird häufig die von der Odometrie ermittelte Pose (vgl. Kapitel 4.2.1) als Referenzsystem verwendet.

In dieser Arbeit wird eine in den Player-Framework portierte Implementierung der Bewegungsdetektion verwendet, die von der „cure“-Bibliothek zur Verfügung gestellt wird [61]. Diese orientiert sich an dem in [92] vorgestellten Verfahren. Als Eingangsdaten zur Bewegungsdetektion werden Laserscannerdaten verwendet. In diese Daten wird eine Fläche gefittet, die den freien Raum des Datensatzes repräsentiert. In Abbildung 7.8 sind diese Flächen in Teilabbildung (a) und (b) für zwei Beispielscans mit unterschiedlicher Roboterpose in derselben Umgebung dargestellt. Teilabbildung (c) zeigt, wie die Fläche des Laserscans der zweiten Roboterpose anhand der Odometrie auf die erste Fläche abgebildet wird. Die Ellipsen markieren Bereiche, in denen die beiden Flächen nicht zur Deckung kommen, was ein Indiz für dynamische Objekte ist. Die hellblauen Ellipsen markieren Bereiche, bei denen die fehlende Deckung der Flächen durch die maximale Reichweite des Laserscanners hervorgerufen wird. Die Berücksichtigung dieser Bereiche lässt sich leicht durch Setzen eines Schwellwerts für die maximale Entfernung, bei der ein dynamisches Objekt noch detektiert wird, unterbinden. In der schwarzen Ellipse kommen die beiden Flächen nicht zur Deckung, da der Bereich in der zweiten Roboterpose im Laserschatten anderer Objekte der Umgebung liegt. Dieser Bereich würde fälschlicherweise als dynamisches Objekt klassifiziert werden. Jedoch ist die Distanz zwischen den beiden Roboterposen in Abbildung 7.8 sehr groß. Da ein typischer Laserscanner mit einer Frequenz von 75 Hz sendet, ist die Posedifferenz zwischen aufeinanderfolgenden

Laserdaten im Allgemeinen sehr klein. Somit kommt es zu kontinuierlichen Übergängen der betrachteten Flächen innerhalb der schwarzen Ellipse. Teilabbildung (c) zeigt zwei Fälle, die nur durch dynamische Objekte hervorgerufen werden können. In der roten Ellipse ist die Fläche des zweiten Scans im Vergleich zum ersten Scan größer geworden, obwohl keine Verdeckung durch sonstige Objekte der Szene besteht. Dieser Fall kann also nur durch ein sich vom Roboter entfernendes, dynamisches Objekt hervorgerufen werden. Der entgegengesetzte Fall ist in der magentafarbenen Ellipse zu sehen. Hier wird die Fläche kleiner, obwohl der Bereich direkt einsehbar ist. Dies kann nur aus einer auf den Roboter hin gerichteten Bewegung eines dynamischen Objekts resultieren.

Um das Verfahren robust gegenüber Sensorrauschen und Odometriefehlern zu machen, wird um den Rand der freien Fläche (blau) ein Bereich definiert, den die Flächenänderung überschreiten muss, um berücksichtigt zu werden.

7.2.2 Bewegungsverfolgung

Zur Bewegungsverfolgung werden in der mobilen Robotik vor allem zwei Verfahren eingesetzt. Dies sind zum einen auf dem Partikelfilter basierte Verfahren [74, 75, 53] und zum anderen auf dem Kalman-Filter basierte Verfahren [21, 11]. Partikelfilter basierte Verfahren können mehrere Hypothesen gleichzeitig aufrechterhalten und verfolgen, während der auf der unimodalen Gaußfunktion basierende Kalman-Filter ausschließlich die Wahrscheinlichkeit für eine Hypothese bestimmt (vgl. Kapitel 6.1.2). Da dies für Tracking-Probleme wie das der Bewegungsverfolgung ausreichend ist, wird der Kalman-Filter häufig eingesetzt. In dieser Arbeit wird die Implementierung der Bewegunggsverfolgung aus dem „cure“-Framework in den Player-Framework portiert [61]. Diese Implementierung verwendet den Kalman-Filter. Somit verwendet jedes gefundene dynamische Objekt seinen eigenen Kalman-Filter. Als Sensor der Bewegungsverfolgung wird die im vorherigen Abschnitt beschriebene Bewegungsdetektion verwendet. Durch die Bewegungsdetektion in aufeinanderfolgenden Laserscanner-Datensätzen lässt sich die Geschwindigkeit und Richtung der Bewegung bestimmen. Diese ermöglicht es über das Bewegungsmodell des Kalman-Filters eine Vorhersage zu treffen, wo das dynamische Objekt im nächsten Schritt zu detektieren sein müsste. Diese Vorhersage ermöglicht es, sich kreuzende dynamische Objekte zu verfolgen. Ferner lässt sich ein dynamisches Objekt verfolgen, selbst wenn keine Dynamik im Bereich des Objekts detektiert wurde. Dies ist z.B. bei einem stehen gebliebenen Menschen der Fall. Bei fehlender Detektion von Dynamik und fehlenden Messungen, die die Anwesenheit eines Objekts in dem vom Kalman-Filter bestimmten Bereich stützen, wird die Varianz der Gaußfunktion, die die Pose des dynamischen Objekts angibt, immer größer. Übersteigt die Varianz einen definierten Schwellwert, wird das dynamische Objekt verworfen. Ferner besitzt die in dieser Arbeit verwendete Implementierung die Möglichkeit, eine Hypothese zu verwerfen, obwohl sie durch Messungen gestützt wird, aber für einen definierten Zeitraum keine Dynamik detektiert worden ist.

Die Bewegungsdetektion bestimmt ausschließlich, wo sich ein dynamisches Objekt befindet

und in welche Richtung es sich mit welcher Geschwindigkeit bewegt. Um dem Objekt ein semantisches Label (Mensch, Roboter, Tür,...) zuzuordnen, müssten die Sensordaten, die von der Bewegungsdetektion als dynamisches Objekt identifiziert worden sind, von einem weiteren Modul klassifiziert werden.

Kapitel 8

Zusammenfassung und Ausblick

Die vorliegende Arbeit behandelt die flexible und effiziente Integration von 3D-Sensordaten in den Hindernisvermeidungs- und Lokalisierungsprozess eines autonomen mobilen Roboters. Obwohl die Welt dreidimensional ist, werden im Bereich der autonomen mobilen Robotik in Industrie und Forschung oft nur zwei Dimensionen berücksichtigt. Die Höhe des Roboters wird bei der Pfadplanung, Hindernisvermeidung und Lokalisierung oft ignoriert. Dies liegt vor allem daran, dass sich dadurch das Ausgangsproblem der autonomen Navigation und die dazu eingesetzten Algorithmen stark vereinfachen. Insbesondere im Bereich der Hindernisvermeidung bildet diese Tatsache jedoch ein enormes Gefährdungspotential für Menschen im Umfeld des Roboters. Es existieren zwar einzelne Systeme, die über 3D-Sensorkonfigurationen verfügen und diese zur Hindernisvermeidung einsetzen. Jedoch sind die Verfahren speziell auf die verwendete Hardware zugeschnitten und nur mit dieser einsetzbar.

Mit dem in dieser Arbeit entwickeltem Virtual Range Scans (VRS)-Verfahren wird diese Lücke geschlossen. Es kann im Gegensatz zu bestehenden Verfahren flexibel beliebige Konfigurationen von Abstandssensoren in den Hindernisvermeidungsprozess integrieren. Ferner ist es in der Lage, bestehende zweidimensionale Hindernisvermeidungsstrategien wiederzuverwenden und diese flexibel auszutauschen. Das VRS-Konzept verfügt über ein Hindernisgedächtnis und berücksichtigt gleichzeitig die Dynamik in der Umgebung des Roboters. Diese Arbeit konnte experimentell belegen, dass dieses Konzept unter Verwendung von verschiedenen Sensorkonfigurationen und 2D-Hindernisvermeidungsstrategien eine sichere Hindernisvermeidung in allen drei Dimensionen realisiert.

Eine der größten Schwierigkeiten beim Arbeiten mit 3D-Daten in der Robotik ist die Rechenkomplexität in Kombination mit der Echtzeitanforderung vieler Systeme. Im Bereich der Lokalisierung wurde dieses Problem in dieser Arbeit durch die Verwendung effizienter Datenstrukturen gelöst, die einen zur Lokalisierung benötigten Vergleich der Sensordaten mit einem 3D-Umgebungsmodell ebenso effizient bestimmt wie vergleichbare Verfahren in 2D. Da der Vergleich zwischen Umgebungsmodell und Sensordaten unabhängig von dem verwendeten Lokalisierungsverfahren ist, liefert das entwickelte Verfahren einen Beitrag, um

beliebige zweidimensionale Lokalisierungsverfahren auf drei Dimensionen zu erweitern.

Die entwickelten Verfahren haben Grenzen. So ist die Lokalisierung mit 3D-Daten im Vergleich mit klassischen 2D-Varianten nur dann robuster, wenn das 3D-Umgebungsmodell mit der Realität besser übereinstimmt als das 2D-Modell mit den 2D-Sensordaten. Das Modell sollte aber auch nicht beliebig viele Polygone besitzen, da die Performanz des verwendeten BSP-basierten Ray Tracing-Verfahrens mit der Anzahl der verwendeten Polygone logarithmisch abnimmt. Grenzen des VRS-Verfahrens liegen vor allem in dem verwendeten Alterungsprozess auf der lokalen Hinderniskarte. Ein zu hoher Schwellwert für das Alter, ab dem ein Hindernis vergessen wird, führt dazu, dass falsch eingetragene Hindernisse lange im Hindernisgedächtnis bleiben. Ein zu kleiner Wert führt dazu, dass die Gedächtnisfunktion der lokalen Hinderniskarte verloren geht.

Im Folgenden werden weiterführende Fragen in den einzelnen Teilgebieten dieser Arbeit beschrieben. Es bieten sich sehr interessante Möglichkeiten, die Ergebnisse mehrerer Teilgebiete dieser Arbeit zu kombinieren.

Objekterkennung: Eine weitere Forschungsrichtung wäre die automatische Integration der Semantik in das 3D-Umgebungsmodell. Hierzu können in der 3D-Punktwolke, die während der Fahrt von den vier Hokuyo-Laserscannern aufgenommen wird, zunächst Häufungspunkte detektiert werden. Aus diesen Häufungspunkten wird anschließend mit einem online Marching Cubes-Verfahren die dem Häufungspunkt zugrunde liegende 3D-Polygonstruktur ermittelt. Dieses Polygonmodell wird mit einem Klassifikator einer gelernten Objektklasse zugewiesen. Über die Klasse wird dem 3D-Polygonmodell eine Semantik zugewiesen, und es kann in das 3D-Umgebungsmodell integriert werden. Wenn die Objekte z.B. zu einer Klasse statischer Objekte gehören, können sie zusätzlich in die im Folgenden beschriebene BSP-Baum basierte Pfadplanung integriert werden.

Pfadplanung: Da die Kollisionsdetektion zwischen zwei BSP-Bäumen zu einem der Standardprobleme der Spieleindustrie gehört, ist dieses Problem sehr gut verstanden und es existieren sehr effiziente Lösungen. Diese ermöglichen die Implementierung eines BSP-Baum basierten 3D-Pfadplaners, der mit Hilfe der effizienten Kollisionsberechnung zwischen dem Roboter- und dem Umgebungsmodell einen möglichen Pfad im 3D-Raum ermittelt. Dieser Pfadplaner könnte zusätzlich das Zonenmodell verwenden, um Gefahrenbereiche zu berücksichtigen. Durch Verwendung des Zonenmodells wäre der Pfadplaner zusätzlich in der Lage, komplexe Aufträge wie z.B. „Fahre in Raum 509 zu Tisch 2“ umzusetzen.

Hindernisvermeidung: Die Implementierung des VRS in dieser Arbeit verwendet eine 2D-Projektion der 3D-Punktwolke. Hier ist jedoch auch eine 3D-Rasterkarte in Kombination mit den effektiven Schnittpunktsberechnungsverfahren der Lokalisierung vorstellbar. Dies würde ebenfalls eine aktive Freigabe einer 3D-Zelle ermöglichen, da nicht mehr die Gefahr von Mehrdeutigkeiten besteht, die durch die Projektion hervorgerufen werden. Eine weitere interessante Forschungsfrage wäre es, das VRS-Konzept auf einer fliegenden Plattform einzusetzen. Da ein fliegendes Objekt Bewegungen in allen sechs Dimensionen erlaubt, könnten diese verwendet werden, um die Möglichkeiten des

VRS-Konzepts zur 3D-Hindernisvermeidung mit sechs Freiheitsgraden zu untersuchen.

Lokalisierung: Immer mehr moderne Grafikkarten besitzen eine so genannte Graphics Processing Unit (GPU). In der „Computer Vision“ werden diese bereits intensiv eingesetzt, um Algorithmen auf die massiv parallel arbeitenden Graphikkartenprozessoren auszulagern. Da das 3D-Umgebungsmodell vieler Robotikanwendungen nur aus wenigen tausend Polygonen besteht, lässt sich dieses in den Grafikkartenspeicher laden. Somit lässt sich ein BSP-Baum basiertes Ray-Tracing, wie es in dieser Arbeit zur Lokalisierung entwickelt wurde, auf den Graphikkartenprozessor auslagern. Moderne Ray-Tracer, die bereits eine vergleichbare Technik verwenden, erreichen, trotz großer Polygonmodelle, mehrere Millionen Schnittpunktberechnungen pro Sekunde. Dies ist eine Größenordnung schneller als die in dieser Arbeit verwendete CPU basierte Implementierung. Somit lässt sich im Fall der MCL die Anzahl der Partikel drastisch erhöhen, was die Performanz der Lokalisierung erhöht bzw. eine Lokalisierung mit sechs Freiheitsgraden erlaubt.

Anhang A

Kinematikherleitungen

A.1 Vorwärtskinematik

Dieser Abschnitt enthält die Herleitung der zeitlichen Ableitung der Orientierung des kinematischen Modells, wie es in Kapitel 4.2.1 verwendet wird.

Um das kinematische Modell zu bestimmen, wird zunächst die Sinusregel auf das in Abbildung A.1 zu sehende Dreieck (Rotationsmittelpunkt-Referenzpunkt-Vorderrad) (OCA) angewendet [66]:

$$\frac{\sin(\delta_f - \beta)}{l_f} = \frac{\sin(\frac{\pi}{2} - \delta_f)}{r} \tag{A.1}$$

Analog ergibt die Sinusregel für das Dreieck (Rotationsmittelpunkt-Referenzpunkt-Hinterrad) OCB:

$$\frac{\sin(\beta - \delta_r)}{l_r} = \frac{\sin(\frac{\pi}{2} + \delta_r)}{r} \tag{A.2}$$

mit $\sin(a-b) = \sin(a)\cos(b) - \sin(b)\cos(a)$ und $\sin(\frac{\pi}{2} \pm a) = cos(a)$ folgt aus Gleichung A.1:

$$\frac{\sin(\delta_f)\cos(\beta) - \sin(\beta)\cos(\delta_f)}{l_f} = \frac{\cos(\delta_f)}{r} \tag{A.3}$$

und aus Gleichung A.2:

$$\frac{\cos(\delta_r)\sin(\beta) - \cos(\beta)\sin(\delta_r)}{l_r} = \frac{\cos(\delta_r)}{r} \tag{A.4}$$

Multiplikation von Gleichung A.3 mit $\frac{l_f}{\cos(\delta_f)}$ ergibt:

$$\tan(\delta_f)\cos(\beta) - \sin(\beta) = \frac{l_f}{r} \tag{A.5}$$

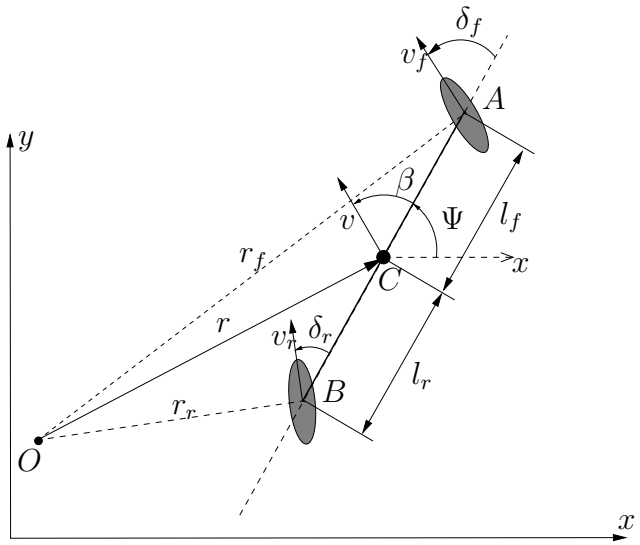


Abbildung A.1: Schematische Darstellung des kinematischen Zweiradmodells mit zwei Freiheitsgraden.

Analog ergibt die Multiplikation von Gleichung A.4 mit $\frac{l_r}{\cos(\delta_r)}$:

$$\sin(\beta) - \tan(\delta_r)\cos(\beta) = \frac{l_r}{r} \tag{A.6}$$

Addieren der Gleichungen A.5 und A.6 ergibt:

$$\cos(\beta)(\tan(\delta_f) - \tan(\delta_r)) = \frac{l_r + l_f}{r}$$

Da die Grundannahme des Modells von geringen Geschwindigkeiten des Fahrzeugs ausgeht, gilt ebenfalls:

$$\dot{\Psi} = \frac{v}{r}$$

Damit ergibt sich die Ableitung der Roboterorientierung zu:

$$\boxed{\dot{\Psi} = \frac{v\cos(\beta)\,(\tan(\delta_f) - \tan(\delta_r))}{l_f + l_r}} \tag{A.7}$$

Für das kinematische Modell muss nun noch der nicht direkt messbare Schwimmwinkel β sowie die Geschwindigkeit v auf die messbaren Radwinkel und Radgeschwindigkeiten zurückgeführt werden.

Zur Bestimmung des Schwimmwinkels wird Gleichung A.5 mit l_r multipliziert.

$$l_r \tan(\delta_f)\cos(\beta) - l_r \sin(\beta) = \frac{l_f l_r}{r} \tag{A.8}$$

Analog ergibt die Multiplikation von Gleichung A.6 mit l_f

$$l_f \sin(\beta) - l_f \tan(\delta_r)\cos(\beta) = \frac{l_r l_f}{r} \tag{A.9}$$

Subtraktion der Gleichung A.8 von A.9

$$\begin{aligned} l_f \sin(\beta) - l_f \tan(\delta_r)\cos(\beta) - l_r \tan(\delta_f)\cos(\beta) + l_r \sin(\beta) &= 0 \\ \sin(\beta)(l_f + l_r) - \cos(\beta)(l_f \tan\delta_r + l_r \tan\delta_f) &= 0 \\ \frac{\sin(\beta)}{\cos(\beta)}(l_f + l_r) &= l_f \tan\delta_r + l_r \tan\delta_f \end{aligned}$$

Damit lässt sich der Schwimmwinkel über die Radwinkel bestimmen.

$$\beta = \arctan\left(\frac{l_f \tan\delta_r + l_r \tan\delta_f}{l_f + l_r}\right) \tag{A.10}$$

Die Unabhängigkeit des Schwimmwinkels von den Radgeschwindigkeiten basiert auf der ursprünglichen Annahme des Modells, dass auf die Räder keine lateralen Kräfte wirken.

Um die Geschwindigkeit v auf die Radwinkel und Radgeschwindigkeiten zurückzuführen, muss berücksichtigt werden, dass es sich bei dem betrachteten Fahrzeug um einen starren Körper handelt. Aus diesem Grund müssen alle Teile des Fahrzeugs dieselbe Winkelgeschwindigkeit $\dot{\Psi}$ besitzen.

$$\dot{\Psi} = \frac{v}{r} = \frac{v_f}{r_f} = \frac{v_r}{r_r}$$

Daraus lassen sich die Bahngeschwindigkeiten für die Radien der einzelnen Räder wie folgt berechnen:

$$v_f = v\frac{r_f}{r}$$
$$v_r = v\frac{r_r}{r}$$

Mit Hilfe des Sinussatzes ergibt sich für das Verhältnis von r_f zu r:

$$\begin{aligned} \frac{r_f}{r} &= \frac{\sin\left(\beta + \frac{\pi}{2}\right)}{\sin\left(\frac{\pi}{2} - \delta_f\right)} \\ &= \frac{\cos(\beta)}{\cos(\delta_f)}. \end{aligned}$$

Analog gilt für das Verhältnis von r_r zu r:

$$\begin{aligned} \frac{r_r}{r} &= \frac{\sin\left(\frac{\pi}{2} - \beta\right)}{\sin\left(\frac{\pi}{2} + \delta_r\right)} \\ &= \frac{\cos(\beta)}{\cos(\delta_r)}. \end{aligned}$$

Damit ergeben sich:

$$v_f \cos(\delta_f) = v\cos(\beta) \tag{A.11}$$

und

$$v_r \cos(\delta_r) = v\cos(\beta). \tag{A.12}$$

Theoretisch lässt sich die Geschwindigkeit v sowohl mit Gleichung A.11 als auch mit Gleichung A.12 auf messbare Größen zurückführen. Um die Auswirkung von Messfehlern zu minimieren, werden die beiden Gleichungen addiert, so dass sich die Bestimmung von v auf alle vier Werte stützt.

$$v = \frac{v_f \cos(\delta_f) + v_r \cos(\delta_r)}{2\cos(\beta)} \tag{A.13}$$

Somit lässt sich die Ableitung nach der Zeit der Orientierung (Gleichung A.7) durch Einsetzen von β (Gleichung A.10) und v (Gleichung A.13) auf messbare Größen zurückführen.

A.2 Inverse Kinematik

Dieser Abschnitt beschreibt die Herleitung des allgemeine inversen kinematischen Modells für ein Zweiradmodell mit zwei Rotationsfreiheitsgraden sowie für den Spezialfall des zero-side-slip-Manövers, wie es in Kapitel 4.2.2 verwendet wird.

Um das Verhältnis der Radwinkel beim zero-side-slip-Manöver zu bestimmen, wird die Beziehung $\beta = 0$ in Gleichung A.10 eingesetzt:

$$\begin{aligned} 0 &= \frac{l_f \tan\delta_r + l_r \tan\delta_f}{l_f + l_r} \\ l_f \tan\delta_r &= -l_r \tan\delta_f \\ \delta_r &= -\frac{l_r}{l_f}\delta_f \end{aligned}$$

Für den Fall, dass der Referenzpunkt in der Mitte zwischen Vorder- und Hinterrad liegt ($l_f = l_r$), folgt $\delta_r = -\delta_f$. Dies berücksichtigend und mit ($l_f + l_r = L$) ergibt sich nach Gleichung A.7 die Orientierungsdifferenz zu:

$$\dot{\Psi} = \frac{2v\tan(\delta_f)}{L} \tag{A.14}$$

Aus dieser Gleichung lassen sich der Radwinkel in Bezug zur Orientierungsänderungsrate ableiten.

$$\delta_f = \arctan\left(\frac{\dot{\Psi}L}{2v}\right) \tag{A.15}$$

$$\delta_r = -\arctan\left(\frac{\dot{\Psi}L}{2v}\right) \tag{A.16}$$

Für die Radgeschwindigkeiten, die bei einem Manöver mit $\beta = 0$ eingestellt werden müssen, folgt aus Gleichung A.11 bzw. A.12:

$$v_f = \frac{v}{\cos\delta_f} \quad \text{und} \quad v_r = \frac{v}{\cos\delta_r}.$$

Im Folgenden wird die inverse Kinematik für beliebige vorgegebene Geschwindigkeitsrichtungen β und Orientierungsänderungen $\dot{\Psi}$ hergeleitet. Aus Gleichung A.10 folgt unter Berücksichtigung, dass der Bezugspunkt $C(x, y)$ in der Mitte der Strecke AB liegt ($l_f = l_r$):

$$\begin{aligned} \tan\beta(l_f + l_r) &= l_f \tan\delta_r + l_r \tan\delta_f \\ 2\tan\beta &= \tan\delta_r + \tan\delta_f \\ 2\tan\beta - \tan\delta_f &= \tan\delta_r \end{aligned} \tag{A.17}$$

Ferner folgt aus Gleichung A.7:

$$\frac{\dot{\Psi}(l_f + l_r)}{v\cos\beta} = \tan\delta_f - \tan\delta_r \tag{A.18}$$

Nun wird $\tan\delta_r$ mit Hilfe von Gleichung A.17 in Gleichung A.18 substituiert. Ferner wird $(l_f + l_r) = L$ gesetzt, da durch die Substitution die Gleichung nur für $l_f = l_r$ gilt:

$$\begin{aligned} \frac{\dot{\Psi}L}{v\cos\beta} &= \tan\delta_f - (2\tan\beta - \tan\delta_f) \\ \frac{\dot{\Psi}L}{v\cos\beta} + 2\tan\beta &= 2\tan\delta_f \\ \frac{\dot{\Psi}L}{2v\cos\beta} + \tan\beta &= \tan\delta_f \end{aligned} \tag{A.19}$$

$$\boxed{\delta_f = \arctan\left(\tan\beta + \frac{\dot{\Psi}L}{2v\cos\beta}\right)} \tag{A.20}$$

Rücksubstitution von $\tan\delta_f$ (Gleichung A.19) in Gleichung A.17 ergibt:

$$\begin{aligned} 2\tan\beta &= \tan\delta_r + \left(\frac{\dot{\Psi}L}{2v\cos\beta} + \tan\beta\right) \\ \tan\beta - \frac{\dot{\Psi}L}{2v\cos\beta} &= \tan\delta_r \end{aligned}$$

$$\boxed{\delta_r = \arctan\left(\tan\beta - \frac{\dot{\Psi}L}{2v\cos\beta}\right)} \tag{A.21}$$

Neben den passenden Radwinkeln für eine vorgegebene Richtung mit einer vorgegebenen Geschwindigkeit und einer bestimmten Orientierungsdifferenzrate müssen ebenfalls die passenden Radgeschwindigkeiten bestimmt werden. Wären die Radgeschwindigkeiten nicht an die Radstellungen angepasst, ist es möglich, dass die Motoren der angetriebenen Räder gegeneinander arbeiten, was zu einem schnellen Verschleiß und einer unvorhersagbaren Bewegung

des Roboters führen würde. Aus Gleichung A.11 und A.12 folgt für die Radgeschwindigkeiten:

$$\boxed{v_f = \frac{\cos(\beta)}{\cos(\delta_f)} v} \quad \text{und} \quad \boxed{v_r = \frac{\cos(\beta)}{\cos(\delta_r)} v}.$$

Anhang B

Flächenextraktion in 3D-Punktwolken

B.1 3D-Houghtransformation

Die Houghtransformation ist eine Standardtechnik in der digitalen Bildverarbeitung. Sie wurde bereits 1962 in den USA als Patent angemeldet und zuerst in [18] beschrieben. Ursprünglich wurde sie verwendet, um Geraden in Bildern zu finden. Die Houghtransformation beruht darauf, dass sich eine Gerade im zweidimensionalen Raum in der Koordinatenform B.1 schreiben lässt. Dabei ist m die Steigung der Geraden und b der Schnittpunkt mit der y-Achse.

$$y_p = m_i * x_p + b_j \tag{B.1}$$

Nun werden für jeden Punkt (x_p, y_p) bzw. für jedes Pixel alle Wertepaare (m_i, b_j) bestimmt, so dass Gleichung B.1 erfüllt ist. Da m_i und b_j reelle Zahlen sind, gibt es unendlich viele Möglichkeiten von Kobinationspaaren, die für jeden Punkt getestet werden müssen. Aus diesem Grund wird der Wertebereich der beiden Parameter in der Praxis diskretisiert und begrenzt, so dass nur noch endliche Wertepaare getestet werden müssen. Die Begrenzung der Parameter stellt jedoch ein Problem dar, da vertikale Linien eine unendliche Steigung haben, so dass sowohl m als auch b im Wertebereich von $-\infty$ bis ∞ liegen. Aus diesem Grund wird Gleichung B.1 mit Hilfe der Polarkoordinatendarstellung in Gleichung B.2 transformiert. Da θ den Winkel der Geraden mit der x-Achse beschreibt, ist der Wertebereich von θ zwischen 0 und π begrenzt und lässt sich somit leicht diskretisieren. Der Abstand ρ des Ursprungs von der Geraden besitzt einen nach oben offenen Wertebereich, dessen obere Grenze lässt sich jedoch einfach an das zu lösende Problem anpassen.

$$x_p \cos\theta_i + y_p \sin\theta_i = \rho_j \tag{B.2}$$

Die Houghtransformation iteriert über alle Punkte und alle Wertepaare und summiert auf, wie oft ein Wertepaar (θ_i, ρ_j) die Geradengleichung für den jeweiligen Punkt (x_p, y_p) erfüllt. Diejenigen Wertepaare, die sehr häufig die Geradengleichung erfüllen, beschreiben die wahrscheinlichsten Geraden der zugrundeliegenden Daten.

Die dreidimensionale Houghtransformation kann analog zur $2D$-Houghtransformation hergeleitet werden. Es werden jedoch keine Geraden auf einen Punkt im zweidimensionalen Houghraum abgebildet, sondern Ebenen auf einen Punkt im dreidimensionalen Houghraum. Dies hat zur Folge, dass die Ausgangsgleichung zur Herleitung der dreidimensionalen Houghtransformation die Hessesche Normalform einer Ebene ist. Sie besagt, dass ein Punkt $\vec{p}$ in der beschriebenen Ebene liegt, wenn das Skalarprodukt des Punktes mit dem normierten Normalenvektor $\vec{n}$ der Ebene gleich dem Abstand ρ der Ebene vom Ursprung ist.

$$p_x * n_x + p_y * n_y + p_z * n_z - \rho = 0 \tag{B.3}$$

Der Suchraum der Parameter (n_x, n_y, n_z, ρ), die diese Ebenengleichung für einen Punkt (p_x, p_y, p_z) erfüllen, kann von vier auf drei Dimensionen reduziert werden, indem analog zum Vorgehen bei der zweidimensionalen Houghtransformation auf Kugelkoordinaten übergegangen wird. Der Normalenvektor wird dadurch zu:

$$\vec{n} = (\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\phi) \tag{B.4}$$

Hierbei beschreibt ϕ den Winkel zur positiven z-Achse (Polarwinkel) mit einem Wertebereich von 0 bis π und θ beschreibt den Winkel zur positiven x-Achse (Azimutwinkel) mit einem Wertebereich von 0 bis 2π. Das Einsetzen von Gleichung B.4 in Gleichung B.3 liefert die dreidimensionale Houghtransformation mit dem Parametersuchraum (θ, ϕ, ρ)

$$\rho_k = p_x \cos\theta_i \sin\phi_j + p_y \sin\theta_i \sin\phi_j + p_z cos\phi_j \tag{B.5}$$

Analog zur zweidimensionalen Houghtransformation besitzt ρ einen Wertebereich zwischen 0 und ∞, dessen Obergrenze aber leicht an das Problem angepasst werden kann. Um rechnerisch handhabbar zu werden, wird der durch (θ, ϕ, ρ) aufgespannte Houghraum durch definierte Schrittweiten zu einem dreidimensionalen Akkumulatorfeld diskretisiert. Danach wird für jeden Punkt der Punktwolke ermittelt, welche Kombinationen aus $(\theta_i, \phi_j, \rho_k)$ Gleichung B.5 erfüllen, und der dem Index ijk entsprechende Quader des Akkumulatorfeldes wird erhöht.

Eine Häufigkeitsanalyse des Akkumulatorfeldes ergibt die den Daten zugrundeliegenden Ebenen, da ein Akkumulatorquader mit einem hohen Summationsindex eine Ebene beschreibt, in der sehr viele der untersuchten Punkte liegen.

B.2 Random Sample Consensus (RANSAC)

Das RANSAC-Verfahren wurde von Fischler und Bolles entwickelt [20]. Es handelt sich um einen allgemeinen Ansatz, um Modelle an verrauschte Daten anzupassen [91]. Das RANSAC-Verfahren wählt iterativ Proben aus dem zu untersuchenden Datensatz aus und bestimmt die optimalen Parameter, um das Modell in diesen Testdatensatz einzupassen. Anschließend werden alle Punkte betrachtet, ob sie mit dieser Hypothese übereinstimmen. Oft wird dieses Verfahren iterativ angewendet, um mehrere Modelle in einem Datensatz zu finden. Dazu

werden alle Punkte, die zu einem akzeptiertem Modell gehören, entfernt und im weiteren nur die restlichen Punkte betrachtet. Durch diese Herangehensweise werden zunächst sehr große, den Datensatz dominierende Strukturen gefunden, und immer feinere Strukturen, je mehr Iterationen vorgenommen werden.

Ebenso wie die Houghtransformation wird dieses Verfahren robust gegen Rauschen durch das Votieren der einzelnen Punkte für die jeweiligen Modelle.

Um in Kombination mit Punktwolken von mehreren hunderttausend Punkten effizient einsetzbar zu sein, kann beim RANSAC- Verfahren an zwei Stellen die Laufzeit beeinflusst werden. Zum einen besitzt die initiale Schätzung des Modells starken Einfluss. Ist eine solche initiale Schätzung nicht möglich, verringert eine iterative Betrachtung von Teilmengen der Punktwolke ebenfalls die Rechenkomplexität des Verfahrens. So wird in [72] ein Octreeverfahren verwendet, um die Punktwolke in gleichmäßige Würfel zu unterteilen. Die Punkte innerhalb eines Würfels werden jeweils mit dem RANSAC-Verfahren ausgewertet. Anschließend werden Ebenen benachbarter Würfel zusammengefügt, falls sie dieselbe Ausrichtung haben.

Anhang C

Hindernisvermeidung

C.1 Vector Field Histogram

Die Vector Field Histogram (VFH)-Methode wurde 1991 von Borenstein und Koren vorgestellt [7]. Sie ist ein zweistufiges Verfahren, das zunächst eine zweidimensionale Rasterkarte anhand der Sensordaten mit Wahrscheinlichkeiten belegt, dass sich in dem jeweiligen Raster ein Hindernis befindet. Diese Rasterkarte ist global und wird anhand der Odometrie aktualisiert. Die Wahrscheinlichkeit, dass ein Raster belegt ist, wird erzeugt, indem die gemessene Entfernung anhand der Odometrie auf ein Raster abgebildet wird. Dieses Raster und dessen Umgebung werden inkrementiert. Somit besitzt ein Raster mit einem hohen Wert eine hohe Wahrscheinlichkeit und ein Raster mit einem geringen Wert eine niederige Wahrscheinlichkiet, dass sich dort ein Hindernis befindet. Zusätzlich wird ein sogenanntes „Aktives Fenster“ definiert. Dieses ist ein an der aktuellen Roboterposition zentrierter quadratischer Bereich, der die Raster markiert, die Einfluss auf die Berechnung der Fahrbefehle haben.

Die zweite Stufe berechnet aus dem aktiven Fenster ein Polarhistogramm, um anhand dessen die optimalen Fahrbefehle zu ermitteln. Dazu wird ausgehend von der Roboterpose das aktive Fenster in polare Sektoren unterteilt. Für jedes Raster des aktiven Fensters, das eine Wahrscheinlichkeit größer Null hat, wird der sogenannte Hindernisvektor $m_{i,j}$ berechnet. Dieser ist proportional zum Quadrat des Wahrscheinlichkeitswerts des Rasters $c^{\star}_{i,j}$ und dem Abstand des Rasters zum Roboter $d_{i,j}$:

$$m_{i,j} = (c^{\star}_{i,j})^2(a - b \cdot d_{i,j}). \tag{C.1}$$

Die Konstanten a und b werden so gewählt, dass $a - b \cdot d_{max} = 0$ gilt, so dass der Hindernisvektor am Rand des aktiven Fensters (d_{max} = halbe Fensterdiagonale) bei Null beginnt und bei geringeren Entfernungen $d_{i,j}$ ansteigt. Für jeden Sektor wird die polare Hindernisdichte h_{Θ} als Summe der Hindernisvektoren $m_{i,j}$ des Sektors berechnet:

$$h_{\Theta} = \sum_{i,j} m_{i,j}. \tag{C.2}$$

Da Hindernisse mit geringer Enfernung große Hindernisvektoren haben, bzw. große Hindernisse viele Hindenisvektoren hervorrufen, enthält das Polarhistogramm in den Sektoren, in denen diese Hindernisse liegen, große Werte.

Zur Berechnung der Fahrbefehle wird das Polarhistogramm über einen festen Schwellwert in freie und blockierte Bereiche aufgeteilt. Alle Sektoren, die einen Wert besitzen, der oberhalb des Schwellwerts liegt, werden als blockiert markiert, alle anderen, sogenannte Täler, dementsprechend als frei. Die optimale Bewegungsrichtung des Roboters liegt in dem Tal, das zum aktuellen Ziel die geringste Winkeldifferenz hat. Werden dem Tal nur wenige Sektoren zugeordnet, wird dieses als schmal klassifiziert und die optimale Bewegungsrichtung ist die Mitte des gewählten Tals. Bei einem breiten Tal ist die optimale Bewegungsrichtung an dem Rand des Tals, das zum aktuellen Ziel die geringste Winkeldifferenz hat.

Die optimale Geschwindigkeit wird über eine empirisch ermittelte Konstante α gesteuert. Es wird der Sektor des Polarhistogramms h_c verwendet, der die Hindernisse repräsentiert, die direkt vor dem Roboter liegen. Das Verhältnis dieses Sektors zur empirisch bestimmten Konstante ergibt den Faktor, um den die maximal erlaubte Geschwindigkeit v_{max} reduziert werden muss. Zusätzlich wird die Geschwindigkeit um das Verhältnis der aktuellen Rotationsgeschwindigkeit ω zur maxmimal erlaubten Rotationsgeschwindigkeit ω_{max} reduziert.

$$v = v_{max} \cdot \left(1 - \frac{min(h_c, \alpha)}{\alpha}\right)\left(1 - \frac{\omega}{\omega_{max}}\right) + v_{min}$$

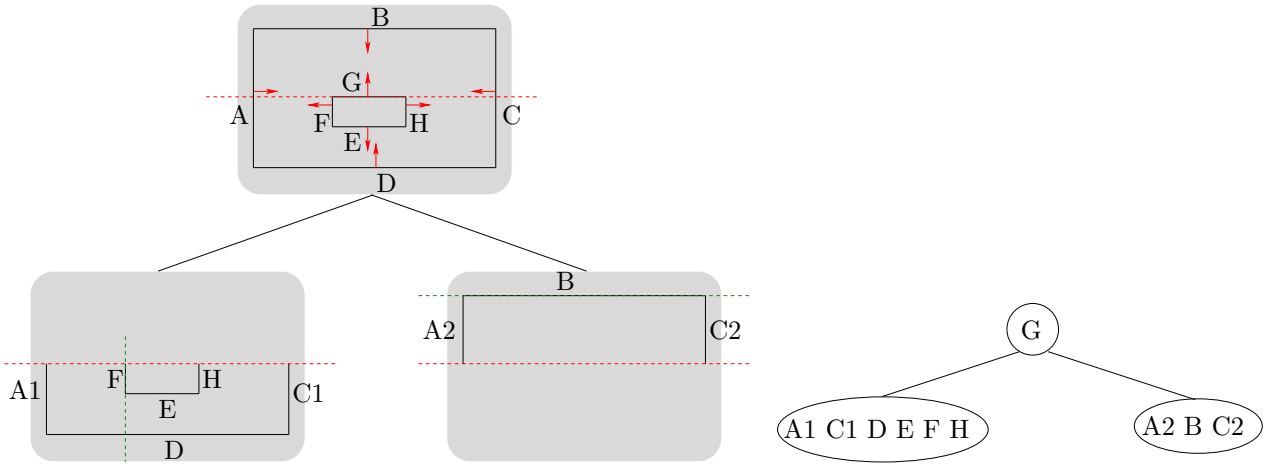
Die Addition von v_{min} stellt sicher, dass der Roboter nicht gegen Null strebende Geschwindigkeiten annimmt und der Roboter stehen bleibt.

Anhang D

BSP-Baum basiertes Ray-Tracing

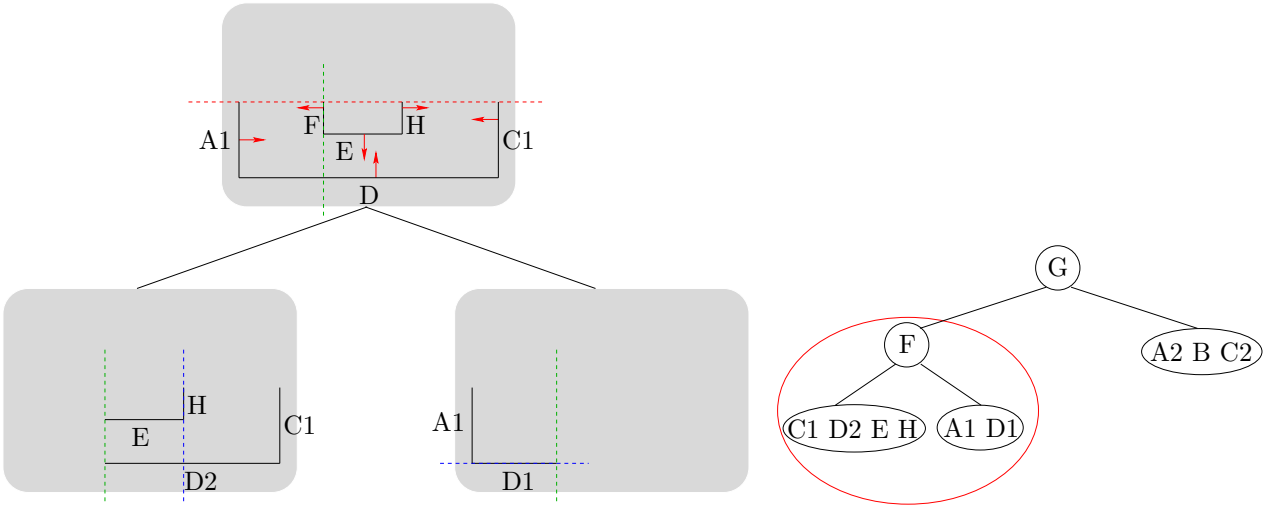
D.1 Erstellung eines BSP-Baums

Im Folgenden wird die Erstellung des in Kapitel 6.1.5 verwendeten Solid Leaf BSP-Baums aus einem Polygonmodell beschrieben. Dabei ist die Trennebene jeweils manuell gewählt. Das Verhältnis zwischen Balance und Splitfaktor wird geschätzt und nicht rechnerisch bestimmt (vgl. Kapitel 6.1.5. Dies hat jedoch ausschließlich Einfluss auf den resultierenden Baum, nicht aber auf den im Folgenden beschriebenen Weg zum Erstellen des Baums.

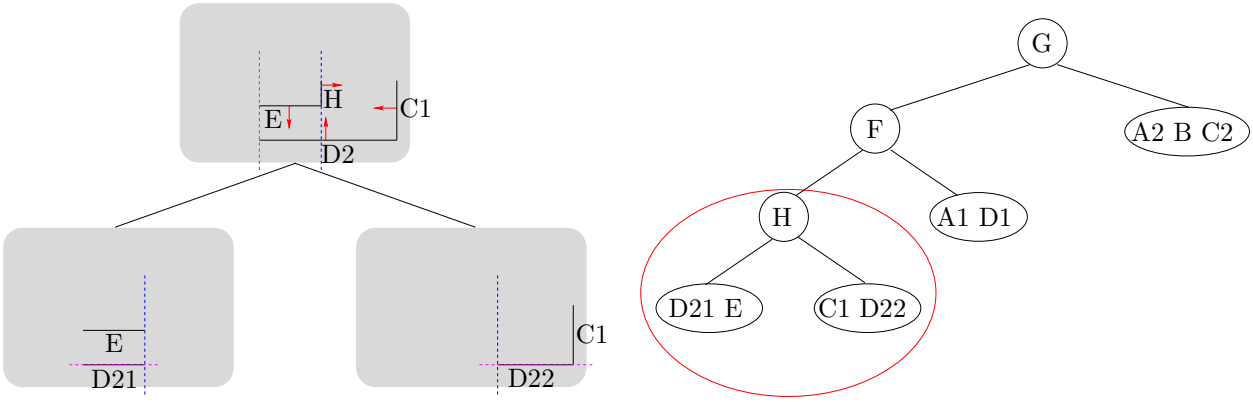


Die Erstellung des BSP-Baums beginnt mit dem Wurzelknoten. Hier werden alle Polygone des 3D-Modells betrachtet. Es wird das Polygon bestimmt, das in der Ebene liegt, die das Modell in zwei Bereiche teilt, die in etwa gleich viele Polygone enthalten. Ferner sollte die Trennebene möglichst wenige Polygone in zwei Teilpolygone aufteilen. In diesem Modell wird die rot markierte und durch das Polygon G definierte Ebene als Trennebene gewählt. Sie trennt die Polygone A und C in zwei Teilpolygone A1, A2 und C1,C2. Alle Polygone, die vor der Ebene G liegen (in Richtung der Normalen von G (rot)) werden dem vorderen Kindknoten zugeordnet (A2, B, C2), die restlichen Polygone dementsprechend dem hinteren

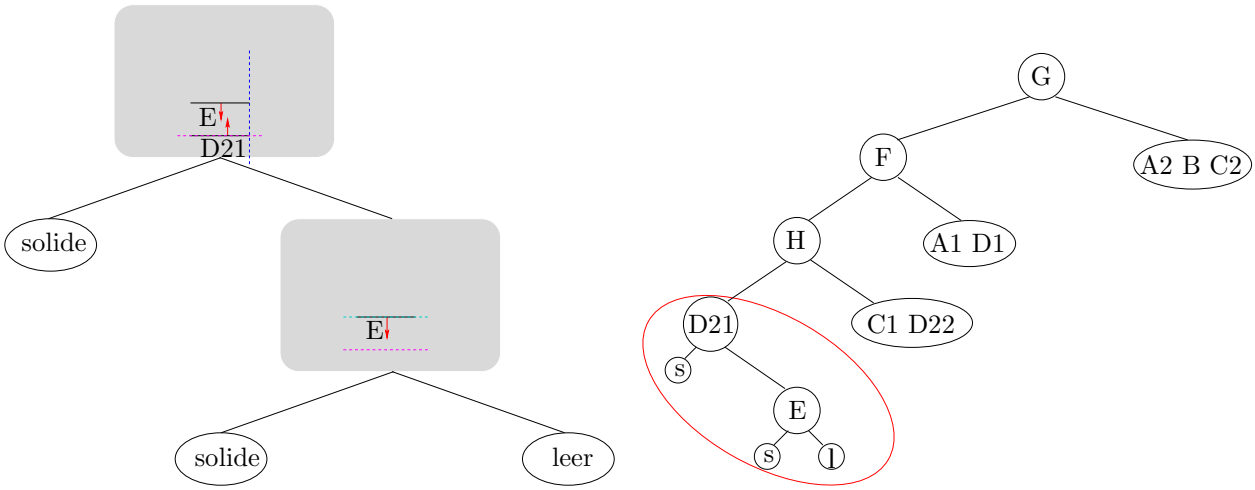
Kindknoten (A1, C1, D, E, F, H). Das Polygon G wird im Wurzelknoten gespeichert, da es die Trennebene definiert.



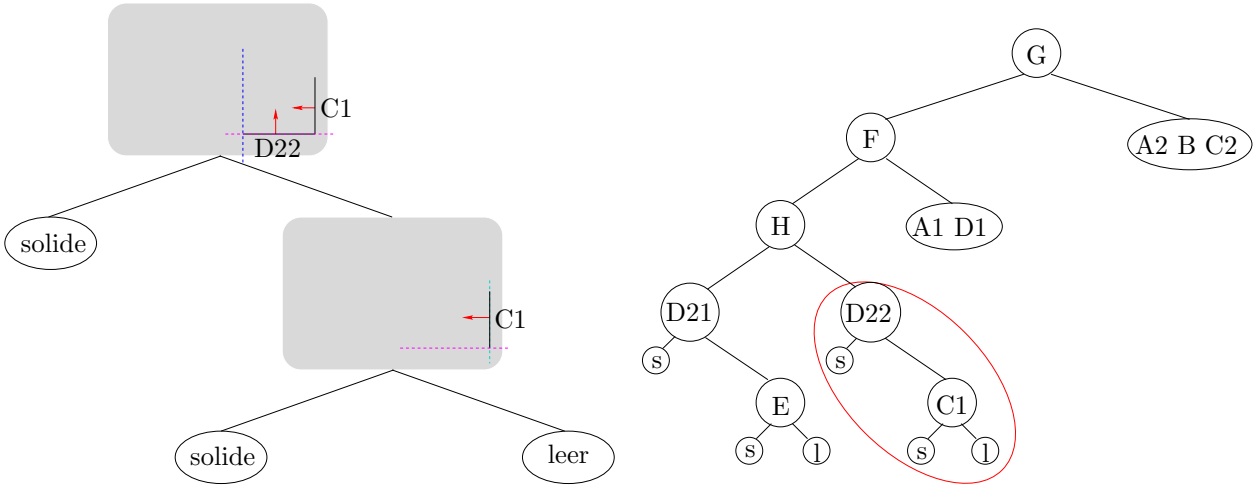
Die Erzeugung des BSP-Baums wird mit dem hinteren Kindknoten fortgesetzt. Es wird zunächst wieder die optimale Trennebene bestimmt. Anhand der gefundenen Trennebene, die durch das Polygon F verläuft, werden die restlich Polygone auf den vorderen und hinteren Kindknoten aufgeteilt. Das Polygon D wird hierbei in die beiden Teilpolygone D1 und D2 geteilt.



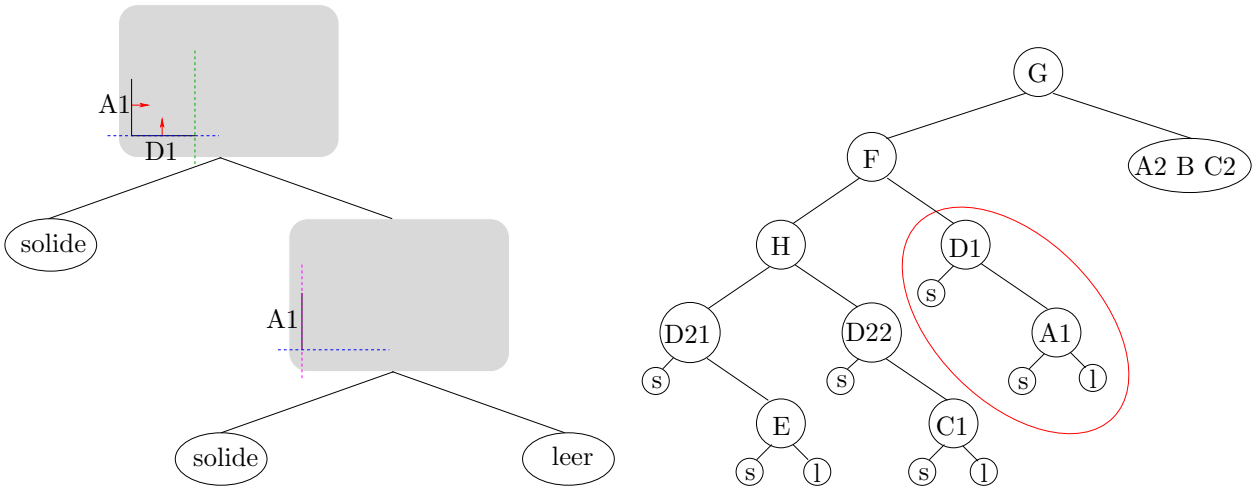
Der hintere Kindknoten des Polygons F wird an der Ebene durch das Polygon H getrennt. Dies führt zu einer weiteren Trennung des Polygons D2 in die Teilpolygone D21 und D22.



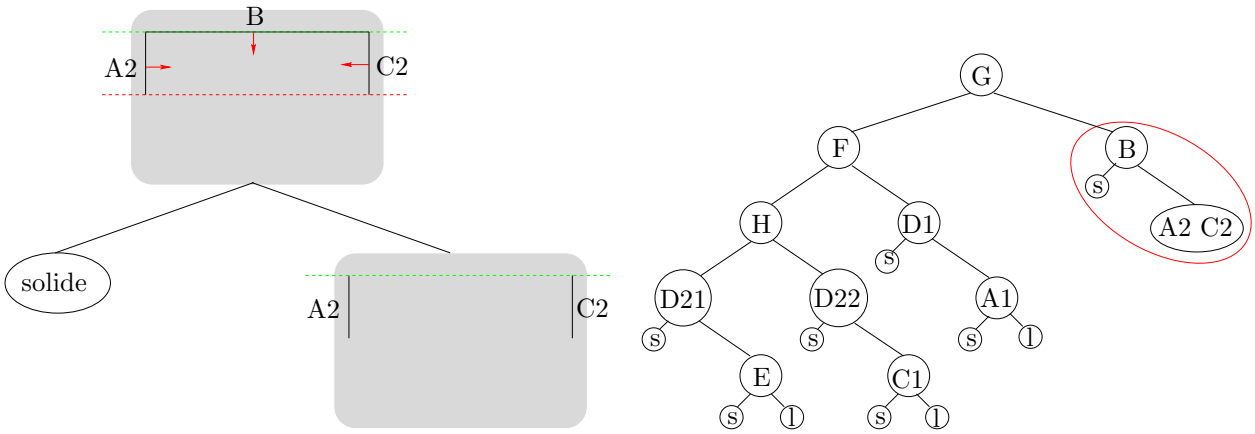
Der hintere Kindknoten von H besteht nur noch aus zwei Polygonen. Somit ist das Polygon, welches die Trennebene definiert, frei wählbar. In diesem Fall wird das Polygon D21 gewählt. Das Polygon E liegt vor dem Polygon D21 und wird in den vorderen Kindknoten verschoben. Hinter dem Kindknoten befindet sich kein Polygon mehr. Da bei einer legalen Solid Leaf BSP-Baumstruktur die Rückseite eines Polygons nicht direkt aus dem leeren Raum sichtbar sein darf, wird der hintere Kindknoten als „solide“ markiert. Der vordere Kindknoten E besitzt nur noch ein Polygon, daher wird dessen vorderer Kindknoten als leer und dessen hinterer Kindknoten als solide markiert.



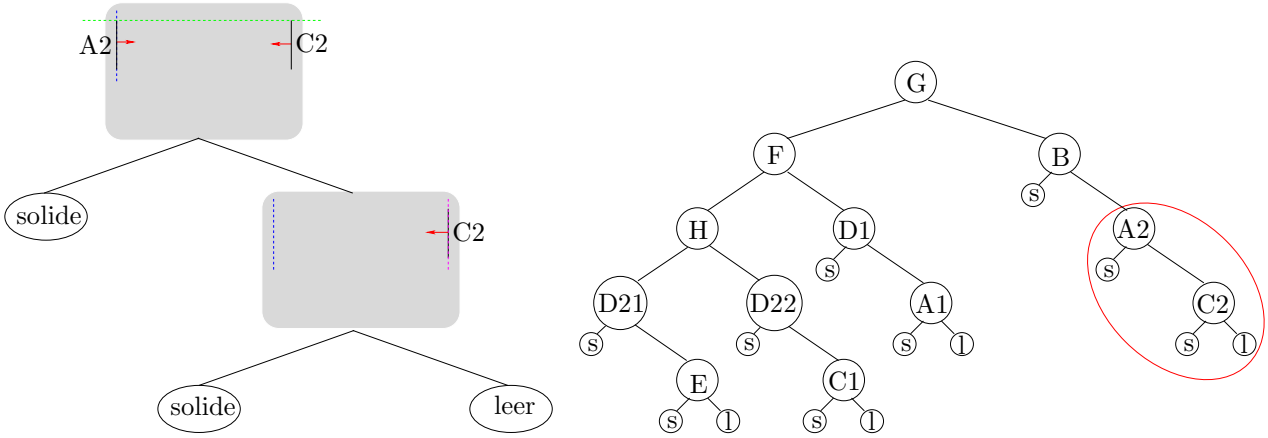
Da der hintere Kindknoten von H abgearbeitet ist, wird zum vorderen Kindknoten übergegangen. Es kann wieder frei zwischen den beiden verbleibenden Polygonen gewählt werden. Hier wird das Polygon D22 gewählt, so dass das Polygon C1 in den vordern Kindknoten wandert. Da der hintere Kindknoten leer ist, wird er als solide markiert.



Da der hintere Kindknoten von F erstellt ist, wird nun der vordere Kindknoten mit den Polygonen D1 und A1 analog zum Knoten D22 verarbeitet.



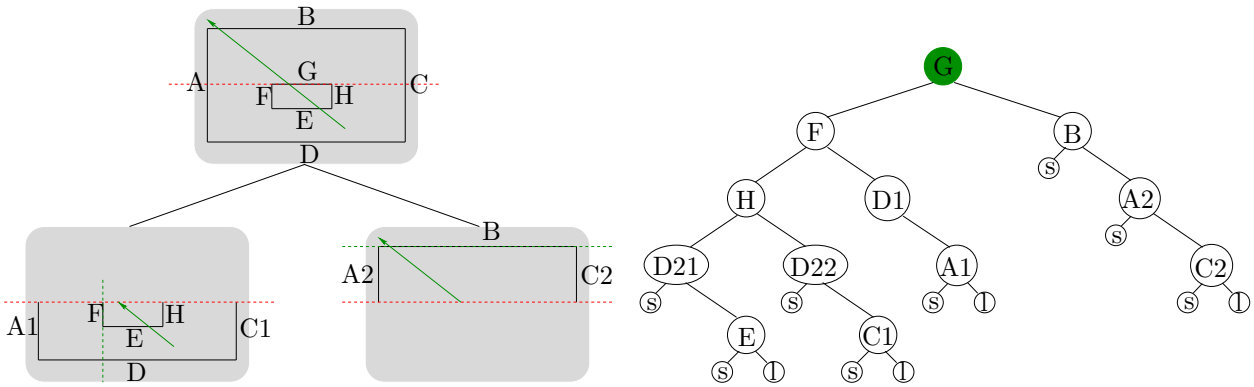
Nachdem der hintere Kindknoten des Wurzelknotens G erstellt ist, wird der vordere Kindknoten betrachtet. Da die verbleibenden drei Polygone eine konvexe Form bilden, ist die Wahl der Trennebene beliebig, weil immer zwei Polygone im vorderen Kindknoten und ein solider hinterer Kindknoten erzeugt werden. Hier wird das Polygon B gewählt.



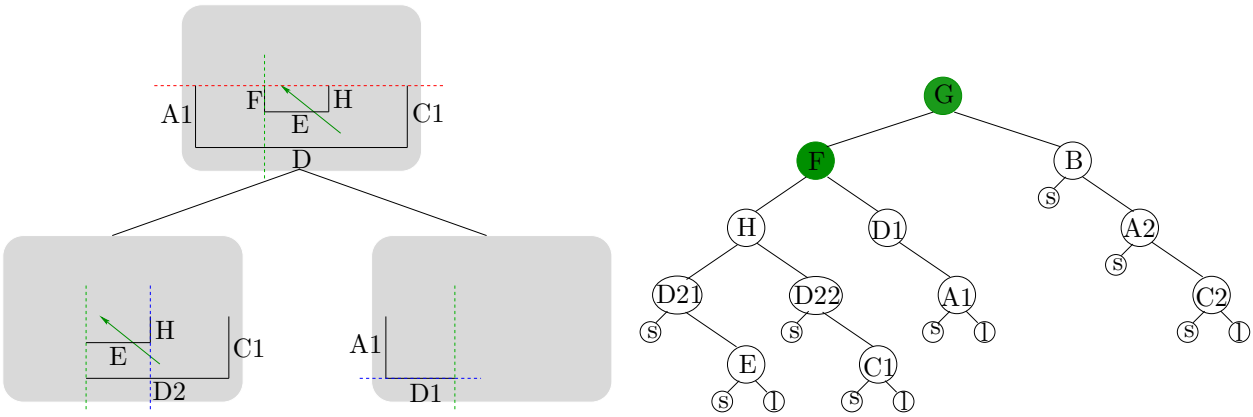
Zum Schluss werden die Polygone A2 und C2 des vorderen Kindknotens von B analog zu den Konten D21,D22 bzw. D1 entwickelt. Hier wird zunächst A2 als Trennebene gewählt.

D.2 Ray-Tracing im BSP-Baum

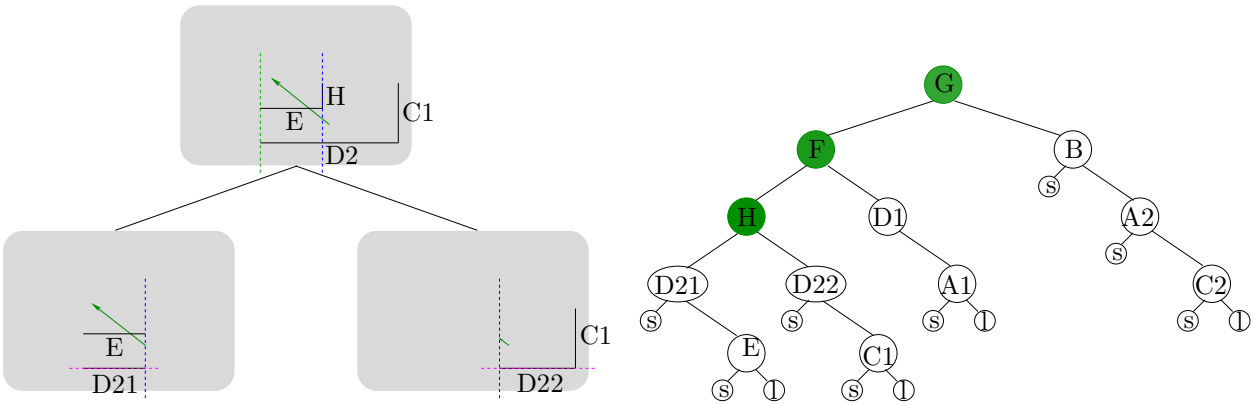
Im Folgenden wird erläutert, in welcher Reihenfolge die Knoten des Solid Leaf BSP-Baums besucht werden, um den ersten Schnittpunkt eines Strahls mit dem Modell zu bestimmen. Hier wird ein „worst case“-Beispiel gewählt, das eine maximale Anzahl von Knoten besucht.



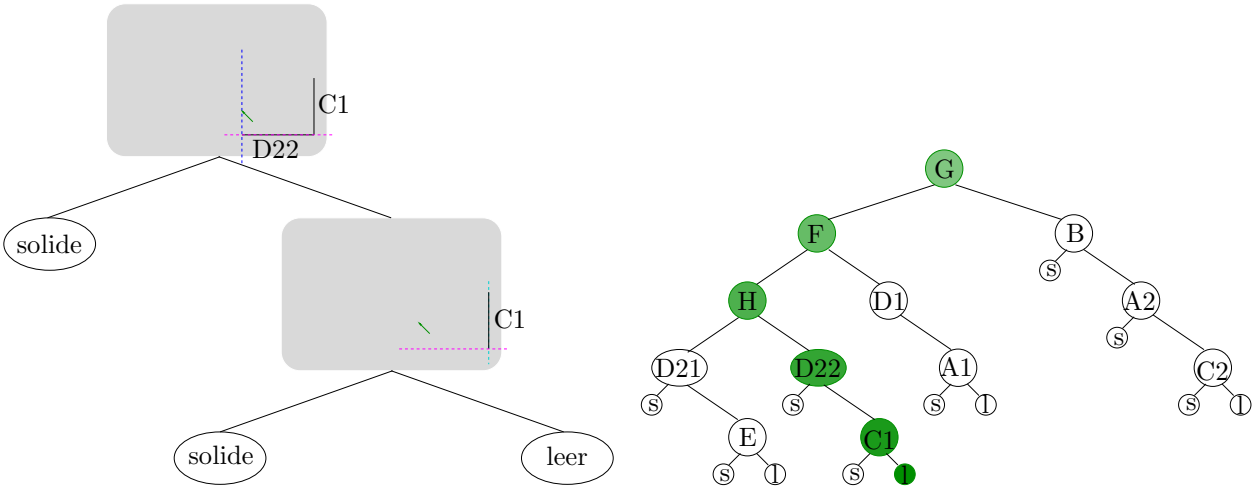
Initial wird die Länge t_{max} des Strahls (grün) auf die Größe des Modells beschränkt. Anschließend wird bestimmt auf welcher Seite der Trennebene (rot) der Ursprung des Strahls liegt. Es wird ebenfalls die Länge des Strahls bis zur Trennebene bestimmt. Da diese Länge geringer ist als t_{max}, schneidet die Trennebene den Strahl und es müssen beide Kindknoten betrachtet werden. Für den hinteren Kindknoten wird t_{max} und für den vorderen Kindknoten der Ursprung des Strahls auf den Schnittpunkt zwischen Trennebene und Strahl gesetzt. Die Traversierung wird mit dem hinterem Kindknoten fortgesetzt, da sich auf dieser Seite der Trennebene der Ursprung des Ausgangsstrahls befindet.



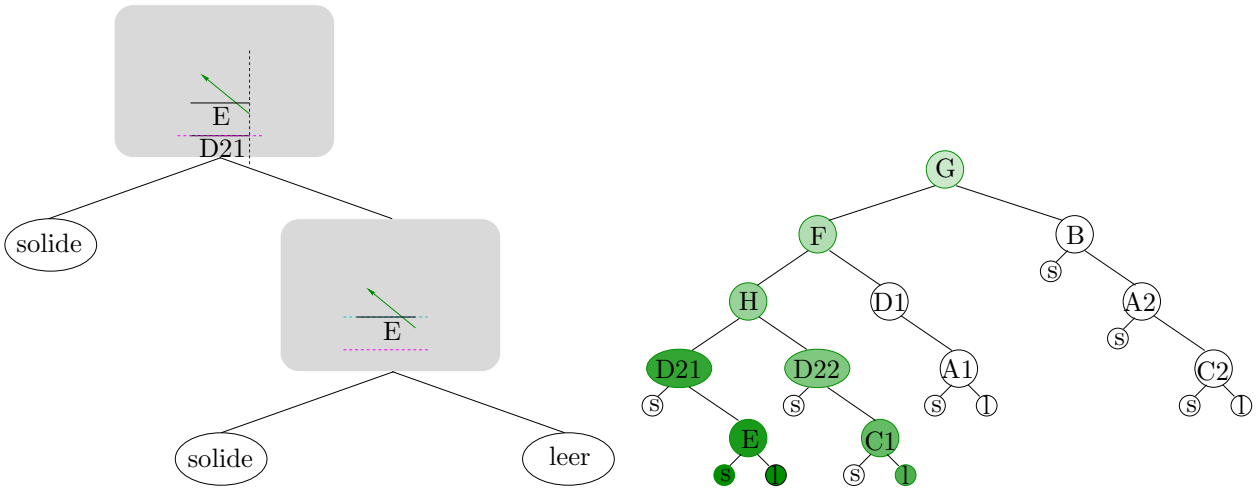
Zunächst wird wieder die Orientierung des Strahls zur Trennebene F (grüne gestrichelte Linie) und die Länge des Strahls bis zur Trennebene bestimmt. Da der Ursprung im hinteren Kindknoten liegt und t_{max} geringer als die Länge des Strahls bis zum Schnittpunkt ist, befindet sich der relevante Bereich des Strahls ausschließlich hinter der Trennebene, so dass nur der hintere Kindknoten weiter traversiert werden muss.



Beim hinteren Kindknoten von F bildet das Polygon H die Trennebene (blaue gestrichelte Linie). Hier tritt wieder der Fall des Wurzelkontens G auf, dass der relevante Bereich des Strahls die Trennebene schneidet. Somit wird der Strahl anhand der Trennebene geclippt, und es wird zunächst der vordere Kindknoten weiter traversiert, da sich in diesem der Ursprung des Strahls befindet.



Zunächst wird der Strahl in Bezug zur Trennebene D22 (magenta) betrachtet. Da sich der Ursprung des Strahls vor der Ebene befindet und der Strahl die Ebene nicht schneidet, befindet sich der relevante Bereich des Strahls ausschließlich im vorderen Kindknoten C1. Für diesen gilt bzgl. des Strahls dasselbe wie für seinen Elternknoten D22, so dass wiederum zum vorderen Kindknoten weitergegangen wird. Dieser ist als leer markiert, so dass kein Schnittpunkt gefunden wird.



Da wie oben beschrieben bei der Traversierung des vorderen Kindknotens von H kein Schnittpunkt gefunden wurde, wird die Traversierung mit dem hinteren Kindknoten D21 fortgesetzt. Der relevante Bereich des Strahls liegt komplett vor der Trennebene und die Traversierung wird mit dem vorderen Kindknoten E fortgesetzt. Da der Strahl die Trennebene von E schneidet, wird die Traversierung mit beiden Kindknoten fortgesetzt. Es wird zunächst der vordere Kindknoten besucht, da sich der Ursprung des Strahls auf dieser Seite befindet. Der vordere Kindknoten ist als leer markiert und es wird zum hinteren Kindknoten von E übergegangen. Da dieser als solide markiert ist, ist der Schnittpunkt gefunden, und das Polygon E wird als erstes vom Strahl geschnitten.

Literaturverzeichnis

[1] S. Albrecht, J. Hertzberg, K. Lingemann, et al. Device level simulation of Kurt3D rescue robots. In *Proc.: Third International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED '06)*, Seiten 18–23, 15–18 Jun. 2006.

[2] D. Arbuckle, A. Howard, und M. Mataric. Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and System (IROS '02)*, Band 1, Seiten 409–414, IEEE Press, New York, 30 Sept.- 4 Okt. 2002.

[3] T. Baier-Löwenstein. *Lernen der Handhabung von Alltagsgegenständen im Kontext eines Service-Roboters*. Doktorarbeit, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, 2008.

[4] A. Beyeler, J.-C. Zufferey, und D. Floreano. 3D Vision-based navigation for indoor microflyers. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07)*, Seiten 1336–1341, Omnipress, Madison, USA, 10-14 Apr. 2007.

[5] P. Biber und T. Duckett. Dynamic maps for long-term operation of mobile service robots. In *Proc.: Robotics: Science and Systems I (RSS '05)*, Seiten 17–24, MIT Press, Cambridge, USA, 8-11 Jun. 2005.

[6] J. Bikker. Real-time ray tracing through the eyes of a game developer. In *Proc.: IEEE Symposium on Interactive Ray Tracing (RT '07)*, Seiten 1–10, IEEE Press, New York, 10. - 12. Sept. 2007.

[7] J. Borenstein und Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Journal Of Robotics And Automation*, 7(3):278–288, 1991.

[8] C. Braillon, C. Pradalier, J. Crowley, et al. Real-time moving obstacle detection using optical flow models. In *Proc.: IEEE Intelligent Vehicles Symposium*, Seiten 466–471, 2006.

[9] J. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25 – 30, 1965.

[10] A. Brooks, T. Kaupp, A. Makarenko, et al. Orca: a component model and repository. *Software Engineering for Experimental Robotic, Springer Tracts in Advanced Robotics, Berlin/Heidelberg*, 30:231 –251, 2007.

[11] A. Brooks und S. Williams. Tracking people with networks of heterogeneous sensors. In *Proc.: Australasian Conference on Robotics and Automation (ACRA '03)*, Seiten 1–7, 2003.

[12] S. Carpin, M. Lewis, J. Wang, et al. Usarsim: a robot simulator for research and education. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07)*, Seiten 1400–1405, Omnipress, Madison, USA, 2007.

[13] T. Collet, B. MacDonald, und B. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proc.: of the Australasian Conference on Robotics and Automation (ACRA '05)*, 2005.

[14] C. P. Connette, A. Pott, M. Hagele, et al. Control of an pseudo-omnidirectional, nonholonomic, mobile robot based on an ICM representation in spherical coordinates. In *Proc.: 47th IEEE Conference on Decision and Control (CDC '08)*, Seiten 4976–4983, 9–11 Dez. 2008.

[15] C. Cote, Y. Brosseau, D. Letourneau, et al. Robotic software integration using MARIE. *International Journal of Advanced Robotic Systems*, 3(1):55 – 60, 2006.

[16] J. Craighead, R. Murphy, J. Burke, et al. A survey of commercial & open source unmanned vehicle simulators. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07)*, Seiten 852–857, Omnipress, Madison, USA, 2007.

[17] F. Dellaert, D. Fox, W. Burgard, et al. Monte carlo localization for mobile robots. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '99)*, Band 2, Seiten 1322–1328, 1999.

[18] R. Duda und P. Hart. Use of the hough transformation to detect lines and curves in pictures. *Comm. ACM*, 15:11–15, January 1972.

[19] A. Eliazar und R. Parr. DP-SLAM 2.0. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '04)*, Band 2, Seiten 1314–1320, Omnipress, Madison, USA, 26 Apr.– 1 Mai. 2004.

[20] M. Fischler und R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM*, 24(6):381–395, 1981.

[21] A. Fod, A. Howard, und M. Mataric. A laser-based people tracker. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '02)*, Band 3, Seiten 3024–3029, 2002.

[22] D. Fox. Adapting the sample size in particle filters through KLD-sampling. *International Journal of Robotics Research (IJRR '03)*, 22(12):985–1003, 2003.

[23] D. Fox, W. Burgard, und S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[24] D. Fox, W. Burgard, und S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195 – 207, 1998.

[25] M. Fritzsche, E. Schulenburg, N. Elkmann, et al. Safe human-robot interaction in a life science environment. In *Proc.: IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR '07)*, Seiten 1–6, 2007.

[26] B. Gerkey, R. Vaughan, K. Stoy, et al. Most valuable player: a robot device server for distributed control. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '01)*, Band 3, Seiten 1226–1231, 2001.

[27] G. Grisetti, C. Stachniss, und W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb. 2006.

[28] J. Gutmann, T. Weigel, und B. Nebel. A fast, accurate and robust method for self-localization in polygonal environments using laser range finders. *Advanced Robotics*, 14(8):651 – 667, 2001.

[29] J.-S. Gutmann und C. Schlegel. AMOS: comparison of scan matching approaches for self-localization in indoor environments. In *Proc.: First Euromicro Workshop on Advanced Mobile Robot*, Seiten 61–67, 1996.

[30] D. Hähnel, W. Burgard, D. Fox, et al. An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, Band 1, Seiten 206–211, 2003.

[31] J. Hertzberg, K. Lingemann, C. Lörken, et al. Does it help a robot navigate to call navigability an affordance? In *Towards Affordance-Based Robot Control*, Seiten 16–26, Springer Berlin / Heidelberg, 2008.

[32] D. Holz, C. Lörken, und H. Surmann:. Continuous 3D sensing for navigation and SLAM in cluttered and dynamic environments. In *Proc.: 11th Intl. Conf. Information Fusion (Fusion '08)*, Seiten 1–7, 30 Jun. – 3 Jul. 2008.

[33] B. Huhle, P. Jenke, und W. Strayer. On-the-fly scene acquisition with a handy multisensor-system. *International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, 5(3/4):255–263, 2007.

[34] G. Kaminka, M. Veloso, S. Schaffer, et al. GameBots: A flexible test bed for multiagent team research. *Comm. ACM*, 45(1):43–45, January 2002.

[35] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '85)*, Band 2, Seiten 500–505, 1985.

[36] I. Kim. UTBot: A virtual agent platform for teaching agent system design. *Journal of Multimedia*, 2(1):48 – 53, 2007.

[37] A. Kitanov, S. Bišević, und I. Petrović. Mobile robot self-localization in complex indoor environments using monocular vision and 3D model. In *Proc.: IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Seiten 1–6, 4-7 Sept. 2007.

[38] R. Kohout. Challenges in real-time obstacle avoidance. In *Proc. AAAI Spring Symposium on Real-Time Autonomous Systems*, 2000.

[39] J. Kramer und M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[40] F. Lamiraux, D. Bonnafous, und L. O. Reactive path deformation for nonholonomic mobile robots. *Transactions on Robotics*, 20:967 – 977, 2004.

[41] R. Lange. *3D Time-of-flight distance measurement with custom solid-state image sensors in CMOS/CCD-technology.* Doktorarbeit, Universiät Siegen, 2000.

[42] S. M. LaValle. *Planning Algorithms*, Kapitel 2, Seiten 377 –381. Cambridge University Press, 2006.

[43] W. E. Lorensen und H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc.: Association for Computing Machinery's Special Interest Group on Graphics and Interactive Techniques (ACM SIGGRAPH '87)*, Band 21, Seiten 163–169, ACM Press, New York, USA, Jul. 1987.

[44] F. Lu und E. Milios. Robot pose estimation in unknown environments by matching 2D range scans. In *Proc.: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, Seiten 935–938, 1994.

[45] F. Martín, V. Matellán, J. M. Cañas, et al. Visual based localization for a legged robot. In *Proc.: RoboCup 2005: Robot Soccer World Cup IX*, Seiten 708 –715, 2006.

[46] S. May. *3D Time-of-Flight Ranging for Robotic Perception in Dynamic Environments.* Doktorarbeit, Universität Osnabrück, 2008.

[47] J. Minguez. The obstacle-restriction method for robot obstacle avoidance in difficult environments. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, Seiten 2284–2290, 2005.

[48] J. Minguez und L. Montano. Nearness diagram navigation (ND): a new real time collision avoidance approach. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, Band 3, Seiten 2094–2100, 2000.

[49] J. Minguez und L. Montano. Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios. *IEEE Journal Of Robotics And Automation*, 20(1):45–59, 2004.

[50] J. Minguez, J. Osuna, und L. Montano. A "divide and conquer" strategy based on situations to achieve reactive collision avoidance in troublesome scenarios. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '04)*, Band 4, Seiten 3855–3862, Omnipress, Madison, USA, 2004.

[51] N. C. Mitsou und C. S. Tzafestas. Temporal occupancy grid for mobile robot dynamic environment mapping. In *Proc.: Mediterranean Conference on Control & Automation (MED '07)*, Seiten 1–8, 2007.

[52] M. Montemerlo, N. Roy, und S. Thrun. Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (CARMEN) Toolkit. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, Band 3, Seiten 2436–2441, 27–31 Oct. 2003.

[53] M. Montemerlo, S. Thrun, und W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '02)*, Band 1, Seiten 695–701, 2002.

[54] O. M. Mozos, P. Jensfelt, H. Zender, et al. From labels to semantics: An integrated system for conceptual spatial representations of indoor environments for mobile robots. In *Proc.: IEEE/RSJ (IROS '07) Workshop: Semantic information in robotics*, 2007.

[55] A. Murarka, J. Modayil, und B. Kuipers. Building local safety maps for a wheelchair robot using vision and lasers. In *Proc.: The 3rd Canadian Conference on Computer and Robot Vision (CRV '06)*, Seiten 25–33, 2006.

[56] T. R. Neumann und H. H. Bülthoff. Behavior-oriented vision for biomimetic flight control. In *Proc.: EPSRC/BBSRC International Workshop on Biologically Inspired Robotics*, Seiten 196–203, 14-16 Aug. 2002.

[57] V. Nguyen, A. Harati, und R. Siegwart. A lightweight SLAM algorithm using orthogonal planes for indoor mobile robotics. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, Seiten 658–663, 2007.

[58] A. Nüchter, K. Lingemann, J. Hertzberg, et al. 6D SLAM - 3D mapping outdoor environments. *Journal of Field Robotics*, 24(8/9):699–722, 2007.

[59] A. Nüchter, H. Surmann, und J. Hertzberg. Automatic model refinement for 3D reconstruction with mobile robots. In *Proc.: Fourth International Conference on 3-D Digital Imaging and Modeling (3DIM '03)*, Seiten 394–401, 2003.

[60] A. Nüchter, O. Wulf, K. Lingemann, et al. 3D Mapping with semantic knowledge. In *Proc.: RoboCup International Symposium*, Seiten 335–346, 18–19 Jul. 2005.

[61] E. Pacchierotti, H. Christensen, und P. Jensfelt. Evaluation of passing distance for social robots. In *Proc.: 15th IEEE International Symposium on Robot and Human Interactive Communication (ROMAN '06)*, Seiten 315–320, 2006.

[62] L. Paletta, S. Frintrop, und J. Hertzberg. Robust localization using context in omnidirectional imaging. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '01)*, Band 2, Seiten 2072–2077, 2001.

[63] R. Philippsen und R. Siegwart. An interpolated dynamic navigation function. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '05)*, Seiten 3782–3789, Omnipress, Madison, USA, 2005.

[64] S. Quinlan und O. Khatib. Elastic bands: connecting path planning and control. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '93)*, Seiten 802–807, 1993.

[65] T. Rabbani. *Automatic Reconstruction of Industrial Installations Using Point Clouds and Images*. Doktorarbeit, NCG, Netherlands Geodetic Commission, Delft, The Netherlands, 2006.

[66] R. Rajamani. *Vehicle Dynamics and Control*. Springer, New York, USA, 2006.

[67] R. Rusu, A. Maldonado, M. Beetz, et al. Extending player/stage/gazebo towards cognitive robots acting in ubiquitous sensor-equipped environments. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07) Workshop on Network Robot Systems*, Omnipress, Madison, USA, 2007.

[68] S. Sanan, D. Santani, K. Krishna, et al. Extension of reeds and shepp paths to a robot with front and rear wheel steer. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '06)*, Seiten 3730–3735, Omnipress, Madison, USA, 2006.

[69] H. Schäfer, A. Hach, M. Proetzsch, et al. 3D obstacle detection and avoidance in vegetated off-road terrain. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '08)*, Seiten 923–928, Omnipress, Madison, USA, 2008.

[70] S. Scherer, S. Singh, L. Chamberlain, et al. Flying fast and low among obstacles. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07)*, Seiten 2023–2029, Omnipress, Madison, USA, 2007.

[71] T. Scherer. *A mobile service robot for automisation of sample taking and sample management in a biotechnological pilot laboratory*. Doktorarbeit, Universität Bielefeld, Technische Fakultät, 2005.

[72] R. Schnabel, R. Wahl, R. Wessel, et al. Shape recognition in 3D point clouds. Forschungsbericht, Universität Bonn, Institut für Informatik II, Fachbereich Computer Graphik, 2007.

[73] E. Schulenburg, N. Elkmann, M. Fritzsche, et al. LiSA: a robot assistant for life sciences. In *KI 2007: Advances in Artificial Intelligence. Proc.: 30th Annual German Conference on AI (KI '07)*, Band 4667 of *LNAI*, Seiten 502–505, Springer, Berlin/Heidelberg, 2007.

[74] D. Schulz, W. Burgard, D. Fox, et al. Tracking multiple moving objects with a mobile robot. In *Proc.: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '01)*, Band 1, Seiten 371 – 377, 2001.

[75] D. Schulz, W. Burgard, D. Fox, et al. People tracking with a mobile robot using sample-based joint probabilistic data association filters. *International Journal of Robotics Research (IJRR)*, 22(2):99–116, 2003.

[76] S. Se, D. Lowe, und J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '01)*, Band 2, Seiten 2051 – 2058, 2001.

[77] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '96)*, Seiten 3375 – 3382, 1996.

[78] C. Stachniss und W. Burgard. Mobile robot mapping and localization in non-static environments. In *Proc.: National Conference on Artificial Intelligence*, Pittsburgh, PA, USA, 2005.

[79] S. Stiene und J. Hertzberg. Sicheres Navigieren in dynamischen Umgebungen mit 3D-Kollisionsvermeidung. In *Fachgespräche Autonome Mobile Systeme*, 2007.

[80] S. Stiene und J. Hertzberg. Virtual range scan for avoiding 3D obstacles using 2D tools. (eingereicht 2009).

[81] S. Stiene, A. Nüchter, K. Lingemann, et al. An experiment in semantic correction of sensor data. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: Workshop on Semantic Information in Robotics at the IEEE International Conference Robotics and Automation (ICRA '07)*, Omnipress, Madison, USA, 2007.

[82] H. Surmann, K. Lingemann, A. Nüchter, et al. A 3D laser range finder for autonomous mobile robots. In *Proc.: 32nd International Symposium on Robotics (ISR '01)*, Band 1, Seiten 153 – 158, Seoul, Korea, April 2001.

[83] H. Surmann, A. Nüchter, und J. Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Journal Robotics and Autonomous Systems (JRAS)*, 45(3-4):181–198, 2003.

[84] S. Thrun, W. Burgard, und D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, Massachusetts, London, England, 2005.

[85] S. Thrun, M. Montemerlo, H. Dahlkamp, et al. Stanley: The robot that won the DARPA grand challenge. *J. Robot. Syst.*, 23(9):661–692, 2006.

[86] I. Ulrich und J. Borenstein. VFH$^+$: reliable obstacle avoidance for fast mobile robots. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '98)*, Band 2, Seiten 1572–1577, 1998.

[87] I. Ulrich und I. Nourbakhsh. Appearance-based place recognition for topological localization. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '00)*, Seiten 1023 –1029, 2000.

[88] R. Vaughan, B. Gerkey, und A. Howard. On device abstractions for portable, reusable robot code. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, Band 3, Seiten 2421–2427, 2003.

[89] D. Vikenmark und J. Minguez. Reactive obstacle avoidance for mobile robots that operate in confined 3D workspaces. In *Proc.: IEEE Mediterranean Electrotechnical Conference (MELECON '06)*, Seiten 1246–1251, 2006.

[90] R. Volpe, I. Nesnas, T. Estlin, et al. The CLARAty architecture for robotic autonomy. In *Proc.: IEEE Aerospace Conference*, Band 1, Seiten 121 – 132, 2001.

[91] R. Wahl und M. G. R. Klein. Identifying planes in point clouds for efficient hybrid rendering. In *Proc.: The 13th Pacific Conference on Computer Graphics and Applications (Pacific Graphics '05)*, 2005.

[92] C.-C. Wang und C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *Proc.: IEEE International Conference on Robotics and Automation (ICRA '02)*, Band 3, Seiten 2918–2924, Omnipress, Madison, USA, 2002.

[93] J. Weingarten und R. Siegwart. 3D SLAM using planar segments. In *Proc.: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, Seiten 3062–3067, 2006.

[94] T. Wiemann. *Automatische Rekonstruktion Planarer 3D-Umgebungen*. Masterarbeit, Universität Osnabrück, Institut für Informatik, Fachbereich Wissensbasierte Systeme, 2007.

[95] O. Wulf, C. Brenneke, und B. Wagner. Colored 2D maps for robot navigation with 3D sensor data. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*, Band 3, Seiten 2991–2996, 2004.

[96] J. Xavier und U. Nunes. Multi-robot interaction using finite state grammars. In L. S. Lopes, T. Belpaeme, und S. J. Cowley, (Hrsg.), *Proc.: Language and Robots*, Seiten 128–135, 10. –12. Dez. 2007.

[97] B.-D. Yim, Y.-J. Lee, J.-B. Song, et al. Mobile robot localization using fusion of object recognition and range information. In S. Hutchinson, N. Amato, S. Chiaverini, et al., (Hrsg.), *Proc.: IEEE International Conference on Robotics and Automation (ICRA '07)*, Seiten 3533–3538, Omnipress, Madison, USA, 2007.

Internet Quellen

[98] LiSA Assistenzroboter in Laboren von Life-Science Unternehmen http://www.lisa-roboter.de. [Stand 19.3.2009].

[99] Produktblätter und Pressemitteilungen des Fraunhofer IPA Care-O-bot 3 http://www.care-o-bot.de. [Stand 19.3.2009].

[100] Universitäre Forschungsaktivitäten der Firma NEOBOTIX http://www.neobotix.de/en/applications/research/Universities.html. [Stand 19.3.2009].