Zusammenfassung

Die autonome Navigation stellt neben der Interaktionsfähigkeit eine Grundlage für die Funktion eines mobilen Serviceroboters dar. Wichtige Teilleistungen sind dabei die Selbstlokalisation, die Pfadplanung und die Bewegungssteuerung unter Vermeidung von Kollisionen. Eine Voraussetzung für viele Navigationsaufgaben ist zudem die Erstellung eines Umgebungsmodells aus sensorischen Beobachtungen, unter Umständen in Verbindung mit einer selbständigen Exploration. Diese Teilprobleme wurden in der vorgelegten Arbeit vor dem Hintergrund der Entwicklung eines interaktiven mobilen Shopping-Lotsen bearbeitet, welcher Kunden eines Baumarktes Informationen zu Produkten zur Verfügung stellen und sie auf Wunsch zum Standort der gesuchten Waren führen kann.

Den methodischen Kern der Arbeit bildet die initiale Umgebungskartierung. Dafür wurde ein Verfahren zum Simultaneous Localization and Mapping (SLAM) entwickelt, welches im Gegensatz zu vergleichbaren Ansätzen nicht auf den Einsatz hochgenauer Laser-Range-Scanner ausgerichtet ist. Stattdessen wurden hauptsächlich Sonar-Sensoren benutzt, die sich durch eine wesentlich geringere räumliche Auflösung und höhere Messunsicherheit auszeichnen. Der entwickelte Map-Match-SLAM-Algorithmus beruht auf dem bekannten Rao-Blackwellized Particle Filter (RBPF), welcher mit einer lokalen Karte zur Repräsentation der aktuellen Umgebungsbeobachtungen sowie einer Map-Matching-Methode zum Vergleich der lokalen und globalen Karte kombiniert wurde. Durch eine speichereffiziente Darstellung der globalen Karte und dynamische Adaption der Partikel-Anzahl ist trotz der aus den sensorischen Beschränkungen resultierenden großen Zustandsunsicherheit die Online-Kartierung möglich.

Durch die Transformation der Beobachtungen in eine lokale Karte und die sensorunabhängige Bewertungsfunktion ist das Map-Match-SLAM-Verfahren für ein breites Spektrum unterschiedlicher Sensoren geeignet. Dies wurde exemplarisch durch die Kartierung unter Nutzung einer Stereo-Kamera-Anordnung und einer einfachen Kamera in Verbindung mit einem Depth-from-Motion-Verfahren gezeigt. Aufbauend auf dem Kartierungsalgorithmus wurde zudem ein SLAM-Assistent entwickelt, welcher während der Kartierungsphase Aktionsvorschläge für den menschlichen Bediener präsentiert, die eine optimale Funktion des SLAM-Algorithmus gewährleisten. Der Assistent stellt damit eine Zwi-

schenstufe zwischen rein manueller Steuerung und komplett autonomer Exploration dar.

Einen weiteren Schwerpunkt der Arbeit stellen die Verfahren für die autonome Funktion des Roboters dar. Für die Selbstlokalisation wird ebenso wie beim SLAM ein Map Matching mit lokalen Karten eingesetzt. Eine Verbesserung der Robustheit und Genauigkeit wird durch die Kombination dieses Ansatzes mit einem vorhandenen visuellen Selbstlokalisations-Verfahren auf Basis einer omnidirektionalen Kamera erzielt.

Für die Bestimmung des optimalen Pfades zu einem Zielpunkt kommen Standard-Algorithmen zur Pfadsuche in Graphen zum Einsatz, die Zellen der Karte werden dazu als Graphknoten interpretiert. Die Arbeit präsentiert vergleichende Untersuchungen zur Effizienz von Algorithmen mit und ohne Suchheuristik (A*/Dijkstra-Algorithmus) in der konkreten Einsatzumgebung. Für die Bewegungssteuerung und Kollisionsvermeidung wurden zwei verschiedene Algorithmen entwickelt: Einem reaktiven Verfahren, welches eine Weiterentwicklung des bekannten Vector Field Histogram (VFH) darstellt, wird ein neues antizipatives Verfahren auf Basis von Sampling und stochastischer Suche im Raum der möglichen Bewegungstrajektorien gegenüber gestellt und experimentell verglichen.

Die entwickelten Methoden kommen auf mehreren Shopping-Robotern zum Einsatz, die sich seit ca. sechs Monaten im dauerhaften öffentlichen Testbetrieb in einem Baumarkt befinden. Neben den Navigationsmethoden gibt die Arbeit einen Überblick über die weiteren Module des Roboters, z.B. für die Nutzer-Interaktion, und beschreibt detailliert die Steuerarchitektur zur Koordinierung der Teilleistungen. Die Eignung aller eingesetzten Methoden für den Einsatz in einer realen Anwendung und die hohe Akzeptanz der Nutzer für das entwickelte Gesamtsystem werden durch die Auswertung von Langzeittests nachgewiesen.

Abstract

Autonomous navigation, in addition to interaction, is a basic ability for the operation of a mobile service robot. Here, important subskills are selflocalization, path planning, and motion control with collision avoidance. A further pre-condition for many navigation tasks ist the generation of an environment model from sensor observationa, often in combination with autonomous exploration. In this thesis, these challenges are considered in the context of the development of an interactive mobile shopping guide, which is able to provide information about the shop's products to customers of a home improvement store and guide them to the respective location.

The focus of this work lies on the initial environment mapping. A method for Simultaneous Localization and Mapping (SLAM) has been developed, which in contrast to other comparable approaches does not assume the use of high-precision laser range scanners. Instead, sonar range sensors are used mainly, which feature an inferior spatial resolution and increased measurement noise. The resulting Map-Match-SLAM algorithm is based on the well known Rao-Blackwellized Particle Filter (RBPF), in combination with local maps for representation of most recent observations and and a map matching function for comparison of local and global maps. By adding a memory-efficient global map representation and dynamic adaption of the number of particles, online mapping is possible even under high state uncertainty resulting from the sensor characteristics.

The use of local maps for representation of the observations and the sensor-independent weighting function make Map-Match-SLAM applicable for a wide range of different sensors. This has been demonstrated by mapping with a stereo camera and with a single camera, in combination with a depth-from-motion algorithm for pre-processing. Furthermore, a SLAM assistant has been developed, which is generating direction hints for the human operator during the mapping phase, in order to ensure a route that enables optimal operation of the SLAM algorithm. The assistant represents an intermediate step between purely manual mapping and completely autonomous exploration.

A second main part of the work presented here are methods for the autonomous operation of the robot. For selflocalization, a map matching approach with local maps is used, similar to the proposed SLAM algo-

rithm. Improvements of robustness and precision are achieved in combination with an existing visual localization approach which is using omnidirectional camera images.

Path planning is done by the utilization of standard graph search algorithms. To that purpose, the grid cells of the global map are regarded as graph nodes. Comparitive analysis is presented for search algorithms with and without heuristics (A*/Dijkstra algorithm), for the specifics of typical operation areas. Two different algorithms have been developed for motion control and collision avoidance: A reactive method, which is an enhancement of the existing Vector Field Histogram (VFH) approach, is experimentally compared with a new anticipative method based on sampling and stochastic search in the trajectory space.

All the developed methods are employed on a team of shopping robots, which have been in permanent public test operation in a home improvement store for six months currently. The description of navigation methods is complemented by an overview of further software componentsof the robots, e.g. for Human-Robot-Interaction, and a detailed description of the control architecture for coordination of the subsystems. Analysis of long term test operation proves that all the applied methods are suitable for real world applications and that the robot is accepted and regarded as a valuable service by the customers.

Danksagung

Mein herzlicher Dank gilt allen, die mir bei und während der Promotion mit Rat und Hilfe zur Seite standen.

Zunächst möchte ich mich bei meinem Betreuer und Leiter des Fachgebietes Neuroinformatik und Kognitive Robotik, Prof. Dr. Horst-Michael Groß bedanken für die Möglichkeit, die vergangenen Jahre in einem ausgesprochen angenehmen Umfeld zu arbeiten. Seine wertvollen Anregungen in unseren fachlichen Diskussionen haben mich oft dazu gebracht, meine eigenen Vorstellungen zu hinterfragen und so zu neuen Erkenntnissen zu gelangen.

Herrn PD Hans-Joachim Böhme als langjährigem Projektleiter für die Entwicklung des Shopping-Roboters möchte ich ebenfalls für viele fachliche und auch persönliche Gespräche meinen wärmsten Dank sagen. Hans, ich weiß wir haben Dein Vertrauen in uns manchmal arg strapaziert, trotzdem hast Du uns gegen alle Zweifel in Schutz genommen und auch für unsere Versäumnisse immer eine gute Entschuldigung gefunden. Sei Dir gewiss, dass wir das sehr zu schätzen wussten.

Auch bei meinen unmittelbaren Projekt-Mitstreitern Alexander König, Steffen Müller, Torsten Wilhelm und Christian Martin möchte ich mich sehr herzlich bedanken. Obwohl so manche Deadline uns große Anstrengungen abverlangte, hat die Arbeit mit Euch immer viel Spaß gemacht. Weiterhin möchte ich auch allen anderen Mitarbeitern des Fachgebietes für die fachliche, technische und organisatorische Unterstützung sowie die äußerst kollegiale Atmosphäre meinen Dank aussprechen: Andrea Scheidig, Klaus Debes, Ute Schütz, Heike Groß, Sabine Schulz, Sven Hellbach, Erik Schaffernicht, Erik Einhorn, Jens Kessler, Carsten Schauer, Volker Stephan, Dima Surmeli.

Besonderer Dank gilt allen Studenten, die durch ihre Unterstützung zum erfolgreichen Abschluss dieser Arbeit beigetragen haben, insbesondere Axel Schmiegel, Holger Täubig, Birthe Babies, Matthias Serfling, Kai Merten, Martin Rulsch, Markus Hahn, Sebastian Keichel, Ronny Stricker, Alexander Trott, Matthias Höchemer und Christian Vollmer.

Eine ganz besondere Hervorhebung verdient Sören Kalesse: Er hat sich in den vergangenen Jahren nicht nur als studentische Hilfskraft sowie im Rahmen von Praktikumssemester, Studien- und Diplomarbeit und weit darüber hinaus in den Projekten des Fachgebietes engagiert, sondern

ist mir in dieser Zeit auch zu einem guten Freund geworden. Mit seiner fachlichen Kompetenz, seinem unermüdlichen Einsatz und seiner stets positiven Ausstrahlung hat er großen Anteil am Erfolg unseres Projektes und am Gelingen dieser Arbeit.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Mobile Serviceroboter 1

1.2 Zur Einordnung der Arbeit: Das Projekt SerRoKon - Öffentlicher Einkaufs-Assistent im Baumarkt 2

1.3 Fokus der Arbeit: Navigation in der Indoor-Umgebung . 4

1.4 Anspruch der Arbeit 7

1.5 Gliederung der Arbeit 8

2 Kartenmodell **13**

2.1 Umgebungsmodelle 14

2.1.1 Geometrische Karten 16

2.1.2 Gridkarten 17

2.1.3 Regionenkarten 18

2.1.4 Topologische Karten 19

2.1.5 Semantische Karten 20

2.2 Kartenmodelle in dieser Arbeit 21

3 Grundlagen des Kartenaufbaus **23**

3.1 Gridkarten-Berechnung aus Beobachtungen 24

3.1.1 Zellen-Belegtheits-Schätzung mittels Entfernungs-Sensoren 24

3.1.2 Verrechnung mehrerer unabhängiger Schätzungen 31
3.1.3 Zellen-Belegtheits-Schätzung mittels 3D-Punktkoordinaten (visuell beobachtbaren Landmarken) 33
3.1.4 Karten-Fusion 38
3.2 Odometriekorrektur 40
3.2.1 Bestimmung des systematischen Fehlers 41
3.2.2 Orientierungs-Korrektur mittels Bodenstruktur 43

4 SLAM - Simultaneous Localization And Mapping 51
4.1 Positionskorrektur durch Map Matching 53
4.2 Probabilistic SLAM 59
4.3 Map-Match-SLAM mit entfernungsmessenden Sensoren . 69
4.3.1 Rao-Blackwellized Particle Filter für Map-Match-SLAM 69
4.3.2 Gemeinsame Kartenrepräsentation (Shared Gridmaps) 79
4.3.3 Nicht-persistente Karte (On-Demand Maps) . . . 87
4.3.4 Dynamisierung der Partikelanzahl 89
4.4 Map-Match-SLAM mit visuell detektierten 3D-Punkten 99
4.4.1 Map-Match-SLAM mit Stereo-Kamera 99
4.4.2 Map-Match-SLAM mit Depth-from-Motion 104
4.5 Der SLAM-Assistent 106
4.6 Zusammenfassung und Bewertung des Map-Match-SLAM 114

5 Autonome Navigation 117
5.1 Selbstlokalisation 118
5.1.1 Map-Match-Selbstlokalisation 119

5.1.2 Visuelle Selbstlokalisation 121
5.2 Navigationsystem 123
5.3 Globale Navigation - Pfadplanung 126
5.3.1 Pfadsuche auf 2D-Karten: Vergleich von Dijkstra- und A*-Algorithmus . . . 126
5.3.2 Planung im erweiterten Zustandsraum 131
5.4 Lokale Navigation 131
5.4.1 Vector Field Histogram 133
5.4.2 Multi-Trajektorien-Navigation 144
5.4.3 Lokale Pfadplanung 156
5.4.4 Experimenteller Vergleich 159

6 Integration in das Roboter-Gesamtsystem 165
6.1 Implementationsdetails 166
6.2 Komponenten der Applikation „Shopping-Assistent“ - Nutzer-Interaktion, Verhalten und Anwendungslogik 173
6.3 Bewertung des Gesamtsystems 180

7 Weiterführende Arbeiten 183
7.1 Visuelle Hindernisdetektion 184
7.2 Dynamische Kartenadaption 189

8 Zusammenfassung und Ausblick 205
8.1 Zusammenfassung 205
8.2 Ausblick 209

Anhänge 213

A Bestimmung des systematischen Odometriefehlers für einen Synchro-Drive-Antrieb 213

B Probabilistische Zustandsschätzung 221
B.1 Rekursive Bayes-Schätzung 223
B.2 Kalmanfilter 226
B.3 Partikelfilter 232

C Visuelle Tiefenschätzung 235
C.1 Tiefenschätzung mit Stereo-Kamera 238
C.2 Monokulare Tiefenschätzung mittels Depth-from-Motion 242
C.2.1 Grundlagen: Homogene Koordinaten 243
C.2.2 Epipolargeometrie 245
C.2.3 Featuredetektion und -tracking 248
C.2.4 Erweiterter Kalmanfilter 251

D Bahnregler zur Trajektorienverfolgung 255
D.1 PID-Regler 256
D.2 PID-Regler mit Vorsteuerung 256
D.3 Backstepping-Controller 258

E SLAM-HP - FastSLAM-Variante für Bearing-Only-SLAM 259

Abkürzungsverzeichnis 267

Abbildungsverzeichnis 269

Literaturverzeichnis 275

Kapitel 1

Einleitung

1.1 Mobile Serviceroboter

Die Entwicklung autonomer mobiler Roboter hat in den letzten Jahren stark an Bedeutung gewonnen. Weltweit ist die Robotik ein stark wachsender Forschungszweig, wobei nach einfachen automatischen Helfern wie Transport-, Reinigungs- und Überwachungsgeräten vor allem so genannte Serviceroboter zunehmend Aufmerksamkeit erfahren. Ziel der Forschung ist die Entwicklung „intelligenter“ Assistenten, die im Dialog mit dem Benutzer diesem Serviceleistungen erbringen oder gar in Zusammenarbeit Aufgaben gemeinsam lösen können. Diese reichen von einfachen mobilen Informationsterminals in belebten öffentlichen Räumen wie z.B. Museen, Flughäfen, Bahnhöfen, aber auch im Einzelhandel, bis hin zu universellen Haushaltshilfen und künstlichen Gefährten, welche z.B. auch manipulatorische Fähigkeiten besitzen sollten und sich an den jeweiligen Nutzer anpassen, um eine persönliche Beziehung herzustellen.

Im folgenden werden exemplarisch einige Roboter aufgeführt, welche bereits über einen signifikanten Zeitraum im öffentlichen Einsatz und in „natürlicher“ Interaktion mit nicht eingewiesenen Nutzern diesen bestimmte Serviceleistungen zur Verfügung stellten (die Kategorisierung anhand der zugrunde gelegten Kriterien ist dabei bei vielen, auch hier nicht genannten, Installationen mehr oder weniger fließend, die Auflistung erhebt bei Weitem keinen Anspruch auf Vollständigkeit):

- Rhino: 6 Tage Museumsführer im Deutschen Museum Bonn, 1997 [Burgard et al., 1999]
- Minerva: 2 Wochen Museumsführer im Smithsonian Museum of American History, Washington D.C., 1998 [Thrun et al., 2000]
- Chips, Sweetlips, Joe Historybot: 5 Jahre Museumsführer im Carnegie Museum of Natural History bzw. Heinz History Center Pittsburgh, 1998 - 2002 [Nourbakhsh et al., 2003]
- „Komm-rein“, „Also-gut“und „Mach-was“: Empfang, Unterhaltung und Information der Besucher im Museum für Kommunikation Berlin, seit 2000 [Graf und Barth, 2002]
- Grace: AAAI Robot Challenge - Teilnahme als Redner an der AAAI National Conference on Artificial Intelligence, 2002 [Simmons et al., 2003]
- Robox: 5 Monate Ausstellungsführer auf der Schweizer Nationalausstellung expo, 2002 [Siegwart et al., 2003]
- Robovie: u.a. 2 Wochen Interaktionspartner und Englisch-„Lehrer“ in einer japanischen Grundschule, 2003 [Kanda et al., 2004], 2 Monate Museumsführer im Osaka Science Museum, 2005 [Shiomi et al., 2007]

Während viele der vorgestellten Systeme zunächst Demonstrations- und Experimentalcharakter aufwiesen, etablieren sich zunehmend auch kommerzielle Anbieter nicht nur für Forschungs- und Experimental-Plattformen, sondern auch für Servicerobotik-Lösungen unter den genannten Rahmenbedingungen [World Robotics, 2008].

1.2 Zur Einordnung der Arbeit: Das Projekt SerRoKon - Öffentlicher Einkaufs-Assistent im Baumarkt

Am Fachgebiet Neuroinformatik und Kognitive Robotik (NIKR) der TU Ilmenau wurde im Zeitraum 2000 bis 2007, seit 2003 in enger Zusammenarbeit mit der Firma MetraLabs GmbH [MetraLabs, 2008], das

Abbildung 1.1: **Links, Mitte:** *Der Shopping-Assistent SCITOS A5, eine gemeinsame Entwicklung des Fachgebiets Neuroinformatik und Kognitive Robotik der TU Ilmenau und der MetraLabs GmbH Ilmenau.* **Rechts:** *Der Service-Roboter PERSES, eine erweiterte B21r-Plattform, wird ebenfalls für Methodenentwicklung und Experimente benutzt.*

Projekt „SerRoKon - Service-Roboter-Konzeption“[1] bearbeitet. Ziel des Projektes war die Entwicklung eines Service-Roboters, welcher als interaktiver Assistent in einem Baumarkt die Kunden bei ihrem Einkauf unterstützt, indem er den Weg zu Artikelstandorten beschreibt oder wahlweise direkt zum Regal führt, sowie Zusatzinformationen zum Angebot liefert. Gleichzeitig sollen damit die qualifizierten Mitarbeiter des Marktes von solchen einfachen Tätigkeiten entlastet werden, so dass ihnen mehr Zeit für kompetente Beratung bleibt und insgesamt der Service am Kunden verbessert wird. Mit der toom BauMarkt GmbH war ein potentieller Nutzer eines solchen Einkaufs-Assistenten als Partner in das Projekt eingebunden, so dass die Entwicklung neben wissenschaftlichen Fragestellungen auch an den Zielvorgaben des praktischen Einsatzes ausgerichtet werden konnte.

Von MetraLabs wurde in Zusammenarbeit mit den Partnern die Hardware-Plattform SCITOS A5 (Abb. 1.1) für den Roboter entworfen und umgesetzt, die aus der Differential-Drive-Antriebseinheit, sensorischer Ausstattung (Odometrie, Sonar, wahlweise Laser und Kameras) sowie Embedded-PC-Hardware besteht [Merten und Gross, 2008]. Das Fach-

[1]Förderung durch den Freistaat Thüringen, Thüringer Ministerium für Wissenschaft, Forschung und Kultur, FKZ B509-03007 (Serrokon-H, 2004-2006) und Thüringer Ministerium für Wirtschaft, Technologie und Arbeit / Thüringer Aufbaubank FKZ 2006-FE-0154 (Serrokon-D, 2006-2007)

gebiet NIKR entwickelte maßgeblich die kognitive Intelligenz, welche Methoden zur autonomen Navigation und zur Interaktion mit den Nutzern, aber auch die Steuerarchitektur und die Verhaltensdefinition (Applikationslogik) beinhaltet.

Das äußere Design wurde in Abstimmung zwischen den Partnern entwickelt, und zielt darauf ab, gleichzeitig den Eindruck einer gewissen Intelligenz zu vermitteln und die Neugier des Nutzers zu wecken. Zu diesem Zweck dient u.a. der bewegliche Kopf, welcher den Benutzer anvisiert, um dadurch die Aufmerksamkeit des Roboters zu signalisieren, und somit auf intuitive Weise die Interaktionsbereitschaft darstellt. Weiterhin wurde auf eine möglichst schlanke Verkörperung geachtet, um die Wahrnehmung als Maschine und eine potentielle Gefahr der Abschreckung zu minimieren.

Die erste Entwicklungsphase des Shopping-Assistent wurde im Juli 2007 mit der öffentlichen Vorstellung abgeschlossen. Derzeit befindet er sich in einer finalen Erprobungs- und Anpassungsphase und soll ab Mitte 2008 nach und nach in den Märkten der toom BauMarkt GmbH Einzug halten.

Neben dem SCITOS-Roboter wurde auch die am Fachgebiet in vorangegangenen Projekten aufgebaute Plattform PERSES (Abb. 1.1 rechts) für Methodenentwicklung und Untersuchungen benutzt. Bei PERSES handelt es sich um einen Roboter des Typs RWI B21r, welcher um ein Display und einen Kopfaufbau für Interaktionsexperimente erweitert wurde. Im Gegensatz zu SCITOS besitzt der B21r einen Synchro-Drive-Antrieb, was teilweise abweichende Eigenschaften zur Folge hat, wie in Kapitel 3 zu sehen sein wird.

1.3 Fokus der Arbeit: Navigation in der Indoor-Umgebung

Das Gesamtsystem Serviceroboter besteht aus Hardware und Software, welche sich in die Haupt-Bestandteile Navigation, Interaktion und die Anwendung (zur Erbringung des Service am Nutzer) aufteilen lässt. Das Zusammenspiel der Komponenten wird durch die Steuerarchitektur bestimmt. Der Nutzer nimmt vor allem die eigentliche Service-Applikation bewusst wahr. Die darunter liegenden Bestandteile für die Navigation

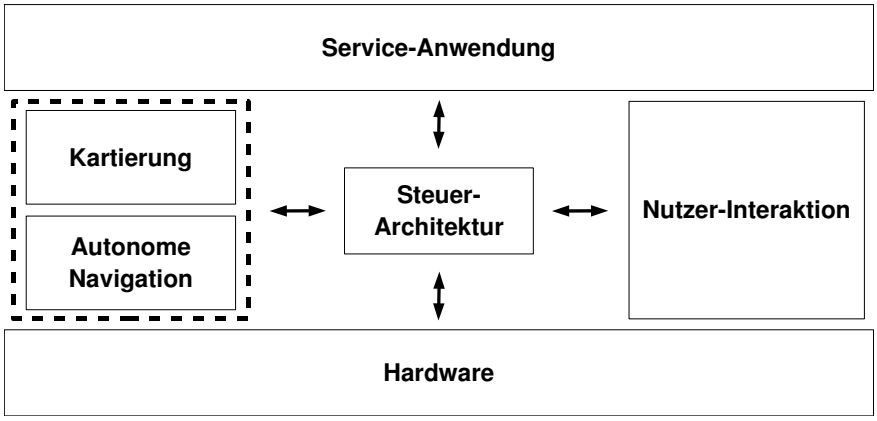

Abbildung 1.2: *Die Komponenten des Systems Serviceroboter: die Bestandteile, die den Fokus dieser Arbeit bilden, sind durch den Rahmen hervorgehoben.*

und Interaktion werden normalerweise als gegeben vorausgesetzt und fallen nur bei Fehlfunktionen oder unerwartetem Verhalten direkt auf. Gerade dadurch sind sie aber für die Gesamtfunktionalität von großer Bedeutung.

Für die Navigation können folgende wesentliche Teilaufgaben unterschieden werden:

Kartierung: Wenn der Roboter z.B. bestimmte Zielpunkte in der Umgebung anfahren soll, benötigt er dazu eine interne Repräsentation der Umgebung. Unter Umständen ist eine Karte des Einsatzgebietes bereits vorhanden (z.B. Gebäudeplan) und kann dem Roboter als digitale Karte mitgegeben werden. Dies ist jedoch nicht immer der Fall oder vorhandene Karten sind für verschiedene Aufgaben nur eingeschränkt geeignet, z.B. ist eine Selbstlokalisation nur möglich, wenn die in der Karte verzeichneten Objekte oder Umgebungseigenschaften durch die Sensorik des Roboters in der selben Form wahrnehmbar sind. Es ist daher sinnvoll, dass der Roboter die Karte aus seinen eigenen Sensormessungen erstellt, das geschieht meist im Rahmen einer Lernphase vor dem eigentlichen autonomen Einsatz. Falls nachträglich relevante Veränderungen in der Umgebung auftreten können, muss der Roboter entsprechend reagieren und die Repräsentation adaptieren.

Exploration: Falls der Roboter zu Beginn keine oder nur eine unvollständige Karte zur Verfügung hat, soll er normalerweise selbständig

die Umgebung erkunden. Ziel ist es dabei, möglichst das gesamte erreichbare Gebiet einmal zu durchstreifen, um z.B. eine komplette Karte zu erstellen.

Selbstlokalisation: Der Roboter muss seine eigene Position innerhalb der Umgebung bestimmen und während der Fahrt kontinuierlich nachverfolgen. Dies kann durch Installation spezieller eindeutiger Markierungen oder externer Positions-Messsysteme erfolgen, oder der Roboter kann eine vorhandene Karte und bordeigene Sensorik nutzen, um sich in der Umgebung zu orientieren.

Pfadplanung & Bewegungssteuerung: Eine der Grundaufgaben eines autonomen mobilen Roboters ist, sich zu einem vorgegebenen Punkt innerhalb seiner Einsatzumgebung zu begeben. Dazu muss er zunächst eine Handlungsfolge planen, und diese anschließend in eine Sequenz von Bewegungskommandos für das Antriebssystem umsetzen. Es muss normalerweise angenommen werden, dass der Roboter einerseits aufgrund von Ungenauigkeiten bei der Handlungsausführung nicht ideal einem Pfad folgen kann und andererseits Abweichungen der realen Umgebung von der internen Repräsentation auftreten, im vorliegenden Fall z.B. durch nicht erfasste Hindernisse wie Menschen. Daher ist es nicht möglich, eine komplette Bewegungssequenz zu berechnen und einfach abzuarbeiten, sondern der Roboter muss während der Bewegung stets seine Umgebung beobachten und seine eigene Position abgleichen, um seine Aktionen jeweils an die aktuelle Situation anzupassen. Je nach Algorithmus können die Planungs- und Ausführungsphase nicht immer streng getrennt werden, daher werden sie in dieser Übersicht in einem Punkt zusammengefasst.

Die vorliegende Dissertation ist fokussiert auf diese methodische Aspekte der Navigation des Shopping-Assistenten, wobei hier vor allem ein Verfahren zur Kartierung den Schwerpunkt bildet. Dieses ist zwar nicht direkt Bestandteil des autonomen Einsatzes, stellt jedoch eine wesentliche Vorleistung und damit einen wichtigen Bestandteil des Gesamtsystems dar. Neben den Navigationsverfahren werden zur Einordnung und zum Verständnis des Zusammenspiels der Einzelkomponenten die Steuerarchitektur sowie weitere funktionelle Bestandteile des Roboters in einem Überblick beschrieben.

1.4 Anspruch der Arbeit

Die Forschungstätigkeit am Fachgebiet NIKR, welche in der vorliegenden Arbeit dokumentiert wird, fand im Wesentlichen vor dem Hintergrund des SerRoKon-Projektes statt und bildet einen wesentlichen Grundpfeiler der Entwicklung des Shopping-Assistenten. Inhalt dieser Arbeit waren die Realisierung und Evaluierung (eines Großteils) der im vorigen Abschnitt beschriebenen Teilleistungen, welche notwendig sind, um den Roboter zu einer autonomen Navigation im Baumarkt zu befähigen. Die Auswahl und Spezifikation der genutzten Methodiken orientierte sich dabei naturgemäß eng sowohl am anvisierten Einsatzszenario, als auch an der Hardware-, insbesondere Sensorikausstattung des Shopping-Assistenten. Die Dissertationsschrift kann somit in Teilen auch als methodisch-funktionelle Beschreibung des entwickelten Shopping-Assistenten betrachtet werden.

Es ist leicht nachvollziehbar, dass die Entwicklung eines solch komplexen Systems in Teilschritten erfolgt: viele Probleme oder Unzulänglichkeiten bestimmter entworfener Lösungen werden erst im Praxistest offensichtlich. Der Erkenntnisgewinn aus dem Test führt damit zur Verbesserung und Erweiterung der Algorithmen oder zu gänzlich neuen Ansätzen. Die finale Lösung muss also im Kontext der Entwicklungsgeschichte betrachtet werden. Im Sinne der Plausibilität der Argumentation und um die Entwicklung bis zu den letztendlich eingesetzten Methoden nachvollziehbar zu machen, werden daher teilweise auch solche älteren Ansätze dargestellt, die sich als erfolglos oder unzureichend bei der Lösung mancher Probleme erwiesen haben, wobei die zeitliche Abfolge das Rückgrat für die inhaltliche Strukturierung bildet. Auf diese Weise soll auch die „Historie“ der Methoden deutlich gemacht werden. Darüber hinaus werden in einigen Abschnitten ebenfalls Ansätze vorgestellt, welche Erweiterungen oder Alternativen zum aktuell für den Baumarkt-Einsatz implementierten Stand darstellen oder die sich auf erweiterte Szenarien beziehen.

Insbesondere bei der Umgebungskartierung sind aktuell solche Methoden sehr verbreitet, die auf dem sensorischen Input von Laser-Range-Scannern aufbauen. Die sehr hohe Messgenauigkeit und räumliche Auflösung des Laser-Entfernungs-Messers ermöglicht nicht nur eine hoch detaillierte Beschreibung der Umgebung, sondern umgekehrt bei bekannter Umgebungskonfiguration eine sehr sichere Vorhersage der zu erwar-

tenden Beobachtung. Dadurch ist es u.a. möglich, durch Abgleich verschiedener Messungen (Scan Matching [Gutmann und Schlegel, 1996]) die relative Lage der Positionen, an denen diese aufgenommen wurden, recht genau zu bestimmen. Die spezifischen Eigenschaften des Sensors erlauben also Methoden, welche beim Einsatz von Entfernungssensoren mit geringerer Genauigkeit und Auflösung (Ultraschall, Radar) oder gänzlich anderer Sensorik, wie z.B. visueller Wahrnehmung, nur beschränkt übertragbar sind.

In dieser Arbeit wurde dagegen das Ziel verfolgt, Algorithmen zu entwickeln, welche nicht auf bestimmte Sensorik angewiesen sind, sondern möglichst einheitlich mit verschiedenen sensorischen Inputs arbeiten können. Dies ist einerseits dem Umstand geschuldet, dass zu Beginn der Entwicklung des Shopping-Assistenten die sensorische Ausstattung noch nicht komplett festgelegt war. Anfänglich wurde beispielsweise aus Kostengründen der vollständige Verzicht auf einen Laser-Range-Scanner erwogen, aktuell erscheint dies jedoch aufgrund der Sicherheitsanforderungen der Zulassungsbehörden nicht möglich, gleichzeitig sind neue Modelle und neue Anbieter am Markt verfügbar, die eine signifikante Verringerung der entsprechenden Kosten ermöglichen. Andererseits soll die Fusion der Beobachtungen mehrerer Sensorsysteme vereinfacht werden. Die Sensormessungen werden dazu in eine einheitliche Repräsentation (lokale Karte, Kapitel 2) transformiert, die als Basis für die weiteren Berechnungen genutzt wird. Als Referenzsensor wird zunächst eine Anordnung von Ultraschall(Sonar)-Entfernungsmessern angenommen, welche z.B. gegenüber dem Laser-Entfernungsmesser eine deutlich höhere Messvarianz und geringe Winkelauflösung aufweisen. Weiterhin wird in den Kapiteln 4 und 5 exemplarisch die Übertragung auf visuell detektierte Feature-Punkte dargestellt.

1.5 Gliederung der Arbeit

Die Arbeit ist folgendermaßen strukturiert:

In Kapitel 2 wird als Grundlage zunächst auf verschiedene bekannte Modelle zur Umgebungsrepräsentation eingegangen und die hauptsächliche Verwendung von Gridkarten als einheitliche Repräsentation und Abstraktion der Sensormessungen in dieser Arbeit motiviert.

Kapitel 3 behandelt ausführlich den Aufbau einer Gridkarte aus den Beobachtungen des Roboters. Dazu werden zunächst die grundlegenden Prinzipien zur Schätzung von Gridzell-Belegungen bei Nutzung verschiedener Sensoren (entfernungsmessend, visuelle Punktschätzungen) erklärt. Insbesondere wird eine sehr effiziente Implementierung der Zellbelegtheitsschätzung bei Verwendung beliebiger Entfernungssensoren erstmals vorgestellt. Zur Erstellung einer globalen Karte müssen die jeweiligen Beobachtungspositionen, an denen der Roboter die Messungen aufgenommen hat, berücksichtigt werden. Als Basis dafür dient zunächst die Odometrie (interne Bewegungsmessung und -integration). Da auch diese ungenau und fehlerbehaftet ist, werden in diesem Kapitel weiterhin Verfahren zur Verbesserung der Odometriemessungen und zur Positionskorrektur anhand externer Beobachtungen vorgestellt.

Darauf aufbauend wird in Kapitel 4 ein probabilistisches Verfahren zum „Simultaneous Localization and Mapping“ (SLAM) entwickelt, welches auf einem Rao-Blackwellized Particle Filter (RBPF) [Murphy, 1999] beruht. Als Sensorik werden dabei ausschließlich Sonar-Entfernungsmesser und Odometrie verwendet. Im Gegensatz zu bekannten RBPF-Gridkarten-Verfahren werden nicht direkt die Sensormessungen zur Bewertung von Hypothesen benutzt, sondern jeweils eine Menge von Messungen zu einer lokalen Karte vereint und die Hypothesen durch Abgleich der lokalen und globalen Karte (Map Matching) bewertet. Neben dem grundlegenden Prinzip werden Erweiterungen zur deutlichen Verbesserung der Zeit- und Speichereffizienz entwickelt (dynamische Partikelanzahl, verteilte Kartenrepräsentation, On-Demand-Mapping). Um die einfache Übertragung auf andere Sensoriken nachzuweisen, wird die Funktion bei Nutzung zweier verschiedener visueller 3D-Featurepunkt-Schätzer (Stereo, monokular) demonstriert. Weiterhin wird ein SLAM-Explorations-Assistent vorgestellt, welcher dem menschlichen Operator während der Kartierung Hilfestellung bei der Wahl einer für den Kartenbau günstigen Route geben soll.

Kapitel 5 widmet sich den eigentlichen autonomen Funktionen des Roboters: Die Selbstlokalisation mittels Partikelfilter in Kombination sowohl mit Map Matching (Entfernungssensoren) als auch Ansichts-Merkmalen (visuelle Sensoren) wird kurz beschrieben, anschließend werden verschiedene Verfahren der Pfadplanung und Bewegungssteuerung (reaktives Verfahren auf Basis eines erweiterten „Vector Field Histo-

gram“, antizipatives Verfahren durch Trajektorien-Sampling) untersucht.

Kapitel 6 beschreibt das Gesamtsystem des Shopping-Roboters und das Zusammenspiel der Einzelmodule für Navigation, Interaktion, Verhaltenssteuerung etc. Neben Implementationsdetails der Steuerarchitektur wird eine vorläufige Auswertung der Gesamtfunktion im Einsatzszenario vorgenommen. Während für die Einzelfunktionen jeweils ein Vergleich der Leistungsfähigkeit verschiedener Verfahren anhand messbarer metrischer Größen (z.B. Lokalisationsgenauigkeit, Fahrgeschwindigkeit, Rechenzeit etc.) möglich ist, muss sich die Bewertungsmetrik für das Gesamtsystem am angestrebten Nutzen orientieren: Entscheidend ist letzten Endes die Zufriedenheit der Anwender (hier: sowohl Baumarkt-Betreiber als auch Kunden) und die Robustheit im dauerhaften Betrieb im Einsatzfeld. Die Bewertung erfolgt daher auf Basis von systematischen Nutzerbefragungen und Datenerhebungen während autonomer Langzeittests.

In Kapitel 7 werden anschließend Konzepte und erste Ergebnisse weiterführender Arbeiten vorgestellt, die sich aktuell noch in Entwicklung befinden, jedoch zum Einsatz auf zukünftigen Versionen des Shopping-Roboters sowie für Service-Roboter in anderen Szenarien vorgesehen sind. Dies umfasst die Nutzung der monokularen 3D-Szenenrekonstruktion zur visuellen Hindernisdetektion sowie die kontinuierliche dynamische Adaption der globalen Umgebungskarte.

Kapitel 8 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf geplante und denkbare Erweiterungen der vorgestellten Algorithmen und Anwendungsszenarien.

Neben Navigationsleistungen für Serviceroboter, die zur Funktionalität des Shopping-Assistenten beitragen, wurden teilweise auch weitere methodische Ansätze entwickelt, bei denen zwar eine inhaltliche Verwandtschaft besteht, die jedoch im hier betrachteten Szenario nicht von Bedeutung sind. Um den eigentlichen Inhalt der Arbeit möglichst klar zu fokussieren und übersichtliche die wesentlichen Aspekte zu präsentieren, werden solche ergänzenden Ansätze in eigenen Abschnitten im Anhang beschrieben. Dies betrifft u.a. ein Odometrie-Korrektur-Verfahren für Synchro-Drive-Roboter (wie z.B. den B21) und eine spezielle Variante eines SLAM-Algorithmus für Landmarken-Detektion mittels einer omnidirektionalen Kamera. Ebenso wird ein neu entwickeltes Verfahren zur visuellen monokularen Szenenrekonstruktion im Anhang vorgestellt,

welches in dieser Arbeit lediglich als „virtueller Sensor“ für die Kartierung und Hindernisdetektion genutzt wird.

Kapitel 2

Kartenmodell

Ein Umgebungsmodell ist eine grundlegende Voraussetzung für viele autonome Leistungen des Roboters. Die Anfahrt spezifischer Punkte innerhalb der Umgebung (wie sie zum Beispiel die Lotsenfunktion des Einkaufsassistenten erfordert) setzt voraus, dass ein (möglichst optimaler) Pfad von der aktuellen Position des Roboters zur Zielposition bestimmt werden kann. Dies erfordert Wissen sowohl über die Struktur der Umgebung als auch über die eigene Position in Relation zur Zielposition und den Objekten der Umgebung. Dieses Wissen wird typischerweise in Form einer Karte gespeichert, die ebenfalls zur Selbstlokalisation genutzt werden kann (siehe Abschnitt 5.1). Die Begriffe Karte und Umgebungsmodell werden in dieser Arbeit teilweise synonym verwendet, wobei eine „Karte“ meist zur Bezeichnung einer konkreten Instanz eines Umgebungsmodells dient.

Das Umgebungswissen kann unter Umständen vom Designer des Robotersystems bzw. der Anwendung vorgegeben sein, z.B. indem bei bekannten Umgebungen ein Grundriss hinterlegt wird. Dieses Vorgehen garantiert jedoch einerseits nicht, dass die Karte mit den aktuellen Gegebenheiten übereinstimmt, falls beispielsweise nachträgliche Änderungen vorgenommen wurden oder keine Daten mit entsprechender Auflösung zur Verfügung stehen. Außerdem ist es häufig wünschenswert, dass die Karte die tatsächlich vom Roboter mit seiner Sensorausstattung wahrnehmbare Struktur wiedergibt, um diese z.B. für eine Selbstlokalisation verwenden zu können. Aus diesen Gründen sollte die Karte direkt aus den Sensorbeobachtungen des Roboters erstellt werden. Der Roboter erwirbt also selbst durch Beobachtungen das Wissen über seine Umgebung und generiert daraus eine Karte. Es ist auch möglich,

eine Kombination von Designerwissen und Beobachtungen zu nutzen, indem eine Grobstruktur vorgegeben wird und diese durch Beobachtungen verfeinert/aktualisiert wird. Außerdem fließt implizit fast immer Vorwissen über die Struktur der Umgebung und die Anforderungen der Anwendung in die Wahl der spezifischen Umgebungsrepräsentation sowie der sensorischen Ausstattung ein.

Umgekehrt kann die Umgebungskarte als abstrakte Repräsentation der Roboterbeobachtungen interpretiert werden: eine Folge von Beobachtungen wird in ein Modell mit höherem Abtraktionsgrad transformiert. Die entstehende Karte kann durch Verknüpfung und Schlussfolgerung aus den Beobachtungen mehr und genauere Informationen enthalten als die Summe der Einzelbeobachtungen. Als Beispiel sei die Erstellung einer zweidimensionalen Karte aus Entfernungsmessungen genannt: Durch eine räumlich-zeitliche Sequenz von Beobachtungen (Längenmessungen von unterschiedlichen Positionen) entsteht eine lokale Karte. Durch Integration mehrerer Einzelmessungen wird Beobachtungsrauschen ausgefiltert, außerdem kann die entstehende Repräsentation eine höhere räumliche Auflösung als die einzelnen Beobachtungen aufweisen. Weiterhin ist es möglich (in gewissem Rahmen), Vorhersagen für Messungen an Positionen zu treffen, von denen aus zuvor noch keine Beobachtung erfolgt ist. Im Kapitel 4 wird im dort vorgestellten „Map-Match-SLAM“-Ansatz eine lokale Karte in der genannten Form als Repräsentation einer zeitlich und räumlich begrenzten Sequenz von Beobachtungen verwendet.

2.1 Umgebungsmodelle

Zur Repräsentation der Umgebungsstruktur stehen verschiedene Modelle zur Auswahl. Die am häufigsten verwendeten Modelle sind geometrische Karten und Grid- bzw. Regionenkarten sowie topologische Karten. Die Modellarten weisen unterschiedliche Komplexität und daraus resultierend jeweils spezifische Vor- und Nachteile auf. Je nach geplanter Anwendung der Karte kann eine zweidimensionale oder dreidimensionale räumliche Abbildung der Umgebung genutzt werden. Eine weitere Dimension kommt hinzu, wenn zeitliche Veränderungen der Umgebung ebenfalls modelliert werden sollen.

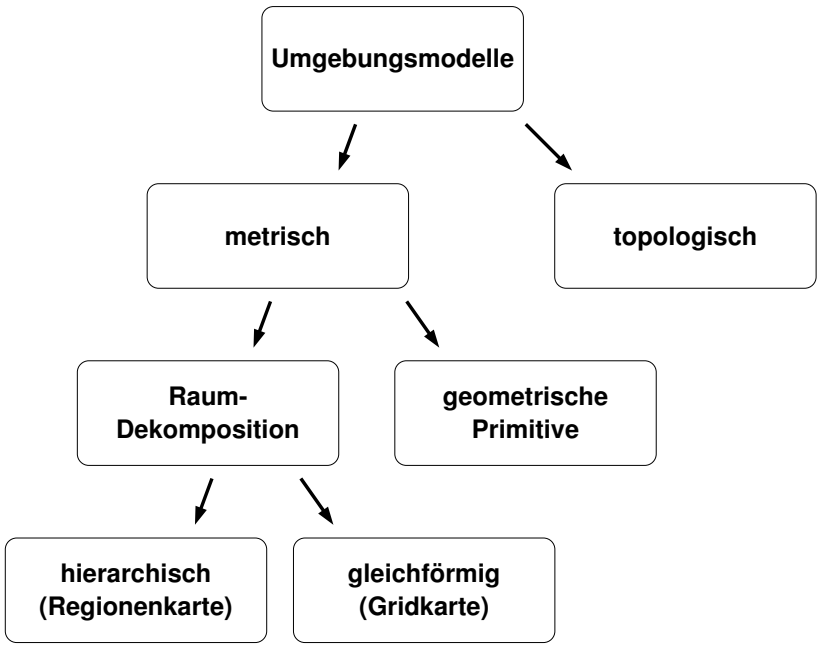

Abbildung 2.1: *Eine Taxonomie der relevanten Umgebungsmodelle für die Navigation eines mobilen Roboters.*

Weiterhin muss zwischen egozentrischer (Beobachter als Referenzsystem) oder allozentrischer (globales Referenzsystem) Beschreibung unterschieden werden [Klatzky, 1998]. Die direkten Sensor-Beobachtungen eines Roboters sind normalerweise inhärent egozentrisch, sie können daher ohne weiteres in eine egozentrische Karte transformiert werden, wobei eines der zuvor genannten Beschreibungsmodelle genutzt wird. Eine egozentrische Karte erfordert jedoch zusätzlichen Aufwand, wenn für einen bewegten Roboter unterschiedliche Beobachtungen von jeweils verschiedenen Beobachtungspositionen fusioniert werden sollen. In diesem Fall müssen alle Objekte/Merkmale in der Karte ständig angepasst werden, um die Eigenbewegung des Beobachters auszugleichen. Für eine allozentrische Karte muss zwar bei jeder Beobachtung eine Transformation von egozentrischen in globale Koordinaten durchgeführt werden (die Beobachterposition muss bekannt sein), dafür hat die Bewegung selbst zunächst keinen Einfluss auf das Umgebungsmodell. Für die Umgebungsmodellierung zum Zweck der Navigation werden daher fast immer die egozentrischen Beobachtungen in eine allozentrische Karte überführt.

2.1.1 Geometrische Karten

Geometrische Karten stellen die Umgebung als Ansammlung einfacher geometrischer Objekte wie z.B. Punkte [Harris und Stephens, 1988], [Shi und Tomasi, 1994], Linien [Nguyen et al., 2005], Ecken [Montemerlo et al., 2002], Kreissegmente, Flächen o.ä. dar. Jedes dieser Objekte wird dabei durch seine Position und eine Menge weiterer Parameter (Form, Größe, Lage im Raum, spezifische Eigenschaften - u.a. abhängig vom Sensor) beschrieben. Diese Objekte werden häufig auch als Features bezeichnet. Ebenfalls verwandt ist der Begriff Landmarke, wobei Landmarken meist komplexere Objekte sind und typischerweise zusätzlich die Forderung gestellt wird, dass diese eindeutig identifizierbar (wiedererkennbar) sind, was gerade bei primitiven Features oft nicht erfüllt ist.

Die in der geometrischen Karte verzeichneten Objekte und deren Parameter werden normalerweise nicht direkt vom Sensor gemessen, sondern in einem Merkmalsextraktions-Schritt aus den Sensorbeobachtungen ermittelt. Die Wahl der verwendeten Features ist dabei in der Regel eine Entscheidung des Designers, welche große Auswirkungen auf die Eigenschaften der entstehenden Karte hat: während simple Primitive mit wenig Aufwand im Sensorinput detektierbar sind, entsteht damit meist ein recht hoher Beschreibungsaufwand, da relativ viele Features gefunden werden. Gleichzeitig ist die Eindeutigkeit und damit Wiedererkennbarkeit der Features gering. Insgesamt wird auf diese Art ein geringes Abstraktionsniveau vom rohen Sensormesswert erreicht. Werden hingegen komplexere Objekte detektiert, so steigt zwar der Aufwand und gegebenenfalls die Fehleranfälligkeit bei der Detektion, andererseits werden weniger Objekte in der Karte gespeichert, so dass insgesamt möglicherweise weniger Parameter zur Beschreibung der Umgebung ausreichen. Komplexere Features sind außerdem besser eindeutig zu identifizieren. Mit wachsender Komplexität der erkannten Objekte kann ein steigender Abstraktionsgrad erreicht werden. Komplexe Features können u.a. auch aus einer Zusammenfassung mehrerer Low-Level-Features entstehen, zum Beispiel indem Gegenstände durch die Menge der auf ihnen gefundenen Punkt-Merkmale beschrieben werden [Lowe, 1999].

Problematisch ist, dass die Feature-Auswahl meist auf einen bestimmten Sensor zugeschnitten ist und die Features spezifisch für bestimmte Um-

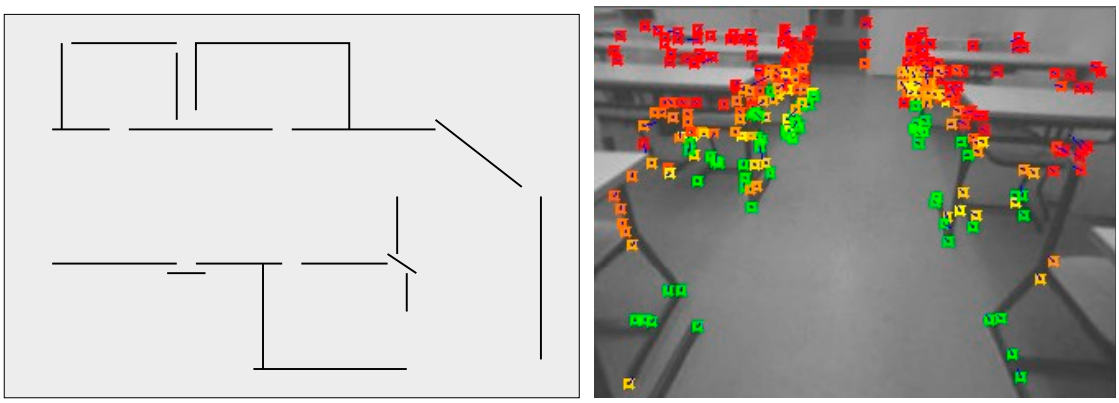

Abbildung 2.2: **Links:** *Prinzipskizze einer geometrischen Karte mit Liniensegmenten als Kartenobjekte.* **Rechts:** *Ausschnitt einer geometrischen Karte mit Punktfeatures. Diese wurden mittels eines Feature-Detektors im Bild bestimmt und ihre Position durch Tracking über mehrere Bilder bestimmt (siehe Abschnitt C.2).*

gebungen sind. Das Vorhandensein spezieller detektierbarer Features muss also a priori bekannt sein. Bei Wechsel in eine andere Umgebung oder Nutzung anderer Sensorik müssen jeweils neue Features gewählt und Detektionsalgorithmen für diese implementiert werden.

Geometrische Karten repräsentieren meist nur die Menge der detektierten Features, der Freiraum zwischen diesen wird jedoch nicht explizit modelliert. Dies ist insbesondere dann von Nachteil, wenn die beobachtbaren Features relativ spärlich auftreten, da dann insgesamt nur relativ wenig Information in der Karte verfügbar ist.

2.1.2 Gridkarten

Gridkarten beruhen auf der Aufteilung der gesamten zu beschreibenden Umgebung in einzelne Zellen [Moravec und Elfes, 1985]. Für jede Zelle wird ein Belegtheitswert gespeichert, welcher angibt, ob sich innerhalb dieser Zelle ein Objekt befindet oder ob diese Freiraum enthält. Die Belegtheit wird meist als Wahrscheinlichkeitswert repräsentiert und durch Verfahren, welche auch unsichere Informationen berücksichtigen können, berechnet (Bayes-Update [Moravec, 1988], Fuzzy-Logik [Oriolo et al., 1997], etc.).

Da Gridkarten keine a-priori-Annahmen über die Struktur wahrnehmbarer Objekte treffen, sind sie sehr gut zur Repräsentation verschiedener Umgebungen als auch für die Nutzung unterschiedlicher Sensoren

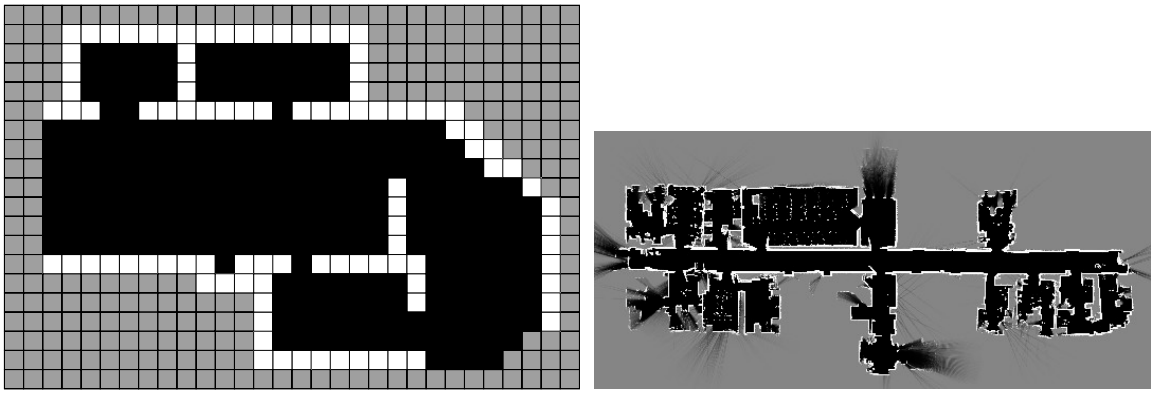

Abbildung 2.3: **Links:** *Prinzipskizze der Gridkarte für die Umgebung aus Abb. 2.2 und 2.4.* **Rechts:** *Eine (Teil-)Gridkarte des Bionik-Gebäudes, Sitz des Fachgebiets Neuroinformatik und Kognitive Robotik der TU Ilmenau. Weiß sind jeweils belegte Zellen dargestellt, schwarz freie.*

geeignet. Ebenso sind sie in der Lage, die verschiedenen Unsicherheits-Profile einzelner Sensoren zu berücksichtigen. Sie eignen sich daher auch sehr gut zur Sensorfusion. Nachteilig wirkt sich hingegen der unter Umständen sehr hohe Speicherbedarf aus. Die Auflösung der Karte wird durch die Zellgröße bestimmt, allerdings bedeutet eine höhere Auflösung (kleinere Zellen) auch eine größere Gesamtzahl an Zellen für die selbe Fläche und damit höhere Speicherkosten. In der überwiegenden Anzahl der Anwendungen werden zweidimensionale Grids genutzt, da diese für die meisten darauf aufbauenden Verfahren ausreichen und der benötigte Speicherplatz bei Hinzunahme der dritten Dimension (Auflösung entlang der Höhe) nochmals deutlich ansteigt. Es existieren allerdings auch Ansätze mit dreidimensionalen Grids [Moravec, 1996], [SeeGrid, 2008].

Da Gridkarten sowohl Hindernisse als auch Freiraum repräsentieren, eignen sie sich sehr gut für Navigationsaufgaben wie Pfadplanung und Kollisionsvermeidung, wobei auch hier die Rechenzeit mit der Auflösung wachsen kann.

2.1.3 Regionenkarten

Für größere zu modellierende Flächen kann es sinnvoll sein, statt einer festen Zellaufteilung eine adaptive Unterteilung des Raumes zu nutzen, welche Bereiche gleicher Belegtheit zu Regionen zusammenfasst. Eine solche adaptive Aufteilung ist z.B. mittels Quadtrees möglich [Kraetzschmar et al., 2000]. Die Repräsentation ist (bei geeigneter Umgebungs-

struktur) äußerst kompakt gegenüber einer Gridkarte mit fester Aufteilung, so dass der Speicherverbrauch deutlich geringer ist. Allerdings wird dazu zusätzlicher Rechenaufwand benötigt, insbesondere wenn die Regionen nicht a priori festgelegt sind, sondern mit jeder Beobachtung angepasst werden. Häufig werden solche Regionenkarten daher für statische Umgebungsmodelle benutzt und aus einer zuvor aus den Beobachtungen erstellten Gridkarte erzeugt.

2.1.4 Topologische Karten

Topologische Karten repräsentieren die Welt als eine Menge voneinander klar abgegrenzter Orte (auch als Plätze/Knoten bezeichnet) und die Verbindungen zwischen diesen [Kuipers, 1978], [Chatila und Laumond, 1985], [Kuipers und Byun, 1991]. Sie werden meist in Form gerichteter oder ungerichteter Graphen dargestellt. Topologische Modelle sind grundsätzlich unabhängig von geometrischer Information, allerdings kann den einzelnen Orten auch eine Position in einem Koordinatensystem zugeordnet werden [Gross et al., 2002b]

Topologische Modelle können bei geeigneter Wahl der Plätze Umgebungen sehr effizient repräsentieren, da meist implizit eine adaptive räumliche Auflösung genutzt wird. So ist es beispielsweise möglich, einen langen gleichförmigen Flur als einen Ort darzustellen, während viele kleinere angrenzende Räume jeweils einen eigenen Ort bilden [Buschka und Saffiotti, 2002]. Auf diese Weise kann eine hohe Abstraktion erreicht werden, feine Details wie die exakte Position und Form von Wänden und Hindernissen werden dafür nicht modelliert.

Die Navigation in einem topologischen Modell erfordert meist eine (eindeutige) Wiedererkennung des Ortes, d.h. dieser muss durch entsprechende Eigenschaften gekennzeichnet sein [Ulrich und Nourbakhsh, 2000] , [Torralba et al., 2003]. Auch hier muss eine Menge von Eigenschaften definiert werden, welche zur Identifikation des Ortes innerhalb der Karte und zur Abgrenzung zwischen verschiedenen Orten dienen. Es existieren allerdings auch Ansätze, solche Merkmale zu lernen [Kuipers und Beeson, 2002]. Ebenso wie bei geometrischen Modellen besteht die Schwierigkeit, dass Merkmale nicht ohne weiteres zwischen verschiedenen Umgebungen übertragbar sind (z.B. die Einteilung in einzelne Räume anhand von Tür-Durchfahrten, welche in Indoor-Anwendungen sinnvoll erscheint, scheitert in Outdoor-Umgebungen).

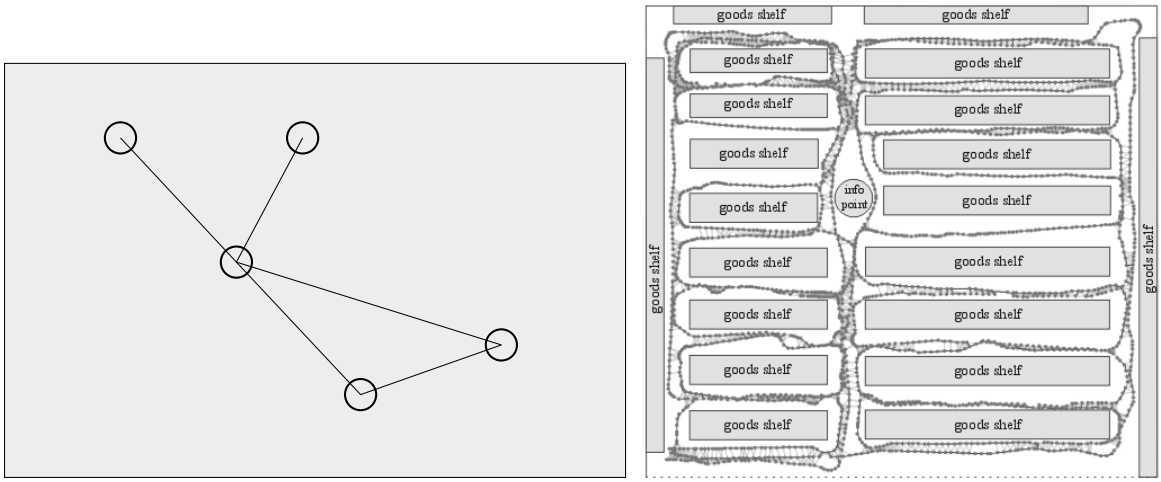

Abbildung 2.4: **Links:** *Prinzipskizze einer topologischen Karte (selbe Umgebung wie Abb. 2.2).* **Rechts:** *Eine vereinfachte topologische Karte eines Baumarktareals: die Knoten sind als Grundlage der Selbstlokalisation mit visuellen Ansichten attributiert.*

Um die jeweiligen Vorteile beider Repräsentationen zu nutzen, können topologische und Gridkarten auf verschiedene Weise miteinander kombiniert werden. Einzelne Teil-Gridkarten können z.B. durch ihre Topologie (aber ohne globale metrische Information) verknüpft werden [Zimmer, 2000]. Andere Ansätze bestimmen aus einer Gridkarte zusätzlich eine globale Topologie [Thrun und Buecken, 1996].

2.1.5 Semantische Karten

Durch Verknüpfung der Raumbeschreibung mit zusätzlichen semantischen Informationen (z.B. Namen/Bedeutungen von Orten oder Objekten) kann die Karte zu einer semantischen Karte erweitert werden. Dies ist prinzipiell nicht auf ein bestimmtes Modell beschränkt, Regionenkarten und topologische Karten sind jedoch aufgrund der inhärenten Segmentierung von Objekten oder Plätzen besonders gut dafür geeignet. Durch die Einbeziehung zusätzlicher Bedeutungsinformation wird die Karte über die reine Raumbeschreibung hinaus erweitert und neue Nutzungsmöglichkeiten eröffnet, z.B. intuitiver Bezug zu bestimmten Orten („Gehe in die Küche!“). Die Attributierung kann manuell erfolgen oder vom Roboter mittels gelernter oder vorgegebener Merkmals-Zuordnung automatisch vorgenommen werden [Rottmann et al., 2005].

Semantische Karten werden hier nur der Vollständigkeit wegen erwähnt, sie sind in dieser Arbeit nicht von Bedeutung.

2.2 Kartenmodelle in dieser Arbeit

Aufgrund der universellen Verwendbarkeit kommen bei den in dieser Arbeit vorgestellten Verfahren hauptsächlich Gridkarten zum Einsatz. Sowohl für die globale Kartierung der Einsatzumgebung als auch als Modell der unmittelbaren lokalen Umgebung zur Kollisionsvermeidung werden solche Karten eingesetzt. Es kommen dabei ausschließlich 2D-Grids zur Anwendung, d.h. alle erkannten Hindernisse werden auf die Bodenebene projiziert. Da die Karten jeweils von einem mobilen Roboter während der Bewegung erstellt und benutzt werden sollen, werden ausschließlich allozentrische Modelle benutzt. Die egozentrischen Sensor-Beobachtungen werden dazu jeweils unter Nutzung der Odometrie (Abschnitt 3.2) und Selbstlokalisation (Abschnitt 5.1) in das globale Koordinatensystem überführt.

Für die globale Kartierung wird ein SLAM-Verfahren benutzt (Kapitel 4), welches auf einem Rao-Blackwellized Particle Filter (RBPF, [Murphy, 1999]) beruht. Die Grundlage dieses Verfahrens bildet ein Partikelfilter, bei dem jedes Partikel eine eigene Gridkarte erzeugt. Im Gegensatz zu anderen RBPF-SLAM-Verfahren ([Montemerlo et al., 2002], [Hähnel et al., 2003], [Eliazar und Parr, 2003]) wird im Bewertungsschritt nicht eine einzelne Beobachtung genutzt, sondern es wird zusätzlich eine *lokale* Gridkarte erstellt, der Vergleich der lokalen und globalen Karte (Map Matching) wird zur Gewichtung der Partikel genutzt. Dieses Vorgehen hat den Vorteil, dass das SLAM-Framework leicht an verschiedene Sensoren angepasst werden kann. Dies wird durch die Nutzung des selben Frameworks sowohl für SLAM mit Sonar-Sensoren (Abschnitt 4.3) als auch für SLAM basierend auf verschiedenen visuellen Verfahren zur lokalen Szenenrekonstruktion (Abschnitt 4.4) demonstriert. Am Beispiel der Sonar-Sensoren wird auch gezeigt, dass durch die Nutzung der lokalen Karte als Zwischenrepräsentation für Sensoren mit hoher Messvarianz eine bessere Bewertung der Beobachtungen möglich ist.

Die lokale Navigation basiert auf einer Gridkarte der unmittelbaren Roboterumgebung, welche jeweils ein Gebiet mit festem Radius um die ak-

tuelle Position des Roboters abdeckt. Obwohl der beschriebene Bereich durch die aktuelle Roboterposition bestimmt wird, handelt es sich um eine allozentrische Beschreibung, die Karte ist jeweils in einem globalen Bezugssystem verankert. Bei Bewegung des Roboters werden lediglich die Grenzen des dargestellten Ausschnitts verschoben. Durch die Nutzung der Gridkarte lassen sich leicht beliebig viele unterschiedliche Sensoren zur Hindernisdetektion kombinieren. Um der Tatsache Rechnung zu tragen, dass jeder Sensor unterschiedliche Erfassungsbereiche haben kann und dadurch bestimmte Hindernisse unter Umständen nicht von allen Sensoren wahrgenommen werden können, wird zunächst für jeden Sensor eine eigene lokale Gridkarte berechnet, wobei all diese Karten eine einheitliche Zellauflösung besitzen. Die Einzelkarten werden dann jeweils bei einer neuen Beobachtung eines Sensors fusioniert und die entstehende Gesamtkarte den Navigationsmodulen (Abschnitt 5.2) zur Verfügung gestellt.

Weiterhin werden auch topologische Karten eingesetzt. Beim assistierten SLAM (Abschnitt 4.5) beschreibt eine einfache topologische Karte die Struktur der Umgebung zusätzlich mit höherem Abstraktionsgrad als die Gridkarte, so dass sie zur Steuerung der Erkundung genutzt werden kann. Für die Selbstlokalisation wird ebenfalls neben der Gridkarte eine topologische Karte verwendet, deren Knoten neben der Position mit den aufgenommenen Bildern einer Panorama-Kamera attributiert sind und so eine visuelle Selbstlokalisation erlauben [Gross et al., 2002b].

Eine geometrische Karte wird in einem monokularen visuellen Szenenrekonstruktions-Verfahren (Abschnitt C.2) genutzt, welches für die im Kamerabild detektierten Punktfeatures durch Tracking über eine Bildsequenz deren 3D-Position ermittelt [Einhorn et al., 2007a]. Diese Karte wird sowohl für SLAM als auch zur Hindernisdetektion eingesetzt, allerdings wird sie dazu jeweils in eine Gridkarte umgerechnet.

Kapitel 3

Grundlagen des Kartenaufbaus

Nachdem im vorhergehenden Kapitel die Wahl einer Gridkarte als zentrale Beobachtungs- und Umgebungsrepräsentation motiviert wurde, soll nun der Aufbau einer solchen Karte aus den Messdaten der Sensoren beschrieben werden. Dazu wird zunächst erklärt, wie aus Einzelmessungen Schätzungen für die Belegtheiten der Gridzellen gewonnen werden. Dies wird am Beispiel von Entfernungssensoren und von 3D-Punktkoordinaten erläutert. Zum Aufbau einer globalen Karte werden viele Messungen von verschiedenen Beobachtungs-Positionen integriert. Es ist daher notwendig, die jeweilige Position, an der der Roboter die Messung aufgenommen hat, zu kennen. Die Positionsbestimmung kann also als eine Voraussetzung zum Kartenbau betrachtet werden. In diesem Kapitel wird zunächst die Odometrie des Roboters als Grundlage der Positionsbestimmung genutzt. Weil diese fehlerbehaftet ist, werden Methoden zur Verbesserung und Korrektur der Odometrie-Messungen untersucht.

Da jedoch die Korrektur der Odometrie die Positionsfehler zwar reduzieren, aber nicht beschränken oder eliminieren kann, ist die robuste und konsistente Kartierung ohne eine geschlossene Betrachtung von Kartenaufbau und Positionsbestimmung nicht möglich. Aufbauend auf dieser Erkenntnis wird im nächsten Kapitel ein „Simultaneous Localization and Mapping“ (SLAM)-Verfahren entwickelt, das auf der Nutzung von Gridkarten als Repräsentation der Umgebungsbeobachtungen basiert.

3.1 Gridkarten-Berechnung aus Beobachtungen

In dieser Arbeit werden im Wesentlichen zwei Arten von Umgebungsbeobachtungen betrachtet: Entfernungsmessungen zwischen einem Sensor und einem Objekt, und direkte Beobachtung der 3D-Position von Objekten oder Punkten in der Umgebung. Diese werden wiederum häufig aus der Verarbeitung von Kamera-Bildern gewonnen. In diesem Abschnitt wird jeweils vorgestellt, wie aus solchen Beobachtungen eine Gridkarte erstellt werden kann, indem die Belegtheitswahrscheinlichkeit der Einzelzellen geschätzt wird. Außerdem wird die Verrechnung einzelner Schätzungen beschrieben, so dass aus mehreren Beobachtungen eine wahrscheinlichkeitstheoretisch exakte Gesamt-Belegtheitsschätzung für jede einzelne Zelle berechnet werden kann. Weiterhin wird die Fusion mehrerer Karten betrachtet, welche aus Messungen unterschiedlicher Sensoren entstanden sind.

3.1.1 Zellen-Belegtheits-Schätzung mittels Entfernungs-Sensoren

Charakterisierung von Entfernungs-Sensoren

Entfernungsmessende Sensoren sind in der Robotik weit verbreitet und stellen neben Kameras die wichtigste Form der Sensorik zur Umgebungsbeobachtung dar. Die Messungen dieser Sensoren beruhen meist auf dem „Time-of-Flight“-Prinzip: Ein Signal wird von einem Sender ausgestrahlt, an einem Objekt im Strahlweg reflektiert und von einem Empfänger wird das zurückkehrende Signal registriert. Unter Kenntnis der Ausbreitungsgeschwindigkeit kann die Zeit zwischen Senden und Empfangen zur Berechnung der Entfernung des reflektierenden Objektes benutzt werden. Alternativ kann auch (insbesondere bei sich schnell ausbreitenden elektromagnetischen Signalen) die Phasenlage zwischen gesendeter und empfangener Welle ausgewertet werden.

Konkrete Implementierungen solcher Entfernungs-Sensoren existieren zum Beispiel als Ultraschall (Sonar)-, Radar-, Infrarot- oder Laser-Entfernungsmesser. Je nach genutztem Medium hat der Sensor unterschiedliche Eigenschaften. Die erreichbare Winkelauflösung hängt von

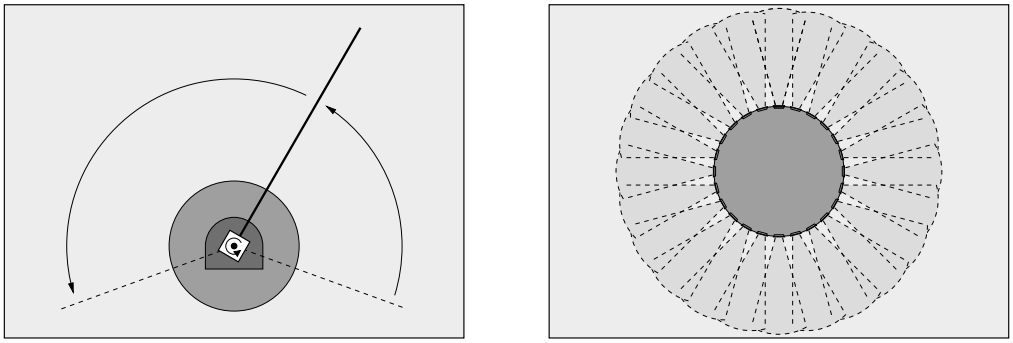

Abbildung 3.1: *Prinzipskizzen zum Aufbau von Entfernungs-Sensoren.* **Links:** *Beim Laser-Range-Scanner wird eine Sende-Empfangs-Einheit gedreht und misst in regelmäßigen Abständen (unterschiedliche Richtungen) den Abstand zum nächsten Objekt.* **Rechts:** *Ein Sonar-Sensor-Array besteht aus vielen gleichartigen Sensoren, welche hier im Beispiel äquidistant an der Außenhülle des Roboters angebracht sind.*

der Fokussierbarkeit des abgesandten Energie-Impulses ab, außerdem beeinflussen unterschiedliche Reflektions- und auch Transparenzeigenschaften verschiedener Materialien das Messergebnis.

Einzelne Sensoren werden manchmal beweglich gelagert, so dass sie Messungen in verschiedene Richtungen vornehmen können und somit hohe Richtungsauflösung mit einem großen Erfassungsbereich kombinieren. Dies ist insbesondere bei Laser-Sensoren üblich, wo mit einem einzelnen Strahl zeitlich aufeinander folgend Messungen über einen großen Winkelbereich erfolgen. Dies schlägt sich auch in der Bezeichnung Laser-Range-*Scanner* nieder.

Eine Alternative dazu ist der Aufbau eines Array aus mehrerer Sensoren, diese Art der Anordnung kommt häufig bei Sonarsensoren zum Einsatz. Ebenso wie das Scanner-Prinzip ist das Ziel die Abdeckung eines größeren Beobachtungsbereiches, ohne die Richtungsauflösung zu verringern. Es werden dabei natürlich mehr Sender- und Empfänger-Einheiten benötigt, andererseits entfällt die unter Umständen sehr aufwändige Mechanik inklusive Ansteuerung/Auswertung für einen beweglichen Sensor. Ein großer Vorteil eines Arrays ist, dass prinzipiell alle Sensoren gleichzeitig arbeiten können, während bei einem Scanner eine bestimmte Zeit nötig ist, um alle Richtungen im Erfassungsbereich einmal zu „sehen“. Statt einer strikten Kopplung je eines Senders und eines Empfängers ist es dann auch möglich, das Signal eines Senders mit mehreren Empfängern aufzufangen und umgekehrt. Die Menge der verwertbaren Information kann dadurch beträchtlich erhöht werden.

Es muss allerdings beachtet werden, dass es je nach Anordnung und Auswertung in diesem Fall auch zu unerwünschtem Crosstalk zwischen den einzelnen Sensoren des Arrays kommen kann, wenn ein Empfänger ein Signal eines „falschen“ Senders empfängt oder das Signal falsch zugeordnet wird. Crosstalk kann durch zeitlich versetzte Ansteuerung benachbarter Sensoren und/oder durch eine intelligente Auswertung der empfangenen Signale verringert bis eliminiert werden. In dieser Arbeit wird Crosstalk nicht im Detail betrachtet, es stellt lediglich eine mögliche Rauschquelle bei der Betrachtung der Zuverlässigkeit von Entfernungsmessungen dar.

Unabhängig vom Mess- und Anordnungsprinzip wird eine Messung eines Entfernungsmessers oder eines entsprechenden Arrays (eine Beobachtung) hier als eine Menge von Entfernungen mit den zugehörigen Messrichtungen und -öffnungswinkeln betrachtet. Es wird dabei idealisierend angenommen, dass alle diese Entfernungen simultan gemessen wurden, auch wenn dies in der realen Messanordnung nicht immer der Fall ist. Insbesondere bei Messungen während der Roboterbewegung kann dies zu erhöhten Ungenauigkeiten führen, auch dieser Effekt wird hier aber lediglich als Teil des Messrauschens betrachtet.

Für viele Entfernungssensoren gilt, dass die Zuverlässigkeit und Genauigkeit der Messung mit zunehmender Entfernung des Objektes abnimmt. Dieser Eigenschaft beruht auf der Abschwächung des Signals im Medium (Luft) und daraus folgenden Schwierigkeiten beim Empfang des reflektierten Anteils. Damit ist auch offensichtlich, dass der Abfall der Zuverlässigkeit bei unterschiedlichen Sensoren verschieden schnell ist: Während bei den hier verwendeten Sonar-Sensoren eine zuverlässige Messung jenseits von 3m Entfernung kaum noch möglich ist, kann ein Laser-Scanner auch auf 50m noch sehr genaue Ergebnisse liefern. Die entfernungsabhängige Ungenauigkeit wird hier explizit durch eine Exponentialfunktion modelliert, die für jeden Sensor spezifisch parametriert und bei der Schätzung der Gridzellen-Belegung berücksichtigt wird (siehe nächsten Unterabschnitt).

Zellen-Belegtheits-Schätzung aus einer Entfernungsmessung

Die Entfernungsmessungen sollen nun in Belegtheitswerte einer Gridkarte umgerechnet werden. Die hier vorgestellte Methode [Thrun et al., 2005] ist relativ einfach gehalten, liefert aber ausreichend gute Ergeb-

nisse und lässt sich sehr effizient implementieren. Der Algorithmus geht davon aus, dass der Erwartungswert der Messung dem tatsächlichen Abstand eines Objektes entspricht, d.h. dass der Sensor keinen systematischen Fehler bei der Entfernungsmessung macht. Falls ein fester, invarianter Offset bekannt wäre, könnte dieser leicht vor der Verarbeitung der Messwerte korrigiert werden.

Die Gridkarte enthält für jede Zelle eine Schätzung, ob diese ein Objekt enthält (also belegt ist). Ein Wert von 1 bedeutet dabei, dass die Zelle mit absoluter Sicherheit belegt ist. Ein Wert von 0 steht für absolut frei. 0.5 bedeutet, dass keine Information über die Zelle vorhanden ist, sie ist mit gleicher Wahrscheinlichkeit frei oder belegt. In der Praxis wird es meist nicht möglich sein, absolute Gewissheit über den Zustand einer Zelle zu erlangen. Es ist daher normalerweise gar nicht gewünscht, dass die Werte 0 und 1 tatsächlich erreicht werden.

Eine Entfernungsmessung der Länge d_M belegt, dass sich im Erfassungsbereich des Sensors ein Objekt in Entfernung d_M befindet, und dass sich kein Objekt näher als d_M am Roboter befindet. Das lässt darauf schließen, dass alle Zellen, welche eine Entfernung $d_{cell} < d_M - \epsilon$ zum Roboter aufweisen, frei sind. Hingegen muss sich in einer oder mehreren Zellen mit der Entfernung $d_{cell} = d_M \pm \epsilon$ das beobachtete Objekt befinden. Da sich aus einer einzelnen Messung keine genauere Information gewinnen lässt, in welcher (bzw. welchen) dieser Zellen das Objekt tatsächlich liegt, wird für alle diese Zellen eine erhöhte Belegtheitswahrscheinlichkeit geschätzt. Für Zellen mit $d_{cell} > d_M + \epsilon$ lässt sich keine Aussage treffen, diese werden durch das reflektierende Objekt verdeckt. Die Nutzung der Unschärfe-Konstante ϵ ist notwendig, da bei der Einteilung der Grundfläche in Zellen nur diskrete Zellpositionen möglich sind, so dass eine exakte Übereinstimmung niemals auftreten wird. Bei Reduktion der Zelle auf ihre Mittelposition erscheint es sinnvoll, ϵ in der Größenordnung einer halben Zelldiagonale zu wählen.

Um der wachsenden Unsicherheit mit größerer Entfernung Rechnung zu tragen, werden die Belegtheits-Schätzwerte *occ* nachträglich mit einer Unschärfefunktion behandelt:

$$occ' = 0.5 + (occ - 0.5) \cdot e^{-\frac{d_{cell}^2}{\sigma^2}} \tag{3.1}$$

Durch diese Funktion werden weiter entfernte Zellen in ihrer Schätzung näher an den Wert 0.5 gezogen, d.h. das Wissen über deren Belegung ist

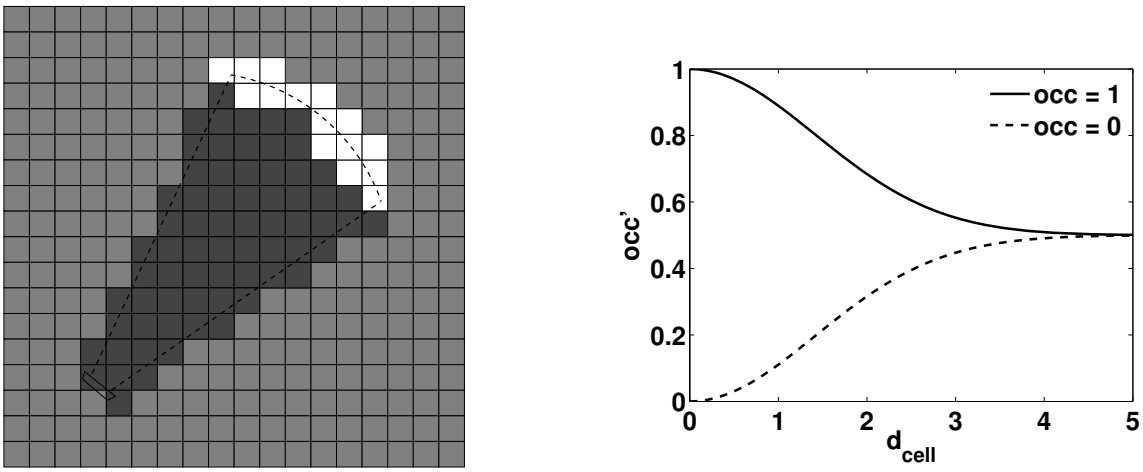

Abbildung 3.2: **Links:** *Schätzung der Zellen-Belegtheit aus den Entfernungs-Messwerten: weiß = belegt, schwarz = frei, grau = unbekannt,* **Rechts:** *Abschwächung der Belegtheitsschätzung mit zunehmender Zellentfernung (Gl. 3.1, $\sigma = 2.0$)*

weniger sicher. Der Reichweitenparameter σ ist dabei Sensor-spezifisch und gibt die Zuverlässigkeit an: je höher Sigma, umso weiter kann der Sensor messen. Über eine Sequenz von Messungen betrachtet bedeutet dies, dass bei größerer Entfernung und kleinerem σ mehr Beobachtungen einer Zelle nötig sind, um eine aussagekräftige Schätzung der Belegtheit zu erhalten.

Eine genauere Schätzung als mit dem hier vorgestellten Verfahren wäre durch Integration der Wahrscheinlichkeitsdichte über die Fläche jeder Zelle möglich, dies wäre jedoch deutlich aufwändiger und führt insbesondere bei Integration vieler Beobachtungen nur zu geringfügig anderen Ergebnissen.

Die Schätzung aller Zellbelegungen wird für jeden Entfernungsmesswert einer Beobachtung unabhängig durchgeführt, je nach Anordnung der Einzelsensoren kann es geschehen, dass eine Zelle dabei mehrfach beobachtet wird, damit existieren mehrere unabhängige Belegtheitsschätzungen, die wie verschiedene Messungen integriert werden (siehe Abschnitt 3.1.2). Es kann außerdem angenommen werden, dass alle Zellen, die innerhalb der vom Roboter selbst belegten Fläche liegen, frei von Hindernissen sind.

Zeiteffiziente Implementierung mittels Look-Up-Tables

Bei der im vorigen Abschnitt beschriebenen Methode muss für *jede* Zelle (in einem bestimmten Bereich um den Roboter) zunächst geprüft werden, ob sie im Erfassungsbereich eines bestimmten Sensors liegt, anschließend kann durch Vergleich der Zellentfernung mit der Messweite die Zellbelegung bestimmt werden. Dieser Aufwand erscheint zunächst nicht sehr hoch, die benötigte Zeit für die Berechnung der relativen Winkel zwischen Zellen und Sensor und den Vergleich mit dem Sensorkegel fällt jedoch insbesondere dann stark ins Gewicht, wenn wie in Kapitel 4 eine große Anzahl von Karten gleichzeitig aktualisiert werden sollen. Daher ist notwendig, das Kartenupdate - bei annähernd gleicher Genauigkeit - zu beschleunigen.

Anstatt die betroffenen Zellen bei jeder Messung neu zu bestimmen, wird dazu für jeden Sensor eine Liste der zu aktualisierenden Zellen vorberechnet und gespeichert. Ein Sensor-Array oder Scanner wird zu diesem Zweck, wie bereits beschrieben, als Menge von Einzelsensoren betrachtet, wobei jedem Sensor ein Beobachtungskegel zugeordnet ist (auch Laserstrahlen werden als extrem enge Kegel betrachtet). Ein abstrakter Sensor wird damit durch die Definition der Positionen und Blickwinkel (im Roboter-Bezugssystem, also relativ zur Roboterposition), der Kegelbreite aller Einzelmessungen sowie der maximal zu betrachtenden Messweite und der Zuverlässigkeitsfunktion bestimmt. Speziell für Laser-Scanner als auch die Sonar-Anordnung der Roboter SCITOS und PERSES gilt, dass die Einzelmessungen identische Kegelbreiten aufweisen und gleichmäßig auf einem Kreis verteilt sind.

Die eigentliche Look-Up-Table (LUT) ist zweistufig aufgebaut: In der ersten Tabelle wird für jede Einzelmessung abhängig von der Blickrichtung des Roboters ein Zellen-Offset des Sensorursprungs zur Roboterposition und eine Sensor-Blickrichtung abgelegt, beides in globalen Koordinaten. In einer zweiten Tabelle wird dann für jede globale Sensor-Blickrichtung eine *nach Entfernung sortierte* Liste aller vom Sensor erfassten Zellen gespeichert. Diese Liste enthält für jede Zelle den Offset gegenüber dem Sensorursprung, die Entfernung und die sich aus der Zuverlässigkeit ergebende Belegtheitsschätzung für die beiden Fälle $d_{cell} < d_M - \epsilon$ und $d_{cell} = d_M \pm \epsilon$. Bei einer Auflösung der LUT von 1° und 24 Sonarsensoren ergeben sich somit z.B. $360 * 24$ Listen

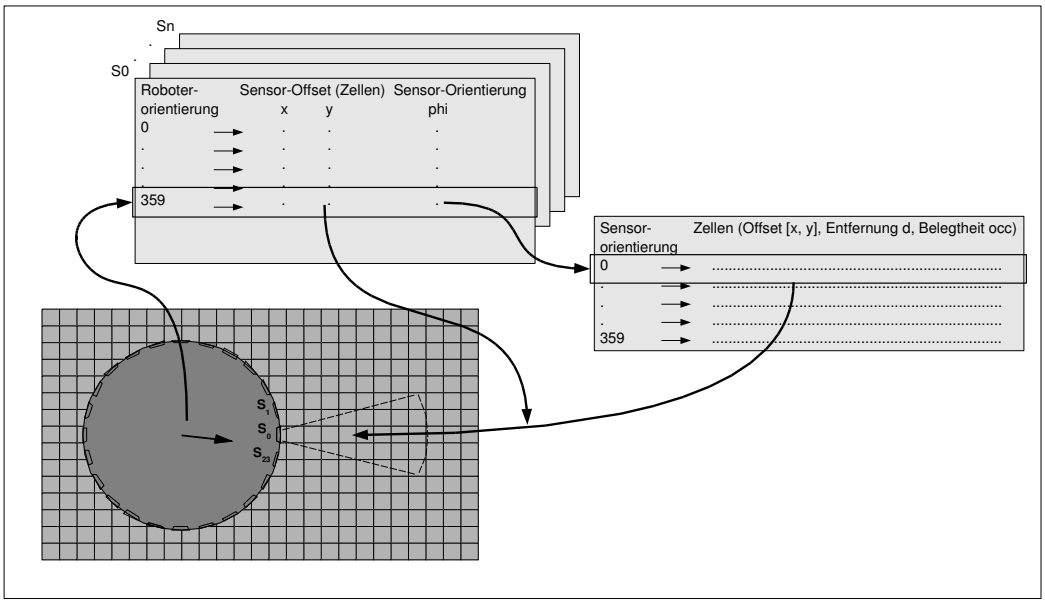

Abbildung 3.3: *Durch die Vorberechnung der von einer Sensormessung jeweils betroffenen Zellen und Speicherung in einer Look-Up-Table (LUT) kann die Belegtheitsschätzung erheblich beschleunigt werden. Die Abbildung veranschaulicht das zweistufige Prinzip der LUT.*

der ersten Stufe mit je 3 Einträgen sowie 360 Listen der zweiten Stufe mit einer variablen Anzahl an Zelleinträgen (Abb. 3.3).

Beim Kartenupdate wird für jeden Sensor aus Roboterpose und Tabelle 1 zunächst der Blickrichtungsindex bestimmt, mit diesem Index wird die richtige Liste in Tabelle 2 gefunden. Eine konkrete Zelle ergibt sich durch Summation der Roboter-Positions-Zelle, des Sensor-Offsets aus Tabelle 1 und des Zell-Offsets aus Tabelle 2. Da die Zellen in der Liste sortiert sind, kann abgebrochen werden, sobald eine Zelle mit Entfernung $d_{cell} > d_M + \epsilon$ erreicht wird.

Die Beschleunigung der Berechnung wird durch 2 Effekte erzielt: Durch die sortierte Speicherung werden tatsächlich nur jene Zellen betrachtet, die überhaupt von einem Sensor beobachtet werden und innerhalb der Messentfernung liegen. Neben dieser Einschränkung des Suchraumes werden bei Nutzung der LUT nur wenige Additionen und Vergleiche pro betroffener Zelle durchgeführt, anstatt bei jeder Beobachtung für jede Zelle Richtung und Entfernung zu jedem Sensor neu zu berechnen.

Gegenüber der ursprünglichen Implementation wurde damit ein Speedup um den Faktor 25 erreicht. Da für jeden Sensor und jede Gridkartenauflösung eine eigene LUT vorgehalten werden muss, wird das durch

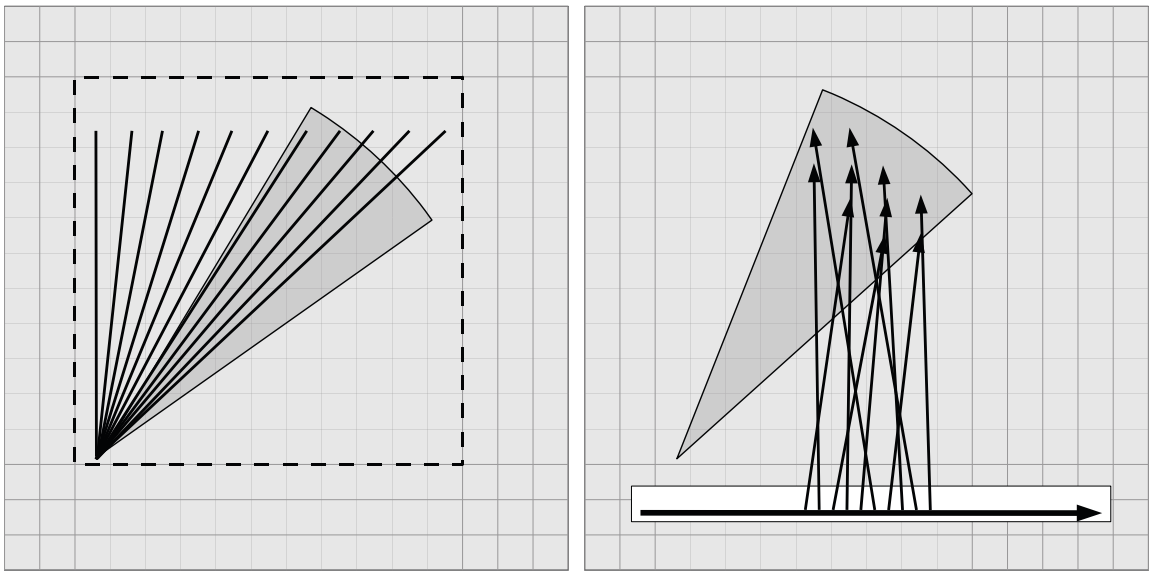

Abbildung 3.4: **Links:** *Bei „naiver“ Implementation muss für jede Zelle immer wieder die Richtung und Entfernung zum Sensor berechnet werden, um erst zu prüfen, ob sie im Erfassungsbereich des Sensors liegt und in diesem Fall anhand der gemessenen Entfernung einen Belegtheitswert zu schätzen. Die zu betrachtenden Zellen werden durch das umschriebene Rechteck bezeichnet. Es müssen dadurch zunächst viele Berechnungen auch für Zellen durchgeführt werden, die gar nicht vom Sensor erfasst werden. Dies ist im Bild für einige Zellen einer Zeile angedeutet.* **Rechts:** *Bei der Nutzung der Look-Up-Tabelle (LUT) wird die Berechnung der Richtung und Entfernung für jede Zelle nur einmal durchgeführt und anschließend bei jeder Sensormessung die gespeicherten Werte genutzt. Über die LUT kann genau auf die Zellen innerhalb des Sensor-Erfassungsbereiches in der Reihenfolge ihrer Entfernung zum Sensor zugegriffen werden. Dadurch ist nicht nur der Aufwand pro Zelle geringer, sondern es wird auch kein unnötiger Aufwand betrieben, um die Zellen auszusortieren, welche aufgrund ihrer Richtung oder Entfernung nicht von der Messung betroffen sind. Durch diese Einschränkung des Suchraumes wird im Durchschnitt eine Beschleunigung der Karten-Aktualisierung um den Faktor 25 erreicht.*

einen Speicherverbrauch von bis zu 100 MB (abhängig von Zellgröße und Reichweite der Sensoren) erkauft.

3.1.2 Verrechnung mehrerer unabhängiger Schätzungen

Im vorigen Abschnitt wurde für jede Zelle die Wahrscheinlichkeit $p(occ|o)$, dass die Zelle belegt ist, aus einer einzelnen Beobachtung o geschätzt.

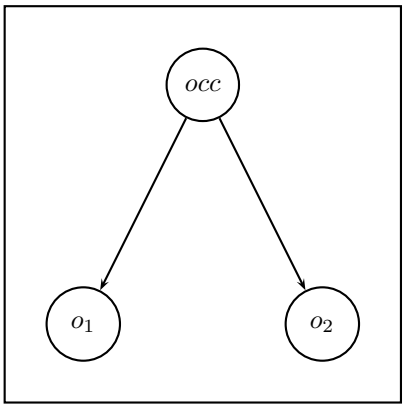

Abbildung 3.5: *Graphisches Modell: Die Belegtheitswahrscheinlichkeit der Zelle wird zwei mal unabhängig voneinander beobachtet.*

Im nächsten Schritt soll aus vielen *unabhängigen* Einzelbeobachtungen $o_1, \ldots, o_t$ eine Gesamt-Belegtheitsschätzung berechnet werden. Dies erfolgt nach der in [Moravec, 1988] vorgestellten Methode:

Gegeben sei für eine Zelle die Sequenz der Einzel-Belegtheitsschätzungen

$$p(occ|o_1), p(occ|o_2), \ldots, p(occ|o_t)$$

Gesucht ist die resultierende Gesamtwahrscheinlichkeit

$$p(occ|o_{1:n})$$

Für zunächst 2 Messungen o_1, o_2 und mit dem Gegenereignis $\overline{occ}$ (Wahrscheinlichkeit für Nicht-Belegtheit mit $p(\overline{occ}) = 1 - p(occ)$) gilt nach Bayes (Anhang B)

$$p(occ|o_2, o_1) = \frac{p(o_2|occ, o_1) \cdot p(occ|o_1)}{p(o_2|o_1)} \tag{3.2}$$

$$\frac{p(occ|o_2, o_1)}{p(\overline{occ}|o_2, o_1)} = \frac{p(o_2|occ, o_1)}{p(o_2|\overline{occ}, o_1)} \cdot \frac{p(occ|o_1)}{p(\overline{occ}|o_1)} \tag{3.3}$$

Durch Einsetzen der Unabhängigkeitsbedingung

$$p(o_2|occ, o_1) = p(o_2|occ) \tag{3.4}$$

und erneutes Anwenden der Bayes-Regel folgt letztendlich

$$\frac{p(occ|o_2, o_1)}{p(\overline{occ}|o_2, o_1)} = \frac{p(occ|o_2)}{p(\overline{occ}|o_2)} \cdot \frac{p(occ|o_1)}{p(\overline{occ}|o_1)} \cdot \frac{p(\overline{occ})}{p(occ)} \tag{3.5}$$

Dies lässt sich verallgemeinern zu der rekursiven Update-Formel:

$$\frac{p(occ|o_{1:t})}{p(\overline{occ}|o_{1:t})} = \frac{p(occ|o_t)}{p(\overline{occ}|o_t)} \cdot \frac{p(occ|o_{1:t-1})}{p(\overline{occ}|o_{1:t-1})} \cdot \frac{p(\overline{occ})}{p(occ)} \tag{3.6}$$

$p(occ)$ bezeichnet hier die a-priori-Schätzung der Belegtheit ohne Beobachtungen, die meist mit 0.5 festgelegt wird. Mit Hilfe der Gleichung 3.6 kann jeweils für eine Zelle aus der alten akkumulierten Belegtheitsschätzung und der neuen Beobachtung eine aktualisierte Gesamt-Belegtheitsschätzung berechnet werden. Abb. 3.6 zeigt, wie sich dadurch aus der Sequenz von Einzelbeobachtungen eine Karte ergibt.

3.1.3 Zellen-Belegtheits-Schätzung mittels 3D-Punktkoordinaten (visuell beobachtbaren Landmarken)

In diesem Abschnitt soll erläutert werden, wie aus einer Menge von 3D-Punkten eine Gridkarte der Umgebung geschätzt werden kann. Die 3D-Punkte stellen dabei die Koordinaten von Objekten oder bestimmten Objektpunkten im Raum dar. Mit der Transformation der 3D-Punktwolke in eine 2-dimensionale Karte geht eine Informationsreduktion einher, die in diesem Fall im Sinne einer einheitlichen Repräsentation und als Grundlage für die weitere Verarbeitung erwünscht ist.

3D-Koordinaten werden selten direkt beobachtet (möglich ist dies aber zum Beispiel mit einem 3D-Entfernungs-Scanner), sondern häufiger aus Kamera-Bildern gewonnen. Da ein einzelnes Kamerabild keine explizite Tiefeninformation enthält, müssen zur Rückgewinnung der Tiefe und damit der 3D-Koordinaten der aufgenommenen Pixel mehrere Bilder verrechnet werden. Es existieren zwar Verfahren zur monokularen Tiefenschätzung, z.B. mittels Textur, Tiefenschärfe, Kontext-/Objekt-Wissen etc., diese sind allerdings wenig genau und können nur die relative Tiefe zwischen verschiedenen Objekten mit einer gewissen Verlässlichkeit schätzen. Zur Kombination verschiedener Kamera-Bilder

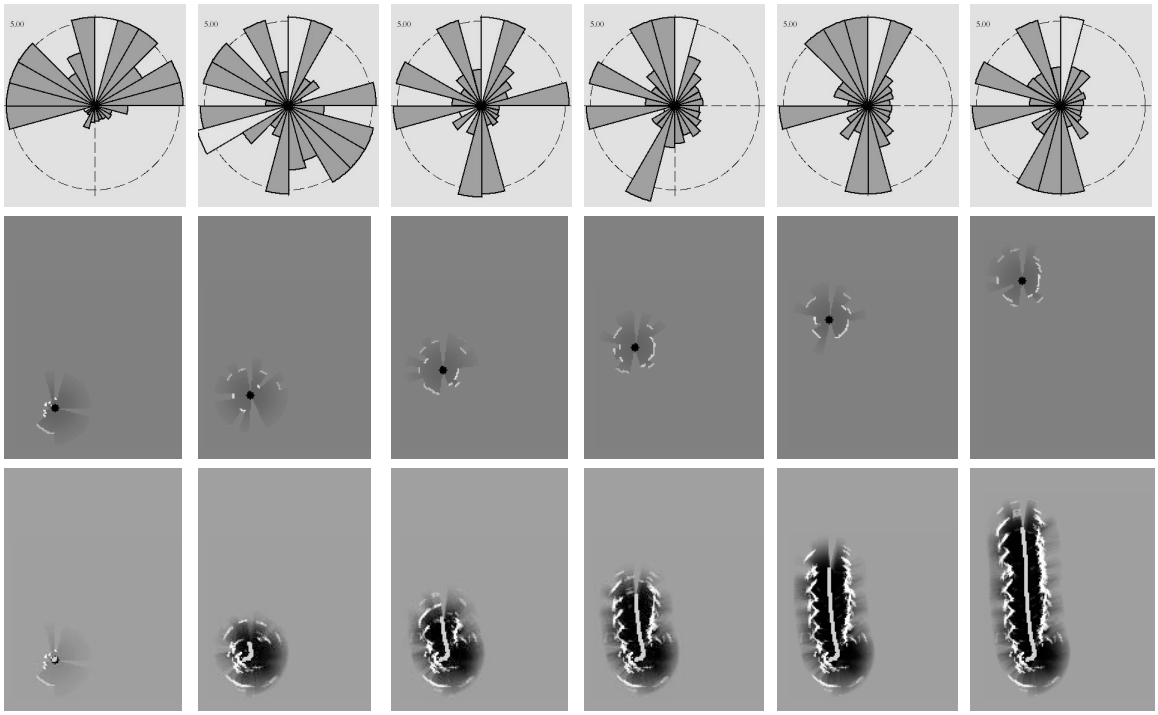

Abbildung 3.6: *Eine Sequenz von Messungen, zugehörigen Zell-Belegtheitsschätzung und akkumulierter Karte: In der ersten Zeile sind einige Messungen eines Sonar-Arrays dargestellt (die nicht ausgefüllten Kegel markieren defekte Sensoren, deren Messung wird ignoriert). Die Entfernungsmessungen der Einzelsensoren sind entsprechend dem roboterinternen Koordinatensystem abgebildet, d.h. bei einer Drehung des Roboters bewegen sich diese nicht mit. Die zweite Zeile zeigt die Belegtheitsschätzungen für die Gridzellen, welche sich aus diesen Messungen ergeben. Zellen mit hoher Belegtheitswahrscheinlichkeit sind hell dargestellt, niedrige Wahrscheinlichkeiten dunkel. Durch die a-priori-Wahrscheinlichkeit im Beispiel $p(occ) = 0.5$ ist die gesamte Fläche zunächst grau. Die Zellauflösung beträgt hier 0.2m. Da der Roboter die Messungen an verschiedenen Positionen aufgenommen hat, sind jeweils unterschiedliche Zellen betroffen. In der letzten Zeile ist die Karte zu sehen, welche sich aus der Verrechnung der Einzelschätzungen ergibt (in diese Karte gehen deutlich mehr als die 6 abgebildeten Messungen ein, aus Platzgründen können nicht alle Einzelschritte gezeigt werden). Zusätzlich ist hier noch zur Verdeutlichung der gefahrene Pfad gezeigt.*

sind prinzipiell zwei Möglichkeiten denkbar: Einerseits können mehrere Kameras die Szene gleichzeitig betrachten („klassischer“ Stereo-Ansatz), andererseits kann sich eine einzelne Kamera durch eine (zum Großteil statische!) Szene bewegen und dabei Bilder von unterschiedlichen Beobachtungspunkten aufzeichnen (Depth from Motion) [Hartley und Zisserman, 2004]. Während bei mehreren feststehenden Kameras die relative Lage von vornherein bekannt ist oder zum Beispiel mittels

Testszenen kalibriert werden kann, ist dies bei einer bewegten Kamera meist weitaus schwieriger. Hier muss die Kameraposition entweder durch einen externen Sensor bestimmt werden, oder aus den aufgenommenen Bildern ebenfalls mit geschätzt werden. In der Robotik kommt häufig auch eine Kombination von Odometrie und bildbasierter Positionskorrektur zum Einsatz.

Als Grundlage werden in dieser Arbeit zwei verschiedene Sensorsysteme und Verfahren zur Schätzung von 3D-Punkten genutzt: zum Einen kommt ein Stereo-Kamera-Aufbau der Firma Videre Design zum Einsatz, bei dem die Berechnung der Disparität und darauf aufbauend der 3D-Koordinaten direkt in Hardware erfolgt (Stereo On Chip (STOC)). Weiterhin wurde ein Szenenrekonstruktions-Verfahren entwickelt, welches während der Eigenbewegung in den Kamera-Bildern einer auf einem Roboter installierten Kamera wiedererkennbare Punktfeatures in der lokalen Umgebung trackt und unter Zuhilfenahme der Odometrie deren 3D-Position berechnet. Beide Verfahren werden im Anhang C detailliert beschrieben.

Während bei Entfernungs-Sensoren jedes beobachtete Objekt auf Höhe des Roboters befindlich und damit als Hindernis betrachtet wurde, ist dies bei 3D-Punkten nicht so. Zwar wird jeder Punkt als ein Objekt aufgefasst, welches potentiell den Weg blockieren könnte, allerdings können auch Punkte auf der Bodenebene oder über der Höhe des Roboters betrachtet werden. Es müssen daher zunächst alle Punkte bzgl. ihrer z-Koordinate (Höhe) mit einer Schwellwert-Operation gefiltert werden, um sie in Boden-/Hindernis- und evtl. irrelevante Punkte zu trennen. Weitere Vorverarbeitungsschritte hängen von den Eigenschaften des zugrunde liegenden Detektionsalgorithmus ab. Meist werden auf einem realen Objekt mehrere Punkte detektiert, vereinzelte Punkte weisen dagegen auf Ausreißer durch Fehldetektionen hin. Diese können durch einen weiteren Filterschritt eliminiert werden, bei dem nur Punkte zugelassen werden, welche eine Mindestanzahl Nachbarn in einem vorgegeben Radius aufweisen. Die verbleibenden Punkte können dann durch Projektion auf die 2D-Ebene in die Gridkarte eingetragen werden.

Falls die Punktkoordinaten exakt bekannt sind, oder die Unsicherheit nicht bestimmt werden kann, so gilt jede Zelle, in welche ein oder mehrere Hindernis-Punkte fallen, als belegt. Diese Zellen müssen also eine Belegtheitswahrscheinlichkeit von 1 bzw. annähernd 1 aufweisen.

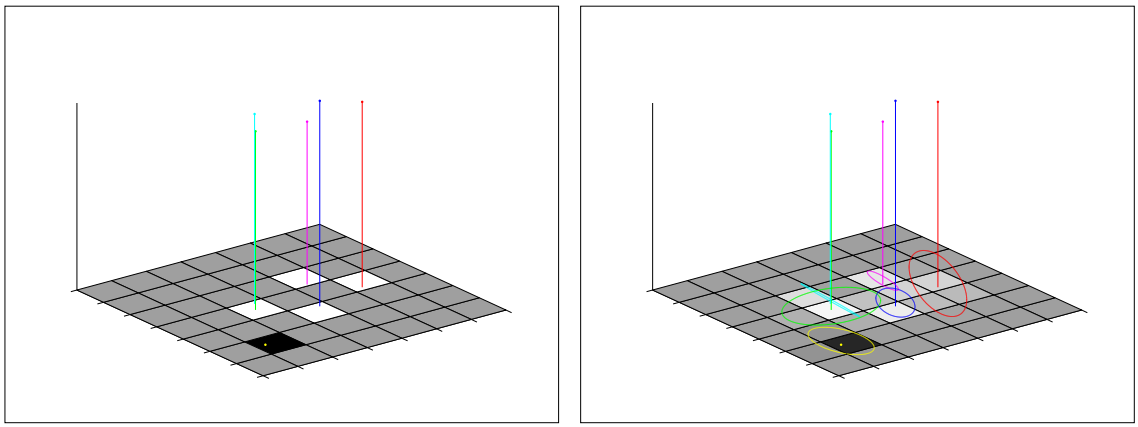

Abbildung 3.7: *Aus 3D-Punkten können ebenfalls Zell-Belegtheitswahrscheinlichkeiten geschätzt werden.* **Links:** *Falls die exakten 3D-Koordinaten bzw. keine Schätzungen der Unsicherheit bekannt sind, so bedeutet jeder Punkt, dass die entsprechende Zelle ein Hindernis enthält, also eine Belegtheitswahrscheinlichkeit von (annähernd) 1 (Hindernis) bzw. 0 (Freiraum) erhält.* **Rechts:** *Falls die Positionsunsicherheit der Punkte z.B. in Form einer Kovarianzmatrix bekannt ist, kann diese Unschärfe in der Verteilung der Zellbelegungen abgebildet werden. Die Ellipsen stellen jeweils die Positionskovarianz des Punktes in der x-y-Ebene dar. Je größer die Unsicherheit, auf umso mehr Zellen wird der Punkt „verteilt“, dadurch wird auch der Maximal-/Minimalwert der Belegtheits-Wahrscheinlichkeit für eine einzelne Zelle geringer. Sowohl im scharfen wie auch im unscharfen Fall wurde eine a-priori-Belegtheitswahrscheinlichkeit von 0.5 angenommen.*

Falls die Unsicherheit der Punktpositionen bekannt ist (z.B. beschrieben durch eine Kovarianzmatrix bei Schätzung der Punktkoordinaten mittels Kalmanfilter), so kann die exakte Verteilung der Zellbelegtheiten berechnet werden. Dazu wird für jede Zelle die Aufenthaltswahrscheinlichkeit eines Hindernis-Punktes errechnet, indem die Wahrscheinlichkeitsdichtefunktion des Punktes (Normalverteilung mit Mittelwert = Punktposition und Kovarianzmatrix) über alle Punkte der Zellfläche ($[x, y] \in cell$) integriert wird. Für die Verrechnung der Belegtheitsschätzungen aus mehreren Einzelpunkten wird folgendes Prinzip angewandt: eine Zelle ist genau dann frei von Hindernispunkten, wenn keiner der gefundenen Punkte in ihr liegt. Da die Punktkoordinaten unabhängig voneinander geschätzt sind, ist die Gesamt-Wahrscheinlichkeit der Nicht-Belegtheit das Produkt der einzelnen Wahrscheinlichkeiten, dass der jeweilige Punkt *nicht* in der Zelle liegt. Für zwei Punkte gilt zum Beispiel

$$
\begin{aligned}
p(\overline{occ}|P_1, P_2) &= p(\overline{occ}|P_1) \cdot p(\overline{occ}|P_2) && (3.7)\\
&= (1 - p(occ|P_1)) \cdot (1 - p(occ|P_2)) && (3.8)\\
&= \left(1 - \int\limits_{[x,y]\in cell} p(P_1 = [x, y])\right) \\
&\quad \cdot \left(1 - \int\limits_{[x,y]\in cell} p(P_2 = [x, y])\right) && (3.9)
\end{aligned}
$$

Verallgemeinert auf n Punkte und unter Berücksichtigung der Normalverteilung für den Ort eines einzelnen Punktes ergibt sich

$$
\begin{aligned}
p(\overline{occ}|P_{1:n}) &= \prod_{i=1}^{n} \left(1 - \int\limits_{[x,y]\in cell} p(P_i = [x, y])\right) && (3.10)\\
&= \prod_{i=1}^{n} \left(1 - \int\limits_{[x,y]\in cell} \mathcal{N}(x, y|\mu_i, C_i)\right) && (3.11)
\end{aligned}
$$

$$
p(occ|P_{1:n}) = 1 - \prod_{i=1}^{n} \left(1 - \int\limits_{[x,y]\in cell} \mathcal{N}(x, y|\mu_i, C_i)\right) \qquad (3.12)
$$

Damit steigt mit *jedem* weiteren Punkt die Belegtheitswahrscheinlichkeit *aller* Zellen, da eine (wenn auch meist sehr geringe) Möglichkeit besteht, dass der Punkt innerhalb einer beliebigen Zelle liegt.

Durch die Detektion von Punkten auf dem Boden ist es unter Umständen auch möglich, Freiraum zu erkennen. Eine Kamera kann typischerweise einen gewissen Höhenbereich wahrnehmen. Offensichtlich sind sichtbare Punkte auf der Bodenebene nur dann ein Hinweis auf Freiraum, wenn nicht gleichzeitig oberhalb dieser weitere Punkte gesehen werden. Ein auf dem Boden stehendes Objekt wird z.B. immer sowohl auf der Bodenebene als auch darüber Wahrnehmungen generieren.

Weiterhin muss beachtet werden, dass Objekte nach oben oder unten aus dem Sichtbereich der Kamera heraustreten und damit „unsichtbar“ werden können. Im vorhergehenden Abschnitt wurden für Entfernungssensoren nur Konfigurationen angenommen, bei denen der Sensor parallel zum Boden ausgerichtet ist und (abstrahiert) in einer Höhen-Ebene misst, so dass dieses Problem nicht zum Tragen kommt. Je nach Kamerakonfiguration muss es jedoch insbesondere bei der Verrechnung mehrerer Beobachtungen berücksichtigt werden. In Abschnitt 4.4.1 wird eine konkrete Implementierung beschrieben.

Da man im Allgemeinen nicht davon ausgehen kann, dass dort, wo keine Punkte detektiert werden, mit Sicherheit keine Objekte existieren, lässt sich aus der aktuellen Beobachtung keine Aussage für die Zellen treffen, die keine Punkte enthalten. Für diese kann also die Belegtheitswahrscheinlichkeit nicht adaptiert werden. In den Beispielen in Abbildung 3.7 wurde die a-priori-Wahrscheinlichkeit 0.5 verwendet und bleibt für alle Zellen erhalten, in denen keine Punkte liegen.

3.1.4 Karten-Fusion

Da ein Robotersystem häufig mit mehreren Sensor-Systemen zur Umgebungsbeobachtung ausgestattet ist, stellt sich natürlich die Frage der Fusion der Sensorbeobachtungen. Die Fusion wird hier auf der Karten-Ebene betrachtet: jeder Sensor erstellt eine separate Gridkarte der Umgebung, diese werden anschließend zu einer einzigen Repräsentation verschmolzen. Dieses Vorgehen kommt u.a. bei der Erstellung lokaler Karten für die Hindernisdetektion und lokale Navigation (Abschnitt 5.4) zur Anwendung. Die Einzel-Karten sollten dazu die selbe Zellgröße aufweisen, ist dies nicht der Fall, muss mindestens eine Karte zunächst in der Auflösung skaliert werden. Dabei werden entweder mehrere Zellen zu einer zusammengefasst, wobei die Belegtheitsschätzung dem Maximum der Teile entspricht, oder eine Zelle wird in mehrere kleine aufgeteilt, dabei bleibt in jeder Teilzelle die Belegtheitsschätzung erhalten.

Die Erstellung unabhängiger Karten für die einzelnen Sensoren ist u.a. deshalb notwendig, weil nicht garantiert werden kann, dass ein Objekt im Erfassungsbereich eines Sensors potentiell überhaupt von einem anderen Sensor wahrgenommen werden kann (Abb. 3.8 links). Ein Sonarsensor und ein Laser-Range-Scanner, welche in unterschiedlicher Höhe Entfernungen messen, würden unter Umständen in der selben Si-

tuation unterschiedliche Entfernungen wahrnehmen. Ein Sensor könnte dann eine bestimmte Zelle immer als „frei“ detektieren, während der andere sie als „belegt“ sieht. Allerdings würde es sich gar nicht um Beobachtungen des selben Zustandes handeln, da das Objekt, welches von einem Sensor wahrgenommen wird, die Messung des anderen gar nicht beeinflusst, z.B. weil es sich unterhalb dessen Messhöhe befindet. Eine direkte Verrechnung dieser Einzelbeobachtungen im Sinne der oben erläuterten Belegtheits-Wahrscheinlichkeits-Schätzung ist also unzulässig und würde fälschlicherweise durch die widersprechenden Beobachtungen zu einem „unbekannten“ Zustand der Zelle führen (falls beide Sensoren gleich oft messen). Wenn dagegen zwei unabhängige Karten jeweils für den Beobachtungsbereich jedes Sensors erstellt werden, so wird sich dieses Objekt korrekt in der einen Karte abbilden, während in der anderen Karte dort Freiraum erscheint.

Bei der Fusion der Karten sollen alle Hindernisse, welche von mindestens einem Sensor wahrgenommmen wurden, in der Gesamt-Karte erhalten bleiben. Es wird also das Maximum der einzelnen Belegtheitswahrscheinlichkeiten gewählt, wenn mindestens einer der Sensoren eine Wahrscheinlichkeit $p(occ) > 0.5$ schätzt. In allen anderen Fällen nehmen alle Sensoren Freiraum wahr, oder haben keine Kenntnis von der Belegtheit der Zelle. Es wird in diesem Fall angenommen, dass der Sensor, der am stärksten Freiraum schätzt (also die geringste Belegtheitswahrscheinlichkeit), auch die sicherste Schätzung darstellt, daher wird diese in die Fusion übernommen.

$$occ_{fusion} = \begin{cases} max(occ_{1:n}) & falls\ max(occ_{1:n}) > 0.5 \\ min(occ_{1:n}) & sonst \end{cases} \tag{3.13}$$

Man könnte an dieser Stelle argumentieren, dass es nicht zulässig ist, in der Fusion Freiraum anzunehmen, wenn einer der Sensoren diese Zelle nicht wahrgenommen hat. An dieser Position könnte sich ein Hindernis befinden, das der Sensor detektieren würde, wenn er diese Zelle beobachtet hätte. Allerdings müsste man dann auch weitergehende Schlüsse aus dem beschränkten Beobachtungsbereich der Sensoren ziehen: Wenn ein Sensor nicht die gesamte Höhe des Raumes beobachten kann, wäre es prinzipiell unmöglich, (bei Verwendung zweidimensionaler Karten) Freiraum zu detektieren, da sich außerhalb des vom Sensor wahrgenommenen Bereiches immer noch ein Hindernis befinden könnte. Die Belegung wäre also nur dort sicher, wo Hindernisse wahrgenommen werden,

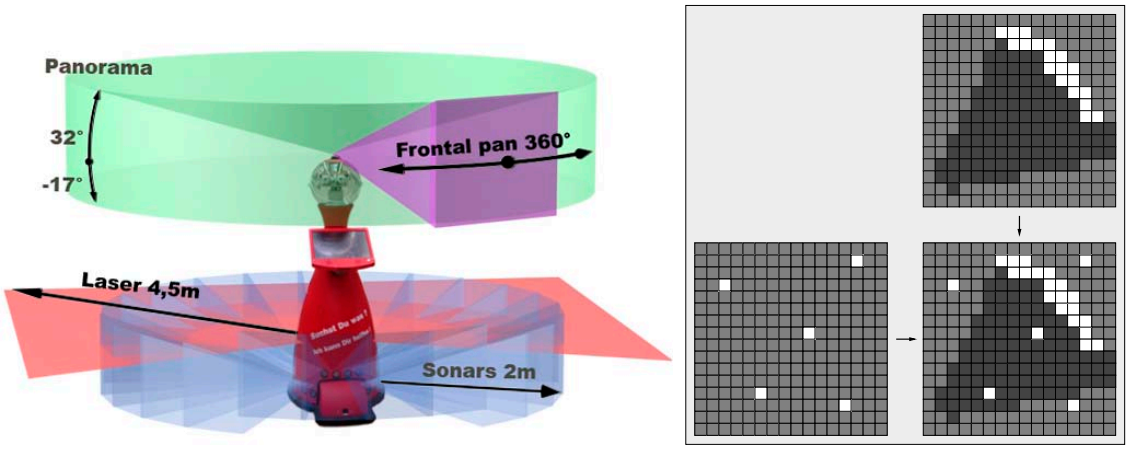

Abbildung 3.8: **Links:** *Verschiedene Sensoren haben unterschiedliche Erfassungsbereiche und können daher u.U. nicht alle die selben Objekte beobachten.* **Rechts:** *Zwei Ausgangskarten, die aus den Beobachtungen verschiedener Sensoren (z.B. Laser- und Sonar-Entfernungssensoren) erstellt wurden, werden fusioniert, indem jeweils die höhere Schätzung für jede Zelle gewählt wird, es sei denn, die Belegtheit einer Zelle ist in einer Karte unbekannt (Wahrscheinlichkeit = 0.5) oder alle Karten schätzen Freiraum für eine Zelle.*

für alle anderen Zellen wäre der Zustand unbekannt. Dies wäre zwar theoretisch exakter, praktisch aber wenig brauchbar.

3.2 Odometriekorrektur

Zum Aufbau einer globalen Karte, aber auch für die autonome Bewegung muss der Roboter ständig seine aktuelle Position in der Umgebung kennen. Die Grundlage der Positionsbestimmung bildet dabei fast immer die Odometrie, also die Messung der Eigenbewegung mittels interner Sensoren, meist Umdrehungssensoren an den Antriebsrädern. Die Bewegungsmessung unterliegt allerdings verschiedenen Abweichungen, welche in systematische und nicht-systematische Fehler unterteilt werden [Borenstein und Feng, 1996]. In die Kategorie der systematischen Fehler fallen z.B. fehlerhafte oder ungenaue Angaben über Raddurchmesser und -abstände oder ungleichmäßige Form der Radlauffläche (all dies kann z.B. durch Abnutzung hervorgerufen werden). Nicht-systematische Fehler bestehen vor allem aus Schlupf an angetriebenen Rädern. Systematische Fehler können durch entsprechende Kalibrationsverfahren bestimmt und eliminiert werden, nicht-systematische

Fehler sind hingegen ihrem Wesen nach nur in der Größe abschätzbar, können aber nicht ausgeschlossen werden.

Odometrie allein ist daher zwar prinzipbedingt nicht geeignet, langfristig eine genaue und konsistente Positionierung zu gewährleisten, so dass Lokalisations-Verfahren notwendig sind, die die Position durch Abgleich mit der Umgebung korrigieren. Diese bauen jedoch auf den Odometriemessungen auf und profitieren von einem geringen Fehler in diesen. Es wird daher angestrebt, eine möglichst hohe Genauigkeit in den Odometrie-Messungen zu erzielen. Deshalb wurden verschiedenen Ansätze implementiert und getestet, um die Odometriemessungen zu verbessern. Diese werden in diesem Abschnitt vorgestellt. Zum Teil waren diese Korrekturverfahren bereits Inhalt der Veröffentlichungen [Schröter et al., 2003], [Schröter et al., 2004] und [Schröter, 2004].

3.2.1 Bestimmung des systematischen Fehlers

Wie bereits im Einführungskapitel dargestellt, wurde vor der Konstruktion der SCITOS-Plattform für den Shopping-Assistenten vor allem der Roboter PERSES für Navigationsexperimente benutzt. PERSES baut auf einem RWI B21r auf, welcher sich mittels eines Synchro-Drive-Antriebs bewegt. Da es sich nicht um einen direkten Beitrag zum Shopping-Assistenten (Differential Drive) handelt, die Odometriekorrektur u.U. jedoch eine Basis für den Einsatz nachfolgender Verfahren auch auf anderen Plattformen darstellt, soll die Korrektur des systematischen Fehlers für den Roboter PERSES in diesem Kapitel nur sehr kurz dargelegt werden. Eine detaillierte Beschreibung der auftretenden Fehler und deren experimenteller Bestimmung findet sich in Anhang A.

Abbildung 3.9 zeigt die auftretenden Odometrie-Fehler bei PERSES an einem sehr einfachen Referenzpfad. Die relativ regelmäßige Struktur des Odometriefehlers ist gut zu erkennen. Bei einer geradlinigen Fahrt wird vom Odometriesystem fälschlich eine Kurvenfahrt gemessen, damit weicht die Odometrieposition und -orientierung schon nach kurzer Fahrtstrecke (im Beispiel ca. 350m) sehr stark von der realen Roboterpose ab.

Die Fehler entstehen durch eine Krümmung der realen Trajektorie bei reiner Translationsbewegung (eigentlich geradliniger Fahrt) des Roboters, in Folge von Abnutzungserscheinungen der Räder. Um einem

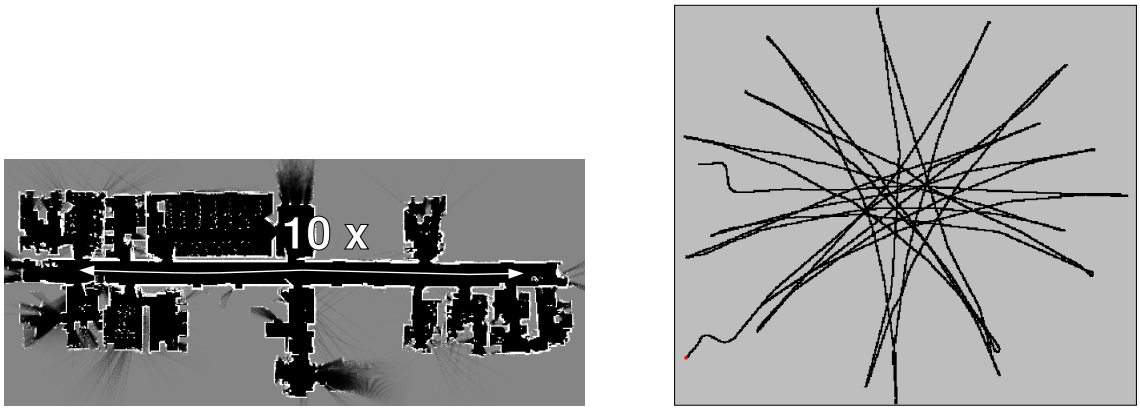

Abbildung 3.9: **Links:** *Schematische Darstellung des tatsächlichen Roboterpfades: Um die auftretenden Fehler deutlicher erkennbar zu machen, fuhr der Roboter den geraden Gang (Länge ca. 35m) zehn mal hin und her.* **Rechts:** *Odometriemessung des Pfades: Es ist gut zu sehen, dass die Messfehler sehr groß und dabei relativ regelmäßig sind. Dies weist darauf hin, dass diese vor allem durch systematische Fehler entstehen und damit relativ gut korrigiert werden können.*

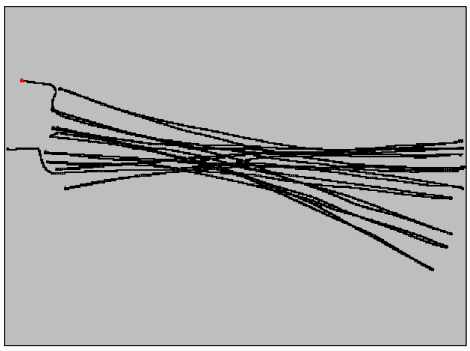

Abbildung 3.10: *Der Pfad mit Odometrie-Korrektur (vgl. Abb. 3.9): Die Odometriemessungen sind immer noch fehlerbehaftet und der Positionsfehler kann weiterhin unbegrenzt anwachsen, aber die Größenordnung der Abweichung ist deutlich geringer.*

geraden Gang zu folgen, muss der Roboter daher kontinuierlich die Fahrtrichtung korrigieren, die Odometrie misst dadurch eine Bewegung ähnlich einer Kreisbahn. Die Bahnkrümmung ist zudem nicht konstant, sondern abhängig von der Roboterorientierung. Durch die Bestimmung dieser Krümmung kann der Fehler modelliert und die Odometriemessungen kontinuierlich korrigiert werden.

In Abb. 3.10 sind die Odometriemessungen nach der Korrektur zu sehen. Es ist offensichtlich, dass auch mit dieser Korrektur die Position immer noch recht ungenau ist und der Positionsfehler unbegrenzt anwachsen kann. Allerdings sind die Fehler deutlich geringer als ohne Korrektur. Damit ist eine Basis gegeben, um durch weiterführende Ansätze (Abschnitt 4.1, Kapitel 4) während des Kartenaufbaus den verbleibenden Odometriefehler zu berücksichtigen und zu korrigieren.

3.2.2 Orientierungs-Korrektur mittels Bodenstruktur

Die Kalibrierung gemäß Abschnitt 3.2.1 und Anhang A verbessert die Odometrie erheblich, trotzdem ist die Abweichung zwischen der Messung und dem real gefahrenen Pfad immer noch recht hoch. Es wurde bereits erwähnt, dass die Messung der absoluten Fahrstrecke relativ genau ist, die Bestimmung der Orientierung/Fahrtrichtung dagegen größere Fehler aufweist. Weiterhin lässt sich leicht nachvollziehen, dass kleine Fehler in der Orientierung sehr große Positions-Abweichungen zur Folge haben können: Schon eine geringe konstante Differenz zwischen realer und geschätzter Orientierung lässt den absoluten Positionsfehler stetig anwachsen (Abb. 3.11 links).

Es erscheint daher vor allem wünschenswert, die Roboterorientierung möglichst genau zu bestimmen. Um eine bessere Genauigkeit zu erreichen, als durch die Kalibrierung möglich ist, muss eine externe Referenz genutzt werden. In der Test-Umgebung des Baumarktes bietet sich dafür die Bodenstruktur an: der Bodenbelag besteht aus einzelnen Fliesen, welche über die gesamte Fläche in der selben Ausrichtung verlegt sind. Die Orientierung der Fliesenkanten bildet damit eine geeignete Referenz für die Roboter-Orientierung (Abb. 3.11 rechts, Abb. 1.1).

Der Boden vor dem Roboter wird dazu mittels einer zu diesem Zweck angebrachten Kamera aufgenommen. Eine senkrechte Blickrichtung der Kamera hat zwei wesentliche Vorteile: die Vorverarbeitung des Bildes wird vereinfacht, da die Bestimmung der Ausrichtung in einer Draufsicht einfacher ist, außerdem ist der beobachtete Bereich kleiner und damit die Wahrscheinlichkeit geringer, dass andere Objekte als der Boden im Bild sichtbar sind und die Orientierungsschätzung stören. Andererseits kann eine schräg angebrachte Kamera für andere Zwecke genutzt werden bzw. schon vorhanden sein, z.B. für eine visuelle Hindernisdetektion. In diesem Fall ist es natürlich sinnvoll, das Bild dieser Kamera für die Odometrie-Korrektur mit zu nutzen. Für die in diesem Abschnitt beschriebenen Untersuchungen wurde allerdings eine senkrechte Kamera genutzt. Die Kamera ist in Fahrtrichtung vorn am PERSES in einer Höhe von ca. 1m angebracht und kann damit einen Bereich von etwa 1m $\times$ 0.6m erfassen.

Das Kamerabild muss zunächst vorverarbeitet werden: Da im Sichtbereich der senkrechten Kamera auch der Roboter selbst teilweise abgebil-

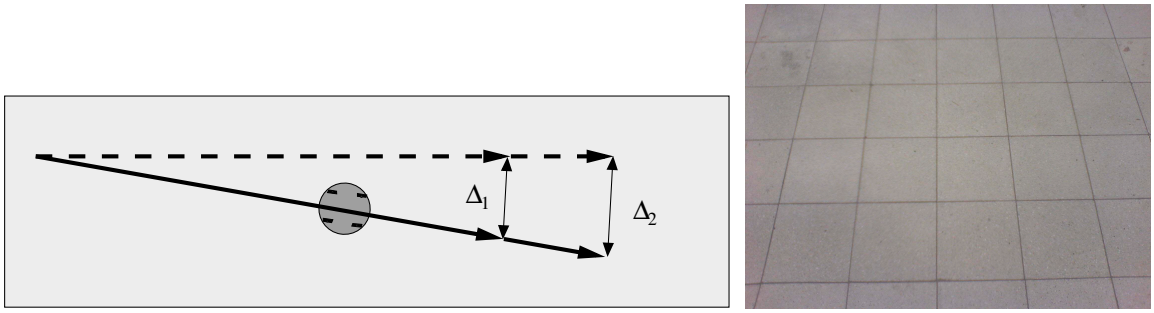

Abbildung 3.11: **Links:** *Schon bei geringem konstanten Orientierungsfehler wächst der Abstand zwischen realer und gemessener Position konstant und unbegrenzt an. Die Korrektur der Roboter-Orientierung mittels einer externen Referenz eliminiert daher die wichtigste Fehlerursache.* **Rechts:** *Bodenfliesen, Nahansicht (siehe auch Abb.1.1)*

det wird, wird nur ein Ausschnitt des Bildes verwendet, typischerweise entspricht dieser ungefähr den oberen zwei Dritteln des Bildes. Weiterhin wirkt sich die Kameraverzerrung negativ auf das Ergebnis aus, insbesondere der radiale Anteil, welcher bewirkt, dass eigentlich gerade und parallele Linien im Kamerabild gekrümmt erscheinen. Hier muss also eine Verzerrungskorrektur auf dem Bild erfolgen. Dazu muss für jedes Pixel die Transformation

$$r^2 = (x_{pixel} - x_{center})^2 + (y_{pixel} - y_{center})^2 \tag{3.14}$$

$$scale = 1 + r^2 \cdot k \tag{3.15}$$

$$\begin{bmatrix} x_{corr} \\ y_{corr} \end{bmatrix} = scale \cdot \begin{bmatrix} x - x_{center} \\ y - y_{center} \end{bmatrix} + \begin{bmatrix} x_{center} \\ y_{center} \end{bmatrix} \tag{3.16}$$

mit dem radialen Korrekturfaktor k berechnet werden [Hartley und Zisserman, 2004]. Es ist allerdings deutlich effizienter, diesen Korrekturschritt nicht auf dem gesamten Eingangsbild durchzuführen. Stattdessen werden zunächst die interessierenden Kantenpixel bestimmt und nur für diese die Lage im unverzerrten Bild berechnet. Auf diese Weise wird die Transformation nur auf einer kleinen Teilmenge aller Bildpixel durchgeführt und damit erheblich Rechenzeit eingespart.

Ursprünglich wurde für die Berechnung der Kantenorientierung im Bild ein lokaler Orientierungstensor benutzt. Die mit der Kantenstärke gewichtete Summation der lokalen Kantenrichtungen aller Bildpixel ergibt die Hauptorientierung des Bildes. Allerdings hat dieser Ansatz Schwierigkeiten bei Kanten, die *fast*, aber nicht exakt horizontal oder verti-

kal im Bild verlaufen: durch die Diskretisierung in Bildpixel erscheinen solche Linien als eine Aneinanderreihung jeweils leicht versetzter horizontaler bzw. vertikaler Linienstücke. Die lokale Orientierung ist dann nicht von einer tatsächlich waagerecht/senkrecht im Bild verlaufenden Linie unterscheidbar. Betragsmäßig kleine Orientierungsänderungen gegenüber dem Bezugssystem werden dadurch in der Schätzung unterdrückt.

Ein besserer Ansatz ist die Nutzung der Hough-Transformation. Dazu wird zunächst für alle Pixel im Bild die Kantenstärke bestimmt, unabhängig von deren -richtung. Pixel, deren Kantenstärke einen Schwellwert unterschreitet, werden für die weitere Verarbeitung nicht berücksichtigt. Dieser Schwellwert wird dynamisch mit Hilfe der maximalen Kantenstärke im Bild angepasst. Für die verbleibenden Kantenpixel wird eine (d, φ)-Hough-Tabelle aufgebaut. Dabei wird jeder Kantenpixel (x,y) in die Menge aller diesen Pixel schneidenden Geraden (d, φ) überführt. Die Parameter d und φ sind die Parameter der Geradengleichung in Hesse'scher Normalform, es gilt

$$d = x \cdot \cos(\varphi) + y \cdot \sin(\varphi) \tag{3.17}$$

Die entsprechenden Werte in der (d, φ)-Tabelle werden für jeden Pixel inkrementiert, die Tabelle akkumuliert so die Geraden über alle Pixel. φ entspricht dabei zugleich der gesuchten Kantenorientierung. Die Auflösung der Tabelle ergibt sich daher aus der gewünschten Genauigkeit der Orientierungsschätzung, für die Odometriekorrektur wurde eine Auflösung von 1° gewählt. Die Zellwerte der Tabelle geben somit nach dem kompletten Durchlauf des Bildes an, wie viele Pixel auf einer Geraden mit den entsprechenden Parametern liegen (Abb. 3.12). Da einzelne Kanten nicht unterschieden werden sollen, sondern nur deren Orientierung gesucht ist, wird abschließend in der Hough-Tabelle eine Integration über alle zu einem Winkel gehörenden Entfernungen durchgeführt. Dabei kommt eine Besonderheit zum Tragen: das zugrunde liegende Bodenmuster weist eine Rotationsperiode von 90° auf, d.h. Orientierungen außerhalb eines Intervalls von (0°, 90°] können nicht (sinnvoll) erkannt werden. Daher wird bei der Integration kein Histogramm über die Orientierungen (0, 179), sondern nur über (0, 89) erstellt, wobei der hintere Bereich auf den vorderen addiert wird. Der Maximalwert in diesem Histogramm ergibt die Kantenorientierung im Bild.

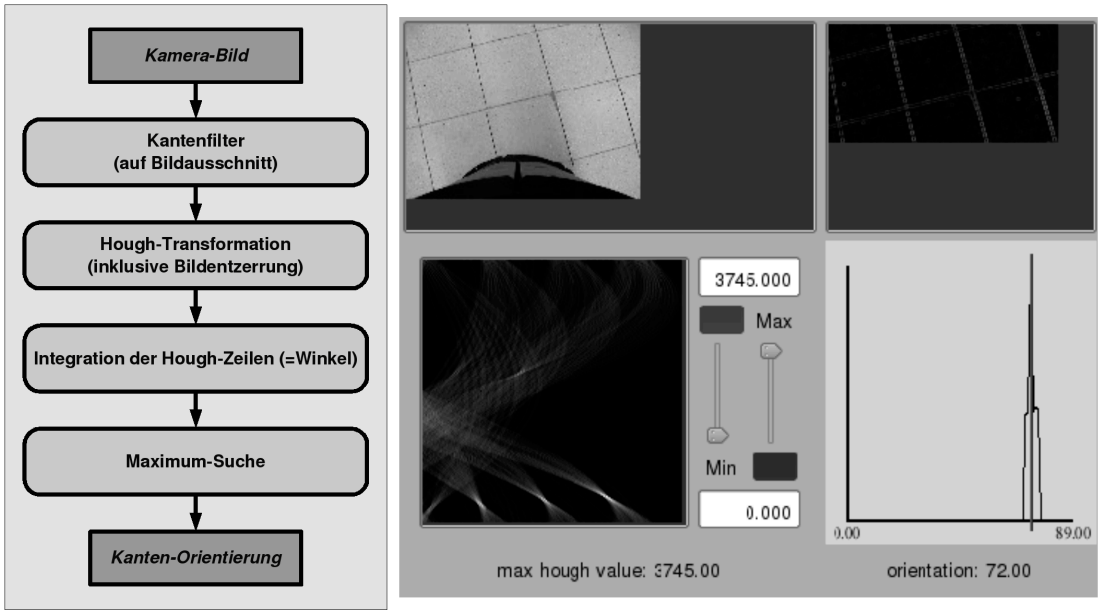

Abbildung 3.12: **Links:** *Ablaufschema der Orientierungsschätzung mittels Hough-Transformation.* **Rechts:** *Zwischenergebnisse am Beispiel: Eingangsbild (links oben), Kantendetektion auf relevantem Bildausschnitt (rechts oben), Hough-Tabelle (links unten), Winkel-Histogramm und berechnete Orientierung (rechts unten).*
In der Hough-Tabelle sind gut die 3 lokalen Maxima am unteren Rand erkennbar, die den 3 parallelen annähernd senkrechten Linien im Bild entsprechen. Ein weiteres (schwächeres) Maximum im mittleren Bereich wird von der ungefähr waagerechten Linie verursacht. Da aufgrund der Ununterscheidbarkeit größerer Verdrehungen nur eine Orientierung zwischen 0 und 90 ° geschätzt wird, stützen sowohl die senkrechten als auch die waagerechte Linie die selbe Winkelschätzung.

Mit der Bodenstruktur als Referenz kann nun kontinuierlich der Orientierungsfehler erfasst und korrigiert werden. Die Korrektur erfolgt dabei nach dem selben Muster wie im Anhang A bei der Bestimmung des systematischen Fehlers beschrieben (Abb. A.3, Gl. A.8 - A.11), wobei allerdings statt der Bahnkrümmung c direkt die Verdrehung α im aktuellen Schritt bekannt ist, diese entspricht der beobachteten Differenz zwischen Referenz- und Odometrieorientierung.

$$d' = \frac{2d\sin(\frac{\alpha}{2})}{\alpha} \tag{3.18}$$

$$\varphi' = \varphi + \frac{\alpha}{2} \tag{3.19}$$

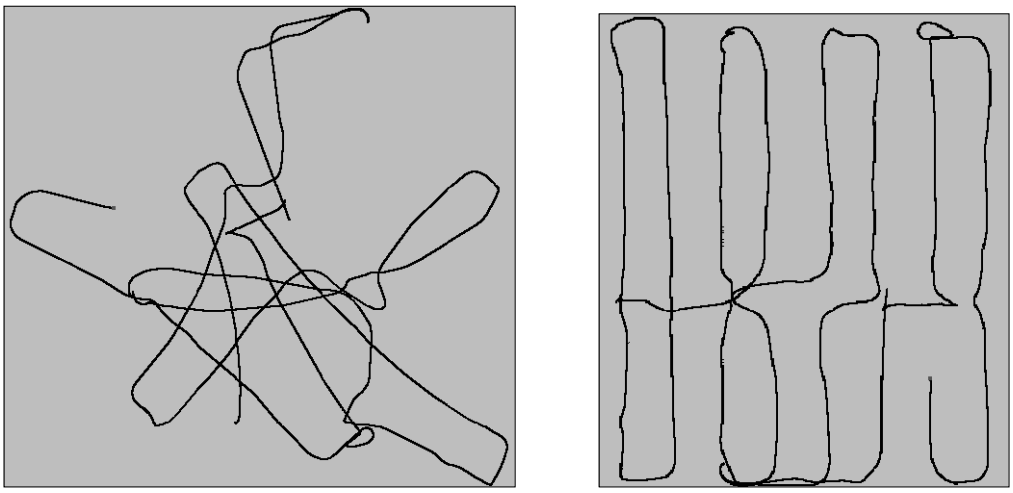

Abbildung 3.13: *Ergebnis der Odometrie-Korrektur mittels Orientierungsreferenz: die linke Darstellung zeigt die unkorrigierte Odometrie (hier auch ohne die im vorhergehenden Abschnitt beschriebene Kalibrierung. Im rechten Bild ist das Korrektur-Ergebnis dargestellt: man erkennt, wie der Roboter verschiedene Schleifen in der aus vielen parallelen Gängen bestehenden Umgebung fährt. Eine numerische Auswertung der Positionsgenauigkeit findet sich in Tabelle 3.1. Eine Grundriss-Skizze der realen Umgebung ist zum Vergleich in Abb. 3.14 abgebildet.*

Für sehr kleine α gilt $\sin\frac{\alpha}{2} \approx \frac{\alpha}{2}$, dann kann d' vereinfacht mit d gleich gesetzt werden. In diesem Fall wird nur die Fahrtrichtung korrigiert.

Abb. 3.13 zeigt den Vergleich von originaler (unkalibrierter) Odometrie, und der daraus mittels Orientierungsreferenz korrigierten Position. Diese Fahrt wurde im Test-Baumarkt aufgenommen, welcher zum Großteil aus einer Anordnung vieler paralleler Gänge besteht. Im korrigierten Pfad ist gut zu erkennen, wie der Roboter in diesen Gängen Schleifen fuhr. Eine quantitative Analyse der verbleibenden Positionsfehler findet sich in Tabelle 3.1. Für die quantitative Analyse wurden in regelmäßigen Abständen Punkte in der Baumarkt-Umgebung markiert und deren Position manuell vermessen. Beim Überfahren dieser Punkte wird die geschätzte Roboterposition mit der tatsächlichen Position verglichen.

Der Positionsfehler wurde auf 2 verschiedene Arten angegeben: Zunächst wurde als Position direkt die kontinuierlich korrigierte Odometrie verwendet, wobei die Start-Orientierung ebenfalls aus der Bodenstruktur bestimmt wurde (da auf diese Weise nur die Orientierung innerhalb eines 90°-Quadranten ermittelt werden kann, muss aber der Quadrant, also die grobe Orientierung, manuell vorgegeben werden). Diese Werte

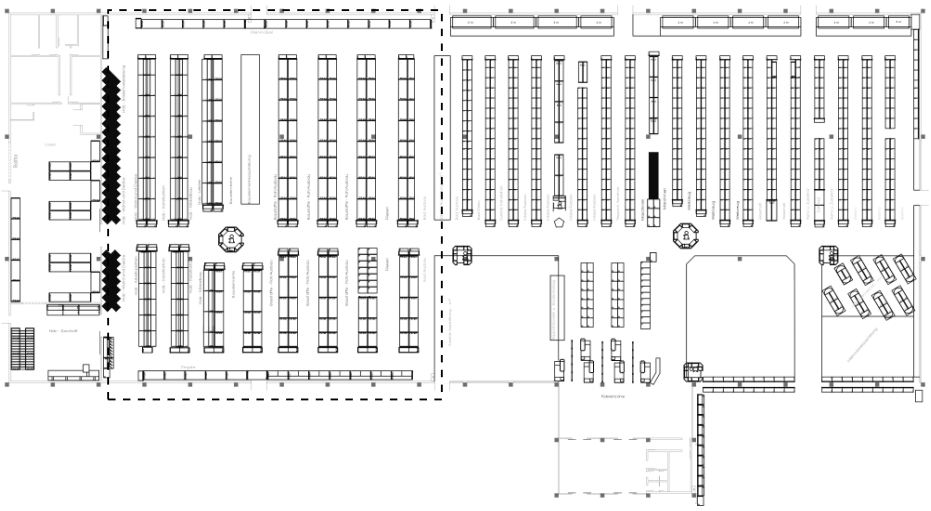

Abbildung 3.14: *Referenzkarte der Baumarkt-Umgebung: Der für die Experimente in diesem Abschnitt genutzte Bereich (u.a. Abb. 3.13) ist durch den gestrichelten Rahmen markiert*

sind in der Tabelle in der Spalte „Odometrie korrigiert (Init Boden)“ aufgeführt. Wie bereits am Anfang zur Motivation dieses Abschnittes beschrieben, ist die Exaktheit der Orientierungsschätzung von großer Bedeutung für die Positionsgenauigkeit. Dies gilt hier insbesondere für die Schätzung der Startorientierung: Jeder Fehler dreht den geschätzten Pfad des Roboters gegenüber dem Bezugssystem und sorgt damit mit steigender Entfernung vom Ausgangspunkt für größere Distanzen. Da die Bestimmung der Orientierung nur auf 1° genau erfolgt, ist ein mittlerer betragsmäßiger Fehler von 0.25° zu erwarten (uniform verteilter Fehler im Intervall (-0.5°, 0.5°)). Der Roboterpfad kann dabei allerdings in sich selbst stimmig, jedoch gegenüber dem Bezugssystem mit den vermessenen Positionsmarken verdreht sein.

Um eine bessere Aussage über die Konsistenz der Schätzungen auf dem Pfad zu erhalten, wurde daher die Start-Orientierung zusätzlich auf eine alternative Weise berechnet: Diese Orientierung wurde dabei so initialisiert, dass die Bewegungsrichtung zwischen dem Startpunkt und dem ersten erreichten Referenzpunkt exakt stimmt, d.h. auf dieser Strecke soll keine seitliche Abweichung vorhanden sein. Da beide Punkte bei $x = 0.00$ liegen, wurde die Orientierung am Startpunkt so bestimmt, dass beim Erreichen des Referenzpunktes die x-Koordinate der geschätzten Roboterposition ebenfalls 0.00 ist. Dies ist natürlich nur durch manuelle Korrektur möglich, da der Roboter selbst keine Kenntnis von der Existenz der Referenzpunkte hat. Der Pfad ist aller-

Strecke	Referenzpos.		Odometrie korrigiert (Init Boden)			Odometrie korrigiert (Init Referenz)		
	x	y	x	y	Dist.	x	y	Dist.
26.15	0.00	25.23	0.38	25.41	0.39	0.00	25.04	0.19
78.83	0.00	-16.70	-0.04	-16.69	0.04	0.22	-16.69	0.22
96.21	0.00	0.00	0.38	-0.02	0.38	0.39	-0.01	0.39
106.62	10.08	0.00	10.46	-0.35	0.52	10.46	-0.26	0.46
137.58	10.08	25.23	11.44	24.71	1.46	11.12	24.91	1.09
163.45	10.08	0.00	10.79	-0.40	0.81	10.86	0.17	0.80
180.50	10.08	-16.70	10.60	-17.07	0.64	10.85	-16.98	0.82
202.35	10.08	0.00	11.10	-0.30	1.06	11.10	-0.22	1.04
236.81	21.01	25.23	22.43	24.36	1.67	22.04	24.63	1.19
268.71	21.01	0.00	21.94	-0.87	1.27	21.94	-0.60	1.11
295.55	10.08	-16.70	10.51	-17.32	0.75	10.77	-17.23	0.87
311.58	21.01	-16.70	21.40	-17.58	0.96	21.65	-17.32	0.89
340.81	31.33	0.00	32.09	-1.35	1.55	32.10	-0.93	1.21
369.17	31.33	25.23	32.42	23.90	1.72	32.04	24.32	1.15
423.72	31.33	-16.70	31.38	-18.04	1.34	31.64	-17.63	0.98
				Mittelwert:	0.97			0.83

Tabelle 3.1: *Die Tabelle zeigt die Positionsfehler bei der Odometriekorrektur mit Hilfe der Bodenstruktur als Referenz. In den Spalten „Odometrie korrigiert (Init Boden)“ beruht die Initialorientierung des Roboters auf der Schätzung der Boden-Orientierung. In den Spalten „Odometrie korrigiert (Init Referenz)“ wurde die Start-Orientierung manuell so korrigiert, dass die Fahrt-Richtung zwischen Startpunkt und erstem Referenzpunkt korrekt ist.*

dings damit besser im Bezugssystem der Referenzpositionen ausgerichtet und erlaubt eine bessere Bewertung der erreichbaren Genauigkeit bei optimaler Initialisierung (bzw. wenn die Referenzpunkte im Roboter-Bezugssystem vermessen wären). Diese Ergebnisse sind in Tabelle 3.1 in der Spalte „Odometrie korrigiert (Init Referenz)“ dargestellt.

Kapitel 4

SLAM - Simultaneous Localization And Mapping

Im vorhergehenden Kapitel „Grundlagen des Kartenaufbaus“ wurden unter anderem Verfahren zur Odometriekorrektur untersucht. Es wurde gezeigt, dass die Genauigkeit der Roboter-Odometriemessungen mit relativ einfachen Verfahren deutlich gesteigert werden kann. Allerdings gilt trotz dieser Verbesserungen, dass die Odometrie keine zuverlässige Positionsschätzung liefern kann, da der Fehler prinzipiell unbegrenzt wachsen kann.

Eine Möglichkeit zur Gewährleistung einer konsistenten und robusten Kartierung wäre die Nutzung einer externen Positionsreferenz (Global Positioning System (GPS), Radio Frequency Identification (RFID) Tagging, etc.). Diese hat allerdings den Nachteil, dass sie auf das Vorhandensein bzw. die Installation weiterer Systeme (neben dem eigentlichen Roboter) in der entsprechenden Umgebung angewiesen ist und die Anbindung an diese Systeme in der Regel zusätzliche spezifische Hardware erfordert. Dafür sind häufig aufwändige Eingriffe in die Umgebung notwendig, die die Autonomie des Roboters in Frage stellen.

Eine hier vorgezogene Alternative ist daher die direkte Nutzung der während der Kartierung gewonnenen Sensor-Information, um gleichzeitig die Position zu korrigieren und damit die Konsistenz der entstehenden Karte zu sichern. Kartierung und Positionsbestimmung werden hierbei als integrierte, gegenseitig abhängige Teilprobleme betrachtet:

Die Erstellung der Karte erfordert die Kenntnis der aktuellen Position des Roboters zu einer Umgebungsbeobachtung. Die Bestimmung der Position zu einer Beobachtung ist wiederum bei Vorhandensein eines Umgebungsmodells möglich. Für dieses Problem und Ansätze zu dessen Lösung hat sich daher der Begriff „Simultaneous Localization and Mapping (SLAM)“ etabliert [Smith und Cheeseman, 1986], [Leonard und Durrant-Whyte, 1991], [Csorba, 1997].

In diesem Kapitel wird zunächst ein einfaches Positionskorrektur-Verfahren beschrieben, welches darauf beruht, bei einem Kreisschluss des Roboterpfades die aktuelle Positionsschätzung an die Umgebungsbeobachtung anzupassen. Da das Verfahren nicht auf bestimmte Sensorik beschränkt sein soll, werden Scan-Matching-Verfahren, wie sie bei Laser-Range-Scannern üblich sind, z.B. [Chen und Medioni, 1991], [Gutmann und Schlegel, 1996], nicht betrachtet: Um die relative Beobachtungsposition für aufeinander folgende Scans zu schätzen und damit die fehlerhafte Odometrie zu korrigieren, sind sehr hoch aufgelöste und genaue Messungen nötig. Insbesondere bei Sonar-Sensoren mit typischerweise recht breitem Öffnungskegel und damit geringer Richtungsauflösung ist ein solches direktes Matching der Messungen nicht anwendbar. Stattdessen wird zur Bestimmung der besten Position ein *Map Matching* durchgeführt: aus den aktuellsten Beobachtungen wird eine lokale Karte erstellt, diese wird mit der globalen Karte verglichen. Durch die Integration mehrerer Messungen von unterschiedlichen Positionen kann trotz der Einschränkungen der Sensoren eine gute Auflösung in der Karte erreicht werden. Der Vergleich der lokalen mit der globalen Karte erlaubt die Bewertung von hypothetischen Roboterpositionen in der unmittelbaren Umgebung der von der Odometrie gemessenen Position. Im Korrektur wird jeweils die Position angenommen, für die sich die beste Übereinstimmung ergibt.

Es wird gezeigt, dass in einfachen Umgebungen ein derartiger simpler Korrektur-Ansatz durchaus zu brauchbaren Ergebnissen führt. Allerdings bleiben auch viel Probleme und ungeklärte Fragen bestehen. Daher wird dieser Ansatz zu einem deutlich mächtigeren probabilistischen Map-Match-SLAM-Verfahren weiterentwickelt: Das Map-Match-Verfahren zur Bewertung einer Positionshypothese wird kombiniert mit dem Modellierungsprinzip des Rao-Blackwellized Particle Filter (RBPF) [Murphy, 1999]. Grundprinzip des RBPF für SLAM ist die Nutzung eines Partikelfilters (siehe Anhang B) zur Approximation der Wahr-

scheinlichkeitsverteilung über die Roboterposition. Jedes Partikel beschreibt damit einen diskreten Pfad und erstellt eine eigene Gridkarte unter Nutzung dieser Pfad-Schätzung. Im Gegensatz zu anderen Verfahren, welche RBPF und Gridkarten für SLAM einsetzen [Hähnel et al., 2003], [Eliazar und Parr, 2003], und dabei typischerweise Laser-Range-Scanner verwenden, ist das hier vorgestellte Verfahren durch die Nutzung der lokalen Karte als Zwischenrepräsentation für verschiedene Sensoren anwendbar, z.B. Sonar-Sensoren (geringe Auflösung, hohe Messvarianz) und visuelle Vorverarbeitungs-Verfahren zur Detektion von 3D-Punkten aus Kamera-Bildern. Während bei den genannten existierenden Verfahren sowohl ein Algorithmus zur Erstellung der globalen Karte aus den Beobachtungen (bzw. deren Wahrscheinlichkeitsverteilung $P(m|x, o)$) als auch ein Modell zur Bewertung der Wahrscheinlichkeit einer Beobachtung bei gegebener Karte $P(o|x, m)$ benötigt wird, ist durch die Transformation der Beobachtungen in die lokale Karte die Bestimmung dieses Beobachtungsmodells nicht mehr notwendig. Stattdessen wird ein Ähnlichkeitsmaß zum Vergleich der lokalen und globalen Karte benutzt, welches unabhängig vom jeweiligen Sensor definiert werden kann. Um die Übertragbarkeit des vorgestellten Ansatzes auf verschiedene Sensoren nachzuweisen, zeigt dieses Kapitel die SLAM-Kartierung sowohl auf der Basis von Entfernungsmessungen der Sonar-Sensoren als auch mit Beobachtungen von 3D-Punkten, welche von einer Stereo-Kamera bzw. einem Depth-from-Motion-System detektiert wurden.

Abschließend wird als Erweiterung des SLAM-Verfahrens ein SLAM-Assistent vorgestellt, dessen Aufgabe ist, während der Kartierung aus den jeweils zur Verfügung stehenden Informationen unter Berücksichtigung des internen Zustandes den (bzgl. der zur Selbstlokalisation und Kartierung benötigten Daten) optimalen Pfad zu berechnen und damit einem uneingewiesenen Nutzer eine bestmögliche Kartierung auch ohne genaue Kenntnis der Funktion des SLAM-Algorithmus zu ermöglichen.

4.1 Positionskorrektur durch Map Matching

Wie bereits beschrieben, treten durch Fehler in der Odometrie Abweichungen zwischen der realen und der geschätzten Roboter-Position auf. Bei der Kartierung führen diese Fehler zu einem fehlerhaften Umge-

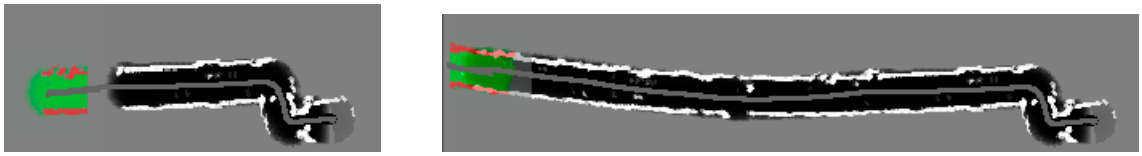

Abbildung 4.1: **Links:** *Die lokale Karte (grün-rot) besteht jeweils aus den aktuellsten Beobachtungen. In die globale Karte (schwarz-weiß) gehen nur die älteren Beobachtungen ein. Bei einer geradlinigen Fahrt liegt damit die lokale Karte vor der globalen Karte.* **Rechts:** *Beim erneuten Betreten einer vorher bereits besuchten Position tritt ein Überlapp zwischen lokaler und globaler Karte auf, durch Abgleich der Karten kann eine Positionskorrektur erfolgen.*

bungsmodell. Kleine Positionsfehler sind besonders bei einem Kreisschluss gut zu erkennen: Erreicht der Roboter-Pfad während der Kartierung eine Position, an der bereits zuvor Beobachtungen aufgenommen wurden, so wird die Karte mit der momentanen Positionsschätzung und neuen Beobachtungen aktualisiert. Bei exakter Positionsschätzung und statischer Umgebung stimmen die Beobachtungen des Umgebungszustandes überein und die Karte verändert sich nicht. Bei einer fehlerhaften Positionsschätzung wird dagegen die Umgebung verändert wahrgenommen (z.B. die Position einer Wand ändert sich), die Karte würde an dieser Stelle verändert und damit das Modell insgesamt inkonsistent.

Andererseits kann die Erwartung, dass die Umgebung an der selben Position bei nachfolgenden Beobachtungen ähnlich aussehen sollte, dazu genutzt werden, Positionsfehler zu detektieren und zu korrigieren. Im hier vorgestellten Ansatz wird dazu bei der Kartierung neben der globalen eine lokale Karte aufgebaut: die lokale Karte wird jeweils aus einer festen Anzahl der zuletzt aufgenommenen Umgebungsbeobachtungen erzeugt und beschreibt damit die aktuelle direkte Umgebung des Roboters. Dieser Ansatz war gemeinsam mit den zuvor vorgestellten Odometriekorrektur-Verfahren Gegenstand der Veröffentlichungen [Schröter et al., 2004] und [Schröter, 2004].

Die globale Karte wird nur mit den Beobachtungen aktualisiert, die nicht mehr in die lokale Karte eingehen, d.h. lokale und globale Karte bestehen aus disjunkten Mengen an Beobachtungen (Abb. 4.1). Die Integration mehrerer Messungen in der lokalen Karte hat den Vorteil, dass bei relativ ungenauen und niedrig aufgelösten Sensoren eine bessere Genauigkeit gegenüber einer Einzelbeobachtung erreicht werden kann.

Da lokale und globale Karte aus unterschiedlichen Beobachtungen entstehen, besteht zwischen diesen beiden Karten erst dann ein Überlapp,

wenn ein Kreisschluss im Roboterpfad auftritt, d.h. wenn der Roboter an eine zuvor bereits besuchte Position zurückkehrt. Dies kann einerseits tatsächlich durch eine Schleife, andererseits auch durch eine Rückkehr auf dem selben Pfad geschehen. In einer statischen Umgebung sollten dann bei korrekter Position die Anordnung von Hindernissen und Freiraum in der lokalen und der globalen Karte weitestgehend übereinstimmen. Durch Vergleich der beiden Karten kann damit nicht nur ein Positionsfehler erkannt werden, sondern es lässt sich auch die beste Kreisschluss-Position bestimmen: Durch Translation und Rotation der lokalen Karte innerhalb eines Suchbereiches wird die Position gefunden, an der die größte Übereinstimmung besteht. Als Maß der Übereinstimmung kommt hier ein Match-Wert zum Einsatz, der die Anzahl der Zellen angibt, welche sowohl in der lokalen als auch der globalen Karte von Objekten belegt sind.

$$match = \sum_{cell \in map} \begin{cases} 1 & falls & (cell_{loc} > 0.5) \wedge (cell_{glob} > 0.5) \\ -1 & falls & (cell_{loc} > 0.5) \wedge (cell_{glob} < 0.5) \\ & oder & (cell_{loc} < 0.5) \wedge (cell_{glob} > 0.5) \\ 0 & & sonst \end{cases} \tag{4.1}$$

Das Verfahren entspricht weitgehend einem bereits in [Schultz et al., 1999] beschriebenen Algorithmus, allerdings mit einer vereinfachten Suchfunktion, welche lediglich die Matchwerte an festen Positionen in einem regelmäßigen Gitter auswertet. Weiterhin hat es sich im Gegensatz zu der dort veröffentlichten Match-Funktion als vorteilhaft erwiesen, eine Übereinstimmung zweier freier Zellen nicht positiv zu bewerten, da dies (wegen der deutlich zahlreicheren freien Zellen) zu einer Maximierung der reinen räumlichen Überlappung von lokaler und globaler Karte auf Kosten der Übereinstimmung der Hindernisse führte. Mit dem Ergebnis der Korrelation wird sowohl die aktuelle Position als auch die zurückliegenden Positionen innerhalb eines gewissen Zeithorizontes korrigiert. Die Korrelation und Korrektur erfolgt in regelmäßigen Abständen von einigen Metern. Das Ergebnis dieses einfachen Korrekturverfahrens ist in Abb. 4.2 zu sehen. Als Ausgangsbasis wurden hier die um den systematischen Fehler bereinigten Daten des Experimentes in Abschnitt 3.2.1 genutzt (Abb. 3.10 rechts). Es ist zu sehen, dass zwar nur noch geringe seitliche Positionsabweichungen auftreten, allerdings wurde dafür ein deutlicher Versatz in Längsrichtung des Ganges hinzugefügt. Die Ursache dafür ist die sehr selbstähnliche Struktur: bei der

Abbildung 4.2: **Links:** *Die Korrelation zwischen lokaler und globaler Karte als Funktion der Positions-Verschiebung in x- und y-Richtung (für eine übersichtlichere Darstellung wurde der Rotationswinkel φ als dritte Dimension hier nicht berücksichtigt.) Dunklere Felder stellen höhere Korrelationswerte dar. Der Punkt (0,0) entspricht der aktuellen Odometrie-Position (unter Berücksichtigung eventueller Korrekturen zu zurückliegenden Zeitpunkten), d.h. bei einem Maximum an dieser Stelle wird keine Positionskorrektur durchgeführt. Der maximale Wert wird im Beispiel bei einer Verschiebung um -0.4m in x und 0.1m in y erreicht.* **Rechts:** *Die letztendlich entstehende Karte bei kontinuierlicher Korrektur mit dem Maximum der Korrelationsfunktion. Der Ausgangspfad entspricht dem in Abb. 3.10 dargestellten. Die seitlichen Abweichungen des Ganges werden zwar weitestgehend korrigiert, allerdings tritt dafür eine unerwünschte Verschiebung in Längsrichtung auf. Diese entsteht häufig durch die hohe Selbstähnlichkeit entlang der Gangrichtung und daraus resultierende Fehlzuordnungen. Eine einmal aufgetretene fehlerhafte „Korrektur“ kann bei der Map-Match-Korrektur nur bedingt wieder berichtigt werden. Die verbleibende Krümmung der Karte entsteht durch kleine Odometriefehler bereits während der ersten Teilstrecke im Gang. Auch dieser Fehler kann durch das Korrekturverfahren nicht eliminiert werden, da bereits kartierte Bereiche stets als Referenz betrachtet und nachträglich nicht verändert werden.*

Korrelation führt eine Verschiebung längs des Ganges zu ähnlich hohen Match-Werten, das Maximum der Korrelationsfunktion ist damit in dieser Längsrichtung nur schwach ausgeprägt und kann sich durch geringe (teils zufällige) Änderungen stark verschieben, so dass es häufig nicht die tatsächlich beste Position beschreibt. Nur an wenigen Stellen lässt die Struktur eine eindeutige Positionierung in der Längsrichtung zu, allerdings kann eine einmal falsch „korrigierte“ Position aufgrund des begrenzten Einflusses späterer Berichtigungen schon zu dauerhaften Störungen führen.

Eine Verbesserung des Korrekturverfahrens stellt daher die Elimination schwacher Maxima dar. In vielen Umgebungen besteht wie im Beispiel

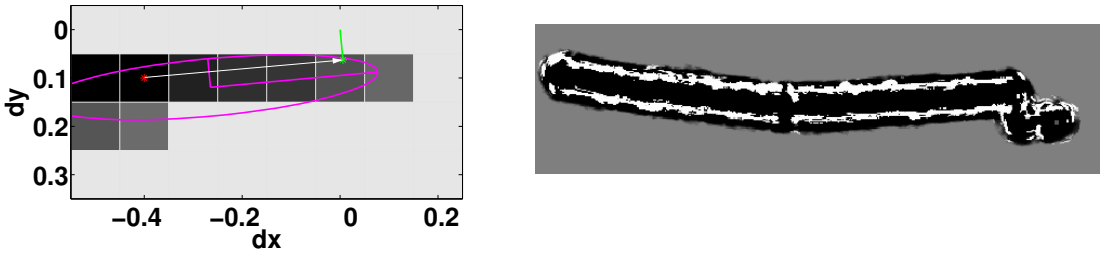

Abbildung 4.3: **Links:** *Eine Verbesserung gegenüber der Korrektur direkt mit dem Maximum der Korrelation wird durch die Analyse der Kovarianz der Korrelation erreicht. Zum Zweck der besseren Erkennbarkeit ist für das Korrelationsgebirge aus Abb. 4.2 nur der Bereich mit den höchsten Werten dargestellt. Die magentafarbene Ellipse stellt die Kovarianzmatrix der Korrelationsfunktion dar (es werden allerdings nur Korrelations-Werte berücksichtigt, die nicht kleiner als 70% des Maximums sind). Die langgestreckte Form der Ellipse macht die relativ hohe Unsicherheit des Maximums entlang dieser Hauptrichtung deutlich. Ebenfalls in magenta sind die beiden Eigenvektoren eingezeichnet. Indem die Korrektur der Position hier nur entlang des kürzeren Eigenvektors erfolgt, wird eine potentielle Fehlzuordnung in der unsicheren Längsrichtung vermieden.* **Rechts:** *Die auf diese Weise aufgebaute Karte zeigt nicht die Längsverschiebungen wie in Abb. 4.2. Die seitlichen Abweichungen werden zum Großteil korrigiert, ohne die korrekten Messungen in der Längsrichtung zu verfälschen. Die geschätzte Endposition entspricht korrekt der Startposition.*

eine hohe Selbstähnlichkeit in einer Richtung, während in anderen Richtungen die Position sehr genau bestimmt werden kann. Die Anzahl der unabhängigen Verschiebungsrichtungen entspricht der Dimensionalität des Suchraumes ($x, y, \varphi = 3$ Dimensionen), wobei die Richtungen nicht parallel zu den Koordinatenachsen liegen müssen (ein Gang kann selbstverständlich schräg im Koordinatensystem verlaufen). Abb. 4.3 (links) zeigt die Bestimmung der Verschiebungsachsen und der zugehörigen Unsicherheiten (wegen der einfacheren Darstellung nur für eine Suche im (x,y)-Raum): Zunächst wird die Kovarianzmatrix über die Werte der Korrelationsfunktion berechnet, wobei nur Werte nahe dem Maximum Berücksichtigung finden. Je schwächer das Maximum ausgeprägt ist, d.h. je mehr ähnlich hohe Werte existieren und je größer deren räumliche Entfernung zum Maximum, umso höher sind die Werte der Kovarianzmatrix. Durch Bestimmung der Eigenvektoren und zugehörigen Eigenwerte der Kovarianzmatrix lässt sich diese Unsicherheit des Maximums nach den einzelnen Richtungen aufschlüsseln. Ein hoher Eigenwert bedeutet, dass das Maximum in Richtung des entsprechenden Eigenvektors nur schwach bestimmt ist und daher keine Positionskorrektur in

Eingaben

1 $M(dx, dy, d\varphi)$ // *Matchwert als Funktion von dx, dy, dφ*

Algorithmus

2 $[dx_{max}, dy_{max}, d\varphi_{max}] = argmax(M(dx, dy, d\varphi))$ // *Punkt des max. Matchings*

3 `über alle` dx, dy, $d\varphi$ `mit` $M(dx, dy, d\varphi) > 0.7 \cdot M(dx_{max}, dy_{max}, d\varphi_{max})$:

4 $\mu = \sum([dx, dy, d\varphi] \cdot M(dx, dy, d\varphi)) \cdot (\sum M(dx, dy, d\varphi))^{-1}$ // *gewichteter Mittelwert*

5 $C = Cov([dx, dy, d\varphi] \cdot M(dx, dy, d\varphi))$ // *gewichtete Kovarianz*

6 $[EV, EW] = SVD(C)$ // *Eigenvektoren&-werte der Kovarianzmatrix durch SVD-Zerlegung*

7 $T = [EV1|EV2|EV3]^T$ // *Transformationsmatrix = transponierte Eigenvektoren*

8 $[dx'_{max}, dy'_{max}, d\varphi'_{max}] = T \cdot [dx_{max}, dy_{max}, d\varphi_{max}]$ // *Transformation in Eigenvektor-Raum*

9 `Wenn` $EW0 < threshold$ `dann`

10 $dx'_{max} = 0$

11 `Wenn` $EW1 < threshold$ `dann`

12 $dy'_{max} = 0$

13 `Wenn` $EW2 < threshold$ `dann`

14 $d\varphi'_{max} = 0$

15 $[dx_{opt}, dy_{opt}, d\varphi_{opt}] = T^{-1} \cdot [dx'_{max}, dy'_{max}, d\varphi'_{max}]$ // *Rück-Transformation*

Rückgabe

16 $[dx_{opt}, dy_{opt}, d\varphi_{opt}]$ // *optimaler Verschiebungsvektor*

Abbildung 4.4: *Pseudocode für die Bestimmung des optimalen Verschiebungsvektors unter Berücksichtigung der Bestimmtheit des Maximums in unterschiedlichen Richtungen.*

dieser Richtung erfolgen sollte. Die Unterscheidung erfolgt durch einen Schwellwert für den Eigenwert.

Da die Eigenvektoren selbst ein Basissystem im Koordinatenraum aufspannen, kann diese Unterdrückung einer bestimmten Richtung dadurch umgesetzt werden, dass die Korrekturposition (also die Maximumposition der Korrelation) in das Eigenvektor-System transformiert wird, dort die Komponente(n), welche *nicht* berücksichtigt werden soll(en), auf 0 gesetzt wird und dann der Punkt durch inverse Transformation wieder ins Ursprungs-System abgebildet wird (Psedocode 4.4). Die (3x3)-Transformationsmatrix ergibt sich direkt durch Aneinanderreihung der drei (3x1)-Eigenvektoren. Abb. 4.3 zeigt das Ergebnis dieser Prozedur als grünen Punkt. Man sieht, dass dieser neue Korrekturwert gegenüber der neutralen (0,0)-Position nur entlang der kürzeren Halbachse der

Kovarianzellipse (entspricht dem Eigenvektor mit geringem Eigenwert) verschoben ist.

In Abb. 4.3 (rechts) ist zu sehen, dass diese Erweiterung des Korrekturverfahrens eine deutliche Verbesserung gegenüber der einfachen Nutzung des Korrelationsmaximums bewirkt (vgl. Abb. 4.2). Allerdings ist auch hier die Qualität der Karte beschränkt, was u.a. daran liegt, dass zwar die beste aktuelle Position bestimmt wird, für die Anpassung des zurückliegenden Roboterpfades aber nur eine einfache Heuristik angewendet wird. Das Verfahren führt für einfache Szenarien zu brauchbaren Karten, für eine allgemeine Anwendung bleiben aber viele Fragen und Probleme offen, u.a.:

- In welchem Suchbereich und mit welcher Suchauflösung soll die Korrelationsfunktion berechnet werden?
- Wie soll die Positionskorrektur den zurückliegenden Pfad anpassen?
- Wie werden Mehrdeutigkeiten (z.B. mehrere getrennte Maxima in der Korrelationsfunktion) berücksichtigt?
- Fehlzuordnungen können nachträglich nicht wieder korrigiert werden.

4.2 Probabilistic SLAM

Die am Ende des vorherigen Abschnittes aufgezählten offenen Probleme des Map-Matching-Verfahrens zur Positionskorrektur ergeben sich zum größten Teil aus der unzureichenden Berücksichtigung und Repräsentation der Unsicherheit: Die Roboterbewegung wird als deterministischer Prozess modelliert, es wird jeweils nur eine feste Position geschätzt. In Wirklichkeit führt die Unsicherheit der Roboterbewegung ebenso wie der Korrelation dazu, dass die Position des Roboters einer Wahrscheinlichkeitsfunktion entspricht, welche sich durch geeignete Methoden auch approximieren lässt. Durch Betrachtung der Positionsunsicherheit ließe sich dann z.B. ableiten, in welchem Bereich die Suche nach einer guten Korrelation zwischen lokaler und globaler Karte sinnvoll ist. Umgekehrt könnten Mehrdeutigkeiten in der Korrelation wiederum in der neu geschätzten Verteilung berücksichtigt werden. Wenn

statt einer festen Position stets eine Verteilung über alle möglichen Positionen erhalten bliebe, wäre es einerseits nicht notwendig, eine feste Korrekturposition auszuwählen (und dabei alle anderen Hypothesen zu verwerfen), andererseits hätte eine einmalige Fehlschätzung der Korrelation (z.B. wegen Störungen der lokalen Karte durch unerwartete dynamische Hindernisse) weniger Auswirkung auf die Gesamtverteilung. Beides bewirkt, dass solche Fehler nicht sofort und unumkehrbar zum Verlust der Positionsschätzung führen können. Auch das Problem der Korrektur des zurückliegenden Pfades kann gelöst werden, indem die gesamte Positionshistorie in die Schätzung der Wahrscheinlichkeitsverteilung einbezogen wird. (Dies wäre auch in der deterministischen Variante möglich, würde aber zu einem extremen Suchaufwand führen, da für jede Position des Pfades eine Menge von Positionsmöglichkeiten abgesucht und jeweils die sich daraus ergebende Karte neu berechnet werden müsste.)

Die Modellierung der Unsicherheit und die Berechnung der entsprechenden Wahrscheinlichkeitsverteilungen ist das Ziel von probabilistischen SLAM-Techniken (der Begriff SLAM wird häufig auch im engeren Sinne nur auf solche probabilistischen Methoden bezogen). Im Kapitel 3 wurde noch zu Grunde gelegt, dass zu der zeitlichen Sequenz von Beobachtungen $o_{1:t}$ die zugehörigen Positionen $x_{1:t}$ bekannt sind. Die Schätzung der Karte m (konkret als Zerlegung in die unabhängigen Zellbelegungen occ_{cell}) bedeutete damit die Suche nach der bedingten Wahrscheinlichkeitsdichte $p(m|o_{1:t}, x_{1:t})$ über den Raum der möglichen Karten m. Beim SLAM sind dagegen die tatsächlichen Positionen unbekannt und müssen somit ebenfalls geschätzt werden. Es wird allerdings normalerweise davon ausgegangen, dass zumindest die fehlerbehafteten Odometrie-Messungen (manchmal alternativ auch Bewegungskommandos) $u_{1:t}$ bekannt sind. Gesucht ist also die Verteilung $p(m, x_{1:t}|o_{1:t}, u_{1:t})$. Die Variable m steht hierbei zunächst für eine abstrakte, beliebig-dimensionale Karten-Variable, deren Behandlung und Interpretation von der konkret verwendeten Repräsentation abhängt.

Die bedingte Verteilung $p(m, x_{1:t}|o_{1:t}, u_{1:t})$ kann unter Nutzung des Bayes-Theorems und einiger Unabhängigkeitsannahmen in eine rekursive Form überführt werden (siehe Anhang B, [Thrun et al., 2005]), mit deren Hilfe aus einer Initialverteilung $p_{init}(m, x)$ mit jeder Bewegung u und jeder Beobachtung o die Schätzung aktualisiert werden kann.

Zunächst wird dazu ein Beobachtungsupdate (observation update) betrachtet:

$$\begin{aligned} p(m, x_{1:t}|o_{1:t}, u_{1:t}) &= p(m, x_{1:t}|o_t, o_{1:t-1}, u_{1:t}) \\ &= \frac{p(o_t|m, x_{1:t}, o_{1:t-1}, u_{1:t}) \cdot p(m, x_{1:t}|o_{1:t-1}, u_{1:t})}{p(o_t|o_{1:t-1}, u_{1:t})} \\ &= \frac{p(o_t|m, x_t) \cdot p(m, x_{1:t}|o_{1:t-1}, u_{1:t})}{p(o_t)} \end{aligned} \tag{4.2}$$

Für das Bewegungsupdate (motion update) lässt sich folgende Umformung finden:

$$\begin{aligned} p(m, x_{1:t}|o_{1:t-1}, u_{1:t}) &= p(m, x_t, x_{1:t-1}|o_{1:t-1}, u_{1:t}) \\ &= p(x_t|m, x_{1:t-1}, o_{1:t-1}, u_{1:t}) \cdot p(m, x_{1:t-1}|o_{1:t-1}, u_{1:t}) \\ &= p(x_t|x_{t-1}, u_t) \cdot p(m, x_{1:t-1}|o_{1:t-1}, u_{1:t-1}) \end{aligned} \tag{4.3}$$

Durch Einsetzen von Formel 4.3 in 4.2 ergibt sich

$$p(m, x_{1:t}|o_{1:t}, u_{1:t}) = \frac{p(o_t|m, x_t) \cdot p(x_t|x_{t-1}, u_t) \cdot p(m, x_{1:t-1}|o_{1:t-1}, u_{1:t-1})}{p(o_t)} \tag{4.4}$$

Diese Formel für die *a-posteriori*-Schätzung enthält das Beobachtungsmodell $p(o_t|m, x_t)$, das Bewegungsmodell $p(x_t|x_{t-1}, u_t)$ sowie die *a-priori*-Schätzung (welche wiederum die *a-posteriori*-Schätzung des vorangegangenen Zeitschrittes ist). Das Beobachtungsmodell beschreibt dabei die Wahrscheinlichkeit einer bestimmten Beobachtung bei gegebenem Umgebungszustand und Beobachtungs-Position, das Bewegungsmodell beschreibt die Wahrscheinlichkeit einer erreichten Zielposition bei Ausführung eines Bewegungsschrittes von einer gegebenen Startposition. Beobachtungsmodell und Bewegungsmodell müssen jeweils für einen konkreten Sensor/Antrieb bestimmt werden. Der Nenner $p(o_t)$ ist von x und m unabhängig und stellt daher lediglich einen konstanten Normierungsterm für die Verteilung dar, seine Größe kann auch durch die Wahrscheinlichkeitsbedingung $\int p(m, x_{1:t}) = 1$ bestimmt werden, normalerweise ist dies jedoch gar nicht notwendig.

Häufig wird statt der kompletten Positionssequenz $x_{1:t}$ auch nur die jeweils aktuelle Position x_t betrachtet, die Formel 4.4 wird dann zu

$$p(m, x_t|o_{1:t}, u_{1:t}) = \frac{p(o_t|m, x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot p(m, x_{t-1}|o_{1:t-1}, u_{1:t-1}) dx_{t-1}}{p(o_t)} \quad (4.5)$$

Gleichung 4.4 (bzw. 4.5) stellt zunächst eine recht abstrakte rekursive Lösung für das SLAM-Problem als rekursive Bayessche Zustandsschätzung dar. Diese lässt sich allerdings nur in Ausnahmefällen geschlossen berechnen (z.B. wenn alle vorkommenden Wahrscheinlichkeiten durch diskrete Verteilungen mit abzählbar endlichen Anzahlen möglicher Zustände beschrieben werden können [Murphy, 1999]). Im Allgemeinen werden statt dessen verschiedene stochastische Zustandsschätzer zur Approximation der Wahrscheinlichkeitsverteilung genutzt, die wichtigsten Vertreter sind Kalmanfilter (KF) [Kalman, 1960] und Partikelfilter (PF). Eine einzelne Original-Quelle lässt sich für den Partikelfilter kaum anführen, Anhang B.3 enthält Verweise zu einigen relevanten methodischen Grundlagen.

Die Eignung eines spezifischen Filters für eine Anwendung ist eng mit der gewählten Art der Umgebungs-Repräsentation verknüpft. Abbildung 4.5 gibt einen Überblick über einige wichtige methodische Ansätze für die Schätzung der oben genannten Wahrscheinlichkeitsverteilung und stellt diese jeweils als Kombination der Art der Beobachtung, des Zustandsschätzers sowie der Umgebungs-Repräsentation dar.

In vielen SLAM-Anwendungen wird die Umgebung als eine Menge von punktförmigen Objektpositionen betrachtet. Diese Punkte entsprechen Landmarken (feste, wiedererkennbare Objekte oder Markierungen) in der Umgebung. Die Landmarken werden durch entsprechende Feature-Detektoren im Sensorinput detektiert. Zusätzlich muss eine Zuordnung zwischen den aktuell beobachteten Landmarken und den Modell-Objekten in der Karte erfolgen. Landmarken-basierte Repräsentationen haben den Vorteil einer relativ hohen Kompaktheit: Die gesamte Karte lässt sich durch wenige Parameter je Landmarke (Position, evtl. Eigenschaften) beschreiben, die Gesamtgröße der Karte hängt damit von der Dichte der Landmarken ab. Sie sind allerdings auf a-priori-Wissen über die Umgebung angewiesen, da zunächst sinnvolle Landmarken definiert werden müssen, um Detektoren dafür einzusetzen zu können. Geeignete Landmarken müssen einerseits robust im Sensorinput detektiert werden, andererseits sollten verschiedene Landmarken voneinan-

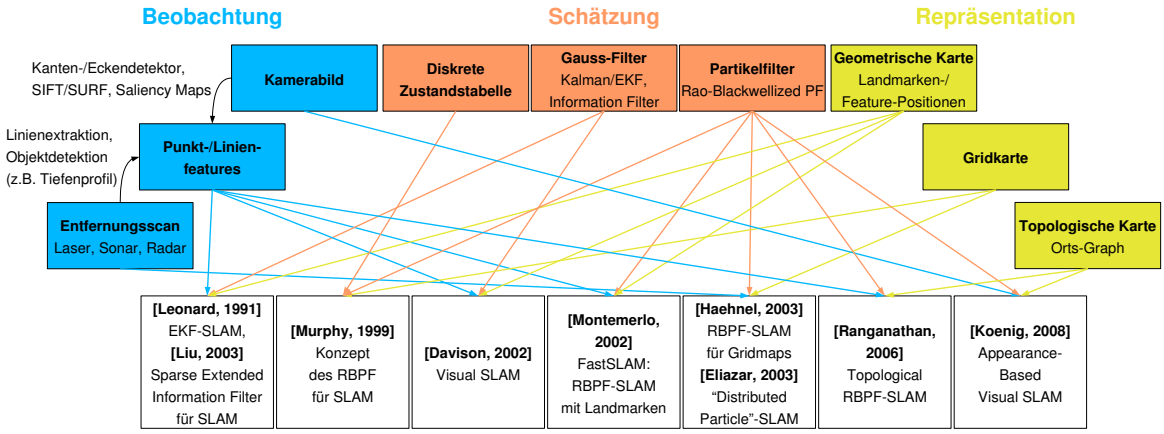

Abbildung 4.5: *Aufgeschlüsselt nach der Art der verwendeten Beobachtungen, dem Zustandsschätzer und der Repräsentation der erzeugten Karte sind einige wesentliche Verfahren dargestellt, die jeweils methodische Neuheiten auf dem Gebiet des probabilistischen Online-SLAM darstellten.*

der unterscheidbar sein, um eine Zuordnung von Beobachtung und Modell zu ermöglichen (u.U. kann die Zuordnung auch allein anhand der räumlichen Lage erfolgen).

Falls eine handhabbare Anzahl von Landmarken genutzt wird, kann der gesuchte SLAM-Zustand durch einen Vektor aller Landmarken-Parameter sowie der Roboterposition beschrieben werden. In diesem Fall eignet sich ein Kalman-Filter für die rekursive Schätzung dieses Zustandes. Der Kalman-Filter ist ein optimaler Schätzer für lineare und (als Erweiterter Kalmanfilter (EKF)[Sorenson, 1970]) nichtlineare Prozesse, allerdings nur für normalverteilte (Gauss-förmige) Wahrscheinlichkeitsverteilungen. Dies bedeutet auch, dass die Schätzung auf eine uni-modale Verteilung beschränkt ist, d.h. es ist nicht möglich, mehrere unterschiedliche Zustands-Hypothesen zu verfolgen. Der Kalman-Filter aktualisiert rekursiv den Mittelwert und die Kovarianzmatrix der geschätzten Gaussverteilung (Abb. 4.6 links). Die Anzahl der Parameter ist daher in $O(N^2)$ zur Größe des Zustandsvektors, dies ist einer der wesentlichen Gründe, weshalb der Kalman-Filter in seiner Standard-Implementierung nur für eine beschränkte Anzahl von Landmarken geeignet ist. Ein eng verwandter Ansatz ist der Information-Filter: Der Information-Filter schätzt ebenfalls wie der Kalman-Filter eine Normalverteilung für den Zustand, modelliert diese aber mittels der Information-Matrix, welche die inverse Kovarianzmatrix darstellt. Während beim Kalman-Filter durch die dem SLAM-Problem inhärente Abhängigkeit zwischen Unsicherheit der Eigenposition und

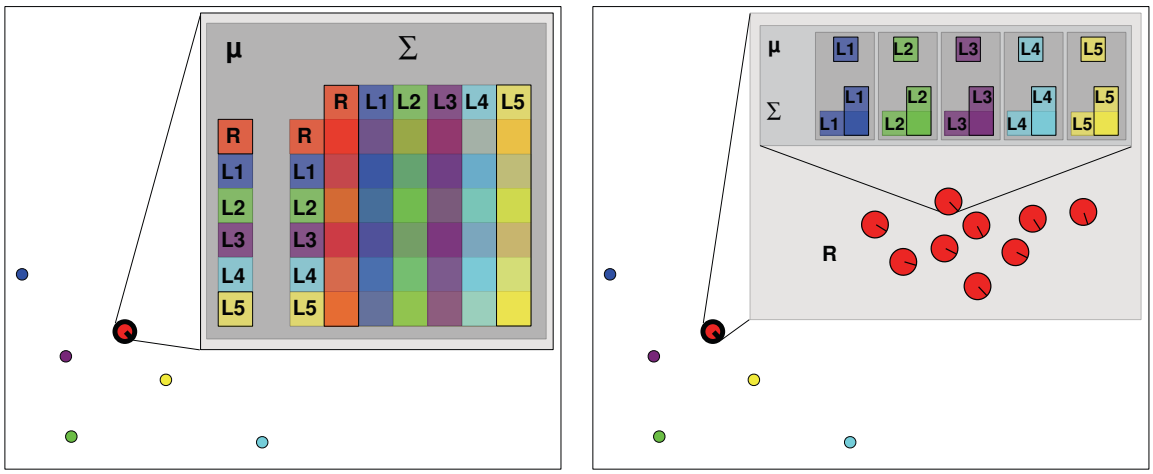

Abbildung 4.6: **Links:** *KF-SLAM mit Landmarken: Ein Kalmanfilter schätzt Mittelwertvektor und Kovarianzmatrix des SLAM-Zustandes, welcher aus der Position des Roboters und allen Landmarken-Positionen besteht. Die Anzahl der Parameter ist $O(N^2)$ (bei einer Gesamtzahl von N Landmarken).* **Rechts:** *RBPF-SLAM mit Landmarken: Ein Partikelfilter schätzt die Roboterposition. Jedes Partikel enthält je Landmarke einen Kalmanfilter. Die Anzahl der Parameter ist $O(P \cdot N)$ (N Landmarken, P Partikel). Da n potentiell unbeschränkt ist, gilt normalerweise $PN \ll N^2$.*

der Landmarken-Positionen alle Elemente der Kovarianzmatrix wichtige Information tragen, ist es durch die duale Transformation beim Information-Filter möglich, den Großteil der Information in wenigen Elementen der Informations-Matrix zu konzentrieren [Thrun et al., 2002], [Liu und Thrun, 2003]. Durch Nullsetzen der restlichen Elemente und eine effiziente Speicherung der so entstehenden spärlichen Matrix kann sowohl der benötigte Speicherplatz als auch die Rechenzeit-Komplexität deutlich verringert werden.

Für Gridkarten, wie sie in dieser Arbeit verwendet werden, sind Kalman-Filter und ähnliche Ansätze aufgrund des meist nochmals deutlich größeren Zustandsvektors nicht geeignet. Während beim KF für jede Landmarke 2 Koordinaten im Raum (teilweise auch mehr je nach Modellierung) geschätzt werden müssen, stellt in einer Gridkarte jede einzelne Zelle eine Zustandsvariable dar. Der Vorteil ist allerdings, dass Gridkarten keine Annahmen benötigen, welche Art von Objekten in der Umgebung anzutreffen sind. Sie können beliebige Umgebungen mit nahezu beliebiger Auflösung (zu Lasten von Speicherplatz und Rechenzeit) darstellen.

Ein einfacher Partikelfilter ist ebenfalls ungeeignet für die Zustandsschätzung in diesem sehr hochdimensionalen Raum. Allerdings existiert mit dem vom PF abgeleiteten Rao-Blackwellized Particle Filter (RBPF) [Murphy, 1999] ein effizientes Werkzeug zur Lösung dieses Problems. Das Grundprinzip des RBPF besteht in der Aufspaltung des hochdimensionalen Zustands X in einen niedrigdimensionalen Teilzustand A, sowie viele Teilzustände $B_1, \ldots B_n$, in welchen die Verteilung jeweils abhängig von A, aber bei Kenntnis von A untereinander unabhängig sind. Am Beispiel von FastSLAM [Montemerlo et al., 2002], [Montemerlo et al., 2003], welches wie die meisten KF-basierten Verfahren ebenfalls mit Landmarken arbeitet, wird die Funktionsweise deutlich (Abb. 4.6 rechts): Während im Kalmanfilter die Unsicherheiten aller Landmarken-Positionen durch die Unsicherheit der Beobachtungsposition verkoppelt sind, können die einzelnen Landmarken unabhängig voneinander geschätzt werden, wenn eine Roboterposition bekannt oder als gegeben angenommen wird. Dies geschieht bei FastSLAM durch die Nutzung eines Partikelfilters für die Schätzung der Roboterposition. Der Partikelfilter erzeugt eine Menge von (in sich deterministischen) Positions-Hypothesen. Unter Annahme einer solchen Hypothese sind die Unsicherheiten der Landmarken entkoppelt, und sie können unabhängig voneinander geschätzt werden. Zu diesem Zweck werden bei FastSLAM wiederum Kalmanfilter eingesetzt, wobei nun in jedem Partikel je ein Kalmanfilter je Landmarke eingesetzt werden kann. Die Größe der Kovarianzmatrix wird damit nicht mehr von der Anzahl der Landmarken bestimmt, die Gesamt-Anzahl der benötigten Kalmanfilter ist linear in der Anzahl der Landmarken sowie der Partikel. Der übergeordnete Partikelfilter nutzt zur Anpassung seiner eigenen Verteilung im Beobachtungsupdate jeweils das jedem Partikel zugeordnete spezifische Umgebungsmodell, welches sich aus der Gesamtheit der Kalmanfilter in diesem Partikel ergibt. Da die Partikelanzahl im Vergleich zur Zahl der Landmarken deutlich leichter beschränkt werden kann, ist die Anzahl der Parameter gegenüber einem normalen Kalman-Filter (in den meisten Umgebungen) wesentlich geringer.

In [Montemerlo et al., 2002] wurde FastSLAM mit einem Laser-Range-Scanner zur Detektion von Landmarken (Tiefensprünge im Scanprofil) genutzt. Auf diese Weise wurden jeweils Richtung und Entfernung der Landmarke relativ zum Roboter beobachtet. Für eine reine Richtungsbeobachtung, wie sie z.B. beim Einsatz einer einzelnen Kamera (keine Tiefeninformation) entsteht ist, erscheinen Kalman-Filter zumindest für

die initiale Schätzung der Landmarken-Position weniger geeignet (dieses spezielle Problem wird auch als Bearing-only SLAM bezeichnet). Anhang E beschreibt eine Erweiterung von FastSLAM für einen visuellen Landmarken-Detektor, welche auch für die inneren Landmarkenschätzer Partikelfilter nutzt.

Das RBPF-Prinzip wurde erstmals in [Hähnel et al., 2003] auf Gridkarten übertragen: Auch hier wird ein Partikelfilter zur Modellierung der Verteilung der Roboterposition eingesetzt, allerdings wird nun innerhalb jedes Partikels aus den Beobachtungen eine Gridkarte erzeugt. Die Anzahl der genutzten Partikel hat auch hier linearen Einfluss sowohl auf den Speicherbedarf als auch auf die Rechenzeit. Durch den Einsatz von Scan Matching auf den verwendeten Laser-Beobachtungen konnte die Positionsunsicherheit und damit die Anzahl der benötigten Partikel stark verringert werden. Eine wichtige Weiterentwicklung stellt insbesondere das „inverse“ Sampling dar, welches in [Grisetti et al., 2005] vorgestellt wurde: Statt wie üblich die Vorschlagsverteilung aus dem Bewegungsmodell zu generieren und die entstehenden Hypothesen mittels des Beobachtungsmodells zu bewerten, wird hier umgekehrt vorgegangen, die Vorschlagsverteilung wird aus dem Beobachtungsmodell erzeugt. Bei der Verwendung von Laser-Entfernungs-Scannern entstehen auf diese Weise sehr schmale Vorschlagsverteilungen, dadurch werden relativ wenige Partikel benötigt (Abb. 4.7 rechts).

Eine weitere wichtige Entwicklung ist das so genannte Distributed Particles (DP)-SLAM [Eliazar und Parr, 2003], [Eliazar und Parr, 2004]: Die Autoren zeigten, dass zwischen den Gridkarten der einzelnen Partikel eine hohe Redundanz besteht. Diese entsteht, indem im Resampling-Schritt des Partikelfilters einzelne Partikel mitsamt ihrer kompletten Karte kopiert werden. Nachfolgende Beobachtungen verändern jedoch jeweils nur einen kleinen Bereich der Karte, so dass zum großen Teil identische Kopien in den Partikeln existieren. Durch die Protokollierung nur der Änderungen jeder Zelle für die einzelnen Partikel konnte die Speicherkomplexität und der Kopieraufwand deutlich verringert werden.

Map-Match-SLAM

Nach Gl. 4.4 wird das Beobachtungsmodell $p(o_t|m, x_t)$ zur Lösung des SLAM-Problems benötigt. Dies ist allerdings für viele Sensoren nur

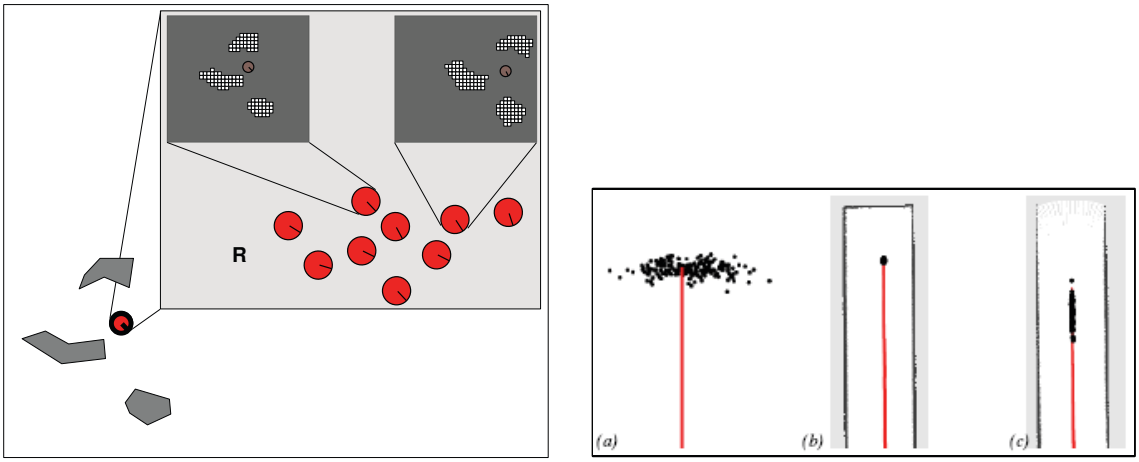

Abbildung 4.7: **Links:** *RBPF-SLAM mit Gridkarten: Jedes Partikel baut seine eigene, unabhängige Gridkarte. Zur Partikel-Gewichtung im Beobachtungs-Update nutzt jedes Partikel jeweils die eigene Karte.* **Rechts:** *Durch die Erzeugung der Partikel-Vorschlagsverteilung (proposal distribution) aus dem Beobachtungsmodell statt aus dem Bewegungsmodell entsteht in vielen Situationen eine deutlich kompaktere Verteilung, dadurch müssen weniger Zustandshypothesen (Partikel) angelegt und bewertet werden. Quelle: [Grisetti et al., 2005]*

schwierig zu bestimmen: Während es für die Berechnung einer Umgebungskarte aus den Messungen meist ausreicht, die wahrscheinlichste Lage der beobachteten Objekte zu schätzen, wird für das vollständige Beobachtungsmodell die komplette Wahrscheinlichkeitsverteilung der Beobachtungswerte zu einer gegebenen Umgebung gesucht. Gerade für Sensoren, welche große Messvarianzen aufweisen und von vielen Parametern (z.B. Materialeigenschaften, Beobachtungswinkel, etc.) abhängen, ist eine zuverlässige Bestimmung dieser Verteilung sehr aufwändig. Dies trifft u.a. auf Sonar-Sensoren zu, welche gegenüber Laser-Entfernungsmessern nicht nur eine typischerweise schlechtere Auflösung, sondern auch deutlich größere Varianz der Messwerte und stärkere Abhängigkeit von den genannten Umgebungs- und Anordnungsparametern aufweisen.

Des weiteren ist die Bestimmung eines solchen Beobachtungs-Modells nicht für alle Sensoren in gleicher Weise möglich. Für die in Abschnitt 3.1.3 vorgestellten 3D-Punkt-Schätzer, welche aus Kamera-Bildern bestimmte Featurepunkte detektieren und deren Lage im Raum bestimmen, ist zum Beispiel folgendes Problem nachvollziehbar: Die Verfahren nutzen meist recht einfach definierte Punktfeatures (z.B. Eckendetektoren [Harris und Stephens, 1988], [Shi und Tomasi, 1994] etc.),

welche zwar beim kontinuierlichen Tracking über mehrere Bilder zugeordnet werden können, allerdings bei einer späteren Rückkehr nicht individuell wiedererkannt werden. Außerdem wird aus Laufzeitgründen meist die Anzahl der getrackten Punkte beschränkt und dazu eine Auswahl aller detektierten potentiellen Features getroffen. Für Objekte, welche eine Vielzahl detektierbarer Punkte auf ihrer Oberfläche aufweisen (insbesondere Wände), ist dann nicht sichergestellt, dass stets die selben Punkte ausgewählt werden. Eine vermeintlich fehlerhafte Messung könnte dann z.B. auch dadurch entstehen, dass in verschiedenen Beobachtungen nicht die selben, sondern unterschiedliche Punkte auf dem selben Objekt detektiert wurden. Die gesamte Punktwolke sollte hingegen in beiden Fällen eine ähnliche Form und Position aufweisen, welche bei der Darstellung in Form einer Karte gut erkennbar ist.

Aus diesen Gründen wird für das hier vorgestellte RBPF-Grid-SLAM-Verfahren auch die bereits im Abschnitt 4.1 vorgestellte Map-Matching-Methode zur Bewertung der Zustandshypothesen benutzt: Die Beobachtungen werden zu einer lokalen Karte verrechnet, die lokale und globale Karte werden miteinander verglichen. Das SLAM-Verfahren soll dadurch besser für Sensoren mit hoher Varianz nutzbar und flexibler auf unterschiedliche Sensor-Typen übertragbar sein. Das Verfahren wurde sowohl für den Einsatz von Sonar-Entfernungsmessern als auch von 3D-Punkten aus Kamerabildern implementiert und getestet.

Da keine hochaufgelösten Sensoren wie Laser-Scanner vorausgesetzt werden, wird auf Scan Matching oder ähnliche Korrekturverfahren (abgesehen von den in Kapitel 3 vorgestellten Odometrie-Korrekturen) verzichtet und lediglich die Odometriedaten zur Generierung der Positionshypothesen genutzt. Dies führt zu einer relativ hohen potentiellen Positionsunsicherheit und damit einer großen Anzahl benötigter Partikel. Um den Speicherbedarf zu beschränken, wird eine gemeinsame Gridkarten-Repräsentation als Alternative zum DP-SLAM-Ansatz verwendet, welche Änderungen nicht für jede einzelne Zelle, sondern für Kartenbereiche konfigurierbarer Größe betrachtet und einfacher zu implementieren ist. Weiterhin wird eine Dynamisierung der Partikelanzahl in das RBPF-Framework integriert, wodurch die Partikelanzahl jeweils optimal an die tatsächlich vorhandene Unsicherheit angepasst werden kann. Der grundlegende Ansatz und einige der hier vorgestellten Erweiterungen wurden bereits als Konferenzbeiträge publiziert in [Schröter et al., 2007a] und [Schröter und Gross, 2007].

4.3 Map-Match-SLAM mit entfernungsmessenden Sensoren

In diesem Abschnitt wird zunächst die grundlegende Struktur des SLAM-Systems, der verwendeten Datenstrukturen und algorithmischen Bestandteile beschrieben und dessen Funktionsfähigkeit an beispielhaften Experimenten nachgewiesen. In nachfolgenden Abschnitten werden Erweiterungen vorgestellt, welche im Wesentlichen die Verringerung des Speicheraufwandes, jedoch auch die Beschleunigung des Verfahrens zum Ziel haben.

4.3.1 Rao-Blackwellized Particle Filter für Map-Match-SLAM

Die Grundlage des SLAM-Algorithmus ist ein Partikelfilter, bei dem jedes Partikel eine Hypothese über den Roboterpfad und die zugehörige Karte darstellt (Abb. 4.8). Die Partikel-eigene Karte entsteht dabei durch die Kombination der Messungen des Entfernungssensors jeweils mit den geschätzten Posen jedes Partikels. Um das Map-Matching-Prinzip für die Hypothesen-Bewertung zu nutzen, ist weiterhin eine zusätzliche lokale (temporäre) Karte in jedem Partikel nötig, welche jeweils aus den aktuellsten Umgebungsbeobachtungen gebildet wird. Da die globale Karte erst dann mit diesen neueren Beobachtungen aktualisiert wird, wenn diese nicht mehr Bestandteil der lokalen Karte sind, wird eine Warteschlange für die Beobachtungen (im Partikelfilter) und die zugehörigen geschätzten Posen ($[x, y, \varphi]$, in jedem Partikel) benötigt. Der Versatz zwischen lokaler und globaler Karte wird durch einen Parameter gesteuert und orientiert sich an der Fahrstrecke des Roboters: sobald eine Beobachtung weit genug von der aktuellen Position entfernt ist, geht sie in die globale Karte ein, die lokale Karte wird jeweils in ihrem Radius beschränkt.

Eine Speicherung des gesamten Pfades, also *aller* Posen ist für die Grundfunktion des Verfahrens eigentlich nicht nötig, da eine Pose keinen Einfluss auf den Ablauf mehr hat, nachdem sie einmal bei der Aktualisierung der globalen Karte benutzt wurde. Der Speicherplatz für den Pfad fällt gegenüber den Kartendaten jedoch kaum ins Gewicht, daher wird im Partikel die Historie aller Posen (Pfad-Schätzung) gespeichert

und kann zur Visualisierung genutzt und exportiert werden. Ebenso werden alle Beobachtungen gespeichert. Da die Anzahl der gespeicherten Posen nicht zwingend mit der Anzahl der gespeicherten Beobachtungen übereinstimmen muss, muss jeder Beobachtung der Index der entsprechenden Pose zugeordnet werden. Jedes Partikel enthält zwar eine eigene, von allen anderen Partikeln verschiedene Posen-Historie. Die Bewegungsupdates erfolgen aber synchron, so dass die Pfade in allen Partikeln zu jedem Zeitpunkt die selbe Anzahl an Posen enthalten. Daher ist ein einzelner Index pro Beobachtung ausreichend, um für jedes ausgewählte Partikel die entsprechende Posen-Schätzung zum Zeitpunkt der Beobachtung zu referenzieren. Diese sind in Abb. 4.8 als Pfeile zwischen Beobachtungen und Pfad dargestellt, solche Referenzen ergeben sich allerdings erst durch Bezug der Indices auf ein konkretes ausgewähltes Partikel. Die zuvor beschriebene Warteschlange zur Trennung der lokalen und globalen Karten-Updates vereinfacht sich dann ebenfalls zu einem einfachen Index für diejenige Beobachtung, welche als nächstes für das Update der globalen Karte verwendet werden soll (Unterscheidung grün/weiß in Abb. 4.8).

Die Partikelzahl wird zunächst als konstant betrachtet, d.h. bei der Initialisierung des Partikelfilters wird eine gewisse Anzahl an Partikeln angelegt, diese Gesamtzahl bleibt bei jeder Anpassung der Verteilung erhalten.

Im der mathematischen Herleitung im vorhergehenden Abschnitt wurden Bewegungs- und Beobachtungsupdate als Einheit betrachtet. Es ist jedoch auch ohne weiteres möglich, diese voneinander zu trennen. Prinzipiell führt jede Bewegung zu einer Erhöhung der Positionsunsicherheit, jede Beobachtung wiederum kann (muss aber nicht) zur Verringerung der Positionsunsicherheit beitragen. Bewegungs- und Beobachtungsupdates können daher prinzipiell unabhängig voneinander beliebig oft durchgeführt werden. Es muss aber beachtet werden, dass bei zu seltenen Beobachtungsupdates die Unsicherheit der Zustandsschätzung stark anwachsen kann, so dass letztendlich eine hinreichend genaue Repräsentation der Wahrscheinlichkeitsverteilung z.B. aufgrund der begrenzten Partikelzahl nicht mehr möglich ist.

Das Beobachtungsupdate besteht beim RBPF wiederum aus zwei Teilen: Zum Einen wird die Beobachtung genutzt, um die Karte jedes Partikels zu aktualisieren (Kartenupdate), außerdem werden die Partikel bewertet, untereinander gewichtet und die Verteilung angepasst (Ver-

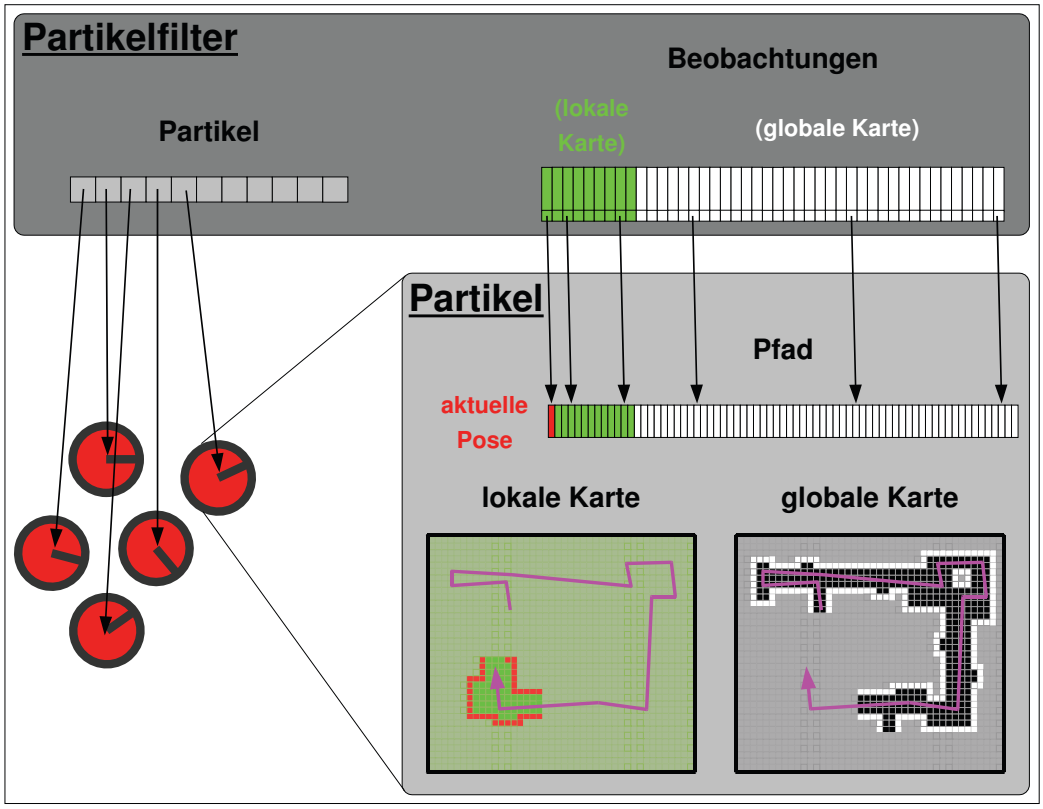

Abbildung 4.8: *Datenorganisation im Map-Match-SLAM-Partikelfilter: Der Partikelfilter verwaltet eine Menge von Partikeln, außerdem speichert er zentral eine Liste aller Beobachtungen, welche jeweils mit dem Index der entsprechenden Pose attributiert sind. Für ein konkretes Partikel ist damit die geschätzte Pose zum Zeitpunkt jeder Beobachtung bezeichnet (dies wird durch die Pfeile zwischen Beobachtungen im Partikelfilter und Pfad im Partikel dargestellt). Jedes Partikel enthält einen Pfad, welcher aus der kompletten Historie der vom Partikel geschätzten Posen besteht. Das neueste Element des Pfades ist jeweils die aktuelle Position. Weiterhin enthält jedes Partikel sowohl eine globale als auch eine lokale Karte. Die lokale Karte wird bei jeder neuen Beobachtung aktualisiert, sie ist aber räumlich auf einen festen Radius um die aktuelle Position beschränkt und dadurch gezwungen, ältere Beobachtungen zu „vergessen“. Sobald sich der Roboter soweit fortbewegt hat, dass eine Beobachtung nicht mehr Bestandteil der lokalen Karte ist, so wird die globale Karte mit dieser Beobachtung aktualisiert. Die globale Karte läuft dadurch der Roboterposition mit einem festen Versatz „nach“.*

teilungsupdate). Da beim Map-Match-SLAM nicht die Einzel-Beobachtungen direkt zur Bewertung der Hypothesen genutzt werden, werden auch Kartenupdate und Verteilungsupdate unabhängig voneinander durchgeführt. Alle drei Schritte (Bewegungsupdate, Kartenupdate, Verteilungsupdate) werden jeweils regelmäßig nach einer bestimmten,

unabhängig parametrierbaren Fahrstrecke ausgelöst, dafür wird die von der Odometrie gemessene Strecke zu Grunde gelegt. Es gelten allerdings folgende Einschränkungen für die sinnvolle Wahl der Update-Intervalle:

1 Für das Bewegungsupdate erscheint eine möglichst hohe Frequenz sinnvoll, um eine hohe Auflösung der Bewegungsmessung zu erreichen. Je größer der Abstand zwischen zwei Bewegungsupdates, umso mehr kleine Bewegungsschritte überlagern sich zu einer integrierten Gesamtbewegung, dadurch wird die Messung insgesamt ungenauer. Allerdings bedeuten mehr Updates natürlich auch mehr Rechenzeit und ebenso mehr Speicherbedarf für die Speicherung des Pfades. Für das Bewegungsupdate wird neben der Fahrstrecken-Schwelle auch eine Rotations-Schwelle benutzt, da z.B. auch dann eine Aktualisierung der Posen-Schätzung erfolgen muss, wenn der Roboter sich auf der Stelle dreht.

2 Für die Belegtheitsschätzung der Gridzellen sind vor allem Messungen von unterschiedlichen Beobachtungspositionen wichtig, daher sollte sich der Roboter zwischen den Kartenupdates eine gewisse Strecke bewegt haben. Der Abstand steht dabei in Beziehung zur Zellgröße: Je größer der Abstand zwischen den Updates und je kleiner die Zelle, um so weniger Beobachtungen werden für jede Zelle berücksichtigt, und um so ungenauer wird die Belegtheitsschätzung. Da die Kartenupdates allerdings bei großen Partikelanzahlen den Hauptanteil der gesamten Rechenzeit benötigen, bestimmt die Frequenz des Kartenupdates maßgeblich die Laufzeit des SLAM-Verfahrens.

3 Das Verteilungsupdate nutzt die lokale Karte, welche aus vielen Beobachtungen besteht. Um die einzelnen Beobachtungen bei der Wahrscheinlichkeitsschätzung nicht zu stark zu gewichten, kann und sollte das Verteilungsupdate daher deutlich seltener als das Kartenupdate erfolgen.

Aus diesen Gründen wird normalerweise folgende Beziehung eingehalten:

$$T_{Bewegungsupdate} \leq T_{Kartenupdate} \leq T_{Verteilungsupdate}$$

Die wesentlichen algorithmischen Bestandteile Bewegungsupdate, Kartenupdate und Verteilungsupdate werden im Folgenden detailliert beschrieben. Zunächst wird außerdem auf die Initialisierung des Partikelfilters eingegangen.

Initialisierung des Partikelfilters

Meist wird bei der Diskussion des SLAM-Problems davon ausgegangen, dass zu Beginn keinerlei Kenntnis über die Umgebung oder die Position vorhanden ist. In diesem Fall kann die Startposition als Referenzposition und damit als Ursprung des zu Grunde liegenden globalen Koordinatensystems betrachtet werden. Da bei Unkenntnis der Umgebung kein Bezugsrahmen existiert, zu dem relativ eine Position ausgerichtet werden muss, existiert auch keine Positionsunsicherheit: Der Roboter befindet sich in dem von ihm selbst gesetzten Referenzsystem im Koordinatenursprung. Es können daher alle Partikel an der Position $[x = 0, y = 0, \varphi = 0]$ initialisiert werden. Die lokalen und globalen Karten aller Partikel sind zu Anfang leer, d.h. für jede Zelle entspricht die Belegtheitswahrscheinlichkeit der a-priori-Schätzung (typischerweise 0.5, andere Werte sind aber möglich, um Vorwissen über die Umgebung zu berücksichtigen).

Es kann allerdings auch der Fall auftreten, dass bereits ein Teil der Umgebung bekannt ist. Das SLAM-Problem besteht dann darin, sowohl die Position relativ zu diesem Fragment zu bestimmen als auch die Umgebungskarte zu erweitern. Die Startposition ist dabei meist nur grob oder gar nicht bekannt. In diesem Fall wird jedes Partikel mit der gegebenen globalen Karte und einer leeren lokalen Karte initialisiert. Die Partikelpositionen können nach einer beliebigen Verteilung angelegt werden, z.B. als Normalverteilung um eine grobe Schätzung der Startposition oder als Gleichverteilung in einem bestimmten Bereich.

Bewegungsupdate

Das Bewegungsupdate wird jeweils dann ausgeführt, wenn der Roboter sich nach Messung der Odometrie um eine bestimmte Strecke vorwärts bewegt oder um einen bestimmten Winkel gedreht hat (typischerweise wenige Zentimeter/Grad). Im Bewegungsupdate wird die neue Wahrscheinlichkeitsverteilung für die Roboter-Position berechnet (die sich in

der Verteilung der Partikel ausdrückt), indem für jedes Partikel eine neue Position geschätzt wird. Dabei kommt das Bewegungsmodell zum Einsatz, welches in Gleichung 4.3 als $p(x_t|x_{t-1}, u_t)$ enthalten ist. Unter der Annahme, dass das Bewegungsmodell ortsinvariant, also die Bewegung unabhängig von der Position ist, lässt sich das Modell vereinfachen zu

$$p(x_t|x_{t-1}, u_t) = p(x_t - x_{t-1}|u_t) \tag{4.6}$$

Das Bewegungsmodell soll also die Wahrscheinlichkeitsverteilung für die tatsächliche Bewegung zwischen zwei Update-Schritten unter Kenntnis der Odometriemessung beschreiben. Um es in einem Algorithmus nutzen zu können, muss es durch eine konkrete Funktion beschrieben werden. Das spezifische Modell, welches hier verwendet wird, ist angelehnt an das in [Hähnel et al., 2003] vorgestellte (Abb. 4.9). Es modelliert die Unsicherheit der tatsächlichen relativen Bewegungsrichtung α, der Bewegungs-Distanz d und der Drehung des Roboters gegenüber der Fahrtrichtung β (bei Odometriemessung $(d_{odo}, \alpha_{odo}, \beta_{odo})$, Abb. 4.9).

$$d \sim d_{odo} \cdot \mathcal{N}(1.0, \sigma_d) \tag{4.7}$$
$$\alpha \sim \alpha_{odo} \cdot \mathcal{N}(1.0, \sigma_{turn}) + d_{odo} \cdot \mathcal{N}(0.0, \sigma_{d,turn}) \tag{4.8}$$
$$\beta \sim \beta_{odo} \cdot \mathcal{N}(1.0, \sigma_{turn}) + d_{odo} \cdot \mathcal{N}(0.0, \sigma_{d,turn}) \tag{4.9}$$

Das Bewegungsmodell wird durch drei Varianz-Parameter bestimmt: σ_d gibt die Unsicherheit der Längenmessung an. σ_{turn} gilt für die Bewegungsrichtung und Drehung, da in der Praxis diese beiden Bewegungen nicht nacheinander, sondern überlagert erfolgen und sich daher kaum trennen lassen. Weiterhin wird ein Bezug zwischen der Fahrstrecke und der Richtung/Drehung hergestellt: Ohne diesen Einfluss würde bei Messung einer reinen Vorwärtsfahrt keine Unsicherheit der Richtung geschätzt werden können. Typischerweise wird aber die Richtung mit steigender Fahrstrecke stärker verfälscht. Bei Annahme einer Kreisbahn als Fehler-Modell (welches für kurze Fahrstrecken eine hinreichend gute Näherung bietet) sind Bewegungsrichtung und zusätzliche Orientierungsänderung gleich stark von diesem Fehler betroffen (vgl. Abschnitt 3.2.1).

Die exakte Bestimmung der Varianz-Parameter ist recht aufwändig und (wegen des Einflusses der nicht-systematischen Odometrie-Fehler) so-

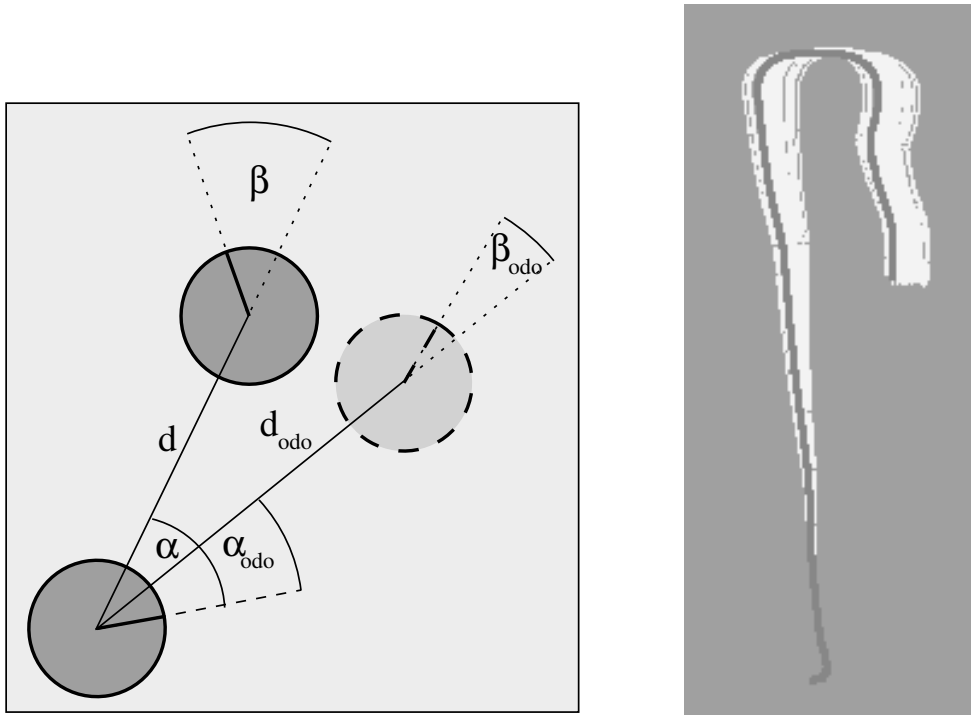

Abbildung 4.9: *Das Bewegungsmodell beschreibt die Unsicherheit der Bewegungsrichtung α, der Bewegungs-Distanz d und der (zusätzlichen) Drehung β, welche durch stochastische Einflüsse von der Odometrie falsch gemessen werden.* **Rechts:** *Auswirkung des Bewegungsmodells: Die Partikel driften auseinander und beschreiben dadurch die Unsicherheit der Positionsschätzung. Dargestellt ist jeweils der komplette Pfad eines Partikels nach einer kurzen Wegstrecke.*

wohl vom Roboter als auch von der Umgebung abhängig. Eine Möglichkeit der Onlineschätzung bestünde in der Erweiterung des Zustandes des Partikelfilters um diese drei Variablen, der approximierte Zustand würde dann auch eine Schätzung der optimalen Parameter des Modells enthalten. Allerdings würde das den Zustandsraum deutlich vergrößern und damit eine vielfach höhere Zahl an Partikeln erfordern, was kaum praktikabel wäre. Einfacher ist es, die Parameter heuristisch zu schätzen, wobei lediglich darauf geachtet werden muss, dass die Varianzen nicht zu klein gewählt werden. Eine zu große Wahl der Varianzen führt dagegen zu einer leichten Erhöhung der benötigten Partikelanzahl, was jedoch weniger starke Auswirkungen hat als eine Erweiterung des Zustandsraumes. Ein kaum endgültig lösbares Problem bei der Nutzung der reinen Odometrie ist die Möglichkeit des Auftretens extremer Störungen z.B. durch unebenen/verschmutzten Untergrund. Bei exakter Beschreibung führt dies dazu, dass eine Restwahrschein-

lichkeit für Fehler nahezu beliebiger Größe besteht (die entstehende Verteilung entspricht etwa der Summe einer Normalverteilung und einer Gleichverteilung über einen sehr breiten Bereich), allerdings ist dies mit dem Partikelfilter kaum modellierbar, weil auf diese Weise viele Partikel benötigt würden, um eigentlich sehr unwahrscheinliche (aber keinesfalls unmögliche!) Positionen abzudecken. Solche extremen Ausreißer werden daher hier nicht betrachtet.

Das Bewegungsmodell wird angewendet, indem jedes Partikel unabhängig von allen anderen Partikeln ein Sample dieser parametrischen Verteilung zieht und dieses benutzt, um seine eigene Position entsprechend der so geschätzten Bewegung $\hat{\Delta x_t^p} = (d^p, \alpha^p, \beta^p)$ zu aktualisieren. Die neue Pose wird dann als neuestes Element in den vom Partikel gehaltenen Pfad eingefügt:

$$x_{t+1}^p = x_t^p + d^p \cdot \cos(\varphi_t^p + \alpha^p) \tag{4.10}$$
$$y_{t+1}^p = y_t^p + d^p \cdot \sin(\varphi_t^p + \alpha^p) \tag{4.11}$$
$$\varphi_{t+1}^p = \varphi_t^p + \alpha^p + \beta^p \tag{4.12}$$

Kartenupdate

Das Kartenupdate wird in regelmäßigen Abständen mit einer neuen Umgebungs-Beobachtung (Sonar-Entfernungsmessung) durchgeführt.
Die Beobachtung wird zunächst von jedem Partikel dazu genutzt, die eigene lokale Karte zu aktualisieren und als neuestes Element in den Beobachtungsspeicher des Partikelfilters (ein lineares Array, bei dem über den Positions-Index direkter Zugriff auf die einzelnen Elemente möglich ist) eingefügt.

Anschließend erfolgt die Aktualisierung der globalen Karten. Der Partikelfilter speichert mittels eines Index, welche Beobachtungen bisher nicht in die globale Karte eingegangen sind. Ausgehend von diesem Index werden jene Beobachtungen gesucht, welche (basierend auf der Messung der Fahrstrecke) eine Mindestentfernung zur aktuellen Roboter-Position haben. Da zu jeder Beobachtung der Index der zugehörigen Positionsschätzung gespeichert ist, kann jedes Partikel die Beobachtung und entsprechende Partikel-spezifische Position zur Aktualisierung der globalen Karte nutzen. Gleichzeitig wird die lokale Karte auf einen begrenzten Radius um die aktuelle Position beschränkt, so dass sie die-

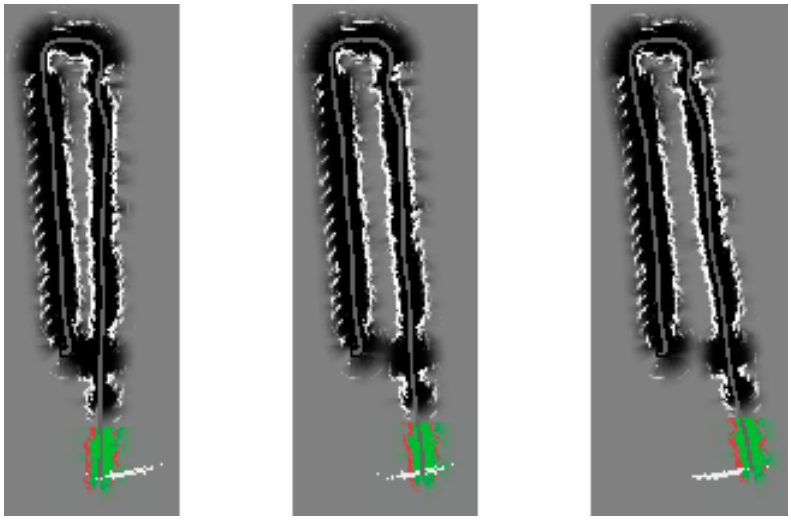

Abbildung 4.10: *Beispielhaft sind hier 3 Partikel jeweils mit ihrer globalen (schwarz/weiß) und lokalen (rot/grün) Karte gezeigt. Der geschätzte Pfad des Partikels wird jeweils durch die dunklere Linie dargestellt. Die aktuellen Positionen aller Partikel sind als helle Wolke sichtbar (auf die Abbildung aller kompletten Pfade wurde zu Gunsten der Übersichtlichkeit verzichtet).*
Hier ist auch gut zu erkennen, dass der wesentliche Anteil der lokalen Karte in Fahrtrichtung hinter *dem Roboter liegt, d.h. auch die lokale Karte bildet Objekte ab, an denen der Roboter sich schon komplett vorbei bewegt hat. Die sich dadurch ergebenden Beobachtungen aus vielen verschiedenen Richtungen sind bei niedrig aufgelösten Sensoren wie Sonar sehr wichtig, um dennoch eine gute Kartenauflösung erzielen zu können.*

se Beobachtungen nicht mehr enthält und damit keine Beobachtungen gleichzeitig für die lokale und globale Karte verwendet werden (Abb. 4.10). Durch diese räumliche Begrenzung wird vermieden, dass die lokale Karte jedes Mal aus der veränderten Menge der lokalen Beobachtungen komplett neu gebaut werden muss oder dass eine große Zahl an lokalen Karten parallel gebaut werden muss, was deutlich aufwändiger wäre.

Die Aktualisierung der lokalen und globalen Karten erfolgt analog zu den im Kapitel 3 vorgestellten Verfahren, insbesondere wird die in Abschnitt 3.1.1 beschriebene beschleunigte Implementierung mit Look-Up-Tables benutzt.

Verteilungsupdate

Im Verteilungsupdate werden die durch die Partikel repräsentierten Zustandshypothesen (Pose/Pfad + Karte) anhand der Beobachtungen be-

wertet und die Wahrscheinlichkeitsverteilung durch Adaption der Hypothesengewichte aktualisiert. Anschließend wird die Partikelverteilung an die Wahrscheinlichkeitsverteilung angepasst (Resampling), dies geschieht, wie meist üblich, indem Partikel mit niedrigem Gewicht entfernt und dafür hoch gewichtete Partikel dupliziert werden.

Das Verteilungsupdate nutzt zur Bewertung der Zustandshypothesen die Übereinstimmung der lokalen und globalen Karte an der Roboterposition. Da die globale Karte verzögert aktualisiert wird, stellt sie die „alten“ Beobachtungen (vom vorherigen „Besuch“ dieser Position) dar, während die lokale Karte die aktuellen Beobachtungen enthält. Der Vergleich ist damit zur Beurteilung des korrekten Kreisschlusses geeignet. Die Berechnung der Bewertung erfolgt analog zum bereits vorgestellten Map-Matching-Korrekturverfahren, wobei die Suche nach einer optimalen Passposition zwischen lokaler und globaler Karte aber entfällt: diese wird implizit durch die Partikelverteilung gewährleistet. Es wird für jedes Partikel nur einmal der Wert der Übereinstimmung zwischen den beiden Karten nach Gl. 4.1 berechnet. Als Partikelgewicht ist der rohe Match-Wert allerdings noch nicht optimal, hier hat sich eine abgeschwächte Exponentialfunktion als gut geeignet herausgestellt:

$$w_{match}^{p} = e^{c \cdot match^{p}} \tag{4.13}$$

c ist in diesem Fall ein Parameter, der die Stärke der Anpassung bei jedem Vergleich beeinflusst. Je seltener das Verteilungsupdate stattfindet, umso stärker sollte die jeweilige Anpassung sein, allerdings sind häufigere Updates mit kleinem c vorzuziehen, da sonst die Gefahr besteht, dass einzelne Beobachtungen zwischen zwei Updates gar nicht mehr berücksichtigt werden, wenn diese schon wieder aus der lokalen Karte entfernt wurden.

Die Gewichtsbestimmung ist keine exakte Berechnung der Beobachtungs-Wahrscheinlichkeitsdichte $p(o_t|m, x_t)$, es handelt sich aber um eine praktikable und gut funktionierende heuristische Abschätzung.

Im abschließenden Resampling werden die Partikel entsprechend ihres relativen Gewichtes neu verteilt: Partikel mit geringer Bewertung werden entfernt, dafür Partikel mit hoher Bewertung entsprechend oft dupliziert. Statt eines stochastischen Resamplings, welches bei kleinen Partikelzahlen mit hoher Wahrscheinlichkeit die Verteilung verfälscht, wird für jedes Partikel aus dem Gewicht (mittels Rundung) ein ganzzahliger

Multiplikator berechnet. Der Multiplikator $mult^p$ gibt an, wie oft ein Partikel in der neuen Verteilung enthalten ist (mit $0 \leq mult^p \leq P$). Da dabei nur ganzzahlige Multiplikatoren möglich sind, würde dies u.U. (insbesondere bei geringen Partikelzahlen) zu einer signifikanten Veränderung der Verteilung führen. Um diese Störung zu beheben, werden persistente Partikel-Gewichte w^p berechnet, welche bewirken, dass der relative Anteil jedes (nicht komplett entfernten) Partikels an der Gesamtverteilung nach dem Resampling gleich bleibt. Diese werden anfänglich mit $w_0^p = 1.0$ initialisiert und während des Resampling rekursiv aktualisiert:

$$w_t^p = \frac{w_{match}^p \cdot w_{t-1}^p}{\sum_{p=1}^{P} w_{match}^p} \cdot \frac{P}{mult^p} \tag{4.14}$$

Im Idealfall entspricht das Verhältnis von Multiplikator $mult^p$ zur Gesamt-Partikelzahl genau dem Verhältnis zwischen der aktuellen Partikelbewertung und der Summe der Bewertungen aller Partikel. In diesem Fall bleibt das Partikelgewicht genau 1.0 (bzw. der vorherige Wert). Wenn der Anteil des Partikels an der Gesamt-Bewertung durch einen ganzzahligen Multiplikator nicht exakt dargestellt werden kann, wird durch die Diskrepanz das neue Gewicht größer bzw. kleiner als 1.0, dadurch wird der exakte gebrochene Multiplikator in den nächsten Zeitschritt übertragen. Lediglich Partikel, deren Multiplikator auf 0 gerundet wird (die also aus dem Partikelset herausfallen), können nicht angepasst werden und führen weiterhin zu einer leichten Veränderung der Verteilung.

4.3.2 Gemeinsame Kartenrepräsentation (Shared Gridmaps)

Ein großes Problem des vorgestellten Ansatzes liegt im Speicherbedarf der Gridkarten. Es erfolgt keine Positionskorrektur durch Scan Matching oder ähnliche Verfahren, dadurch wird die Unsicherheit lediglich durch die Genauigkeit der Odometrie bestimmt. Erst beim Kreisschluss wird die korrekte relative Position erkannt. Um die dadurch entstehende deutlich höhere Unsicherheit z.B. gegenüber dem in [Hähnel et al., 2003] beschriebenen Verfahren repräsentieren zu können, werden recht hohe

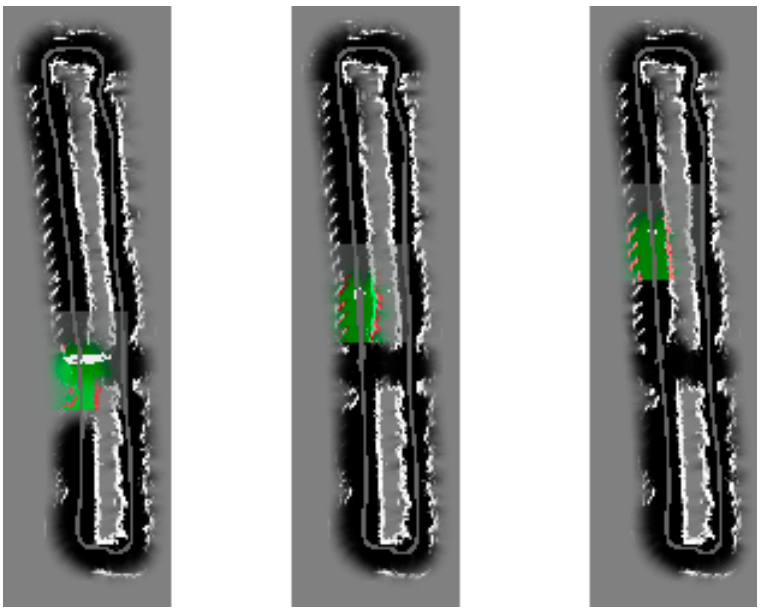

Abbildung 4.11: *Beim Kreisschluss wird durch das Map Matching die Partikelverteilung auf die korrekte Position konzentriert. Die präzise Ausrichtung von lokaler und globaler Karte ist gut zu erkennen.*

Partikelanzahlen benötigt. Die tatsächlich benötigten Partikel hängen insbesondere von der Umgebung und dem Bewegungsmodell, aber auch der Zellauflösung ab, in den hier dargestellten Experimenten lag die Partikelzahl zwischen 200 und 5000. In Kombination mit den gegenüber anderen Modellformen vergleichsweise aufwändigen Gridkarten entstehen sehr hohe Speicherkosten. Schon bei der in Abb. 4.12 gezeigten Karte mit einer Fläche von ca. 50m $\times$ 50m ergibt sich mit einer Zellauflösung von 0.1m bereits eine Speicherbelegung von 250KB (1 Byte pro Zelle, pro Partikel), bei 1000 Partikeln sind dies 250MB.

Diese separate Betrachtung aller Karten berücksichtigt jedoch nicht die inhärente Redundanz des Partikelfilter-Ansatzes, wie bereits in [Eliazar und Parr, 2003] gezeigt wurde: Die einzelnen Partikelkarten entstehen jeweils unter Annahme unterschiedlicher Positionsschätzungen und sind damit alle untereinander verschieden. Beim Resampling werden allerdings häufig Partikel entfernt und dafür neue Partikel eingefügt. Diese neuen Partikel werden als Kopien der gut bewerteten Hypothesen erzeugt, d.h. auch deren Karte wird kopiert. Zwar entstehen durch nachfolgende Kartenupdates, aufgrund der durch das stochastische Bewegungsmodell divergierenden Positionen, wiederum Unterschiede zwischen diesen ursprünglich identischen Kopien. Typischerweise betreffen diese aber nicht die gesamte Fläche der Karte, sondern nur Teilgebie-

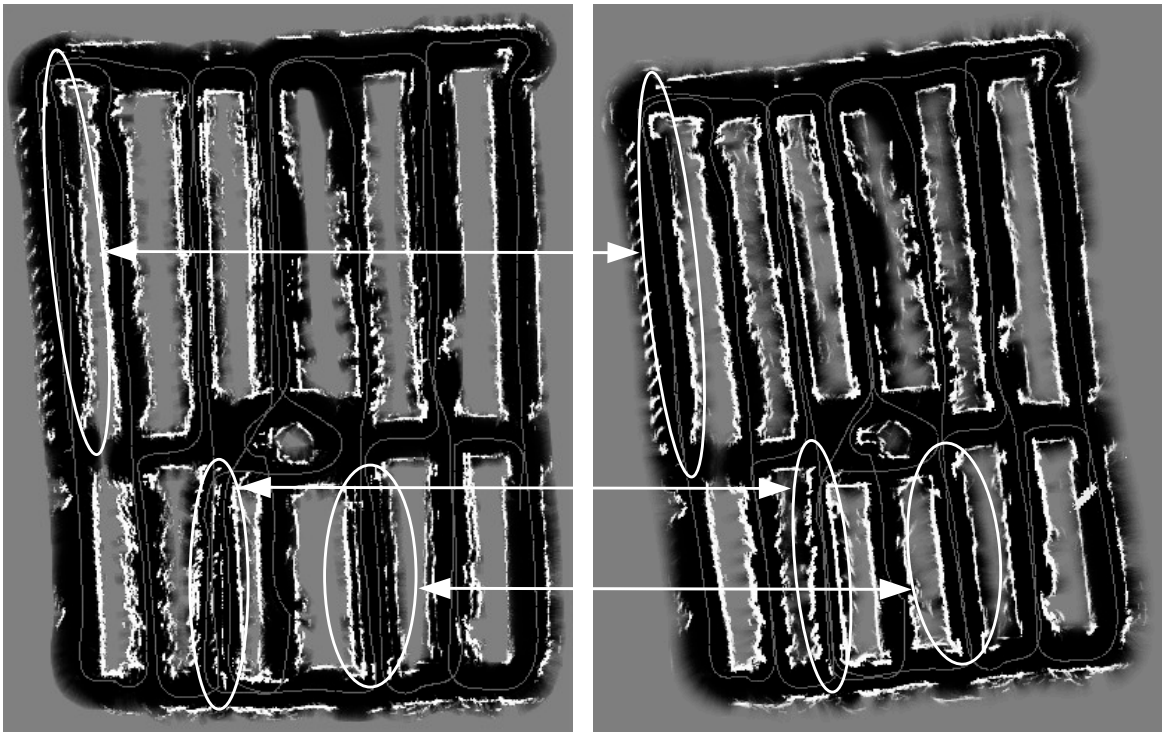

Abbildung 4.12: *Die Roboter-Trajektorie und zugehörige Karte (Teilfläche toom Baumarkt Erfurt, Fläche ca. 40m × 50m): Links ist die resultierende Karte bei Nutzung der Odometrie-Pfadmessung (Gesamt-Pfadlänge 630m) dargestellt, rechts die Karte eines ausgewählten SLAM-Partikels (sowie die Positionen aller Partikel als gelbe Punktwolke). Bei Nutzung der reinen Odometrie sind in vielen Bereichen Inkonsistenzen in der Karte erkennbar, die durch Verschiebungen der gemessenen Positionen bei aufeinander folgenden Durchfahrten durch einen Gang entstehen. Durch den SLAM-Ansatz wird eine konsistente Karte erzeugt. Einige Bereiche, in denen die Unterschiede gut erkennbar sind, sind hervorgehoben. Laufzeit und Speicherbedarf werden in Abb. 4.15 bzw. Abschnitt 4.3.2 betrachtet. Ein Referenz-Grundriss der Umgebung ist in Abb. 3.14 dargestellt.*

te, da nur jeweils ein kleiner Bereich um die aktuelle Roboterposition beobachtet wird. Ein Großteil der zuvor kopierten Karte bleibt damit identisch. Abb. 4.13 verdeutlicht diesen Sachverhalt.

In [Eliazar und Parr, 2003] wurde diese Redundanz ausgenutzt, indem die komplette Vererbungshierarchie der Partikel gespeichert wurde sowie jedes Partikel seine Änderungen in einer gemeinsamen Karte vermerkte. Dadurch wird genau dann zusätzlicher Speicher benötigt, wenn tatsächlich eine Zellbelegung durch ein Partikel geändert wird. Außerdem werden Kopieroperationen im Resampling-Schritt eingespart. Andererseits wird der Zugriff auf eine Partikel-Karte langsamer, da je-

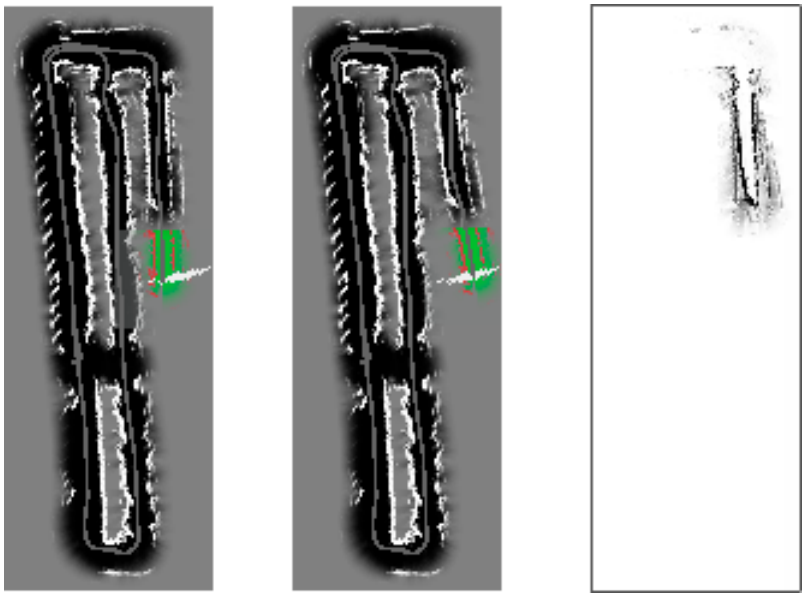

Abbildung 4.13: *Die Abbildung zeigt zwei Partikel und deren Karten zum selben Zeitpunkt. Beide Partikel entstanden nach dem Kreisschluss als identische Kopien. Anschließend schätzten beide eine leicht unterschiedliche Fortsetzung des Pfades und aktualisierten jeweils ihre Karte entsprechend. Die Differenz der entstehenden Karten ist in der rechten Abbildung dargestellt. Diese verdeutlicht, dass Unterschiede zwischen den Karten nur in dem Bereich existieren, der nach dem Kopieren beobachtet wurde, der größere Teil der Karte ist hingegen immer noch in beiden Partikeln identisch. Das Verhältnis zwischen identischem und unterschiedlichem Anteil wird mit wachsender Karte noch deutlich größer.*

de Zelle u.U. nicht nur auf Änderungen durch die aktuelle Partikel-Generation, sondern auch durch deren Vorfahren geprüft werden muss.

In dieser Arbeit wird eine alternative Lösung genutzt, welche als Hauptziel ebenfalls die Verringerung des Speicherverbrauches verfolgt. Diese beruht darauf, dass die gesamte Kartenfläche in Untereinheiten konstanter Größe (Patches) zerlegt wird. Innerhalb der Patches werden kleine Belegtheitskarten als so genannte Shared Data gehalten: Shared Data ist ein Konzept, welches in vielen Programmier-Bibliotheken unterstützt wird. Dabei können viele Objekte gleichzeitig auf die selben Nutzdaten verweisen, welche nur einmal im Speicher liegen. Ein originales Shared Data besteht jeweils aus den Nutzdaten und einer Referenz darauf. Eine Kopie legt zunächst lediglich einen neuen Verweis auf die Nutzdaten des Originalobjektes an. Nachfolgend wird zwischen modifizierenden und nicht-modifizierenden (konstanten) Zugriffen unterschieden: während ein konstanter Zugriff lediglich dem Verweis auf

die Nutzdaten folgt, wird bei einem modifizierenden Zugriff auf eine beliebige der Referenzen zunächst eine physische Kopie der Nutzdaten erzeugt, so dass die anderen Referenzen nicht verändert werden. Shared Data bietet sich dann an, wenn große Datenmengen von verschiedenen Objekten referenziert, aber nicht oder nur selten verändert werden sollen. Es bildet ein abstraktes Konzept, welches unabhängig von der tatsächlichen Anwendung spezifiziert und implementiert werden kann. Der Vorteil der Nutzung einer solchen abstrakten Implementation ist, dass für die konkreten Daten nur noch alle Zugriffe als modifizierend oder nicht-modifizierend gekennzeichnet werden müssen, damit ist die vollständige Funktionalität bereits gewährleistet.

Eine Gridmap, die aus solchen Patches zusammengesetzt ist, wird hier als Shared Gridmap bezeichnet. Damit wird die Eigenschaft ausgedrückt, dass Teile der Karte zwischen mehreren Karten-Instanzen geteilt werden können (im SLAM-Framework: Karten in den Partikeln). Durch die Nutzung des Shared-Data-Konzeptes werden beim Kopieren eines Partikels lediglich eine Anzahl Verweise auf das entsprechende Referenzpartikel erzeugt. Erst wenn ein Kartenupdate eine Zelle eines Patches verändert, wird für diesen Patch eine echte Kopie der Belegtheitswerte im Speicher angelegt.

Gegen die Nutzung des zuvor vorgestellten Distributed-Particles-Ansatzes nach [Eliazar und Parr, 2003] und für die Shared Gridmaps sprechen neben pragmatischen Gründen (einfachere Implementierung) zwei Überlegungen: Bei DP-SLAM werden die Änderungen für jede Zelle einzeln betrachtet, bei spärlichen Laser-basierten Grid-Karten erscheint dies gerechtfertigt. Bei den hier verwendeten Sonar-Karten ist jedoch immer ein gewisser zusammenhängender Bereich von Änderungen betroffen. Dies wird durch das Konzept der Patches besser repräsentiert. Außerdem ist ein wesentlicher Bestandteil der Argumentation für DP-SLAM die Verringerung des Kopier-Aufwandes im Resampling, allerdings wurde als Nachteil bereits genannt, dass der Aufwand für die Abfrage einer Zellbelegung ansteigt. Durch das Map-Matching-Konzept finden vergleichsweise selten Verteilungsupdates und damit Resampling-Operationen auf den Partikeln statt (siehe Abschnitt 4.3.1), dafür ist es allerdings deutlich öfter nötig, Zellbelegungen der Karte zu erfragen, dies geschieht sowohl im Karten-Update als auch im Verteilungsupdate für eine große Anzahl von Zellen. Im Gegensatz zum DP-SLAM ist bei

Shared Gridmaps der Zugriff auf eine Zelle kaum langsamer als in einer normalen Gridmap, damit sind diese hier besser geeignet.

Das Prinzip der Shared Gridmap weist auf den ersten Blick einige Gemeinsamkeiten mit einer in [Grisetti et al., 2007] vorgestellten Repräsentation auf. Auch dort teilen die Partikel untereinander lokale Kartenbereiche, welche ebenfalls als Patches bezeichnet werden. Allerdings werden bei genauerem Hinsehen wesentliche Unterschiede deutlich: dort werden die Partikel explizit in Cluster eingeteilt, innerhalb des Clusters wird eine lokale Karte von allen Partikeln geteilt. Weiterhin teilen alle Partikel eines Cluster auch alle Änderungen an der lokalen Karte, d.h. eine nachträgliche Vereinzelung der geteilten Karten bei modifizierenden Zugriffen findet nicht statt. Dies beruht auf der Annahme, dass Cluster von Partikeln mit gleicher topologischer Kartenstruktur und lediglich geringen Positionsunterschieden innerhalb des Clusters existieren, wobei die Partikel sich relativ zu einer einmal bekannten lokalen Karte gleich lokalisieren müssen. Daraus folgt, dass die lokalen Karten eine feste relative Lage zu den Partikeln besitzen und sich daher in ihrer absoluten Position mit dem jeweiligen Partikel verschieben. Im Unterschied dazu liegen die Patches bei Shared Gridmaps an festen Positionen im globalen Bezugssystem und die Partikel nehmen unterschiedliche Positionen in Bezug zu diesen ein. Eine explizite Clusterung mit entsprechenden Entscheidungskriterien entfällt dafür. In Bezug auf die Speicherkosten verhält sich der von Grisetti et. al. vorgestellte Ansatz teilweise ähnlich zum Shared-Gridmap-Konzept (Abb. 4.15).

In den Abbildungen 4.10 bis 4.13 lässt sich verfolgen, wie die Positions-Unsicherheit (repräsentiert durch die Breite der Partikelverteilung) während der Roboterbewegung ansteigt, bei einem Kreisschluss minimiert wird, um anschließend wieder anzuwachsen. Die Verringerung der Unsicherheit erfolgt stets durch Entfernung schlecht bewerteter Partikel und dafür zusätzliches Einfügen neuer Kopien der gut bewerteten Partikel. Durch die Nutzung der Shared Gridmaps hat dies unmittelbare Auswirkungen auf den Gesamt-Speicherbedarf des Partikelfilters: bei jedem Löschen eines Partikels wird entsprechend Speicher freigegeben, die dafür erzeugte Partikel-Kopie benötigt hingegen zunächst noch keinen eigenen Kartenspeicher. Bei den darauf folgenden Kartenupdates werden aber jedes mal zuvor geteilte Karten-Patches verändert, diese erzeugen dazu eine eigene Kopie der internen Belegtheitsdaten, wodurch der Speicherverbrauch langsam wieder ansteigt.

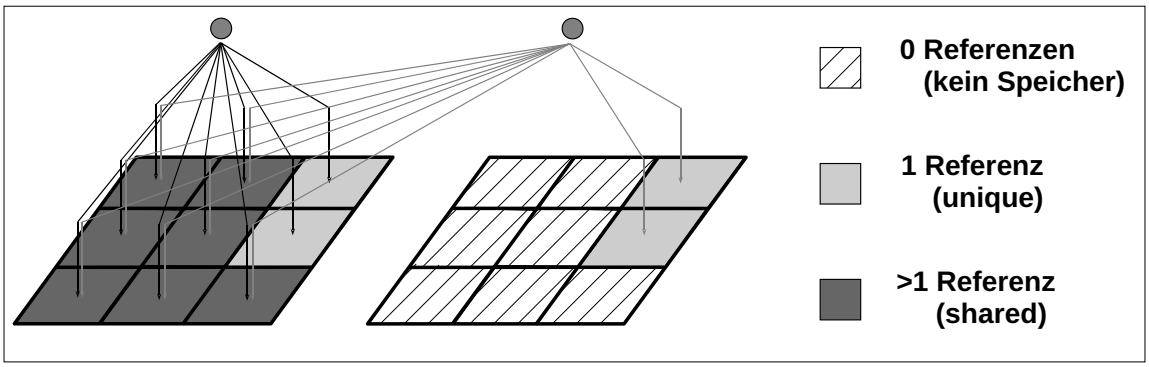

Abbildung 4.14: *Die Grafik zeigt die Struktur der Shared Gridmaps: Jede Karte besteht aus vielen Patches (Standardgröße ca. 100 × 100 Gridzellen), jeder Patch kann seinen Inhalt mit anderen Karten teilen. Dies geschieht, wenn der Inhalt beider Karten innerhalb dieses Patches identisch ist. Im Bild sind zwei Shared Gridmaps dargestellt, die einen großen Teil ihres Inhalts teilen. Diese geteilten Inhalte belegen nur einmal Speicher. Die beiden Patches am rechten Rand sind verschieden, daher wird dafür in jeder Karte eigener Speicher benötigt.*

Die maximalen Speicherkosten werden damit nicht mehr ausschließlich durch die Größe der Umgebung bestimmt, sondern wesentlich auch durch die Schleifengröße. Insbesondere bei großen Umgebungen, die sich in viele kleinere Schleifen zerlegen lassen, kann eine sehr große Speicher-Einsparung erreicht werden.

Abbildung 4.15 stellt sowohl Rechenzeit- als auch Speicherverbrauch einer einfachen Gridmap- und Shared-Gridmap-Implementation gegenüber. Dort ist der zeitliche Verlauf der benötigten Rechenzeit jeweils für ein Karten- und Verteilungsupdate sowie des Gesamt-Kartenspeichers über 500 Partikel dargestellt. Es ist gut zu erkennen, wie bei der normalen Gridmap der Speicherbedarf mit wachsender Kartengröße kontinuierlich wächst, während bei der Shared Gridmap der Speicherbedarf jeweils bei einem Kreisschluss stark abfällt, um dann wieder langsam anzuwachsen. Die maximalen Speicherkosten beider Ansätze unterscheiden sich nur gering, allerdings ist zu bedenken, dass bei weiter wachsender Karte die normalen Gridmaps ständig mehr Speicher benötigen, während bei Shared Gridmaps der Maximalwert kaum noch steigt. Der anfänglich deutlich höhere Speicherbedarf der Shared Gridmap ist darin begründet, dass durch die Aufteilung in Patches konstanter Größe die Karte bei Ausdehnung jeweils um einen festen Betrag wächst, während die normale Gridmap schrittweise vergrößert wird.

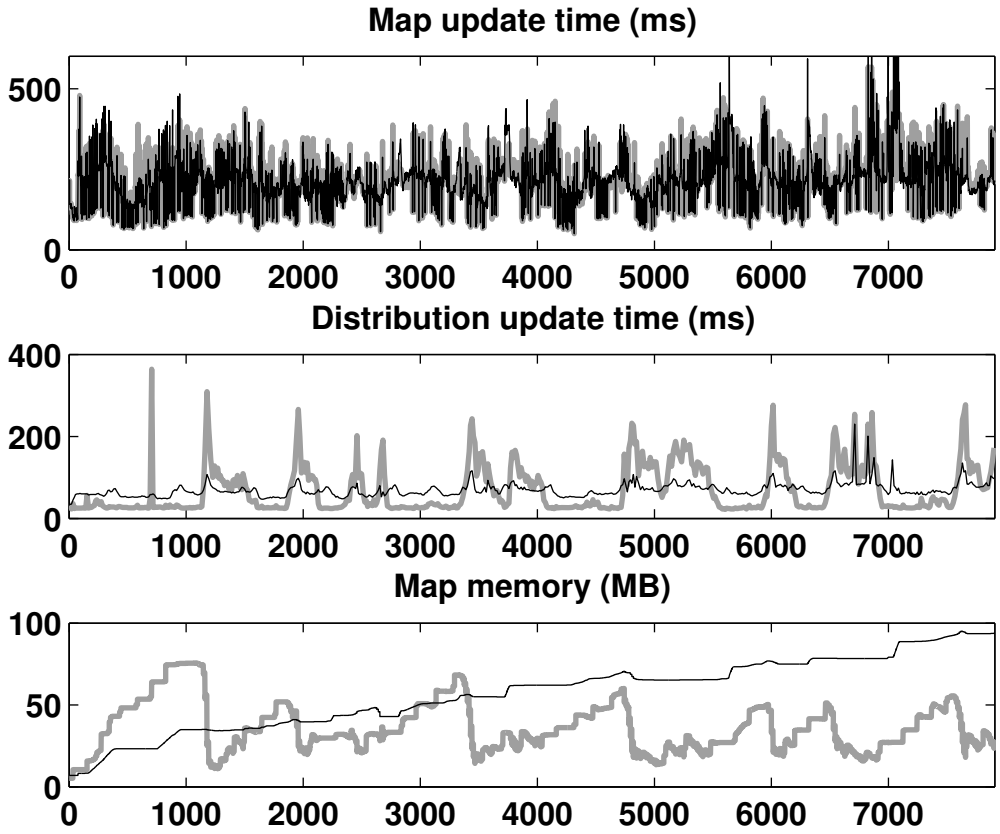

Abbildung 4.15: *Rechenzeit- und Speicherkosten für Map-Match-SLAM mit normalen Gridmaps (schwarz) und Shared Gridmaps (grau). Die Darstellung bezieht sich auf die Karte in Abb. 4.12 mit 500 Partikeln. In der Gesamt-Rechenzeit bestehen nur marginale Unterschiede. Horizontal sind alle Verarbeitungsschritte des Partikelfilters (Bewegungs-, Karten- und Verteilungsupdates) in fortlaufender Nummerierung aufgetragen, die Abbildungen entsprechen also dem zeitlichen Verlauf der dargestellten Größen.*

Es ist weiterhin zu erkennen, dass kaum Unterscheide in der Laufzeit bestehen. Für die Datenaufzeichnung bewegte sich der Roboter hier ca. 35 Minuten durch die Umgebung, die Durchschnittsgeschwindigkeit betrug knapp 0.35m/s. Mit 500 Partikeln und der Zellgröße 0.1m benötigten beide Ansätze (im Offline-Betrieb auf den zuvor gespeicherten Daten) fast exakt die selbe Gesamtzeit von etwa 16 Minuten (auf einem Core2-Prozessor, der etwa auch der Hardwareausstattung des Shopping-Assistenten entspricht), wovon ca. 95% auf die Kartenupdates (lokale + globale Karte) entfallen. Das Verfahren ist damit unter diesen Randbedingungen auch geeignet zum Online-Betrieb.

In den bisherigen Ausführungen wurde nur der Speicherbedarf der Partikel-Karten betrachtet. Nachdem die Kosten für die Karten-Repräsentation deutlich verringert wurden, zeigte sich, dass nun die Pfadinformationen der Partikel beim Gesamt-Speicherbedarf deutlich überwiegen. Allerdings sind hier die selben Überlegungen zur Redundanz gültig wie für die Karten. Daher wurden nun auch die Partikel-Pfade auf eine ähnliche Weise transformiert, indem jeder Pfad in Abschnitte konstanter Länge zerlegt wird, diese Abschnitte können wiederum zwischen mehreren Partikeln geteilt werden.

4.3.3 Nicht-persistente Karte (On-Demand Maps)

Mit den Shared Gridmaps wurde ein Weg gefunden, den Speicherverbrauch effektiv zu beschränken, dieser ist damit im Wesentlichen abhängig von der Schleifengröße zwischen den Kreisschlüssen und weniger von der Gesamtgröße der Karte. Innerhalb einer großen Schleife können die Speicherkosten auf diese Weise allerdings nicht verringert werden.

Abb. 4.16 zeigt ein Beispiel einer solchen Umgebung. Die realen Umgebungsdimensionen betragen hier ungefähr 80m $\times$ 100m. Während bei der Karte in Abb. 4.12 ein Kreisschluss jeweils nach spätestens ca. 120m erreicht wurde, beträgt die Schleifengröße hier über 300m. Diese große Kreislänge hat zwei Auswirkungen, welche beide die Speicherkosten stark ansteigen lassen: Zum Einen wird durch die große offene Schleife die Positionsunsicherheit vor dem Kreisschluss sehr groß, um dies zu repräsentieren, sind deutlich mehr Partikel nötig als bei der zuvor gezeigten Karte. Dies ist ein Problem, welches alle Partikel-basierten SLAM-Verfahren betrifft, es wirkt sich aber beim her vorgestellten Map-Match-SLAM wegen des Verzichts auf Scan Matching und der damit verbundenen allgemein höheren Unsicherheit besonders stark aus. Außerdem sind trotz Shared Gridmaps die maximalen Speicherkosten extrem hoch, da diese gerade von der Schleifengröße abhängen. In dieser Umgebung würde bei 5000 Partikeln und einer Zellgröße von 0.1m die maximale Speicherbelegung allein für die globalen Karten ca. 3-4GB betragen. Nach dem Kreisschluss würde die Speicherbelegung zwar extrem schrumpfen, aus offensichtlichen Gründen ist jedoch der maximale Bedarf über die gesamte Laufzeit entscheidend.

Es ist anzumerken, dass bei Nutzung von 5000 Partikeln die Kartierung in Echtzeit nicht mehr möglich ist. Während allerdings die Rechenzeit insofern eine „weiche“ Schranke ist, dass nach endlich langer Zeit trotzdem ein Ergebnis erzielt werden kann, ist der Speicherbedarf eine harte Einschränkung, da normalerweise der im System verfügbare Speicher beschränkt ist und das Programm bei Erreichen der Grenze abgebrochen werden muss. Um wenigstens den Speicherbedarf so weit zu verringern, dass eine Offline-Kartierung möglich ist, wurde eine weitere Variante der Kartenverwaltung implementiert. Diese macht sich zu Nutze, dass im Partikelfilter bereits alle für die Kartierung verwendeten Beobachtungen jeweils mit Verweis auf die entsprechende Pose im Pfad eines Partikels gespeichert werden. Es ist damit also prinzipiell möglich, zu jedem Zeitpunkt die Karte jedes einzelnen Partikels komplett neu zu erzeugen.

Praktisch wird zur Bestimmung des korrekten Pfades die Karte jeweils nur im Verteilungsupdate benötigt, um lokale und globale Karte vergleichen zu können. Zu diesem Zweck ist allerdings in jedem Partikel nur ein begrenzter Bereich der globalen Karte um die aktuell geschätzte Position von Interesse. Jedes Partikel rekonstruiert daher einen lokalen Ausschnitt der globalen Karte, indem es entlang des eigenen gespeicherten Pfades Beobachtungen sucht, welche innerhalb eines Radius zur aktuellen Position liegen. Aus diesen Beobachtungen wird eine temporäre globale Karte erzeugt und nach dem Vergleich mit der lokalen Karte wieder gelöscht. Da die Karte aus den selben Beobachtungen entsteht, ist sie in dem lokalen Bereich identisch mit einer kontinuierlich erweiterten, persistenten Karte. Der benötigte Speicherplatz ist nur noch von der Größe der lokalen Karte und nicht mehr von der Umgebungsgröße abhängig. Selbstverständlich kostet die Neu-Erzeugung der Karte für jeden Karten-Vergleich sehr viel Zeit, allerdings wird im Gegenzug die harte Einschränkung durch den begrenzten zur Verfügung stehenden Speicher umgangen. Die Implementation mit nicht-persistenten Karten ist nur als Ausweich-Lösung für den Offline-Betrieb bei sehr großen Schleifen gedacht. Für Umgebungen mit zahlreichen Schleifen, wie z.B. das Zieleinsatzfeld Baumarkt, ist diese Variante nicht notwendig.

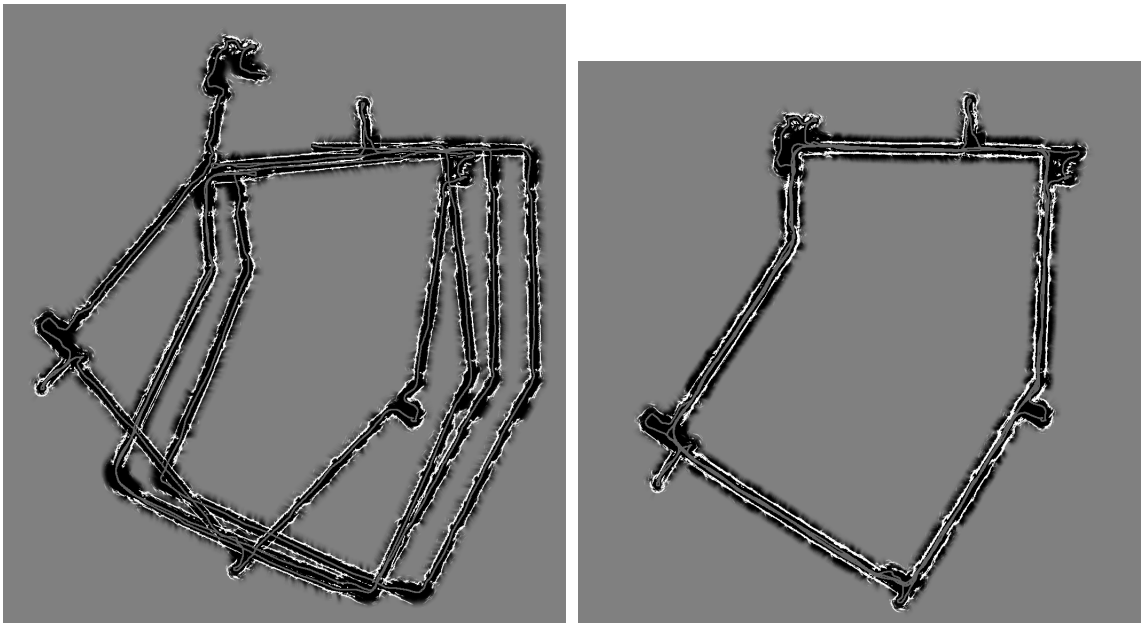

Abbildung 4.16: *Die Kartenfläche beträgt ca. 80m × 100m, die Kreislänge ungefähr 300m (links Odometrie, rechts Ergebnis des Map-Match-SLAM). Der Roboter fuhr fast vier Runden (dreimal im Uhrzeigersinn, einmal gegen den Uhrzeigersinn, Gesamtstrecke 1150m), wobei zunächst nur der Hauptgang befahren wurde, erst in der letzten Runde auch seitlich abzweigende Gebiete einbezogen wurden. Der Positionsfehler der Odometrie beträgt bis zu 10m auf einer einzelnen Runde, vor allem durch den zusätzlichen Orientierungsfehler wächst dieser bei aufeinander folgenden Runden jedoch deutlich stärker an. Zur Kartierung unter Nutzung der reinen Odometrie (kein Scan Matching o.ä.) wurden 5000 Partikel benötigt, daher konnten die Karten nicht mehr persistent in den Partikeln gespeichert werden. Die abgebildete Karte wurde nachträglich aus dem rekonstruierten Pfad berechnet.*

4.3.4 Dynamisierung der Partikelanzahl

Im vorigen Abschnitt wurde argumentiert, dass die große Unsicherheit eine hohe Partikelanzahl voraussetzt. Allerdings ist nur der *Maximal*wert der Unsicherheit während der Kartierung tatsächlich sehr hoch. In Abb. 4.17 ist zu erkennen, dass die Unsicherheit nach dem Kreisschluss fast völlig verschwindet und auch nachfolgend bei Weitem nicht wieder so hoch ansteigt wie vor dem Kreisschluss. Dies liegt darin begründet, dass der Roboterpfad nach der ersten Schleife die meiste Zeit durch die dann bereits kartierte Umgebung führt und sich darin recht genau positionieren kann. Lediglich kurze Abzweige erkunden zusätzliches Gebiet, wobei die Unsicherheit temporär ansteigt, bei Rückkehr zu einer bekannten Position jedoch schnell wieder absinkt. Die Positions-

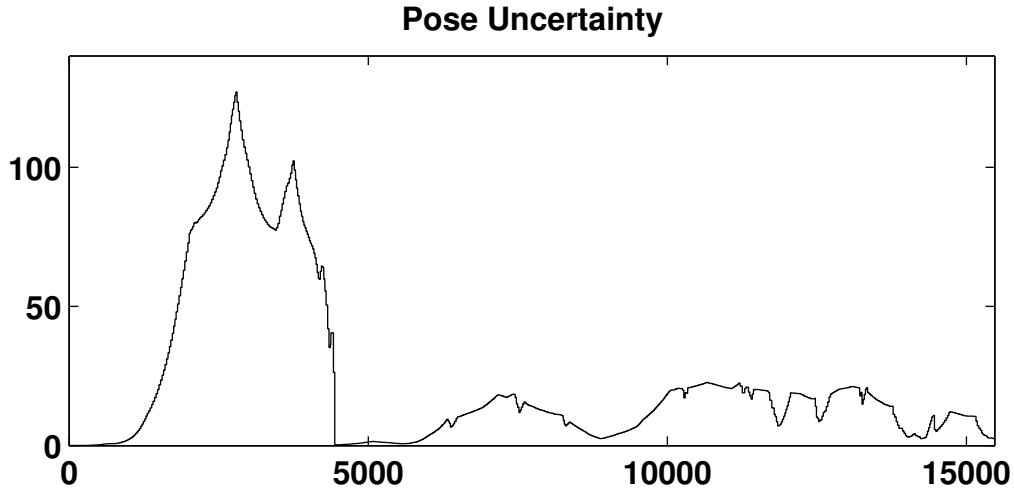

Abbildung 4.17: *Der Verlauf der Positions-Unsicherheit (Kovarianz der Partikelpositionen) während der Kartierung (bezogen auf die Karte in Abb. 4.16): Zwar steigt diese vor dem ersten Kreisschluss sehr stark an, nach dem Kreisschluss bleibt sie jedoch die meiste Zeit sehr gering und erreicht nicht mehr als 25% des Maximalwertes.*

Unsicherheit wurde hier durch die Kovarianz über die Partikelpositionen abgeschätzt. Die Kovarianz eignet sich allerdings nur für unimodale Verteilungen als Maß der Unsicherheit, diese Einschränkung wurde in diesem Experiment jedoch zumeist erfüllt. Eine Methode zur Schätzung der Unsicherheit, die auch bei multimodalen Verteilungen anwendbar ist, wird nachfolgend erläutert.

Mit den in den vorangegangenen Abschnitten vorgestellten Anpassungen (Shared Gridmaps, Abschnitt 4.3.2 , On-Demand Maps, Abschnitt 4.3.3) wurde vor allem das Ziel verfolgt, die Speicherkosten zu verringern. Bei der Nutzung der Shared Gridmaps wird der Speicherverbrauch nicht mehr im Wesentlichen durch die Partikelzahl, sondern deutlich stärker durch die tatsächlich vorhandene Unsicherheit bestimmt (siehe Abb. 4.15). Allerdings wird unnötig Rechenzeit aufgewendet, wenn mehr Partikel benutzt werden, als zur Approximation der Verteilung nötig sind. Es erscheint also trotz der Speicheroptimierungen immer noch sehr sinnvoll, die Partikelanzahl dynamisch an die jeweilige Verteilung anzupassen. In diesem Abschnitt wird beschrieben, wie die jeweilige Partikelzahl bestimmt wird und welche Modifikationen am Algorithmus dazu vorgenommen wurden.

Bestimmung der benötigten Partikelanzahl

Zwar erscheint es einleuchtend, die Partikelzahl an die aktuellen Zustandsunsicherheit anzupassen, allerdings ist nicht offensichtlich, wie die Unsicherheit zu bestimmen ist und welche funktionale Abhängigkeit zwischen einer wahrgenommenen Unsicherheit und der Partikelzahl bestehen sollte. Die zur Veranschaulichung genutzte Kovarianz der Partikelpositionen ist, wie bereits erwähnt, lediglich eine recht grobe Annäherung, da sie zum Beispiel bei multimodalen Verteilungen eine sehr schlechte Approximation der realen Unsicherheit berechnet.

In [Fox, 2001] wird ein Ansatz zur dynamischen Adaption der Partikelzahl für die Monte-Carlo-Lokalisation vorgestellt, der nicht auf der expliziten Berechnung eines Unsicherheitswertes beruht. Obwohl in der dort betrachteten Anwendung der Partikelfilter benutzt wird, um die Roboterposition auf Grundlage einer *vorgegebenen* Karte zu bestimmen (Selbstlokalisation), lässt sich die Methode in ähnlicher Weise beim SLAM-Problem nutzen. Die von Fox vorgeschlagene Methode beruht darauf, die Abweichung zwischen der durch den Partikelfilter geschätzten Verteilung $\boldsymbol{p}$ und einer angenommenen exakten a-posteriori-Verteilung $\hat{\boldsymbol{p}}$ zu bestimmen und zu beschränken. Es wird jeweils die Partikelzahl gesucht, die den Abstand $K(\hat{\boldsymbol{p}}, \boldsymbol{p})$ zwischen der exakten und der geschätzten Verteilung mit einer festgelegten Wahrscheinlichkeit $1 - \delta$ nicht größer als eine Distanzschwelle ϵ werden lässt. Als Abstandsmaß kommt dabei die Kullback-Leibler-Divergenz (KLD) zum Einsatz, das Verfahren wird deshalb als KLD-Sampling bezeichnet.

Zur Approximierung der KL-Distanz wird zunächst eine Bin-Aufteilung (regelmäßige Diskretisierung) des Zustandsraumes vorgenommen und die Abdeckung des Raumes durch die Anzahl der von Partikeln belegten Bins abgeschätzt. Fox zeigt, dass die Anzahl der Partikel n in guter Näherung bestimmt werden kann durch

$$n = \frac{k-1}{2\epsilon} \left\{ 1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta} \right\}^3 \tag{4.15}$$

wobei k die Anzahl der Bins und $z_{1-\delta}$ das obere $(1-\delta)$-Quantil der Standard-Normalverteilung ist (d.h. für eine Zufallsvariable $X \sim \mathcal{N}(0,1)$ gilt $P(X < z_{1-\delta}) = 1 - \delta$). Während die gewünschte Distanz ϵ und die Einhaltungswahrscheinlichkeit $1-\delta$ in ihrer Bedeutung gut verständlich

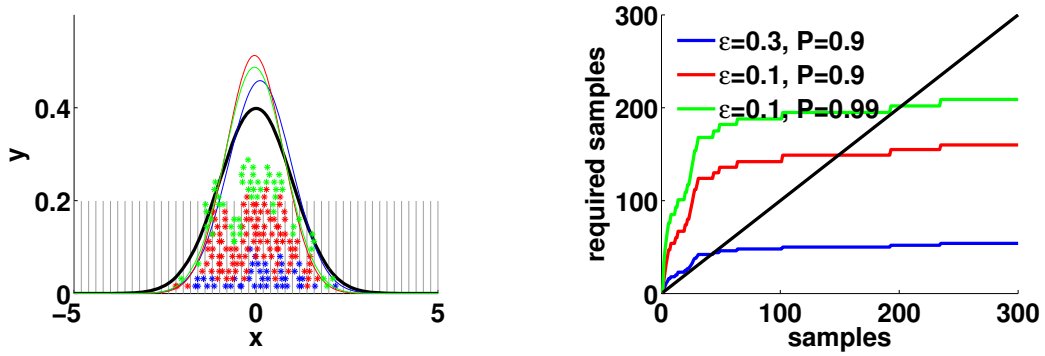

Abbildung 4.18: *Veranschaulichung des KLD-Sampling an einer 1D-Normalverteilung (links als schwarze Kurve, senkrechte Linien = Bin-Einteilung des Zustandsraumes): Die Verteilung wird mittels Partikeln approximiert, indem sukzessive zufällig Samples gezogen werden (im Bild als Sternchen). Nach jedem gezogenen Sample wird die Anzahl der belegten Bins aktualisiert und daraus die Anzahl der* benötigten *Partikel nach Gleichung 4.15 berechnet. Zur Verdeutlichung wurden drei verschiedene Wertepaare für die Parameter ϵ, $P(K(\hat{\boldsymbol{p}}, \boldsymbol{p}) \leq \epsilon)$ gewählt (farbig im rechten Bild, schwarz der Verlauf der tatsächlichen Partikelzahl). Sobald die Zahl der vorhandenen Partikel die Zahl der benötigten Partikel erreicht, wird das KLD-Sampling beendet. Die Farben der Partikel (links) verdeutlichen dies: Bei der Parameterwahl [0.3, 0.9] werden nur die 42 blauen Partikel erzeugt, bei [0.1, 0.9] zusätzlich die roten (insgesamt 149), bei [0.1, 0.99] zusätzlich die grünen (insgesamt 202). Aus den Partikelsets wurden zum Vergleich der Repräsentationsgenauigkeit wieder Normalverteilungen geschätzt (farbige Kurven links).*

sind und damit leicht Werte für diese Parameter festgelegt werden können, ist für die Bin-Aufteilung keine Berechnungsgrundlage herleitbar. Diese richtet sich nach der gewünschten Lokalisierungsgenauigkeit und wurde anhand der Performanz des Algorithmus experimentell bestimmt, mit einer Bin-Größe von 0.5m $\times$ 0.5m $\times$ 10° wurde dabei dort eine recht grobe Einteilung gewählt.

Aufgrund der hohen Dimensionsanzahl ist es beim SLAM nicht möglich, eine Bin-Einteilung des gesamten Zustandsraumes vorzunehmen. Diese müsste mindestens die aktuelle Position und die Karte jedes Partikels berücksichtigen. Stattdessen wird auch hier lediglich die aktuelle Position genutzt. Es zeigt sich, dass Partikel mit unterschiedlicher Positionshistorie nur sehr selten ähnliche oder gleiche aktuelle Positionen einnehmen, daher ist diese Vereinfachung zulässig. Weiterhin wird die Roboter-Orientierung φ nicht in der Bin-Einteilung berücksichtigt, eine unterschiedliche Partikelorientierung führt automatisch recht schnell zur Diversifizierung der (x,y)-Positionen. Für die Bin-Größe erwies sich,

dass eine wesentlich feinere Einteilung als bei der Lokalisation im Originalverfahren notwendig ist. Da die Anforderungen an die Positionsgenauigkeit und damit die räumliche Abdeckung durch die Partikel mit der Zellgröße der Karte steigen, wird die Bin-Größe ebenfalls an die Zell-Größe gekoppelt.

Eine Vereinfachung der Formel 4.15 lässt sich in der Form

$$n = (k-1) \cdot c \quad c = \frac{1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right\}^3 \tag{4.16}$$

$$n' = (k-1) \cdot c' \tag{4.17}$$

finden. Eine gröbere Schätzung der benötigten Partikelzahl n' wird hier berechnet, indem der veränderliche Faktor c durch eine Konstante c' ersetzt wird. Auf diese Weise ist eine etwas intuitivere Interpretation („durchschnittlich c' Partikel je belegtes Bin/Gridzelle“) möglich, ebenso wird die Berechnung geringfügig vereinfacht. Es lässt sich zeigen, dass für sinnvolle Werte von δ der exakte Werte von c für $k > 1$ monoton fällt und typischerweise von einem Startwert von ca. $3 \cdot (2\epsilon)^{-1}$ gegen $(2\epsilon)^{-1}$ konvergiert. n' überschätzt also die Anzahl der benötigten Partikel für steigende k leicht, dies ist aber eher zu tolerieren als eine Unterschätzung.

Die Anzahl der benötigten Partikel n wird während des Resampling iterativ berechnet, indem nach jedem neu gesampelten Partikel die Anzahl der belegten Bins k und damit auch n/n' aktualisiert wird. Sobald die Anzahl der gezogenen Partikel die wachsende Schwelle n/n' (oder eine festgelegte Maximalanzahl) erreicht, wird das Resampling beendet und das neu erzeugte Partikelset ersetzt das alte.

Eingaben

1 $P = \{(x_1, w_1), \ldots, (x_N, w_N)\}$ // *N Partikel*
// *jeweils Kombination aus Zustand x und Gewicht w*

Initialisierung

2 $P_{new} \leftarrow \emptyset$ // *neues Partikelset = leer*
3 $M_{1:N} \leftarrow 0$ // *Multiplikator für alle Partikel = 0*
4 $N_{new} \leftarrow 0$ // *Anzahl neue Partikel = 0*
5 $W \leftarrow \sum_{j=1:N} w_j$ // *Summe aller Partikelgewichte*
6 $b_{1:k} \leftarrow 0$ // *alle Bins = leer*
7 $B \leftarrow 0$ // *Anzahl der belegten Bins = 0*

Algorithmus

Label 1: **// Auswahl eines Partikels**
8 `für` $i = 1 : N$ // *für jedes Partikel:*
9 $d_i = \frac{w_i}{W} - \frac{M_i}{N_{new}}$ // *Differenz zwischen gewünschtem und*
// *tatsächlichem Anteil an der Verteilung*
10 $i' = argmax(d_i)$ // *wähle Partikel mit max. Differenz*
11 $M_{i'} = M_{i'} + 1$ // *erhöhe Multiplikator*
12 $N_{new} = N_{new} + 1$ // *erhöhe Zahl neuer Partikel*
13 $l = bin(x_{i'})$ // *berechne Binindex für gewähltes Partikel*
14 `Wenn` $b_l = 0$`, dann` // *bisher unbelegtes Bin?*
15 $b_l = 1$ // *markiere Bin als belegt*
16 `B = B + 1` // *erhöhe Anzahl belegter Bins*
17 `Wenn` $N_{new} < c' \cdot B$`, dann`
18 **goto Label 1** // *weitere Partikel benötigt*
19 `für` $i = 1 : N$ // *für jedes Partikel:*
20 $w_i = \frac{w_i}{W} \cdot \frac{N_{new}}{M_i}$ // *Gewicht gleicht verbleibenden Unterschied*
// *zwischen gewünschtem und tatsächlichem*
// *Anteil an der Verteilung aus*
21 `für` $j = 1 : M_i$ // *erzeuge M_i Kopien von Partikel i*
22 $P_{new} = P_{new} \cup P_i$

Rückgabe

23 $P_{new} = \{(x_1, w_1), \ldots, (x_{N_{new}}, w_{N_{new}})\}$ // *neues Partikelset mit N_{new} Partikeln*

Abbildung 4.19: *Pseudocode für das Resampling des Partikelsets mit impliziter Bestimmung der benötigten Partikelanzahl*

Optimale Samplewahl für kleine Partikelzahlen

Da das Resampling prinzipiell nach dem Hinzufügen jedes einzelnen Partikels abbrechen könnte, muss besonders darauf geachtet werden, dass die Partikel-Verteilung in jedem Schritt möglichst exakt die Gewichtsverteilung wiedergibt. Dafür wird eine Sampling-Methode gesucht, die möglichst mit jedem einzelnen gezogenen Partikel die (durch die diskrete Partikelzahl unvermeidliche) Differenz zwischen der entstehenden und der gewünschten Partikel-Verteilung minimiert. Dies geschieht, indem in jedem Sampling-Schritt das Partikel bestimmt wird, für welches der relative Anteil in der bisher erzeugten Verteilung (allen zuvor gezogenen Partikeln der neuen Generation) und der gewünschten Verteilung (bestimmt durch die Gewichte der Partikel der vorherigen

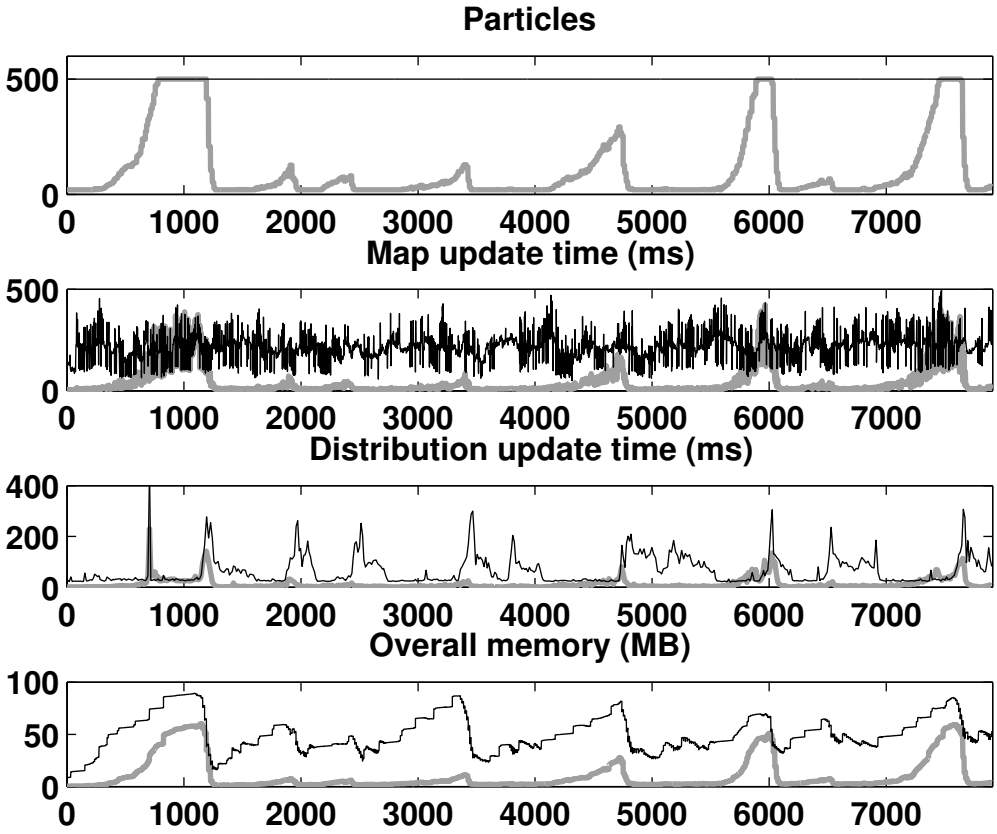

Abbildung 4.20: *Vergleich des Verlaufs der Partikelzahl, Rechenzeit und des Speicherbedarfs bei konstanter (schwarz) und dynamischer (grau) Partikelzahl. Die summierte Gesamtzeit betrug für die dynamische Partikelzahl 200s gegenüber 920s für die konstante Partikelzahl.*

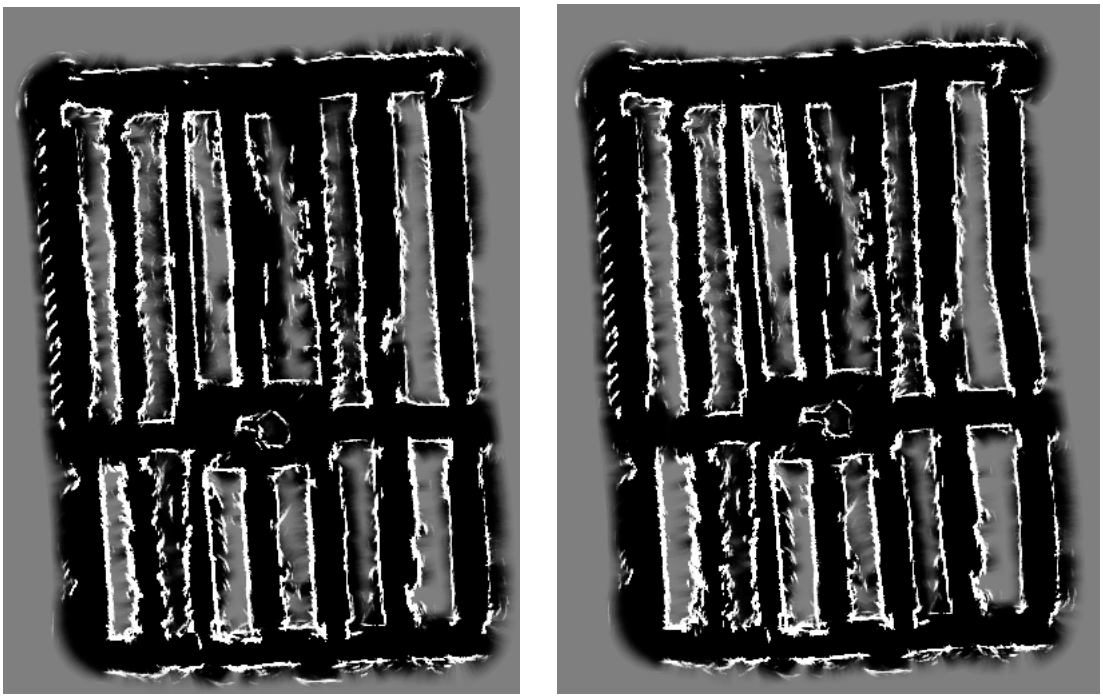

Abbildung 4.21: *Vergleich der Ergebniskarten mit konstanter (links) und dynamischer (rechts) Partikelzahl. Die Unterschiede sind relativ gering (weitere Angaben sowie original Odometriepfad siehe Abb. 4.12)*

Generation) die größte Differenz aufweist. (Pseudocode Abb. 4.19). Auf diese Weise wird der maximale Fehler für ein einzelnes Partikel im letzlich erzeugten Partikel-Set, unabhängig vom (zuvor nicht bekannten) Moment des Abbruchs, minimiert.

Abb. 4.20 zeigt den Verlauf von Partikelzahl, Rechenzeit und Speicher im Vergleich zwischen SLAM mit konstanter und mit dynamischer Partikelzahl. Die dynamische Partikelzahl wurde auf minimal 20 und maximal 500 Partikel beschränkt, im konstanten Fall wurde sie auf 500 festgelegt. Als Bin-Größe zur Bestimmung von k in Gleichung 4.15 wurde die Zellgröße der Karte gewählt, mit $\epsilon = 0.3$ und $P(K(\hat{\boldsymbol{p}}, \boldsymbol{p}) \leq \epsilon) = 0.9$, woraus sich $z_{1-\delta} \approx 1.28$ ergibt. Mit der dynamischen Anpassung der Partikelzahl wurde insgesamt für Bewegungs-, Karten- und Verteilungsupdates etwa 22% der Rechenzeit gegenüber der konstanten Maximalanzahl benötigt (200s gegenüber 920s). Wie in Abb. 4.21 zu erkennen ist, bestehen nur geringe Unterschiede zwischen den Karten, welche nicht größer sind als die Abweichungen zwischen verschiedenen Durchläufen mit gleichen Parametereinstellungen. Die Dynamisierung der Partikelzahl hat also (bei entsprechender Parametrierung) keine signifikanten Auswirkungen auf das Ergebnis.

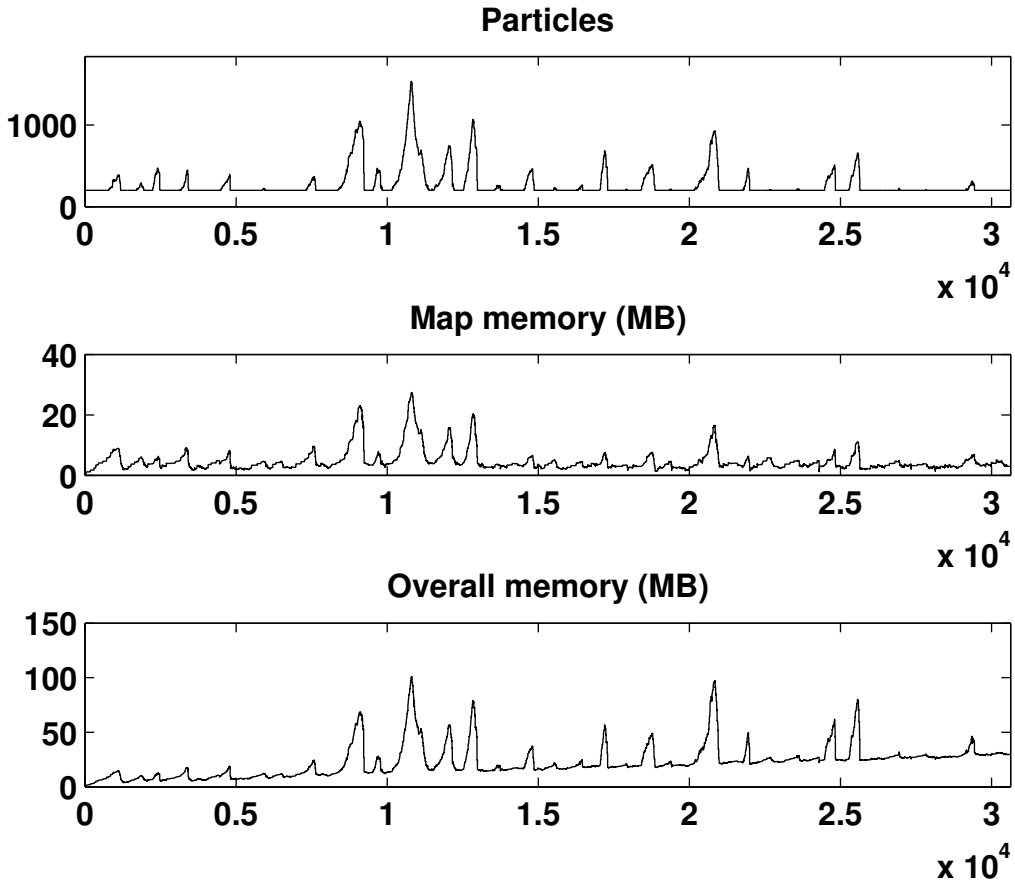

Abbildung 4.22: *Der Verlauf der Partikelanzahl sowie des Speicherbedarfs während der Kartierung des Baumarktes (Abb. 4.23). Es ist gut zu erkennen, wie durch den Einsatz der Shared Gridmaps trotz wachsender Kartenfläche der Speicherbedarf für die Karten nicht nennenswert anwächst. Lediglich beim Anstieg der Partikelzahl wird entsprechend mehr Speicher benötigt. Das leichte Anwachsen des Gesamt-Speicherbedarfs bei nahezu gleichbleibender Partikelzahl resultiert aus dem immer länger werdenden Pfad, der pro Partikel geschätzt wird.*

Zum Abschluss dieses Abschnitts zu SLAM mit entfernungsmessenden Sensoren soll hier noch die Kartierung des gesamten Areals des toom Baumarktes Erfurt (die Testumgebung für den Shopping-Assistenten) gezeigt werden (Abb. 4.23). Die Grundfläche beträgt hier ca. 50m $\times$ 120m, aufgrund der vielen kleinen Schleifen ist diese Umgebung recht anspruchsvoll, eignet sich aber auch sehr gut für das implementierte SLAM-Verfahren. Die gesamte Fahrstrecke des Roboters während der Kartierung betrug hier ca. 2300m.

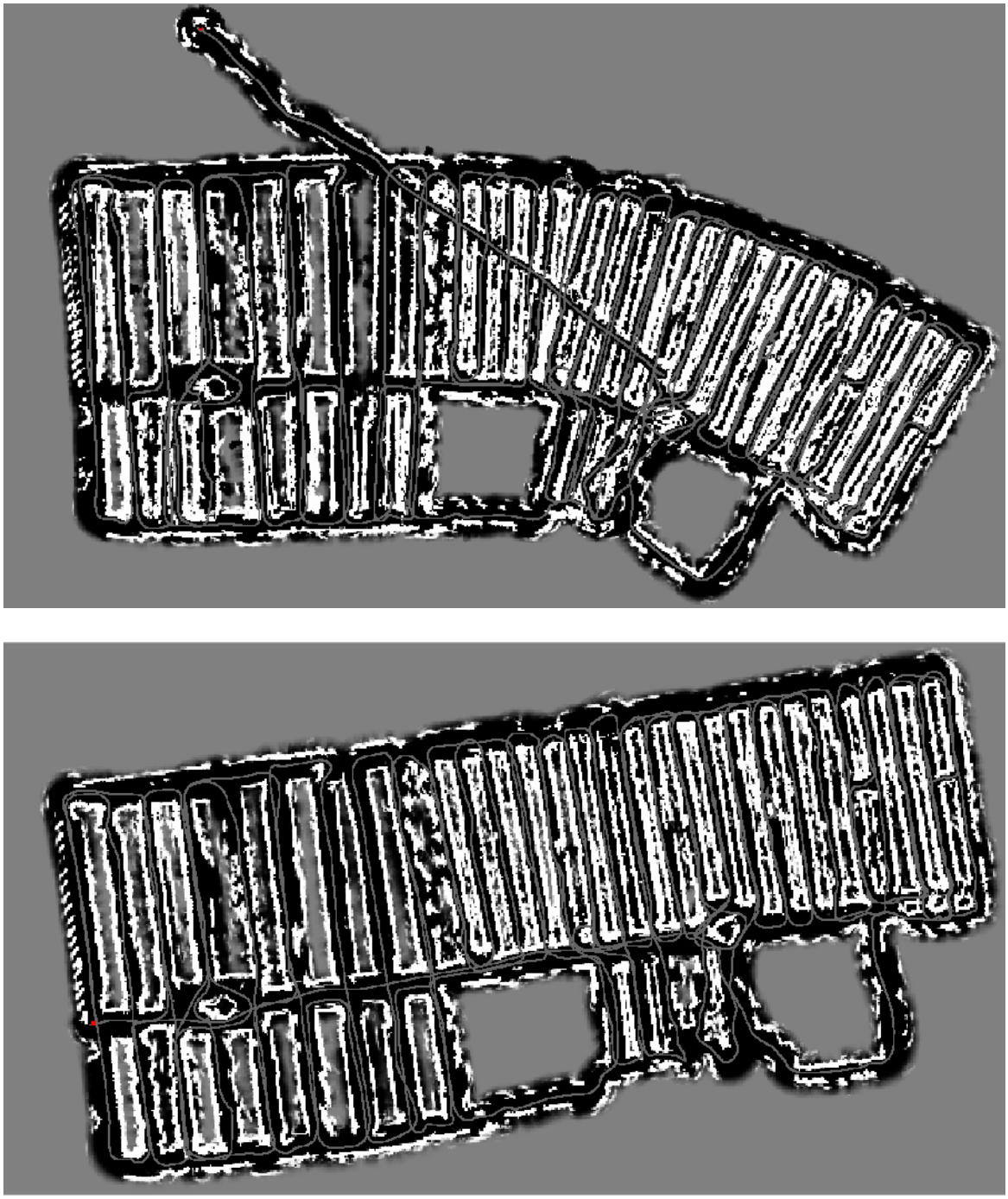

Abbildung 4.23: *Kartierung des gesamten Test-Baumarktes (Fläche ca. 120m × 50m, der obere Bereich besteht aus 29 parallelen Gängen): Odometrie (oben) sowie SLAM-Pfad und -Karte (unten). Die gefahrene Strecke beträgt insgesamt ca. 2300m, und besteht aus über 40 einzelnen Schleifen. Der Verlauf der Partikelanzahl ist in Abb. 4.22 dargestellt. Ein Referenz-Grundriss findet sich in Abb. 3.14*

4.4 Map-Match-SLAM mit visuell detektierten 3D-Punkten

Um die sehr einfache Übertragbarkeit des Map-Match-SLAM-Ansatzes auf andere Sensoren zu zeigen, werden in diesem Abschnitt zwei Möglichkeiten für SLAM mit visueller Sensorik vorgestellt: dazu wird sowohl eine Stereo-Kamera-Anordnung als auch eine einzelne Kamera in Kombination mit einem Depth-from-Motion-Verfahren genutzt, um Gridkarten der Umgebung zu generieren. Gegenüber dem zuvor vorgestellten Algorithmus wird jeweils lediglich das Kartenupdate verändert, um die lokalen und globalen Gridkarten aus den Sensormessungen zu erstellen. Die vorgestellten Experimente beschränken sich dabei auf relativ kleine Umgebungen und sollten lediglich als „Proof-of-Concept“ verstanden werden. Die Anwendung des Map-Match-Konzeptes für SLAM mit visuellen Sensoren ist ebenfalls Inhalt des in Kürze erscheinenden Beitrages zur IROS-Konferenz [Schröter und Gross, 2008].

4.4.1 Map-Match-SLAM mit Stereo-Kamera

Im ersten Beispiel wird eine Stereo-Kamera [Videre Design, 2008] verwendet, welche direkt ein Disparitätsbild aus zwei synchron aufgenommenen Kamera-Bildern berechnet und zusätzlich zum Rechner überträgt. Die Grundlagen der Disparitätsberechnung werden im Anhang (C.1) beschrieben. Über die Kenntnis der Kamera-Abbildung (intrinsische Kamera-Parameter, welche im Rahmen der ohnehin benötigten Stereo-Kalibrierung bestimmt werden) kann das Disparitätsbild in eine 3D-Punktwolke transformiert werden, wobei jeder Punkt einem Pixel im Bild entspricht. Disparität und 3D-Punkte können nur für Objekt-Kanten oder hinreichend strukturierte Oberflächen bestimmt werden (Abb. 4.24).

Die Disparitäts- bzw. Tiefeninformationen der Kamera werden benutzt, um die Zell-Belegtheitswahrscheinlichkeiten innerhalb des Kamera-Blickfelds zu schätzen (siehe Abschnitt 3.1.3), über die Verrechnung vieler Beobachtungen an unterschiedlichen Positionen entsteht auch hier eine globale Karte. Dazu ist die Kenntnis der Position der Kamera im Roboter-Koordinatensystem (extrinsische Kamera-Parameter) nötig, also die Höhe gegenüber der Bodenebene, Neigung und Offset zum Robo-

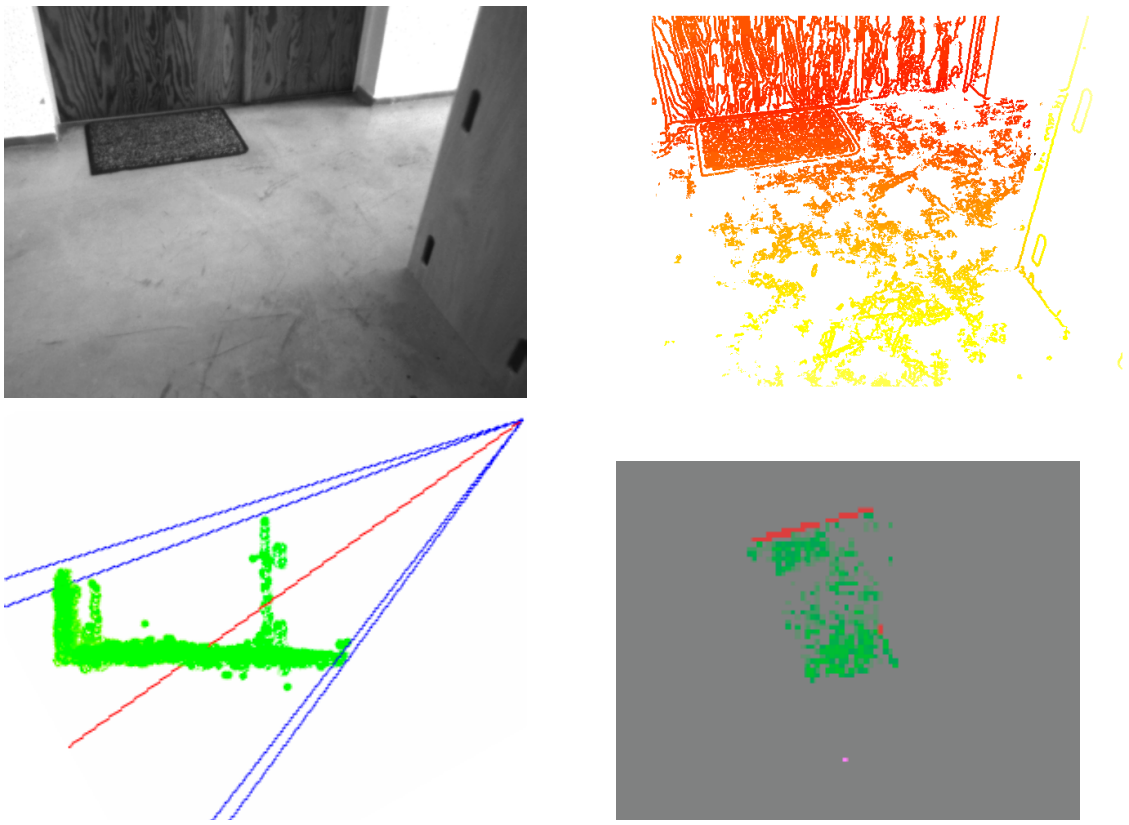

Abbildung 4.24: **Links oben:** *Originalbild einer (der linken) Kamera der Stereo-Anordnung.* **Rechts oben:** *Disparitätsbild - die Disparität ist als Farbverlauf von gelb (geringe Tiefe) über orange zu rot (große Tiefe) dargestellt. An den äußeren Rändern des Ausgangsbildes wird keine Disparität bestimmt: einerseits unterscheidet sich der Sichtbereich der beiden Kameras leicht, dadurch sind Objekte am linken Rand des linken Bildes für die rechte Kamera nicht sichtbar und es kann keine Korrespondenz gefunden werden. Außerdem wird die Disparität mittels Suche nach der Übereinstimmung eines Korrespondenzfensters in beiden Bildern bestimmt, die Bildränder müssen dabei ausgelassen werden, da andernfalls das Vergleichs-Fenster aus dem Bild ragen würde.* **Links unten:** *Durch die Kenntnis der optischen Abbildung kann jeder Pixel mit bekannter Disparität als 3D-Punkt im Kamera-Koordinatensystem dargestellt werden (grün). Hier ist ein Blick von der Seite auf die Szene gezeigt, zusätzlich wird zur Orientierung der Sichtkegel (blau) sowie die optische Achse (rot) der Kamera eingeblendet.* **Rechts unten:** *Durch Kenntnis der extrinsischen Kameraparameter können die 3D-Punkte in das Weltkoordinatensystem überführt (Abb. 4.25 und zur Schätzung von Belegtheitswahrscheinlichkeiten einer Gridkarte benutzt werden (rot = Hindernisse, grün = Freiraum, magenta = Roboterposition).*

terzentrum. Insbesondere Höhe und Neigungswinkel wirken sich stark auf die Belegtheitsschätzungen aus, daher sollten diese recht genau bestimmt werden. Dies geschieht am besten wiederum mit einer einfachen

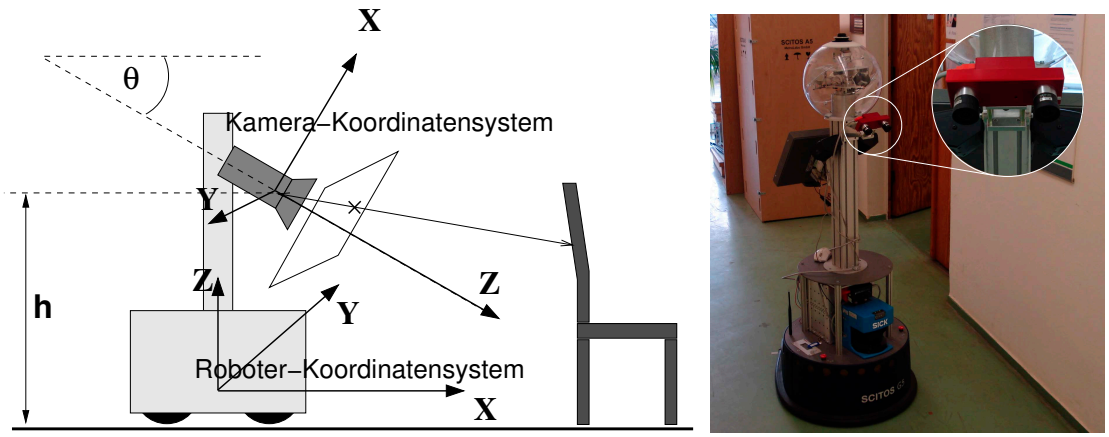

Abbildung 4.25: **Links:** *Die durch die Kamera bestimmten 3D-Koordinaten der Objekte liegen zunächst im Kamera-Koordinatensystem vor. Wenn die relative Lage zwischen Kamera und Roboter (Position des Kamerazentrums und Blickrichtung der Kamera, bezogen auf das Roboter-Koordinatensystem) bekannt ist, können diese 3D-Punkte in das Roboter-Koordinatensystem transformiert werden. Für eine statische Kamera müssen die entsprechenden Daten nur einmal bestimmt (gemessen) werden. Praktisch wurde die Kamera hier so angebracht, dass nur die Höhe über dem Boden h und die Kippung gegenüber der Horizontalen θ bestimmt werden müssen. Dies geschieht anhand einer Kalibrierungs-Aufnahme, welche ausschließlich Untergrund zeigt, durch Suche der Parameter (h, θ), für welche die geschätzten Höhen aller Punkte (quadratische Summe) die geringste Abweichung von 0 ergibt.* **Rechts:** *Der Roboter SCITOS G5 mit der Stereo-Kamera.*

Kalibrierungsaufnahme, in der möglichst viele Punkte über eine große Fläche auf der Bodenebene verteilt liegen, hierfür eignet sich z.B. ein beliebiges auf Papier ausgedrucktes Muster. Da die Kamera eine begrenzte Wahrnehmungstiefe hat (je nach Optik wird die Disparität im Abstand von einigen Metern geringer als ein Bildpixel und kann damit nicht mehr erkannt werden), ist sie schräg angebracht, so dass sie den Boden vor dem Roboter erfassen kann.

Mit der Kenntnis der Kamera-Parameter kann die zunächst im Kamera-Koordinatensystem vorliegende 3D-Punktwolke in das Roboter-Koordinatensystem (Abb. 4.25 und durch Hinzunahme der (jeweils von einem Partikel geschätzten) Roboter-Position in das globale Weltkoordinatensystem transformiert werden. Zur Schätzung der Zellbelegungen wird zunächst für jede Zelle der Umgebung die Menge der darin liegenden Hindernis- und Bodenpunkte bestimmt. Wenn in einer Zelle Hindernispunkte gefunden werden, so wird diese als belegt geschätzt. Falls Bodenpunkte, aber keine Hindernispunkte gefunden werden, wird die

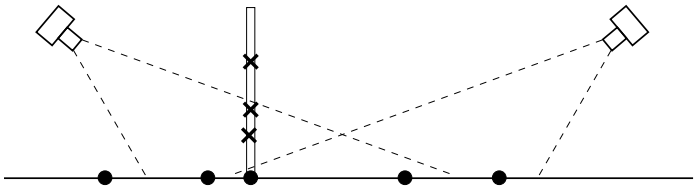

Abbildung 4.26: *Durch die Schrägstellung der Kamera können Hindernisse, welche oberhalb des Sichtbereiches der Kamera liegen, nicht wahrgenommen werden. Eine Position, an der von einer Beobachtungsposition ein Hindernis gesehen wird, kann von einer anderen Position aus als frei detektiert werden. (Das Objekt in der Mitte wird von der linken Position als Hindernis, von der rechten Position aber als Freiraum erkannt.) Es muss daher beim Kartenupdate darauf geachtet werden, dass eine als belegt geschätzte Zelle nicht durch spätere Beobachtungen wieder frei wird.*

Zelle frei geschätzt. Falls weder Hindernis- noch Bodenpunkte in einer Zelle gefunden werden, so liegt keine Information über die Zellbelegung vor, d.h. diese bleibt unverändert.

Die rekursive Verrechnung der Einzelschätzungen für die Zellwahrscheinlichkeiten erfolgt nach der selben Methode (Abschnitt 3.1.2), die auch bei den Sonarsensoren zum Einsatz kam, allerdings mit einer Anpassung: Durch die Schrägstellung der Kamera ist der Sichtbereich nach oben beschränkt, mit zunehmender Entfernung ist eine geringere Höhe im Bild sichtbar. Oft erlauben Hindernisse oder Wände aufgrund der schwachen Strukturierung nicht auf ihrer gesamten Oberfläche eine Bestimmung der Disparität, sondern es werden nur bestimmte Markierungen oder Kanten wahrgenommen. Wenn sich eine Wand an der Grenze des Blickbereiches befindet, ist oft zwar der Übergang zwischen Boden und Wand als klare Kante erkennbar, der obere Bereich der Wand befindet sich aber nicht im Bild. Dies bedeutet, dass an der entsprechenden Position Bodenpunkte, aber keine Hindernispunkte wahrgenommen werden (Abb. 4.26). Erst wenn der Roboter sich weiter nähert, erfasst die Kamera größere Bereiche der Wand, findet markante Features und erkennt ein Hindernis. Dies könnte dazu führen, dass in der Beobachtungssequenz die Zelle zu einer Zeit frei und zu einer anderen Zeit belegt geschätzt wird und damit ihr Gesamtzustand wechseln würde oder unbestimmt wäre. Um zu vermeiden, dass eine belegte Zelle wieder als frei geschätzt wird, sobald die Kamera nur noch den Fußpunkt des entsprechenden Objektes sehen kann, wird das rekursive Update der Zelle verworfen, wenn die Zelle zuvor mit einer gewissen Sicherheit belegt war, die aktuelle Beobachtung aber nur Bodenpunkte sieht. Zwar ist der

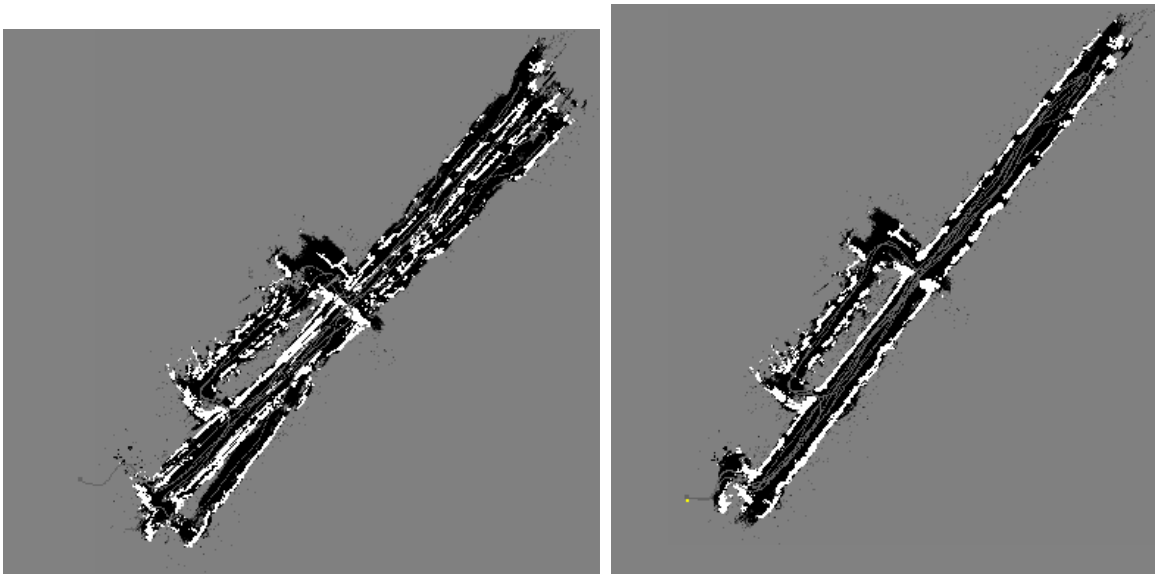

Abbildung 4.27: **Links:** *Karte und Roboterpfad bei Nutzung der Stereo-Kamera-Daten in Kombination mit der unkorrigierten Odometrie.* **Rechts:** *Karte und Pfad, welche mittels SLAM berechnet wurden. Die Umgebung entspricht der bereits in Abb. 3.9 dargestellten, d.h. die Länge des Ganges beträgt auch hier wieder ca. 35m. Vom Roboter wurde insgesamt eine Strecke von ca. 360m zurückgelegt.*

Sichtbereich der Kamera bekannt, aber nicht die Höhe des zuvor eingetragenen Hindernisses (aufgrund der Repräsentation in einer 2D-Karte), so dass nicht entschieden werden kann, ob das Objekt ganz verschwunden oder nur aus dem Sichtbereich herausgetreten ist. Auf tatsächlichen Freiraum, der nie (oder sehr selten durch Fehlmessungen) als belegt erkannt wird, hat dies keine Auswirkungen. Eine Einschränkung besteht allerdings darin, dass dynamische Hindernisse nicht wieder aus der Karte entfernt werden können, sobald sie einmal eingetragen sind. Für die Kartierung ist dies akzeptabel, bei einer Nutzung einer solchen Karte z.B. für eine lokale Hindernisdetektion wird allerdings ein zusätzlicher Vergessens-Mechanismus benötigt.

Abgesehen vom Kartenupdate wird die selbe Systemstruktur und der selbe Algorithmus benutzt, der im vorhergehenden Abschnitt für SLAM mit Sonar-Sensoren beschrieben wurde. Als Kartenauflösung wurde im hier dargestellten Experiment eine Zellgröße von 0.1m gewählt.

4.4.2 Map-Match-SLAM mit Depth-from-Motion

Alternativ zur Tiefenschätzung aus den Bildern zweier gleichzeitig die Szene betrachtenden Kameras können auch die zeitlich aufeinander folgenden Bilder einer einzelnen, sich bewegenden Kamera genutzt werden, um die 3D-Positionen der sichtbaren Punkte im Raum zu bestimmen. Ein dazu implementiertes Verfahren [Einhorn et al., 2007a] wird ebenfalls im Anhang (C.2) beschrieben. Das Verfahren beruht darauf, dass gut lokalisierbare Punkte im Bild detektiert und über nachfolgende Bilder verfolgt werden. Unter Nutzung der Odometrie als Initialschätzung der Kamerabewegung wird die Relativposition der Punkte zum Roboter im 3D-Raum mittels einzelner Kalmanfilter bestimmt (Abb. 4.28). Allerdings werden die Feature-Punkte nur so lange verfolgt, wie sie jeweils im Bild wiedergefunden werden können: Es gibt keine global eindeutige Identifikation der Feature-Punkte und demzufolge keine Wiedererkennung bei einem Kreisschluss, insofern handelt es sich nicht um ein „klassisches“ Visual-SLAM-Verfahren.

Statt dessen wird hier wiederum aus den 3D-Koordinaten eine 2D-Gridkarte erstellt. Im Gegensatz zum Stereo-Verfahren mit zwei Kameras werden nur solche Features detektiert, die sich über mehrere Bilder hinweg relativ eindeutig zuordnen lassen. Es werden daher pro Bild deutlich weniger Punkte gefunden und deren Tiefe geschätzt. Im Gegensatz zum sehr dichten Tiefenfeld der im vorhergehenden Abschnitt benutzten Stereo-Kamera wird eine spärliche 3D-Punkt-Wolke erzeugt. Diese genügt jedoch i.A., um kleinere Objekte mit erkennbaren Begrenzungskanten oder strukturierte Flächen als Hindernisse zu erkennen. Da aufgrund der geringen Strukturiertheit in der Testumgebung kaum Features auf dem Boden detektiert werden konnten, wurde auf eine Schätzung von Freiraum komplett verzichtet und lediglich Punkte oberhalb einer Schwellenhöhe als Hindernisse in die Karte eingetragen. Der Anstellwinkel der Kamera konnte dementsprechend geringer gewählt werden, so dass insgesamt der Sichtbereich höhere Objekte erfasst und weiter in den Raum reicht. Allerdings wird die Schätzung der Punkt-Position aufgrund der geringen relativen Bewegung im Bild (bei fester Diskretisierung durch das Pixelraster) mit steigender Entfernung zur Kamera ungenauer. Da für die Kartierung nur Punkte in einem relativ geringen Abstand zum Roboter genutzt werden, hat dies aber auf die entstehende Karte kaum Einfluss.

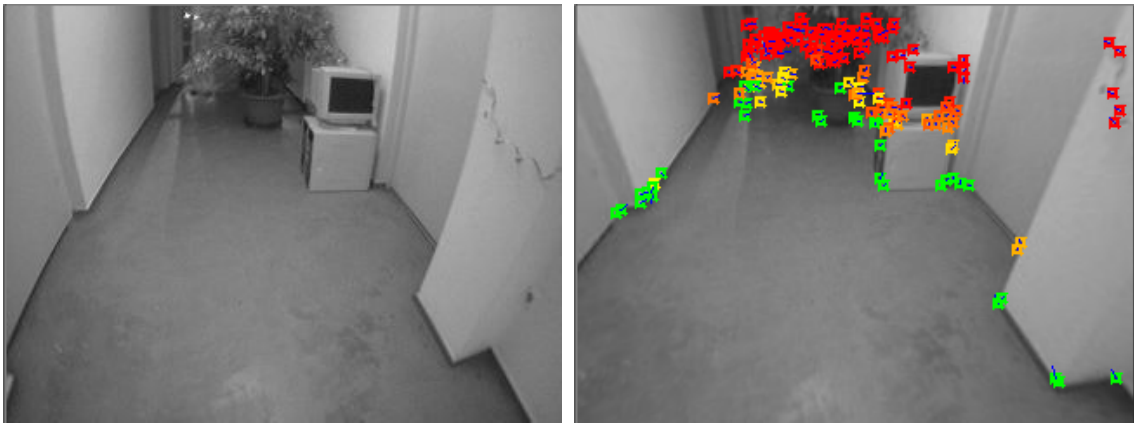

Abbildung 4.28: **Links:** *Ein Originalbild aus der Sequenz der Kamera-Aufnahmen.* **Rechts:** *Das Bild mit überlagerter Darstellung der detektierten Features. Die Farbe der Markierungspunkte kodiert die geschätzte Höhe der Punkte im Raum (von unten nach oben: grün-gelb-orange-rot). Aufgrund der vor der Feature-Detektion erfolgten Korrektur der Kamera-Verzerrung ist der Bildausschnitt leicht unterschiedlich, obwohl es sich um die selbe Aufnahme handelt.*

Auch hier wurde, abgesehen vom Kartenupdate, der Map-Match-SLAM-Algorithmus unverändert genutzt. Da kein Freiraum in die Karte eingetragen wird, ist die Karteninformation deutlich unschärfer und damit die Bestimmung der jeweils korrekten Position schwieriger, dennoch gelingt eine konsistente Rekonstruktion des Roboterpfades (Abb. 4.29).

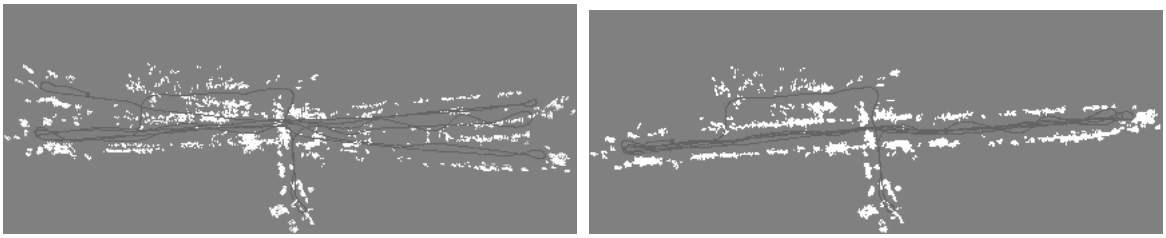

Abbildung 4.29: **Links:** *Karte und Roboterpfad mit einer einzelnen Kamera nach Odometrie (selbe Umgebung wie Abb. 4.27, Länge des Ganges ca. 35m).* **Rechts:** *Karte und Pfad nach dem SLAM-Algorithmus. Die gesamte Fahrstrecke beträgt knapp 300m.*

4.5 Der SLAM-Assistent

Die beschriebene Lösung des SLAM-Problems mit einem Rao-Blackwellized Particle Filter (RBPF) beruht auf der Approximation der Wahrscheinlichkeitsverteilung des Zustandes durch eine Menge von Partikeln. Prinzipiell sind Partikelfilter nur dann gut zur Approximation des Zustandes geeignet, wenn zu jedem Zeitpunkt genügend Partikel zur Verfügung stehen. Im Abschnitt 4.3.4 wurde daher eine Methode vorgestellt, die Anzahl der jeweils benötigten Partikel zu bestimmen und entsprechend viele Partikel im Resampling zu generieren. In einer konkreten Implementation muss jedoch mit Rücksicht auf den Speicher- und Rechenzeitbedarf die maximale Anzahl an Partikeln beschränkt werden. Bei großer Zustands-Unsicherheit ist es daher trotz dynamischer Anpassung möglich, dass die Anzahl der generierten Partikel nicht ausreicht, um die Wahrscheinlichkeitsverteilung des Zustandes mit hinreichender Genauigkeit zu beschreiben, was in vielen Fällen dazu führt, dass der Partikelfilter nicht zur korrekten Kartenschätzung konvergiert.

Um dies zu verhindern, sollte die Zustands-Unsicherheit während der Fahrt dadurch beschränkt werden, dass zu große offene Schleifen vermieden werden. Dies bedeutet, dass der Roboter nach einer gewissen Fahrstrecke zu einer bereits bekannten Position zurückfahren sollte, um einen Kreisschluss durchzuführen und damit die Unsicherheit zu verringern.

Aus verschiedenen Gründen ist jedoch nicht vorgesehen, dass der Roboter eine Umgebung vollkommen autonom exploriert und kartiert: es kann zum Beispiel auf diese Weise nicht sichergestellt werden, dass die Karte das Gebiet komplett erfasst, wenn zum Kartierungszeitpunkt bestimmte Wege durch Hindernisse blockiert waren. Weiterhin existieren in nicht speziell für den Robotereinsatz vorbereiteten Umgebungen (z.B. der Baumarkt für den Einsatz des Shopping-Assistenten) in Einzelfällen Hindernisse, die vom Roboter selbst mit seiner Sensorik kaum wahrgenommen werden können, insbesondere Bodenschwellen und -unebenheiten. Da diese nur räumlich begrenzt auftreten, ist es bei vorhandener Karte mit geringem Aufwand möglich, die entsprechenden Gebiete als unzugänglich zu markieren, so dass der Roboter nicht mit solchen „unsichtbaren“ Hindernissen in Kontakt kommt. Während der Kartierung können sie aber eine Gefahr darstellen.

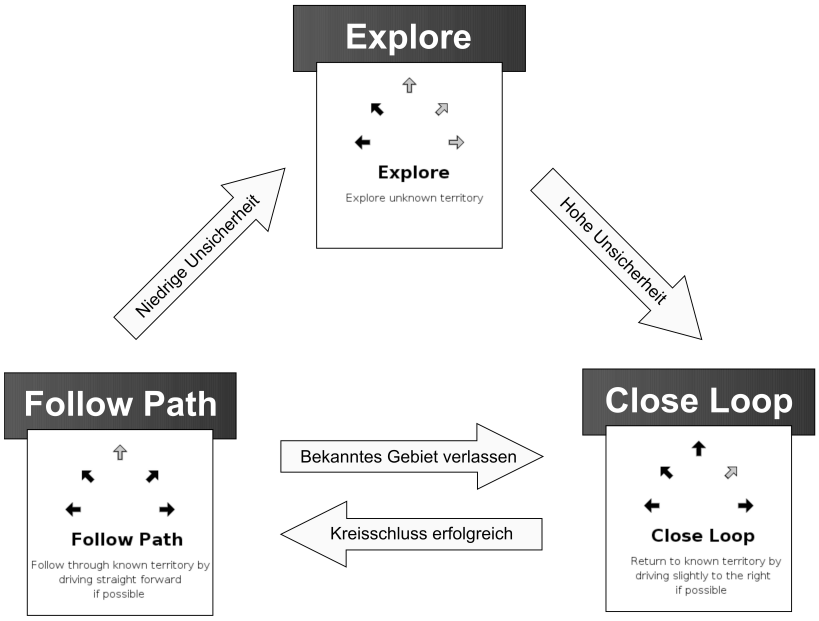

Abbildung 4.30: *Die 3 Zustände des SLAM-Assistenten mit Übergangsbedingungen.*

Es wird daher davon ausgegangen, dass die initiale Kartierung einer Umgebung stets durch eine von Hand gesteuerte Fahrt geschieht. In diesem Fall ist der menschliche Bediener für die Wahl des Kartierungspfades und damit auch für die Einhaltung der Unsicherheitsgrenzen verantwortlich. Um die internen Zustände des SLAM-Systems zu veranschaulichen und auch mit der Methode nicht vertrauten Nutzern eine optimale Kartierung zu ermöglichen, sollen Handlungsvorschläge in Form von empfohlenen Fahrtrichtungen vom System generiert und dem Nutzer präsentiert werden, der dann selbst entscheidet, ob und wie er diese ausführen kann. Dazu wurde als Erweiterung zum SLAM-Algorithmus im Rahmen einer Diplomarbeit [Keichel, 2007] ein so genannter SLAM-Assistent entwickelt, welcher diese Aufgabe erfüllt.

Wie bereits beschrieben, sollte ein Kreisschluss dann angestrebt werden, wenn die Zustandsunsicherheit zu groß wird. Eine freie Erkundung der Umgebung, mit dem Ziel, neues Gebiet zu kartieren, kann fortgesetzt werden, sobald die Unsicherheit unter eine ebenfalls zu definierende untere Schwelle fällt. Um dies zu realisieren, werden im SLAM-Assistenten unterschiedliche Phasen während der Kartierung und Übergangsbedingungen zwischen diesen definiert.

- Initial befindet sich der Assistent im **Explore**-Modus. Hier ist eine freie Bewegung möglich, um zuvor unbekanntes Gebiet zu erkunden und zu kartographieren.

- Vom Explore-Modus soll in den **Close Loop**-Modus umgeschaltet werden, sobald die Unsicherheit so groß wird, dass eine nicht mehr zu bewältigende Menge an Partikeln (Rechenzeit, Speicherbedarf) zur Modellierung benötigt würde. Dazu wird die im Resampling implizit bestimmte Anzahl der benötigten Partikel ausgewertet. Es wird also eine Partikelzahl als Schwelle festgelegt, bei der vom Explore- in den Close Loop-Modus gewechselt wird. In diesem Modus wird angestrebt, auf möglichst kurzem Weg zu einer bekannten Position zurückzukehren, um einen Kreisschluss zu erreichen.

- Ein Kreisschluss gilt als vollzogen, sobald ein gewisser Anteil (i.d.R. mindestens 50 %) der Partikel auf bekanntes Gebiet zurückgekehrt sind. Typischerweise muss der Roboter sich jedoch eine gewisse Zeit in bekanntem Gebiet bewegen, so dass über mehrere aufeinander folgende Verteilungsupdates die Zustands-Unsicherheit signifikant verringert wird. Deshalb wird weiterhin ein **Follow Path**-Modus eingeführt, im dem nach dem Kreisschluss so lange ein bekannter Pfad verfolgt wird, bis die Unsicherheit genügend verringert wurde, d.h. bis die benötigte Partikelzahl wiederum kleiner als ein Schwellwert für den Übergang in den Explore-Modus ist. Falls im Follow-Path-Modus das bekannte Gebiet verlassen wird, ist zunächst erneut ein Kreisschluss notwendig.

In jedem Modus werden vom Assistenten Vorschläge für die Fahrtrichtung berechnet und als grobe Richtung zwischen „scharf links“, „leicht links“, „geradeaus“, „leicht rechts“ und „scharf rechts“ angezeigt. Eine Vorschlagsrichtung entgegen der aktuellen Blickrichtung (also Umkehr) wird ebenfalls als „scharf rechts“ bzw. „scharf links“ angezeigt, bis der Roboter sich entsprechend weit gedreht hat.

Um die Berechnung von Vorschlägen zum Kreisschluss und zur Pfadverfolgung zu ermöglichen, wird zusätzlich zur Gridkarte eine topologische Karte der Umgebung erstellt. Diese wird ebenso wie die Gridkarte während des Kartenupdates aktualisiert. Die topologische Karte ist ein ungerichteter Graph aus Knoten und Kanten. Für die Knoten ist eine Minimalentfernung d_{thresh} definiert, es wird jeweils dann ein

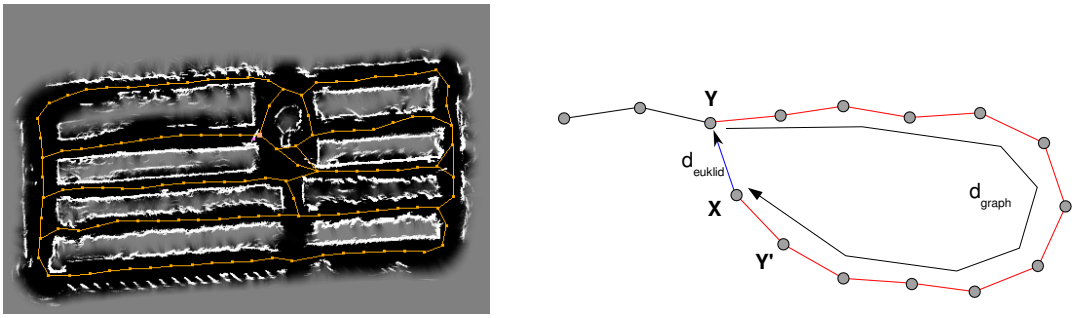

Abbildung 4.31: **Links:** *Für den SLAM-Assistenten enthält jedes Partikel neben der Gridkarte zusätzlich eine topologische Karte: Die topologische Karte bildet einen Graphen aus Knoten (Positionen) und Kanten (tatsächlich befahrene Verbindungen zwischen Positionen). Sie wird benutzt, um im Assistenten Kreisschluss-Möglichkeiten und bekannte Pfade zu suchen.* **Rechts:** *Für die Bestimmung einer Kreisschluss-Möglichkeit wird zum aktuellen Knoten X der Knoten Y mit der minimalen euklidischen Distanz d_euklid gesucht. Gleichzeitig muss zwischen X und Y die Länge der kürzesten Verbindung im Graphen d_graph eine festgelegte Schwelle überschreiten, um ein Minimallänge der Schleife zu gewährleisten. Durch diese Bedingung wird die Auswahl der Rückkehr zum Vorgängerknoten Y' verhindert.*

Knoten an der aktuellen Position eingefügt, wenn die euklidische Distanz zum nächsten Knoten d_{min} mehr als d_{thresh} beträgt. Andernfalls gilt der nächste Knoten als aktuell besuchter Knoten. Eine Kante wird immer dann zwischen 2 Knoten u und v eingefügt, wenn der Roboter sich unmittelbar von u nach v bewegt hat (Abb. 4.31).

Im Explore-Modus kann der Nutzer frei die Umgebung erkunden, deshalb wird hier die topologische Karte nicht benötigt. Es wird lediglich versucht, aus der lokalen Karte die befahrbaren Richtungen zu detektieren und anzuzeigen. Dazu wird ausgewertet, ob die lokale Karte in der jeweiligen Richtung ein Hindernis enthält. Aufgrund des begrenzten Sichtbereiches und relativ großer Messvarianzen (welche aufgrund der geringen Zahl an Beobachtungen gerade für den Bereich vor dem Roboter starke Auswirkungen haben können) ist jedoch nicht immer eine zuverlässige Erkennung möglich.

Im Close Loop-Modus wird mit Hilfe der topologischen Karte eine Möglichkeit zum Kreisschluss bestimmt. Dazu wird ein Knoten gesucht, welcher eine möglichst geringe euklidische Distanz zur aktuellen Position hat, gleichzeitig aber innerhalb des Graphen eine signifikant größere Entfernung zum aktuellen Knoten aufweist (Abb. 4.31 rechts). Diese

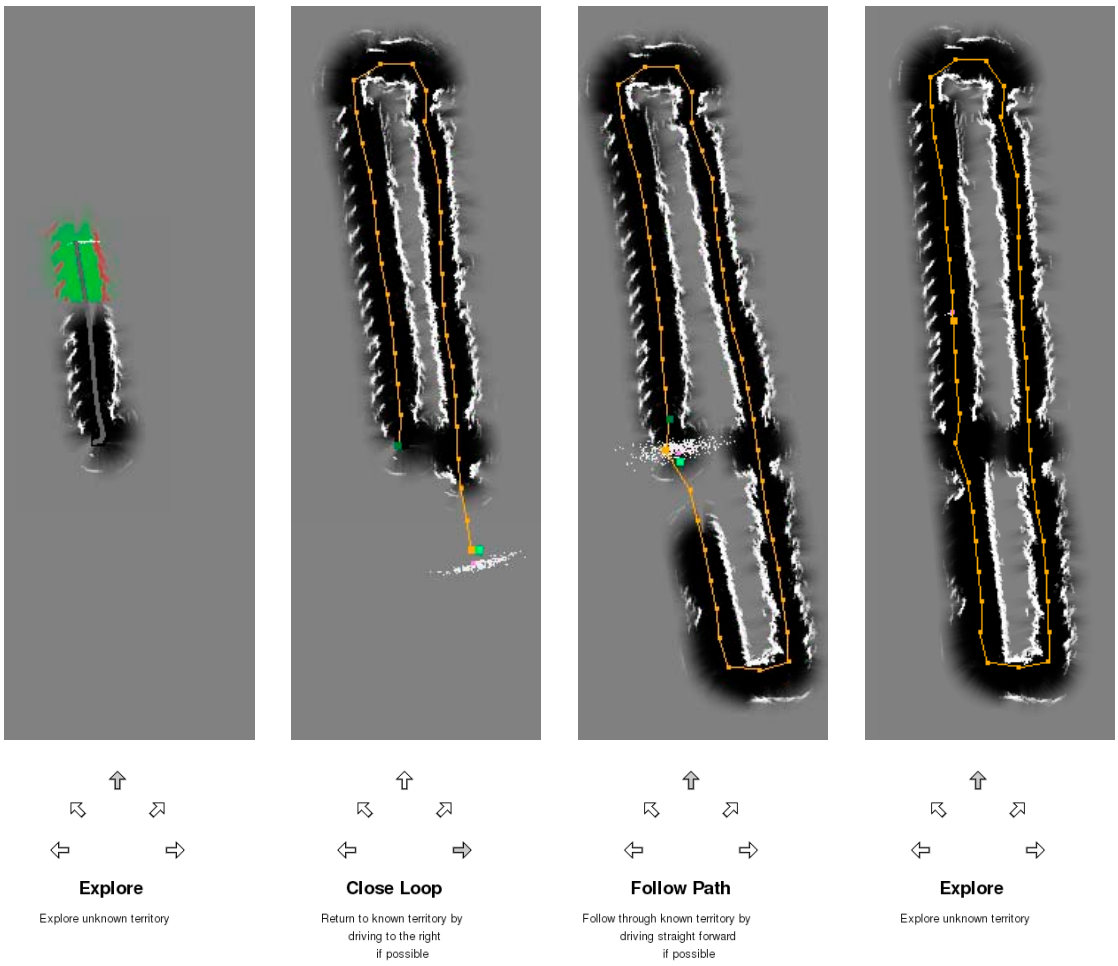

Abbildung 4.32: *Beispiel für den Modus-Übergang von Explore über Close Loop zu Follow Path und zurück zu Explore: dargestellt ist neben der Karte jeweils die Ausgabe des Assistenten.*

Suche wird jeweils in den topologischen Karten aller einzelnen Partikel durchgeführt, die Richtung wird dann über alle Partikel gemittelt. Da die direkte Route zwischen der aktuellen und der angestrebten Position zuvor nicht erkundet wurde (was aus der Entfernungsbedingung folgt), kann aber nicht sichergestellt werden, dass überhaupt eine direkte Verbindung existiert. Die angezeigte Richtung stellt somit nur einen Hinweis dar, der Nutzer muss selbst entscheiden, auf welche Weise dieser befolgt werden kann. Unter Umständen ist eine andere Kreisschluss-Möglichkeit in der tatsächlichen Umgebung besser zu erreichen. Die Vorschläge werden während der Bewegung kontinuierlich neu berechnet, daher wird automatisch eine neue Richtung angezeigt, wenn der Roboter sich einer anderen Möglichkeit zum Kreisschluss annähert.

Im Follow Path-Modus ist bereits ein Kreisschluss erfolgt, d.h. der Roboter ist an eine bekannte Position zurückgekehrt. Die topologische Karte wird hier benutzt, um den Roboter möglichst lange in bekanntem Gebiet zu halten. Dazu wird im Graphen jeweils der Pfad von einem Knoten zum nächsten verfolgt. An Abzweigungen (Knoten mit mehr als einem weiterführenden Nachbarn) werden solche Zweige vermieden, die eine Mindestlänge unterschreiten, um nicht in eine Sackgasse zu geraten. Weiterhin werden die Zweige bevorzugt, die während der bisherigen Erkundung weniger oft befahren wurden, um eine gleichmäßige Abdeckung anzustreben. Zu diesem Zweck enthält jeder Knoten einen „Besuchszähler“, der angibt, wie oft der Roboter diese Position zuvor passiert hat.

Ein Minimal-Beispiel für einen vollständigen Übergang zwischen allen Zuständen ist in Abb. 4.32 dargestellt. Die Abbildungen 4.33 bis 4.35 zeigen die Funktion des SLAM-Assistenten während einer längeren Kartierungsfahrt in einer realen Umgebung.

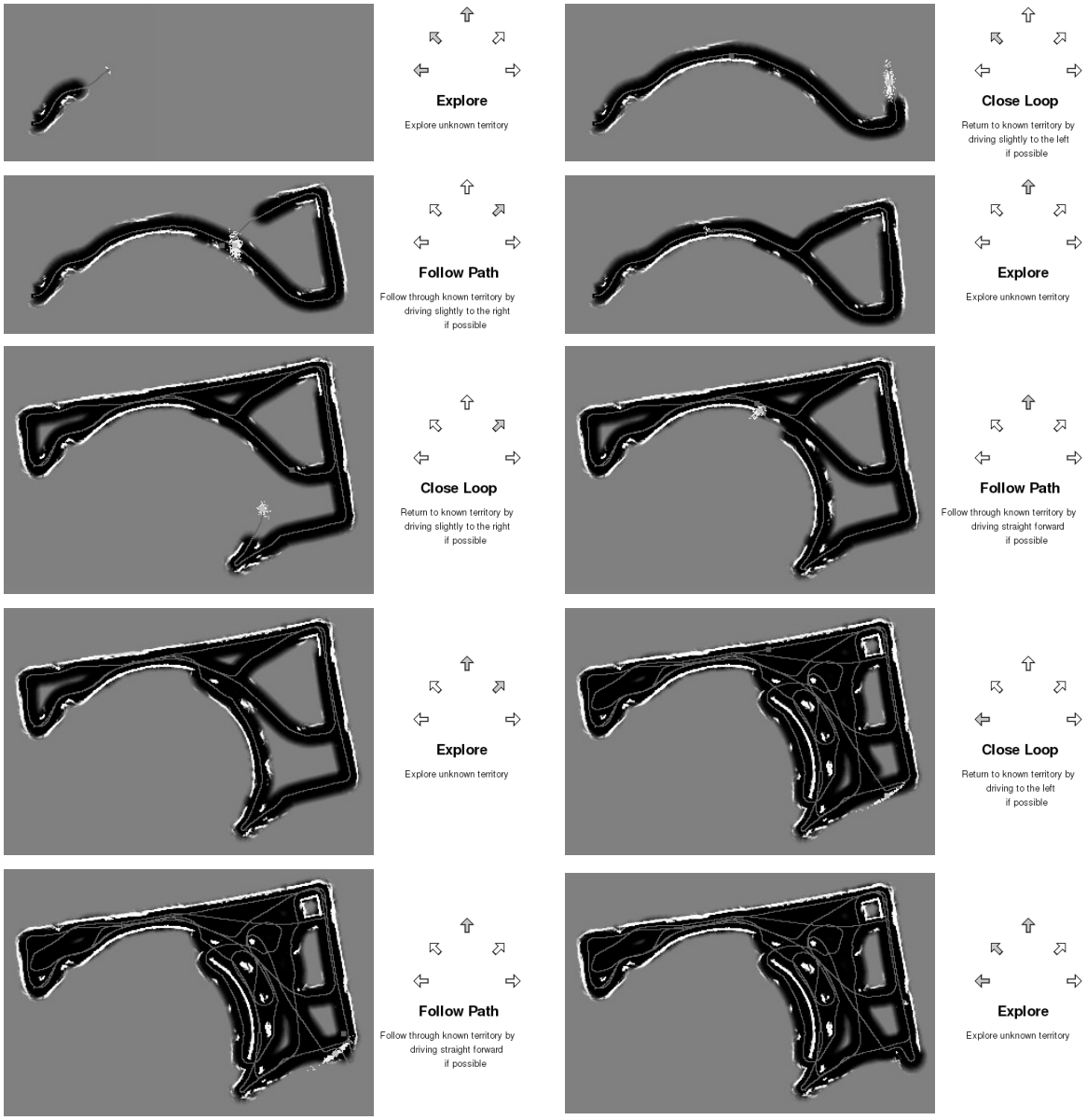

Abbildung 4.33: *Eine Kartierungssequenz mit dem Assistenten (zeilenweise von oben nach unten). Dargestellt sind jeweils die Karte (Gesamtareal ca. 50m × 40m) und der geschätzte Roboter-Pfad für ein einzelnes Partikel sowie rechts daneben die Ausgabe des Assistenten zum entsprechenden Zeitpunkt. Die während der Kartierung gefahrene Strecke beträgt ca. 480m. Man sieht, wie mehrfach bei Erreichen der Unsicherheitsschwelle in den Modus „Close Loop“ , anschließend beim Kreisschluss in den Modus „Follow Path“ und schließlich nach ausreichender Verringerung der Zustandsunsicherheit zurück in den Modus „Explore“ umgeschaltet wird. Der vollständige Verlauf der Partikelzahl und der Assistenten-Modi ist in Abb. 4.35 dargestellt.*

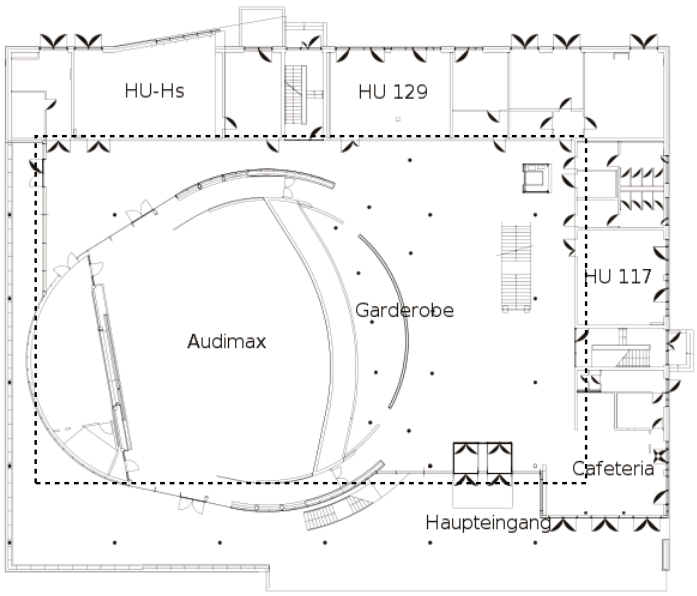

Abbildung 4.34: *Zum Vergleich der Gebäude-Grundriss: Der in der Sequenz in Abb. 4.33 kartierte Bereich ist mit dem gestrichelten Rahmen markiert.*

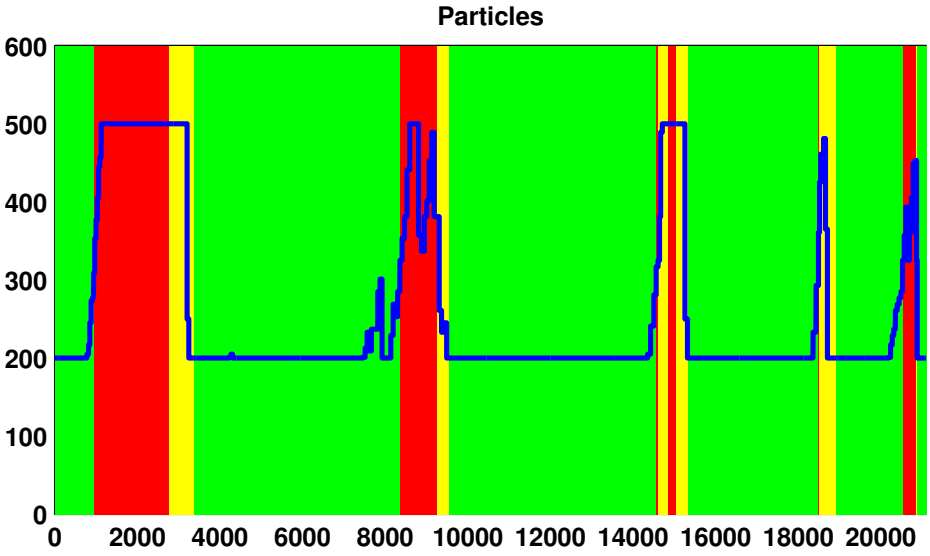

Abbildung 4.35: *Der vollständige Verlauf der Partikelzahl und des Assistentenzustandes während der in Abb. 4.33 gezeigten Kartierung (grün = Explore, rot = Close Loop, gelb = Follow Path). Die Partikelzahl war auf minimal 200 und maximal 500 begrenzt. Horizontal sind sind hier wieder die Updateschritte des Partikelfilters (Bewegungs-, Karten- und Verteilungsupdates) in fortlaufender Numerierung angetragen.*

4.6 Zusammenfassung und Bewertung des Map-Match-SLAM

Mit dem Map-Match-SLAM wurde in diesem Kapitel ein universeller SLAM-Algorithmus auf Basis des Rao-Blackwellized Partikelfilters vorgestellt. Die Besonderheit des Ansatzes besteht darin, dass neben der globalen Karte jeweils eine lokale Karte aus aus den Sensor-Beobachtungen erstellt wird. Die Bewertung der einzelnen Zustands-Hypothesen, welche in Form von Partikeln im Partikelfilter vorliegen, geschieht dann jeweils durch Vergleich der lokalen und globalen Karte. Dieser Vergleich ersetzt die in der mathematischen Formulierung geforderte Berechnung der Wahrscheinlichkeit jeder Beobachtung und stellt insofern eine Vereinfachung und keine exakte Lösung dar.

Die ursprünglich in der Berechnung der Beobachtungswahrscheinlichkeit enthaltene Berücksichtigung der spezifischen Charakteristik des Sensors/Beobachtungssystems wird durch die Erstellung der lokalen Karte in das Kartenupdate verschoben. Der Aufbau einer lokalen Karte kann hier als Vorverarbeitungsschritt betrachtet werden. Er erfüllt damit eine ähnliche Funktion wie z.B. ein Feature-Extraktor, welcher aus den rohen Sensormesswerten abstraktere Informationen in Form von Feature-Positionen berechnet. Im Unterschied zu diesen ist das Konzept der lokalen Gridkarte aber universell einsetzbar und recht einfach auf viele unterschiedliche Sensoren übertragbar.

Ein großer Vorteil dieses Ansatzes ist, dass das Sensormodell $p(o|x, m)$ nicht benötigt wird, um die Beobachtungen zur Bewertung der Zustandshypothesen zu nutzen. Dadurch ist Map-Match-SLAM auch für solche Sensoren geeignet, für die ein solches Modell nur mit großem Aufwand oder gar nicht zu bestimmen ist. Statt dessen wird implizit (durch die Berechnung der lokalen Karte) auch für die Gewichtsberechnung das inverse Modell $p(m|x, o)$ benutzt, welches schon bei der Erstellung der globalen Karte angewendet wird. Dieses ist meist einfacher zu bestimmen oder zumindest heuristisch zu lösen (wie sich z.B. bei der Transformation der Disparitätswerte einer Stereokamera in eine Gridkarte zeigt).

In diesem Kapitel wurde gezeigt, wie sich durch den Ansatz des Map Matching im Partikelfilter das SLAM-Problem mit verschiedenen Sensoren auf Basis eines einheitlichen Frameworks lösen lässt. Während

die meisten existierenden SLAM-Ansätze entweder die Berechnung von Gridkarten mittels hoch aufgelöster Laser-Entfernungsscans oder von Feature-Karten aus visuellen Inputs zum Ziel haben, wurden hier Gridkarten sowohl auf Basis von Sonar-Messungen (welche in der räumlichen Auflösung und der Genauigkeit Laser-Messungen deutlich unterlegen sind) als auch aus binokularen (Stereo) und monokularen Kamerabildern erzeugt.

Es ergeben sich allerdings auch Probleme, die mit anderen Verfahren, welche die besonderen Eigenschaften der jeweiligen Sensoren stärker ausnutzen, besser zu lösen sind. So ist z.B. der Match-Wert zwischen der lokalen und globalen Karte eigentlich nicht sehr gut als Bewertungsmaß geeignet, da er nicht stetig mit dem steigenden Versatz zwischen den Kartenpositionen abfällt. Statt dessen können kleine Positionsunterschiede zu großen Änderungen des Match-Wertes führen, andererseits kann ein sehr großer Positionsfehler unter Umständen den selben Wert erzeugen wie eine äußerst geringe, gerade noch bemerkbare Abweichung. Der Match-Wert stellt also nicht wirklich ein Maß des Abstandes zwischen optimaler Position und bewerteter Position dar. Dies trägt dazu bei, dass eine höhere Anzahl an Partikeln benötigt wird, da möglichst viele Partikel bei einem Kreisschluss in unmittelbarer Nähe der korrekten Position liegen sollten.

Eine Alternative zum einfachen Matching der lokalen und globalen Karte bestünde zum Beispiel darin, eine der beiden Karten zunächst mittels einer Faltungsoperation zu dilatieren, so dass ein (nach außen abfallender) „Saum“ um die Hinderniszellen entstünde. Damit würde der Match-Wert bei zunehmender Positionsverschiebung kontinuierlich abnehmen, allerdings geht dies wiederum zu Lasten der Rechenzeit, außerdem wird auf diese Weise auch das Maximum der Korrelation weniger eindeutig und damit ist zu erwarten, dass insgesamt die geschätzte Verteilung breiter wird. Als eine andere Alternative wäre eine Anwendung des Iterative Closest Point (ICP) [Chen und Medioni, 1991] [Besl und McKay, 1992], eines Standard-Algorithmus zum Scan Matching auf Laser-Messungen, mit den Zellen der Karte denkbar. Hier wäre allerdings der Aufwand noch höher und der Erfolg erscheint wegen der Auflösungsbeschränkung der Gridkarte fraglich.

Generell stoßen Partikelfilter bei der Modellierung großer Unsicherheiten an ihre Grenzen. Dies macht sich hier durch die Nutzung der reinen Odometrie für die Bewegungsschätzung innerhalb einer Schleife beson-

ders bemerkbar. Bei größeren Schleifen werden dadurch selbst unter Annahme eines relativ scharfen Bewegungsmodells extrem viele Partikel benötigt, um die Positionsunsicherheit mit ausreichender Genauigkeit modellieren zu können. Eventuell könnte die in [Grisetti et al., 2005] vorgeschlagene Lösung, die Vorschlagsverteilung aus dem Beobachtungsmodell zu generieren, zur Verringerung dieses Problems beitragen. Allerdings wäre dies im vorgestellten Map-Match-SLAM-Framework nur beim bzw. nach einem Kreisschluss sinnvoll, da vorher die lokale und globale Karte nicht überlappen und damit eine Gleichverteilung generiert würde (da an keiner Position eine Bewertung möglich ist). Weiterhin würde dies wiederum ein Suchproblem ergeben, ähnlich dem in Abschnitt 4.1 vorgestellten Algorithmus, allerdings mit dem Unterschied, dass auf der berechneten Korrelationsfunktion nun viele Hypothesen verfolgt würden. Daraus würde letztendlich auch eine entsprechende Erhöhung der Rechenzeit resultieren, die gegen die mögliche Verringerung der Partikelanzahl aufgewogen werden muss.

Kapitel 5

Autonome Navigation

Nachdem in den Kapiteln 3 und 4 Verfahren zur Umgebungskartierung vorgestellt worden sind, soll in diesem Kapitel auf die eigentliche autonome Funktion des Roboters eingegangen werden. Die Fähigkeit zur Kartierung bildet dabei eine Grundlage für einige Teilfunktionen: Für die Selbstlokalisation sowie für die Pfadplanung zur Zielanfahrt wird die zuvor mittels SLAM erstellte globale Karte der Umgebung genutzt. Für die lokale Navigation (Bewegungssteuerung und Kollisionsvermeidung) wird eine lokale Karte genutzt, die kontinuierlich mit neuen Beobachtungen aktualisiert wird.

In diesem Kapitel werden zunächst in einem kurzen Überblick die Modelle und Algorithmen vorgestellt, die der Roboter für die Selbstlokalisation während des autonomen Einsatzes verwendet. Eine detailliertere Darstellung wird in naher Zukunft in der Dissertation von Alexander König vorliegen, in der die Selbstlokalisation einen wesentlichen Bestandteil bildet.

Anschließend wird das Navigationssystem für die autonome Bewegung in der Einsatzumgebung beschrieben, welches als wesentliche Teile die Pfadplanung und die Bewegungssteuerung enthält. Dabei werden jeweils verschiedene implementierte und getestete Verfahren vorgestellt und auf Basis experimenteller Ergebnisse bewertet.

5.1 Selbstlokalisation

Im vorhergehenden Kapitel wurde ein Verfahren für Simultaneous Localization and Mapping vorgestellt (SLAM), welches eine Gridkarte der Umgebung erstellt und dabei auch kontinuierlich die Position des Roboters in der Umgebung schätzt. Auch während des autonomen Einsatzes muss der Roboter ständig seine eigene Position bestimmen, um z.B. in der Lage zu sein, einen Pfad zu einer Zielposition zu planen und sich entlang des Pfades zielgerichtet zu bewegen. Prinzipiell wäre es dazu möglich, die SLAM-Routine einfach auch während der autonomen Bewegung kontinuierlich weiterlaufen zu lassen. Da zuvor die gesamte Einsatzfläche bereits kartiert wurde, befindet sich der Algorithmus im Gegensatz zur initialen Kartierung niemals in einer offenen Schleife, eine Selbstlokalisation ist meist recht eindeutig möglich. Gleichzeitig würde die Karte ständig an wahrgenommene Änderungen der Umgebung adaptiert.

Ein solcher Ansatz birgt allerdings im praktischen Einsatz auch gewisse Risiken: Während der SLAM-Algorithmus eine statische Umgebung annimmt (was über einen kurzen Zeitraum normalerweise gewährleistet werden kann), so gilt dies bei einem langfristigen Einsatz nicht mehr. Hier kann es geschehen, dass leichte Fehllokalisierungen z.B. als Folge von Sensorverdeckungen oder von tatsächlichen Umgebungsänderungen (temporär abgestellte Objekte etc.) auftreten. Diese hätten durch eine sofortige Kartenadaption u.U. auch dann permanente Auswirkungen auf die Karte, wenn die Lokalisationsfehler nur räumlich und zeitlich begrenzt auftreten und später (an anderer Position) problemlos automatisch korrigiert werden.

Gleichzeitig ist es auf diese Weise schwierig, die Geschwindigkeit der Anpassung zu wählen: Manche wahrgenommene Umgebungsänderungen sollten so schnell wie möglich in der Karte reflektiert werden, wie beispielsweise die Blockierung eines Ganges durch ein Hindernis. Andere, die z.B. durch Personen hervorgerufen werden, sollten nicht die Karte verändern, da mit hoher Wahrscheinlichkeit bei einer späteren Beobachtung die Person sich nicht mehr an der selben Stelle befindet. Selbst wenn Objekte durch Erkennungstechniken klassifiziert werden können (Objekte in Bewegung, Personen, etc.), so kann diese Unterscheidung nicht zuverlässig anhand der ersten Beobachtung getroffen werden, erst bei späterer erneuter Betrachtung der selben Position wird erkennbar,

ob es sich um eine kurzfristige oder langfristige Änderung handelt. Aus der ersten Beobachtung einer Diskrepanz zwischen Karte und Umgebung ergeben sich also eigentlich 2 unterschiedliche Hypothesen für den Zustand bei der nächsten Beobachtung. Dies kann zum Beispiel durch die Verwendung mehrerer Karten, evtl. mit unterschiedlichen Adaptionsgeschwindigkeiten, realisiert werden. Der SLAM-Algorithmus ist dazu nicht ohne Weiteres geeignet. Diese Fragen werden in den Abschnitten 5.4 und 7.2 aufgegriffen.

Ein weiterer, pragmatischer Grund, nicht einen kompletten SLAM-Algorithmus für die Selbstlokalisation einzusetzen, ist natürlich auch die Rechenkomplexität. Stattdessen wird die initiale Kartierungsphase von der autonomen Einsatzphase getrennt betrachtet. Nach der Kartierung wird die Karte „eingefroren“ und fortan eine feststehende globale Karte als Basis der Selbstlokalisation genutzt.

Neben der Lokalisation auf Basis der Gridkarte und Entfernungssensoren wird außerdem ein Verfahren genutzt, welches ein Ansichten-basiertes, topologisches Modell (ein Graph, dessen Knoten mit visuellen Merkmalen attributiert sind) benutzt. Die eigentlichen Verfahren zur Selbstlokalisation sollen hier nur kurz in einem Überblick beschrieben werden. Sie beruhen jeweils auf der Monte Carlo Localization (MCL) [Fox et al., 1999], [Dellaert et al., 1999], einem auf Partikelfiltern basierenden Verfahren zur Schätzung der Wahrscheinlichkeitsverteilung für den aktuellen Roboterzustand. Die Grundlagen des verwendeten Ansichten-basierten Ansatzes zur Lokalisation sowie Teilergebnisse wurden publiziert in [König et al., 2000], [Gross et al., 2001], [Gross et al., 2002b], [Gross et al., 2002a], [Schröter et al., 2002] und [Gross et al., 2003]. [König et al., 2003a], [König et al., 2003b] und [König und Bischoff, 2004] gehen insbesondere auf die Eignung unterschiedlicher Bildmerkmale zur robusten Lokalisation bei Veränderungen der Beleuchtung und der Umgebung ein. [Schröter et al., 2005] präsentiert experimentelle Ergebnisse zur Selbstlokalisation mit einer Kombination visueller und entfernungsmessender Sensoren.

5.1.1 Map-Match-Selbstlokalisation

Im Gegensatz zum SLAM besteht der Zustand beim Lokalisationsproblem nur in der Roboterpose x. Die gesuchte Verteilung ist damit

$p(x_t|o_{1:t}, u_{1:t})$. Diese kann rekursiv berechnet werden (vgl. Formeln 4.2 bis 4.5).

$$p(x_t|o_{1:t}, u_{1:t}) = \frac{p(o_t|x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1}) dx_{t-1}}{p(o_t)} \tag{5.1}$$

bzw. getrennt nach Beobachtungs- und Bewegungsupdate:

$$p(x_t|o_{1:t}, u_{1:t}) = \frac{p(o_t|x_t) \cdot p(x_t|o_{1:t-1}, u_{1:t})}{p(o_t)} \tag{5.2}$$

$$p(x_t|o_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, u_t) \cdot p(x_t - 1|o_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{5.3}$$

Das Bewegungsupdate erfolgt dabei identisch zum SLAM (Unterabschnitt 4.3.1 „Bewegungsupdate“). Im Beobachtungsupdate wird die Beobachtungswahrscheinlichkeit $p(o_t|x_t)$ für jedes Partikel unter Nutzung der einheitlich vorgegebenen Karte berechnet (im Gegensatz zu SLAM, wo je Partikel eine eigene geschätzte Karte existiert).

Für die Partikelgewichtung und somit den Vergleich der aktuellen Beobachtung mit der Gridkarte kommt bei der Selbstlokalisation ebenso wie beim SLAM jeweils im Partikel eine lokale Karte aus den Beobachtungen gebildet und diese mit der globalen Karte verglichen. Obwohl diese Methode wiederum nur eine Annäherung der in Gl. 5.2 formulierten Beobachtungswahrscheinlichkeit darstellt, erwies sie sich als robuster gegenüber Sensorstörungen und geringen Veränderungen der Umgebung. Diese erhöhte Robustheit kann unter anderem damit begründet werden, dass alle Entfernungsmessungen eines Sonar- oder Laser-Entfernungssensors genutzt werden können, ohne numerische Instabilitäten zu riskieren (im Gegensatz dazu wird bei anderen Verfahren häufig vorab eine Beschränkung auf wenige Entfernungen eines Scans vorgenommen), Außerdem bleibt z.B. bei Verdeckungen durch dynamische Hindernisse wie Personen etc. die Information aus zurückliegenden Beobachtungen noch einige Zeit in der lokalen Karte erhalten und kann somit weiterhin die richtige Positionshypothese stützen.

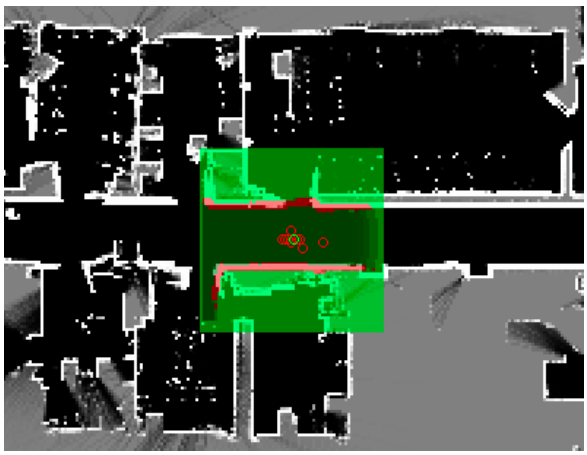

Abbildung 5.1: *Die Partikelbewertung bei der Lokalisation geschieht analog zum im vorhergehenden Kapitel beschriebenen Map-Match-SLAM durch Vergleich der lokalen Karte jedes Partikels (im Bild grün=Freiraum/rot=Hindernis) mit der globalen Karte (schwarz=Freiraum/weiß=Hindernis). In der dargestellten Situation unterscheidet sich die aktuelle Umgebung (Tür vom Gang zum oben angrenzenden Raum in lokaler Karte geschlossen) deutlich vom Zustand während der ursprünglichen Kartierung (Tür in globaler Karte offen). Trotzdem wird ein sehr guter Match-Wert an der richtigen Position erzielt. Die roten Kreise kennzeichnen hier die Partikelverteilung.*

5.1.2 Visuelle Selbstlokalisation

Gerade in der Baumarkt-Umgebung haben Entfernungs-Sensoren aufgrund der gleichförmigen Struktur (viele ähnliche Gänge) nur eine geringe Unterscheidungsfähigkeit zwischen verschiedenen Positionen. Durch die Integration der Bewegung und der Beobachtungshistorie im MCL-Algorithmus ist zwar trotzdem ein robustes Tracking der Position möglich, allerdings ist die Positions-Unsicherheit meist ungleichmäßig verteilt: Während seitlich im Gang eine hohe Genauigkeit erreicht werden kann, da der Abstand zur Seitenwand fast immer direkt gemessen werden kann, ist in Längsrichtung die Unsicherheit meist höher, weil die erwarteten Messungen an unterschiedlichen Längs-Positionen sich kaum unterscheiden. Dies führt außerdem dazu, dass eine globale Selbstlokalisation (ohne vorgegebene Startposition) sehr lang dauern kann.

Aus diesen Gründen wurden bereits sehr frühzeitig Ansichten-basierte visuelle Selbstlokalisations-Verfahren entwickelt [Gross et al., 2002b, Gross et al., 2003]. Insbesondere Panoramabilder, die mit einer omnidirektionalen Kamera aufgenommen werden, erlauben aufgrund des

großen Erfassungsbereichs und der hohen Wahrnehmungsreichweite eine deutlich bessere Schätzung der (groben) Position. Die Positionsauflösung ist allerdings normalerweise geringer als mit Entfernungs-Sensoren, da sich der Sensoreindruck bei kleinen Verschiebungen der Aufnahmeposition nur geringfügig ändert.

In der Kartierungsphase wird dazu neben der Gridkarte ein topologisches Modell der Umgebung erstellt (Abb. 5.2). Dieses besteht aus einem Graphen, dessen Knoten jeweils einer Position im Raum entsprechen und mit einer Panorama-Ansicht an der entsprechenden Stelle attributiert sind. Die Kanten geben direkte befahrbare Verbindungen zwischen den Knoten an. Da beim Kartenaufbau mit dem zuvor vorgestellten SLAM-Algorithmus der komplette Roboterpfad rekonstruiert wird, kann dieser z.B. anschließend zur Erstellung der topologischen Karte genutzt werden.

Aus den Bildern der Panorama-Kamera werden zunächst beschreibende Merkmale bestimmt: Das Bild wird dazu in einzelne radiale Sektoren unterteilt, jeder Sektor wird einzeln beschrieben (Abb. 5.2). Als Merkmale wurden dazu u.a. Farbmomente, Farbhistogramme und Gradientenhistogramme [König et al., 2003a] untersucht und verglichen. Zweck der Merkmalsextraktion ist sowohl die Daten-Kompression, um einen handhabbaren Rechenzeit- und Speicherbedarf zu erreichen, als auch die Gewährleistung von Invarianz gegenüber Änderungen der Beleuchtung oder Teilen der Umgebung, Verdeckungen etc. Während der autonomen Fahrt werden die jeweils aktuellen Beobachtungen im Beobachtungsupdate mit den erwarteten Ansichts-Merkmalen an den geschätzten Partikelpositionen verglichen, um die Partikelgewichte zu bestimmen. Dazu werden die dieser Position benachbarten Knoten des Graphen betrachtet und die Ähnlichkeiten der Ansichten gemittelt. Auf diese Weise ist auch eine Interpolation der geschätzten aktuellen Position zwischen den Stützstellen möglich. Es zeigt sich, dass die optimale Wahl der Merkmale stark von der Umgebung abhängt: während in Innenräumen Farbhistogramme zu guten Ergebnissen führen, sind in weiträumigen Outdoor-Umgebungen z.B. Gradientenhistogramme besser geeignet [Schenderlein, 2007]. Dies ist darin begründet, dass in von Menschen gestalteten Umgebungen stärkere Farbunterschiede auftreten, die eine Bestimmung der Position ermöglichen.

Im Gegensatz zur Selbstlokalisation mit Entfernungssensoren ist bei der Ansichten-basierten Lokalisation die typische Unsicherheit der Po-

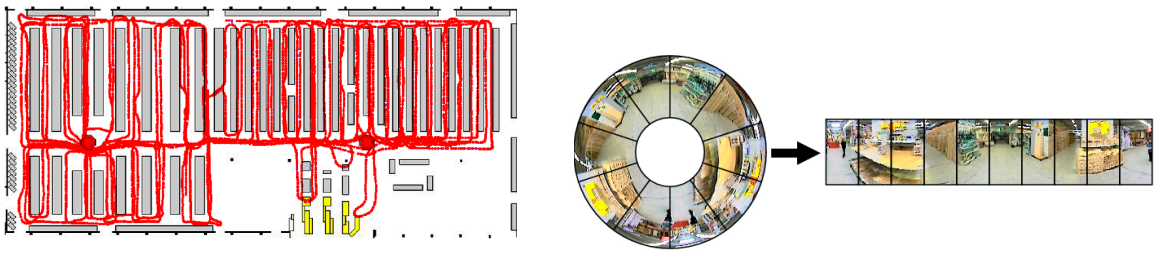

Abbildung 5.2: **: Links:** *Beispiel einer topologischen Karte für die visuelle Selbstlokalisation, hinterlegt mit einem Gebäudeplan.* **Rechts:** *Das Bild der omnidirektionalen Kamera wird in Sektoren unterteilt, auf diesen werden Merkmale wie Farbmittelwert, Farbhistogramm oder Kantenhistogramm berechnet und gespeichert bzw. verglichen.*

sitionen genau umgekehrt verteilt: entlang eines Ganges kann (entsprechende Farbunterschiede vorausgesetzt) die Position genauer bestimmt werden als in seitlicher Richtung. Zwar ändert sich die Umgebungsansicht bei Bewegung quer zum Gang sehr stark, allerdings müsste die Fläche für eine gute Bewertung auch entsprechend dicht mit Referenzpunkten abgedeckt sein. D.h. im Gegensatz zur Kartierung mit Entfernungssensoren müsste jeder Gang mehrfach mit unterschiedlichen Wand-Distanzen kartiert werden, was hohen Aufwand erfordert. Weiterhin enthält das visuelle Modell keine Information über nicht befahrbare Bereiche (Hindernisse), so dass bei der Lokalisation solche Partikel, die den befahrbaren Freiraum verlassen, unzureichend bewertet werden können, da in diesen Bereichen keine Referenzansichten existieren. Bei entsprechend dichter Flächenabdeckung könnte hier das Fehlen von Referenzen in unmittelbarer Partikelnähe zur Abwichtung genutzt werden, mit einem spärlichen Modell wie in Abb. 5.2 ist dies jedoch nicht möglich. Aufgrund dieser Verteilung der Unsicherheit ist allerdings eine gemeinsame Nutzung von Entfernungssensoren und Ansichten sehr gut zur Lokalisation geeignet. Mit der Kombination von Sonar-Sensoren und Farbmerkmalen aus Panorama-Bildern wurde in der Baumarkt-Umgebung ein mittlerer Positionsfehler von ca. 0.25m und ein maximaler Fehler von ca. 0.85m erreicht [Schröter et al., 2005].

5.2 Navigationsystem

Das Navigationssystem des Roboters führt alle Berechnungen und Aktionen aus, die die autonome Bewegung betreffen. Es nutzt dazu die be-

rechneten Daten anderer unabhängiger Teilsysteme, wie z.B. Selbstlokalisation, die globale Karte und die ständig aktualisierte lokale Karte, die einen Radius von einigen Metern um den Roboter erfasst (Abb. 5.3). Im Gegensatz zu den lokalen Karten der Selbstlokalisation weist diese eine deutlich höhere Auflösung als die globale Karte auf (Gridkarte mit Zellauflösung von typischerweise 3-5cm). Aktuell ist der Shopping-Roboter mit einem Ring aus Sonar-Sensoren (Erfassungsbereich insgesamt 360°) und einem Laser-Range-Scanner (Erfassungsbereich ca. 240° frontal) ausgestattet. In naher Zukunft soll weiterhin eine visuelle Hindernisdetektion, aufbauend auf dem in Anhang C.2 beschriebenen monokularen Tiefenschätzungs-Verfahren, integriert werden (Abschnitt 7.1). Die lokale Karte fusioniert die Beobachtungen der verschiedenen Sensoren), indem zunächst für jeden Sensor eine eigene lokale Karte generiert und ständig aktualisiert wird. Diese einzelnen lokalen Karten werden dann zu einer gemeinsamen Karte zusammengefasst, indem alle Hindernisse der Einzelkarten in die Gesamtkarte übertragen werden. Auf diese Weise wird vermieden, dass Objekte, die von einem Sensor nicht gesehen werden (z.B. aufgrund der Höhe), von diesem aus der gemeinsamen Karte entfernt werden können (dies wurde bereits ausführlicher in Abschnitt 3.1.4 diskutiert).

Die einzelnen lokalen Karten werden außerdem unabhängig von der Selbstlokalisation nur auf Basis der Odometriemessungen als Positionsschätzung aktualisiert. Dies ist zulässig, da innerhalb der begrenzten Größe der Karte die Odometriefehler vernachlässigbar gering sind. Durch dieses Vorgehen ergibt sich der Vorteil, dass die gemessene Bewegung garantiert stetig ist, während im Gegensatz dazu die vom Partikelfilter in der Selbstlokalisation geschätzte Position aufgrund der diskreten Hypothesen Sprünge aufweisen kann. Dies würde jeweils zu inkonsistenten Zuständen der Karte führen oder wiederum nachträglich eine Korrektur erfordern. Da die globale und lokale Karte dadurch mit verschiedenen Bezugssystemen arbeiten (lokal stetige Odometriekoordinaten gegenüber global konsistenten MCL-Koordinaten), muss der Navigator zu jedem Zeitpunkt die Position in beiden Koordinatensystemen kennen und jeweils beim Zugriff auf eine der Karten die entsprechende Position verwenden.

Der Navigator kann in unterschiedlichen Modi arbeiten, wie z.B. Anfahrt einer globalen Zielposition (Pfadverfolgung), Anfahrt einer relativen lokalen Zielposition oder zufällige Exploration. In all diesen Modi

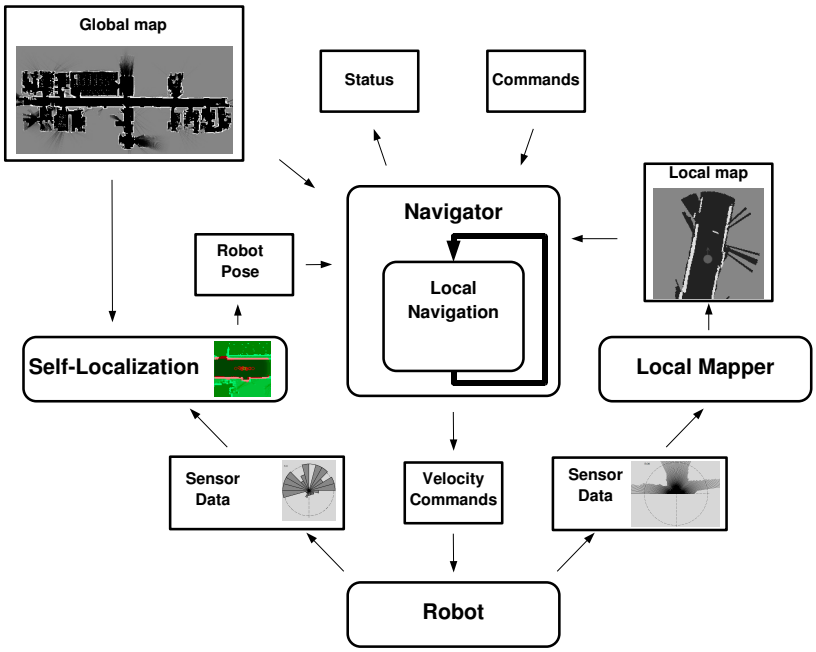

Abbildung 5.3: *Das Navigationssystem nutzt die Ausgangsdaten anderer unabhängiger Teilsysteme wie Selbstlokalisation und lokale Kartierung. Die lokale Navigation bildet ein (austauschbares) Teilmodul des Navigators. Die lokale Karte für die Hindernisvermeidung weist eine andere Auflösung auf als die globale Karte und benutzt ein eigenes Koordinatensystem (siehe Abschnitt 5.2), daher werden bei der Selbstlokalisation für die Partikel eigene lokale Karten erstellt.*

wird eine Kollisionsvermeidung gewährleistet. Weiterhin ist es möglich, den Roboter zu stoppen (wobei der Motor eine Gegenkraft gegen jede Bewegung ausübt) oder Fahrkommandos zwar zu berechnen, aber nicht an den Antrieb weiterzugeben. In diesem Simulationsmodus kann der Roboter zu experimentellen Zwecken manuell gesteuert werden, und gleichzeitig in jeder Situation die hypothetische Aktionswahl des Navigators dargestellt werden. Die Umschaltung zwischen den verschiedenen Modi und die Übergabe der entsprechenden Parameter (z.B. die Zielposition) erfolgt durch Kommandos, die im Normalbetrieb von den anwendungsspezifischen Verhaltensmodulen (Abschnitt 6.2) generiert werden. Im Experimentalbetrieb können Navigator-Kommandos auch manuell mittels einer entsprechenden graphischen Oberfläche gesetzt werden.

Der am häufigsten benutzte Modus ist die autonome Zielanfahrt einer globalen Position. Während andere Modi teilweise auch ohne globale Karte funktionieren, ist für diese globale Zielanfahrt eine Umgebungskarte erforderlich. Dem Navigator wird zunächst in einem Kommando

die Zielposition mitgeteilt, und er berechnet daraufhin den optimalen Pfad. In einem weiteren Kommando wird der Pfadverfolgungs-Modus aktiviert. Der Navigator wertet die zur Verfügung stehenden Daten über aktuelle Position, Pfadverlauf und lokale Umgebung aus und ruft zyklisch das lokale Navigationsmodul auf, um die beste Fahr-Aktion zu bestimmen. Dies geschieht so lange, bis das Ziel erreicht ist oder ein neues Kommando zum Verlassen des Zielmodus empfangen wird. Der aktuelle Status wird jeweils nach außen gemeldet und ist für andere Module und graphische Darstellungen verfügbar.

5.3 Globale Navigation - Pfadplanung

Die Pfadplanung bestimmt auf der globalen Karte den kürzesten Pfad von der aktuellen Roboterposition zur Zielposition. Dieser Pfad berücksichtigt zunächst nur die Objekte in der globalen Karte sowie die Abmessungen des Roboters (Durchfahrt-Breite) und verfolgt als konkurrierende Ziele die Minimierung der Pfadlänge und die Einhaltung eines (in gewissen Grenzen) möglichst großen Sicherheitsabstandes zu Hindernissen. Der Pfad besteht aus einer Abfolge von $[x, y]$-Positionen und stellt damit eine starke Abstraktion der tatsächlich möglichen Roboterbewegung dar. Während der autonomen Fahrt werden unter Berücksichtigung des Pfades und weiterer Daten die Fahr-Aktionen des Roboter bestimmt.

Weiterhin wurde im Rahmen einer Diplomarbeit ein Ansatz untersucht, die Aktionen des Roboters direkt in einem erweiterten $[x, y, \varphi, v, w]$-Raum zu planen, welcher im Roboterzustand die Pose und Geschwindigkeit (Translation und Rotation) enthält. Aufgrund von Laufzeitproblemen und praktischen Erwägungen sowie der den Aufwand nicht rechtfertigenden Ergebnisse führte dieser Ansatz nicht zu einer Verbesserung des Fahrverhaltens, trotzdem sollen die Erkenntnisse hier vorgestellt werden.

5.3.1 Pfadsuche auf 2D-Karten: Vergleich von Dijkstra- und A*-Algorithmus

In diesem Abschnitt sollen zwei Standard-Algorithmen zur Pfadsuche hinsichtlich des Suchaufwandes, bezogen auf das Baumarkt-Szenario,

verglichen werden. Weiterhin wird die konkrete Implementierung für die Suche des Pfades auf einer Gridkarte beschrieben. Für die Pfadplanung wird die Gridkarte als Graph interpretiert, in dem jede Zelle einen Knoten V darstellt. Es werden allerdings nur Knoten berücksichtigt, deren Belegtheitswahrscheinlichkeit eine Schwelle occ_{free} nicht überschreitet. Kanten E existieren dabei zwischen den Zellen, die einander direkt oder diagonal benachbart sind. Um die Annäherung des Pfades zu Hindernissen zu bewerten, werden die Kanten jeweils umgekehrt proportional zur Entfernung zwischen der Ausgangszelle und der nächsten belegten Zelle (jedoch maximal 2m) gewichtet. Diese Kantengewichte müssen jeweils dann neu berechnet werden, wenn sich die Karte (oder Teile davon) ändert, nicht jedoch bei einem neuen Zielpunkt. Bei Wahl eines Zieles wird jeweils mittels eines Pfad-Such-Algorithmus ein Potentialfeld über die Karte berechnet, welches zu jeder Zelle/Knoten die minimale Entfernung zum Zielpunkt V_Z darstellt. Die Abarbeitung der Knoten erzeugt dabei für jede Zelle/Knoten den optimalen Pfad zum Ziel, dessen Länge dieser Minimaldistanz entspricht (als Sequenz von Zeigern jeweils auf die Vorgängerknoten).

Einer der einfachsten und meist genutzten Pfad-Such-Algorithmen ist der Dijkstra-Algorithmus [Dijsktra, 1959]. Der Dijkstra-Algorithmus (Abb. 5.4) berechnet das gesamte Potentialfeld für n Zellen in $O(n \log n)$ (da hier im speziellen Fall die Anzahl der Kanten in $O(n)$ ist). Die Ausbreitung der Suche erfolgt dabei gleichmäßig in alle Richtungen, d.h. Zellen mit geringerer Entfernung (Kosten) zum Ziel $d(cell)$ werden stets vor Zellen mit höherer Entfernung bearbeitet. Insbesondere bedeutet dies, dass vor dem Erreichen der Zelle, welche der Roboterposition entspricht (der frühestmögliche Abbruch-Zeitpunkt, zu dem ein Pfad von der aktuellen Position zum Ziel bekannt ist), alle anderen Zellen mit geringerer Entfernung abgearbeitet wurden. Wünschenswert ist stattdessen eine gerichtete Suche, bei der möglichst nur solche Zellen bearbeitet werden, die vom Ziel aus in Richtung der Startzelle liegen. Dies wird durch den A*-Algorithmus [Hart et al., 1968] erreicht. A* erweitert den Dijkstra-Algorithmus um eine Heuristik $H(cell)$, die für jede Zelle die Entfernung zum Startpunkt abschätzt. Es wird nun die geschätzte *Gesamt*-Pfadlänge $d(cell)+H(cell)$ als Kriterium für die Reihenfolge der Abarbeitung der Knoten gewählt. Dadurch werden solche Knoten bevorzugt, für die die Heuristik eine geringe Entfernung zum Ziel schätzt: die Suche erfolgt gerichtet (Abb. 5.5).

Die erreichbare Effizienz-Steigerung wird wesentlich durch die Wahl der Heuristik-Funktion H bestimmt. Für H sind nur solche Funktionen zulässig, die die tatsächliche Entfernung von V zum Startpunkt S nicht überschätzen (also eine untere Schranke bilden). Andernfalls ist nicht garantiert, dass der optimale Pfad gefunden wird. Für eine möglichst hohe Effizienz sollten die Kosten aber auch nicht zu stark unterschätzt werden. Je genauer H die Entfernung abschätzt, um so weniger Knoten müssen bearbeitet werden. Im Idealfall (H schätzt für jeden Knoten exakt die Restkosten) würden nur die Knoten vom Algorithmus betrachtet, die tatsächlich auf dem kürzesten Pfad liegen. Dies ist jedoch mit der gewählten Gewichtsfunktion nicht möglich. Die reale Pfadlänge entspricht der Summe der Kantengewichte über die vom Pfad durchlaufenen Kanten E. Um die Optimalität zu gewährleisten, muss die Heuristik eine untere Schranke für die Anzahl der zu durchlaufenden Kanten (bzw. Anzahl der zu durchlaufenden Zellen) und eine untere Schranke für das mittlere Kantengewicht bestimmen. Die Anzahl der Zellen kann durch die euklidische Distanz oder etwas genauer durch eine Zusammensetzung einer horizontalen/vertikalen und einer diagonalen Strecke (da Zell-Nachbarschaften nur in gerader oder 45°-Richtung existieren) geschätzt werden. Das mittlere Kantengewicht kann nur durch das Minimum aller Kantengewichte zulässig abgeschätzt werden. Leider stellt dies nicht in allen Fällen eine gute Approximation dar, so dass die Effizienzsteigerung durch den A*-Algorithmus je nach Start- und Zielpunkt unterschiedlich hoch ausfällt (Abb. 5.6 und 5.7). Trotzdem benötigt die Pfadplanung auf einer Karte der Fläche 100m $\times$ 50m mit einer Zellauflösung von 0.1m weniger als 1s, was für die Anwendung ausreichend ist.

Eingaben

1 V // alle Zellen
2 $f(V)$ // Distanz jeder Zelle nur nächsten nicht-freien Zelle
3 Z, S // Ziel- und Startposition [x,y]
4 r // Radius um Startposition

Initialisierung

5 $cell_Z \leftarrow$ `argmin` $dist(cell, Z) : occ(cell) \leq occ_{free}$ // nächste freie Zelle in Umgebung // der Zielposition
6 $Cells_S \leftarrow \{cell : dist(cell, S) < r, occ(cell) \leq occ_{free}\}$ // alle freien Zellen in Radius // um Startposition
7 $d(V) \leftarrow \infty$ // Zielentfernung für alle Zellen = ∞
8 $p(V) \leftarrow V$ // Pfad-Vorgänger für jede Zelle = selbst
9 $Q \leftarrow [cell_Z, 0, cell_Z]$ // Queue = Zielzelle, Entfernung zum Ziel = 0, // kein Vorgänger (Pfadende)

Algorithmus

Label 1: **// Bearbeite Zelle mit geringster Entfernung:**
10 `falls` $Q = \emptyset$`: return false` // Queue leer: Algorithmus beendet // (ohne Pfad zu finden)
11 $[c, d, p]$ `= pop(`Q`)` // aktuelle Zelle = Front der Queue
12 `falls` $d(c) < \infty$`:` // Zelle schon bearbeitet
13 **goto Label 1**
14 $d(c)$ `=` d`, p(c) = p` // speichere Zielentfernung und Pfad-Vorgänger
15 `falls` $c \in Cells_S$
16 $Cells_S = Cells_S \setminus c$ // entferne erreichte Zellen aus Liste der Startzellen
17 `falls` $Cells_S = \emptyset$`: return true` // alle Startzellen bearbeitet: Pfad gefunden
18 `für` $c_N \in N(c)$ // betrachte Nachbarzellen von c
19 `falls` $occ(c) \leq occ_{free}$ `oder` $occ(c_N) > occ_{free}$ // keine Ausbreitung von // belegten zu freien Zellen!
20 `push(Q, [`$c_N, d(c) + dist(c, c_N)\dot{f}(c_N), c$`])` // Entfernung des Vorgängers // + Kantengewicht

Rückgabe

21 $true/false$ // Pfad gefunden?

Abbildung 5.4: *Pseudocode für die Implementierung des Dijkstra-Algorithmus auf der Karte. Wenn der Algorithmus erfolgreich beendet wird, kann der Pfad von jedem bearbeiteten Knoten zum Ziel durch Verfolgen der Vorgänger-Zeiger abgefragt werden. Um auch bei leichten Positions-Abweichungen noch einen Pfad an der jeweiligen Roboterposition bestimmen zu können, wird eine Region um die Startposition als Ziel definiert (Zeile 6, 15-17). Prinzipiell werden werden sowohl freie als auch belegte Zellen in der Pfadplanung berücksichtigt, allerdings kann ein Pfad nie von einer freien zu einer belegten Zelle führen (Zeile 19). Umgekehrt ist dies jedoch möglich, so kann ein Pfad auch von einer als belegt gekennzeichneten Position bestimmt werden, z.B. falls die reale Position entgegen der Karteninformation doch frei ist. Für die A*-Variante werden die Zellen in der Queue Q nicht nach $d(c)$, sondern nach $d(c) + H(c)$ sortiert, dazu ist lediglich Zeile 20 um den Sortierschlüssel zu erweitern.*

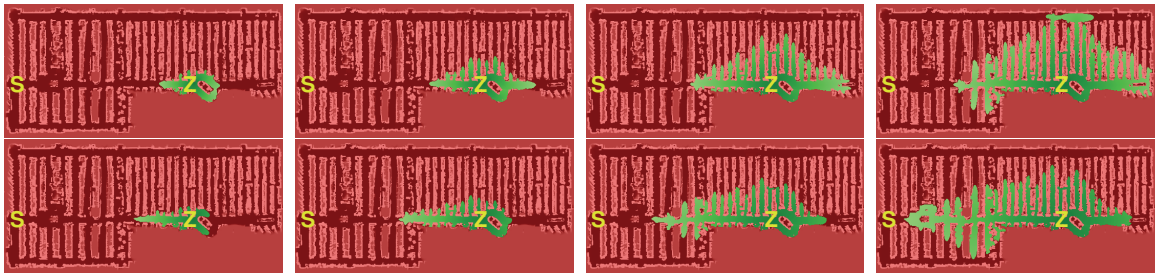

Abbildung 5.5: *Visualisierung der Such-Reihenfolge vom Ziel Z ausgehend bei Dijkstra- (oben) und A*-Algorithmus (unten). Grün markiert sind die bearbeiteten Knoten nach 10000, 20000, 40000 und 70000 Schleifendurchläufen (von links nach rechts). Beim Dijkstra-Algorithmus erfolgt die Ausbreitung relativ gleichmäßig in alle Richtungen, bei A* dagegen deutlich zielgerichteter in Richtung des Startpunktes S.*

Abbildung 5.6: *Nachdem bei der Suche die Startzelle (sowie alle Zellen in einem vorgegebenen Radius r um die Startzelle) erreicht wurde, endet der Algorithmus. Der Dijkstra-Algorithmus hat zu diesem Zeitpunkt 156567 Zellen bearbeitet (links), A* lediglich 94053 (Mitte). Im rechten Bild ist der resultierende Pfad dargestellt, dieser ist bei beiden Algorithmen gleich (da beide den optimalen Pfad berechnen). Start- und Zielpunkt sind hier identisch zu Abb. 5.5.*

Abbildung 5.7: *Vergleich der beiden Algorithmen an einem weiteren Zielpunkt: Der Dijkstra-Algorithmus (links) bearbeitet insgesamt 176676 Zellen, der A* (Mitte) 160290. Rechts ist wiederum der resultierende Pfad abgebildet. Die gewählte Heuristik geht von maximalem Hindernisabstand aller Zellen aus, um die Kosten mit Sicherheit nicht zu überschätzen, und stellt damit in einem Gebiet, in dem viele schmale Gänge existieren, eine relativ ungenaue Abschätzung dar. Dadurch fällt der Effizienzgewinn des A*-Algorithmus hier deutlich geringer aus als im Beispiel der Abbildungen 5.5 und 5.6.*

5.3.2 Planung im erweiterten Zustandsraum

Im vorigen Abschnitt wurden Untersuchungen zur Planung im 2D-Raum vorgestellt. Dieser Ansatz generiert zunächst einen Pfad, welcher lediglich aus einer Abfolge von Positionen $[x, y]$ besteht. Anschließend werden in der Ausführungsphase Bewegungskommandos generiert, um diesem Pfad so gut wie möglich zu folgen. Alternativ dazu wurde in einer Diplomarbeit [Schmiegel, 2004] ein Verfahren entwickelt, direkt eine Abfolge von Bewegungsschritten zu planen, die dem Roboter einen Übergang vom aktuellen Zustand zum Zielzustand ermöglicht. Als Zustand des Roboters wird dabei die Kombination aus seiner Pose und der aktuellen Translationsgeschwindigkeit v sowie Rotationsgeschwindigkeit ω benutzt. Somit ergibt sich ein 5-dimensionaler Zustandsraum $[x, y, \varphi, v, \omega]$. Das entwickelte Verfahren stellt eine Erweiterung von [Stachniss und Burgard, 2002] dar. Allerdings ist für die Repräsentation und Rechnung innerhalb dieses 5D-Raumes ein wesentlich höherer Speicher- und Rechenaufwand nötig als für die Planung im 2D-Raum. Gleichzeitig zeigte sich, dass u.a. aufgrund der notwendigen Diskretisierung des Zustandsraumes und des hohen Rechenaufwandes zum damaligen Zeitpunkt nicht mit der gewünschten Sicherheit in jedem Zeitschritt eine zulässige oder gar optimale Bewegungsfolge generiert werden konnte. Dies führte dazu, dass relativ oft die Fahrt unterbrochen werden musste, um eine Kollision aufgrund Überschreitung der zulässigen Reaktionszeit zu vermeiden. Damit konnte die theoretische Möglichkeit, gegenüber einer Kombination von 2D-Planung und sukzessiver Fahrkommandogenerierung bessere Fahrtrajektorien zu erzielen und das Ziel schneller und gleichzeitig sicher zu erreichen, in der Praxis nicht umgesetzt werden. Aus diesem Grund wurde dieser Ansatz zunächst nicht weiter verfolgt.

5.4 Lokale Navigation

Ziel der lokalen Navigation ist es, eine Bewegung entsprechend dem globalen Ziel des Navigators (meist eine Verfolgung des zuvor geplanten Pfades) zu realisieren und dabei Kollisionen mit durch die Sensoren wahrgenommenen Hindernissen zu vermeiden. Die Umsetzung des geplanten Pfades in konkrete Roboter-Bewegungen sowie die Reaktion auf alle Abweichungen von der globalen Karte, die in der Pfadplanung

nicht berücksichtigt werden (dynamische Hindernisse wie z.B. Personen) geschieht hier. Das lokale Navigationsmodul stellt ein Teilsystem des Navigators dar (Abb. 5.3). Es bildet keine eigene Repräsentation der lokalen Umgebung, sondern nutzt die ständig vom Local Mapper aktualisierte Gridkarte, die die Wahrnehmungen der einzelnen Sensoren integriert.

In diesem Abschnitt werden zwei verschiedene lokale Navigationsverfahren vorgestellt und experimentell verglichen: aktuell kommt auf dem Shopping-Assistenten ein Algorithmus zum Einsatz, der auf dem Vector Field Histogram (VFH)-Verfahren [Borenstein und Koren, 1991], [Ulrich und Borenstein, 1998] aufbaut. Dieser stellt einen rein reaktiven Ansatz dar, der jeweils die Bewegungsrichtung und -geschwindigkeit zum aktuellen Zeitpunkt bestimmt, ist allerdings leicht parametrierbar und effizient berechenbar. In der Baumarkt-Umgebung wird damit bei Höchstgeschwindigkeiten von 1m/s ein robustes und zügiges Fahrverhalten erreicht. Speziell in engen Umgebungen mit schmalen Durchfahrten (z.B. Türen) und nur wenigen geraden Strecken (Wohnungen, Büros, ..) führt dieses VFH-Verfahren jedoch wegen der häufigen Richtungswechsel zu niedrigeren Durchschnittsgeschwindigkeiten. Deshalb wurde ein alternatives Verfahren entwickelt, welches versucht, eine optimale lokale Trajektorie für die aktuell wahrgenommen Hindernisgeometrie zu ermitteln. Dabei werden jeweils verschiedene Alternativen aus dem Trajektorienraum durch Sampling bestimmt und mittels einer Kostenfunktion bewertet. Der Roboter bewegt sich jeweils für einen gewissen Zeitraum entlang der best-bewerteten Trajektorie, ein Geschwindigkeitsregler stellt dabei sicher, dass der Roboter der vorgegebenen Kurve folgt. Dieses Verfahren wird im Anschluss an das erweiterte VFH-Verfahren vorgestellt, dessen Grundlagen wurden bereits in [Schröter et al., 2007b] beschrieben.

Abschließend wird die Möglichkeit betrachtet, jeweils in jedem Bewegungsschritt eine lokale *Planung* unter Berücksichtigung des global geplanten Pfades und der lokalen Karte durchzuführen. Diese Methode bietet sich insbesondere in Kombination mit einer reaktiven Bewegungssteuerung wie dem VFH an.

5.4.1 Vector Field Histogram

Der Pseudocode für das gesamte „Vector Field Histogram“-Verfahren inklusive aller nachfolgend beschriebenen Erweiterungen ist in Abb. 5.13 aufgelistet. Im Folgenden wird jeweils bei der Beschreibung einzelner Abschnitte auf die entsprechenden Zeilen des Codes verwiesen, um eine einfachere Zuordnung zu ermöglichen.

Im originalen VFH-Verfahren wird ein Polarhistogramm über der Belegtheitskarte gebildet, welches die Hinderniswahrnehmungen in der jeweiligen Richtung (ausgehend von der Roboterposition) akkumuliert. Dabei werden die Zellen der zu Grunde liegenden Gridkarte mit zunehmender Entfernung zur Roboterposition abgewichtet und aufsummiert. Mittels eines Schwellwertes werden die Histogramm-Segmente in vom Roboter aus gesehen freie und versperrte Richtungen unterteilt (mit einer festen Sektor-Auflösung von z.B. 15°) und auf dieser Basis die zu fahrende Richtung ausgewählt.

Die Akkumulation *aller* Zellen in einer Beobachtungsrichtung erhöht die Robustheit bei ungenauen Sensoren, da jede Zelle exakt mit ihrer Belegtheitswahrscheinlichkeit berücksichtigt wird. Andererseits führt es aber auch dazu, dass nicht nur die Entfernung zu einem Hindernis das Roboterverhalten bestimmt, sondern auch die Ausdehnung des Hindernisses. Nicht nur die Objektfront geht in das Histogramm ein, sondern jede Zelle, die von einem Objekt belegt wird (Abb. 5.8). Zwar ist bei einer Einzelbeobachtung normalerweise nur die Front für den Sensor (z.B Sonar) sichtbar, so dass dieser Effekt wenig Einfluss hat. Wenn allerdings der Roboter um ein Hindernis herum fährt oder z.B. eine Tür durchquert und dabei eine Wand von beiden Seiten sieht, belegen diese aus unterschiedlichen Blickwinkeln erfassten Objekte auch eine ihrer Ausdehnung entsprechende Fläche in der Karte. Im polaren Belegungshistogramm werden sie somit stärker abgebildet als eine gegenüberliegende Wand, die nur von vorn gesehen wird, dadurch wird fälschlich eine Annäherung zu einer Seite gegenüber einer mittigen Durchfahrt als günstiger bewertet.

Statt dessen wird im hier vorgestellten Algorithmus zwar ebenfalls eine Einteilung der Umgebung in Richtungssegmente vorgenommen, es wird aber nur jeweils die Entfernung zur nächsten belegten Zelle innerhalb des Segmentes bestimmt. Dies geschieht (analog zur Simulation der erwarteten Beobachtung in der Lokalisation, Abschnitt 5.1) durch

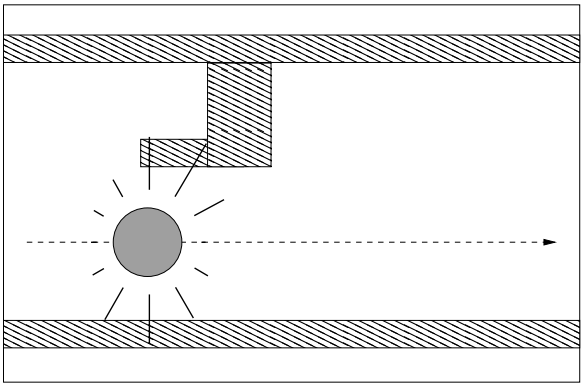

Abbildung 5.8: *Das Belegtheitshistogramm des „Vector Field Histogram“-Algorithmus ist durch die radialen Striche angedeutet: Obwohl der Abstand zum Hindernis auf beiden Seiten gleich ist, gehen die verdeckten Objekte ebenfalls in das Histogramm ein und erhöhen so die Werte auf der linken Seite (bezogen auf die Fahrtrichtung). Dadurch wird die Durchfahrt nicht mittig passiert, sondern der Roboter nähert sich der rechten Seite an. Das wird vermieden, wenn jeweils nur die Entfernung zur nächsten belegten Zelle benutzt wird.*

einen virtuellen Scan in der lokalen Umgebungskarte, der mittels Look-Up-Tabellen effizient berechnet wird. Die Unterscheidung der freien und belegten Segmente erfolgt anhand des Vergleichs der je Segment „gemessenen“ Entfernung mit einer Entfernungsschwelle d_{free}. Auch dies ist ein Vorteil gegenüber dem Polarhistogramm, denn eine Entfernung ist als Schwelle deutlich intuitiver zu wählen als eine abstrakte Belegtheitsschwelle im Histogramm, insbesondere wenn die Schwelle dynamisch in Abhängigkeit von der Robotergeschwindigkeit angepasst wird, um ein rechtzeitiges Ausweichen vor Hindernissen zu gewährleisten (siehe nachfolgende Erläuterung).

Zunächst werden allerdings die Hindernisse der lokalen Karte um die Länge des Roboterradius r dilatiert (vergrößert). Aus Effizienzgründen wird diese Operation jedoch nicht auf der Karte selbst ausgeführt, sondern auf dem virtuellen Entfernungsscan (Abb. 5.9, Pseudocode Zeilen 5-8). Der Gesamt-Scan besteht aus einer Menge von Einzelmessungen, die jeweils einen Kreiskegel mit gleicher Winkelbreite berücksichtigen (Abb. 3.1, 3.2). Die nächste belegte Zelle innerhalb des Kegels bestimmt die Messlänge. Vereinfacht kann die Messung (für schmale Kegel) auch als Strahl entlang der Mittelachse des Kegels betrachtet werden, die gemessene Länge beschreibt dann einen Punkt auf diesem Strahl. Ein Kreisbogen mit dem Radius r um diesen Endpunkt stellt genau die Dilatation des wahrgenommenen Hindernisses um r dar. Dieser Kreisbogen schneidet wiederum einige der benachbarten Messstrahlen und beschreibt damit, wo diese Strahlen das *dilatierte* Objekt wahrnehmen würden. Der Schnittpunkt kann aus der Messlänge und dem jeweiligen

Winkel zwischen den Strahlen berechnet werden (Abb. 5.9 links). Nach dem Sinussatz gilt in dem dort gekennzeichneten Dreieck

$$\frac{\sin\alpha}{r} = \frac{\sin\beta}{d'} = \frac{\sin\gamma}{d} \tag{5.4}$$

Mit den bekannten Größen d, r und α lässt sich damit d' berechnen:

$$\gamma = \arcsin(\frac{d}{r} \cdot \sin\alpha) \tag{5.5}$$

$$\beta = \pi - (\alpha + \gamma) \tag{5.6}$$

$$d' = \frac{\sin\beta}{\sin\alpha} \cdot r \tag{5.7}$$

Die Gleichung 5.5 bestimmt dabei entweder 0, 1 oder 2 gültige Werte für γ (wegen der nicht eindeutigen Umkehrbarkeit und gleichzeitig des beschränkten Wertebereiches der Sinus-Funktion). Falls keine Lösung existiert, so schneidet der Kreis nicht den Strahl, dieser Fall gilt z.B. für den äußersten rechten Strahl in Abb. 5.9 (links). Der beobachtete Punkt hat dann auf den entsprechenden anderen Strahl keinen Einfluss. Falls zwei Lösungen existieren, so werden beide bis Gleichung 5.7 berechnet und der kürzere der beiden resultierenden Abstände als dilatierte Entfernung verwendet.

Der virtuelle Scan auf der originalen Karte wird also in einen Scan für eine dilatierte Karte transformiert, indem für jeden einzelnen Strahl die minimale Entfernung der Schnittpunkte mit den Kreisbögen für alle Messungen bestimmt wird. Auf diese Weise ist die Komplexität der Berechnung weder von der Anzahl der Zellen der Karte noch vom Radius r, sondern nur von der Anzahl der Messungen im virtuellen Scan abhängig und damit deutlich geringer als bei einer Dilatation des zweidimensionalen Kartenfeldes. Ein solches Verfahren ist aus der Literatur bisher nicht bekannt.

Im Gegenzug zur Vergrößerung der Hindernisse um den Roboterradius kann der Roboter selbst als punktförmig angenommen werden. Dies entspricht der Transformation in den Konfigurationsraum der zulässigen (kollisionfreien) Positionen für einen kreisförmigen Roboter. Die Länge einer Messung im virtuellen Scan nach der Dilatation gibt an, wie weit der Roboter sich in dieser Richtung bewegen kann, bevor es zu einer

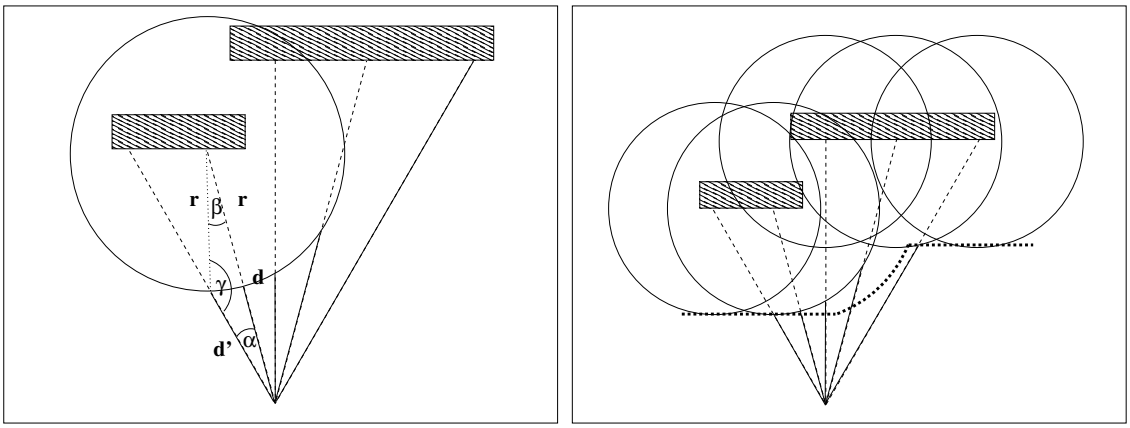

Abbildung 5.9: **Links:** *Dilatation für einen einzelnen Strahl - Der Kreisbogen um den Endpunkt des Strahls gibt die Ausmaße des dilatieren Punktes an. Die Schnittpunkte des Kreisbogens mit den Strahlen verkürzt die gemessenen Entfernungen. Die Längen entsprechen denen, die die einzelnen Strahlen messen würden, wenn der Punkt zuvor auf der Karte dilatiert worden wäre. Nicht alle Strahlen schneiden den Kreisbogen, der äußerste rechte Strahl „sieht" auch nach der Dilatation an dem betrachteten Punkt vorbei.* **Rechts:** *Dilatation für alle Strahlen. Die Kreisbögen um die einzelnen Endpunkte überlagern sich für alle Strahlen, die kürzeste Länge auf dem Strahl bestimmt jeweils den Messwert. Die breite gestrichelte Linie deutet die exakte Hindernisform bei (deutlich aufwändigerer) Berechnung der Dilatation auf der Karte an. Die vereinfachten dilatierten Entfernungen entsprechen dieser Form näherungsweise. Je geringer der Abstand zwischen einzelnen Strahlen, um so besser wird der exakte Verlauf approximiert, bei einer unendlichen Dichte der Strahlen würde keine Abweichung mehr auftreten.*

Kollision kommt. Die Dilatation und nachfolgende Operation im Konfigurationsraum stellt eine methodische Erweiterung des VFH-Verfahrens dar.

Im nächsten Schritt werden die Richtungssektoren durch Vergleich mit der Entfernungsschwelle d_{free} in freie und gesperrte Richtungen unterschieden (Pseudocode Zeilen 9-13). Durch die zuvor beschriebene Dilatation kann sich der Roboter in jede Richtung bewegen, in welcher der resultierende Abstand größer als 0 ist. Um einen gewissen Sicherheitsabstand zu Hindernissen zu wahren, erscheint es dennoch als sinnvoll, d_{free} größer als 0 zu wählen. Es ist allerdings zu beachten, dass dieses Schwelle nur dann tatsächlich einen Abstand zu Hindernissen garantiert, wenn der Roboter sich frontal auf diese zu bewegt. Der seitliche Abstand kann hingegen die Schwelle unterschreiten, dies ist jedoch unproblematisch (und bei schmalen Durchfahrten sogar notwendig), da trotzdem immer kollisionsfreie Zustände garantiert werden und der Sei-

tenabstand im Gegensatz zum frontalen Abstand auch bei Fehlern wie z.B. verspätetem Bremsen unproblematisch ist. Die Bestimmung der zu fahrenden Richtung aus den freien/gesperrten Richtungen folgt wiederum dem originalen VFH: Die einzelnen freien Richtungen werden zu freien „Lücken“ zusammengefasst (Pseudocode Zeilen 14-16), und jeweils die befahrbaren Richtungen bestimmt (Zeilen 17-28): bei schmalen Lücken wird nur die exakt mittige Durchfahrt zugelassen, bei breiteren Freiräumen wird lediglich ein Randbereich (wiederum ein Sicherheitsbereich) gesperrt, so dass eine Menge an möglichen Richtungen verbleibt. Im Gegensatz zum originalen VFH-Algorithmus müssen bei vorheriger Hindernisdilatation die Lücken keine Mindestbreite aufweisen, da der Roboter als punktförmig angenommen werden kann. Aus den als befahrbar bestimmten Richtungen wird diejenige mit der geringsten Differenz zur Richtung des Pfadverlaufs (aus der globalen Pfadplanung) ausgewählt (Zeilen 33-35).

Wenn die Unterscheidung der Hindernis- und Freiraum-Richtungen nur anhand der konstanten Abstandsschwelle d_{free} erfolgt, so führt dies häufig zu einer unnötigen Hindernis-Annäherung, da Objekte, die zwar bereits in der lokalen Karte erfasst sind, aber noch einen hohen Abstand aufweisen, keinen Einfluss auf das Fahrverhalten haben, auch wenn sie im Pfad des Roboters liegen (Abb. 5.10 links). Um ein frühzeitigeres Ausweichen vor Hindernissen zu ermöglichen, wird statt dessen neben dem minimalen Abstand zusätzlich eine maximale Entfernung d_{early} definiert, ab der bereits eine Reaktion erfolgen soll. Diese maximale Ausweich-Entfernung wird linear abhängig von der Roboter-Geschwindigkeit gewählt, so dass bei höherer Geschwindigkeit bereits bei größeren Entfernungen eine Reaktion erfolgt, während bei kleinen Geschwindigkeiten der Roboter näher frontal auf Hindernisse zu fährt. Die zuvor beschriebene Suche nach freien Richtungen erfolgt nun in einer Schleife mit sukzessive abnehmender Schwelle, beginnend bei der geschwindigkeitsabhängigen maximalen Entfernung d_{early} und schrittweise verringert bis zur minimalen Entfernung d_{free} (Pseudocode Zeilen 29-32). Die Schleife wird dann abgebrochen, wenn bei einer Entfernung eine befahrbare Richtung im zulässigen Bereich gefunden wird. Auf diese Weise erfolgt eine frühzeitige Ausweichbewegung vor einem Hindernis, wenn diese möglich ist. Andererseits stoppt der Roboter nicht sofort, wenn bei höherer Entfernungsschwelle keine zulässige Fahrrichtung existiert (z.B. in engen Hindernissituationen, Abb. 5.10 rechts), sondern der Algorithmus sucht zunächst mit kürzerer Entfernung weiter,

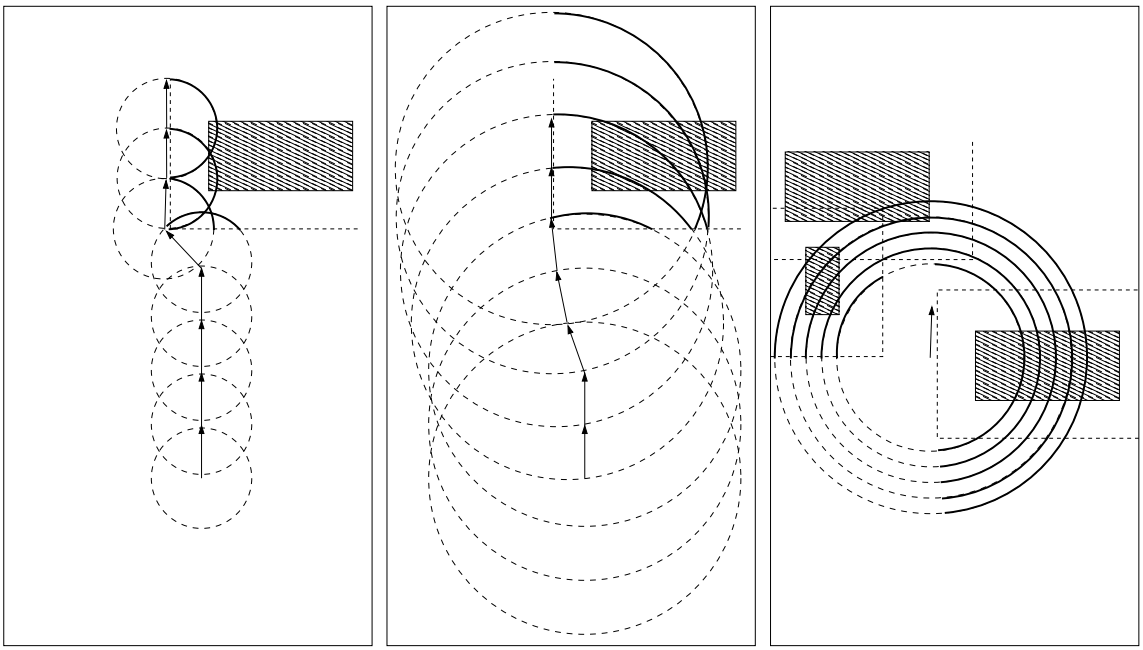

Abbildung 5.10: **Links:** *Wenn im Vector Field Histogram zur Unterscheidung der freien (gestrichelt) und versperrten (durchgezogen) Richtungen eine kleine Entfernungsschwelle (Kreisradius) gewählt wird, reagiert der Roboter erst bei geringem Abstand auf Hindernisse (schraffiertes Rechteck) und muss dann die Fahrtrichtung relativ stark korrigieren, um auszuweichen. Die gestrichelten Linien geben die um den Roboterradius verbreiterten Umrisse des Hindernisses an. Der Roboter kann damit als punktförmig angenommen werden, jede Position jenseits der gestrichelten Linien ist zulässig. Die diskrete Sektoreinteilung der Richtungen wurde für eine übersichtlichere Darstellung hier ausgelassen.* **Mitte:** *Mit einer größeren Entfernungsschwelle reagiert der Roboter deutlich früher auf das Hindernis und muss dabei die Fahrtrichtung weniger stark ändern. Dadurch ist auch eine höhere Fahrgeschwindigkeit möglich.* **Rechts:** *Falls nur sehr wenig Freiraum zwischen den Hindernissen existiert, wird mit einer großen Entfernungsschwelle u.U. keine befahrbare Richtung gefunden, die auch die Entfernung zum Ziel verkürzt (die rückwärtigen Richtungen im Beispielbild scheiden wegen dieser zusätzlichen Bedingung aus). In einer solchen Situation wird die Schwelle, beginnend bei der maximalen Entfernung d_{early}, sukzessive verringert und jeweils neu die befahrbaren Richtungen bestimmt. Sobald eine gültige Bewegungsrichtung gefunden wird, wird die Schleife beendet. Falls auch nach Verkürzung der Entfernung bis zum minimalen Abstand d_{free} keine gültige Richtung gefunden wird, muss der Roboter anhalten.*

fährt dadurch normalerweise näher an das Hindernis heran und reagiert später mit einer Richtungskorrektur. Da bei früherem Ausweichen eine geringere Auslenkung gegenüber der Geradeaus-Fahrt notwendig ist, wird durch den Einsatz der dynamischen Entfernungsschwelle eine Fahr-

weise mit weniger Richtungskorrekturen und höherer Durchschnittsgeschwindigkeit erreicht (Abb. 5.10 Mitte). Auch die dynamische Wahl der Schwelle zur Unterscheidung freier und belegter Sektoren ist nicht Teil des originalen VFH-Algorithmus, sondern stellt eine eigene Erweiterung dar.

Zusätzlich zur Bewegungsrichtung wird aus dem virtuellen Entfernungsscan ein Geschwindigkeitsfaktor f_v berechnet, der in Abhängigkeit von der Hindernisannäherung die Fahrgeschwindigkeit begrenzt (Pseudocode Zeilen 36-38). Die Maximalgeschwindigkeit v_{max} ist ein in der Konfiguration vorgegebener Parameter des Navigationssystems, der Faktor f_v gibt die vorgeschlagene Geschwindigkeit als relativen Anteil von v_{max} an. f_v kann also unabhängig von der physisch möglichen oder konfigurierten Maximalgeschwindigkeit Werte zwischen 0 und 1 annehmen. Ziel der Geschwindigkeitsbegrenzung ist es, bei geringerem Abstand zu Hindernissen langsamer zu fahren, um besser auf Hindernis-Annäherungen reagieren zu können und damit die Sicherheit zu erhöhen. Dazu wird ein Referenzabstand d_{ref} definiert, der mindestens eingehalten werden muss, um die Maximalgeschwindigkeit v_{max} zu erlauben (entspricht $f_v = 1.0$). Mit abnehmender Entfernung sinkt auch f_v linear ab. Zugleich haben aber nicht alle Hindernisse den selben Einfluss, sondern dieser hängt von ihrer Position relativ zur Fahrrichtung ab: Objekte vor dem Roboter stellen ein großes Sicherheitsrisiko dar, Objekte an den Seiten sind weniger relevant und Objekte hinter dem Roboter sollten keinen Einfluss haben, da sich der Roboter auf jeden Fall von diesen entfernt. Deshalb wird der Einfluss der einzelnen Entfernungsmessungen mittels einer Gauss-förmigen Funktion gewichtet, deren Mittelwert auf der aktuellen Blickrichtung ($\varphi = 0$) liegt. Die Standardabweichung σ wird so gewählt, dass der Funktionswert bis zum Winkel $\varphi = 90°$ auf fast 0 abfällt (Abb. 5.11).

$$d'_\varphi = \frac{d_\varphi}{d_{ref}} \tag{5.8}$$

$$f_\varphi = \exp\left\{-\frac{1}{2}(\frac{\varphi}{\sigma})^2\right\} \tag{5.9}$$

$$f_v = \min_\varphi(1 - (1 - d'_\varphi) \cdot f_\varphi) \tag{5.10}$$

Die vorgeschlagene Bewegungsrichtung wird vom Navigator benutzt, um ein Geschwindigkeitskommando, bestehend aus Translations- und

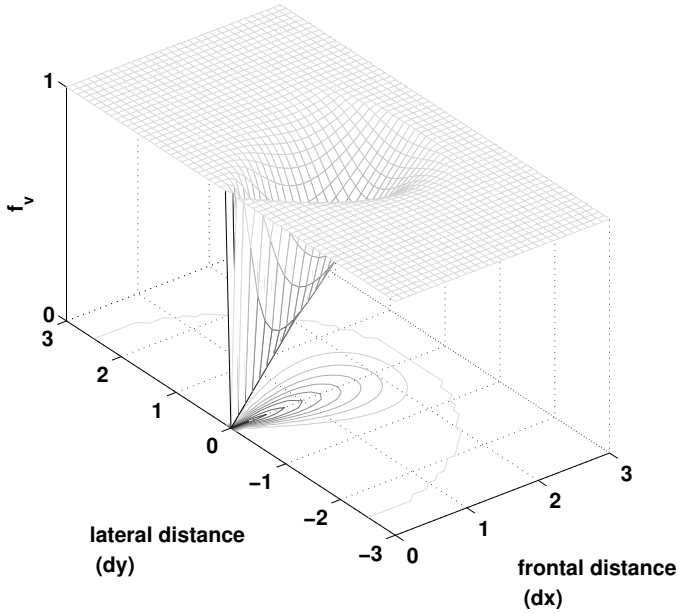

Abbildung 5.11: *Der Geschwindigkeitsfaktor f_v ist hier in Abhängigkeit von der relativen Hindernisposition [dx, dy] als 3D-Funktionsplot sowie als Konturplot dargestellt. Für diese Darstellung wurden die Werte $d_{ref} = 1.5m$ und $\sigma = 0.4 \cdot \sqrt{0.5}$ verwendet. Es ist gut zu erkennen, dass Hindernisse in frontaler Richtung bereits bei großen Abständen eine Drosselung der Geschwindigkeit bewirken, während seitliche Hindernisse wenig bis gar keinen Einfluss haben.*

Rotationsgeschwindigkeit $[v_{trans}, v_{rot}]$, zu berechnen und an den Roboter zu senden. v_{trans} und v_{rot} sind jeweils Funktionen der Bewegungsrichtung relativ zur aktuellen Orientierung (Abb. 5.12). Diese Funktionen geben die Geschwindigkeit als Anteil an der maximalen Translations- bzw. Rotationsgeschwindigkeit an, welche in der Konfiguration des Navigators festgelegt sind. Die Translationsgeschwindigkeit wird zudem mit dem Geschwindigkeitsfaktor, welcher in der lokalen Navigation berechnet wurde, verglichen, das Minimum beider Werte bestimmt die tatsächlich zu fahrende Geschwindigkeit.

Falls keine zulässige Bewegungsrichtung gefunden werden kann (d.h. selbst bei minimaler Entfernungsschwelle d_{free} existiert keine freie Richtung, die genügend nahe an der Pfadrichtung liegt), so deutet dies auf ein dynamisches Hindernis hin, welches den Pfad blockiert: Die Pfadplanung geht davon aus, dass die entsprechende Stelle passierbar ist, d.h. in der globalen Karte existiert hier Freiraum. Die lokale Navigation stellt jedoch fest, dass dies nicht der Realität entspricht. In einem solchen Fall meldet der Navigator mittels seiner Status-Flags, dass der Pfad blockiert ist. Er versucht jedoch weiterhin, eine freie Richtung zu finden, so dass die Fahrt automatisch fortgesetzt wird, falls der Weg frei wird, z.B. weil ein Mensch den Pfad blockierte und nun freigibt.

Der Navigator reagiert also - abgesehen vom Stoppen des Roboters - nicht selbständig auf eine Blockade des gewählten Pfades. Er ist allerdings in der Lage, auf ein Kommando von außen hin entsprechend zu

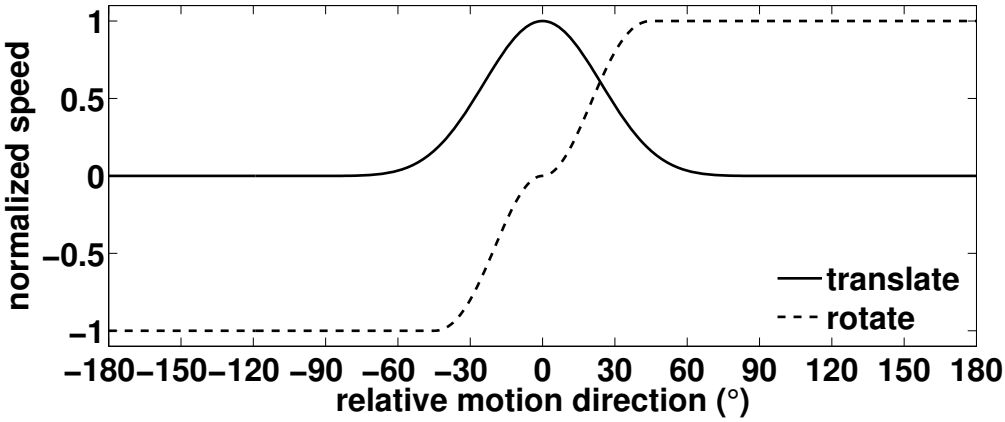

Abbildung 5.12: *Die Rotations- und Translationsgeschwindigkeit als Funktion der Bewegungsrichtung (relativ zur aktuellen Orientierung). Die Rotationsgeschwindigkeit nimmt mit steigender relativer Richtung zu, während die Translationsgeschwindigkeit abnimmt. Beide Funktionen folgen annähernd einer Gaussfunktion (für die Rotation multipliziert mit dem Vorzeichen der Bewegungsrichtung). Die relative Fahrtrichtung ändert sich sowohl durch den Pfadverlauf als auch die Drehung des Roboters während der Fahrt kontinuierlich. Die Parameter der Geschwindigkeitsfunktionen wurden experimentell so adaptiert, dass der Roboter schnell, aber ohne signifikantes Schwingen diesen gewünschten Richtungen folgen kann.*

handeln: Durch das Kommando „AdaptMap“ wird dem Navigator befohlen, die in der lokalen Karte wahrgenommene aktuelle Hinderniskonfiguration in die globale Karte zu übertragen. Dazu wird in der lokalen Karte ein virtueller Entfernungsscan berechnet, welcher einen Halbkreis vor dem Roboter in Pfadrichtung abdeckt. Der Scan wird dann dazu benutzt, die globale Karte zu adaptieren. Das „AdaptMap“-Kommando wird normalerweise vom aktiven Verhaltensmodul ausgelöst, welches die Zielanfahrt steuert (Abschnitt 6.2)

Um den Erhalt der ursprünglich erstellten Umgebungskarte zu gewährleisten, werden diese dynamischen Veränderungen in eine zusätzliche temporäre Karte eingetragen. Für die Pfadplanung werden jeweils die statische und die temporäre Karte zunächst in einer Kopie verschmolzen. Die Einträge in der temporären Karte sind nur für kurze Zeit gültig und werden fortschreitend vergessen, indem die Belegtheitswerte vor jedem Planungsvorgang mittels einer fallenden Exponentialfunktion zum neutralen Wert 0.5 hin abgewichtet werden. Die Zeitkonstante für dieses Vergessen ist ebenfalls konfigurierbar und liegt typischerweise bei einigen Minuten. Damit werden dauerhaft blockierte Stellen zwar nach

einiger Zeit wieder als frei markiert, was dazu führen kann, dass der Roboter an dieser Stelle später erneut umkehren muss. Dafür ist aber gewährleistet, dass Hindernisse, welche u.U. schon nach kurzer Zeit wieder verschwinden (Menschen, Fahrzeuge, Warenpaletten), mit Sicherheit aus der Karte getilgt werden, ohne dass die entsprechende Position zwingend vorher aufgesucht werden muss. Eine vollständige dynamische Modellierung der Umgebung oder eine Klassifizierung der blockierenden Hindernisse und entsprechende Prognose über den zukünftigen Zustand stellt eine eigenständige komplexe Aufgabe dar und findet an dieser Stelle nicht statt (siehe Abschnitt 7.2).

Nach dem Eintragen des dynamischen Hindernisses in die (temporäre) globale Karte kann das Ziel erneut gesetzt werden (ebenfalls durch ein Navigatorkommando vom aktiven Verhaltensmodul). Der Navigator berechnet dann erneut einen Pfad zum Ziel, diesmal unter Berücksichtigung des zuvor unbekannten Hindernisses. Dies kann dazu führen, dass das Objekt in einem Bogen umfahren wird, oder dass ein völlig anderer Pfad gewählt werden muss, weil z.B. ein Gang komplett versperrt ist.

Ein experimenteller Vergleich zwischen dem hier entwickelten erweiterten Vector Field Histogram und einem nachfolgenden vorgestellten antizipativen Verfahren auf Basis von Trajektorien-Sampling erfolgt in Abschnitt 5.4.4.

Eingaben

$P = [x, y, \varphi], v$ // *Roboterpose, Translationsgeschwindigkeit*

M, $\varphi_{desired}$ // *lokale Umgebungskarte, Richtung des Pfadverlaufs*

Algorithmus

Virtueller Entfernungsscan: **// je Sektor Entfernung zum nächsten Hindernis**

`S = virtual_scan(M, P)` // *virtueller Entfernungsscan in Karte M an Position P*

Dilatation: **// Vergrößerung der Hindernisse um Roboterradius**

für $s \in S$ // *für alle Sektoren*

  $d'_s = d_s - r$ // *verkürze Scan um r*

  für $s2 \in S \setminus s$ // *für alle anderen Sektoren*

    $d'_s = \min(d'_s, dilatate(\Delta\varphi(s, s2), d_{s2}, r))$ // *Gl. 5.5 - 5.7*

$threshold = d_{early}(v)$ // *initialisiere Entfernungsschwelle mit Maximalwert*

$S_{free} = \emptyset,\ L = \emptyset,\ R = \emptyset$ // *Liste der freien Sektoren, Lücken, Richtungen = leer*

Unterscheide freie und versperrte Sektoren: **// $d' > threshold$?**

für $s \in S$ // *für alle Sektoren*

  falls $d'_s > threshold$: $S_{free} = S_{free} \cup s$

falls $S_{free} = S$: Abbruch, $r_{return} = \varphi_{desired}$ // *kein Hindernis*

Fasse freie Sektoren zu „Lücken“ zusammen: **// zusammenhängende freie Sektoren**

für $s \in S_{free}$ // *für alle freien Sektoren*

  falls $s - 1 \in S_{free}$: füge s zu Lücke l_{s-1} hinzu

  sonst: erzeuge Lücke l_s, $L = L \cup l_s$

Bestimme fahrbare Richtungen: **// pro Lücke ein oder mehrere Richtungen**

für $l \in L$ // *für alle Lücken*

  falls $width(l) < 2 \cdot border$: // *border = bevorzugter Randabstand*

    falls $|(left(l) + right(l))/2 - \varphi_{desired}| < \pi/2$:

      $R = R \cup (left(l) + right(l))/2$ // *zentrale Durchfahrt*

  sonst:

    falls $\varphi_{desired} \in [right(l) + border, left(l) - border]$:

      Abbruch, $r_{return} = \varphi_{desired}$ // *Pfadrichtung befahrbar*

    sonst:

      falls $|(right(l) + border) - \varphi_{desired}| < \pi/2$:

        $R = R \cup right(l) + border$ // *rechtsseitige Durchfahrt*

      falls $|(left(l) - border) - \varphi_{desired}| < \pi/2$:

        $R = R \cup left(l) - border$ // *linksseitige Durchfahrt*

falls $R = \emptyset$: // *keine befahrbaren Richtungen*

  falls $threshold = d_{free}$: Abbruch, $\varphi_{return} = \emptyset$ // *minimale Schwelle erreicht*

  $threshold = threshold - step$ // *verringere Entfernungsschwelle*

  goto Unterscheide freie und versperrte Sektoren

Wähle beste Richtung: **// geringste Abweichung zu Pfadrichtung**

$d\varphi = \inf$

für $r \in R$ // *für alle befahrbaren Richtungen*

  falls $|r - \varphi_{desired}| < d\varphi$: $r_{return} = r,\ d\varphi = |r - \varphi_{desired}|$

Bestimme Geschwindigkeitsfaktor f_v: **// in Abhängigkeit vom Hindernisabstand**

$f_v = \inf$

für $s \in S, |\varphi_s| < \pi/2$ // *nur frontale Entfernungen*

  falls $f_v(s, d'_s) < f_v : f_v = f_v(\varphi_s, d'_s)$ // *Gl. 5.8 - 5.10*

Rückgabe

r_{return}, f_v

Abbildung 5.13: *Pseudocode des erweiterten Vector Field Histogram.*

5.4.2 Multi-Trajektorien-Navigation

Das erweiterte Vector-Field-Histogram (VFH)-Verfahren, welches im vorhergehenden Abschnitt beschrieben wurde, funktioniert robust und effizient in der Baumarkt-Umgebung. Durch die Diskretisierung der Umgebung in radiale Sektoren und die approximierte Hindernisdilatation weist es jedoch Probleme bei sehr engen Durchfahrten wie z.B. Türen auf, da es hier häufig geschieht, dass alle frontalen Sektoren als versperrt erkannt werden, obwohl der Roboter gerade noch durch die Durchfahrt passen würde. Ein weiteres Problem entsteht bei der Übertragung auf nicht-holonome Roboter: Das VFH-Verfahren geht davon aus, dass der Roboter kreisförmig ist und sich auf der Stelle um den Kreismittelpunkt drehen kann. In Umgebungen mit genügend Freiraum kann das Verfahren auch für Roboter eingesetzt werden, die dieser Forderung nur annähernd genügen. Für stark nicht-holonome Roboter oder enge Umgebungen ist es hingegen nicht geeignet.

Weiterhin ist das VFH-Verfahren aufgrund seiner rein reaktiven Natur anfällig dafür, beim Ausweichen vor Hindernissen in einer Sackgasse zu enden. Dieses Problem wird durch die Kombination mit einer lokalen Pfadplanung, wie in Abschnitt 5.4.3 beschrieben, weitgehend behoben. Die hier wiederum erfolgte hierarchische Trennung in lokale Planung und Bewegungssteuerung ist jedoch nicht optimal, eine integrierte Lösung wäre vorzuziehen.

Aus diesen Gründen wird hier ein alternatives Verfahren vorgeschlagen, welches eine antizipative Aktionsauswahl beinhaltet. Diese wird gewährleistet, indem jeweils Trajektorienabschnitte betrachtet werden, die vom Roboter gefahren werden sollen. Eine solche Trajektorie beschreibt also jeweils auch die zukünftigen Positionen und erlaubt damit eine Beurteilung der Güte innerhalb eines begrenzten Zeitfensters in die Zukunft.

Klothoiden

Als Trajektorienmodell werden Klothoid-Kurven genutzt: Klothoiden sind parametrische Kurven, die durch Anfangskrümmung c_0 und lineare

Änderung der Krümmung c_1 über die Kurvenlänge l (beginnend im Startpunkt der Klothoide mit $l = 0$) definiert sind (Abb. 5.14):

$$c(l) = c_0 + c_1 \cdot l \tag{5.11}$$

Die Krümmung c entspricht dem Inversen des Radius r einer entsprechenden Kreisbahn und stellt damit für ein entlang der Kurve bewegtes Objekt auch das Verhältnis von Rotationsgeschwindigkeit ω und Translationsgeschwindigkeit v in dem jeweiligen Punkt dar.

$$c = \frac{1}{r} = \frac{\omega}{v} \tag{5.12}$$

Aus der Definition können die Posen auf der Klothoid-Kurve (bestehend aus dem Tripel $[x, y, \varphi]$, jeweils als Funktion der fortlaufenden Länge l, $0 \leq l < \infty$) als Integral über den Verlauf beschrieben werden:

$$\begin{pmatrix} x(l) \\ y(l) \\ \varphi(l) \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ \varphi_o \end{pmatrix} + \int_{l'=0}^{l} \begin{pmatrix} \cos(\varphi(l')) \\ \sin(\varphi(l')) \\ c(l') \end{pmatrix} dl' \tag{5.13}$$

Da die Integrale i.A. nicht aufgelöst werden können, um eine geschlossene Form der Gleichung zu erhalten, werden die Kurven in Einzelpunkte mit Abstand $d\hat{l}$ diskretisiert und rekursiv approximiert:

$$\begin{pmatrix} x(\hat{l}) \\ y(\hat{l}) \\ \varphi(\hat{l}) \end{pmatrix} = \begin{pmatrix} x(\hat{l}-1) \\ y(\hat{l}-1) \\ \varphi(\hat{l}-1) \end{pmatrix} + \begin{pmatrix} \cos(\varphi(\hat{l}-1)) \\ \sin(\varphi(\hat{l}-1)) \\ c(\hat{l}-1) \end{pmatrix} d\hat{l} \tag{5.14}$$

Bei genügend kleiner Schrittweite $d\hat{l}$ sind die Abweichungen dieser Approximation von der realen Kurve vernachlässigbar gering.

Klothoid-Kurven sind aufgrund der linearen Krümmungsänderung gut geeignet zur Modellierung von Fahrzeug-Trajektorien. Für ein Fahrzeug, welches sich entlang einer klothoidalen Fahrspur bewegt, bedeutet diese beispielsweise eine ebenso lineare Änderung des Lenkwinkels und der Zentripetalkraft. Dies ist deutlich günstiger als ein abrupter Wechsel der Krümmung z.B. beim direkten Übergang von einer Geraden auf eine Kreisbahn. Klothoiden werden daher u.a. im Straßen- und Eisenbahnbau als Übergang zwischen verschiedenen Geradenstücken oder Stücken

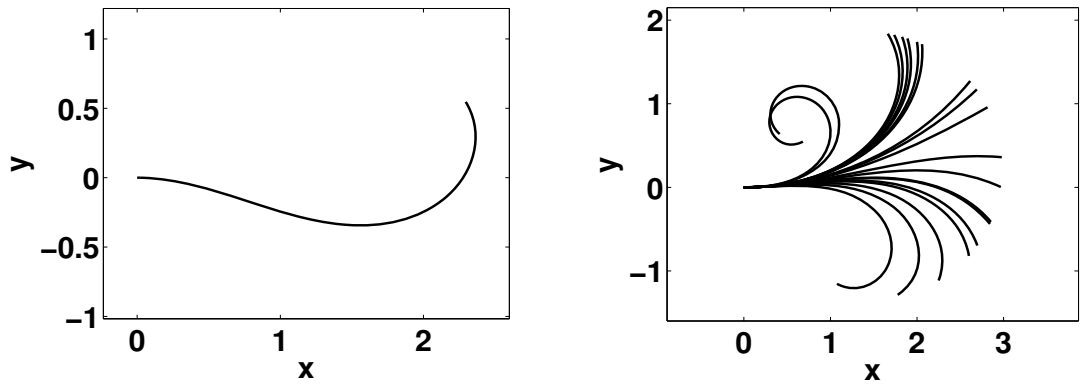

Abbildung 5.14: **Links:** *Eine Klothoide mit Anfangskrümmung* $c_0 = -0.8$ *und Krümmungsänderung* $c_1 = 1.0$ *(beginnend bei* $[x, y, \varphi] = [0, 0, 0]$*).* **Rechts:** *Ein Schwarm von 20 Klothoid-Kurven mit einheitlicher Anfangskrümmung* $c_0 = 0.3$ *und normalverteilt zufälliger Krümmungsänderung* $c_1 \sim N(0, 0.7)$ *(beginnend bei* $[x, y, \varphi] = [0, 0, 0]$*).*

verschiedener Krümmung genutzt. Aus dem selben Grund sind sie gut geeignet als prototypische Trajektorien für einen mobilen Roboter.

Trajektorien-Auswahl mittels Sampling

In einem ersten Entwicklungsschritt des vorgestellten Verfahrens werden Trajektorien jeweils in kurzen Abständen ausgewählt, indem im aktuellen Roboter-Zustand, bestehend aus Position und Geschwindigkeit $[x, y, \varphi, v, \omega]^t$, einmalig aus dem Raum der zulässigen Trajektorien, d.h. im Parameterraum $[c_0, c_1]$ eine konstante Anzahl von Kurven gesampelt wird. Aus den Parametern $[c_0, c_1]$ wird für jede Trajektorie nach Gl. 5.14 eine diskrete Folge der entstehenden zukünftigen Positionen $[x,y,\varphi]$ approximiert. Die Länge der Kurven ist dabei beschränkt, die Bewegung wird also nur in einem begrenzten Zeitfenster prädiziert. Dieses ist jedoch deutlich länger als die Zeit bis zur nächsten Auswahl, das Ende der Kurve wird niemals tatsächlich vom Roboter erreicht. Einschränkungen bzw. Vorgaben für die möglichen Parameter ergeben sich neben dem aktuellen Roboterzustand aus der zum vorherigen Zeitschritt $t-1$ gewählten Trajektorie. Der Startpunkt einer potentiellen Trajektorie ist stets die aktuelle Roboterposition. Auch die Anfangskrümmung

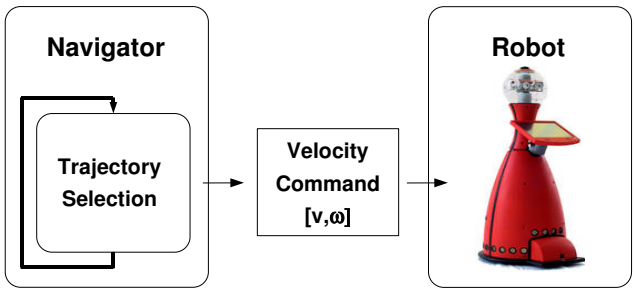

Abbildung 5.15: *Das Modul für die lokale Navigation wählt jeweils die beste Trajektorie aus und leitet daraus direkt ein Fahrkommando ab. Dieses wird an die Motorsteuerung des Roboters übergeben.*

c_0^t der Klothoide ist theoretisch durch die aktuelle Translations- und Rotationsgeschwindigkeit festgelegt.

$$c_0^t = \frac{\omega^t}{v^t} \tag{5.15}$$

Somit bleibt nur die Krümmungsänderung c_1^t als freier Parameter. Praktisch hat sich jedoch gezeigt, dass eine geringe Varianz in c_0 ebenfalls zugelassen werden sollte, um den Raum der möglichen Trajektorien und damit die Flexibilität zu vergrößern.

Nach der Generierung einer Menge möglicher Trajektorien werden diese mittels einer Kostenfunktion bewertet, um die optimale Bahn auszuwählen. Die Kostenfunktion beinhaltet verschiedene Teilziele und besteht daher aus einer gewichteten Summe mehrerer Terme, die jeweils eines der Ziele berücksichtigen:

$cost_{closest_obstacle}$: Entlang der Trajektorie werden die Hindernisabstände bestimmt, indem in regelmäßigen Abständen Normalenvektoren berechnet und entlang der Normalen belegte Zellen in der lokalen Hinderniskarte gesucht werden. Für ein einzelnes gefundenes Hindernis (eine belegte Zelle auf der Normalen) berechnen sich die Kosten aus der Entfernung von der aktuellen Position entlang der Trajektorie d_{traj} sowie der Entfernung von der Trajektorie entlang der Normalen d_{norm} (Abb. 5.16 links):

$$cost_{obstacle} = (1 - \frac{d_{traj}}{d_{traj,max}}) \cdot (1 - \frac{d_{norm}}{d_{norm,max}}) \tag{5.16}$$

$d_{traj,max}$ und $d_{norm,max}$ beschreiben dabei die gesamte Trajektorienlänge bzw. die gesamte Normalenlänge. Der Maximalwert über alle Hindernisse bestimmt $cost_{closest_obstacle}$:

$$cost_{closest_obstacle} = \max_{obstacle} (cost_{obstacle}) \tag{5.17}$$

Dieser Kostenterm berücksichtigt damit sowohl die extremste Annäherung an Hindernisse, als auch die Entfernung, die bis dahin zurückzulegen ist und damit auch die Zeit, die für eine Korrektur verbleibt.

$cost_{sum_obstacles}$: Neben der maximalen Annäherung an Hindernisse wird auch die durchschnittliche bzw. aggregierte Hindernisnähe jeder Trajektorie berücksichtigt. Dazu werden die einzelnen Werte von $cost_{obstacle}$ über alle Normalen aufsummierte:

$$cost_{sum_obstacles} = \sum_{obstacle} (cost_{obstacle}) \tag{5.18}$$

$cost_{target}$: Der Roboter soll einerseits Kollisionen mit Hindernissen vermeiden, andererseits aber sich dem Zielpunkt annähern. Daher muss jede Trajektorie auch danach bewertet werden, wie stark sie die Entfernung zum Ziel gegenüber der aktuellen Roboterposition verringert. Die stärkste Annäherung zum Ziel muss nicht unbedingt am Endpunkt der Kurve erreicht werden, sondern kann an einem beliebigen Punkt auf der Kurve liegen (Abb. 5.16 rechts). Da der Endpunkt auch normalerweise nicht erreicht wird, sondern vorher eine neue Trajektorie ausgewählt wird, wird diese minimale Entfernung zwischen der Kurve (bzw. der entsprechenden diskreten Folge von Positionen) und dem Zielpunkt $\min_l D\left[x(l), y(l)\right]$ zur Bewertung benutzt. Dazu wird auf das während der Pfadplanung berechnete Potentialfeld zugegriffen, welches zu jeder Position die Zielentfernung (unter Berücksichtigung des kürzesten tatsächlichen Pfades von dieser Position zum Ziel) enthält. Um einen normierten Kostenwert zu erhalten, wird die Weite der Zielannäherung durch die Trajektorie berücksichtigt, also die Differenz der Entfernung zum Ziel zwischen der aktuellen Position

und der dem Ziel nächsten Position auf der jeweiligen Trajektorie. Im optimalen Fall (die Trajektorie führt in gerader Linie direkt auf den Zielpunkt zu) entspricht diese Annäherungsweite der Trajektorienlänge L selbst. Im schlechtesten Fall (die Trajektorie führt in gerader Linie vom Zielpunkt weg) beträgt sie $-L$. Der Kostenwert berechnet sich aus dem Quotienten zwischen der tatsächlichen und der maximal möglichen Zielannäherung.

$$cost_{target} = \frac{D\left[x_0, y_0\right] - \min\limits_{l} \; D\left[x(l), y(l)\right]}{L} \tag{5.19}$$

$cost_{curvature}$: Bei gleichen Kosten bzgl. der Hindernis- und Zielannäherung sollen solche Trajektorien bevorzugt werden, die zu einer möglichst geradlinigen Fahrt führen. Dies bedeutet, dass geringe Krümmungen gewünscht sind. Der Kostenterm $cost_{curvature}$ ist proportional zur Krümmung der Kurve nach einer definierten Distanz, sie berücksichtigt damit sowohl die initiale Krümmung als auch deren Änderung.

$$cost_{curvature} = c_0 + c_1 \cdot l_c \tag{5.20}$$

$cost_{change}$: Falls mehrere mögliche Trajektorien ähnlich gut die gewünschten Ziele erfüllen, kann es passieren, dass in aufeinanderfolgenden Zeitschritten gegensätzliche Alternativen gewählt werden. Dies kann z.B. dann geschehen, wenn sich ein Hindernis zentral vor dem Roboter befindet und entweder links oder rechts umfahren werden kann. In diesem Fall wird ein Alternieren zwischen den Optionen möglicherweise zu vielen Richtungskorrekturen und damit unruhiger und insgesamt langsamerer Fahrt führen. Um diesen Effekt zu vermeiden, soll nach Möglichkeit eine Trajektorie bevorzugt werden, deren Parameter ähnlich zu den im vorausgegangenen Zeitschritt gewählten sind.

$$cost_{change} = \left|(c_0, c_1)^{t-1} - (c_0, c_1)^{t}\right| \tag{5.21}$$

Alle Einzelkosten werden jeweils so normiert, dass ihre Wertebereich ungefähr im einheitlichen Intervall (0,1) liegen, und anschließend gewichtet

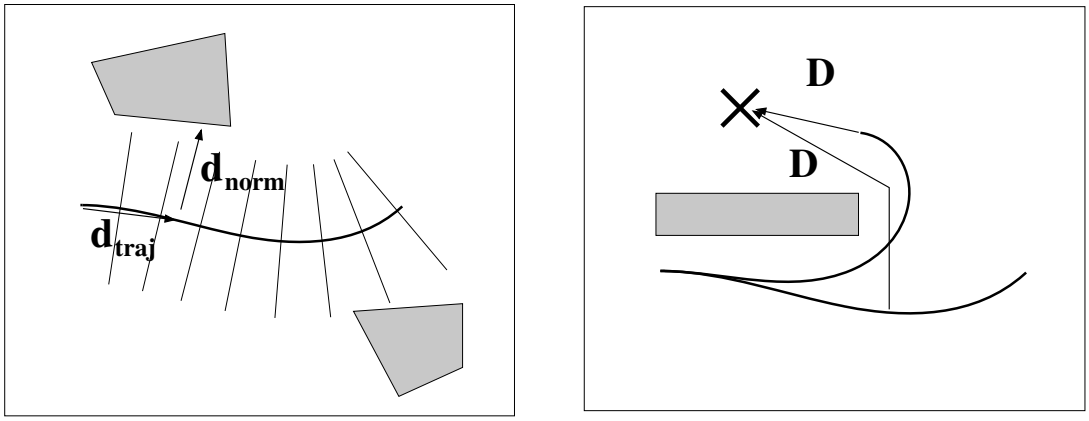

Abbildung 5.16: **Links:** *Die Hindernisannäherung einer Trajektorie wird bewertet, indem entlang der Normalenrichtungen nach belegten Zellen der lokalen Karte gesucht wird. Sowohl der Abstand des Hindernisses von der Trajektorie d_{norm} als auch die Entfernung entlang der Trajektorie d_{traj} geht in die Kostenfunktion ein.* **Rechts:** *Die minimale Pfadlänge D zwischen einem Punkt der Trajektorie und dem Zielpunkt bestimmt die Zielannäherungskosten.*

zum Gesamtkostenwert addiert. Abb. 5.17 zeigt als Beispiel eine Menge von bewerteten Trajektorien.

$$\begin{aligned} cost = \quad & \alpha \cdot cost_{closest_obstacle} \quad + \beta \cdot cost_{sum_obstacles} \quad + \gamma \cdot cost_{target} \\ & + \delta \cdot cost_{curvature} \quad + \epsilon \cdot cost_{change} \end{aligned} \tag{5.22}$$

In der Notwendigkeit zur Bestimmung dieser relativen Kostengewichte liegt ein großer Nachteil des vorgestellten Verfahrens. Die Wahl der Gewichte ist unter anderem von den Eigenschaften der Umgebung abhängig. Es erweist sich nicht immer als einfach, ein Gleichgewicht zwischen den Einzelzielen so zu erreichen, dass insgesamt ein ausgewogenes Fahrverhalten erzielt wird. Eine Überbetonung der Hinderniskosten führt dazu, dass der Roboter häufig in Sackgassen endet, da er die Sicherheit der Zielannäherung vorzieht, während umgekehrt eine stärkere Gewichtung der Zielannäherungskosten das Risiko von Kollisionen erhöht. Eventuell besteht die Möglichkeit, die Gewichtung dynamisch in Abhängigkeit von der jeweiligen Situation zu wählen, entsprechende Ansätze wurden aber noch nicht untersucht.

Andererseits weist die Bewertung mittels Kostenfunktion eine hohe Flexibilität auf, weitere Teilkosten können bei Bedarf leicht integriert wer-

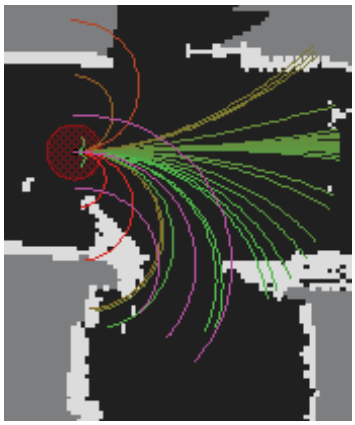

Abbildung 5.17: *Beispiel-Ansicht eines zufällig gezogenen Trajektorien-Sets (40 Trajektorien), projiziert auf die lokale Karte. Die Bewertungen der Trajektorien sind durch ihre Farben dargestellt (grün = geringe Kosten, rot = hohe Kosten). Die ausgewählte Kurve ist in magenta gezeichnet, zusammen mit der linken und rechten Begrenzung des Roboters entlang dieser Route. Es ist zu erkennen, dass diese ausgewählte Trajektorie wahrscheinlich keine kollisionsfreie Durchfahrt durch die Tür ermöglicht. Da die Trajektorie jedoch nicht bis zum Kollisionspunkt verfolgt wird, sondern vorher eine neue Auswahl erfolgen wird, ist dies nicht kritisch.*

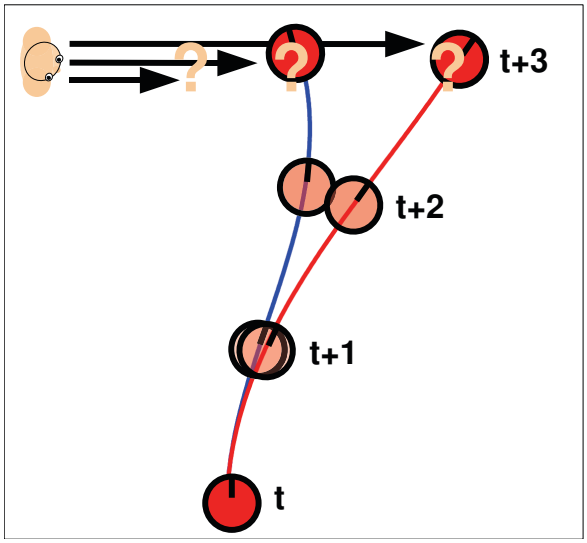

Abbildung 5.18: *Ein bewegtes Hindernis kreuzt die Spur des Roboters. Je nach relativer Geschwindigkeit wird das Objekt vor oder hinter dem Roboter passieren oder mit diesem kollidieren. Da bei gegebener Trajektorie und Robotergeschwindigkeit die Position des Roboters zu jedem zukünftigen Zeitpunkt bekannt ist, kann (mit Schätzung der Geschwindigkeit des Hindernisses) vorhergesagt werden, ob eine bestimmte Trajektorie zur Kollision mit dem bewegten Objekt führt. Dies würde im Beispiel bewirken, dass abhängig von der Geschwindigkeit des Hindernisses die blaue oder die rote Trajektorie gewählt wird, um die Gefahr einer Kollision zu minimieren.*

den. So wäre es z.B. möglich, auch bewegten Hindernissen auszuweichen, wenn deren Bewegungsrichtung und -geschwindigkeit geschätzt werden kann. Da für eine bestimmte Trajektorie die Position des Roboters (unter Annahme der eigenen Geschwindigkeit) zu jedem Zeitpunkt bekannt ist, kann auf einfache Weise bestimmt werden, ob diese Trajektorie zu einer Kollision mit dem bewegten Objekt führen würde, oder ob beide einander gefahrlos passieren können (Abb. 5.18). Noch stärker als in einer statischen Umgebung zeigt sich hierin der prädiktive Aspekt dieses Ansatzes.

Aus der best-bewerteten Trajektorie werden die zu fahrenden Rotations- und Translationsgeschwindigkeiten v, ω analog zu Gl. 5.15 bestimmt,

indem statt der Startkrümmung die zu erreichende Krümmung in einem Punkt kurz vor dem Roboter bestimmt wird. Da damit nur das Verhältnis zwischen v und ω festgelegt ist, existiert ein weiterer Freiheitsgrad zur Wahl der Geschwindigkeiten. Die Translationsgeschwindigkeit kann beispielsweise konstant oder in Abhängigkeit von den Hinderniskosten der Trajektorie gewählt werden. In der Implementation wurde als obere Schranke eine maximale Zentripetalkraft gewählt, was bewirkt, dass die Bahngeschwindigkeit mit zunehmender Bahnkrümmung verringert werden muss.

Mit diesem Ansatz konnten zunächst recht vielversprechende Ergebnisse erzielt werden, die die eingangs zur Motivation vorhergesagten Vorteile gegenüber dem VFH-Verfahren gerade bei engen Durchfahrten auch praktisch zeigten [Schröter et al., 2007b].

Trajektorien-Auswahl mittels Particle Swarm Optimization

Es erwies sich jedoch auch, dass sowohl das einfache Trajektorien-Sampling als auch die Ableitung von Fahrkommandos direkt aus dem Trajektorienverlauf keine optimalen Lösungen darstellten. Für die Trajektorienauswahl muss die dem Sampling zugrunde liegende Verteilung festgelegt werden, hierfür wird eine Normalverteilung über $[c_0, c_1]$ benutzt. Bei Wahl einer geringen Varianz werden insbesondere in Situationen, die eine starke Abweichung von der geraden Fahrt (Ausweichbewegungen, Abbiegen) erfordern, nicht immer gute Lösungen gefunden. Eine hohe Varianz und damit ein großer Suchraum erfordert hingegen das Sampling sehr vieler Hypothesen für eine ausreichende Abdeckung des Suchraumes.

Ein weiterer Kritikpunkt betrifft die Verfolgung der ausgewählten Trajektorien: Die einfache Ableitung des Fahrkommandos aus der Trajektorie gewährleistet nicht, dass der Roboter der vorgegebenen Bahnkurve tatsächlich gut folgt. Meist treten permanente Abweichungen auf, die dazu führen, dass trotz einer eigentlich gefundenen guten Bewegungstrajektorie die Durchfahrt zwischen Hindernissen nicht gelingt.

Aus diesen Gründen wurden zwei Erweiterungen integriert: Für die Trajektorienauswahl kommt statt dem reinen Sampling ein Suchverfahren im Parameterraum zum Einsatz, hierfür wurde die Particle Swarm Optimization (PSO) gewählt. Weiterhin wurde ein Trajektorienregler integriert, der explizit die Verfolgung der jeweils ausgewählten Bahn sicher-

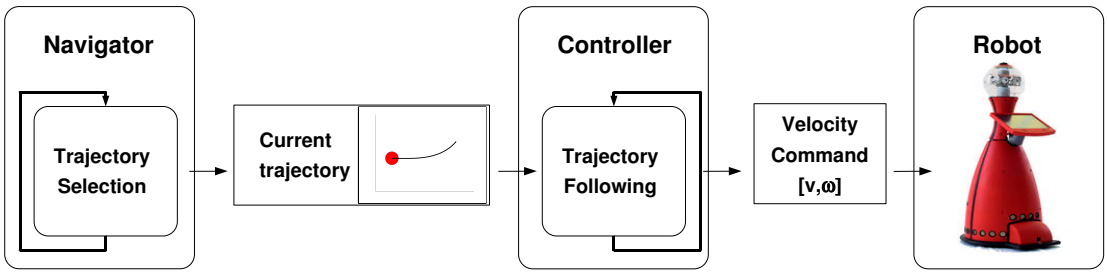

Abbildung 5.19: *Trajektorienverfolgung mit Bahnregler: Die lokale Navigation als Teil des Navigators bestimmt die beste Trajektorie, der Trajektorienregler regelt die Robotergeschwindigkeit so, dass jeweils die gewählte Trajektorie verfolgt wird.*

stellen soll. Durch die Verwendung des Reglers konnte die Intervallzeit für die Auswahl einer neuen Trajektorie deutlich erhöht werden, d.h. der Roboter folgt jeweils für eine festgelegte Zeit (typischerweise 0.5s bis 1s) der aktuellen Trajektorie. Um Kollisionen mit dynamischen Hindernissen zu vermeiden, wird währenddessen explizit auf Hindernisse auf der Bahn getestet. Falls solche detektiert werden, wird vorzeitig eine neue Auswahl angestoßen. Abb. 5.19 zeigt die Struktur des sich ergebenden Systems aus Trajektorienauswahl und -regler.

Durch Einsatz der Particle Swarm Optimization [Kennedy und Eberhart, 1995], [Particle Swarm Central, 2008] soll eine effizientere Suche im Parameterraum als mit reinem Sampling erzielt werden, speziell eine bessere Abdeckung des Suchraumes bei gleichem Aufwand. PSO ist ein stochastisches Optimierungsverfahren, welches prinzipielle Ähnlichkeiten zu Partikelfiltern und Evolutionären Algorithmen aufweist. Es wird ebenfalls eine Menge von Partikeln benutzt, wobei jedoch im Gegensatz zum Partikelfilter keine Wahrscheinlichkeitsverteilung im Zustandsraum modelliert wird. Stattdessen vollführt jedes Partikel nach bestimmten Regeln eine Bewegung im Zustandsraum (in diskreten Zeitschritten) und bewertet die dabei erreichten Zustände mittels der Fitness- bzw. Kostenfunktion, um so ein Optimum zu finden. Jedes Partikel i ist zum Zeitpunkt t definiert durch:

- den aktuellen Zustand $\boldsymbol{x}_t^{(i)}$
- die Geschwindigkeit $\boldsymbol{v}_t^{(i)}$
- das bisherige lokale Optimum (Optimum für dieses Partikel) $\hat{\boldsymbol{x}}^{(i)}$

Weiterhin ist das bisherige globale Optimum (Optimum über alle Partikel) $\hat{x}$ bekannt. Für ein Minimierungsproblem gilt beispielsweise

$$\hat{\boldsymbol{x}}^{(i)} = \min_t \boldsymbol{x}_t^{(i)} \qquad \hat{\boldsymbol{x}} = \min_i \hat{\boldsymbol{x}}^{(i)} \tag{5.23}$$

Die einzelnen Partikel sind nicht unabhängig voneinander, sondern interagieren über die gemeinsame Kenntnis des globalen Optimums. In jedem Schritt erfährt jedes Partikel eine Beschleunigung (Veränderung der Geschwindigkeit) sowohl auf das lokale als auch auf das globale Optimum zu, wobei der genaue Betrag dieser Beschleunigung jeweils unabhängig und gleichverteilt zufällig bestimmt wird.

$$\boldsymbol{v}_{t+1}^{(i)} = \omega \cdot \boldsymbol{v}_t^{(i)} + c_1 \cdot \boldsymbol{R}_{1,t}^{(i)} \cdot \left(\hat{\boldsymbol{x}}^{(i)} - \boldsymbol{x}_t^{(i)}\right) + c_2 \cdot \boldsymbol{R}_{2,t}^{(i)} \cdot \left(\hat{\boldsymbol{x}} - \boldsymbol{x}_t^{(i)}\right) \tag{5.24}$$

$$\boldsymbol{x}_{t+1}^{(i)} = \boldsymbol{x}_t^{(i)} + \boldsymbol{v}_{t+1}^{(i)} \tag{5.25}$$

ω, c_1 und c_2 sind hier feste Parameter des Verfahrens, $\boldsymbol{R}_{1,t}^{(i)}$ und $\boldsymbol{R}_{2,t}^{(i)}$ sind Zufallsvektoren der selben Dimension wie der Zustand $\boldsymbol{x}$ und werden je Partikel und Zeitschritt neu gewählt.

Für die Bestimmung der optimalen Trajektorie wird weiterhin in jedem Bewegungsschritt eine Initialverteilung von Trajektorien im Raum [c_0, c_1] durch Sampling erzeugt. Anschließend wird die initiale Verteilung jedoch mittels einer festgelegten Anzahl an PSO-Iterationen (normalerweise genügen ca. 5-8 Schritte) verbessert.

Trajektorien-Verfolgung

Die jeweils bei der Optimierung gefundene optimale Klothoide wird an den Trajektorienregler übergeben, und der Roboter soll dieser bis zur Auswahl einer neuen Trajektorie folgen. Die Regelung erfolgt, indem ein virtueller Roboter sich mit der gewünschten Bahngeschwindigkeit entlang der aktuell vorgegebenen Position bewegt. Er bestimmt damit zu jedem Zeitpunkt eine Referenz- oder Sollpose $P_r = [x_r, y_r, \varphi_r]$. Damit der reale Roboter dieser Bewegung folgt, müssen die Abweichungen zwischen der realen Pose $p = [x, y, \varphi]$ und der zeitlich veränderlichen Referenzpose minimiert werden, d.h. die Differenzpose muss zu Null

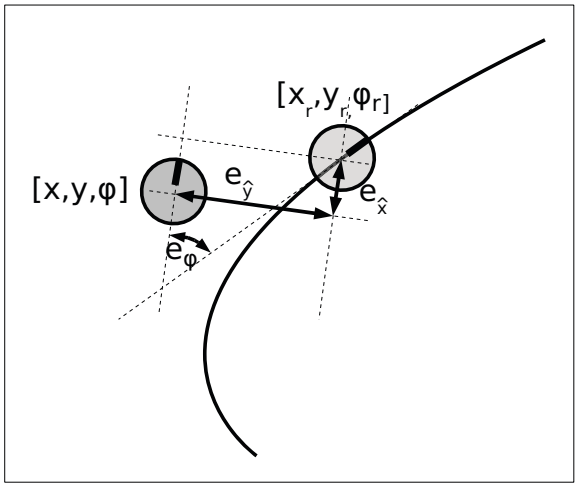

Abbildung 5.20: *Die Referenzpose $p_r = [x_r, y_r, \varphi_r]$ bewegt sich mit festgelegter Bahngeschwindigkeit entlang der ausgewählten Trajektorie. Die Abweichung zwischen der Referenzpose und der realen Roboterpose $p = [x, y, \varphi]$ wird durch die Transformation nach Gl. 5.26 in separate Fehler $e_{\hat{x}}$ in tangentialer Richtung und $e_{\hat{y}}$ in lateraler Richtung umgerechnet.*

ausgeregelt werden. Für eine einfache Separierbarkeit der Fehler wird die Differenzpose so affin transformiert, dass sich getrennte laterale und tangentiale Fehler $e_{\hat{x}}$, $e_{\hat{y}}$ (längs bzw. quer zur aktuellen Blickrichtung, Abb. 5.20) ergeben:

$$\begin{pmatrix} e_{\hat{x}} \\ e_{\hat{y}} \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x_r - x \\ y_r - y \end{pmatrix} \qquad e_\varphi = \varphi_r - \varphi \tag{5.26}$$

Die Regelung geschieht mittels der Steuergrößen v und ω (Translations- und Rotationsgeschwindigkeit). Da sie durch ein Programm auf dem Steuerrechner geschieht, handelt es sich um eine zeitdiskrete Regelung. Die Fehler werden zu diskreten (äquidistanten) Zeitpunkten bestimmt und die Geschwindigkeiten jeweils an die Motorsteuerung des Roboters übergeben. Für die Regelung der Trajektorienverfolgung wurden drei verschiedene Ansätze untersucht: Der erste und einfachste Ansatz benutzt für jeden der Fehler $e_{\hat{y}}$, $e_{\hat{x}}$ und e_φ je einen Proportional-Integral-Differential (PID)-Regler und verrechnet deren Ausgänge zu den Geschwindigkeiten v und ω (Abb. D.1 links).

Der zweite Ansatz entsteht durch Erweiterung um eine so genannte Vorsteuerung: der Roboter wird direkt mit den Referenzgeschwindigkeiten v_r und ω_r angesteuert, die PID-Regler werden zusätzlich benutzt, regeln nun aber nur den viel kleineren Restfehler aus (Abb. D.1 rechts). Bei der dritten Variante wird statt der PID-Controller ein Backstepping-Controller [Kanayama et al., 1991], [Yang et al., 2004] benutzt, der ebenfalls die Fehler $e_{\hat{x}}$, $e_{\hat{y}}$ und e_φ nichtlinear auf die Geschwindigkeiten v und ω abbildet, jedoch ohne Differenzierung und Integration des

Fehlers auskommt. Die drei Konzepte werden in Anhang D detaillierter vorgestellt.

Da kein analytisches dynamisches Modell des Roboterantriebs existiert, mussten die Parameter für alle Regler-Varianten jeweils heuristisch bestimmt und experimentell optimiert werden. Es erweist sich, dass aufgrund der geringen Parameterzahl und des gutmütigen Verhaltens durch den Verzicht auf Integral- und Differentialanteile für den Backstepping-Regler am ehesten eine Parameterkonfiguration gefunden werden kann, die eine gute Trajektorienverfolgung gewährleistet. Es wird damit keine Aussage getroffen, welcher der Regler bei *optimaler* Parametrierung das beste Fahrverhalten zeigt, allerdings ist gerade die Bestimmung der richtigen Parameter nicht trivial und der Aufwand dafür stellt ein wesentliches Entscheidungskriterium für die Auswahl dar.

Im Abschnitt 5.4.4 werden experimentelle Ergebnisse des Multi-Trajektorien-Verfahrens für typische Hindernissituationen dargestellt und mit dem Verhalten des zuvor vorgestellten VFH-Ansatzes verglichen.

5.4.3 Lokale Pfadplanung

Im vorhergehenden Abschnitt wurde ein Bewegungssteuerungs-Verfahren vorgestellt, welches auf dem Sampling von Trajektorien und damit auf der Prüfung ausgewählter Punkte in einem vorab definierten Raum von Parameterraum beruht. Durch die Bewertung der mit den jeweiligen Parametern entstehenden Trajektorien werden die zukünftigen Positionen innerhalb eines kurzen Zeithorizontes prädiziert und damit eine gewisse Antizipation erreicht. Dem soll hier nun noch einmal ein explizit planender Ansatz gegenüber gestellt werden.

Der globale Pfad wird bei Angabe eines Zielpunktes auf der globalen Karte geplant. Da der Pfadsuch-Algorithmus für alle bei der Suche bearbeiteten Knoten jeweils den kürzesten Pfad zum Zielknoten bestimmt (Abschnitt 5.3.1), ist während der anschließenden Fahrt zum Ziel keine Planung des globalen Pfades mehr erforderlich, es kann jeweils an der aktuellen Roboterposition der Pfad zum Ziel abgefragt werden. Der globale Pfad berücksichtigt allerdings keine Hindernisse, die nicht in der (statischen) globalen Karte verzeichnet sind. Ziel der lokalen Pfadplanung ist daher, den globalen Pfad in jedem Bewegungsschritt, unter

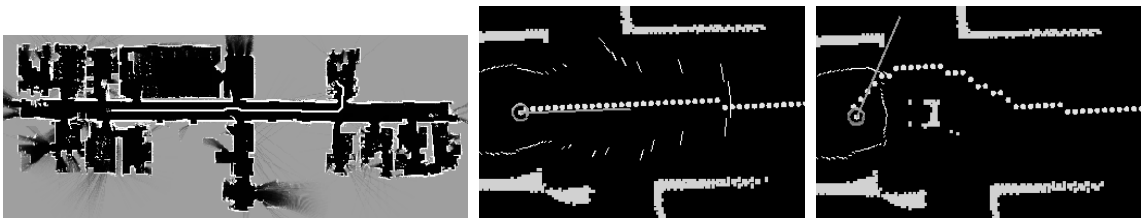

Abbildung 5.21: **Links:** *Der global geplante Pfad auf der globalen Karte.* **Mitte/Rechts:** *Der lokal geplante Pfad ohne bzw. mit dynamischem Hindernis. Wenn der Raum vor dem Roboter frei ist, so entspricht der lokale Pfad dem Verlauf des globalen Pfades. Sobald ein Hindernis im Weg steht, wird der lokale Pfad so angepasst, dass dieses umfahren wird. Der globale Pfad ändert sich dabei nicht. Die weißen Markierungen auf der lokalen Karte zeigen jeweils die Größe des Freiraumes nach der Hindernis-Dilatation (Abschnitt 5.4.1).*

Berücksichtigung der lokalen Hinderniskarte (welche die Sensorbeobachtungen akkumuliert), anzupassen.

Lokale und globale Karte werden dazu zu einer Karte verschmolzen, auf welcher anschließend die lokale Planung erfolgt. Da eine erneute Planung auf der gesamten globalen Karte zu aufwändig ist, um sie in jedem Bewegungsschritt durchzuführen, wird statt dessen nur ein begrenztes Gebiet berücksichtigt. Dazu wird ein Zwischenziel bestimmt, indem der globale Pfad von der aktuellen Position aus einige Meter weit verfolgt wird. In dem Bereich zwischen der aktuellen Roboterposition und diesem Zwischenziel wird ein lokaler Pfad berechnet. Da lokale und globale Karte unterschiedliche Auflösung aufweisen und in verschiedenen Koordinatensystemen definiert sind (siehe Einleitung Abschnitt 5.2), ist zur Überlagerung beider Karten eine Koordinatentransformation und Auflösungs-Resampling notwendig. Dies erfordert insbesondere eine gute Genauigkeit und Robustheit der Selbstlokalisation, um einen konsistenten Abgleich der Positionen zwischen den Karten zu ermöglichen.

Abb.5.22 (Mitte, rechts) zeigt den entstehenden Kartenausschnitt, auf dem die lokale Planung erfolgt. Die eigentliche Pfadsuche in dieser vereinigten Karte erfolgt mit den selben Parametern wie die ursprüngliche globale Pfadsuche. Durch die Einbeziehung der lokalen Karte wird jeweils der kürzeste Pfad zum Zwischenziel unter Berücksichtigung der zu jedem Zeitpunkt wahrgenommenen Hindernisse generiert. Es muss jedoch auf Basis dieses Pfades wiederum eine Sequenz von Fahrkommandos erzeugt werden. Das lokale Planungsverfahren wurde daher nicht allein, sondern nur in Kombination mit dem reaktiven VFH-Verfahren eingesetzt. Vor jedem Aufruf des VFH (10 mal je Sekunde) wird da-

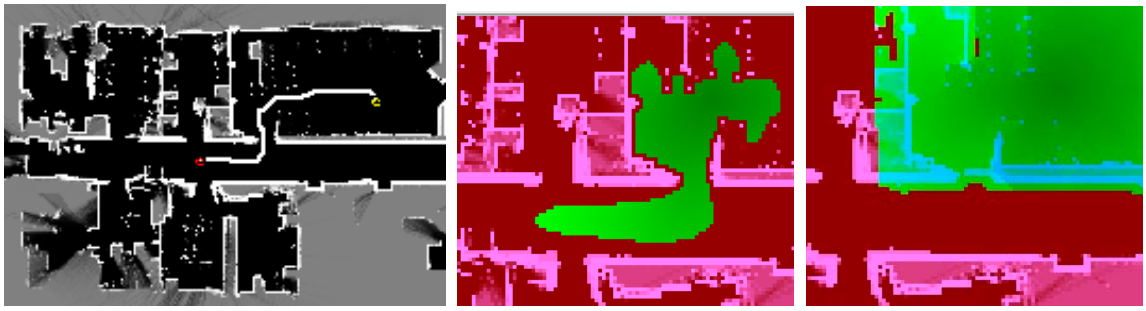

Abbildung 5.22: **Links:** *Der geplante Pfad führt hier auf der globalen Karte durch eine Tür in den angrenzenden Raum. Die Karte berücksichtigt allerdings nicht, dass die Tür aktuell geschlossen ist.* **Mitte:** *Solange die geschlossene Tür nicht wahrgenommen wird, beruht die verfügbare Information nur auf der globalen Karte, der lokale Pfad folgt damit dem globalen Pfad.* **Rechts:** *Der Roboter hat sich so weit der Durchfahrt genähert, dass die Sensoren die geschlossene Tür erkennen. In der Überlagerung von globaler und lokaler Karte existiert damit keine Verbindung mehr zwischen der Roboterposition und dem aktuellen Zwischenziel, die lokale Pfadplanung kann keinen Pfad generieren. Grün markiert sind jeweils die bei der Pfadsuche bearbeiteten Zellen/Knoten (vgl. Abb. 5.5-5.7). Durch die lokale Planung wird eine Pfad-Blockierung sehr viel eher und robuster erkannt als bei reiner Nutzung des reaktiven VFH-Verfahrens.*

zu eine lokale Planung durchgeführt, statt des globalen wird dann der lokale Pfad als Vorgabe für die gewünschte Fahrtrichtung benutzt.

Aufgrund der Wahl eines Zwischenziels und der Beschränkung des bei der Planung berücksichtigten Gebietes ist nicht garantiert, dass die lokale Planung tatsächlich immer den optimalen Pfad bezüglich des globalen Zielpunktes generiert. Wenn der ursprünglich geplante globale Pfad durch ein Hindernis versperrt ist, werden z.B. normalerweise keine Alternativ-Routen untersucht, da das bei der Planung genutzte Gebiet durch den globalen Pfad eingeschränkt wird und andere Fahrtmöglichkeiten meist nicht enthält. Allerdings wird in diesem Fall zuverlässig erkannt, dass kein Pfad zum Zwischenziel planbar ist (Abb. 5.22). Analog zu dem in Abschnitt 5.4.1 erläuterten Vorgehen wird in diesem Fall das Hindernis (aus der lokalen Karte) temporär in die globale Karte eingetragen und ein neuer globaler Pfad geplant. Die frühzeitige und robuste Erkennung einer solchen Blockade des geplanten Pfades stellt einen großen Vorteil und eine wesentliche Motivation für den Einsatz der lokalen Pfadplanung dar.

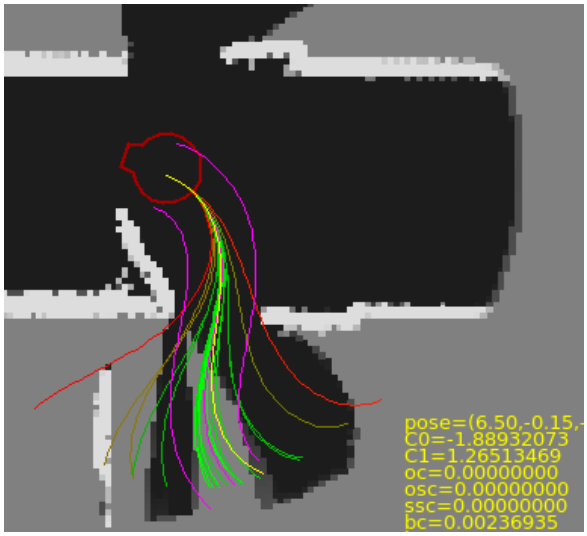

Abbildung 5.23: *Die Abbildung visualisiert die Trajektorien-Auswahl bei einer Türdurchfahrt (siehe auch Abb. 5.24). Die Bewertungen der einzelnen Trajektorien sind durch Farben dargestellt (grün = geringe Kosten, rot = hohe Kosten). Die gefundene optimale Trajektorie gewährleistet eine sehr geradlinige Durchfahrt, dadurch wird auch eine hohe Geschwindigkeit ermöglicht.*

5.4.4 Experimenteller Vergleich

In den vorherigen Abschnitten wurden verschiedene Verfahren zur lokalen Navigation entwickelt: Zunächst wurde in Abschnitt 5.4.1 eine Erweiterung des Vector Field Histogram (VFH) vorgestellt, die reaktiv mit einer Diskretisierung der Umgebung in radiale Sektoren und der Bestimmung der besten Fahrtrichtung an der aktuellen Position arbeitet. Dieses Verfahren wird mit der lokalen Pfadplanung aus Abschnitt 5.4.3 kombiniert, um eine Verbesserung gegenüber dem rein reaktiven Verhalten zu erzielen. Eine Alternative dazu stellt das antizipative Verfahren auf Basis einer Trajektorienauswahl aus Abschnitt 5.4.2 dar.

Das erweiterte VFH kommt bereits auf den Prototypen des Shopping-Roboters zum Einsatz und befindet sich seit mehreren Monaten im Dauer-Testbetrieb, die erzielten Ergebnisse im Baumarkt bestätigen die Eignung des Verfahrens für diese Umgebung. Das Multi-Trajektorien-Verfahren stellt hingegen eine neuere Entwicklung dar.

Die beiden lokalen Navigationsansätze werden einander hier in Beispielszenarien gegenüber gestellt. Dazu werden zwei typische Hindernissituationen untersucht. Obwohl ein solches stark eingeschränktes Experiment nicht den großflächigen und langfristigen Test ersetzen kann, zeigen sich wesentliche Eigenschaften der vorgestellten Systeme bereits in solchen beschränkten Szenarien. Wichtige Aspekte sind hier unter kontrollierten Bedingungen sogar besser darstellbar als in einer größer angelegten Auswertung, in der sich zwangsläufig verschiedene Einflüsse überlagern und somit die Ursachen für Fehlverhalten teilweise schwer nachvollziehbar sind.

Das erste Experiment zeigt eine enge Türdurchfahrt. Es handelt sich hierbei um eine sehr anspruchsvolle Aufgabe, da dem Roboter bei der Durchfahrt nur wenige cm Platz nach beiden Seiten bleiben. Aufgrund der Richtungsdiskretisierung geschieht es beim VFH-Verfahren in solchen Situationen häufig, dass zeitweise keine Möglichkeit zur Vorwärtsbewegung erkannt wird und der Roboter lange vor der Durchfahrt stehen bleibt. Dies ist gut in Abb. 5.24 (oben) erkennbar: An der Position direkt vor der Tür hält sich der Roboter sehr lange auf. Aufgrund wechselnder Wahrnehmungen alterniert er hier häufig zwischen Anfahren und Abbremsen, was in einer zitternden Bewegung vor der Tür resultiert. Ein weiteres Problem wird durch die lokale Planung hervorgerufen: Für die lokale Planung werden lokale und globale Karte überlagert, dazu müssen sie unter Nutzung der Selbstlokalisation in ein gemeinsames Koordinatensystem transformiert werden (siehe Abschnitt 5.2). Eine genaue „Synchronisation“ ist also abhängig von einer genauen Selbstlokalisation. Geringste Fehler können bereits zu Abweichungen in der Größenordnung einer Kartenzelle führen. Aufgrund der sehr schmalen Durchfahrt bedeuten solche Fehler allerdings, dass die Durchfahrt in der kombinierten Karte nicht mehr die notwendige Mindestbreite für den Roboter aufweist, da Hindernisse aus lokaler und globaler Karte versetzt eingetragen werden und die verbleibende freie Passage damit verringern. Die lokale Planung erkennt dann fälschlich einen versperrten Pfad. Dieser Fall kann zum Beispiel durch eine (eventuell auch nur teilweise) geschlossene Tür auch tatsächlich auftreten, kann also nicht ausgeschlossen werden. Besonders im Geschwindigkeitsdiagramm ist zu erkennen, dass der Roboter bereits vor Erreichen der Tür mehrfach kurz abbremst, dies ist durch den beschriebenen Effekt zu erklären.

In der selben Situation weist die Steuerung durch Trajektorienauswahl klare Vorteile auf: Abb. 5.23 verdeutlicht noch einmal am konkreten Beispiel das Prinzip dieses Steuerungsansatzes. In Abb. 5.24 (unten) sind die entsprechenden Ergebnisse dargestellt. Es ist gut nachvollziehbar, dass hier der Roboter in einer glatten und zügigen Bewegung die Durchfahrt passiert, dabei zwar die Geschwindigkeit in der Kurve verringert, aber vor Erreichen des Zieles nicht anhält.

Eine zweite Testsituation ist in Abb. 5.25 dargestellt. Hier wird ein „Slalom“-Kurs zwischen mehreren eigens platzierten Objekten angedeutet. Im Vergleich zum ersten Test ist hier wesentlich mehr Freiraum vorhanden, der Roboter kann die Zwischenräume zwischen den Hindernissen

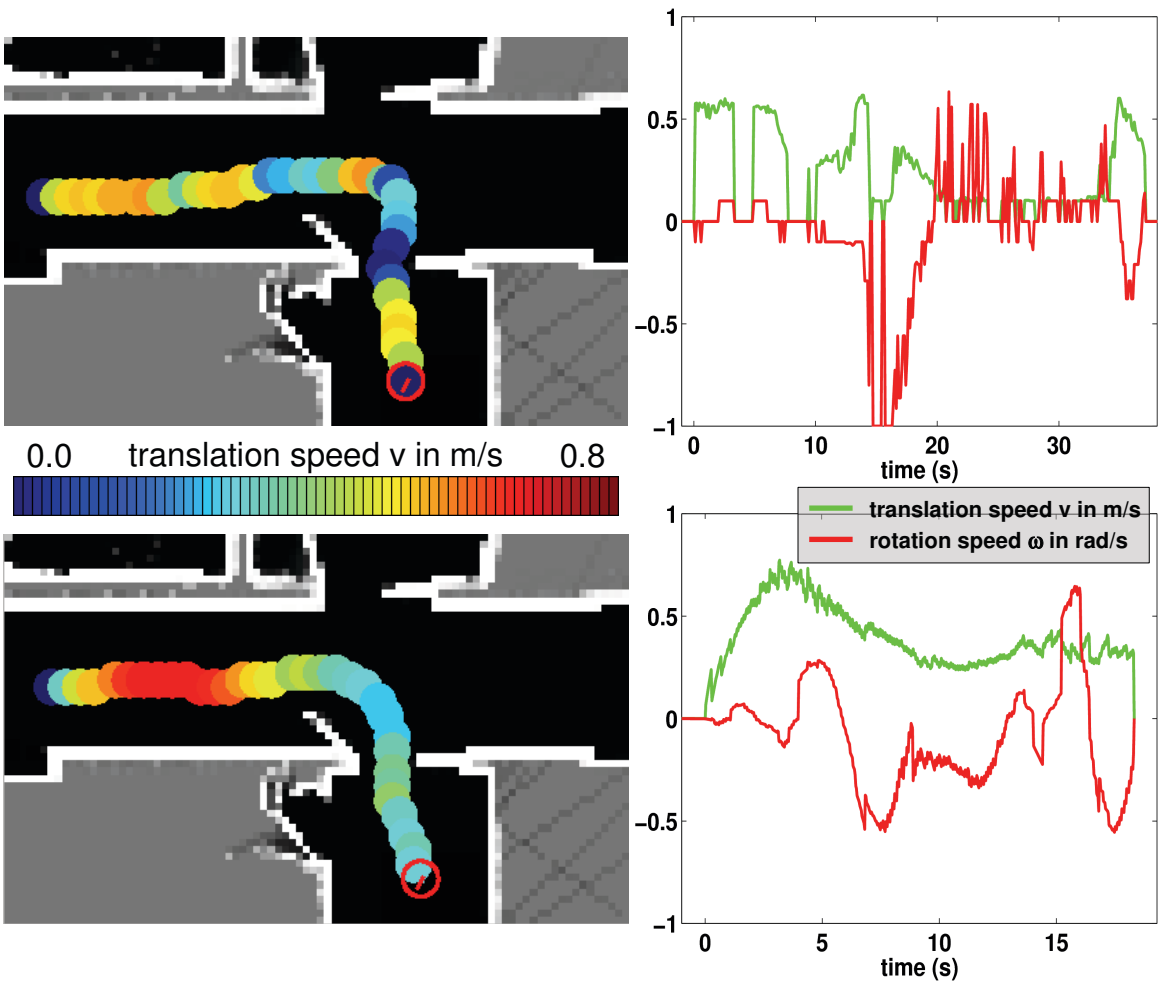

Abbildung 5.24: *Vergleich des Vector Field Histogram-Ansatzes (oben) und der Trajektorienauswahl (unten) bei einer Türdurchfahrt. Dargestellt ist jeweils links die Roboter-Bahn mit Farbkodierung der Geschwindigkeiten, sowie rechts der Verlauf der Translations- (grün) und Rotationsgeschwindigkeit (rot) über die Zeit vom Start bis zum Ziel. Es ist zu erkennen, dass mit der Trajektoriensteuerung der Roboter deutlich stetiger fährt und wesentlich schneller die Durchfahrt passiert. Das VFH-Verfahren hingegen hält sich lange vor der Tür auf.*

problemlos passieren. Für eine zügige Passage ist hier vor allem eine geradlinige Bahn mit nicht zu starken Ausweichbewegungen notwendig.

In diesem Test sind sogar leichte Vorteile für das VFH-Verfahren erkennbar. Dieses absolviert die Teststrecke problemlos. Zwar wird die Geschwindigkeit für Richtungskorrekturen vor den Hindernissen einige Male verringert, insgesamt ist aber eine zügige Bewegung zu beobachten. Das Trajektorien-Verfahren weist hier zum Teil Probleme auf, die aus der beschränkten Komplexität der Trajektorien resultieren. Häufig

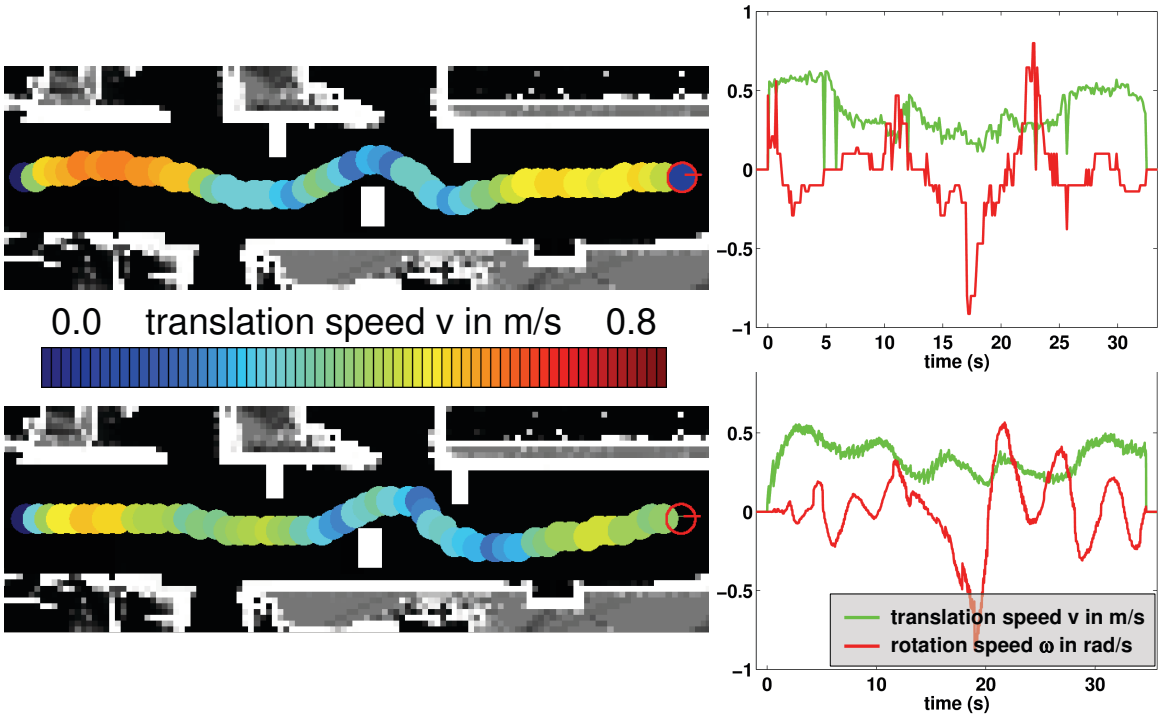

Abbildung 5.25: *Bei dieser Hinderniskonstellation, die eine relativ geradlinige Durchfahrt ermöglicht, erzielen beide Verfahren etwa vergleichbare Ergebnisse. Die Fahrt des Trajektorienauswahl-Verfahrens ist sogar etwas ungünstiger. Dies liegt in der beschränkten Komplexität der Klothoid-Trajektorien begründet, die keine komplette Bahn um die Hindernisse herum erzeugen können. Abb. 5.26 verdeutlicht dieses Problem noch stärker.*

werden solche Trajektorien ausgewählt, die erst spät vor dem Hindernis seitlich abbiegen. Die Klothoidkurven sind allerdings nicht in der Lage, bei einem Hindernis kurz vor dem Roboter eine Ausweichbewegung um dieses herum zu repräsentieren. Dies würde eine Kurve mit zwei Wendepunkten erfordern, wozu eine höhere Ordnung von Kurven notwendig ist. In einer solchen Situation muss der Roboter folglich zunächst der seitlichen Bewegung relativ weit folgen, bevor erneut ein Zurückschwenken in die ursprüngliche Fahrtrichtung möglich ist. Dies resultiert häufig in weiträumigen Ausweichtrajektorien, oder im Versagen der Trajektorienauswahl, falls dafür kein Platz vorhanden ist. Da dieses Problem hauptsächlich aus der begrenzten Komplexität der Klothoidkurven mit lediglich linearer Krümmungsänderung resultiert, ließe es sich wahrscheinlich durch die Verwendung komplexerer Trajektorien mit dem selben Verfahren reduzieren oder ganz beheben. Dabei ist allerdings zu beachten, dass bei einer Erhöhung der Anzahl der Trajektorien-Parameter der Suchraum für die Auswahl exponentiell anwächst, und

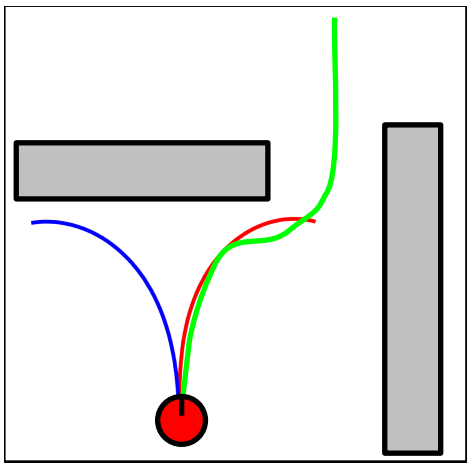

Abbildung 5.26: *Die Abbildung verdeutlicht die Einschränkungen der Klothoid-Kurven: die blaue und rote Kurve stellen Klothoiden der hier verwendeten Komplexität dar. Um in der Abbildung jedoch die Durchfahrt passieren zu können, wäre eine Bahn entlang der grünen Kurve notwendig, die eine höhere Komplexität aufweist und nicht durch eine der hier verwendeten Klothoiden repräsentiert werden kann. Unter den zur Auswahl stehenden Klothoiden wird auf Basis der Bewertungsfunktion in einer solchen Situation häufig eine im globalen Sinne suboptimale Wahl getroffen, die stark von der eigentlich gewünschten Bahn abweicht.*

damit das Finden einer guten Lösung in der zur Verfügung stehenden Zeit deutlich erschwert wird. Weitere Untersuchungen hierzu sind notwendig.

Es zeigt sich, dass das Verfahren der Multi-Trajektorien-Navigation gegenüber dem ausführlich getesteten VFH-Verfahren sowohl Vor- als auch Nachteile aufweist. Im Gegensatz zum VFH, das in der Baumarkt-Umgebung robust funktioniert, erscheint der aktuelle Stand des antizipativen Verfahrens noch nicht für den realen Einsatz geeignet. Aufgrund der erwarteten und teilweise auch bereits nachweisbaren Vorteile (besseres Verhalten bei engen Durchfahrten, Integration dynamischer Hindernisse, Einsatz auch für nicht-holonome Roboter) erscheint es aber sinnvoll, diesen Ansatz weiter zu verfolgen und dabei die genannten Vorschläge zur Behebung der wesentlichen Probleme, z.B. die Nutzung komplexerer Trajektorien, aufzugreifen. Es bleibt auch zu untersuchen, ob und wie eine Kombination der Ansätze Vector Field Histogram und Trajektorien-Auswahl/-Verfolgung erfolgen kann, um die Vorteile beider Verfahren zu vereinen.

Kapitel 6

Integration in das Roboter-Gesamtsystem

Nachdem in den vorangegangenen Kapiteln eine methodische Beschreibung einzelner Teilfunktionen erfolgte, soll nun zum Abschluss der Shopping-Assistent als Gesamtsystem betrachtet werden. Erste Ideen zum Szenario des Shopping-Roboters und dem möglichen Ablauf einer Einkaufstour wurden bereits in [Gross und Böhme, 2000b] und [Gross und Böhme, 2000a] beschrieben.

Zunächst wird in diesem Kapitel die Software-Architektur erläutert, welche eine Grundlage für alle Teilsysteme bildet und deren Zusammenarbeit koordiniert. Frühe Schritte zur Entwicklung der Steuerarchitektur und des Gesamtsystems „Shopping-Assistent“ wurden dokumentiert in [Böhme und Gross, 1999] und [Böhme et al., 2002], die aktuellere Entwicklung wurde u.a. in [Martin et al., 2005b] und [Böhme et al., 2006] dargestellt.

Im Anschluss daran werden in einem kurzen Überblick einige weitere wesentliche Module vorgestellt, die im Zusammenspiel mit den zuvor beschriebenen Teilfunktionen zur Gesamtfunktionalität des Shopping-Assistenten beitragen. Dies betrifft im wesentlichen Grundfunktionalität für die Mensch-Roboter-Interaktion wie Nutzerdetektion und -tracking, sowie Verhaltensdefinitionen und die Nutzeroberfläche. Für Detektion und Verfolgung potentieller Nutzer werden mehrere unterschiedliche Sensoren (z.B. Entfernungssensoren + Vision [Wilhelm et al., 2002a]) sowie verschiedene Merkmalsdetektoren kombiniert (Hautfarbe + Gesichtsdetektion + Körpersilhouetten [Böhme et al., 2001a]).

Die Modellierung von Personenhypothesen geschieht mittels probabilistischer Zustandsschätzer ([Martin et al., 2004], [Müller et al., 2007a]). Weiterhin kann anhand des Gesichtsbildes das Alter, Geschlecht und der emotionale Ausdruck des Interaktionspartners bestimmt werden ([Wilhelm et al., 2004], [Wilhelm et al., 2005], [Martin und Gross, 2008]), zukünftig soll dies für eine dynamische Anpassung des Dialogverhaltens des Roboters eingesetzt werden.

Abschließend wird in diesem Kapitel der aktuell erreichte Stand in der Entwicklung zum fertigen, kommerziell verwertbaren Produkt dargelegt und ein vorläufige Bewertung vorgenommen, die auf Erfahrungen aus dem mehrmonatigen dauerhaften Betrieb im Einsatzfeld sowie Nutzerbefragungen beruht.

6.1 Implementationsdetails

Mit dem Beginn der Entwicklung eines Roboters als Gesamtsystem aus Hard- und Softwarekomponenten stellte sich auch die Aufgabe, eine Software-Architektur zu schaffen. Während zuvor meist einzelne methodische Aspekte innerhalb von getrennten Experimental-Applikationen entwickelt und getestet wurden, mussten diese nun koordiniert in einer Anwendung zusammenarbeiten, Ressourcen teilen und geeignet zum Gesamtverhalten des Systems beitragen. Am Fachgebiet Neuroinformatik und Kognitive Robotik waren bereits verschiedene Hardware-Plattformen vorhanden und wurden jeweils für Entwicklung und Tests spezifischer Methoden genutzt. Wesentliche Ziele für die zu entwickelnde Architektur wurden folgendermaßen formuliert:

- Die Architektur muss einzelne Teilmodule zu einem homogenen Gesamtsystem integrieren. Jedes Teilmodul kann unabhängig von allen anderen Modulen in seiner internen Arbeitsweise verändert oder ausgetauscht werden. Die Entwicklung eines Moduls für eine bestimmte Funktionalität erfordert kein oder nur minimales Wissen über andere Module.

- Eine effiziente Verarbeitung und effizienter Datenaustausch zwischen den einzelnen Modulen sowie zwischen Modulen und Hardware wird gewährleistet. Es werden außerdem Schnittstellen zur Datenaufzeichnung und -wiedergabe geschaffen, um neben Tests

direkt auf einem Roboter auch Untersuchungen und Analysen unter Nutzung gespeicherter Sensor- oder Verarbeitungsdaten zu ermöglichen.

- Die Entwicklung neuer Module soll mit einem Minimum an Aufwand möglich sein, indem Teilaufgaben getrennt werden, so dass jedes Modul nur seine wesentliche Funktionalität implementiert. Gleichzeitig sollen einheitliche, standardisierte Schnittstellen genutzt werden, Vorlagen für neue Module können so einfach erstellt werden.

- Existierende Module sollen durch einfache Scripte und Konfigurationsdateien zu einer Experimentalumgebung oder Anwendung zusammen geschaltet oder deren Konfiguration verändert werden können, ohne dass dazu Programmierkenntnisse notwendig sind.

- Es soll eine möglichst flexible Austauschbarkeit zwischen verschiedenen Roboter-Plattformen mit unterschiedlicher Hardware- und Sensorausstattung gewährleistet werden. Eine Applikation sollte mit einem Minimum an Änderungen von einer Roboterplattform auf eine andere transferiert werden können. Falls aufgrund unterschiedlicher sensorischer Ausstattung eine Teilfunktion auf andere Weise erfolgen muss (z.B. Nutzerdetektion mit omnidirektionaler Kamera statt Frontalkamera), so sollte dies völlig transparent (unsichtbar) für andere Module geschehen.

Auf Basis dieser Anforderungen entstand die dreistufige **Robots Abstracting Application Architecture (RAAA)** (Abb. 6.1) [Martin et al., 2005b]. Die unterste Ebene bilden in dieser Architektur Module zur direkten Hardware-Ansteuerung (Sensorik, Antrieb) sowie zur Verarbeitung der Daten (sogenannte Skills - Bewegungssteuerung, Selbstlokalisation, Nutzerdetektion, etc.). Die Skill-Ebene wird gekapselt durch das Robot-Interface, welches eine abstrakte Schnittstelle zur Hardware und den Skills bildet. Auf der darüber liegenden Ebene sind die Behaviours angesiedelt, diese stellen komplexe Verhaltensweisen dar, in denen verschiedene Skills zusammenwirken (z.B. Lotsenfahrt oder zufälliges Erkundungsverhalten zur Suche nach einem Benutzer). Die oberste Ebene wird durch die eigentliche Applikation gebildet. Hier steht die Applikationslogik im Vordergrund, die sich zu den entsprechen-

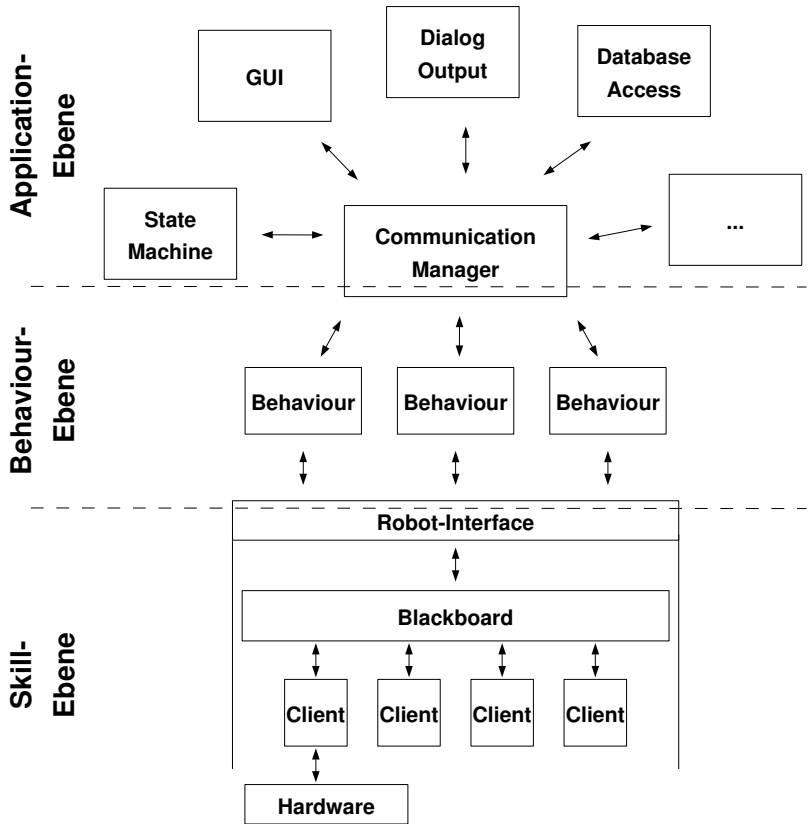

Abbildung 6.1: *Die* Robot Abstracting Application Architecture *besteht aus 3 logischen Ebenen: Skills, Behaviours und Applikationslogik. Die Trennung und daraus resultierende Abstraktion einzelner Funktionalitäten erleichtert die Übertragbarkeit und Wiederverwendbarkeit der Einzelteile sowohl zwischen verschiedenen Applikationen als auch verschiedenen Hardware-Plattformen.*

den Zeitpunkten im Ablauf der logisch darunter liegenden Behaviours bedient.

Im Sinne eines übersichtlichen Funktionsumfangs und der effizienten Implementation wurde auf Sprachunabhängigkeit und Nutzung plattform-/sprachübergreifender Schnittstellen verzichtet. Alle Software-Bestandteile sind/werden in C++ programmiert, die Einzelmodule werden zu einzelnen Bibliotheken verlinkt, in einer Applikation werden jeweils die benötigten Bibliotheken dynamisch geladen. Sowohl auf den Robotersystemen als auch auf Desktoprechnern für Entwicklung und Auswertung von Algorithmen kommt im Fachgebiet NIKR zur Zeit ausschließlich Linux zum Einsatz, die Software-Architektur ist jedoch nicht prinzipiell auf dieses System beschränkt.

Skill-Ebene

Die Skill-Ebene stellt in einer Architektur für ein „intelligentes“ System naturgemäß den umfangreichsten Bestandteil dar. Die Verarbeitung der sensorischen Inputs findet zum größten Teil hier statt. Auch die Aktionssteuerung geschieht hier, zum Teil beeinflusst durch die Behaviours (Abschnitt „Behaviour-Ebene“), die z.B. Teilziele für die jeweilige Situation vorgeben.

Der Datenaustausch zwischen den Skills beruht auf einer Blackboard-Struktur: alle Hardwaredaten und alle von einem Skill berechneten Ergebnisdaten liegen als Datenelemente in einer gemeinsamen Datenstruktur, dem so genannten Blackboard. Die Blackboard-Datenelemente haben eine einheitliche Grundstruktur und Basismethoden, welche beispielsweise den gegenseitig exklusiven Lese- oder Schreibzugriff und die Benachrichtigung der abhängigen Module bei Modifikation von Daten realisieren. Alle den Datenelementen zu Grunde liegenden Klassen sind über eindeutige Identifikatoren referenzierbar und können unter Angabe ihrer ID instantiiert werden. Auf diese Weise ist es möglich, Datenelemente auf dem Blackboard durch Angabe einer Konfigurationsdatei, durch Abspielen eines zuvor aufgezeichneten Daten-Logfiles oder durch Skills selbst anlegen zu lassen. Zur Laufzeit ist jedes Datenelement außerdem durch einen eindeutigen Namen (z.B. „Robot.Sonar“ oder „Image.Panorama“) gekennzeichnet.

Die Skills sind als sogenannte Blackboard-Clients realisiert. Jeder Blackboard-Client implementiert eine Standard-Schnittstelle zum Zugriff auf das Blackboard und dessen Datenelemente sowie zur Ablaufsteuerung. Dies umfasst z.B. die Initialisierung und Konfiguration mittels einer einheitlichen Parameterstruktur, sowie das Starten und Stoppen eines Clients. Ein Client benennt Blackboard-Datenelemente, auf die er Lese- oder Schreibzugriff benötigt. Um von den vorab nicht bekannten Blackboard-Element-Namen unabhängig zu sein, definiert der Programmierer des Clients lediglich „logische“ Namen, welche die Bedeutung der entsprechenden Daten innerhalb des Skills beschreiben (z.B. „RangeData“). Vor der Verknüpfung des Clients mit dem Blackboard muss eine Zuordnung der logischen Namen mit den „physischen“ Blackboard-Namen erfolgen (z.B „RangeData“ $\leftrightarrow$ „Robot.Laser“). Diese Abbildung wird von der Instanz, welche die Clients anlegt, (normalerweise auch basierend auf Konfigurationsskripten) festgelegt. Beim Herstellen der Ver-

bindung von Client und Blackboard werden die entsprechenden Daten-Elemente gesucht und dem Client direkt als Datenzeiger zur Verfügung gestellt. Durch diese Mechanismen wird eine simple und flexible Verknüpfung über den Elementnamen, gleichzeitig aber ein schneller Zugriff zur Laufzeit ermöglicht.

Jeder Client kann sich über Veränderungen, die von einem anderen Client an Daten auf dem Blackboard vorgenommen werden, informieren lassen. Er registriert sich dazu als Callback-Listener. Ein Client, welcher Daten verändert, ruft nach Beendigung des Schreibzugriffs eine Methode auf, die dafür sorgt, dass alle Listener benachrichtigt werden. Clients kommunizieren untereinander ausschließlich über das Blackboard. Alle Daten, die von einem Client zu einem anderen weitergeleitet werden sollen, müssen als Blackboard-Datenelemente angelegt werden und stehen damit auch zentral für alle anderen Clients zur Verfügung.

Das Daten-Logging ist als zentraler Dienst des Blackboard integriert. Wenn dieses aktiviert ist, werden die dafür ausgewählten Daten-Elemente bei jeder Änderung jeweils mit dem aktuellen Zeitstempel in ein Logfile geschrieben, welches zu einem späteren Zeitpunkt wieder (in Echtzeit oder beschleunigt/verlangsamt) ins Blackboard zurück gespielt werden kann. Die einzelnen Datenelemente müssen dafür jeweils eine standardisierte Log-Methode implementieren, sie legen damit fest, in welchem Format ihre internen Daten im File gespeichert werden. Durch den Logging/Playback-Mechanismus können beispielsweise jederzeit Sensordaten (u.a. auch Kamerabilder) aufgezeichnet und später zum Zwecke der Entwicklung und des Tests von Algorithmen auf einem beliebigen anderen Rechner wiedergegeben werden.

Zu den meisten Blackboard-Daten sind außerdem ergänzend Visualisierungen vorhanden, welche die Daten als Widgets darstellen können. Es existiert ein grafisches Frontend für das Blackboard, welches dynamisch den Zustand des Blackboards und alle vorhandenen Datenelemente sowie alle Clients darstellt. Weiterhin ist über diese Oberfläche die Parameter-Einstellung der Clients zur Laufzeit und die Steuerung von Logging- und Playback-Funktionalität möglich.

Die Skill-Ebene allein stellt damit bereits ein mächtiges Framework für experimentelle Anwendungen dar. Für die meisten Testprogramme ist lediglich das Blackboard und einer oder wenige Clients notwendig. Solche Anwendungen können sehr einfach erstellt werden, es muss nur ein Konfigurationsfile mit der Definition der Clients und ihrer jeweils

benötigten Blackboard-Datenelemente sowie ihrer jeweiligen Parametrierung erstellt werden.

Robot-Interface

Um die selbe Anwendung auf verschiedene Roboter zu übertragen, sind unter Umständen auf jedem Roboter unterschiedliche Skills und Hardware-Clients notwendig. Beispielsweise definiere ein abstraktes Verhalten, dass eine bestimmte Reaktion erfolgen soll, wenn ein potentieller Nutzer sich dem Roboter nähert. Auf welche Weise der Nutzer detektiert werden kann, hängt allerdings von der jeweils spezifischen sensorischen Ausstattung des Roboters ab. So könnten verschiedene Kamera-Systeme, Entfernungs-Sensoren oder eine Kombination aus diesen genutzt werden. Auf jedem Roboter werden dazu jeweils die Clients zur Hardware-Ansteuerung sowie zur Nutzer-Detektion und evtl. -tracking aus den entsprechenden Sensordaten benötigt. Das selbe Problem besteht, wenn Roboter mit verschiedenen Antriebssystemen sich autonom bewegen sollen und daher unterschiedliche Navigationskonzepte verwendet werden.

Solche Details sollen allerdings vor der Anwendung/dem Verhalten verborgen sein, da es für dessen Funktion nicht relevant ist. Der Roboter (als Einheit von Hardware und Ansteuerungs-/Auswertungssoftware) soll statt dessen nach außen eine standardisierte Schnittstelle zeigen, über welche es möglich ist, die wesentlichen Funktionen ohne Kenntnis des konkreten Aufbaus anzusprechen. Dies ist Aufgabe des Robot-Interface.

Das Robot-Interface besteht aus zwei Teilen, dem Status- und dem Command-Interface. Das Status-Interface realisiert alle lesenden Zugriffe auf den Roboter, das Command-Interface die Schreibzugriffe. Ein abstraktes Basis-Interface definiert alle Status-Informationen und alle Kommandos, die von einem Roboter unterstützt werden sollen. Als Kommandos stehen zum Beispiel das Setzen einer Zielposition für die Navigation und die Aktivierung der autonomen Fahrt, das Ausrichten des Blickes in einer bestimmten Richtung oder die Ausgabe von gesprochenem Text zur Verfügung. Das Status-Interface liefert Zustandsinformationen u.a. zur aktuellen Position und dem geplanten Pfad, wahrgenommene Personen in der Umgebung des Roboters, oder Analysedaten über den aktuell mit dem Roboter interagierenden Nutzer. Diese Bei-

spiele dienen nur der Illustration, auf eine vollständige Auflistung soll hier verzichtet werden.

Für einen konkreten Roboter wird dann jeweils ein spezielles Robot-Interface definiert, welches diese abstrakt Definition unter Berücksichtigung der konkreten Ausstattung implementiert. Das Robot-Interface legt dazu die Blackboard-Clients an, die zur Hardware-Ansteuerung und zur Auswertung der Daten benötigt werden. Die konkrete Definition dieser Clients und deren Parameter erfolgt wiederum mittels anpassbarer Konfigurations-Dateien, auf diese Weise ist auch ein spezielles Interface für einen konkreten Roboter noch flexibel in Bezug auf die verwendeten Algorithmen zur Datenverarbeitung.

Um mit den selben universellen Mechanismen auf die benötigten Daten zugreifen zu können, verhält sich das Robot-Interface „nach innen“ (also gegenüber dem Blackboard) ebenfalls als Blackboard-Client.

Behaviour-Ebene

Als Behaviour werden im Kontext der RAAA Module bezeichnet, welche unabhängig von der konkreten Plattform bestimmte Verhaltensweisen des Roboters implementieren. Sie können dazu über das Robot-Interface auf die Ergebnisse verschiedener Skills zum Beispiel zur Nutzerdetektion zugreifen und Aktionen des Roboters steuern. Als Beispiel sei ein Guide-Behaviour genannt, welches die Anfahrt einer Zielposition bei gleichzeitigem Kontakt-Halten mit dem Interaktionspartner realisiert. Die Behaviours, welche Bestandteil der Anwendung „Shopping-Assistent“ sind, werden in Abschnitt 6.2 genauer beschrieben.

Durch die separate Implementation der einzelnen Behaviours sind diese inhärent unabhängig von einer spezifischen Applikation und können leicht zwischen verschiedenen Anwendungen transferiert werden. Gleichzeitig können durch die Anbindung an den zentralen Communication Manager (siehe Abschnitt „Application-Ebene“) in einer konkreten Anwendung Zustände signalisiert und Events ausgelöst werden, die bestimmte Reaktionen durch andere Module hervorrufen. Auf diese Weise kann z.B. das Guide-Behaviour ein Signal „User Lost“ senden, wenn der Kontakt zum Nutzer verloren geht, das Dialog-Modul reagiert darauf mit einer passenden Sprachausgabe. Ein weiteres Beispiel ist das Anzeigen einer bestimmten Oberfläche auf dem Display durch das GUI-

Modul, wenn im FindUser-Behaviour ein potentieller Interaktionspartner angesprochen wird.

Application-Ebene

Auf der obersten Ebene befinden sich alle Module, die für die Anwendung und die spezifische Applikationslogik benötigt werden. Da diese von der jeweiligen Anwendung abhängen, ist die Menge und Auswahl der Module jeweils verschieden, einige Standard-Module werden jedoch häufig verwendet. Dazu zählen z.B. das State-Machine-Modul (Zustandsgraph mit Übergangsbedingungen zwischen den Zuständen), das GUI-Modul (konfigurierbare graphische Oberflächen zur Informationsdarstellung und Eingaben) sowie das Dialog-Output-Modul (Generierung von Sprachausgabe und Display-Text).

Die einzelnen Module kommunizieren untereinander über den Communication Manager. Jedes Modul stellt dazu eine einheitliche Schnittstelle zur Verfügung, über die Signale und Events generiert sowie entgegen genommen werden können. Der Communication Manager verknüpft diese untereinander (wobei die konkrete Verschaltung wiederum frei konfiguriert werden kann) und regelt die Abarbeitung der generierten Events.

Die Behaviours können jeweils durch ein Modul aktiviert oder deaktiviert werden, z.B. kann das State-Machine-Modul bestimmte Zustände der Applikation jeweils mit festgelegten Behaviours verknüpfen. Das Robot-Interface ist aus softwaretechnischer Sicht ebenfalls in einem Modul gekapselt und kann somit bei Bedarf von anderen Modulen angesprochen werden.

6.2 Komponenten der Applikation „Shopping-Assistent“ - Nutzer-Interaktion, Verhalten und Anwendungslogik

Neben der Hardware-Anbindung (Roboter-Antrieb und -Odometrie, Kopfsteuerung, Sonar, Laser, Kameras) sind die in Kapitel 5 vorgestellten Verfahren zur Lokalisation und zur autonomen Navigation sowie die

Erstellung der lokalen Karte aus den Sensormesswerten von Laser- und Sonar-Sensoren ebenfalls als Skills implementiert. In diesem Abschnitt sollen einige weitere wichtige Skills in einem Überblick vorgestellt werden, welche der Detektion und dem Tracking von Interaktionspartnern dienen. Weiterhin werden die als Behaviour-Module implementierten Verhaltensweisen beschrieben.

Skills

Um potentielle und aktive Nutzer des Roboters zu erkennen, kommt ein Personen-Tracker zum Einsatz [Müller et al., 2007a]. Ziel des Trackings ist, alle Personen in der unmittelbaren Umgebung des Roboters zu erfassen und dabei während eines zusammenhängenden Interaktionszyklus jeweils die Position des aktuellen Interaktionspartners zu verfolgen. Da ein einzelner Sensor oder Detektor in einer realen Umgebung mit wechselnden Bedingungen nicht robust genug ist, um eine stabile Erkennung und Nachverfolgung zu gewährleisten, kommen dafür verschiedene Input-Cues zum Einsatz. Diese werden im Personen-Tracker probabilistisch integriert [Martin et al., 2005a]. Sowohl die verschiedenen Detektoren, welche unterschiedliche Eingangsdaten verwenden, als auch der Tracker selbst sind als Skills implementiert (Abb. 6.2, 6.3).

Hautfarbe: Ein einfaches, wenn auch wenig spezifisches Merkmal zur Personendetektion ist die Suche nach Hautfarbe im Kamerabild. Da für die Farbklassifikation nur eine geringe Auflösung benötigt wird, andererseits ein großer Sichtbereich von Vorteil ist, nutzt dieser Detektor das Panoramabild der omnidirektionalen Kamera.

Bewegung: Vor einem statischen Hintergrund zeichnen sich Menschen durch ihre Bewegung aus. Diese kann durch Vergleich aufeinander folgender Bilder einer Kamera detektiert werden. Ebenso wie die Hautfarbe wird die Bewegung im Panoramabild detektiert. Da eine vollständige Kompensation der Eigenbewegung nicht möglich ist, wird dieses Merkmal nur bei stehendem Roboter berücksichtigt.

Beinpaar: Ein weiteres verbreitetes Merkmal zur Personendetektion ist die Suche nach Beinpaaren im Laser-Scan. Die Laser-Messungen sind genügend hoch aufgelöst, so dass sich die Beine

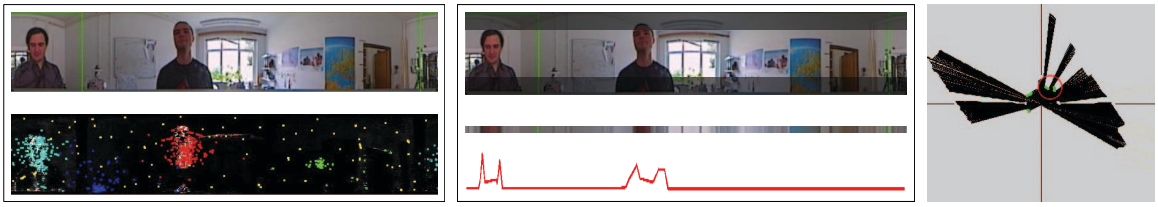

Abbildung 6.2: **Links:** *Durch ein Farbmodell werden Hautfarbpixel im Panoramabild bestimmt und mittels eines Multi-Hypothesen-Partikelfilters einzelne Hautfarb-Cluster getrackt.* **Mitte:** *Durch Integration über die y-Achse des Panoramabildes entsteht ein Spaltensummen-Bild, auf diesem werden durch Detektion von Veränderungen in aufeinander folgenden Bildern Bewegungen detektiert.* **Rechts:** *Durch Segmentierung des Laser-Scans und Klassifikation von Segmentpaaren werden potentielle Beinpaare gefunden.*

einer vor dem Roboter stehenden Person als getrennte Minima im Entfernungsprofil ausbilden und somit erkannt werden können. Da der Laser-Scanner nur den Bereich vor dem Roboter erfasst, ist die Erkennung von Beinpaaren auf diese Richtung eingeschränkt.

Gesicht: Ein häufig benutztes Merkmal zur Detektion von Personen ist die Erkennung von Gesichtsstrukturen im Kamerabild [Viola und Jones, 2001]. Dieses ist ein sehr spezifisches Merkmal, allerdings muss die entsprechende Person ungefähr frontal zum Roboter ausgerichtet sein, um erkannt zu werden. Da die Auflösung bestimmt, bis zu welcher Entfernung Gesichter im Bild erkannt werden können, wird hierfür der hoch aufgelöste Bild-Ausschnitt benutzt. Dieser kann zwar gesteuert werden, so dass er jeden Richtung um den Roboter erfassen kann, jedoch ist zu einem Zeitpunkt jeweils nur ein bestimmter Bereich sichtbar. Der hoch aufgelöste Bereich wird jeweils dem Interaktionspartner nachgeführt. Weitere Personen können nur dann über ihr Gesicht detektiert werden, wenn sie ungefähr in der selben Richtung zum Roboter stehen.

Kartenvergleich: Sowohl die globale Umgebungskarte als auch die Position des Roboters (Selbstlokalisation, Abschnitt 5.1) sind bekannt. Durch Vergleich der statischen globalen Karte mit der lokalen Karte, welche die aktuelle Umgebung des Roboters modelliert, können Unterschiede in der Objektkonfiguration erkannt werden. Diese bilden Hypothesen für Nutzerpositionen, allerdings können sie auch durch andere Objekte verursacht werden.

Abbildung 6.3: **Links:** *Personen werden im Kamerabild durch Suche nach entsprechenden Gesichtsstrukturen gefunden.* **Rechts:** *Durch Vergleich der dynamischen lokalen Karte mit der statischen globalen Karte können dynamische Objekte gefunden werden, diese stellen Hypothesen für Positionen von (potentiellen) Interaktionspartnern dar.*

Alle Einzeldetektoren bestimmen jeweils eine Wahrscheinlichkeitsverteilung für die Position der detektierten Person(en) in Form einer Gauss-Verteilung in polaren $[r, \varphi]$ - Koordinaten. Da zum Beispiel für Hautfarb- und Bewegungsdetektion auf dem Kamerabild keine Entfernung bestimmt werden kann, wird für diese eine sehr große Kovarianz für den Abstand r festgelegt. Die Detektionen werden den aktuellen Personen-Hypothesen im Tracker zugeordnet und durch Kalmanfilter aggregiert (Abb. 6.4), oder eine neue Hypothese generiert, wenn die Detektion keiner zuvor getrackten Person entspricht.

Aufbauend auf der Detektion wird außerdem das Gesicht des Interaktionspartners analysiert, um dessen Alter und Geschlecht zu bestimmen und aus der Mimik Rückschlüsse auf Emotionen zu ziehen [Wilhelm et al., 2005]. Zukünftig sollen diese Daten verwendet werden, um den Dialog spezifisch an den jeweiligen Nutzer und dessen Bedürfnisse anzupassen.

Behaviours

Folgende Verhaltensweisen sind als Behaviour-Module implementiert und können von der Anwendung in den entsprechenden Situationen aktiviert werden:

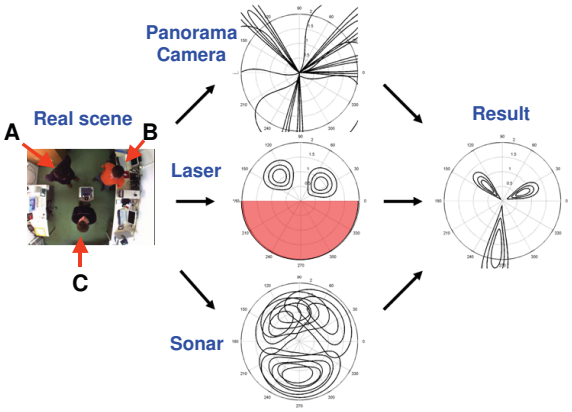

Abbildung 6.4: *Die Ergebnisse der einzelnen Sensoren und Detektoren werden probabilistisch mittels Kalmanfilter-Update integriert, die erreichte Genauigkeit und Robustheit wird gegenüber den Einzeldetektoren erhöht.*

FollowPath: Der Roboter bewegt sich autonom zu einer Zielposition und vermeidet dabei Kollisionen mit Hindernissen. Falls das Robot-Interface (als Weiterleitung vom Navigator, Abschnitt 5.2) meldet, dass ein unerwartetes Hindernis den Weg blockiert und damit der geplante Pfad nicht zum Ziel führt, veranlasst das Behaviour eine entsprechende Anpassung der globalen Karte. Anschließend wird erneut geplant und möglicherweise ein alternativer Pfad zum Ziel gewählt. Falls das Ziel nicht erreichbar ist (Pfad blockiert und keine Umgehung möglich, z.B. eine geschlossene Tür), muss das FollowPath-Verhalten abgebrochen oder ein anderes Ziel gewählt werden.

FindUser: Der Roboter pendelt zufällig zwischen verschiedenen vorgegebenen Stationen in der Umgebung und versucht, einen Nutzer zu finden. Wird ein potentieller Nutzer detektiert, so bewegt sich der Roboter zu diesem hin, spricht ihn an und lädt zur Interaktion ein. Dem Verhalten liegt die Beobachtung zu Grunde, dass potentielle Nutzer den Roboter deutlich besser als interaktiven Assistenten wahrnehmen, wenn dieser durch Bewegung aktiv erscheint und darüber hinaus von sich aus Kontakt aufnimmt. Die während der Suche anzufahrenden Ziele können jeweils mit einer Gewichtung versehen werden, so dass sich der Roboter unterschiedlich häufig an verschiedenen Positionen aufhält (je nach Wahrscheinlichkeit, interaktionswillige Nutzer anzutreffen).

Guide: Der Roboter führt den Nutzer zu einer Zielposition. Das FollowPath-Verhalten wird hierbei erweitert um das Bemühen, Kontakt zum Nutzer zu halten. Bei Zurückbleiben des Nutzers bremst oder wartet der Roboter, bei Kontaktverlust wird der Nutzer zur

Rückkehr aufgefordert. Das Verhalten kann beendet werden z.B. durch Erreichen des Ziels oder durch endgültiges Verlieren des Nutzers.

Follow: Der Roboter folgt dem Nutzer und weicht dabei Hindernissen aus. Ebenso wie beim Guide-Behaviour wird versucht, den Kontakt zum Nutzer nicht abreißen zu lassen.

Außer beim reinen Folgeverhalten Follow wird jeweils auch die Selbstlokalisation (Abschnitt 5.1) benötigt, da die Kenntnis der aktuellen Position eine Voraussetzung für gerichtete Fahrt des Navigationsskills ist.

Sowohl im Guide- als auch im Follow-Verhalten wird kontinuierlich die Position des Interaktionspartners verfolgt. Falls der Nutzer sich z.B. absichtlich zu weit vom Roboter entfernt oder durch Hindernisse die Sichtverbindung (Kamera, Laser) verloren geht, so kann der Roboter den Kontakt verlieren. In diesem Fall wird der Roboter den Nutzer auffordern, zurück zu kommen. Zur Fortsetzung der Interaktion muss dieser seine Bereitschaft durch Eingabe auf dem Touchscreens bestätigen. Zukünftig soll hier eine Unterscheidung zwischen verschiedenen Personen anhand des Gesichts und der allgemeinen Erscheinung (Kleidung, Körperstatur) die automatische Wiedererkennung ermöglichen. Dadurch wird der Roboter selbst in die Lage versetzt, zu erkennen, dass der zuvor bediente Nutzer zurückkehrt, so dass eine explizite Rückmeldung durch diesen nicht mehr erforderlich ist.

Anwendung

Zur Bedienung des Shopping-Assistenten steht dem Nutzer eine graphische Oberfläche zur Verfügung, über die alle angebotenen Dienste erreichbar sind (Abb. 6.5). Der wichtigste Service ist die Suche nach angebotenen Artikeln mittels einer Stichwortsuche oder in einer nach Produktgruppen gegliederten hierarchischen Auflistung. Diese ist an das Warenwirtschaftssystem des jeweiligen Marktes angekoppelt und beinhaltet zusätzlich die Standorte der erfassten Artikel. Zu gefundenen Produkten wird jeweils der Standort auf einer Übersichtskarte sowie der kürzeste Weg (Pfadplanung) angezeigt, weiterhin wird dem Kunden angeboten, ihn direkt dorthin zu führen. Als zusätzliche Dienste sind beispielsweise die Preisabfrage mittels eines integrierten Barcode-Scanners oder die Auflistung aktueller Sonderangebote integriert. Falls

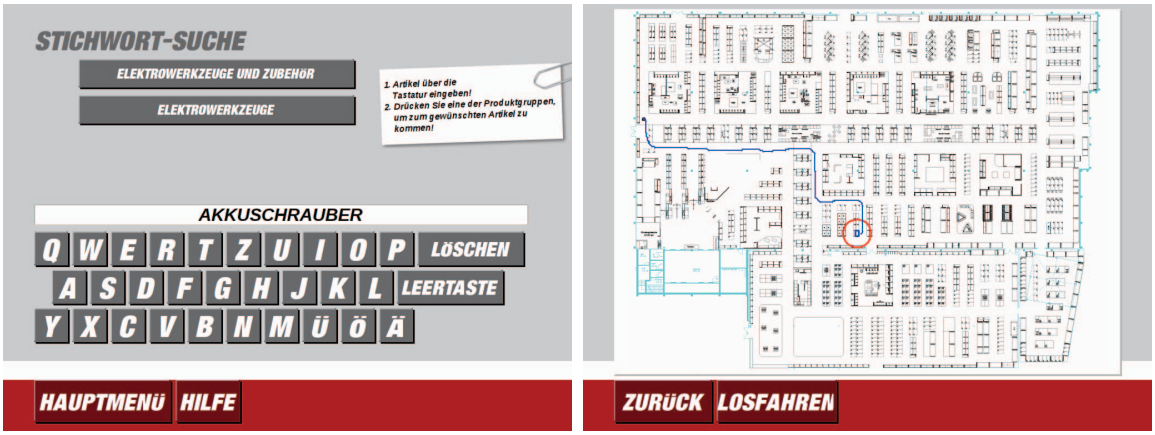

Abbildung 6.5: *Ausgewählte Ansichten der graphischen Oberfläche des Shopping-Assistenten, welche mittels eines Touch-Displays zu bedienen ist.*

der Kunde zusätzliche Beratung benötigt, so kann er direkt über den Roboter eine Audio+Video-Verbindung zu einem Fachberater (Desktop-Terminal) herstellen.

Einen zentralen Bestandteil der Anwendung bildet das State-Machine-Modul, welches den Ablauf steuert. Die wichtigsten Zustände sind:

Init: Initialisierung der Anwendung nach dem Einschalten.

SearchUser: Suche nach einem Nutzer. Aktiviert das Verhalten FindUser.

ContactUser: Ein potentieller Nutzer wurde gefunden, er wird angesprochen und auf eine Eingabe auf dem Touchscreen gewartet.

Dialog: Der Nutzer bedient die grafische Oberfläche auf dem Touchscreen. Über Sprachausgaben und Steuerung der Blickrichtung wird Kontakt zum Nutzer gehalten. Falls innerhalb einer gewissen Zeit keine Eingabe auf dem Touchscreen erfolgt, wird mit SearchUser ein neuer Nutzer gesucht.

Guide: Der Nutzer hat über die graphische Oberfläche einen Artikel gefunden und sich dafür entschieden, sich zu dessen Standort führen zu lassen. Das Verhalten GuideUser wird aktiviert.

GoHome: Der Roboter bewegt sich allein zu einer vorgegebenen Zielposition und ignoriert dabei alle potentiellen Nutzer (z.B. Fahrt zur Ladestation bei niedrigem Batterie-Ladestand). Aktiviert das Verhalten FollowPath.

Da die Anwendung auf die Erbringung des Service am Kunden ausgerichtet ist, sind die wichtigsten Bedingungen für Zustandsübergänge die Detektion bzw. der Verlust eines Nutzers sowie jeweils die Eingaben des Nutzers zur Auswahl eines bestimmten Dienstes.

Zur Koordination mehrerer Roboter untereinander und zur Anbindung an die Infrastruktur des Marktes (Warenwirtschaft, Fachberater, Zustandsüberwachung) existiert ein Server als Steuerzentrale, mit welchem sich die Roboter jeweils verbinden. Über diesen ist auch eine zentrale Überwachung des Systemzustandes jedes Roboters möglich.

6.3 Bewertung des Gesamtsystems

Wie bereits in Abschnitt 1.2 beschrieben wurde, war in die Entwicklung des Shopping-Lotsen mit der toom BauMarkt GmbH von Anfang an ein Partner involviert, der diese stets unter dem Gesichtspunkt des praktischen Einsatzes betrachtete. Gerade aus der Diskussion von wissenschaftlichen und kommerziellen Partnern entstanden wichtige Impulse und Erkenntnisse. Erst so wurde es möglich, dass beide Seiten jeweils auf ihrem Gebiet die Voraussetzungen für das integrierte System „Shopping-Lotse im Baumarkt“ schufen. Neben dem eigentlichen Robotersystem umfasst dies u.a. eine entsprechende Strukturierung und Präsentation des Warenbestandes sowie die Anbindung an Infrastruktur und Service-Angebote durch die Baumarkt-Betreiber.

Zur Evaluierung der Kundenakzeptanz wurden ab 2006 in verschiedenen Entwicklungsphasen, in Kooperation mit Usability-Forschern des Instituts für Medien- und Kommunikationswissenschaft der TU Ilmenau, Feldtests und Kundenbefragungen in einem Testbaumarkt durchgeführt [Pöschl et al., 2008], [Pöschl et al., 2009].

Es zeigte sich bereits beim Einsatz von Prototypen zu einem frühen Zeitpunkt, dass der entwickelte Shopping-Roboter von einem Großteil der Kunden gut angenommen wurde. Die Kunden hatten während der Testphasen die Möglichkeit, (teils mit und teils ohne vorherige Einweisung) selbstständig das Artikelverzeichnis und die Lotsenfunktion sowie Zusatzangebote wie einen integrierten Barcode-Scanner zu benutzen. Anschließend wurden sie zu ihrer Zufriedenheit, der empfundenen Nützlichkeit und ihren Erwartungen an den Shopping-Roboter befragt.

Abbildung 6.6: *Baumarkt-Kunden bei der Nutzung des Shopping-Lotsen.*

Exemplarisch für die Ergebnisse der Nutzerbefragung sollen hier lediglich einige Zahlen genannt werden: von den im Testzeitraum beobachteten und befragten 240 Kunden nutzten ca. 25% den Roboter. Ein Teil dieser Nutzer wurde unvorbereitet vom Roboter selbst angesprochen, während andere ihn bereits vom Sehen kannten oder teilweise in früheren kurzen Experimentalphasen getestet hatten, und gezielt den Service suchten. Im Interview bescheinigten fast alle Nutzer dem Roboter, sowohl nützlich zu sein (mittlere Bewertung 2.7 auf einer Punkte-Skala von 1 bis 3) als auch Spaß zu machen (mittlere Bewertung 2.8). 80% aller Befragten (Nutzer und Nichtnutzer) äußerten grundsätzliches Interesse an einer (erneuten) zukünftigen Nutzung.

Seit Ende März 2008 befinden sich drei Shopping-Roboter um dauerhaften Testbetrieb im toom Baumarkt Coburg (Bayern). Ziel dieser bis zum Ende des Jahres 2008 angelegten finalen Erprobungsphase, die zwei weitere Märkte einschließen wird, ist der Test der Roboter im Dauereinsatz im Zusammenspiel mit dem normalen Marktbetrieb, sowie die Überprüfung des Nutzungsverhaltens der Kunden [Gross et al., 2008]. Weiterhin befinden sich hier zum ersten Mal mehrere Roboter im gemeinsamen Einsatz und müssen untereinander koordiniert werden (optimale Flächenabdeckung etc.). Jeder Roboter kann täglich für 6 bis 8 Stunden agieren und kehrt bei niedrigem Batterie-Ladestand automa-

tisch zu seiner Ladestation zurück. Zu Spitzenzeiten wurden die Roboter vom Personal deaktiviert, da bei hohem Kundenaufkommen die Bewegungen stark eingeschränkt werden und die durchschnittliche Fahrtzeit zu einem gesuchten Artikel steigt. Zudem werden die Roboter auf Nutzersuche dann von vielen Kunden eher als Hindernis wahrgenommen. Mit diesen Einschränkungen kam jeder der Roboter bis Ende Juli im Durchschnitt auf eine Einsatzzeit von 4.1 Stunden pro Tag und legte dabei insgesamt 210 km zurück.

Insgesamt benutzten 3764 Kunden die Roboter, die im Schnitt ca. 1.5 Artikel suchten. Nach Abzug der von den Kunden explizit abgebrochenen Touren wurden 2799 Zielanfahrten durchgeführt, bei denen in 84% der gesuchte Artikel erreicht wurde. In den verbleibenden 16% wurde entweder der Kontakt zum Nutzer verloren (z.B. weil dieser das Interesse verlor und sich entfernte) und nach einer Wartezeit der Zyklus abgebrochen, oder das Ziel war aufgrund eines versperrten Weges nicht erreichbar. Zum Abschluss des Interaktionszyklus wurden die Kunden vom Roboter selbst um Auskunft gebeten, wie zufrieden sie waren und ob sie sich eine erneute Nutzung bei einem künftigen Besuch des Marktes vorstellen könnten, sie konnten über Buttons auf dem Touch-Display beide Fragen jeweils mit „ja“ oder „nein“ beantworten. 93.4% beantworteten die Frage nach der Zufriedenheit positiv, 92% auch die nach der künftigen Nutzung.

Diese Ergebnisse zeigen, dass der Roboter mit dem erreichten Entwicklungsstand von uneingewiesenen Nutzern im Shopping-Umfeld angenommen und als hilfreich empfunden wird. Einen weiteren wichtigen Aspekt stellt die kommerzielle Bewertung durch den Baumarkt-Betreiber dar. Von dieser Seite besteht natürlich die Erwartung, sich durch das neuartige „Erlebnis“-Angebot des Roboter-unterstützten Einkaufens gegenüber den Wettbewerbern zu profilieren und den Kundenstamm stärker zu binden bzw. neue Kundengruppen anzusprechen. Weiterhin sollen Synergie-Effekte bei Arbeitsabläufen (Warenverwaltung, Angebots-Bewerbung) genutzt werden und das Personal unterstützt werden, indem das einfache, aber zeitraubende und häufig nachgefragte Zeigen eines Artikelstandortes weitgehend automatisiert wird. Qualifizierte Mitarbeiter stehen damit mehr für Beratungsleistungen zur Verfügung, was wiederum die Kundenzufriedenheit erhöht. Wenn diese Erwartungen erfüllt werden können, werden die Shopping-Roboter voraussichtlich ab 2009 in weiteren toom-Baumärkten Einzug halten.

Kapitel 7

Weiterführende Arbeiten

Dieses Kapitel gibt einen Ausblick auf neuere Arbeiten, die sich noch in der Entwicklungsphase befinden und bisher nicht vollständig praktisch getestet wurden, die aber in naher Zukunft zu Lösungen führen sollen, welche in den Shopping-Lotsen sowie in weitere Service-Roboter in ähnlichen Szenarien integriert werden können. Es sollte daher explizit als ein erweiterter Ausblick betrachtet werden, der zwar bereits konkrete, zum großen Teil auch in Simulationen oder Beispielszenarien untersuchte Konzepte vorstellt, jedoch keine finalen Ergebnisse präsentiert. Die vorgestellten Methoden und Resultate sind also als Zwischenschritte auf dem Weg zu einer jeweils abgeschlossenen Lösung anzusehen. Für die Beurteilung des aktuellen Standes des Shopping-Roboters ist dieses Kapitel insofern nicht relevant.

Zunächst wird ein Verfahren zur visuellen Hindernisdetektion vorgestellt, dass direkt auf der monokularen 3D-Szenenrekonstruktion (Anhang C.2) aufbaut. Es wird das Konzept zur Integration in das in Abschnitt 5.2 dargestellte Navigationssystem beschrieben sowie erste Ergebnisse unter Nutzung einer Vorversion gezeigt. In einem weiteren Ansatz wird ein Gesamtsystem zur kontinuierlichen dynamischen Kartenaktualisierung auf Basis der Umgebungsbeobachtungen beschrieben. Dieses integriert mehrere aus der Literatur entnommene Verfahren und legt besonderen Wert auf die Kombination von Stabilität und Plastizität des Umgebungsmodells. Auch hier werden Teilergebnisse präsentiert.

7.1 Visuelle Hindernisdetektion

Motivation

In den Abschnitten 5.2 und 5.4 wurde bereits erklärt, wie im Navigationssystem eine lokale Karte der Umgebung aus den Sensorbeobachtungen erstellt wird und es wurden verschiedene lokale Navigationsverfahren vorgestellt, die diese lokale Karte nutzen, um während der Bewegung den wahrgenommenen Hindernissen auszuweichen. Die lokale Karte integriert die Beobachtungen der Roboter-Sensoren, die der Umgebungserfassung dienen: ein Ring von Sonar-Sensoren, der den kompletten 360°-Bereich um den Roboter abdeckt, sowie ein Laser-Range-Scanner, der das Gebiet vor dem Roboter mit einem Gesamt-Sichtbereich von ca. 240° erfasst. Beide Sensorsysteme zusammen erfassen den gesamten Bereich vor dem Roboter bis zu einer Höhe von ca. 40cm. Wie in Abschnitt 6.3 dargestellt wurde, befindet sich der Roboter derzeit mit dieser sensorischen Ausstattung im Dauertest und erweist sich dort als sehr erfolgreich.

Allerdings existieren auch Einschränkungen, welche Hindernisse wahrgenommen werden können. Der Laser-Scanner misst aufgrund seines Messprinzips mit einem Lichtstrahl nur Entfernungen in einer scharf begrenzten Ebene. Objekte, die sich über oder unter dieser Ebene befinden, werden nicht wahrgenommen. Bei horizontalem Aufbau in einer Höhe von etwa 0.4m bedeutet dies, dass Hindernisse, die flacher als diese Höhe sind, nicht im Sichtbereich des Laser-Scanners liegen. Dies könnte zwar behoben werden, indem der Scanner geneigt eingebaut wird, so dass der Messstrahl schräg auf den Boden fällt und damit Objekte unterschiedlicher Höhe erkennen kann. Allerdings wäre die Messreichweite dadurch dann stark beschränkt (durch die Weite, bei der der Strahl auf den Boden trifft), Hindernisse werden maximal bis zu dieser Entfernung erkannt. Noch gravierender ist, dass die Erkennung von Freiraum stark eingeschränkt wird, da durch die Schrägstellung ein Objekt an einer Beobachtungs-Position erfasst werden kann, beim Näherkommen aber unter dem Messstrahl liegt und damit übersehen wird. Dies führt dazu, dass die Nicht-Beobachtung von Hindernissen nicht mehr als „Beweis“ für Freiraum gelten kann (hier tritt das selbe Problem auf, welches in Abb. 4.26 für eine Kamera dargestellt ist). Ein 3D-Laser-Scanner könnte dieses Problem beheben, allerdings sind ökonomisch sinnvoll einsetzbare

Systeme noch nicht in der Lage, während der Bewegung des Roboters kontinuierlich und schnell genug die komplette Umgebung zu erfassen.

Im Vergleich zum Laser-Scanner weisen die einzelnen Sonar-Sensoren jeweils einen deutlich breiteren Öffnungswinkel (Sichtkegel) von etwa 15° (horizontal und vertikal) auf. Damit erfassen sie einen größeren Höhenbereich, sie sind außerdem unterhalb des Laser-Scanners angebracht, so dass sich die Sichtbereiche beider Systeme ergänzen. Allerdings weisen die Sonarsensoren eine deutlich geringere Messgenauigkeit und (eben aufgrund des Öffnungskegels) geringere räumliche Auflösung auf, zudem sind sie sowohl in der Maximal- als auch Minimalentfernung beschränkt. Objekte in mehr als 2m Entfernung werden nur unzuverlässig erfasst, Objekte in weniger als ca. 0.2m gar nicht (bedingt durch die Kopplung von Ultraschall-Sender und Empfänger und daraus notwendiger Totzeit zwischen Abstrahlung und Empfang des Echos).

Unter diesen Bedingungen ist es nicht möglich, die Erkennung aller potentiell auftretenden Hindernisse zu gewährleisten. Insbesondere sind solche Hindernisse nicht wahrnehmbar, die sich oberhalb der Sensoren befinden, aber nicht bis zum Boden reichen. Ein Beispiel dafür (welches allerdings im Baumarkt-Szenario weniger relevant ist) ist ein Tisch, bei diesem werden zwar die Tischbeine erkannt, die Tischplatte aber bleibt für die Abstandssensoren unsichtbar, da sie darunter hindurch messen (Abb. 7.1). Im Baumarkt betrifft dieses Problem zum Beispiel Bretter oder Latten, die aus einem Regal oder Einkaufswagen herausragen können.

Generell wird eine hundertprozentige Erkennung aller Hindernisse nicht zu garantieren sein, da zum Beispiel immer bestimmte nicht oder wenig reflektierende Oberflächen auftreten können. Zudem kann auch eine sich bewegende Person oder zum Beispiel ein Einkaufswagen in der Bewegung mit dem stehenden Roboter zusammenstoßen, dieser ist nicht dafür ausgelegt, aktiv auszuweichen. In einer nicht kontrollierbaren Umgebung muss daher (neben der Minimierung des Kollisionsrisikos) Sorge getragen werden, dass bei einer Kollision weder der Roboter noch Objekte der Umgebung oder gar Personen Schaden nehmen. Dazu ist der Roboter u.a. mit Kontaktsensoren (englisch: Bumper) und einer Motor-Abschaltung bei zu hohen Gegenkräften ausgerüstet.

Dennoch muss natürlich das Ziel sein, eine möglichst hohe Erkennungsrate für Hindernisse zu gewährleisten, um die Wahrscheinlichkeit einer Kollision zu minimieren. Aufgrund des in Relation zu den anderen Sen-

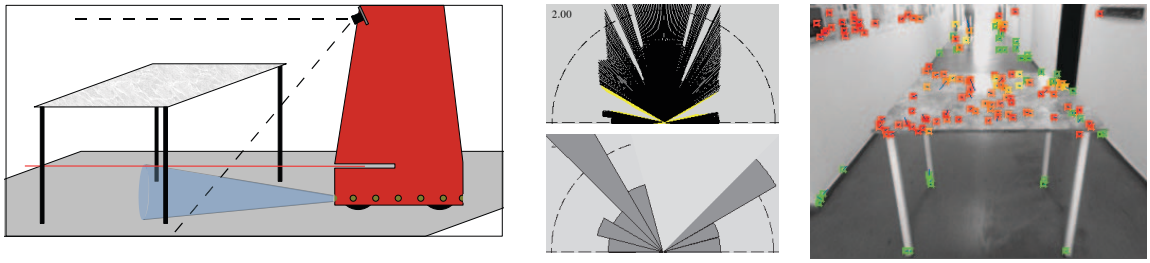

Abbildung 7.1: *Die Abbildungen verdeutlichen an einer Beispielsituation die Einschränkungen der Hinderniswahrnehmung mit Sonar- und Laser-Entfernungssensoren. Da diese Sensoren jeweils einen begrenzten Höhenbereich erfassen, können sie manche Hindernisse nicht detektieren. Der Laser-Scanner (Mitte oben) erkennt in dieser Situation nur die Tischbeine, der Raum dazwischen wird als frei angenommen. Für die Sonarsensoren (Mitte unten) ist aufgrund der geringen Breite der Beine und der damit verbundenen schwachen Reflektionen die genaue Wahrnehmung noch schwieriger: Die gelben Kegel in frontaler Richtung zeigen an, dass hier die Sensoren kein Echo wahrnehmen und damit keine Entfernung messen können. Eine Kamera hat einen wesentlich größeren Sichtbereich, sie kann die Tischplatte sehen (rechts) und damit erkennen, dass der Roboter nicht zwischen den Tischbeinen hindurch fahren darf. Dass die Tischbeine selbst im Kamerabild nicht detektiert werden, ist hier nicht kritisch, da einerseits bei der Projektion auf eine zweidimensionale Karte die Tischfläche die gesamte Hindernis-Ausdehnung anzeigt, darüber hinaus in der Kombination der verschiedenen Sensoren die Beine robust vom Laser-Scanner detektiert werden.*

soren großen Erfassungsbereichs bei gleichzeitig hoher Messgeschwindigkeit scheint visuelle Sensorik sehr gut geeignet zur Detektion von Hindernissen in Kombination mit Laser und Sonar. Allerdings ist hierbei der Aufwand zur Extraktion der relevanten Information aus den reinen Sensordaten beträchtlich höher.

Hindernisdetektion durch visuelle Tiefenschätzung

Bereits in Abschnitt 4.4 wurden visuelle Verfahren zur Umgebungserfassung und darauf aufbauende Methoden zur Erstellung von Gridkarten benutzt. Während das Ziel dort die Gewinnung konsistenter globaler Karten mittels SLAM war, liegt nun der Fokus auf dem kontinuierlichen Update einer lokalen Karte, um eine schnelle und robuste Erkennung von Hindernissen zu ermöglichen. Aufgrund des geringeren Hardware- und Kalibrierungsaufwandes wird ein monokulares Verfahren hier be-

vorzugt. Dem steht der höhere methodische Aufwand zur Extraktion der Tiefeninformation aus dem Kamerabild im Vergleich zu einer Stereo-Anordnung gegenüber.

Die Berechnung und Aktualisierung einer lokalen Karte aus der Sequenz der Kamerabilder besteht im Prinzip lediglich aus einer Kombination des in Abschnitt C.2 beschriebenen Verfahrens zur monokularen Tiefenschätzung mit der Umrechnung der detektierten 3D-Punkte in eine Gridkarte aus Abschnitt 3.1.3. Die Nutzung der resultierenden Karte zur Kollisionsvermeidung stellt allerdings besonders hohe Anforderungen: sichtbare Hindernisse müssen mit hoher Sicherheit und vor allem möglichst frühzeitig erkannt werden, um ein Ausweichverhalten realisieren zu können. Gleichzeitig muss die False-Positive-Rate (fälschliche Erkennung von Hindernissen an eigentlich freien Positionen) auf ein Minimum beschränkt werden, da solche Fehldetektionen den Roboter unnötig zum Anhalten oder Ausweichen veranlassen und damit eine gewünschte stetige Fahrt mit hoher Geschwindigkeit empfindlich stören.

Zwar weist das beschriebene Verfahren zur Tiefenschätzung durch die Kombination einer guten Initialisierung mittels Multi-Baseline-Stereo-Verfahren und nachfolgender rekursiver Beobachtungsupdates im Kalmanfilter bereits eine gegenüber den Einzelverfahren verbesserte Konvergenzgeschwindigkeit auf. Allerdings ist es immer noch stark abhängig von der Odometriemessung. Insbesondere die gute zeitliche Synchronisation von Odometrie und Kamerabildern ist hier kritisch, diese kann aber nur mit hohem technischen Aufwand garantiert werden. Besser ist es, die Schätzung der Eigenbewegung ebenfalls aus der Bildsequenz zu schätzen. Bei ausschließlicher Nutzung von Punktfeatures sind der erreichbaren Genauigkeit allerdings Grenzen gesetzt. Das aktuelle Verfahren zeigt daher insbesondere bei Rotation des Roboters noch Schwierigkeiten, die Kamerabewegung und damit auch die 3D-Positionen der Umgebungspunkte robust zu schätzen.

Ergebnisse

Die Abbildungen 7.2 und 7.3 zeigen Ergebnisse der visuellen Hindernisdetektion für eine typische Situation im Baumarkt. Während der Roboter sich auf die Hindernisse zu bewegt, detektiert das monokulare Szenenrekonstruktionsverfahren eine große Anzahl von Punkten sowohl auf diesen Objekten als auch auf dem Untergrund. Diese wer-

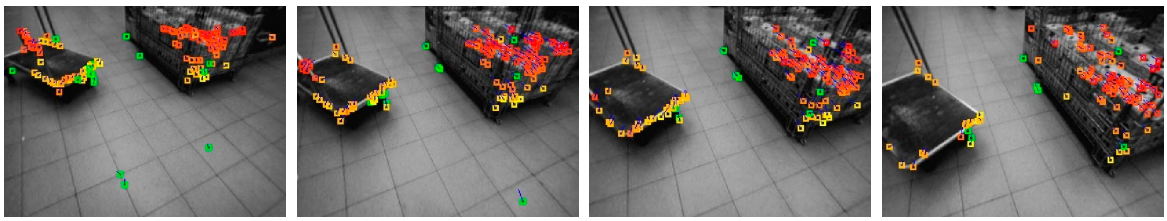

Abbildung 7.2: *Die Bilder zeigen Ausschnitte aus einer Bildsequenz während der Bewegung des Roboters, zwei Hindernisse sind sichtbar. Die Ladefläche des flachen Transportwagens links im Bild ist nicht vom Laser-Scanner zu erkennen, da sie sich unterhalb von dessen Messebene befindet. Aufgrund des geringen Querschnitts und dadurch geringer Reflektion ist auch für die Sonarsensoren eine Beobachtung nur auf kurze Distanz möglich. Hier zeigt sich der Nutzen der visuellen Hindernisdetektion. Die Farben der im Bild detektierten Punkte beschreiben die geschätzte Höhe (grün (niedrig) - gelb - orange - rot (hoch)). Beide Hindernisse werden sehr gut erkannt. Abb. 7.3 zeigt eine lokale Karte, in die die geschätzten Punkte eingetragen sind.*

den in eine Karte eingetragen, indem zunächst ein Vergleich mit einer Höhenschwelle erfolgt. Die Schwelle muss größer als die zu erwartende Varianz der Höhenschätzung für die einzelnen Punkte sein, da sonst solche Featurepunkte, die eigentlich auf dem Boden liegen, aufgrund von Ungenauigkeiten der Schätzung als Hindernis eingetragen werden können. Diese würde also zu False-Positive-Detektionen von Hindernissen führen. Die Schwelle wurde im Beispiel mit 15cm gewählt. Es wurde hier außerdem vorausgesetzt, dass keine Hindernisse negativer Höhe gegenüber dem Boden wie Löcher, Treppen nach unten etc. auftreten. Bei negativen Höhenschätzungen wird daher immer implizit angenommen, dass diese Punkte eigentlich auf der Bodenebene liegen und somit keine Hindernisse darstellen.

Die Verbesserung des existierenden visuellen Szenenrekonstruktions-Verfahrens und damit die Fähigkeit zur robusten visuellen Hindernisdetektion ist ein Kernziel eines aktuellen Projektes am Fachgebiet NIKR. Sowohl die Hindernisdetektion als auch vor allem die robuste Schätzung der Kamerabewegung soll durch die zusätzliche Nutzung weiterer Features, insbesondere Kanten, verbessert werden. Weiterhin soll eine genauere und robustere Zuordnung der Punkte über aufeinander folgende Bilder durch Verbesserungen der Feature-Detektoren und der Matching-Kriterien ermöglicht werden, dies führt im Endergebnis wiederum zu einer höheren Genauigkeit der geschätzten 3D-Punkte.

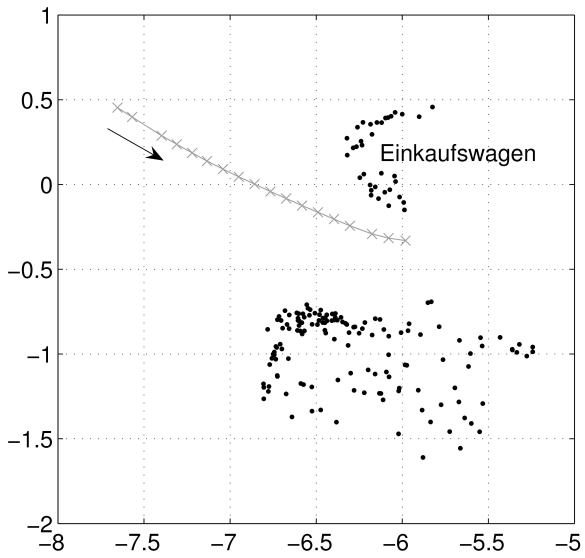

Abbildung 7.3: *In die lokale 2D-Karte wurden diejenigen detektierten Featurepunkte aus Abb. 7.2 eingetragen, deren geschätzte Höhe über dem Boden mehr als 15cm beträgt. Die grauen Kreuze im linken Bereich beschreiben die Positionen des Roboters, jeweils im Moment einer Bildaufnahme. Beide Hindernisse sind gut abgebildet, allerdings treten noch Falsch-Positiv-Detektionen von Hindernissen auf. Dadurch sind die Umrisse der Objekte nicht exakt reproduziert, die lokale Navigation würde hier irrtümlich die befahrbare Breite als zu schmal für eine Durchfahrt einschätzen.*

7.2 Dynamische Kartenadaption

Motivation und Überblick

In Kapitel 4 wurde ein probabilistisches SLAM-Verfahren zur Umgebungskartierung entwickelt, das als Ergebnis ein statisches Modell der Umgebung zur Zeit der Kartierung generiert. Dieses Modell wird im späteren Einsatz vom Roboter als Basis der autonomen Navigation benutzt (Kapitel 5), der Roboter hat also zunächst nur Kenntnis vom Zustand der Umgebung während der Kartierung.

Am Ende von Abschnitt 5.4.1 wurde bereits eine recht einfache Methode beschrieben, die eigentlich statische globale Karte an Änderungen der Umgebung anzupassen: Wenn ein Hindernis während einer Zielanfahrt den geplanten Pfad so blockiert, dass es durch das lokale Kollisionsvermeidungs-Verfahren nicht passiert werden kann, so wird die Karte an dieser Stelle entsprechend der aktuellen Wahrnehmung (lokale Karte) angepasst. Diese Änderung ist allerdings nur temporär und wird recht schnell wieder vergessen, d.h. die globale Karte fällt immer wieder auf den initialen statischen Zustand zurück. Für die Planung und Bewegungssteuerung ist diese Form der Kartenadaption zunächst ausreichend: Die Karte wird nur dann angepasst, wenn die Änderung in der Umgebung tatsächlich gravierenden Einfluss auf die Bewegung des Roboters hat (eine Durchfahrt ist blockiert). Kleinere Veränderungen, die durch die lokale Navigation ausgeglichen werden können und den

Verlauf des Pfades nicht signifikant beeinflussen, haben bei dieser Vorgehensweise hingegen keine Auswirkung auf die Karte.

Insbesondere für die Selbstlokalisation (Abschnitt 5.1) haben jedoch auch solche kleinen Änderungen unter Umständen große Bedeutung, selbst wenn sie sich noch nicht hinderlich auf die Bewegungsmöglichkeiten des Roboters auswirken. Wenn die vom Roboter mittels seiner Sensorik wahrgenommene Beobachtung der Umgebung an einer bestimmten Position zu stark von der Erwartung abweicht, kann dies möglicherweise dazu führen, dass die eigene Position fehlerhaft geschätzt wird. Dabei wird zunächst zum Beispiel der mittlere Fehler der Positionsschätzung zwar anwachsen, jedoch beschränkt bleiben. Solange die Änderungen gering und räumlich begrenzt sind, ist das in dieser Arbeit beschriebene Selbstlokalisations-Verfahren normalerweise robust genug, um die korrekte Position zu erkennen, sobald die Umgebung wieder dem bekannten Zustand entspricht, zum Beispiel weil der Bereich der Änderungen verlassen wird (Abb. 7.4).

In einer belebten Umgebung wie dem Baumarkt-Szenario muss jedoch damit gerechnet werden, dass ständig kleine Veränderungen vorgenommen werden und diese sich im Laufe der Zeit kontinuierlich akkumulieren, so dass die initial erstellte Karte immer weniger der realen Umgebung entspricht. Dies führt dazu, dass mit fortschreitender Einsatzzeit die Lokalisationsgenauigkeit abnimmt und ab einem gewissen Punkt die Wahrscheinlichkeit für katastrophale Lokalisationsfehler (die nicht mehr selbständig korrigiert werden können) untolerierbar groß wird. Ein System, welches sich hier im Dauereinsatz befindet, muss also entweder in regelmäßigen Abständen eine neue Karte generieren bzw. zur Verfügung gestellt bekommen, oder es muss kontinuierlich die wahrgenommenen Umgebungsveränderungen in die Karte integrieren.

Typischerweise ist keine scharfe Kenntnis über die aktuelle Roboterposition vorhanden, sondern es wird bei der Selbstlokalisation mittels Partikelfilter (Abschnitt 5.1) eine Wahrscheinlichkeitsverteilung für die Position geschätzt. Obwohl diese Verteilung bei Nutzung einer zuvor erstellten Karte und bekannter Startpose fast immer monomodal mit relativ geringer Varianz verteilt ist, bleibt doch eine Unsicherheit bestehen, die oft in der Größenordnung einer oder mehrerer Zellen der globalen Karte liegt. Eine naive Strategie zur Kartenadaption, die einfach ständig mit den aktuellen Beobachtungen und der geschätzten wahrscheinlichsten Position die Gridzellen aktualisiert, führt daher re-

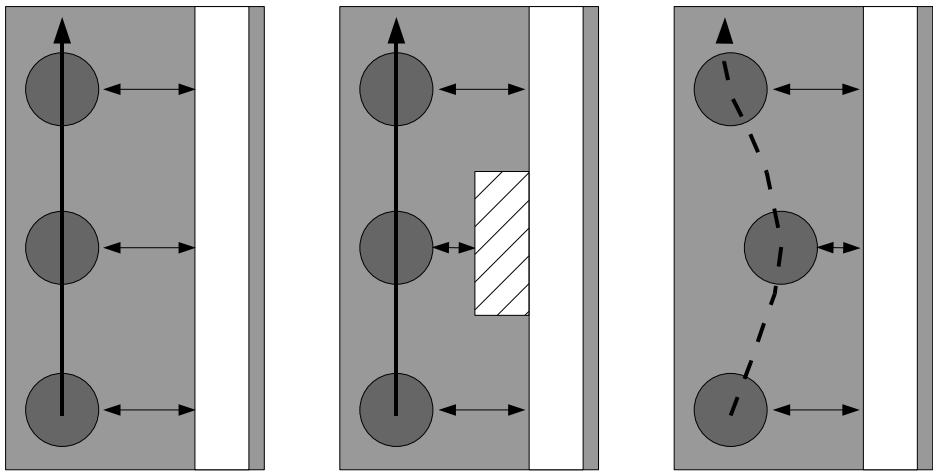

Abbildung 7.4: *Ein vereinfachtes Beispiel für die Auswirkung einer geringen Umgebungsveränderung auf die Lokalisationsgenauigkeit: Der Roboter bewegt sich parallel zu einer Wand, der gemessene Abstand zur Wand bleibt dabei konstant (links). Durch ein zusätzliches Objekt verändert sich die Umgebung gegenüber der Erwartung (Mitte, schraffiertes Objekt). Dadurch wird bei gleicher Fahrtroute eine andere Entfernung auf der rechten Seite beobachtet. Wenn die Selbstlokalisation die Möglichkeit einer solchen Veränderung nicht berücksichtigt, so kann die Beobachtung nur erklärt werden, indem eine Position näher zur Wand angenommen wird (rechts). Die Veränderung führt also zu einem Lokalisationsfehler. Dieser ist bei einer solch kleinen Umgebungsveränderung allerdings gering und hat kaum Auswirkungen auf die Roboterfunktion.*

lativ schnell zu Verfälschungen, da Änderungen nicht nur aufgrund tatsächlicher Veränderungen der Umgebung, sondern auch durch Abweichungen der geschätzten von der realen Position entstehen. Da die Karte anschließend wieder zur Selbstlokalisation benutzt wird, besteht dadurch auch die Gefahr, dass dauerhaft an einer veränderten Stelle die Position falsch geschätzt wird und der Fehler somit konsolidiert oder gar immer weiter verstärkt wird.

Weiterhin sind in der Praxis viele Änderungen nicht von Dauer, sondern treten nur kurzfristig auf, zum Beispiel weil Objekte (im Baumarkt: Warenpaletten, Einkaufswagen, etc.) sich nur für einen begrenzten Zeitraum an einem Ort befinden. Sobald das Objekt wieder entfernt wird, wird der alte Zustand wieder hergestellt. Gleichzeitig ist aber eine Vorhersage darüber, wie lange eine Veränderung bestehen bleiben wird, fast nie möglich, d.h. es nicht entscheidbar, welche Beobachtungen die Karte verändern sollten und welche nicht. Ein Verfahren, das jeweils nur den zuletzt beobachteten Zustand abbildet, wird in solchen Situa-

tionen ständig die Karte anpassen, dabei aber bei jeder Änderung eine Abweichung von der Erwartung feststellen, da der vorhergehende Zustand bereits vergessen wurde. Dies führt, wie beschrieben, jeweils zu einem erhöhten Lokalisationsfehler. Es ist anzumerken, dass auch der dauerhafte Einsatz des zuvor in dieser Arbeit beschriebenen SLAM-Verfahrens nicht zur Lösung dieses Problems geeignet ist, da dieses eine statische Umwelt annimmt und auch jeweils nur den zuletzt beobachteten Zustand modelliert.

Für eine robuste Selbstlokalisation ist es also vorteilhaft, bei der Erkennung einer Umgebungs-Veränderung zwar den neuen Zustand zu modellieren, gleichzeitig aber den vorherigen Zustand nicht komplett zu vergessen. Es muss somit eine Form der Modellierung gefunden werden, die sowohl die Stabilität der einmal gelernten Karte, aber auch die Plastizität gegenüber Veränderungen gewährleisten kann.

Aus den genannten Gründen wurde u.a. im Rahmen einer Diplomarbeit [Merten, 2006] ein Verfahren zur kontinuierlichen Kartenadaption entwickelt. Die wesentlichen Ziele bei der Entwicklung sind im folgenden formuliert:

- Die Umgebungsbeobachtungen durch die Sensoren des Roboters (hier: Laser- und Sonar-Entfernungsmessungen) sollen neben der Selbstlokalisation auch zur kontinuierlichen Adaption der Karte benutzt werden.

- Um fehlerhafte Veränderungen der Karte aufgrund falscher oder unklarer Positionsschätzung zu beschränken, soll die Zuverlässigkeit der Lokalisation zu jedem Zeitpunkt abgeschätzt werden und dieses Gütemaß genutzt werden, um die Stärke der Kartenveränderung zu steuern.

- Die Anpassung an den neuen Zustand soll schnell geschehen, d.h. es sollen möglichst wenige Beobachtungen notwendig sein, bis die Karte den veränderten Zustand repräsentiert.

- Gleichzeitig soll trotz des Lernens des neuen Zustandes das Wissen um den alten Zustand erhalten bleiben, so dass ebenfalls darauf zurückgegriffen werden kann.

- Das gesamte Verfahren soll effizient implementiert sein und wenig Aufwand benötigen, so dass es neben den anderen Modulen der normalen autonomen Funktion laufen kann.

Um diese Ziele zu erreichen, wurden verschiedene der Literatur entnommene Verfahren [Milstein, 2005], [Biber und Duckett, 2005], [Stachniss und Burgard, 2005], [Gutmann und Fox, 2002] mit eigenen Ideen kombiniert.

Einen Ansatz aus [Biber und Duckett, 2005] aufgreifend, wird die Umgebung gleichzeitig durch mehrere Karten dargestellt, welche sich mit unterschiedlicher Geschwindigkeit an die aktuellen Beobachtungen anpassen und damit wiederum unterschiedliche Zeithorizonte erfassen. Dies ermöglicht gleichzeitig die sofortige Integration neuer Beobachtungen in das Umgebungsmodell als auch die Bewahrung des Wissens über den vorherigen Zustand. In der hier genutzten Implementierung wurden allerdings zunächst lediglich zwei Karten mit unterschiedlichen Zeitkonstanten verwendet (eine Erweiterung auf mehr Zeitebenen wäre aber leicht möglich). Weiterhin werden für die beiden Karten unterschiedliche Modellierungstechniken eingesetzt, um gleichzeitig die verschiedenen Vorteile der einzelnen Modelle nutzen zu können. Neben einer Gridkarte, deren Anpassungsgeschwindigkeit durch die Modellierung von Zellkorrelationen noch gesteigert wird, wird als Gegenstück eine referenz-basierte Karte verwendet, die eine hohe Stabilität eines einmal gelernten Zustandes aufweist. Abb. 7.5 stellt das Gesamtsystem dar. Die einzelnen Komponenten werden nachfolgend in eigenen Abschnitten detailliert beschrieben.

Güteschätzung der Selbstlokalisation

[Gutmann und Fox, 2002] beschreibt mit der *Adaptive MCL* eine Methode, um die aktuelle Güte der Positionsschätzung g_t anhand der mittleren Partikelbewertung $p(o_t|x_t^{(i)})$ (über alle Partikel i im aktuellen Schritt t) zu bestimmen. Dort wird dieses Gütemaß benutzt, um bei unsicherer Positionsschätzung zusätzliche Partikel global in der Umgebung einzustreuen und damit eine globale Relokalisation zu ermöglichen. Bei der hier vorgestellten Kartenadaption wird dieses Maß hingegen benutzt, um die „Lernrate“ für die Karte dynamisch anzupassen. Es werden dazu zwei zeitliche Mittelwerte über die zurückliegenden Partikel-

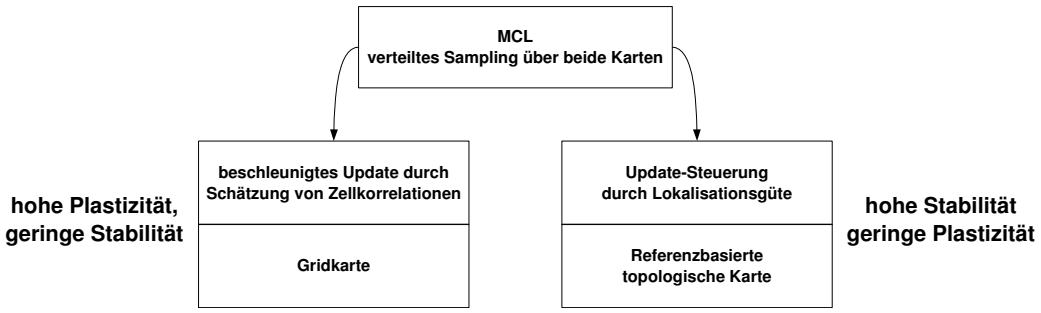

Abbildung 7.5: *Für die dynamische Kartenadaption werden zwei unterschiedliche Kartentypen genutzt, die verschiedene Adaptionsgeschwindigkeiten aufweisen. Die Anpassungsgeschwindigkeit der Gridkarte wird zusätzlich durch die Schätzung von Zellkorrelationen beschleunigt. Die Adaption der referenzbasierten Karte wird hingegen gebremst, wenn die Lokalisationsgüte sinkt. Für die Lokalisation werden die Partikel vom Partikelfilter anteilig in beiden Karten eingestreut, dadurch wird automatisch das besser zur aktuellen Beobachtung passende Modell ausgewählt.*

Bewertungen berechnet, die sich unterschiedlich schnell an die aktuellen Werte anpassen: $\bar{p}_l$ und $\bar{p}_s$ (Abb. 7.6). Die Berechnung erfolgt in beiden Fällen rekursiv:

$$\bar{p}_{l,t} = (1-\gamma_l)\cdot\bar{p}_{l,t-1} + \gamma_l \cdot \frac{1}{n}\cdot\sum_{i=1}^{n} p(o_t|x_t^{(i)}) \tag{7.1}$$

$$\bar{p}_{s,t} = (1-\gamma_s)\cdot\bar{p}_{s,t-1} + \gamma_s \cdot \frac{1}{n}\cdot\sum_{i=1}^{n} p(o_t|x_t^{(i)}) \tag{7.2}$$

Der Abwertungsfaktor γ bestimmt dabei den Einfluss zurückliegender Beobachtungen und damit die Länge des Zeitfensters (l = long/lang, s = short/kurz) für den gleitenden Mittelwert bzw. die Geschwindigkeit der Anpassung an die aktuelle Bewertung. Es gilt

$$0 < \gamma_l \ll \gamma_s < 1 \tag{7.3}$$

$\bar{p}_l$ folgt also der aktuellen Bewertung sehr langsam bzw. mittelt über einen langen Zeithorizont. Dieser Wert bildet damit einen Erwartungswert für die mittlere Partikel-Bewertung in der aktuellen Umgebung. $\bar{p}_s$ stellt hingegen die aktuelle Bewertung dar. Zweck der Mittelung ist bei diesem Wert lediglich die Elimination extremer Ausreißer. Das

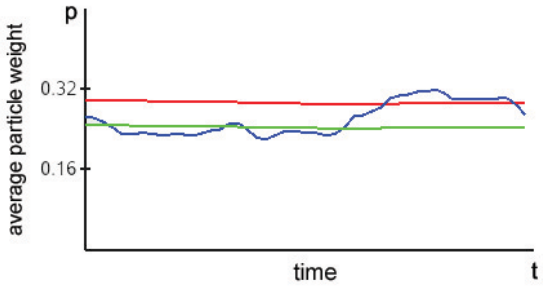

Abbildung 7.6: *Beispielhafter zeitlicher Verlauf der über zurückliegende Schritte und über alle Partikel gemittelten Bewertungen $\bar{p}_l$ (rot) und $\bar{p}_s$ (blau). Ein hohes Verhältnis $\bar{p}_s : \bar{p}_l$ bedeutet, dass aktuell viele Partikel sehr gut bewertet werden, dies lässt auf eine hohe Verlässlichkeit der Positionsschätzung schließen. Ein niedriges Verhältnis bedeutet hingegen eher eine geringere Qualität der Schätzung. Der grün gezeichnete Wert bezeichnet eine Schwelle (fixiert relativ zu $\bar{p}_l$), ab der die Adaptionsgeschwindigkeit der Karte verringert wird.*

Verhältnis der beiden Werte stellt ein Maß für die relative mittlere Partikelbewertung und damit für die Wahrscheinlichkeit einer guten Positionsschätzung dar.

$$g_t = \frac{\bar{p}_{s,t}}{\bar{p}_{l,t}} \tag{7.4}$$

Referenzbasierte Karte

Für eine der beiden dynamische Karten wird eine samplebasierte Beobachtungsrepräsentation nach [Biber und Duckett, 2005] genutzt. Diese samplebasierte Repräsentation basiert auf der Speicherung einer Menge von Referenzbeobachtungen an Referenzpositionen in der Umgebung. Neue Referenzpositionen werden jeweils dann eingefügt, wenn die aktuelle Roboterposition eine Abstandsschwelle (zum Beispiel 2m) zu allen existierenden Referenzpositionen überschreitet (Abb. 7.7). Diese wird beim Erzeugen mit der aktuellen Beobachtung (den aktuellen Entfernungsmessungen in einer Anzahl definierter Richtungen) initialisiert. Beim erneuten Besuch einer existierenden Referenzposition wird die aktuelle Beobachtung zusätzlich eingefügt. Jede Position speichert eine konstante Anzahl an Beobachtungen. Sobald diese erreicht ist, werden zufällig ausgewählte ältere Beobachtungen durch neue verdrängt.

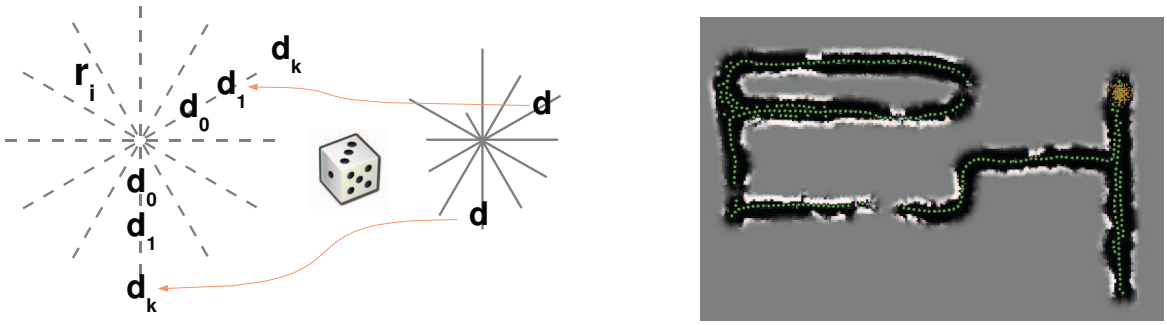

Abbildung 7.7: **Links:** *Die Referenz- bzw. Sample-basierte Karte modelliert die Umgebung durch eine Menge von Referenzpunkten. Für jeden Referenzpunkt r_i werden die beobachteten Entfernungen in einer Menge von Richtungen φ_j beschrieben. Diese gespeicherten Referenzentfernungen sind Samples (Beispiele) für alle realen Beobachtungen an der entsprechenden Position. Für jede Richtung jedes Referenzpunktes wird dabei eine feste Anzahl k Distanzen gespeichert. Die Distanzen für verschiedene Richtungen φ_j sind jeweils unabhängig, d.h. aus einer realen Beobachtung werden jeweils nur einzelne Richtungen in die Referenz übernommen. Für jede Beobachtung, die einem Referenzpunkt R_i zugeordnet wird, wird dazu eine zufällige Menge an Richtungen ausgewählt (deren Anzahl durch die gewünschte Adaptionsgeschwindigkeit sowie die geschätzte Güte der Lokalisation bestimmt wird). Die Distanzen in den gewählten Richtungen ersetzen wiederum jeweils einen zufälligen der k gespeicherten Distanz-Werte. Die erwartete Beobachtung jedes Referenzpunktes wird durch den Median der gespeicherten Distanzen je Richtung bestimmt, dadurch wird zusätzlich eine gewisse Stabilität gegen Fehlmessungen erzielt.* **Rechts:** *Beispiel für die Verteilung der Referenzpunkte in der Umgebung. Die hinterlegte Gridkarte wurde aus den Median-Beobachtungen aller Referenzpunkte erzeugt und verdeutlicht damit die repräsentierte Umgebung.*

Die Einfügewahrscheinlichkeit wird zusätzlich durch die zuvor beschriebene Güteschätzung der Lokalisation moduliert, d.h. je höher die Zuverlässigkeit der Positionsschätzung, umso größer ist jeweils die Wahrscheinlichkeit, dass eine neue Beobachtung eingefügt wird und umso schneller wird die Umgebungsrepräsentation an neue Beobachtungen angepasst.

Für die Anwendung zur Selbstlokalisation wird aus allen Referenzbeobachtungen an allen Referenzpositionen wieder eine Karte erzeugt. Dazu wird je einzelner Beobachtungs-Richtung an jeder Position der Median der dafür gespeicherten Distanzen gebildet. Im Gegensatz zum Mittelwert nimmt der Median immer einen Wert aus der zugrunde liegenden Menge von Elementen an, so dass bei einer multimodalen Verteilung der beobachteten Entfernungen (wie sie typischerweise durch hinzu-

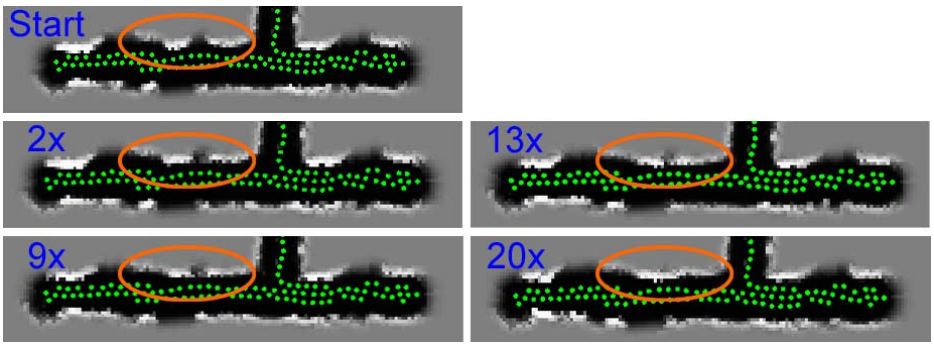

Abbildung 7.8: *Die referenzbasierte Karte ist stärker auf Stabilität als auf Plastizität ausgelegt: Im markierten Bereich ist in der Karte eine freie Durchfahrt vermerkt. Diese Durchfahrt wurde durch ein Hindernis versperrt. Trotz wiederholter Beobachtung behält die Karte zunächst den geschätzten Zustand bei und wechselt nur sehr langsam. Erst nachdem der Roboter 20 mal die betreffende Stelle passiert und das Hindernis beobachtet hat, wird dies in der Karte vermerkt. Die referenzbasierte Karte adaptiert sich damit viel langsamer als die Gridkarte (vgl. Abb. 7.9).*

kommende oder verschwindende Objekte entstehen) keine unplausiblen Zwischenwerte erzeugt werden. Aus den so reproduzierten „Median-Beobachtungen“ wird wiederum eine Gridkarte erstellt, die analog zu Abschnitt 5.1 zur Partikelbewertung in der MCL benutzt werden kann (Abb. 7.7 rechts). Da beim Median im Extremfall bis zu 50% der Werte als Ausreißer betrachtet werden und damit keinen Einfluss auf das Ergebnis haben, ist die Adaptionsgeschwindigkeit dieser referenzbasierten Karte recht gering. Ein neuer Zustand der Umgebung (beispielsweise ein Objekt, welches zuvor nicht vorhanden war) muss relativ oft beobachtet werden, bevor an den entsprechenden Referenzpositionen so viele neue Beobachtungssamples eingefügt wurden, dass der Median verändert wird (Abb. 7.8).

Gridkarte mit Zellkorrelationen

Neben der referenzbasierten Karte wird eine Gridkarte als Basis der Selbstlokalisation benutzt. Diese wird ebenfalls mit jeder neuen Beobachtung aktualisiert. In Kapitel 3 wurde dargelegt, dass dem dort beschriebenen Update-Algorithmus für Gridkarten eigentlich die Annahme der Unabhängigkeit der einzelnen Zellen zugrunde liegt. Dies stellt allerdings eine Vereinfachung dar. Ein einfaches Beispiel, wo diese offensichtlich verletzt wird, ist eine Tür, welche offen oder geschlossen sein

kann: Im jeweiligen Zustand werden einige Zellen belegt sein (dort wo sich die Tür gerade befindet) und andere frei. Die Tür kann jedoch nicht an zwei Orten gleichzeitig sein, d.h. bestimmte Zellbelegungen schließen einander aus. Gleichzeitig belegt die Tür normalerweise in jedem Zustand einige Zellen. Ist eine davon belegt/frei, so steigt die Wahrscheinlichkeit, dass auch die weiteren diesem Zustand entsprechenden Zellen belegt/frei sind. Durch Kenntnis solcher Korrelationen kann eine schnellere Adaption der Karte erfolgen, da die Beobachtung einer Zelle Informationen auch über aktuell aufgrund von Verdeckungen oder zu großer Entfernung nicht direkt beobachtete Zellen liefern kann.

Die Abhängigkeiten zwischen den Zellen werden parallel zu den Kartenupdates gelernt, indem Korrelationen zwischen jeweils zwei direkt benachbarten Zellen geschätzt werden. Dieser Ansatz geht auf [Milstein, 2005] zurück. Prinzipiell könnten Korrelationen nicht nur zwischen benachbarten, sondern zwischen beliebigen Zellen bestimmt werden, allerdings sind einerseits signifikante Beziehungen am ehesten zwischen räumlich nahe beieinander gelegenen Zellen zu erwarten, andererseits muss die Menge der einbezogenen Zellpaare aus Platz- und Zeitgründen begrenzt werden.

In [Merten, 2006] wurde ein einfacher Ansatz benutzt, der darauf beruht, beim Update der Gridzellen aus einer neuen Beobachtung jeweils das Verhältnis der Änderungen benachbarter Zellen zu betrachten (occ_t^i wird hier verwendet als Kurzform für $P(occ^i|o_{1:t})$ - die geschätzte Wahrscheinlichkeit der Belegtheit von Zelle i nach der Beobachtung zum Zeitpunkt t):

$$K^{ij} = \frac{\Delta occ^i}{\Delta occ^j} = \frac{occ_t^i - occ_{t-1}^i}{occ_t^j - occ_{t-1}^j} \tag{7.5}$$

Ein positives Verhältnis deutet auf positive Korrelation (gleichartige Änderung), ein negatives Verhältnis hingegen auf negative Korrelation (entgegengesetzte Änderung). Diese gegenseitigen Verhältnisse K^{ij} der Änderung können jeweils für ein Zellpaar mittels eines gleitenden Mittelwertes über alle gemeinsamen Beobachtungen der Zellen gelernt werden. Später wird beim Update einer Zelle wiederum direkt die Änderung der Nachbarzelle abgeschätzt:

$$occ_t^j = occ_{t-1}^j + K^{ij} \cdot \left(occ_t^i - occ_{t-1}^i\right) \tag{7.6}$$

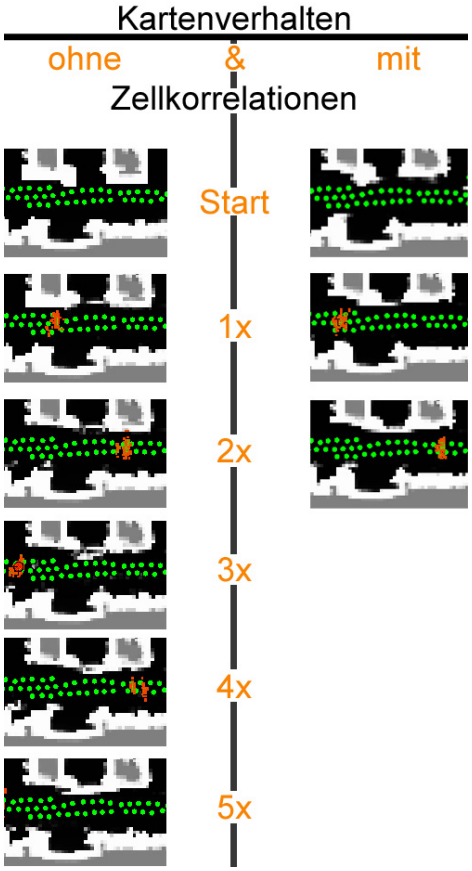

Abbildung 7.9: *Die Abbildung zeigt den Effekt der Nutzung von Zellkorrelationen zur Beschleunigung der Kartenanpassung an Umgebungsveränderungen: Der Durchgang (im Bild oben) wurde zuvor mehrfach abwechselnd als frei und durch ein Hindernis versperrt beobachtet. Daraus wurden entsprechende Korrelationen für die Zellen an dieser Stelle beobachtet, die sich gemeinsam ändern (Hindernis vorhanden/nicht vorhanden). Im Experiment wird zunächst angenommen, dass der Durchgang frei ist. Tatsächlich ist er jedoch aktuell durch das Hindernis versperrt. Der Roboter fährt fünf mal an dem Hindernis vorbei, bevor die Karte den neuen Zustand vollständig abbildet (links). Bei Nutzung der Korrelationsinformation für zusätzliche Updates neben den Sensorbeobachtungen passt sich die Karte deutlich schneller an, da die Beobachtung jeder Zelle direkt deren Nachbarzellen in Richtung des neuen Zustandes verändert. Bereits nach der zweiten Vorbeifahrt zeigt die Karte den neuen Zustand (rechts).*

Dieser Ansatz bietet zwar auf den ersten Blick eine intuitive Möglichkeit zur Schätzung und Nutzung der Zellkorrelationen, ist aber aus verschiedenen Gründen wenig plausibel: Es erfolgt keine theoretisch fundierte Berechnung der tatsächlichen Korrelation zwischen den Zellen. Die Schätzung des Verhältnisses der Änderung von Zellen kann theoretisch dazu führen, dass eine Beobachtung einer Zelle eine sehr starke Veränderung einer nicht beobachteten Zelle hat. Um Instabilitäten zu vermeiden, müssen künstlich Schwellwerte für maximale Änderungen bestimmt werden. Außerdem erfolgt ein Lernen von Zusammenhängen zwischen Zellen ausschließlich im Moment einer tatsächlichen Veränderung der Zellbelegung, tatsächlich wäre es besser, Informationen aus allen Beobachtungen zu ziehen.

Eine bessere Möglichkeit ist die Schätzung der gemeinsamen Verteilung jeweils zweier benachbarter Zellen: Das Occupancy-Grid-Modell beruht darauf, dass jede Zelle zwei Zustände annehmen kann (frei/belegt), zwischen diesen beiden Zuständen wird eine Wahrscheinlichkeitsverteilung geschätzt. Für die Kombination zweier Zellen ergeben sich damit vier mögliche Zustände (frei+frei / frei+belegt / belegt+frei / be-

legt+belegt). Die Wahrscheinlichkeit für jeden dieser Zustände ergibt sich aus dem Produkt der Einzelwahrscheinlichkeiten für den freien bzw. belegten Zustand der Zellen (dies gilt nur unter der Annahme, dass die Beobachtungen der Zustände beider Zellen unabhängig sind). Diese Wahrscheinlichkeiten werden über alle gleichzeitigen Beobachtungen zweier benachbarter Zellen akkumuliert:

$$\boldsymbol{H}^{i,j} = \begin{pmatrix} h_{11}^{ij} & h_{10}^{ij} \\ h_{01}^{ij} & h_{00}^{ij} \end{pmatrix} = \sum_{t=1}^{T} \begin{pmatrix} P(occ^i|o_t)P(occ^j|o_t) & P(occ^i|o_t)P(\overline{occ}^j|o_t) \\ P(\overline{occ}^i|o_t)P(occ^j|o_t) & P(\overline{occ}^i|o_t)P(\overline{occ}^j|o_t) \end{pmatrix} \tag{7.7}$$

Während die einzelnen Beobachtungen als unabhängig betrachtet werden, ergibt sich über den zeitlichen Verlauf eine Abhängigkeit zwischen den Zellzuständen. Über diesen kann aus einer Beobachtung von occ^i eine Prädiktion von occ^j erfolgen („virtuelle Beobachtung“):

$$P'(occ^j|o_t) = P(occ^i|o_t)\frac{h_{11}^{ij}}{h_{11}^{ij} + h_{01}^{ij}} + (1 - P(occ^i|o_t))\frac{h_{10}^{ij}}{h_{10}^{ij} + h_{00}^{ij}} \tag{7.8}$$

Die Modellierung der Zellkorrelationen und Nutzung der sich daraus ergebenden Information für zusätzliche Updates der Nachbarzellen führt dazu, dass die Karte bei wiederkehrenden Änderungen schneller angepasst wird, als dies nur durch die Beobachtungen allein möglich wäre (Abb. 7.9).

MCL mit verteiltem Sampling über mehreren Karten

In den vorhergehenden Abschnitten wurden zwei verschiedene Karten mit speziellen Update-Mechanismen beschrieben. Während die referenzbasierte Karte vor allem hohe Stabilität gegenüber Veränderungen aufweisen soll, wurde bei der Gridkarte gerade Wert auf eine sehr schnelle Adaption an Umgebungsänderungen gelegt. Beide Karten zusammen sollen das Wissen sowohl um langfristig stabile Aspekte der Umwelt als auch den aktuellen Zustand aus den letzten Beobachtungen repräsentieren. Um dieses Wissen für die Selbstlokalisation zu nutzen,

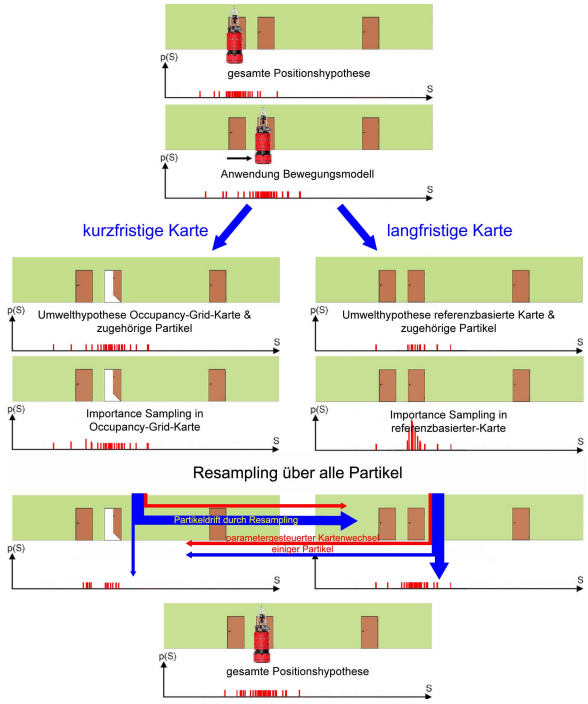

Abbildung 7.10: *Die beiden Karten (referenzbasiert = langsam adaptierend, Gridkarte = schnell adaptierend) werden in der MCL als Alternativen integriert, indem die Partikel über beide Karten verteilt werden. Dies entspricht der Erweiterung des Zustandsraumes um eine zusätzliche Dimension. Durch die Verteilung der Partikel auf beide Karten wird für die Positionsschätzung automatisch diejenige Karte gewählt, die die Beobachtungen am besten erklärt. Die Abbildung ist angelehnt an eine Darstellung zur MCL aus [Thrun et al., 2005].*

müssen beide Karten in das auf einem Partikelfilter basierende MCL-Framework (Abschnitt 5.1) integriert werden.

Zu diesem Zweck wird der zu schätzende Zustand um eine (im Fall von zwei Karten-Alternativen binäre) Variable erweitert, die beschreibt, welche Karte dem aktuellen Zustand entspricht. Ein ähnlicher Ansatz wurde in [Stachniss und Burgard, 2005] vorgestellt, allerdings wurden dort nicht mehrere alternative globale Karten, sondern lokale Ausschnitte (Patches) mit jeweils mehreren möglichen Zuständen betrachtet.

Durch die Erweiterung des Zustandes beschreiben die Partikel nicht nur eine Verteilung über die möglichen Positionen, sondern auch über den möglichen Karten. Je nachdem, welche der Karten besser zu den aktuellen Beobachtungen passt, werden die Partikel, die dieser Karte zugeordnet sind, höher gewichtet. Während des Resamplings werden somit auch mehr Partikel in die jeweils bessere Karte verschoben. Allerdings existiert auch für die Karten ein „Übergangsmodell“, das wie das Bewegungsmodell für den Positionsübergang vom Zeitpunkt $t-1$ zu t eine zufällige Komponente enthält. Dies stellt sicher, dass stets auch Partikel in der momentan weniger gut bewerteten Karte liegen, so dass ein Wechsel möglich ist, wenn an einer anderen Position diese Karte besser passt. Durch die Schätzung einer Verteilung über die Karten ist keine explizite Auswahl der aktuellen Karte notwendig, die Beobachtungen werden immer mit beiden Karten verglichen und damit

Abbildung 7.11: *Ein einfaches Experiment zur Auswirkung der dynamischen Adaption der Karte auf die Lokalisationsgenauigkeit: Während der Roboter wiederholt zwischen den Punkten P1 und P2 pendelte, wurde an der grün markierten Stelle ein Objekt so verschoben, dass es abwechseln die Durchfahrt versperrte oder freigab.*

automatisch die Karte gewählt, die besser zu den Beobachtungen passt (solange genügend Partikel in beiden Karten liegen). Abb. 7.10 stellt das Vorgehen graphisch dar.

Experimente und Bewertung

Abb. 7.11 zeigt ein sehr einfaches Experiment zur Auswertung der Resultate, die durch die Anpassung der Karte an Umgebungsveränderungen erzielt werden. Zunächst wurde das MCL-Verfahren aus Abschnitt 5.1 mit einer vorgegebenen statischen Gridkarte eingesetzt. Während der Roboter zwischen den beiden Positionen P1 und P2 pendelte (insgesamt ca. 20 mal), wurde an der markierten Stelle mehrfach ein Hindernis platziert und wieder entfernt, so dass die Umgebung temporär nicht mehr der Karte entsprach. Die von Roboter bestimmte eigene Position wurde mit der manuell korrigierten Referenzpose verglichen, um den Verlauf des Lokalisationsfehlers zu bestimmen. Bei Nutzung nur der statischen Karte steigt der Lokalisationsfehler an der veränderten Stelle jedes mal leicht an, im Mittel ergibt sich ein Fehler zur Referenzpose von 0.55m.

Für die dynamische Kartenadaption wurde zunächst in einer „Trainingsphase“ zusätzlich zu der vorhandenen Gridkarte eine referenzbasierte Karte der Experimentierumgebung gebaut. Außerdem wurden die Zellkorrelationen (insbesondere an der veränderlichen Stelle) vorgelernt, indem mehrfach Änderungen dort beobachtet wurden. In der Testphase wurde wiederum der Zustand an dieser Stelle mehrfach geändert. Allerdings wurden nun beide Karte zur Lokalisation benutzt und gleichzeitig aus den Beobachtungen aktualisiert. Dadurch wird innerhalb kurzer Zeit jeweils die Karte an den aktuellen Zustand angepasst, danach wird jeweils an der Stelle der Veränderung ebenso eine normale Lokalisations-

genauigkeit erreicht wie in einer statischen Umgebung. Insgesamt sank auf diese Weise der durchschnittliche Fehler gegenüber der Referenzpose auf 0.35m. Obwohl dieses Experiment nur beschränkt aussagefähig bezüglich absoluter Fehler in einem realistischen Szenario ist, zeigt es doch, dass durch kontinuierliche Kartenanpassung eine Verbesserung der Lokalisationsgenauigkeit in einer veränderlichen Umgebung erzielt wird.

Mit dem in diesem Abschnitt vorgestellten Ansatz existiert ein Konzept, das verschiedene Algorithmen und Modelle aus der Literatur aufgreift, um Selbstlokalisation und dynamische Kartenadaption zu einem Gesamtsystem zu kombinieren, und für das positive Ergebnisse vorliegen. Allerdings gibt es auch noch nicht abschließend geklärte Details, wie zum Beispiel die Modellierung der Zellkorrelationen. Ebenso steht eine umfassende experimentelle Evaluation noch aus, bevor dieser Ansatz in den autonomen Serviceroboter integriert werden kann.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die wesentlichen Aussagen und Ergebnisse der Arbeit werden hier noch einmal in kompakter Form zusammengefasst.

In Kapitel 1 werden vor dem Hintergrund der Entwicklung von autonomen interaktiven Servicerobotern im Allgemeinen und eines Shopping-Lotsen für Baumarkt-Kunden im Speziellen Navigationsleistungen definiert, welche notwendig sind, um den autonomen Einsatz eines solchen Roboters zu ermöglichen. Besonderer Wert wird dabei einerseits darauf gelegt, dass keine spezielle Vorbereitung der Umgebung notwendig ist, somit wird z.B. die Platzierung von Lokalisationshilfen wie RFID Tags oder künstlichen Landmarken ausgeschlossen. Weiterhin sollen bei der Entwicklung der Methoden im Vorhinein möglichst wenige Einschränkungen bezüglich der genutzten Sensorik getroffen werden. Insbesondere wird für die Kartierung nach Verfahren gesucht, welche nicht auf die sehr genauen und hoch aufgelösten Messungen eines Laser-Range-Scanners angewiesen sind, sondern auch mit anderen Sensoren, wie z.B. Sonar-Range-Sensoren mit wesentlich geringerer räumlicher Auflösung, robust funktionieren. Beide Vorgaben sind motiviert durch das Bestreben, möglichst große Flexibilität sowohl bei der Wahl der sensorischen Ausstattung eines Serviceroboters als auch beim Einsatz eines solchen Roboters in verschiedenen Umgebungen zu gewährleisten.

Da die Repräsentation eines Umgebungsmodells eine wesentliche Grundlage fast aller Navigationsleistungen darstellt, werden in Kapitel 2 verschiedene Standardmodelle für eine solche Repräsentation mit ihren spezifischen Eigenschaften vorgestellt und in eine Taxonomie eingeordnet. Aufgrund der Möglichkeit zur Repräsentation der Beobachtungen eines weiten Spektrums von Sensoren als auch der guten Eignung z.B. für Pfadplanung und Hindernisdetektion werden Gridkarten als verwendetes Kartenmodell für globale und lokale Karten gewählt.

Kapitel 3 beschreibt zunächst allgemeine Methoden zur Generierung einer solchen Gridkarte aus verschiedenen Sensorbeobachtungen. Neben der Wiederholung bekannter mathematischer Grundlagen zur probabilistischen Schätzung von Zellbelegungen mittels Entfernungssensoren wird vor allem eine hocheffiziente optimierte Implementierung vorgestellt, die auf der Vorberechnung eines Teils der benötigten Daten und deren Speicherung in einer Look-Up-Table beruht. Durch diese Optimierung wurde gegenüber einfacheren Implementierungen eine Geschwindigkeitssteigerung um ca. den Faktor 25 erreicht.

Weiterhin werden Methoden zur Schätzung von Zellbelegungen einer 2D-Gridkarte aus Punkten im 3D-Raum vorgestellt, die zum Beispiel die Generierung von Gridkarten mittels visueller Sensorik ermöglichen. Ebenso wird das Grundprinzip für die Fusion der Beobachtungen mehrerer Sensoren auf Basis jeweils einzeln erzeugter Gridkarten beschrieben.

Einen weiteren Bestandteil dieses Kapitels stellen Methoden zur Odometriekorrektur als eine Grundlage für einfache Kartierungsverfahren dar. Neben einer Methode der Odometriekalibrierung durch Bestimmung des systematischen Fehlers für einen Synchro-Drive-Antrieb wird ein Korrekturverfahren mit externer Referenz vorgestellt, welches eine spezielle Bodenstruktur zur Bestimmung der Roboterorientierung ausnutzt.

Kapitel 4 bildet den inhaltlichen Schwerpunkt der Arbeit. Zunächst werden einfache deterministische Positionskorrektur-Verfahren für einen mobilen Roboter unter Nutzung von Odometrie und Sonar-Sensoren entwickelt, die auf der Nutzung lokaler Karten zur abstrakten Beobachtungsrepräsentation und Map Matching zwischen der lokalen und globalen Karte beruhen. Diese Verfahren werden anschließend zu einem wesentlich mächtigeren probabilistischen Ansatz für Simultaneous Localization and Mapping (SLAM) erweitert. Das Prinzip des Map Matching wird dazu mit einem Rao-Blackwellized Particle Filter (RB-

PF) kombiniert, welcher die Grundlage verschiedener State-of-the-Art-Algorithmen für SLAM mit Gridkarten darstellt. Mit dem Map-Match-SLAM-Algorithmus wird erstmals ein Verfahren vorgestellt, welches die robuste Kartierung großer Indoor-Umgebungen ausschließlich auf Basis von Odometrie und Sonar-Sensoren ermöglicht. Es werden zudem methodische Erweiterungen entwickelt, die die Speicher- und Rechenzeit-Komplexität signifikant verringern, und damit eine Online-Kartierung ermöglichen. Dies umfasst eine spezielle Kartenrepräsentation zur Reduktion der Redundanz im Partikelfilter sowie eine dynamische Steuerung der Partikelanzahl.

Der Map-Match-SLAM-Algorithmus ist allerdings nicht auf die Nutzung von Sonar-Sensoren oder anderen entfernungsmessenden Sensoren beschränkt, sondern kann für alle Sensoren genutzt werden, deren Beobachtungen in einer Gridkarte dargestellt werden können. Dies wird exemplarisch durch die Kartierung mit einer Stereo-Kamera und einer einfachen monokularen Kamera in Verbindung mit einem Depth-from-Motion-Verfahren gezeigt.

Die Entwicklung eines SLAM-Assistenten schließt den Teil zur Umgebungskartierung ab. Aufgabe des SLAM-Assistenten ist es, während einer manuell gesteuerten Kartierungsfahrt dem menschlichen Bediener Hinweise für eine optimale Fahrtroute zu geben. Er überwacht dazu die Zustandsunsicherheit und detektiert Möglichkeiten für Kreisschlüsse. Diese werden dem Bediener jeweils in Form einer vorgeschlagenen Fahrtrichtung angezeigt, wenn ein Kreisschluss aufgrund zu großer Zustandsunsicherheit notwendig wird. Durch den SLAM-Assistenten ist eine robuste Kartierung mit dem SLAM-System möglich, ohne dass der Bediener Kenntnisse über dessen Funktion und die Bedeutung der Zustandsunsicherheit haben muss. Er stellt eine Zwischenstufe zwischen einer rein manuell gesteuerten Kartierung und einer komplett autonomen Exploration dar.

Kapitel 5 beinhaltet die verschiedenen methodischen Beiträge für die eigentliche autonome Funktion des Roboters. Für die Selbstlokalisation kommt dazu ebenso wie beim SLAM ein Partikelfilter in Kombination mit Map Matching mit einer lokalen Karte zum Einsatz. Im Gegensatz zum SLAM wird hierbei allerdings eine einzige globale Karte fest vorgegeben, die von allen Partikeln zum Vergleich mit den jeweils eigenen lokalen Karten genutzt wird.

Die Aufgabe der Zielanfahrt wird in eine Planungs- und eine Ausführungsphase unterteilt. In der Planungsphase wird ein Pfad von der aktuellen Position zum Zielpunkt geplant. Dazu wird die Gridkarte in einen Graphen transformiert, indem jede Zelle als Knoten interpretiert wird, Kanten existieren zwischen direkt benachbarten Zellen. Somit ist es möglich, effiziente Standard-Algorithmen für die Pfadsuche in Graphen zu verwenden. Hier wurden konkret zwei Algorithmen implementiert und verglichen: Neben dem Dijkstra-Algorithmus ist dies der A*-Algorithmus, welcher zusätzlich zu den Kantenlängen eine Heuristik für eine gerichtete Suche nach dem kürzesten Pfad verwendet. Es wird gezeigt, dass der A*-Algorithmus in der konkret untersuchten Umgebung aufgrund deren komplexer Struktur nur geringe Vorteile gegenüber dem Dijkstra-Algorithmus aufweist.

In der Ausführungsphase der Zielanfahrt wird der während der Planung generierte Pfad benutzt, um zum Ziel zu fahren, dabei werden durch die lokale Bewegungssteuerung Kollisionen mit Hindernissen vermieden. Die Kollisionsvermeidung basiert auf einer lokalen Karte als zentrale Repräsentation der Umgebungswahrnehmungen einer beliebigen Anzahl von Sensoren. Auch diese wird wiederum in Form einer Gridkarte repräsentiert. Auch für die Bewegungssteuerung wurden zwei verschiedene Verfahren entwickelt und vergleichend getestet: Zunächst wird ein rein reaktives Verfahren vorgestellt, welches eine Weiterentwicklung des aus der Literatur bekannten Vector Field Histogram (VFH) darstellt. Durch die Kombination mit jeweils einem zusätzlichen Planungsschritt, dessen Ziel vor allem die Anpassung des Pfades an die aktuelle Hindernissituation durch Berücksichtigung der lokalen Karte ist, wird die klare Trennung zwischen Planungs- und Ausführungsphase aufgehoben. Dieser Kombination wird ein antizipatives Verfahren gegenübergestellt, das auf der Auswahl jeweils einer Fahrtrajektorie in regelmäßigen kurzen Zeitintervallen beruht. Für die Trajektorienauswahl werden Ansätze mittels Sampling und mittels stochastischer Suche entwickelt. Um die Verfolgung der jeweils gewählten Trajektorie zu gewährleisten, muss ein Trajektorienregler integriert werden. Für diesen Regler werden drei verschiedene Möglichkeiten dargestellt und diejenige ausgewählt, für die das Problem der Parameterwahl am leichtesten zu lösen ist. Die beiden Verfahren für die lokale Bewegungssteuerung werden experimentell in abgeschlossenen und überschaubaren Beispiel-Szenarien verglichen, und jeweils die Vor- und Nachteile aufgezeigt.

Kapitel 6 bildet zunächst den Abschluss des Hauptteils der Arbeit. Hier werden die zuvor vorgestellten Methoden in das Gesamtsystem des entwickelten Shopping-Lotsen eingeordnet und weitere Komponenten der Anwendung, z.B. für die Mensch-Roboter-Interaktion, in einem Überblick dargestellt. Daneben wird die flexible Steuerarchitektur des Roboters im Detail beschrieben, welche eine modulare Struktur aufweist und damit sowohl eine einfache Übertragung einer Applikation zwischen verschiedenen Roboter-Plattformen als auch die Austauschbarkeit einzelner Module gewährleistet.

Der Nachweis der Eignung der vorgestellten Methoden und Implementierungen gelingt durch die Auswertung von Langzeit-Tests mit mehreren Robotern in einem Baumarkt in Kombination mit Befragungen von Nutzern. Für das Gesamtsystem des autonomen Shopping-Lotsen konnte im dauerhaften Testbetrieb über mehrere Monate eine außerordentlich hohe Akzeptanz durch die Baumarkt-Kunden festgestellt werden.

8.2 Ausblick

In den Kapiteln 4 und 5 wurden schon bei der Bewertung einzelner in dieser Arbeit entwickelter Methoden Ansätze für weitere Verbesserungen genannt. Weitere umfangreiche Ausblicke auf bereits laufende Arbeiten, die über den derzeit existierenden Shopping-Roboter hinausgehen, wurden in Kapitel 7 gegeben. Zunächst sollen all diese Ansätze hier noch einmal aufgegriffen und in kompakter Form zusammengefasst werden.

Im Kapitel 4 wurde erfolgreich ein SLAM-Verfahren entwickelt, welches auf einem Rao-Blackwellized Particle Filter aufbaut und das Map Matching zwischen lokaler und globaler Karte als Grundprinzip für die Bewertung der Zustandshypothesen einsetzt. Allerdings stellt dies keine mathematisch fundierte Approximation der Wahrscheinlichkeit einer Hypothese dar, sondern eine einfache heuristische Lösung. Insbesondere ist der Matchwert nicht geeignet, um die Distanz einer Zustandshypothese zum realen Zustand zu bewerten: Der Match-Wert fällt nicht kontinuierlich mit der Verschiebung der lokalen und globalen Karten gegeneinander, wie sie bei steigender Fehllokalisation auftritt, sondern sie fällt zunächst bei einem leichten Fehler sprunghaft ab und ändert

sich bei weiterem Anwachsen kaum noch. In Abschnitt 4.6 wurden bereits weiterführende Ansätze benannt, die untersucht werden sollten, um dieses Verhalten zu verbessern. Falls hier eine Lösung gefunden werden kann, bei der die Partikelgewichte sich kontinuierlich mit dem Versatz der Karten ändern, so ist zu erwarten, dass eine bessere Konvergenz der Partikel in der lokalen Umgebung des Optimums erzielt wird, dadurch sollten weniger Partikel benötigt werden. Allerdings darf dies nicht signifikant zu Lasten der Rechenzeit bei der Gewichtsberechnung gehen, da dadurch der wesentliche Vorteil einer verringerten Partikelzahl bereits zunichte gemacht würde.

Ein weiterer Punkt, an dem bereits Bedarf für weitere Entwicklung benannt wurde, ist die Multi-Trajektorien-Navigation aus Abschnitt 5.4.2. Insbesondere die als Basis der Trajektorien gewählten Klothoidkurven mit lediglich zwei Parametern $[c_0,\ c_1]$ zur Beschreibung des Krümmungsverlaufs haben sich nicht in allen Situationen als ausreichend für die Bestimmung einer geeigneten Bewegungstrajektorie erwiesen. Hier bleibt zu untersuchen, ob die Möglichkeit der Nutzung von Trajektorien höherer Komplexität besteht, ohne dass die Suche im Trajektorienraum so lange dauert, dass eine Lösung nicht mehr in der verfügbaren Zeit gefunden werden kann. Eventuell kann die Komplexität der Kurven oder die Geschwindigkeit des Roboters dynamisch in Abhängigkeit von der jeweiligen Situation angepasst werden, so dass ausreichende Zeit zur Bestimmung einer Lösung garantiert werden kann. Ein weitere nicht abschließend beantwortete Frage stellt die Wahl der optimalen Kostengewichte dar. Diese erfolgt bisher rein empirisch durch den Entwickler des Verfahrens. Besser wäre hier eine Möglichkeit zur automatischen, eventuell sogar dynamischen Gewichtung.

Kapitel 7 präsentierte bereits sehr ausführlich Konzepte für Erweiterungen des Robotersystems, die Ergänzungen der bereits implementierten und getesteten Funktionalität darstellen. Sowohl die visuelle Hindernisdetektion als auch die dynamische Kartenadaption sind auf lange Sicht notwendige Bestandteile eines autonomen Serviceroboters, für die daher möglichst bald einsatzfähige Lösungen bereitstehen sollen.

Durch die Langzeit-Tests des Robotersystems konnte gezeigt werden, dass die in dieser Arbeit entwickelten Methoden für die Anwendung eines Shopping-Lotsen in der Baumarkt-Umgebung gut geeignet sind. Diese Aussage lässt sich sicherlich bis zu einem gewissen Grad auch auf andere Umgebungen und Anwendungen übertragen. Aus neuen Szena-

rien ergeben sich aber mit Sicherheit auch bestimmte neue Herausforderungen. Dies trifft zum Beispiel auf das Projekt „CompanionAble“ zu, in welchem das Fachgebiet NIKR gegenwärtig gemeinsam mit fast 20 europäischen Partnern an der Integration einer Smart-Home-Umgebung und eines mobilen Serviceroboters als „intelligenter“ Helfer für allein lebende ältere Menschen arbeitet. Hier ergeben sich nicht nur völlig andere Ansprüche an die Gestaltung der Interaktion durch eine dauerhafte 1:1-Beziehung zwischen dem Roboter und seinem Interaktionspartner, sondern eine häusliche Umgebung mit u.a. deutlich weniger Freiraum, wechselnd geöffneten und geschlossenen Türen und vielfältigen Konfigurationen von Hindernissen stellt sicherlich auch andere Anforderungen an die Selbstlokalisation, die Bewegungsplanung und -steuerung sowie die Umgebungsmodellierung. Eine entsprechende Weiterentwicklung der vorhandenen Methoden ist hier also notwendig.

Anhang A

Bestimmung des systematischen Odometriefehlers für einen Synchro-Drive-Antrieb

In diesem Anhang soll die in Abschnitt 3.2.1 kurz zusammengefasste Korrektur des systematischen Odometriefehlers für einen Synchro-Drive-Roboter detaillierter beschrieben werden. Als Experimentalplattform wurde dafür der B21r-Roboter PERSES benutzt. Die Besonderheit des Synchro-Drive ist, dass mehrere (in diesem Fall vier) Räder existieren, welche synchron ausgerichtet (Bestimmung der Fahrtrichtung) und angetrieben (Vorwärtsbewegung) werden. Der B21r setzt dafür 3 Motoren ein: 2 von diesen sind gekoppelt und steuern den Vortrieb der Räder, ein weiterer deren Ausrichtung. Da spezifisch beim B21r der Oberbau mit dem Antrieb so verkoppelt ist, dass eine Änderung der Radorientierung auch den „Körper“ mit dreht, „blickt“ der Roboter stets in Fahrtrichtung. Prinzipiell ist dies aber beim Synchro-Drive nicht notwendig, so könnte ein Roboter mit einem solchen Antrieb auch die Blickrichtung unabhängig von der Fahrtrichtung ändern. Allerdings wäre dann ein zusätzlicher Antrieb zur Drehung der Orientierung notwendig. Ebenso wie der Antrieb ist die Messung von Rotation und Translation unabhängig voneinander und erfolgt durch Encoder an

den entsprechenden Motoren. Zusätzlich existiert ein interner Sensor in Form einer Lichtschranke, der bei Drehung in eine Referenzorientierung (definierte Stellung zwischen Ober- und Unterbau) ein Signal auslöst.

Der mechanische Aufbau eines Synchro-Drive ist zwar deutlich komplizierter als ein Differential-Drive, durch die Entkopplung des Antriebs für Translations- und Rotations-Bewegung ist die Ansteuerung jedoch wiederum sehr einfach. Da alle Räder sich gleich schnell bewegen, treten theoretisch geringere Schlupfkräfte auf, was in einer genaueren Odometrie resultieren sollte. Andererseits muss jede Richtungsänderung durch Drehung der Räder auf der Stelle (gleiten statt rollen) durchgeführt werden, was Schlupf und Abrieb verursacht, da die Auflagefläche des Rades nicht ideal punktförmig ist.

Theoretisch sollte der Unterbau bei PERSES seine Lage in einem globalen Bezugssystem während der Fahrt nicht ändern. Lediglich der Oberbau dreht sich analog zur Änderung der Orientierung (Abb. A.1 links). Die Verdrehung zwischen Ober- und Unterbau kann durch den internen Referenzsensor (Kontakt bei definierter Referenzstellung) initialisiert und anschließend mit Messungen am Rotationsmotor nachverfolgt werden und bestimmt gleichzeitig die Fahrtrichtung. Sie wird auch als Torsion bezeichnet.

Bei PERSES ist zu beobachten, dass die Odometrie zwar bei der Messung der gefahrenen Strecke relativ genau ist, die Orientierungsmessung aber stark von der tatsächlichen Fahrrichtung abweicht. Insbesondere zeigt sich, dass der Roboter bei Vorgabe einer reinen Translation (Rotationsgeschwindigkeit $v_r = 0$) eine Kreisbahn beschreibt, während entsprechend der Motoransteuerung eine Geradeaus-Fahrt gemessen wird (Abbildung A.4). Weiterhin tritt der Effekt auf, dass der Unterbau mit den Rädern dabei einer (geringen) Drehung um sich selbst unterliegt, was nur dadurch geschehen kann, dass die Räder, abweichend von der Spezifikation, unterschiedliche Strecken zurücklegen (Abb. A.1 Mitte). Da dieser Fehler regelmäßig und wiederholbar auftritt, handelt es sich offenbar um einen systematischen Fehler. Als hauptsächliche Ursache wird die unterschiedliche Abnutzung der Antriebsräder (verursacht durch häufigen Einsatz auf unebenen, z.B. gefliesten, Böden) vermutet, welche bei Betrachtung der Räder auch deutlich zu erkennen ist. Ziel der Odometriekorrektur ist also die Bestimmung dieses systematischen Fehlers und darauf aufbauend die Ermittlung der tatsächlich gefahrenen Trajektorie zur Korrektur der Odometriemessungen.

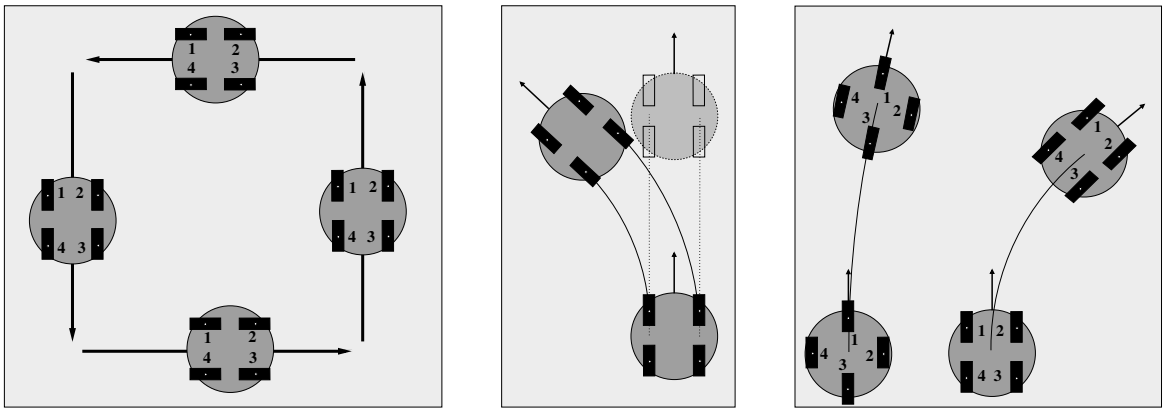

Abbildung A.1: **Links:** *Der Unterbau bleibt bei korrekter Bewegung unverändert gegenüber dem Bezugssystem, lediglich die einzelnen Räder (überproportional vergrößert) drehen sich in Fahrtrichtung.*
Mitte: *Leicht im Umfang abweichende Räder (z.B. durch Abnutzung) legen unterschiedliche Strecken zurück, dadurch dreht sich der Roboter-Unterbau ungewollt gegenüber dem Bezugssystem. Durch diese konstante leichte Rotation wird die Geradeaus-Fahrt zu einer Kreisbahn. Da der Roboter diese Verdrehung nicht beobachten kann, sondern nur die (bei reiner Translation unveränderte) Orientierung der Fahrtrichtung relativ zum Unterbau (Torsion) misst, nimmt er fälschlicherweise eine geradlinige Bewegung an. Dies äußert sich im Odometriefehler.*
Rechts: *Bei Änderung der Torsion ändert sich die Lage der einzelnen Räder relativ zur Fahrtrichtung. Je nach Lage eines Rades hat die Abnutzung unterschiedlichen Einfluss auf die Bewegung. Ein Rad mit zu geringem Umfang (im Beispiel Rad 3) hat wenig Effekt, wenn es sich bzgl. der Fahrtrichtung nahe am Roboterzentrum befindet. Je weiter außen es steht, um so mehr wird die gerade Fahrt zur Kreisbahn, da die zurückgelegte Strecke bei gleicher Umdrehungszahl an diesem Rad niedriger ist.*

Da die Räder während einer Orientierungsänderung an Ort und Stelle bleiben, verändert sich die Lage der einzelnen Räder *relativ zur Fahrtrichtung* ständig, nur bei reiner Geradeaus-Fahrt bleibt die Stellung gleich. Fehler in den Rädern haben je nach aktueller Anordnung einen unterschiedlich starken Effekt auf die gemessene Bewegung: Ein Rad mit verringertem Umfang gegenüber den anderen Rädern führt zu einer Kreisbahn, wenn es sich bzgl. der Fahrtrichtung außerhalb der Robotermitte befindet. In zentraler Stellung hat es dagegen kaum Einfluss auf die Form der gefahrenen Strecke (Abb. A.1 rechts).

Es ist also zu erwarten, dass der Fehler von der relativen Fahrtrichtung $\varphi_{torsion}$ gegenüber dem Unterbau abhängig ist. Diese ist durch den oben beschriebenen Mechanismus mit internem Referenzsensor definiert messbar. Bei gleicher Fahrtrichtung $\varphi_{torsion}$ sollte die Bahnkrümmung

c bzw. der Kreisradius $r = c^{-1}$ immer etwa gleich groß sein. Da sich die Entfernung zwischen einem Rad und der Roboter-Mittelachse (Gerade in Fahrtrichtung durch den Mittelpunkt) gemäß einer Sinuskurve verhält, sollte ferner ein ähnlich gearteter Zusammenhang zwischen $\varphi_{torsion}$ und c bestehen.

Um die Fehlerfunktion zu bestimmen, wurden zunächst die Krümmungen für verschiedene relative Richtungen $\varphi_{torsion}$ gemessen. Ideal wäre es, den Roboter dazu so lange „geradeaus“ fahren zu lassen, bis er einen (Halb-)Kreis beschrieben hat, und dann dessen Radius zu messen. Allerdings sind die tatsächlich auftretenden Fehler dafür zu gering, die Kreisradien betragen bis zu mehrere hundert Meter. Eine solche Messung ist also praktisch kaum möglich, stattdessen muss die Krümmung an kürzeren Kreisfragmenten bestimmt werden. Dazu wurden 2 verschiedene Verfahren getestet: In beiden Fällen fährt der Roboter eine möglichst lange Strecke mit mit einer Rotationsgeschwindigkeit von $v_r = 0$. Es ergibt sich ein Kreisabschnitt, der vermessen werden soll. Bei der ersten Variante wird vom Start- zum Endpunkt die Distanz entlang der ursprünglichen Orientierung am Startpunkt sowie die seitliche Abweichung gemessen (Abb. A.2 links). Aus diesen kann mittels Einsetzen in die Kreisgleichung die Krümmung berechnet werden:

$$r^2 = (x^2) + (y - r)^2 \tag{A.1}$$

$$r = \frac{x^2 + y^2}{2y} \tag{A.2}$$

$$c = \frac{2y}{x^2 + y^2} \tag{A.3}$$

Es ist also nur das Messen der beiden Längen x und y notwendig, allerdings muss die Ausrichtung exakt entlang der Anfangsorientierung des Roboters erfolgen. Kleine Abweichungen hierbei machen die hohe theoretische Genauigkeit beim Messen der Entfernungen zunichte.

Eine andere Variante ist das Messen der Orientierungsänderung des Roboters zwischen Start- und Endpunkt. Zusätzlich wird die gefahrene Strecke verwendet, diese wird von der Odometrie immer noch recht genau wiedergegeben. Der Winkel verhält sich zur vollen Drehung wie die Strecke zum Kreisumfang:

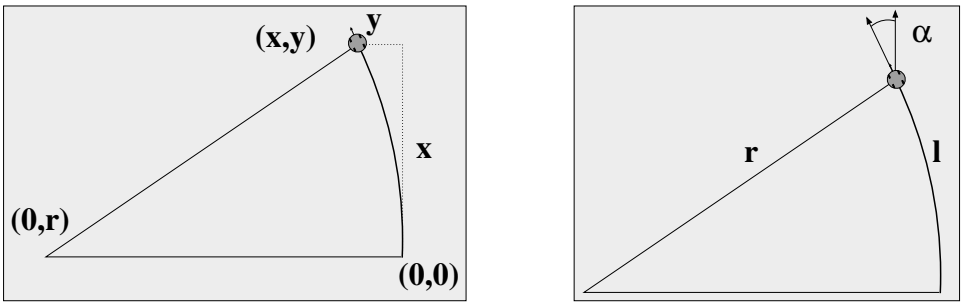

Abbildung A.2: *Die Bahnkrümmung kann durch Messung der seitlichen Abweichung von der geraden Bahn (links) oder durch Messung der Orientierungsänderung bestimmt werden (rechts).*

$$\frac{\alpha}{2\pi} = \frac{l}{2\pi r} \tag{A.4}$$

$$r = \frac{l}{\alpha} \tag{A.5}$$

$$c = \frac{\alpha}{l} \tag{A.6}$$

Für die Winkelmessung wurde ein auf dem Roboter installierter Laser-Range-Scanner benutzt: In einem geraden Gang wurde der Winkel gegenüber der Seitenwand aus den Lasermessungen ermittelt, damit kann die Verdrehung des Roboters im Bezugssystem recht genau bestimmt werden. Da dies auch relativ einfach automatisiert werden kann, ist diese zweite Methode deutlich effizienter. Abbildung A.3 (links) zeigt die auf diese Weise gewonnenen Messpunkte (Torsion und zugehörige Bahnkrümmung).

Um eine Prädiktion der Bahnkrümmung für eine beliebige Torsion zu ermöglichen, muss eine geeignete Funktion gefunden werden, welche die experimentell bestimmten Messwerte beschreibt und eine Interpolation zwischen diesen erlaubt. Sowohl die Lage der Messpunkte als auch die vorgenannten Überlegungen zum Beitrag der einzelnen Räder deuten auf einen sinusförmigen Zusammenhang hin. Eine generalisierte Sinusfunktion (mit fester Periodenlänge 2π) wird beschrieben durch

$$c(\varphi_{torsion}) = offset + scale \cdot \sin(\varphi_{torsion} + phase) \tag{A.7}$$

Die optimalen Parameter $offset$, $scale$ und $phase$ wurden mittels einer einfachen Suche über den plausiblen Wertebereich und Minimierung

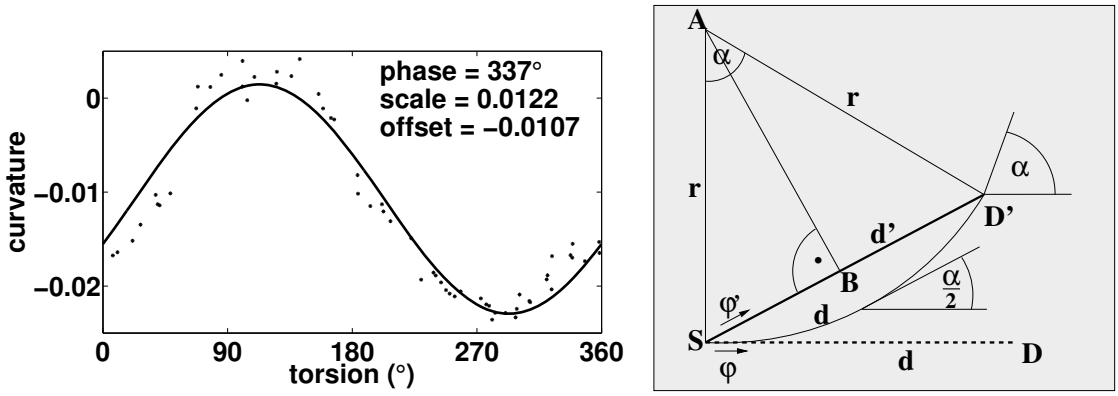

Abbildung A.3: **Links:** *Die gemessene Bahnkrümmung in Abhängigkeit von der Torsion (Fahrtrichtung). Zusätzlich ist eine Best-Fit-Sinusfunktion dargestellt, deren Parameter mittels einer einfachen Optimumsuche bestimmt wurden.* **Rechts:** *Korrektur der Odometrie-Messung bei bekannter Bahnkrümmung c (und damit Radius r): Die Odometrie misst die Strecke d und Fahrtrichtung φ und ermittelt damit als neue Position den Punkt D. Tatsächlich folgt der Roboter jedoch der Kreisbahn zum Endpunkt D'. Dieser wird bestimmt, indem die Ersatzgrößen d' und φ' berechnet werden (Gl. A.10, A.11).*

des quadratischen Fehlers zu den Messpunkten ermittelt. Es ist gut zu erkennen, dass die Punkte recht gut dieser Sinusfunktion folgen, dass jedoch auch Abweichungen bestehen bleiben. Als Ursache dieser Abweichungen kommen sowohl überlagerte nichtsystematische Fehler während der Messfahrt als auch ein unzureichendes Modell der systematischen Fehler in Frage. Da jedoch das Ziel nicht darin besteht, die Position aus der Odometrie exakt zu bestimmen (was ohnehin wegen der unvermeidlichen nichtsystematischen Fehler illusorisch wäre), werden die verbleibenden Modellfehler zu Gunsten eines einfachen Modells und einer einfachen Bestimmung der Parameter (wie oben beschrieben) in Kauf genommen.

Mit der bekannten Krümmung kann nun die Odometriemessung korrigiert werden. Dazu werden die jeweils gemessenen kurzen Geradenstücken in entsprechende Kreisbögen umgerechnet und damit eine korrigierte Endposition berechnet.

Für das rechtwinklige Dreieck ABS in Abb. A.3 (rechts) gilt (unter Nutzung der Beziehung $c = r^{-1}$)

$$\sin\left(\frac{\alpha}{2}\right) = \frac{\frac{d'}{2}}{r} \tag{A.8}$$

$$= \frac{d'}{2} \cdot c(\varphi_{torsion}) \tag{A.9}$$

$$d' = \frac{2 \cdot \sin(\frac{\alpha}{2})}{c(\varphi_{torsion})} \tag{A.10}$$

Für φ' gilt

$$\varphi' = \varphi + \frac{\alpha}{2} \tag{A.11}$$

Durch Einsetzen der Beziehung aus Gleichung A.6 ergibt sich

$$d' = \frac{2 \cdot \sin\left(\frac{d \cdot c(\varphi_{torsion})}{2}\right)}{c(\varphi_{torsion})} \tag{A.12}$$

$$\varphi' = \varphi + \frac{d \cdot c(\varphi_{torsion})}{2} \tag{A.13}$$

Abbildung A.4 stellt originale und korrigierte Odometrie gegenüber. Beide Male fuhr der Roboter einen ca. 35 Meter langen Gang (Abb. 3.9) insgesamt 10 mal hin und her, um letztendlich wieder an der Ausgangsposition anzukommen. Da aufgrund der beschriebenen Fehler der Roboter bei reiner Translations-Ansteuerung eine Kreisbahn beschreibt, muss durch leichte Rotationen gegengesteuert werden, um eine annähernd geradlinige Fahrt zu erreichen. Dies führt dazu, dass die Odometrie fälschlich jeweils eine Kurvenfahrt misst.

Am linken Bild ist sehr gut zu erkennen, dass die auftretenden Fehler relativ regelmäßig sind. Im rechten Bild sind die Odometriemessungen nach der beschriebenen Korrektur zu sehen. Es ist offensichtlich, dass auch mit dieser Korrektur die Position immer noch recht ungenau ist und der Positionsfehler unbegrenzt anwachsen kann. Allerdings sind die Fehler deutlich geringer als ohne Korrektur. Damit ist eine Basis gegeben, um durch weiterführende Ansätze (Abschnitt 4.1, Kapitel 4) während des Kartenaufbaus den verbleibenden Odometriefehler zu berücksichtigen und zu korrigieren.

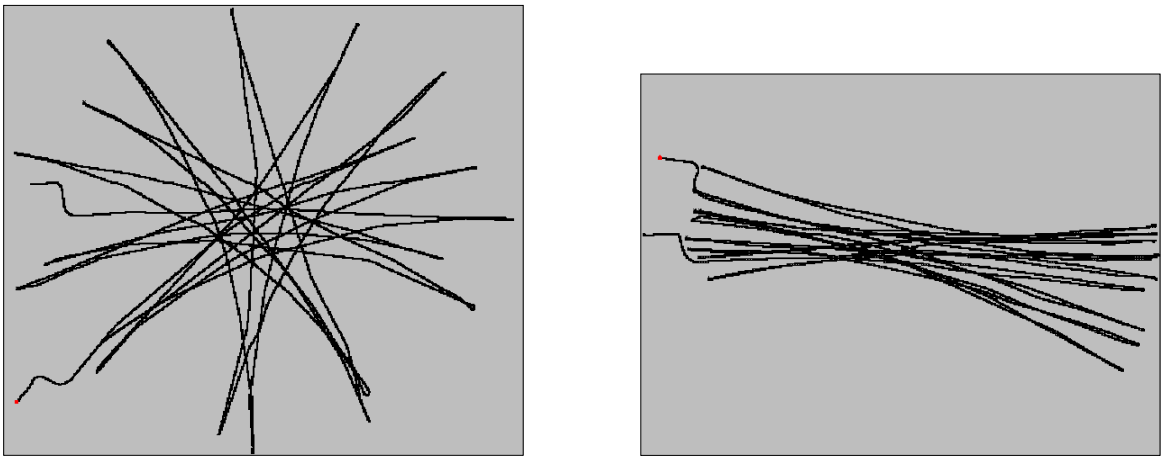

Abbildung A.4: **Links:** *Roboter-Pfad nach Original-Odometrie.* **Rechts:** *Pfad mit Odometrie-Korrektur. Die Odometriemessungen sind immer noch fehlerbehaftet und der Positionsfehler wächst unbegrenzt, aber die Größenordnung der Abweichung ist deutlich geringer.*

Anhang B

Probabilistische Zustandsschätzung

Ein grundlegendes Problem in der Robotik ist die Schätzung eines Systemzustandes x aus Beobachtungen, welche in Form von Sensormessungen vorliegen. Dabei kann es sich sowohl um statische (zeitinvariante) als auch um dynamische (zeitveränderliche) Zustände handeln. Der Fall der kontinuierlichen Schätzung eines dynamischen Zustandes wird auch als Tracking bezeichnet. Konkrete Beispiele für solche Zustandsschätzungs-Probleme, welche auch in dieser Arbeit betrachtet werden, sind der Aufbau einer Gridkarte (x = Belegtheit einer Zelle bzw. Belegtheitsvektor für viele Zellen, statisch), Selbstlokalisation eines mobilen Roboters (x = Roboterposition, dynamisch) oder Simultaneous Localization and Mapping (SLAM) (x = Zell-Belegtheitsvektor + Roboterposition, dynamisch).

Bei der probabilistischen Zustandsschätzung werden alle beteiligten Variablen (u.a. Zustände, Aktionen, Beobachtungen) als Zufallsgrößen betrachtet. Dies ist notwendig, weil die Systemprozesse und Beobachtungsmethoden meist nicht rein deterministisch, sondern nur durch probabilistische Modelle beschrieben werden können. Ursache dafür sind sowohl die zu Grunde liegenden physikalischen Prozesse als auch die Menge der möglichen Einflussgrößen in realen Systemen und deren häufig nicht genau bestimmbare Auswirkung auf die Zielgröße.

Ziel der mathematischen Modellierung ist die Darstellung der Abhängigkeitsverhältnisse zwischen den Wahrscheinlichkeitsdichteverteilungen der einzelnen Variablen $p(V)$ und darauf aufbauend die Schätzung

der unbekannten Verteilungen aus den vorhandenen Informationen. Das zentrale Element bildet das Bayes-Theorem, aus diesem lassen sich Formeln für die so genannte Rekursive Bayes-Schätzung ableiten. Diese bilden eine rekursive Darstellung für die Wahrscheinlichkeitsverteilung des gesuchten Zustandes, d.h. die vorherige Schätzung wird jeweils beim Gewinn neuer Information (i.A. eine neue Sensormessung) aktualisiert, ohne dass die vorausgegangenen Beobachtungen erneut einbezogen werden müssten. Algorithmen bzw. Berechnungsvorschriften zur (rekursiven) Zustandsschätzung in probabilistischen Modellen werden auch mit dem Begriff (rekursive) Bayes-Filter bezeichnet.

Je nach Form und Repräsentation der zu Grunde liegenden Wahrscheinlichkeitsverteilungen für Systemzustand und Beobachtung sind diese Formeln jedoch im Allgemeinen nicht direkt zur Berechnung der aktuellen Schätzung anwendbar. Nur für bestimmte Klassen von Verteilungen ist eine analytische Lösung der rekursiven Formel tatsächlich berechenbar. Neben der theoretischen Beschreibung existieren daher spezielle Bayes-Filter für jeweils spezifische Problemklassen. Die bekanntesten und auch im Bereichder Robotik am häufigsten eingesetzen Varianten sind hier der Kalmanfilter (KF), besonders in der Variante als Erweiterter Kalmanfilter (EKF), und der Partikelfilter (PF). Der Zweck dieser speziellen Implementationen besteht in der vereinfachten Schreibweise und effizienten Berechnung der Schätzung für bestimmte, häufig auftretende Formen von Wahrscheinlichkeitsverteilungen (Kalmanfilter) oder der möglichst genauen und effizienten Approximation für solche Fälle, in denen die exakte Berechnung nicht möglich ist (Partikelfilter).

In diesem Kapitel sollen die theoretische Herleitung der Rekursiven Bayes-Schätzung und die darauf aufbauenden Kalmanfilter und Partikelfilter für eine abstrakte Zustandsschätzung beschrieben werden. Es bildet damit eine theoretische Grundlage für verschiedene in der Arbeit vorgestellte Anwendungsfälle und Algorithmen.

B.1 Rekursive Bayes-Schätzung

Die bedingte Wahrscheinlichkeit einer Zufallsgröße a bei bekannter Zufallsgröße b ist definiert als

$$p(a|b) = \frac{p(a,b)}{p(b)} \tag{B.1}$$

wobei $p(a,b)$ die Verbundwahrscheinlichkeit von a und b bezeichnet. Aus dem Vertauschen von a und b folgt ebenso

$$p(b|a) = \frac{p(a,b)}{p(a)} \tag{B.2}$$

Durch Zusammenfassung der beiden Gleichungen ergibt sich das Bayes-Theorem

$$p(a|b) = \frac{p(b|a) \cdot p(a)}{p(b)} \tag{B.3}$$

Für die Anwendung des Bayes-Theorems wird zunächst der Fall betrachtet, dass ein statischer Zustand aus einer Menge von Beobachtungen geschätzt werden soll. Abb. B.1 zeigt ein graphisches Modell für ein solches Problem. Der Wert der Zufallsvariable x ist zunächst unbekannt. Durch Messungen entstehen jedoch untereinander unabhängige Zufallsgrößen $o_1, \ldots, o_n$, welche beobachtet werden. Es wird dabei vorausgesetzt, dass der Zusammenhang zwischen dem Zustand x und jeweils einer einzelnen Messung o in Form der bedingten Wahrscheinlichkeitsdichte $p(o|x)$ bekannt ist. Im Sinne der Allgemeinheit der Lösung wird angenommen, dass sowohl x als auch o auf einem kontinuierlichen Wertebereich definiert sind. Für diskrete Zufallsgrößen kann eine Lösung leicht aus dem kontinuierlichen Fall durch Vereinfachung abgeleitet werden.

Das Ziel besteht darin, jeweils nach einer Beobachtung o_t zum Zeitpunkt t die neue Schätzung des Zustandes $p(x|o_{1:t})$ zu berechnen. Diese Schätzung wird häufig auch als *Belief* bezeichnet. Vor der ersten Beobachtung muss zunächst noch eine a-priori Schätzung $p(x)$ definiert werden, die das Wissen über den Zustand x in Abwesenheit jeglicher Beobachtungen beschreibt. Im allgemeinsten Fall wird sie als Gleichverteilung über den gesamten Zustandsraum angenommen. Es kann hier

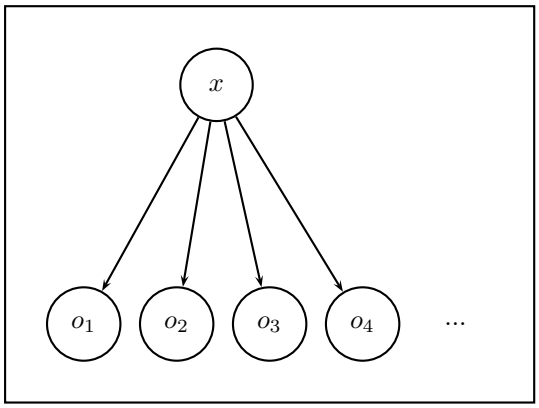

Abbildung B.1: *Graphisches Modell einer Folge von Beobachtungen o_t für einen statischen Zustand x.*

allerdings auch bereits anwendungsspezifisches Wissen einfließen, z.B. können bei einer Lokalisationsaufgabe evtl. bestimmte Positionen von vorn herein ausgeschlossen werden.

Durch Anwendung des Bayes-Theorems ergibt sich

$$p(x|o_{1:t}) = p(x|o_t, o_{1:t-1}) \tag{B.4}$$

$$= \frac{p(o_t|x, o_{1:t-1}) \cdot p(x|o_{1:t-1})}{p(o_t|o_{1:t-1})} \tag{B.5}$$

und unter Annahme der Unabhängigkeit zwischen den einzelnen Beobachtungen

$$p(x|o_{1:t}) = \frac{p(o_t|x) \cdot p(x|o_{1:t-1})}{p(o_t)} \tag{B.6}$$

Für den Nenner $p(o_t)$ gilt

$$p(o_t) = \int_x p(o_t|x) \cdot p(x)\ dx \tag{B.7}$$

Praktisch stellt er jedoch lediglich einen Normierungsfaktor dar und muss daher nicht explizit bestimmt werden bzw. ergibt sich ohnehin aus der Anforderung $\int p(x)\ dx = 1$.

Im dynamischen Fall (Abb. B.2 links) erweitert sich das Modell zu einer Folge von zeitlich aufeinanderfolgenden Zuständen x_t. Die Zustände bil-

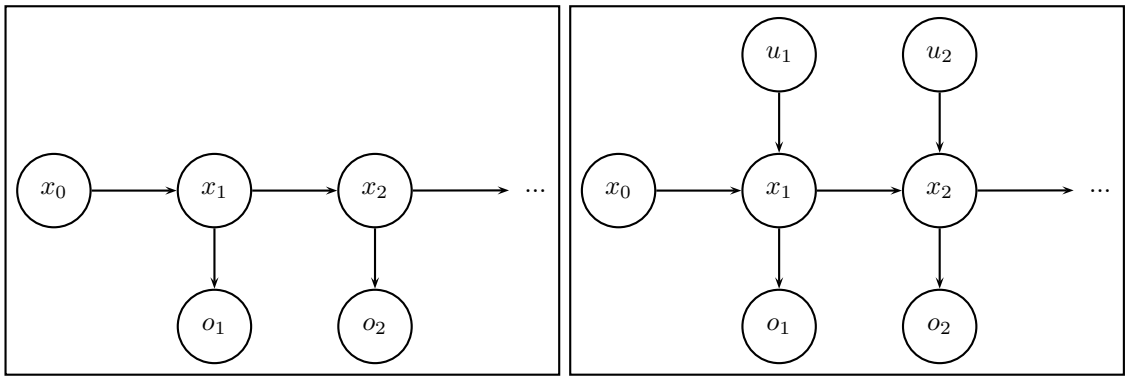

Abbildung B.2: **Links:** *Graphisches Modell eines dynamischen Zustandes x_t und jeweils zugehöriger Beobachtung o_t.* **Rechts:** *Das Modell bei Hinzunahme jeweils eines Steuerkommandos u_t, welches die Änderung des Systemzustands beeinflusst.*

den eine Markov-Kette, d.h. (vereinfacht) jeder Zustand ist jeweils nur abhängig von seinem Vorgängerzustand. Obwohl es sich meist um einen zeitlich kontinuierlichen Prozess handelt (z.B. die Bewegung eines Roboters), wird typischerweise nur der Zustand zum Zeitpunkt der Beobachtung geschätzt. Dadurch wird eine Folge von diskreten Zeitpunkten t ausgewählt, somit entsteht die diskrete Abfolge von Zuständen x_t.

Gleichung B.6 kann direkt übertragen werden:

$$p(x_t|o_{1:t}) = \frac{p(o_t|x_t) \cdot p(x_t|o_{1:t-1})}{p(o_t)} \tag{B.8}$$

Wegen des zwischen den Beobachtungen veränderten Zustandes ist dies aber nun nicht mehr ausreichend für die Rekursion, es wird noch der Bezug zur vorherigen Schätzung $p(x_{t-1}|o_{1:t-1})$ benötigt. Hier gilt analog zu Gleichung B.7

$$p(x_t) = \int_{x_{t-1}} p(x_t|x_{t-1}) \cdot p(x_{t-1}) \; dx_{t-1} \tag{B.9}$$

Damit ergibt sich insgesamt

$$p(x_t|o_{1:t}) = \frac{p(o_t|x_t) \cdot \int\limits_{x_{t-1}} p(x_t|x_{t-1}, o_{1:t-1}) \cdot p(x_{t-1}|o_{1:t-1})\ dx_{t-1}}{p(o_t)}$$
$$= \frac{p(o_t|x_t) \cdot \int\limits_{x_{t-1}} p(x_t|x_{t-1}) \cdot p(x_{t-1}|o_{1:t-1})\ dx_{t-1}}{p(o_t)} \quad \text{(B.10)}$$

Zusätzlich wird häufig im Modell die Möglichkeit von Steueraktionen u_t berücksichtigt, mit denen der Verlauf der Zustandsänderung beeinflusst wird (z.B. ein Fahrkommando für einen mobilen Roboter). Das entstehende graphische Modell ist in Abb. B.2 (rechts) dargestellt. Die Gleichungen B.9 und B.10 werden dann erweitert (bei Unabhängigkeit zwischen allen o und u) zu

$$p(x_t|u_t) = \int\limits_{x_{t-1}} p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1})\ dx_{t-1} \quad \text{(B.11)}$$

$$p(x_t|o_{1:t}, u_{1:t}) = \frac{p(o_t|x_t) \cdot \int\limits_{x_{t-1}} p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1})\ dx_{t-1}}{p(o_t)} \quad \text{(B.12)}$$

Vor allem in praktischen Anwendungen wird $p(o_t|x_t)$ häufig als Beobachtungsmodell und $p(x_t|x_{t-1}, u_t)$ als Prozessmodell oder Bewegungsmodell bezeichnet.

B.2 Kalmanfilter

Der Kalmanfilter [Kalman, 1960] stellt eine spezielle Form des Bayes-Filters dar, bei dem konkrete Annahmen über die Form der auftretenden Wahrscheinlichkeitsverteilungen getroffen werden. Ein Zustand zu beliebigem Zeitpunkt $p(x_t|o_{1:t}, u_{1:t})$ muss normalverteilt sein, d.h. die Verteilung kann jeweils durch eine Gauss-Funktion im n-dimensionalen Raum mit den Parametern Mittelwert μ_t und Kovarianzmatrix Σ beschrieben werden:

$$x_t \sim \mathcal{N}(\mu_t, \Sigma_t) \tag{B.13}$$

$$\sim \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\Big\{-\frac{1}{2}(x_t-\mu_t)^T\Sigma^{-1}(x_t-\mu_t)\Big\} \tag{B.14}$$

Weiterhin werden sowohl das Prozessmodell $p(x_t|x_{t-1}, u_t)$ als auch das Beobachtungsmodell $p(o_t|x_t)$ als lineare Funktionen mit normalverteiltem, mittelwertfreiem und unkorreliertem Rauschen angenommen:

$$x_t = \boldsymbol{A}x_{t-1} + \boldsymbol{B}u_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \boldsymbol{R}) \tag{B.15}$$

$$o_t = \boldsymbol{C}x_t + \delta_t \qquad \delta_t \sim \mathcal{N}(0, \boldsymbol{Q}) \tag{B.16}$$

Die neue Schätzung wird jeweils rekursiv in zwei Schritten berechnet: Zunächst wird eine *prediction* $p(x_t|o_{1:t-1}, u_{1:t})$ ermittelt. In diese geht die Beobachtung o_t nicht ein, sie berücksichtigt lediglich die Schätzung aus dem vorherigen Zeitschritt sowie das Prozessmodell. Die prediction entspricht dem Integral in Gleichung B.12:

$$p(x_t|o_{1:t-1}, u_{1:t}) = \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1})\, dx_{t-1} \tag{B.17}$$

Die prädizierten Mittelwert- und Kovarianz-Parameter werden als $\bar{\mu}_t$ und $\bar{\Sigma}_t$ bezeichnet. Die prediction wird dann im *correct*-Schritt unter Berücksichtigung der neuen Beobachtung zur gesamten Schätzung $p(x_t|o_{1:t}, u_{1:t})$ verrechnet:

$$p(x_t|o_{1:t}, u_{1:t}) = \frac{p(o_t|x_t) \cdot p(x_t|o_{1:t-1}, u_{1:t})}{p(o_t)} \tag{B.18}$$

Die Herleitung der Kalman-Gleichungen aus der allgemeinen Bayes-Schätzung orientiert sich an [Thrun et al., 2005], ist hier aber zur besseren Übersichtlichkeit etwas verkürzt. Für eine ausführlichere Darstellung und Nachweis der Korrektheit aller Umformungen sei auf diese Quelle verwiesen.

Für die im predict-Schritt (Gl. B.17) benutzten Verteilungen lassen sich folgende Gleichungen aufstellen:

$$p(x_t|x_{t-1}, u_t) = \mathcal{N}(\boldsymbol{A}x_{t-1} + \boldsymbol{B}u_t, \boldsymbol{R}) \quad \text{(B.19)}$$

$$p(x_{t-1}|o_{1:t-1}, u_{1:t-1}) = \mathcal{N}(\mu_{t-1}, \Sigma_{t-1}) \quad \text{(B.20)}$$

Daraus folgt

$$\begin{aligned} & p(x_t|o_{1:t-1}, u_{1:t}) \\ & = \eta \int_{x_{t-1}} \exp\Big\{ \quad -\frac{1}{2}(x_t - (\boldsymbol{A}x_{t-1} + \boldsymbol{B}u_t))^T \boldsymbol{R}^{-1}(x_t - (\boldsymbol{A}x_{t-1} + \boldsymbol{B}u_t)) \\ & \qquad\qquad -\frac{1}{2}(x_t - \mu_{t-1})^T \Sigma_{t-1}^{-1}(x_t - \mu_{t-1}) \Big\}\, dx_{t-1} \end{aligned} \quad \text{(B.21)}$$

η stellt einen konstanten Normierungsfaktor dar, der sich (falls nötig) aus der Bedingung für Wahrscheinlichkeitsverteilungen $\int p(x)\ dx = 1$ rekonstruieren lässt. In den folgenden Gleichungen werden alle konstanten Faktoren in η zusammengefasst, der Wert des Normierungsfaktors ist also nicht unbedingt konstant über mehrere Formeln.

Das Integral kann aufgelöst werden, indem durch Umsortieren der einzelnen Terme des Exponenten wiederum ein Produkt zweier Exponentialfunktionen gebildet wird, von denen eine unabhängig von x_{t-1} ist (und somit aus dem Integral herausgelöst werden kann), das Integral über den verbleibenden zweiten Faktor wird konstant:

$$\Psi_t = (\boldsymbol{A}^T \boldsymbol{R}^{-1} \boldsymbol{A} + \Sigma_{t-1}^{-1})^{-1} \quad \text{(B.22)}$$

$$\nu_t = \Psi_t[\boldsymbol{A}^T \boldsymbol{R}^{-1}(x_t - \boldsymbol{B}u_t) + \Sigma_{t-1}^{-1}\mu_{t-1}] \quad \text{(B.23)}$$

$$
\begin{aligned}
& p(x_t|o_{1:t-1}, u_{1:t}) \\
&= \eta \int_{x_{t-1}} \exp\Big\{ -\frac{1}{2}(x_t - \boldsymbol{A}\mu_{t-1} - \boldsymbol{B}u_t)^T \\
&\qquad \cdot (\boldsymbol{A}\Sigma_{t-1}\boldsymbol{A}^T + \boldsymbol{R})^{-1}(x_t - \boldsymbol{A}\mu_{t-1} - \boldsymbol{B}u_t) \\
&\qquad -\frac{1}{2}(x_t - \nu_t)^T \Psi_t^{-1}(x_t - \nu_t) \Big\}\, dx_{t-1} \qquad \text{(B.24)} \\
&= \eta \exp\Big\{ -\frac{1}{2}(x_t - \boldsymbol{A}\mu_{t-1} - \boldsymbol{B}u_t)^T \\
&\qquad \cdot (\boldsymbol{A}\Sigma_{t-1}\boldsymbol{A}^T + \boldsymbol{R})^{-1}(x_t - \boldsymbol{A}\mu_{t-1} - \boldsymbol{B}u_t)\Big\} \\
&\quad \cdot \int_{x_{t-1}} \exp\Big\{ -\frac{1}{2}(x_t - \nu_t)^T \Psi_t^{-1}(x_t - \nu_t) \Big\}\, dx_{t-1} \qquad \text{(B.25)}
\end{aligned}
$$

$$
\int_{x_{t-1}} \exp\Big\{ -\frac{1}{2}(x_t - \nu_t)^T \Psi_t^{-1}(x_t - \nu_t) \Big\}\, dx_{t-1} = const, \text{unabhängig von } x_t \qquad \text{(B.26)}
$$

$$
p(x_t|o_{1:t-1}, u_{1:t}) = \mathcal{N}(\boldsymbol{A}\mu_{t-1} + \boldsymbol{B}u_t, \boldsymbol{A}\Sigma_{t-1}\boldsymbol{A}^T + \boldsymbol{R}) \qquad \text{(B.27)}
$$

$$
\begin{aligned}
\bar{\mu}_t &= \boldsymbol{A}\mu_{t-1} + \boldsymbol{B}u_t & \text{(B.28)} \\
\bar{\Sigma}_t &= \boldsymbol{A}\Sigma_{t-1}\boldsymbol{A}^T + \boldsymbol{R} & \text{(B.29)}
\end{aligned}
$$

Im correct-Schritt wird wiederum die prädizierte Zustandsschätzung mit der Beobachtungs-Wahrscheinlichkeit für die tatsächliche Beobachtung o_t multipliziert:

$$
p(o_t|x_t) = \mathcal{N}(\boldsymbol{C}x_t, \boldsymbol{Q}) \qquad \text{(B.30)}
$$

$$
\begin{aligned}
& p(x_t|o_{1:t}, u_{1:t}) \\
&= \eta \ \exp\{- \ \frac{1}{2} \ ((o_t - \boldsymbol{C}x_t)^T \boldsymbol{Q}_t^{-1}(o_t - \boldsymbol{C}x_t) \\
&\qquad\qquad + (x_t - \bar{\mu}_t)^T \bar{\Sigma}_t^{-1}(x_t - \bar{\mu}_t)) \ \} \\
&= \eta \ \exp\{- \ \frac{1}{2} \ (x_t - (\bar{\mu}_t + (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1}\boldsymbol{C}^T\boldsymbol{Q}^{-1}(o_t - \boldsymbol{C}\bar{\mu}_t)))^T \\
&\qquad\qquad \cdot (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1}) \\
&\qquad\qquad \cdot (x_t - (\bar{\mu}_t + (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1} \\
&\qquad\qquad \cdot \boldsymbol{C}^T\boldsymbol{Q}^{-1}(o_t - \boldsymbol{C}\bar{\mu}_t))) \ \} \qquad \text{(B.31)}
\end{aligned}
$$

$$
\begin{aligned}
p(x_t|o_{1:t}, u_{1:t}) = \mathcal{N}(&\bar{\mu}_t + (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1}\boldsymbol{C}^T\boldsymbol{Q}^{-1}(o_t - \boldsymbol{C}\bar{\mu}_t), \\
&(\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1} \ ) \qquad \text{(B.32)}
\end{aligned}
$$

$$
\begin{aligned}
\mu_t &= \bar{\mu}_t + (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1}\boldsymbol{C}^T\boldsymbol{Q}^{-1}(o_t - \boldsymbol{C}\bar{\mu}_t) \qquad \text{(B.33)} \\
\Sigma_t &= (\boldsymbol{C}^T\boldsymbol{Q}^{-1}\boldsymbol{C} + \bar{\Sigma}_t^{-1})^{-1} \qquad \text{(B.34)}
\end{aligned}
$$

Mit der Definition des Kalman-Gain $\boldsymbol{K}_t$ ergibt sich

$$
\begin{aligned}
\boldsymbol{K}_t &= \bar{\Sigma}_t\boldsymbol{C}^T(\boldsymbol{C}\bar{\Sigma}_t\boldsymbol{C}^T + \boldsymbol{Q})^{-1} \qquad \text{(B.35)} \\
\mu_t &= \bar{\mu}_t + \boldsymbol{K}_t(o_t - \boldsymbol{C}\bar{\mu}_t) \qquad \text{(B.36)} \\
\Sigma_t &= (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{C})\bar{\Sigma}_t \qquad \text{(B.37)}
\end{aligned}
$$

Erweiterter Kalmanfilter

Insbesondere die Einschränkung auf ein lineares Prozess- und Beobachtungsmodell ist eine Forderung, die in der Praxis häufig nicht erfüllt wird. Für nichtlineare Prozesse wurde daher der Erweiterte Kalmanfilter (EKF) [Sorenson, 1970] entwickelt, der für beliebige (stetig differenzierbare) Zustandsübergangs- und Beobachtungsfunktionen anwendbar ist.

$$x_t = f(x_{t-1}, u_t) + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \boldsymbol{R}) \tag{B.38}$$
$$o_t = h(x_t) + \delta_t \qquad \delta_t \sim \mathcal{N}(0, \boldsymbol{Q}) \tag{B.39}$$

Die Funktionen f und h werden direkt benutzt, um den neuen Zustands-Erwartungswert $\bar{\mu}_t$ bzw. die erwartete Beobachtung zu prädizieren, und ersetzen damit die linearen Transformationsmatrizen $\boldsymbol{A}$, $\boldsymbol{B}$ und $\boldsymbol{C}$ in den Gleichungen B.28 und B.36.

$$\bar{\mu}_t = f(\mu_{t-1}, u_t) \tag{B.40}$$
$$\mu_t = \bar{\mu}_t + \boldsymbol{K}_t(o_t - h(\bar{\mu}_t)) \tag{B.41}$$

Für die Adaption der Kovarianzmatrix und die Berechnung des Kalman-Gains $\boldsymbol{K}_t$ müssen allerdings die Funktionen f und h im Punkt der aktuellen Schätzung linearisiert werden. Dies geschieht durch Berechnung der Jakobi-Matrizen, welche die partiellen Ableitungen der Funktionen nach dem Zustandsvektor darstellt. Die Vorgehensweise entspricht der Taylor-Entwicklung bis zum 1. Glied.

$$\boldsymbol{A}_t = \left.\frac{\partial f(x_{t-1}, u_t)}{\partial x_{t-1}}\right|_{\mu_{t-1}, u_t} \tag{B.42}$$
$$\boldsymbol{C}_t = \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{\bar{\mu}_t} \tag{B.43}$$

Mit dem Einsetzen der (jetzt in jedem Schritt t unter Berücksichtigung des aktuell geschätzten Zustandes neu berechneten) Transformationsmatrizen $\boldsymbol{A}_t$ und $\boldsymbol{C}_t$ gelten die Gleichungen B.29, B.35 und B.37 analog zum einfachen Kalmanfilter.

$$\bar{\Sigma}_t = \boldsymbol{A}_t \Sigma_{t-1} \boldsymbol{A}_t^T + \boldsymbol{R} \tag{B.44}$$
$$\boldsymbol{K}_t = \bar{\Sigma}_t \boldsymbol{C}_t^T (\boldsymbol{C}_t \bar{\Sigma}_t \boldsymbol{C}_t^T + \boldsymbol{Q})^{-1} \tag{B.45}$$
$$\Sigma_t = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{C}_t) \bar{\Sigma}_t \tag{B.46}$$

Im Gegensatz zum Standard-Kalmanfilter ist der EKF kein exakter Bayes-Filter. Wegen der notwendigen Linearisierung kann er schon

bei geringen Fehlern in der Zustandsschätzung schnell divergieren, da Abweichungen direkt zu einer falschen Annahme des Prozessmodells führen. Er ist daher insbesondere auf eine gute Initialschätzung des Erwartungswertes und der Kovarianzmatrix des Zustandes angewiesen. Weiterhin stellt durch die nichtlinearen Transformationen die reale Zustandsverteilung keine Gauss-Verteilung mehr dar. Mit steigender Abweichung zwischen dem realen Prozessmodell und der Linearisierung (im jeweiligen Zustand) wächst auch die Ungenauigkeit der Zustandsschätzung durch eine Gauss-förmige Funktion.

B.3 Partikelfilter

Der Partikelfilter stellt eine nicht-parametrische Form des Bayes-Filters dar, d.h. die geschätzte Verteilung wird (im Gegensatz z.B. zum Kalmanfilter) nicht durch eine parametrische Funktion abgebildet, sondern in diesem Fall durch eine Menge diskreter Samples, die Partikel genannt werden. Bei dieser Repräsentation steht jedes einzelne Sample für einen Punkt im Zustandsraum und bildet eine diskrete Hypothese, die Gesamtmenge der Samples ist nach der Zustands-Wahrscheinlichkeit im Zustandsraum verteilt. Die methodischen Ursprünge des Partikelfilters gehen zurück auf Monte-Carlo-Methoden [Metropolis und Ulam, 1949] sowie die Technik des Sampling Importance Resampling [Rubin, 1987], [Smith und Gelfand, 1992].

Der Partikelfilter ist durch die Darstellung einzelner, diskreter Hypothesen in der Lage, nahezu beliebige Wahrscheinlichkeitsverteilungen für den Zustand zu schätzen sowie beliebige Prozess- und Beobachtungsmodelle zu berücksichtigen. Insbesondere wegen der Möglichkeit zur Repräsentation multimodaler Verteilungen ist er für viele reale Probleme dem Kalmanfilter vorzuziehen.

Diese Vorteile werden allerdings mit u.U. sehr hohem Rechenaufwand erkauft. Die Anzahl der zur Verteilungs-Repräsentation genutzten Partikel M bestimmt sowohl die Genauigkeit der Schätzung als auch den Bedarf an Rechenzeit und Speicherplatz. Theoretisch ist eine exakte Schätzung des Zustandes nur mit einer unbegrenzten Zahl an Partikeln ($M \rightarrow \infty$) gewährleistet. In realen Implementierungen ist dies selbstverständlich nicht möglich, tatsächlich ist aber bei ausreichend großer Partikelzahl der verbleibende Schätzfehler häufig vernachlässigbar. Die

Anzahl der benötigten Partikel ist von dem jeweiligen Prozess abhängig und richtet sich nach der Komplexität der zu approximierenden Zustandsverteilung und des Prozess- und Beobachtungsmodells. Die Anzahl kann konstant oder adaptiv (nach dem aktuellen Zustand oder den zur Verfügung stehenden Rechenresourcen) gewählt werden.

Der Algorithmus für den Partikelfilter lässt sich leichter aus der Gleichung für die allgemeine Bayes-Schätzung B.12 herleiten, wenn man statt Schätzung des Zustandes $p(x_t|o_{1:t-1}, u_{1:t})$ die Schätzung des Verbundes aus aktuellem und vorherigem Zustand x_t und x_{t-1} betrachtet. Der Partikelfilter kann auf triviale Weise dafür angepasst werden, indem jedes Partikel neben der aktuellen Hypothese stets den eigenen geschätzten Zustand aus dem vorherigen Zeitschritt (vor dem letzten Update) speichert.

$$p(x_t, x_{t-1}|o_{1:t}, u_{1:t}) = p(x_t, x_{t-1}|o_t, o_{1:t-1}, u_{1:t}) \tag{B.47}$$

$$= \frac{p(o_t|x_t, x_{t-1}, o_{1:t-1}, u_{1:t}) \cdot p(x_t, x_{t-1}|o_{1:t-1}, u_{1:t})}{p(o_t|o_{1:t-1}, u_{1:t})} \tag{B.48}$$

$$= \frac{p(o_t|x_t) \cdot p(x_t, x_{t-1}|o_{1:t-1}, u_{1:t})}{p(o_t)} \tag{B.49}$$

$$p(x_t, x_{t-1}|o_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, o_{1:t-1}, u_{1:t}) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t}) \tag{B.50}$$

$$= p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1}) \tag{B.51}$$

$$p(x_t, x_{t-1}|o_{1:t}, u_{1:t}) = \frac{p(o_t|x_t) \cdot p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1})}{p(o_t)} \tag{B.52}$$

Auf diese Weise entfällt die Notwendigkeit zur Integration über den vorherigen Zustand x_{t-1}. Für eine Partikel-basierte Repräsentation gilt: Wenn die Partikel im Raum (x_t, x_{t-1}) nach der Wahrscheinlichkeit $p(x_t, x_{t-1}|o_{1:t}, u_{1:t})$ verteilt sind, so stellt deren Randverteilung in x_t die Wahrscheinlichkeitsverteilung $p(x_t|o_{1:t}, u_{1:t})$ dar.

Damit kann das aktualisierte Partikelset X_t jeweils entsprechend Gl. B.52 rekursiv aus dem vorherigen Partikelset X_{t-1} berechnet werden,

Eingaben

X_{t-1} // *Partikelset für a-priori Belief*

Initialisierung

$\bar{X}_t$, $X_t \leftarrow \emptyset$ // Proposal-*Partikelset* + *a-posteriori Partikelset*

Algorithmus

für $m = 1 : M$ // *für jedes Partikel* $x_{t-1}^m \in X_{t-1}$
  sample $\bar{x}_t^m$ zufällig aus $p(x_t|x_{t-1}^m, u_t)$ // *erzeuge Proposal-Verteilung*
  // *nach* $p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1})$
  $w_t^m = p(o_t|\bar{x}_t^m)$ // *wichte Verteilung nach Beobachtungsmodell*
  $\bar{X}_t = \bar{X}_t + \{\bar{x}_t^m, w_t^m\}$
für $m = 1 : M$ // *für jedes Partikel* $\bar{x}_t^m \in \bar{X}_t$
  sample i aus w_t^i // *proportional zur Gewichtsverteilung*
  $X_t = X_t + x_t^i$ // *Resampling platziert Partikel nach* $p(x_t|o_{1:t}, u_{1:t})$

Rückgabe

X_t // *neues Partikelset für a-posteriori Belief*

Abbildung B.3: *Pseudocode für das rekursive Update des Partikelfilters.*

indem zunächst für jedes Partikel aus dem Prozessmodell gesampelt wird ($p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|o_{1:t-1}, u_{1:t-1})$) und anschließend der gewonnen Hypothese ein Gewicht entsprechend der Beobachtungswahrscheinlichkeit $p(o_t|x_t)$ zugewiesen wird. Das *gewichtete* Partikel-Set beschreibt bereits die aktualisierte Schätzung. Um die Partikel möglichst optimal zu verteilen, so dass viele Partikel in Bereichen mit hoher Zustandswahrscheinlichkeit positioniert werden, wird jedoch (meist) noch ein Resampling-Schritt angefügt. Dieser soll die durch die Partikel (und deren Gewichte) geschätzte Verteilung nicht verändern, sondern lediglich die Partikel entsprechend der geschätzten Verteilung neu anordnen (und dabei die Gewichte zurücksetzen). Der gesamte Ablauf eines Update-Schrittes ist als Pseudocode in Abb. B.3 dargestellt.

Anhang C

Visuelle Tiefenschätzung

Bei der Beobachtung einer Szene mit einer Kamera wird ein Abbild der Welt auf dem optischen Sensor der Kamera erzeugt. Die ursprünglich dreidimensionale reale Welt wird dabei auf die zweidimensionale Abbildungsebene projiziert. Die Tiefe der Szene (der Abstand der abgebildeten Objekte zur Aufnahmeposition) geht bei dieser Projektion verloren und kann aus einem einzelnen Abbild nicht wiederhergestellt werden. Das menschliche Sehsystem ist zwar (ständig und unbewusst) in der Lage, unter starker Einbeziehung von Kontextwissen und Erwartungen auch zweidimensionale Abbildungen als räumliche Objekte zu interpretieren, für technische Systeme stellt dies jedoch eine große Herausforderung dar. Gegenüber vielen entfernungsmessenden Sensoren haben Kameras allerdings den Vorteil eines deutlich größeren Erfassungsbereiches (abgesehen von aufwändigen und meist langsameren 3D-Scanner-Anordnungen). Es ist daher wünschenswert, auch die Kamera-Informationen geeignet auswerten und nutzen zu können.

In dieser Arbeit wurden mehrfach visuelle Verfahren zur Szenenrekonstruktion benutzt (SLAM, Abschnitte 4.4.1, 4.4.2, Hindernisdetektion für lokale Navigation, Abschnitt 7.1). Es handelt sich dabei um Verfahren, die die Tiefe und damit die Koordinaten bestimmter Punkte im Raum schätzen, indem die Informationen aus mehreren Kamera-Aufnahmen der selben Szene integriert werden. Zwar sind aus der Literatur auch Verfahren bekannt, die aus einem einzelnen Bild mittels Heuristiken und Einbeziehung von Kontextwissen die Tiefen der dargestellten Objekte schätzen können, diese sind jedoch im Allgemeinen weniger genau und robust und wurden daher hier nicht berücksichtigt.

Für die Aufnahme der unterschiedlichen Abbildungen der Szene existieren zwei prinzipiell verschiedene Möglichkeiten: Beim „klassischen“ Stereo-Verfahren werden zwei oder mehr Kameras benutzt, die gleichzeitig die Szene aus unterschiedlichen Aufnahmepositionen beobachten. Typischerweise werden die Kameras zunächst untereinander kalibriert, um deren relative Position zu bestimmen. Diese wird anschließend nicht mehr verändert, auch wenn die gesamte Kamera-Anordnung sich frei bewegen kann. Die Tiefenschätzung erfolgt damit jeweils aus mehreren gleichzeitig aufgenommenen Einzelbildern, deren relative Beobachterposition gegeben ist.

Im Gegensatz dazu wird bei Depth-from-Motion eine einzelne Kamera durch die Szene bewegt, um in zeitlicher Abfolge Bilder von unterschiedlichen Positionen aufzunehmen. Zum Problem der unbekannten Tiefe aller sichtbaren Objekte kommt hier die unbekannte relative Aufnahmeposition der einzelnen Bilder hinzu. Während bei Stereo-Aufnahmen mit zwei Kameras bereits aus einem einzelnen Bildpaar eine Tiefenschätzung robust und genau funktioniert (Einschränkungen werden im nachfolgenden Abschnitt betrachtet), sind bei unbekannter Position deutlich mehr Aufnahmen notwendig, um eine zuverlässige Rekonstruktion zu ermöglichen. Das Problem kann etwas vereinfacht werden, wenn eine Positionsschätzung für die Kamera verfügbar ist. Dies ist zum Beispiel bei einer auf einem mobilen Roboter montierten Kamera der Fall, wenn über die Odometrie die Roboter- und damit die Kamerabewegung (mit einer zu berücksichtigenden Ungenauigkeit) gemessen wird.

Bei beiden Verfahren ist für die Bestimmung der Tiefe zunächst eine Korrespondenzsuche notwendig: Für einen Bildpunkt in einem Bild muss dessen Position im entsprechenden Bild (ggf. in allen Bildern einer Sequenz) bestimmt werden. Dabei wird das Konzept der Epipolarlinien benutzt (Abb. C.1): Seien I_1 und I_2 zwei Bilder der selben Szene von unterschiedlichen Aufnahme-Positionen. Dann wird ein Objekt $\boldsymbol{X}$, welches in $I1$ an einem Bildpunkt $\boldsymbol{x}_1$ liegt, in I_2 auf der Projektion des Blickstrahls durch diesen Bildpunkt liegen. Dieser Strahl wird auf I_2 als Epipolarlinie $\boldsymbol{l}_2$ projiziert, die Position entlang der Linie hängt von der (zunächst unbekannten) Tiefe des Objektes ab. Bei Kenntnis der relativen Aufnahmepositionen der Bilder I_1 und I_2 kann jedem Pixel $\boldsymbol{x}_1$ des einen Bildes eine Epipolarline $\boldsymbol{l}_2$ im anderen Bild zugeordnet werden. Die Korrespondenzsuche kann damit auf diese Epipolarlinie beschränkt

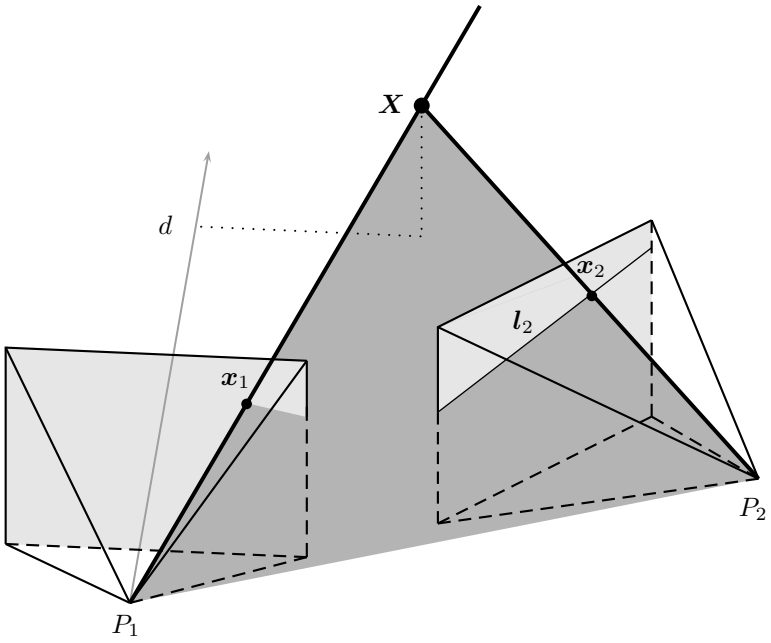

Abbildung C.1: *Epipolargeometrie an zwei Kameras: Der Bildpunkt $\boldsymbol{x}_1$ im Kamerabild I_1 wird im Bild I_2 auf die Epipolarlinie $\boldsymbol{l}_2$ abgebildet.*

werden. Durch Zuordnung der genauen Korrespondenz-Position $\boldsymbol{x}_1$ auf der Epipolarlinie wird die Tiefe des abgebildeten Objektes bestimmt.

Es ist darauf zu achten, dass eine Tiefenbestimmung nur für die Bildpunkte bzw. -regionen möglich ist, die tatsächlich in mehreren Bildern sichtbar sind. Die Kamera-Positionen dürfen sich daher nur in einem bestimmten Bereich untereinander unterscheiden. Andererseits verändert sich bei einer größeren Änderung des Beobachterwinkels zu einem bestimmten Szenenpunkt dessen Position im Bild stärker, wodurch die Bestimmung der Tiefe mit höherer Genauigkeit und Robustheit (u.a. wegen der beschränkten Auflösung durch das diskrete Pixelraster, aber auch Bildrauschen) möglich ist. Der Abstand der Kamerapositionen bestimmt damit die minimale und maximale detektierbare Tiefe bzw. umgekehrt ist die notwendige und zulässige Bewegung der Kamera vom Abstand der beobachteten Objekte zur Kamera abhängig.

In diesem Kapitel werden im Anschluss zwei konkrete Verfahren zur visuellen Tiefenschätzung vorgestellt. Das erste Verfahren benutzt eine Stereo-Kamera-Anordnung. Als Besonderheit wird bei dieser Kamera des Herstellers Videre Design [Videre Design, 2008] die Berechnung

der Tiefeninformation aus zwei kalibrierten Einzelbildern direkt in der Kamera-Hardware mit Hilfe eines klassischen korrelationsbasierten Ansatzes durchgeführt und steht somit unmittelbar für weitere Verarbeitung zur Verfügung.

Neben dem höheren Preis für zwei Kameras (und eventuelle Hardware zur Synchronisation/Verarbeitung) stellt u.U. die notwendige Stereo-Kalibrierung ein Problem für Anwendungen mit mobilen Systemen dar. Diese erfordert eine sehr genaue, unveränderliche Fixierung sowohl der Bildsensoren als auch der Abbildungsoptik, welche aufgrund von Vibrationen über eine unbegrenzte Zeitdauer oft nur mit hohem Aufwand zu gewährleisten ist. Als Alternative zum Stereo-Kamera-Ansatz wurde daher ein hybrides monokulares Depth-from-Motion-Verfahren entwickelt, welches eine initiale Multi-Baseline-Tiefenschätzung mit einer iterativen Verbesserung durch Kalmanfilter kombiniert [Einhorn et al., 2007a].

C.1 Tiefenschätzung mit Stereo-Kamera

Bei der Stereo-Kamera sind zwei einzelne Kameras mit gleicher Brennweite (jeweils identische Bildsensoren und Abbildungsoptiken) koplanar in einem Gehäuse untergebracht. Die beiden Bildebenen liegen also in einer gemeinsamen Ebene, außerdem besteht zwischen den Sensoren lediglich eine Verschiebung entlang der horizontalen Bild-Achse (keine vertikale Verschiebung oder Rotation). Aufgrund dieser Anordnung befinden sich (bei idealisierter Abbildung) korrespondierende Punkte stets auf einer gemeinsamen horizontalen Linie (gleiche Pixelzeile) in beiden Bildern. Die Verschiebung eines Punktes zwischen den Bildern wird als Disparität d bezeichnet. Diese ist umgekehrt proportional zu seiner Tiefe z, und kann mit Hilfe des Basisabstandes b und der Brennweite f berechnet werden (Abb. C.2):

$$\frac{u_l}{x} = \frac{f}{z} \tag{C.1}$$

$$\frac{u_r}{x+b} = \frac{f}{z} \tag{C.2}$$

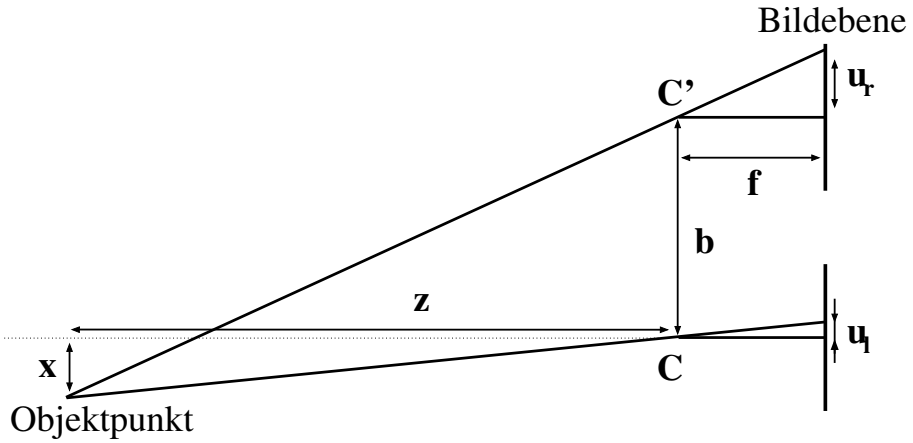

Abbildung C.2: *Durch den seitlichen Versatz der Kameras wird der selbe Objektpunkt an unterschiedlichen Stellen auf dem Bildsensor abgebildet. Die Abbildung zeigt zwei idealisierte (Lochkamera-Modell), parallel ausgerichtete Kameras mit den Kamerazentren C und C'. Die Disparität ist bei konstanter Brennweite f und Basisabstand b umgekehrt proportional zum Objektabstand z (Gl. C.3).*

$$\begin{aligned} d &= u_r - u_l \\ &= (x+b)\frac{f}{z} - x\frac{f}{z} \\ &= \frac{b \cdot f}{z} \end{aligned} \tag{C.3}$$

Vor dem Einsatz muss eine Kalibrierung der Kamera erfolgen. Diese erfüllt zwei Aufgaben: Einerseits wird für jede Kamera eine Korrekturtransformation berechnet, die die Verzerrungen der realen Optik ausgleicht und das aufgenommene Bild korrigiert, so dass ein Bild einer (virtuellen) Lochkamera entsteht. Dabei wird u.a. auch die Brennweite der jeweiligen Kamera bestimmt. Weiterhin wird bei der Kalibrierung die exakte Transformation zwischen den Kameras ermittelt. Dieser Vorgang wird auch Rektifizierung genannt. Die Kalibrierung erfolgt durch Aufnahme geeigneter Testmuster aus unterschiedlichen Ansichten und ist gültig, so lange die Einstellungen der Kameras und Optik unverändert bleiben.

Die Korrespondenzsuche wird durch die ausschließlich horizontale Verschiebung im Bild stark vereinfacht: Die Epipolarlinien verlaufen jeweils horizontal entlang der selben Zeile im Bild (Epipolar-constraint).

Abbildung C.3: *Linkes und rechtes Bild eines Stereo-Paares. Die Beschneidung am rechten Rand des rechten Bildes entsteht bei der Rektifizierung, da aufgrund von Abbildungsfehlern der Kameraoptik der Bildinhalt leicht verschoben werden muss. Die Markierungslinien verdeutlichen, dass die einander entsprechenden Bildinhalte jeweils auf der selben Zeile liegen. In horizontaler Richtung sind diese jedoch verschoben, wobei der Versatz um so geringer ist, je weiter die Objekte von der Kamera entfernt sind.*

Da einzelne Pixel zu unspezifisch für eine zuverlässige Zuordnung sind, wird eine lokale Umgebung jedes Punktes (Korrelationsfenster) für die Korrespondenzsuche benutzt. Innerhalb eines festgelegten Disparitätsbereiches wird dazu die Übereinstimmung des Korrelationsfensters geprüft, die tatsächliche Disparität wird durch die Stelle mit der maximalen Übereinstimmung gefunden.

$$d(u,v) = \underset{d\in[d_{min},d_{max}]}{argmin} SAD(K_l(u,v), K_r(u+d,v)) \tag{C.4}$$

$$= \underset{d\in[d_{min},d_{max}]}{argmin} \sum_{\substack{u'\in u\pm s \\ v'\in v\pm s}} |I_l(u',v') - I_r(u'+d,v')| \tag{C.5}$$

Allerdings wird dieses Optimum nur dann berücksichtigt, wenn es sich deutlich auszeichnet. Die Suche nach der optimalen Verschiebung wird zudem in beide Richtungen (jeweils vom linken zum rechten sowie vom rechten zum linken Bild) durchgeführt. Nur bei Übereinstimmung in beiden Richtungen, also einer 1:1 Zuordnung, ist die Disparität gültig (Left-right-constraint). Aus diesen beiden Bedingungen ergeben sich wesentliche Eigenschaften des entstehenden Disparitätsbildes: Eine Korrespondenz zwischen Bildpunkten kann nur dort gefunden werden, wo

Abbildung C.4: *Original-Disparität (links) und Disparität nach der Maskierung mit einem Kantenfilter (rechts): Die Disparitätswerte sind als Farbverlauf gelb (nah) - orange - rot - schwarz (fern) kodiert. Durch den Einsatz des Korrelationsfensters bei der Suche nach korrespondierenden Kanten werden die zugeordneten Punkte in horizontaler Richtung „verschliffen“, die Disparität erscheint dadurch flächiger, ist aber ungenauer. Dies kann zum Beispiel beim Einsatz für die Kollisionsvermeidung eines mobilen Roboters problematisch werden, da Hindernisse vergrößert werden und somit der befahrbare Freiraum (z.B. eine Türdurchfahrt) enger erscheint. Durch die Maskierung werden Disparitätswerte im Idealfall nur an den Pixeln berücksichtigt, wo entsprechende Bildinhalte zugeordnet werden können, die unkorrekte Verbreiterung der Kanten wird dadurch aufgehoben. Allerdings sind auch dieser Verbesserung Grenzen gesetzt: fehlerhafte Disparitäten bleiben immer dort bestehen, wo verschiedene Kanten unterschiedlicher Tiefe im Korrelationsfenster zusammentreffen oder dicht nebeneinander liegen, z.B. an der Flasche auf dem Boden, aber auch an der Stuhllehne vor dem dahinterliegenden Türrahmen (vgl. Originalbild C.3).*

deutliche Grauwertänderungen in horizontaler Richtung (z.B. vertikale oder diagonale Kanten) vorhanden sind. Diese dürfen sich allerdings nicht in zu kurzer Folge periodisch wiederholen, da die Zuordnung dann uneindeutig sein kann. Bei relativ homogenen Oberflächen werden die Disparitäten nur an Objektkanten gefunden (vgl. Wände in Abb. C.4). Nur für stärker strukturierte Oberflächen entsteht ein dichtes Disparitätsbild. Die Nutzung des Korrelationsfensters zur Korrespondenzsuche verursacht außerdem einen weiteren Störeffekt: Kanten zwischen homogenen Flächen erscheinen im Disparitätsbild gegenüber dem Originalbild verbreitert, da das Korrelationsfenster wie ein Faltungskern auf dem (virtuellen) Kantenbild wirkt (Abb. C.4). Diese negative Auswirkung kann durch eine nachträgliche Maskierung mit dem Kantenbild unterdrückt werden, dadurch werden Disparitäten nur tatsächlich auf ausgeprägten Kanten berücksichtigt. Allerdings ist dafür zusätzlich

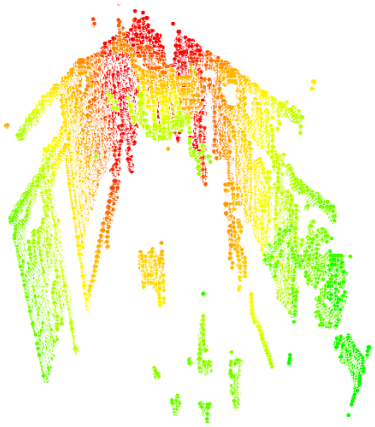

Abbildung C.5: *In der Darstellung als 3D-Punkte im Raum wird die detektierte Tiefe deutlich. Die räumliche Tiefe (Abstand zur Kamera) der einzelnen Punkte ist auch hier farblich kodiert: grün (nah) - gelb - orange - rot (fern). Die Betrachterposition für diese Darstellung befindet sich über der realen Aufnahmeposition der Kamera. ImVordergrund sind Teile des Stuhl und der Flasche im Raum stehend sichtbar, zur Linken befindet sich die Tür. Der Betrachter blickt außerdem von oben auf die Deckenbeleuchtung herab.*

die Berechnung dieses Kantenbildes mittels eines geeigneten Operators (z.B. Canny-Edge-Detektor, [Canny, 1986]) notwendig, was weiteren Rechenaufwand bedeutet.

Aus den Disparitätswerten kann für jeden bekannten Pixel direkt aus Gl. C.3 dessen Tiefe z berechnet werden. Durch die zuvor erfolgte Kalibrierung kann weiterhin jeder Pixel korrekt in einen von der Kamera ausgehenden Raumstrahl zurücktransformiert werden. Die Kombination von Strahl und Tiefe ergibt die Koordinaten des Punktes im Raum, relativ zur Kamera. Im Endergebnis entsteht aus den beiden Kamerabildern eine 3D-Punktwolke im Kamera-Koordinatensystem, bei mobilen Anwendungen muss diese normalerweise in ein globales Koordinatensystem transformiert werden (Abb. C.5).

C.2 Monokulare Tiefenschätzung mittels Depth-from-Motion

Bei der monokularen Tiefenschätzung werden statt zweier gleichzeitig aufgenommener Bilder mehrere nacheinander aufgenommene Bilder der selben, bewegten Kamera genutzt. Das hier vorgestellte Verfahren wurde als Kombination verschiedener bekannter Ansätze in einer Diplom-

arbeit am FG NIKR [Einhorn, 2007] entwickelt und bereits in [Einhorn et al., 2007a] und [Einhorn et al., 2007b] publiziert.

C.2.1 Grundlagen: Homogene Koordinaten

Homogene Koordinaten sind eine Erweiterung des euklidischen Vektorraumes, die gegenüber diesem den Vorteil haben, dass sich in ihnen alle geometrischen Transformationen, insbesondere auch Translation und Projektion, durch lineare Gleichungssysteme (also durch Multiplikation des Punktvektors mit einer Transformationsmatrix) darstellen lassen. Sie eignen sich daher besser zur Beschreibung der Abbildungen zwischen 3D-Szenen und Kamera-Bildern als herkömmliche euklidische Koordinaten.

Ein karthesisches Koordinatensystem wird aus dem reellwertigen euklidischen Raum $\mathbb{R}^n$ durch Hinzufügen einer Dimension in den homogenen Koordinatenraum überführt. Die Menge der Ursprungsgeraden des entstehenden $\mathbb{R}^{n+1}$ (ohne den Nullpunkt) wird als projektiver Raum $\mathbb{P}^n$ bezeichnet. Es gilt damit $\mathbb{R}^n \subset \mathbb{P}^n \subseteq \mathbb{R}^{n+1} \setminus \{0\}$. Diese Beziehung ist am Beispiel von $\mathbb{R}^2$ und $\mathbb{P}^2$ in Abb. C.6 dargestellt.

Zum dreidimensionalen euklidischen Raum $\mathbb{R}^3$ gehört demzufolge der projektive Raum $\mathbb{P}^3$, welcher in den vierdimensionalen euklidischen Raum $\mathbb{R}^4$ eingebettet ist. Im folgenden werden zur Unterscheidung homogene Koordinaten in $\mathbb{P}^3$ als $\tilde{\boldsymbol{X}}$ bezeichnet, euklidische Koordinaten des $\mathbb{R}^3$ hingegen als $\boldsymbol{X}$. $\tilde{\boldsymbol{x}}$ und $\boldsymbol{x}$ bezeichnen entsprechend homogene und euklidische Koordinaten im zweidimensionalen (Bild-)Raum $\mathbb{P}^2$ bzw. $\mathbb{R}^2$.

Ein Punkt $\boldsymbol{X} = (x, y, z)^T \in \mathbb{R}^3$ kann durch folgende Abbildung in einen Punkt $\tilde{\boldsymbol{X}} = (x, y, z, w)^T \in \mathbb{P}^3$ überführt werden:

$$\mathcal{H} : (x, y, z)^T \mapsto (x, y, z, 1)^T \tag{C.6}$$

Der euklidische Raum bildet also einen Unterraum, der durch die Ebene $w = 1$ definiert wird. Abbildung C.6 verdeutlicht diesen Zusammenhang (wegen der einfacheren Darstellung allerdings für $\mathbb{P}^2$ und

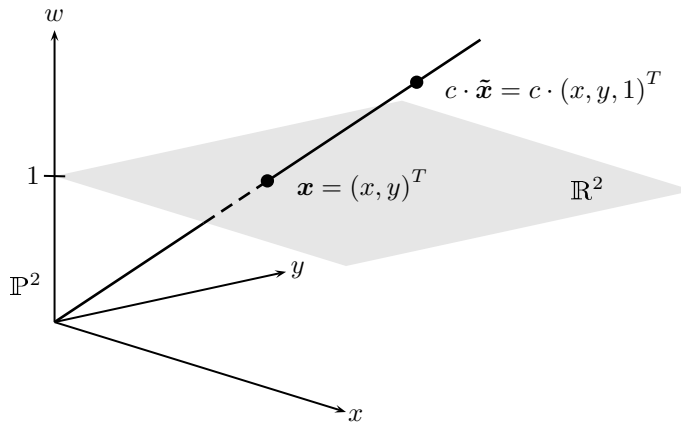

Abbildung C.6: *Der projektive Raum $\mathbb{P}^2$ wird durch die Menge der Ursprungsgeraden im $\mathbb{R}^3$ gebildet. $\mathbb{R}^2$ stellt wiederum einen Unterraum von $\mathbb{P}^2$ dar. Da alle Punkte eines Strahls auf den Schnittpunkt mit der Ebene $w = 1$ abgebildet werden, hat ein Punkt $\boldsymbol{X} \in \mathbb{R}^n$ unendlich viele Entsprechungen $\tilde{\boldsymbol{X}} \in \mathbb{P}^n$.*

$\mathbb{R}^2$). Umgekehrt können Punkte $\tilde{\boldsymbol{X}} = (x, y, z, w)^T \in \mathbb{P}^3$ auf Punkte $\boldsymbol{X} = (x, y, z)^T \in \mathbb{R}^3$ abgebildet werden:

$$\mathcal{H}^{-1} : (x, y, z, w)^T \mapsto (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T \tag{C.7}$$

Für Punkte mit $w = 0$ ist diese Abbildung nicht definiert. Solche Punkte besitzen keine Entsprechung im $\mathbb{R}^3$. Abgesehen von dieser Ausnahme stellen alle Vielfachen eines homogenen Punktes $c \cdot \tilde{\boldsymbol{X}}$ $(c \in \mathbb{R} \setminus \{0\})$ offensichtlich Repräsentanten des selben euklidischen Punktes dar:

$$\begin{aligned}
\mathcal{H}^{-1}\left(c \cdot \tilde{\boldsymbol{X}}\right) &= \mathcal{H}^{-1}(cx, cy, cz, cw)^T \\
&= (\frac{c \cdot x}{c \cdot w}, \frac{c \cdot y}{c \cdot w}, \frac{c \cdot z}{c \cdot w})^T \\
&= (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^T \\
&= \boldsymbol{X}
\end{aligned}$$

Ein Punkt $\boldsymbol{X} \in \mathbb{R}^3$ besitzt folglich unendlich viele homogene Repräsentanten $c \cdot \tilde{\boldsymbol{X}}$ $(c \in \mathbb{R} \setminus \{0\})$, die im projektivem Raum $\mathbb{P}^3$ auf einer gemeinsamen Ursprungsgeraden liegen (siehe auch Abb. C.6).

C.2.2 Epipolargeometrie

Die relativen Positionen zwischen den Bildern und damit die Epipolarlinien sind durch die bewegte Kamera nicht mehr von vornherein bekannt. Da die Kamera auf einem mobilen Roboter montiert ist, können diese jedoch durch die Messung der Eigenbewegung mittels Odometrie abgeschätzt werden. Es werden nur jeweils wenige aufeinander folgende Bilder verrechnet, so dass Fehler, welche aus der Ungenauigkeit der Odometrie resultieren, beschränkt sind. Durch die Kombination der Odometrie und der relativen Lage der Kamera zum Roboter kann zu jedem Bild I die entsprechende Kamera-Projektionsmatrix $\boldsymbol{P}$ (Beschreibung der Abbildung im Welt-Koordinatensystem) bestimmt werden [Hartley und Zisserman, 2004]:

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{K}\boldsymbol{R}[\boldsymbol{I}_{3x3}| - \boldsymbol{c}] \\ &= \boldsymbol{K}\boldsymbol{A}\boldsymbol{R}_{cam}\boldsymbol{R}_{robot} \begin{pmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \end{pmatrix} \end{aligned} \tag{C.8}$$

$\boldsymbol{K}$ ist dabei die konstante Kamera-Kalibrierungsmatrix, die die intrinsischen Kamera-Parameter enthält und einmal (normalerweise mittels bildbasierter Kalibrierung) bestimmt werden muss. $\boldsymbol{c} = (x, y, z)^T$ ist die Position des Kamerazentrums in Weltkoordinaten. $\boldsymbol{R}_{robot}$ beschreibt die Rotation zwischen Welt-Koordinatensystem und Roboter-Koordinatensystem (Blickrichtung des Roboters), während $\boldsymbol{R}_{cam}$ die Rotation vom Roboter-Koordinatensystem ins Kamera-Koordinatensystem (relative Blickrichtung der Kamera) darstellt. A ist eine Ersetzungsmatrix, die lediglich eine Vertauschung der Koordinatenachsen bewirkt, um der unterschiedlichen Definition der einzelnen Dimensionen in Kamera- und Weltkoordinaten Rechnung zu tragen (siehe Abb. C.7).

$$\boldsymbol{A} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix}$$

Für einen Roboter, der sich auf der Bodenebene bewegt und eine Kamera, die in Fahrtrichtung blickend, jedoch gegenüber der Horizontalen geneigt am Roboter angebracht ist, ergeben sich die folgenden Rotati-

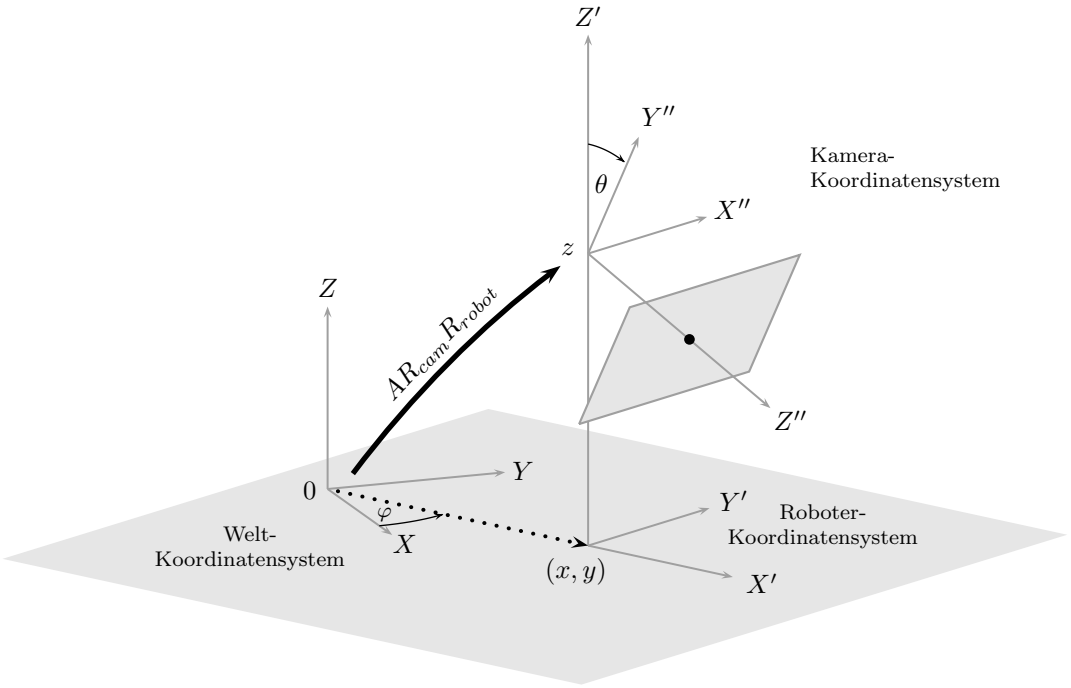

Abbildung C.7: *Veranschaulichung der Transformationen zwischen Kamera-, Roboter- und Weltkoordinaten.*

onsmatrizen (mit der aktuellen Roboter-Blickrichtung φ und der Kameraneigung θ):

$$\boldsymbol{R}_{cam} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\boldsymbol{R}_{robot} = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Die so ermittelte Projektionsmatrix $\boldsymbol{P}$ definiert die Abbildung des 3D-Punktes $\tilde{\boldsymbol{X}}$ den Bildpunkt $\tilde{\boldsymbol{x}}$:

$$\tilde{\boldsymbol{x}} = \boldsymbol{P}\tilde{\boldsymbol{X}} \tag{C.9}$$

Die Rücktransformation ist hingegen nicht eindeutig, da bei der Projektion die Tiefeninformation verloren geht. Dem Punkt $\tilde{x}$ entspricht daher ein Strahl mit Ausgangspunkt im Kamerazentrum $\tilde{c}$, der rückprojizierte

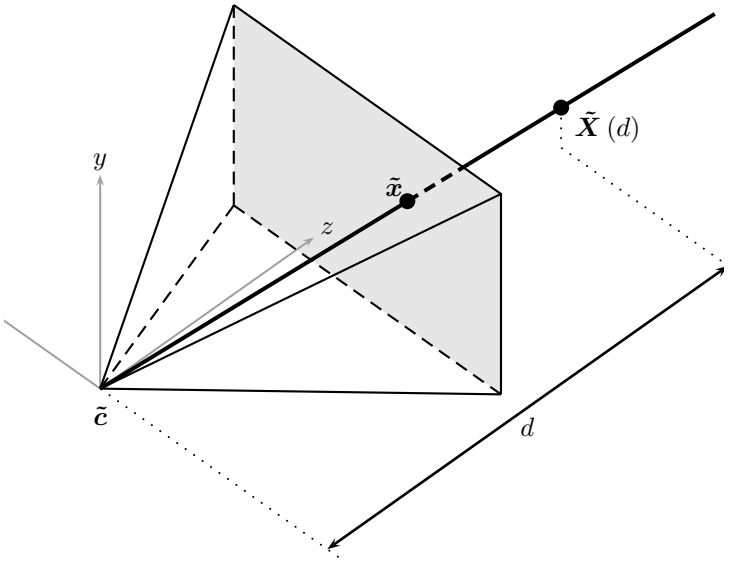

Abbildung C.8: *Aus dem Bild-Punkt, welcher durch Projektion auf die Kamera-Ebene entsteht, kann der originale 3D-Punkt nicht eindeutig rekonstruiert werden, da die Tiefeninformation bei der Projektion nicht erhalten bleibt. Die Rücktransformation ergibt daher einen Strahl im 3D-Raum, jeder Punkt auf diesem Strahl kann dem Bild-Punkt zugeordnet werden.*

Punkt $\tilde{X}$ ist abhängig von der zu bestimmenden Tiefe d bzw. der inversen Tiefe λ (Abb. C.8). λ wird in der Literatur auch als Disparität bezeichnet. Im Gegensatz zum vorhergehenden Stereo-Abschnitt ist damit jedoch nicht der Versatz zwischen abgebildeten Punkten auf der Bildebene gemeint, sondern das Inverse der geometrischen Tiefe im 3D-Raum $\lambda = d^{-1}$.

$$\begin{aligned} \tilde{\boldsymbol{X}}(d) &= d\boldsymbol{P}^{+}\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{c}} \\ \tilde{\boldsymbol{X}}(\lambda) &= \boldsymbol{P}^{+}\tilde{\boldsymbol{x}} + \lambda\tilde{\boldsymbol{c}} \end{aligned} \tag{C.10}$$

$vecP^{+}$ bezeichnet hier die Pseudoinverse der Projektionsmatrix $\boldsymbol{P}$. Für die Abbildung eines Punktes $\tilde{\boldsymbol{x}}_i$ mit zugehörigem (unbekanntem) λ in Bild I_i auf das Bild I_j ergibt sich somit (durch Einsetzen von Gl. C.10 in Gl. C.9) folgende Epipolarlinie (vgl. Abb. C.1):

$$\begin{aligned}\tilde{\boldsymbol{x}}_j(\lambda) &= \boldsymbol{P}_j(\boldsymbol{P}_i^{+}\tilde{\boldsymbol{x}}_i + \lambda\tilde{\boldsymbol{c}}_i) \\ &= \boldsymbol{P}_j\boldsymbol{P}_i^{+}\tilde{\boldsymbol{x}}_i + \lambda\boldsymbol{P}_j\tilde{\boldsymbol{c}}_i \end{aligned} \tag{C.11}$$

C.2.3 Featuredetektion und -tracking

Um einerseits eine robuste Zuordnung der Bildpunkte über mehrere Bilder einer Sequenz hinweg zu ermöglichen und gleichzeitig die benötigte Rechenzeit zu beschränken, wird im Gegensatz zum zuvor vorgestellten Stereo-Ansatz nicht versucht, Korrespondenzen für alle Bildpunkte des Referenzbildes zu finden. Stattdessen werden in jedem Bild zunächst mittels eines Merkmals-Detektors signifikante Punkte (Feature-Punkte) ausgewählt. Ziel ist es dabei, solche Bildpunkte zu detektieren, die sich in anderen Bildern ebenfalls leicht finden und möglichst eindeutig zuordnen lassen. Als sehr schneller Detektor kommt dafür der FAST-Eckendetektor [Rosten und Drummond, 2006] zum Einsatz. Es entsteht somit also im Ergebnis kein dichtes Tiefenbild, sondern es wird lediglich auf Basis dieser Punktfeature-Detektionen eine begrenzte Anzahl an 3D-Punkten geschätzt (Abb. C.11).

Die eigentliche Bestimmung der 3D-Position geschieht durch eine Kombination zweier Verfahren: Multi-Baseline-Stereo und Erweiterter Kalmanfilter (EKF) (Abb. C.9 links). Für einen Feature-Punkt $f^{(k)}$, welcher in einem Bild I_t an Position $\boldsymbol{x}_t^{(k)} = [u_t^{(k)}, v_t^{(k)}]^T$ erstmalig detektiert wird (d.h. es ist keine Zuordnung zu Feature-Punkten vorhergehender Bilder möglich) erfolgt eine initiale Positionsbestimmung durch eine Multi-Baseline-Tiefenschätzung. Dazu werden in den gespeicherten vorhergehenden Bildern $I_{t-n} \dots I_{t-1}$ (unter Nutzung der zu jedem Bild zugeordneten Kameramatrizen P_t) jeweils die entsprechenden Epipolarlinien bestimmt und entlang dieser die Übereinstimmung eines Korrelationsfensters um die Featureposition geprüft. Durch Einsetzen des Punktes $\tilde{\boldsymbol{x}}_t^{(k)}$ und einer angenommenen Disparität λ in Gl. C.11 ergibt sich direkt die resultierende Position $\boldsymbol{x}_{t'}^{(k)}(\lambda) = [u_{t'}^{(k)}(\lambda), v_{t'}^{(k)}(\lambda)]^T$, an der der Punkt $\boldsymbol{X}_t^{(k)}$ im Bild $I_{t'}$ abgebildet würde. Damit wird die optimale Disparität λ^* bestimmt, für die in der Summe über alle genutzten Bilder die Summe der absoluten Differenzen (SAD) zwischen dem Referenzbild

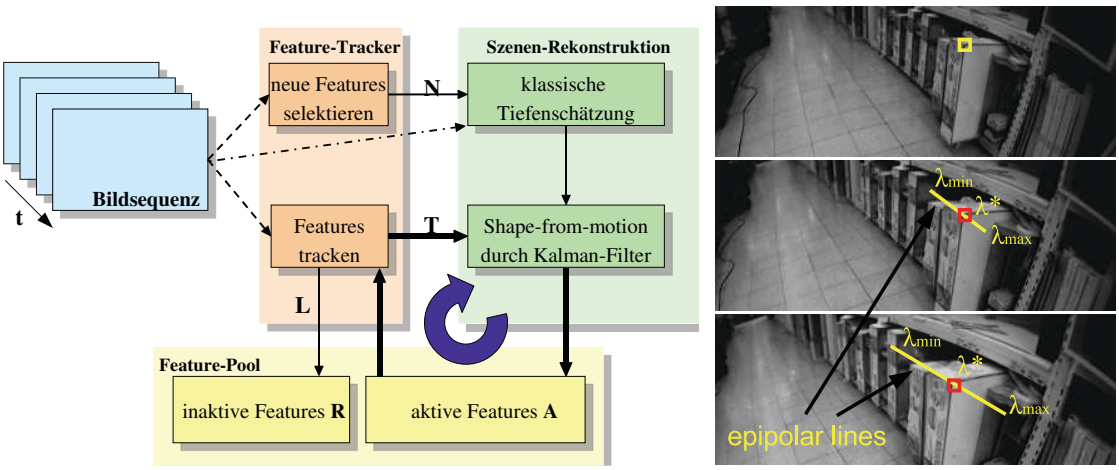

Abbildung C.9: **Links:** *Überblick über das Gesamtsystem zur monokularen Tiefenschätzung: Bei der erstmaligen Detektion eines Features im Kamerabild wird über die zurückliegenden Bilder mittels einer Multi-Baseline-Tiefenschätzung eine initiale Schätzung der 3D-Position ermittelt. Durch Tracking des Features in nachfolgenden Bildern wird diese Initialschätzung mittels eines Erweiterten Kalmanfilters verfeinert.* **Rechts:** *Beispiel der Multi-Baseline-Schätzung: Für den hier erstmals detektierten Featurepunkt wird in den 2 vorhergehenden Bildern entlang der Epipolarlinien die Übereinstimmung des Bildinhaltes geprüft. Die beste Position bestimmt die inverse Tiefe λ^* des zugehörigen 3D-Punktes.*

und allen Vergleichsbildern über das Korrelationsfenster minimiert wird (Abb. C.9 rechts).

$$\lambda^* = \underset{\lambda}{argmin} \left(\sum_{t'=t-n}^{t-1} SAD \left(I_t \left(u_t^{(k)}, v_t^{(k)} \right), I_{t'} \left((u_{t'}^{(k)}(\lambda), v_{t'}^{(k)}(\lambda) \right) \right) \right) \tag{C.12}$$

Mittels Gl. C.10 werden aus der Disprität/Tiefe die Koordinaten des Punktes berechnet, mit dieser wird ein Erweiterter Kalmanfilter initialisiert (für jeden Punkt wird ein eigener EKF verwendet.

Über Beobachtungen des selben Feature-Punktes in nachfolgenden Bildern wird mittels des EKF die geschätzte 3D-Position fortlaufend korrigiert und dadurch verbessert. Dazu muss ein einmal gefundener Punkt in allen weiteren Bildern verfolgt werden, um eine Beziehung zwischen den beobachteten Bildpunkten und geschätzten 3D-Punkten zu ermöglichen. Mittels eines Linking-Algorithmus werden detektierte Feature-Punkte in aufeinanderfolgenden Bildern einander zugeordnet.

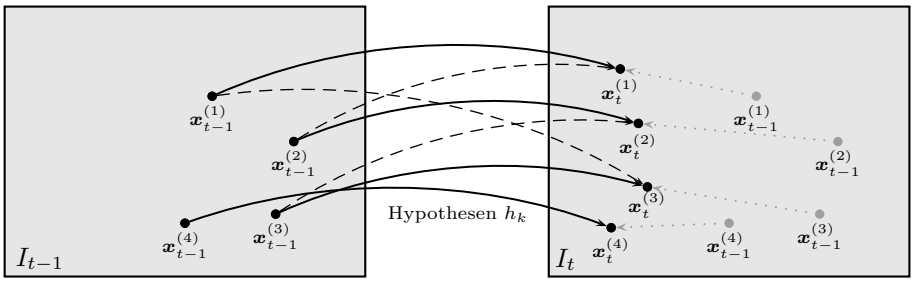

Abbildung C.10: *Die im Bild I_t gefundenen Feature-Punkte müssen den zuvor in den vorhergehenden Bildern der Sequenz gefundenen Features zugeordnet werden. Dazu werden Match-Hypothesen für Paare $(f_t^{(k)}, f_{t-1}^{(j)})$ aufgestellt und jeder Hypothese Kosten zugeordnet, die die Übereinstimmung bzgl. Bildinhalt und erwarteter Position bewerten. Ein Linking-Algorithmus wählt dann diejenigen Hypothesen aus, für die die Summe der Match-Kosten minimal wird.*

Jeder hypothetischen paarweisen Zuordnung eines Features $f_t^{(k)}$ des aktuellen Bildes zu einem zuvor detektierten Feature $f_{t-1}^{(j)}$ werden dazu Kosten zugeordnet, die die Wahrscheinlichkeit der Übereinstimmung beschreiben. Die Matchkosten für ein Featurepaar berücksichtigen

- die SAD zwischen den Korrelationsfenstern um $f_t^{(k)}$ und $f_{t-1}^{(j)}$
- den Abstand von $f_t^{(k)}$ zur durch $f_{t-1}^{(j)}$ bestimmten Epipolarlinie
- den Abstand von $f_t^{(k)}$ zum ins Bild I_t rückprojizierten Punkt $\tilde{\boldsymbol{x}}_t^{(j)}$

Zwei der drei Kostenanteile beruhen also auf der aktuellen 3D-Positionsschätzung des jeweiligen Features. Dies ist gerechtfertigt, da bereits mit der ersten Detektion eines Features und der Initialisierung des Kalmanfilters eine relativ gute Schätzung der Position des entsprechenden 3D-Punktes ermittelt wird. Die Wichtung der Kosten untereinander wurde experimentell optimiert.

Der Linking-Algorithmus versucht die Feature-Punkte des jeweils aktuellen Bildes den zuvor gefundenen Features eineindeutig zuzuordnen und dabei die entstehenden Gesamtkosten zu minimieren (Abb. C.10). Eine garantiert optimale Zuordnung aller Features ist zwar nur mit einem NP-vollständigen und damit langsamen Algorithmus möglich, doch

auch mit einem Greedy-Algorithmus, der nacheinander jeweils die bestbewertete Match-Hypothese wählt und alle konkurrierenden Hypothesen eliminiert, wird fast immer ein nahezu gleich gutes Ergebnis erzielt.

C.2.4 Erweiterter Kalmanfilter

Wie bereits erklärt wurde, wird für jeden zu schätzenden 3D-Punkt ein EKF verwendet, der dessen Zustand als Position im Raum $[x, y, z]$ schätzt. Der Initialzustand ergibt sich dabei jeweils aus der Bestimmung der Punkt-Position mittels des Multi-Baseline-Stereo-Verfahrens. Neben dem Mittelwert muss für den EKF auch die initiale Kovarianz festgelegt werden. Diese wird in der benutzten Implementierung für alle Punkte gleich und konstant gesetzt, als günstig hat sich hier der Wert

$$\Sigma_{init} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

erwiesen (in m).

Nach der Zuordnung der im aktuellen Bild gefundenen Featurepunkte zu den getrackten 3D-Punkten können die beobachteten Bild-Positionen jeweils benutzt werden, um die Schätzung der 3D-Position anzupassen. Für den EKF müssen dazu die Zustandsübergangs-Funktion $f(\boldsymbol{X}_{t-1})$ sowie die Beobachtungs-Funktion $h(\boldsymbol{X}_t)$ festgelegt werden (vgl. Abschnitt B.2). Da der Rekonstruktionsalgorithmus von einer statischen Umgebung ausgeht und die Umgebungspunkte daher ihre Positionen nicht verändern, ergeben sich folgende Zustandsübergangsmatrix $\boldsymbol{A}_t$, Steuermatrix $\boldsymbol{B}_t$ und Kovarianzmatrix des Prozessrauschens $\boldsymbol{R}$ im EKF (Abschnitt B.2):

$$f(\boldsymbol{X}_{t-1}) = \boldsymbol{X}_{t-1} \tag{C.13}$$

$$\boldsymbol{A}_t = \boldsymbol{I}_{3x3} \qquad \boldsymbol{B}_t = \boldsymbol{0}_{3x3} \qquad \boldsymbol{R} = \boldsymbol{0}_{3x3} \tag{C.14}$$

Die Abbildung des Punktes auf die Bildposition (Beobachtungsfunktion) ist in Gleichung C.9 beschrieben. Diese bezieht sich allerdings auf

homogene Koordinaten, während die Zustandsschätzung und die Beobachtung in euklidischen Koordinaten vorliegt (da unterschiedliche homogene Koordinaten zu einem euklidischen Punkt möglich sind, wäre die Schätzung sonst nicht eindeutig). Für die Beschreibung der Beobachtung wird daher der geschätzte Zustand in homogene Koordinaten transformiert und der abgebildete homogene Bildpunkt in euklidische Koordinaten zurücktransformiert:

$$h(\boldsymbol{X}_t) = \mathcal{H}^{-1}(\boldsymbol{P}_t \mathcal{H}(\boldsymbol{X}_t)) + \delta_t \tag{C.15}$$

Mit $\tilde{\boldsymbol{X}}_t = (\boldsymbol{X}^T, 1)^T$ ergibt sich durch Einsetzen von Gl. C.7

$$h(\boldsymbol{X}_t) = \left(\frac{\boldsymbol{P}_{[1]}\tilde{\boldsymbol{X}}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}}, \frac{\boldsymbol{P}_{[2]}\tilde{\boldsymbol{X}}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} \right) + \delta_t$$

$$\boldsymbol{C}_t = \begin{pmatrix} \frac{\boldsymbol{P}_{[1,1]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[1]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,1]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2}, \frac{\boldsymbol{P}_{[1,2]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[1]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,2]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2}, \frac{\boldsymbol{P}_{[1,3]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[1]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,3]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2} \\ \frac{\boldsymbol{P}_{[2,1]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[2]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,1]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2}, \frac{\boldsymbol{P}_{[2,2]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[2]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,2]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2}, \frac{\boldsymbol{P}_{[2,3]}}{\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}}} - \frac{\boldsymbol{P}_{[2]}\tilde{\boldsymbol{X}}\boldsymbol{P}_{[3,3]}}{(\boldsymbol{P}_{[3]}\tilde{\boldsymbol{X}})^2} \end{pmatrix} \tag{C.16}$$

wobei $\boldsymbol{P}_{[i]}$ jeweils den i-ten Zeilenvektor und $\boldsymbol{P}_{[i,j]}$ das Element in Zeile i und Spalte j der Matrix $\boldsymbol{P}$ bezeichnet.

Die Kovarianzmatrix für das Messrauschen $\boldsymbol{Q}$ ist abhängig vom verwendeten Kamerasystem, dem Verhalten des jeweils eingesetzten Featuretrackers und des Linking-Algorithmus, außerdem variiert die Beobachtungsgenauigkeit wahrscheinlich in unterschiedlichen Umgebungen mit verschiedenen Objekt- und Oberflächenstrukturen. Eine genaue experimentelle Bestimmung ist daher äußerst schwierig. Einfacher ist eine heuristische Bestimmung mittels Abschätzung einer oberen Schranke und experimenteller Validierung. Auf diese Weise wurde der Wert

$$\boldsymbol{Q} = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix} \tag{C.17}$$

festgelegt.

Durch Einsetzen der Matrizen $\boldsymbol{A}_t$, $\boldsymbol{B}_t$, $\boldsymbol{C}_t$, $\boldsymbol{R}$ und $\boldsymbol{Q}$ in die EKF-Gleichungen B.40 bis B.46 ergibt sich die konkrete Update-Formel, mit der die Punktschätzung rekursiv an die aktuelle Beobachtung angepasst wird.

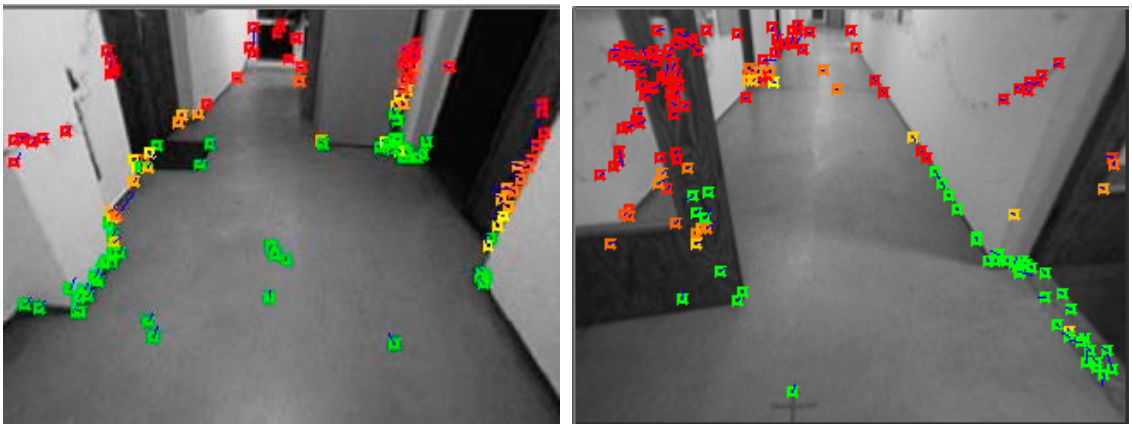

Abbildung C.11: *Beispiele für die Ergebnisse der Monokularen Tiefenschätzung: In den (entzerrten) Originalbildern sind die getrackten Feature-Punkte markiert, deren Farbe kodiert die geschätzte Höhe: von Bodenhöhe bis ca. 1m (Kamerahöhe) verläuft die Farbe von grün über gelb und orange zu rot. Besonders im rechten Bild ist zu erkennen, dass die Genauigkeit der Schätzung für weiter entfernte Punkte deutlich abnimmt. Dies resultiert daraus, dass für weiter entfernte Punkte die Verschiebung im Bild bei Bewegung der Kamera sehr gering ausfällt, wodurch Rauschen (Bildrauschen, minimale Detektions- und Trackingungenauigkeiten) die Positionsschätzung stark verfälschen kann. Bei Ännäherung an diese Punkte wird jedoch die Schätzung korrigiert und nähert sich schnell der realen Position an.*

Anhang D

Bahnregler zur Trajektorienverfolgung

In Abschnitt 5.4.2 wurden drei verschiedene Reglerkonzepte zur Steuerung der Trajektorien-Verfolgung im Kontext der lokalen Navigation durch Trajektorienauswahl vorgestellt. Aufgabe des jeweiligen Reglers ist, in diskreten Abständen Fahrkommandos zu generieren, welche gewährleisten, dass der Roboter sich entlang einer vorgegebenen Trajektorie bewegt. Dazu wird zu jedem Zeitpunkt eine Referenzposition auf der gewählten Trajektorie bestimmt, dem Regler wird jeweils die Abweichung zwischen der Referenzposition und der tatsächlichen Roboterposition in Form der tangentialen, lateralen und angularen Fehler $e_{\hat{x}}$, $e_{\hat{y}}$ und e_{φ} übergeben. Zur Ansteuerung des Roboters sind daraus Fahrkommandos in der Form von Tupeln aus Translationsgeschwindigkeit v und Rotationsgeschwindigkeit ω zu generieren.

In diesem Anhang werden die drei Ansätze PID-Regler, PID-Regler mit Vorsteuerung sowie Backstepping-Controller im Detail beschrieben und jeweils die mathematischen Formeln zur Berechnung dargestellt.

D.1 PID-Regler

Bei dem einfachen PID-Ansatz wird zunächst je ein unabhängiger PID-Regler für die tangentialen, lateralen und angularen Fehler eingesetzt. Allgemein wird ein PID-Regler beschrieben durch folgende Gleichung:

$$u(t) = PID(e(t)) = K_p \left(e(t) + \frac{1}{K_i} \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \right) \qquad \text{(D.1)}$$

Zur Konfiguration eines Reglers müssen also jeweils die drei Parameter K_p, K_i und K_d bestimmt werden. Die Regler errechnen je einen Regelausgang $u_{\hat{x}}$, $u_{\hat{y}}$ und u_φ

$$u_{\hat{x}} = PID_{\hat{x}}(e_{\hat{x}}) \qquad u_{\hat{y}} = PID_{\hat{y}}(e_{\hat{y}}) \qquad u_\varphi = PID_\varphi(e_\varphi) \qquad \text{(D.2)}$$

Durch die zuvor erfolgte Transformation der Fehler in das Roboter-Koordinatensystem und die damit verbundene Auftrennung in die tangentialen und lateralen Anteile können diese Regelgrößen den einzelnen Geschwindigkeiten zugeordnet werden: Intuitiv verlangt die Reduktion des tangentialen Fehlers eine Anpassung der Translationsgeschwindigkeit v, während der laterale und angulare Fehler die Rotationsgeschwindigkeit ω bestimmt:

$$v = u_{\hat{x}} \qquad \omega = u_{\hat{y}} + u_\varphi \qquad \text{(D.3)}$$

Als nachteilig erweist sich bei diesem Vorgehen die hohe Anzahl benötigter Parameter (je drei Parameter pro PID-Regler, Gl. D.1), deren Bestimmung allgemein immer dann schwierig ist, wenn kein analytisches Modell des Systems vorliegt.

D.2 PID-Regler mit Vorsteuerung

Die Referenz-Translationsgeschwindigkeit v_r, mit der sich die Referenzposition auf der Trajektorie bewegt, ist als Parameter vorgegeben und

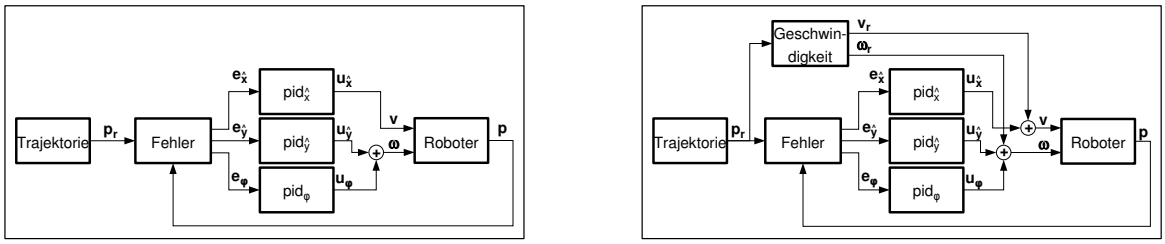

Abbildung D.1: *Die Blockschaltbilder zeigen die beiden Varianten der Trajektorienverfolgung mittels PID: Beim reinen PID-Regler (links) ergeben sich die Ansteuerungsgeschwindigkeiten für den Roboter allein aus den Fehlerausgaben der Regler. Unter Einbeziehung der Vorsteuerung (rechts) wird der Roboter im Wesentlichen mit den aus der Trajektorie abgeleiteten Referenzgeschwindigkeiten angesteuert, die Regler minimieren nur die verbleibenden, geringeren Fehler.*

kann auch zur Ansteuerung des Roboters benutzt werden. Auch eine Referenz-Rotationsgeschwindigkeit kann an jedem Punkt der Trajektorie berechnet werden: Nach Gl. 5.15 ergibt sich mit der Klothoiden-Krümmung c an der aktuellen Position

$$\omega_r = v_r \cdot c \tag{D.4}$$

Bei Ansteuerung des Roboters allein mit den Referenzgeschwindigkeiten v_r, ω_r verbleiben noch geringe Abweichungen von der Referenzposition. Diese werden wiederum mittels PID-Reglern ausgeglichen, die Reglerausgänge ergeben sich identisch zu Gl. D.2. Die Ansteuergeschwindigkeiten berechnen sich aus der Überlagerung der Referenzgeschwindigkeiten und der Regler-Ausgänge.

$$v = u_{\hat{x}} + v_r \qquad\qquad \omega = u_{\hat{y}} + u_{\varphi} + \omega_r \tag{D.5}$$

Da die Fehler, die von den PID-Reglern ausgeglichen werden müssen, bei dieser Variante wesentlich kleiner sind als bei der Nutzung nur der PID-Regler, ist die Regelung deutlich stabiler. Insbesondere zeigt sich, dass kein Differentialglied zur Dämpfung eventueller Schwingungen mehr notwendig ist ($K_d = 0$). Damit sind nur noch sechs Reglerparameter zu bestimmen. Der Einsatz der Vorsteuerung ist dem reinen PID-Regler in jedem Fall vorzuziehen.

D.3 Backstepping-Controller

Beim Backstepping-Verfahren wird aus bestimmten Stabilitätsbedingungen direkt ein Reglergesetz (als Alternative zum PID-Regler nach Gl. D.1) hergeleitet. In [Yang et al., 2004] wurde auf diese Weise ein Regler für die Trajektorienverfolgung bestimmt, der direkt die Posenfehler $e_{\hat{x}}$, $e_{\hat{y}}$ und e_{φ} in die Steuergrößen v und ω umsetzt, ohne dabei jedoch wie der PID-Regler eine Differenzierung oder Integration der Fehler zu benötigen:

$$v = v_r \cos e_{\varphi} + k_{\hat{x}} e_{\hat{x}} \qquad \omega = \omega_r + v_r (k_{\hat{y}} e_{\hat{y}} + k_{\varphi} \sin e_{\varphi}) \tag{D.6}$$

Die Einbeziehung der Vorsteuerung durch die Referenzgeschwindigkeiten v_r, ω_r ergibt sich bei diesem Regler direkt aus der analytischen Herleitung. Zur Parametrierung des Reglers sind hier drei Parameter zu bestimmen. Die Suche nach geeigneten Parametern gestaltet sich damit einfacher als bei den PID-Ansätzen.

Anhang E

SLAM-HP - FastSLAM-Variante für Bearing-Only-SLAM

Bearing-Only-SLAM bezeichnet ein spezielles SLAM-Problem, das dadurch entsteht, dass einige Sensoren nur die relative Richtung (englisch: bearing) beobachteter Objekte messen können. Ein leicht zu erkennendes Beispiel für solche Sensoren sind Kameras: wenn Landmarken oder Objekte im aufgenommenen Bild detektiert werden, so kann deren relative Richtung zum Beobachter mittels der Kamerageometrie bestimmt werden, aber nicht die Entfernung. Zwar kann u.U. aufgrund von Vorwissen über die zu erwartende Abbildungsgröße eine Entfernung geschätzt werden, oder über die Beobachtung und Wiedererkennung in mehreren Bilder (Multi-Kamera-Anordnung oder bewegte Kamera, siehe Anhang C) die räumliche Tiefe bestimmt werden, dennoch ergeben sich Unterschiede im Vergleich zum Beispiel zu einem Laser-Range-Scanner, welcher mit jeder Messung die Richtung und Entfernung der beobachteten Objekte erfasst.

In Abschnitt 4.2 wurde bereits das Prinzip des FastSLAM-Algorithmus [Montemerlo et al., 2002] vorgestellt. FastSLAM modelliert die Positionen von detektierten Landmarken als Normalverteilungen (auch als Gauss-Verteilung bezeichnet) und nutzt dazu jeweils einen Kalmanfilter für jede Landmarke. Bei der Anwendung von FastSLAM für das Bearing-Only-SLAM-Problem ergeben sich daraus Einschränkungen: Aufgrund der unbekannten Entfernung bestimmt die Beobachtung kei-

ne Normalverteilung im zweidimensionalen $[x, y]$-Raum. Diese ist aber Voraussetzung für die Modellierung mit einem Kalmanfilter. Eine einfache Möglichkeit zur Lösung dieses Problems besteht darin, für den Ort der beobachteten Landmarke eine feste Entfernung mit sehr hoher Varianz entlang der Blickrichtung anzunehmen, wie dies z.B. beim Personentracking mit visueller Sensorik (Abschnitt 6.2) beschrieben wurde. Allerdings kann dort davon ausgegangen werden, dass tatsächlich nur Objekte in unmittelbarer Nähe des Roboters von Interesse sind, so dass der modellierte Ort des beobachteten Objektes durch Vorwissen stark eingeschränkt werden kann. Im Allgemeinen führt diese Herangehensweise zu einer schlechten Repräsentation der tatsächlichen Wahrscheinlichkeitsverteilung für den Ort des Objektes.

Eine andere, häufig angewandte Methode ist die verzögerte Initialisierung von Landmarken: Dabei werden zunächst mehrere Messungen von unterschiedlichen Beobachter-Positionen zu einer Ortsschätzung für die jeweilige Landmarke verrechnet [Bailey, 2003], [Costa et al., 2004]. Zwar ergibt auch dies normalerweise keine exakte Normalverteilung, die reale Verteilung nähert sich aber einer solchen an und kann nach wenigen Beobachtungen meist recht gut von einem Kalmanfilter repräsentiert werden. Die Gauss-Ähnlichkeit kann anhand von statistischen Maßen geprüft werden, um eine Landmarke jeweils erst dann in die Umgebungsrepräsentation zu integrieren, wenn die Positionsschätzung gut zur Modellierung durch den Kalmanfilter ggeignet ist. Der Nachteil dieses Vorgehens ist, dass eine Liste der Beobachtungen und der jeweiligen Beobachtungsposition (bzw. deren Wahrscheinlichkeitsverteilung) gespeichert und bei jeder neuen Beobachtung durchsucht werden muss, um über die Kombination die neue Verteilung zu schätzen und ggf. die Landmarke in das Modell aufzunehmen. Bei Nutzung eines „klassischen“ Kalmanfilter-SLAM-Verfahrens ist zusätzlich das Problem zu lösen, welchen Einfluss die (ebenfalls durch eine Normalverteilung beschriebene) Unsicherheit über die Beobachtungsposition bei der Verrechnung der Beobachtungen zur Initialisierung einer Landmarke hat.

Ein weiterer möglicher Ansatz ist die Nutzung eines Partikelfilters für die Repräsentation der Landmarken-Position. Da Partikelfilter nicht auf Normalverteilungen beschränkt sind, ist es damit möglich, direkt nach der ersten Beobachtung den Zustandsschätzer für eine Landmarke mit der exakten Verteilung (aus dem Sensormodell) zu initialisieren. [Fitzgibbons und Nebot, 2002] kombiniert Partikelfilter und Kalmanfilter,

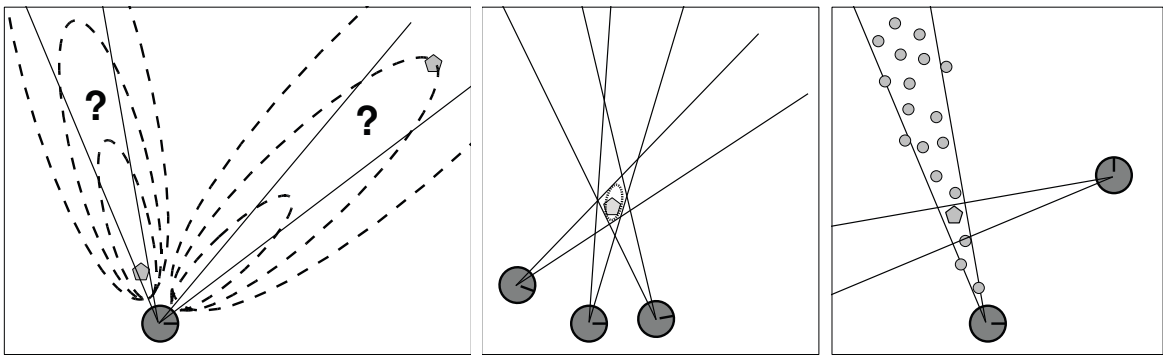

Abbildung E.1: **Links:** *Für visuelle Sensoren kann meist keine Entfernung aus den einzelnen Beobachtungen bestimmt werden. Für die Wahrscheinlichkeitsverteilung des Ortes eines beobachteten Objektes ergibt sich daher eine Normalverteilung nur in der relativen Richtung, aber eine Gleichverteilung in der Tiefe. Diese kann nur sehr grob in eine 2D-Normalverteilung transformiert werden, die Initialisierung eines Kalmanfilters aus einer einzelnen Beobachtung ist somit nicht ohne Weiteres möglich.* **Mitte:** *Die Kombinationmehrerer Beobachtungen der selben Landmarke führt zu einer deutlich Gaussähnlicheren Verteilung. Die Initialisierung des Kalmanfilters erfolgt in diesem Fall verzögert, dies erfordert eine temporäre Speicherung und spätere Zuordnung der einzelnen Beobachtungen.* **Rechts:** *Die Nutzung eines Partikelfilters für die Modellierung der Wahrscheinlichkeitsverteilung des Landmarken-Ortes ermöglicht die direkte Initialisierung mit der ersten Beobachtung. Weitere Beobachtungen aktualisieren die geschätzte Wahrscheinlichkeitsverteilung durch Anpassung der Partikelverteilung.*

indem für erstmalig beobachtete Landmarken zunächst ein Partikelfilter initialisiert wird. Dieser wird mit weiteren Beobachtungen aktualisiert. Sobald die repräsentierte Verteilung eine bestimmte Ähnlichkeit zu einer Gauss-Verteilung erreicht, wird der Partikelfilter durch einen Kalmanfilter ersetzt, da dieser effizienter berechenbar ist und für eine zweidimensionale Verteilung weniger Speicherplatz benötigt.

Aus diesen Vorüberlegungen wurde in einer Diplomarbeit [Täubig, 2004] eine Kombination aus FastSLAM und der Repräsentation der Landmarken mittels Partikelfiltern entwickelt. Die Kalmanfilter, welche im originalen FastSLAM-Ansatz die Landmarken modellieren, wurden dazu jeweils durch einen Partikelfilter ersetzt. Um die Unterscheidung zwischen dem äußeren Partikelfilter für die Roboterposition und den inneren Landmarken-Partikelfiltern zu erleichtern, wird der äußere Filter als Makro-Partikelfilter bezeichnet, die inneren als Mikro-Partikelfilter. Für das Verfahren wurde die Bezeichnung „SLAM-HP: SLAM mit Hierarchischen Partikelfiltern“ geprägt [Täubig und Schröter, 2004].

Im Bewegungsupdate werden jeweils die Partikelpositionen des Makropartikelfilters angepasst, dies geschieht genauso wie in Kapitel 4 für Map-Match-SLAM beschrieben. Das Beobachtungsupdate bewertet parallel die Mikropartikel und Makropartikel jeweils anhand der aktuellen Schätzung der Landmarken (Mikropartikel) und der Roboterposition (Makropartikel). Anschließend werden die Verteilungen der Mikro- und Makropartikel adaptiert. Der Ablauf ist im folgenden als Pseudocode beschrieben (Abb. E.2). Der Pseudocode für das Beobachtungsupdate berücksichtigt jeweils nur eine einzelne Landmarke. Wenn mehrere Landmarken gleichzeitig beobachtet werden, so können diese sequentiell als Einzelbeobachtungen behandelt werden. Eine Landmarke, die erstmalig beobachtet wird, wird initialisiert, indem für diese Landmarke ein Mikropartikelfilter in jedem Makropartikel angelegt wird und die Partikel mit einer Initialverteilung eingestreut werden, die sich aus einer Normalverteilung im Winkel und einer Gleichverteilung in der Entfernung (mit vorgegebener Maximalentfernung) zusammensetzt.

Eine Besonderheit tritt beim Resampling der Mikropartikel auf: Diese schätzen die Wahrscheinlichkeitsverteilung für eine statische Größe (jeweils die Position einer Landmarken). Das Bewegungsmodell besagt demzufolge, dass die Partikel ihre Position nie verändern. Wenn weiterhin im Resampling lediglich eine Untermenge der existierenden Partikel exakt kopiert wird (vgl. Abschnitt 4.3.1), so können niemals Mikropartikel an anderen Positionen als den während der Initialisierung gewählten liegen. Da die einzelnen Partikel Hypothesen für die reale Position der entsprechenden Landmarke darstellen, wäre für eine hinreichende Genauigkeit der letztendlichen Schätzung eine extreme Zahl an Partikeln notwendig, um den Raum sehr dicht abzudecken. Statt dessen wurde in dieser Arbeit eine andere Alternative genutzt: Obwohl die Landmarken eigentlich unbeweglich sind, wird eine leichte Rauschbewegung für die Partikel eingeführt. Diese sind dadurch in der Lage, ihre Position im Raum zu verschieben und somit auch gegenüber der Initialisierung neue Positionen einzunehmen. Es zeigt sich, dass dies die Konvergenzgeschwindigkeit etwas verringert, die letzten Endes erreichbare Genauigkeit der Schätzung aber verbessert.

Die Implementierung dieses Verfahrens wurde anhand von Experimenten in einem einfachen realen Szenario verifiziert: Ein Khepera-Miniaturroboter mit einer omnidirektionalen Kamera bewegte sich auf einer Fläche von ca. 1m$times$1m. Innerhalb der Umgebung wurden als Land-

Eingaben

K `Makropartikel mit` $x_t^{(k)}$) // *Schätzung der Roboterposition*
`je Makropartikel:` I `Mikropartikelfilter` // *1 Schätzer je Makropartikel*
// *je Landmarke*
`je Mikropartikelfilter:` L `Mikropartikel mit` $m_{i,t}^{(k,l)}$ // *Schätzung der*
// *Landmarke* i

$o_t = [i, \varphi]$ // *Beobachtung der Landmarke* i *unter Winkel* φ
// *(relativ zu Roboterorientierung)*

Initialisierung

$w_t^{(k)} \leftarrow 0$ // *Makropartikel-Gewichte*

Algorithmus

`für` $k = 1 : K$ // *für alle Makroartikel*
  `für` $l = 1 : L$ // *für alle Mikropartikel*
    $w_{i,t}^{(k,l)} = p\left(o_t | x_t^{(k)}, m_{i,t}^{(k,l)}\right)$ // *Beobachtungswahrscheinlichkeit*
    $w_t^{(k)} = w_t^{(k)} + \frac{1}{L} \cdot w_{i,t}^{(k,l)}$ // *mittlere Beobachtungswahrscheinlichkeit*
    // *aller Mikropartikel*
  `Resampling der Mikropartikel`
`Resampling der Makropartikel`

Abbildung E.2: *Im Beobachtungsupdate werden die Gewichte der einzelnen Mikropartikel* $w_{i,t}^{(k,l)}$ *aus der Beobachtungswahrscheinlichkeit für die aktuelle Beobachtung* o_t *berechnet, jeweils aus der im Makropartikel geschätzten Roboterposition* $x_t^{(k)}$ *und der im Mikropartikel geschätzten Landmarkenposition* $m_{i,t}^{(k,l)}$*. Das Gewicht eines Makropartikels* $w_t^{(k)}$ *ergibt sich aus dem mittleren Gewicht seiner Mikropartikel für die beobachtete Landmarke* i*.*

marken verschiedenfarbige Objekte plaziert. Da der Schwerpunkt bei diesen Untersuchungen auf dem Nachweis der Funktionsfähigkeit des SLAM-HP-Verfahrens lag, wurde die Landmarkendetektion möglichst einfach gehalten, indem wenige, gut unterscheidbare Farben für die einzelnen Landmarken verwendet wurden. Für die Landmarkendetektion wurde ein Beobachtungsmodell erstellt, welches die Fehlerverteilung für die detektierte Richtung der Landmarke (Position im Panoramabild) angibt. Da die Detektion der einzelnen Farben unterschiedlich zuverlässig und genau ist, ergibt sich für jede Landmarkenfarbe ein spezifisches Beobachtungsmodell. Die Bestimmung der Ground Truth für die Positionen der Landmarken und des Roboters erfolgte, indem das befahrene Areal mit einem metrischen Raster hinterlegt wurde und während der Fahrt in regelmäßigen Abständen die aktuelle Position protokolliert wurde.

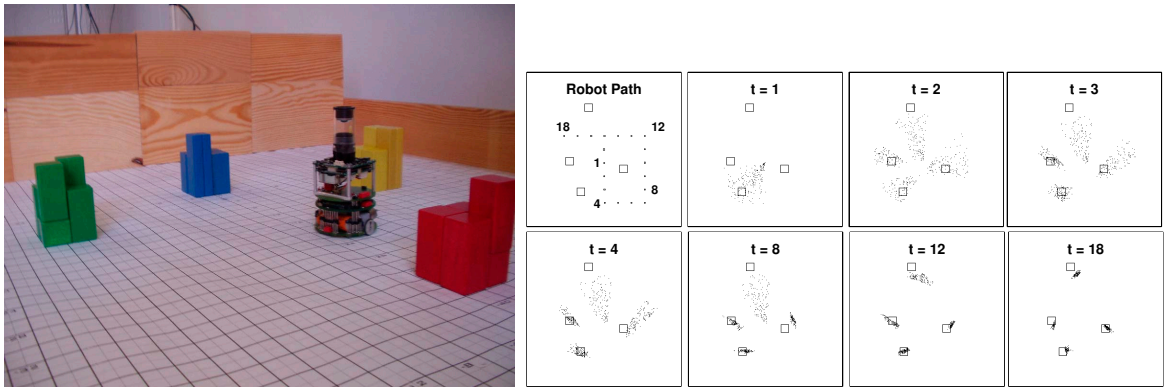

Abbildung E.3: **Links:** *Der Khepera-Roboter zwischen einigen farbigen Landmarken.* **Rechts:** *Beispielexperiment mit 4 Landmarken (markiert durch Quadrate, die die tatsächliche Ausdehnung wiedergeben). Die erste Darstellung zeigt den Roboterpfad und die Positionen der 4 Landmarken. Die weiteren Abbildungen stellen jeweils die Verteilung der Partikel für jede Landmarke innerhalb eines exemplarisch ausgewählten Makropartikels dar: Es ist gut zu erkennen, dass die Verteilung zunächst sehr breit ist und kaum Ähnlichkeit mit einer Normalverteilung aufweist. Mit fortgesetzten Beobachtungen konvergieren die Schätzungen und nähern sich sehr gut den realen Positionen an.*

Abb. E.3 zeigt beispielhaft den Verlauf der Landmarken-Modellierung während einer Sequenz von Bewegungen und Beobachtungen des Roboters. Wie erwartet zeigen die geschätzten Verteilungen für die Landmarkenpositionen anfangs kaum Ähnlichkeit mit einer Normalverteilung und wären damit für eine Modellierung mittels Kalmanfilter kaum geeignet. Durch fortlaufende Beobachtungen der Landmarken insbesondere von deutlich unterschiedlichen Beobachtungspositionen konvergieren die ursprünglich sehr breiten Verteilungen und nähern sich gut den realen Landmarken an. Das SLAM-HP-Verfahren ist also (bei geeigneter Detektion der Landmarken) gut zur Lösung des Bearing-Only-SLAM geeignet.

Die korrekte Bestimmung des Roboter-Pfades wird in Abb. E.4 gezeigt. Wie erwartet entspricht der SLAM-Pfad deutlich genauer dem realen Roboter-Pfad als die von der Odometrie gemessene Bewegung. Der Fehler gegenüber der Ground Truth beträgt maximal wenige cm.

Außer dem prinzipiellen Nachweis der Funktionsfähigkeit des Algorithmus wurden u.a. Untersuchungen zur Bestimmung der Anzahl der benötigten Partikel durchgeführt. Es wurde gezeigt, dass in dem einfachen Experimental-Szenario 200 Makropartikel mit jeweils 100 Mi-

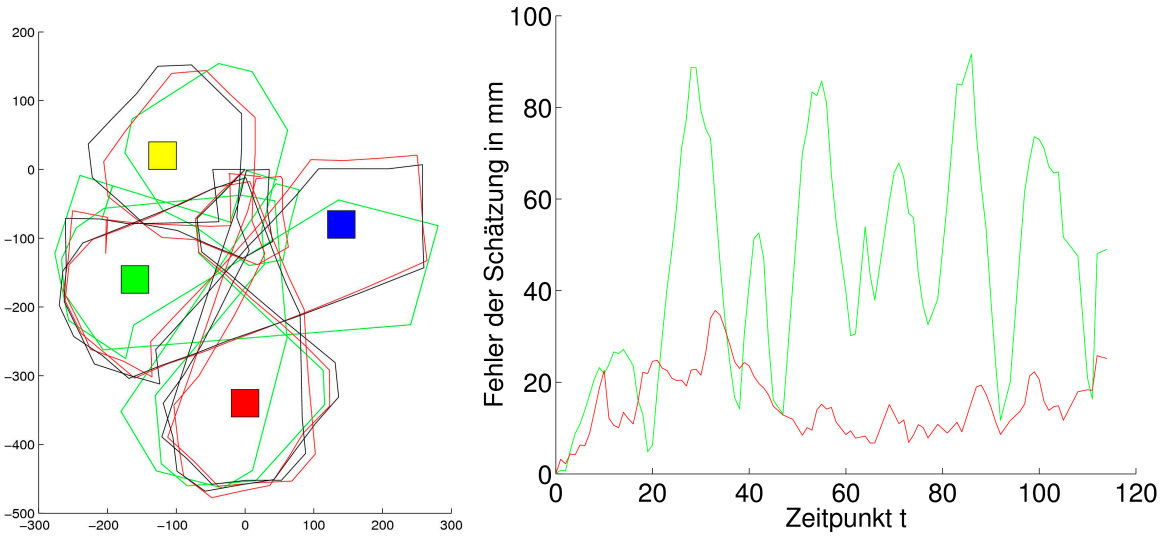

Abbildung E.4: **Links:** *Darstellung der Landmarken-Positionen und des Roboter-Pfades (schwarz = Ground Truth, grün = Odometrie, rot = SLAM).* **Rechts:** *Verlauf des Positionsfehlers über die Zeit (grün = Odometrie, rot = SLAM). Der SLAM-Pfad weist bei einer befahrenen Fläche von ca. 50cmtimes50cm und einer Gesamtstrecke von mehreren Metern einen maximalen Fehler von 4cm zur Ground Truth auf.*

korpartikeln je Landmarke ausreichen, um eine robuste und genaue Schätzung des Roboterpfades und der Landmarkenpositionen zu gewährleisten.

Abkürzungsverzeichnis

DP	Distributed Particles(-SLAM)
EKF	Extended Kalman Filter/Erweiterter Kalmanfilter
GPS	Global Positioning System
KF	Kalmanfilter
KLD	Kullback-Leibler Distance/Kullback-Leibler-Divergenz
LUT	Look-Up-Table/Look-Up-Tabelle
MCL	Monte Carlo Localization
NIKR	(Fachgebiet) Neuroinformatik und Kognitive Robotik
PF	Partikelfilter
PID	Proportional-Integral-Differential(-Regler)
PSO	Particle Swarm Optimization
RAAA	Robots Abstracting Application Architecture
RBPF	Rao-Blackwellized Particle Filter
RFID	Radio Frequency Identification
SLAM	Simultaneous Localization and Mapping
STOC	Stereo On Chip
VFH	Vector Field Histogram

Abbildungsverzeichnis

1.1 Shopping-Assistent SCITOS 3
1.2 Komponenten des Service-Roboters 5

2.1 Taxonomie Umgebungsmodelle 15
2.2 Beispiel: Geometrische Karte 17
2.3 Beispiel: Gridkarte 18
2.4 Beispiel: Topologische Karte 20

3.1 Scanner und Sensor-Array 25
3.2 Belegtheits-Schätzung 28
3.3 Beschleunigte Belegtheits-Schätzung mit Look-Up-Tables 30
3.4 Gegenüberstellung Belegtheitsschätzung ohne/mit Look-Up-Table 31
3.5 Beobachtungen einer Zelle: Graphisches Modell 32
3.6 Bildsequenz: Kartenaufbau 34
3.7 Belegtheits-Schätzung aus 3D-Punkten 36
3.8 Karten-Fusion 40
3.9 PERSES Odometrie-Fehler 42
3.10 Korrektur des systematischen Odometrie-Fehlers: Ergebnis 42
3.11 Orientierungsreferenz: Motivation 44
3.12 Orientierungsreferenz: Algorithmus 46
3.13 Orientierungsreferenz: Ergebnis 47
3.14 Referenzkarte 48

4.1 Map Matching: lokale/globale Karte 54
4.2 Map Matching: Einfache Korrelation 56
4.3 Map Matching: Korrelation + Kovarianz 57
4.4 Pseudocode: optimaler Verschiebungsvektor 58
4.5 SLAM: Übersicht State-of-the-Art 63
4.6 SLAM: Kalman-Filter/RBPF für Landmarken 64
4.7 SLAM: RBPF für Gridkarten 67
4.8 Map-Match-SLAM: Überblick Datenstruktur 71
4.9 Bewegungsmodell 75
4.10 Kartenupdate 77
4.11 Kreisschluss 80
4.12 SLAM-Karte 81
4.13 Vergleich der Partikel-Karten 82
4.14 Shared Gridmaps: Struktur 85
4.15 Shared Gridmaps: Auswertung 86
4.16 On-Demand Maps: Kartierung einer großen Schleife . . . 89
4.17 Positionsunsicherheit 90
4.18 Beispiel: KLD-Sampling 92
4.19 Pseudocode: dynamisches Resampling 94
4.20 Vergleich konstante und dynamische Partikelzahl - Datenplot 95
4.21 Vergleich konstante und dynamische Partikelzahl - Karten 96
4.22 toom Baumarkt Erfurt - Datenplot SLAM 97
4.23 toom Baumarkt Erfurt - Karte 98
4.24 Verarbeitung der Stereo-Daten 100
4.25 Kamera- und Roboterkoordinatensystem 101
4.26 Sichtbereich der Stereo-Kamera 102
4.27 SLAM mit Stereo-Kamera 103
4.28 Depth-from-Motion 105
4.29 SLAM mit Depth-from-Motion 105

4.30 Zustandsdiagramm für den SLAM-Assistenten 107
4.31 Topologische Karte des SLAM-Assistenten 109
4.32 Beispielsituationen der Assistenten-Modi 110
4.33 Kartierungs-Sequenz mit Assistent 112
4.34 Referenz-Karte für die assistierte SLAM-Kartierung . . . 113
4.35 Zustandsverlauf während der assistierten Kartierung . . . 113

5.1 Selbstlokalisation mit Entfernungssensoren 121
5.2 Visual MCL: Segmentierung der Ansicht 123
5.3 Überblick Navigationssystem 125
5.4 Pseudocode: Dijkstra-Algorithmus auf der Gridkarte . . . 129
5.5 Suchverlauf Dijkstra vs. A* 130
5.6 Pfadplanung Beispiel 1 130
5.7 Pfadplanung Beispiel 2 130
5.8 Belegtheitshistogramm bei VFH 134
5.9 Hindernis-Dilatation auf Basis des virtuellen Scans . . . 136
5.10 Dynamische Entfernungsschwelle im VHF 138
5.11 Zulässige Geschwindigkeit in Abhängigkeit vom Hindernisabstand 140
5.12 Geschwindigkeit als Funktion der Bewegungsrichtung . . 141
5.13 Pseudocode: Erweitertes Vector Field Histogram 143
5.14 Klothoid-Kurven 146
5.15 Fahrtsteuerung durch Trajektorien-Sampling 147
5.16 Kostenfunktion zur Trajektorien-Bewertung 150
5.17 Beispiel Trajektorien-Auswahl 151
5.18 Multi-Trajektorien-Navigation mit dynamischen Hindernissen 151
5.19 Koordination Navigator/Trajektorien-Regler 153
5.20 Transformation des Positionsfehlers bei der Trajektorienverfolgung 155
5.21 Lokale Pfadplanung 157

5.22 Lokale Pfadplanung: Frühzeitige Erkennung eines versperrten Weges 158
5.23 MT-Navigation: Trajektorienauswahl bei Türdurchfahrt . 159
5.24 Vergleich VFH und Multi-Trajektorien-Navigation: Experiment 1 161
5.25 Vergleich VFH und Multi-Trajektorien-Navigation: Experiment 2 162
5.26 Einschränkungen der Klothoid-Trajektorien 163

6.1 Robot Abstracting Application Architecture 168
6.2 Personen-Tracker-Cues - Hautfarbe, Bewegung, Beinpaar 175
6.3 Personen-Tracker-Cues - Gesicht, dynamische Objekte . . 176
6.4 Probabilistische Sensorfusion für das Personentracking . 177
6.5 Graphische Oberfläche des Shopping-Assistenten 179
6.6 Shopping-Roboter im Baumarkt 181

7.1 Hinderniswahrnehmung durch Sonar, Laser und visuelle Sensorik 186
7.2 Bildsequenz: visuelle Hindernisdetektion im Baumarkt . . 188
7.3 Visuelle Hindernisdetektion: lokale Karte 189
7.4 Beispiel: Lokalisationsfehler durch Veränderung der Umgebung 191
7.5 Übersicht über die Komponenten der dynamischen Kartenadaption 194
7.6 Mittlere Partikelgewichte für die Gütebewertung der Selbstlokalisation 195
7.7 Umgebungsrepräsentation mittels Samples von Referenzbeobachtungen 196
7.8 Stabilität der topologischen Karte 197
7.9 Kartenupdate unter Nutzung von Zellkorrelationen . . . 199
7.10 Verteiltes Sampling über mehrere Karten 201
7.11 Experiment zur dynamischen Kartenadaption 202

A.1 Synchro-Drive: Einfluss der Radstellung 215
A.2 Odometrie-Kalibrierung: Messung des Kreisradius 217
A.3 Odometrie-Kalibrierung: Messdaten 218
A.4 Odometrie-Kalibrierung: Ergebnis 220

B.1 Rekursive Schätzung eines statischen Zustandes 224
B.2 Rekursive Schätzung eines dynamischen Zustandes 225
B.3 Pseudocode: Rekursives Update des Partikelfilter 234

C.1 Epipolargeometrie an zwei Kameras 237
C.2 Herleitung der Disparität 239
C.3 Stereo-Bildpaar 240
C.4 Stereo-Disparität mit und ohne Kanten-Maskierung . . . 241
C.5 Darstellung der Disparitäten als 3D-Punktwolke 242
C.6 Homogene Koordinaten 244
C.7 Transformation zwischen Koordinatensystemen 246
C.8 Rückprojektion eines Bild-Punktes in den 3D-Raum . . . 247
C.9 Architekturübersicht Monokulare Tiefenschätzung 249
C.10 Match-Hypothesen für das Featuretracking 250
C.11 Ergebnisse Monokulare Tiefenschätzung 253

D.1 Trajektorienverfolgung mittels PID-Reglern 257

E.1 Bearing-Only-SLAM 261
E.2 Pseudocode: SLAM-HP Beobachtungsupdate 263
E.3 SLAM-HP: Khepera-Szenario, Landmarken-Schätzung . . 264
E.4 SLAM-HP: Pfadschätzung 265

Literaturverzeichnis

[Bailey, 2003] Bailey, T. (2003). Constrained Initialisation for Bearing-Only SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, Seiten 1966–1971.

[Besl und McKay, 1992] Besl, P. und McKay, H. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

[Biber und Duckett, 2005] Biber, P. und Duckett, T. (2005). Dynamic Maps for Long-Term Operation of Mobile Service Robots. In *Proceedings of Robotics: Science and Systems*, Seiten 17–24.

[Blanco et al., 2007] Blanco, J.-L., Fernandez-Madrigal, J.-A., und Gonzalez, J. (2007). A New Approach for Large-Scale Localization and Mapping: Hybrid Metric-Topological SLAM. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Seiten 2061 – 2067.

[Blanco et al., 2008] Blanco, J.-L., Fernandez-Madrigal, J.-A., und Gonzalez, J. (2008). Toward a Unified Bayesian Approach to Hybrid Metric-Topological SLAM. *IEEE Transactions on Robotics*, 24(2):259 – 270.

[Borenstein und Feng, 1996] Borenstein, J. und Feng, L. (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880.

[Borenstein und Koren, 1991] Borenstein, J. und Koren, Y. (1991). The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 7(3):278 – 288.

[Burgard et al., 1999] Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., und Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2):3 – 55.

[Buschka und Saffiotti, 2002] Buschka, P. und Saffiotti, A. (2002). A virtual sensor for room detection. In *IEEE/RSJ International Conference on Intelligent Robots and System (IROS 2002)*, Seiten 637–642.

[Böhme und Gross, 1999] Böhme, H.-J. und Gross, H.-M. (1999). Ein Interaktives Mobiles Service-System für den Baumarkt. In *15. Fachgespräch Autonome Mobile Systeme (AMS 1999)*, Seiten 344–351.

[Böhme et al., 2000] Böhme, H.-J., Gross, H.-M., Key, J., und Wilhelm, T. (2000). PERSES - a PERsonal SErvice System. In *Proc. SOAVE2000 - SelbstOrganisation von Adaptivem VErhalten, Ilmenau*, Seiten 57–66.

[Böhme et al., 2006] Böhme, H.-J., Scheidig, A., Wilhelm, T., Schröter, C., Martin, C., König, A., Müller, S., und Gross, H.-M. (2006). Progress in the Development of an Interactive Mobile Shopping Assistant. In *Proceedings of the Joint Conference on Robotics: 37th International Symposium on Robotics (ISR 2006) and 4th German Conference on Robotics (Robotik 2006).*

[Böhme et al., 2005] Böhme, H.-J., Wilhelm, T., und Gross, H.-M. (2005). Gesichtsanalyse für die intuitive Mensch-Roboter-Interaktion. In *Proceedings Autonome Mobile Systeme (AMS 2005)*, Seiten 67–73.

[Böhme et al., 2001a] Böhme, H.-J., Wilhelm, T., Hempel, T., Schröter, C., und Gross, H.-M. (2001a). An Approach to Multimodal Human-Machine Interaction for Intelligent Service Robots. In *Proceedings of the Fourth European Workshop on Advanced Mobile Robots (Eurobot 2001)*, Seiten 17–24.

[Böhme et al., 2001b] Böhme, H.-J., Wilhelm, T., Hempel, T., Schröter, C., und Gross, H.-M. (2001b). Multimodale Mensch-Maschine-Interaktion für Servicerobotik-Systeme. In *Proceedings Autonome Mobile Systeme (AMS 2001)*, Seiten 113–119.

[Böhme et al., 2002] Böhme, H.-J., Wilhelm, T., Schröter, C., und Gross, H.-M. (2002). PERSES - ein interaktiver Einkaufsassistent. In *Robotik 2002: Leistungsstand - Anwendungen - Visionen - Trends*, Seiten 641–646.

[Böhme et al., 2004] Böhme, H.-J., Wilhelm, T., Schröter, C., König, A., und Martin, C. (2004). Eine hybride Steuerarchitektur für einen interaktiven mobilen Serviceroboter. In *Proceedings of the 3rd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE 2004)*, Seiten 187–197.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679 – 698.

[Chatila und Laumond, 1985] Chatila, R. und Laumond, J. (1985). Position referencing and consistent world modeling for mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 138–145.

[Chen und Medioni, 1991] Chen, Y. und Medioni, G. (1991). Object modeling by registration of multiple range images. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 2724–2729.

[Clemente et al., 2007] Clemente, L., Davison, A., Reid, I., Neira, J., und Tardos, J. (2007). Mapping Large Loops with a Single Hand-Held Camera. In *Proceedings of Robotics: Science and Systems*.

[Costa et al., 2004] Costa, A., Kantor, G., und Choset, H. (2004). Bearing-only Landmark Initialization with Unknown Data Association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, Seiten 1764 – 1770.

[Csorba, 1997] Csorba, M. (1997). *Simultaneous Localization and Map Building*. PhD thesis, University of Oxford.

[Davison, 2003] Davison, A. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the International Conference on Computer Vision*, Seiten 1403 – 1410.

[Davison und Murray, 2002] Davison, A. und Murray, D. (2002). Simultaneous Localisation and Map-Building Using Active Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):865–880.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W., und Thrun, S. (1999). Monte Carlo Localization for Mobile Robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 1322 – 1328.

[Dijsktra, 1959] Dijsktra, E. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271.

[Eberhart und Kennedy, 1995] Eberhart, R. C. und Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Seiten 39–43.

[Einhorn, 2007] Einhorn, E. (2007). Robuste 3D-Szenenrekonstruktion für einen mobilen Roboter. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Einhorn et al., 2007a] Einhorn, E., Schröter, C., Böhme, H.-J., und Gross, H.-M. (2007a). A Hybrid Kalman Filter Based Algorithm for Real-time Visual Obstacle Detection. In *Proceedings of the 3rd European Conference on Mobile Robots (ECMR)*, Seiten 156–161.

[Einhorn et al., 2007b] Einhorn, E., Schröter, C., Böhme, H.-J., und Gross, H.-M. (2007b). A Hybrid Kalman Filter Based Algorithm for Real-time Visual Obstacle Detection. In *Proceedings of the 52nd International Scientific Colloquium (IWK)*, Seiten 353–358.

[Eliazar und Parr, 2003] Eliazar, A. I. und Parr, R. (2003). DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Seiten 1135–1142.

[Eliazar und Parr, 2004] Eliazar, A. I. und Parr, R. (2004). DP-SLAM 2.0. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 1314– 1320.

[Fitzgibbons und Nebot, 2002] Fitzgibbons, T. und Nebot, E. (2002). Bearing-Only SLAM using Colour-based Feature Tracking. In *Proceedings of the 2002 Australasian Conference on Robotics and Automation*, Seiten 234–239.

[Fox, 2001] Fox, D. (2001). KLD-Sampling: Adaptive Particle Filters. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, Seiten 713–720.

[Fox, 2003] Fox, D. (2003). Adapting the Sample Size in Particle Filters Through KLD-Sampling. *The International Journal of Robotics Research*, 22(12):985–1003.

[Fox et al., 1999] Fox, D., Burgard, W., Dellaert, F., und Thrun, S. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proceedings of the AAAI National Conference on Artifical Intelligence*, Seiten 343–349.

[Graf und Barth, 2002] Graf, B. und Barth, O. (2002). Entertainment robotics: examples, key technologies and perspectives. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems–Workshop „Robots in Exhibitions“*.

[Graf et al., 2000] Graf, B., Baum, W., Traub, A., und Schraft, R. (2000). Konzeption dreier Roboter zur Unterhaltung der Besucher eines Museums. In *Robotik 2000: Leistungsstand - Anwendungen - Visionen - Trends*, Seiten 529–536.

[Grisetti et al., 2005] Grisetti, G., Stachniss, C., und Burgard, W. (2005). Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 2443–2448.

[Grisetti et al., 2007] Grisetti, G., Tipaldi, G. D., Stachniss, C., Burgard, W., und Nardi, D. (2007). Fast and Accurate SLAM with Rao-Blackwellized Particle Filters. *Robotics and Autonomous Systems*, 55(1):30–38.

[Gross und Böhme, 2000a] Gross, H.-M. und Böhme, H.-J. (2000a). PERSES - a Vision-based Interactive Mobile Shopping Assistant. In

Proc. IEEE Int. Conference on Systems, Man and Cybernetics (SMC 2000), Seiten 80–85.

[Gross und Böhme, 2000b] Gross, H.-M. und Böhme, H.-J. (2000b). PERSES - a Vision-based Interactive Mobile Shopping Assistant: Scenario, Behavior-based Control Architecture and First Results. In *Proc. Fourth Int. Conference on Cognitive and Neural Systems*, Seite 29.

[Gross et al., 2002a] Gross, H.-M., Böhme, H.-J., Schröter, C., und König, A. (2002a). Panoramic View based Monte Carlo Self-localization for Mobile Robots Operating in Complex Real-world Environments. In *Proceedings of the International Workshop on Dynamic Perception*, Seiten 171–176.

[Gross et al., 2008] Gross, H.-M., Böhme, H.-J., Schröter, C., Müller, S., König, A., Martin, C., und Merten, M.and Bley, A. (2008). ShopBot: Progress in Developing an Interactive Mobile Shopping Assistant for Everyday Use. In *Proc. IEEE Internat. Conf. on Systems, Man and Cybernetics (IEEE-SMC 2008)*, Seiten 3471–3478.

[Gross et al., 2001] Gross, H.-M., Böhme, H.-J., und Wilhelm, T. (2001). A contribution to vision-based localization, tracking and navigation methods for an interactive mobile service-robot. In *2001 IEEE Intern. Conference on Systems, Man and Cybernetics (SMC 2001)*, Seiten 672–677.

[Gross und König, 2004] Gross, H.-M. und König, A. (2004). Robust Omniview-based Probalistic Self-Localization for Mobile Robots in Large Maze-like Environments. In *Proc. 17th Int. Conference on Pattern Recognition (ICPR 2004)*, Seiten 266–269.

[Gross et al., 2002b] Gross, H.-M., König, A., Böhme, H.-J., und Schröter, C. (2002b). Vision-Based Monte Carlo Self-localization for a Mobile Service Robot Acting as Shopping Assistant in a Home Store. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, Seiten 256–262.

[Gross et al., 2003] Gross, H.-M., König, A., Schröter, C., und Böhme, H.-J. (2003). Omnivision-based Probalistic Self-Localization for a

Mobile Shopping Assistant Continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Seiten 1505–1511.

[Gross et al., 2006] Gross, H.-M., Richarz, J., Müller, S., Scheidig, A., und Martin, C. (2006). Probabilistic Multi-modal People Tracker and Monocular Pointing Pose Estimator for Visual Instruction of Mobile Robot Assistants. In *Proc. 2006 IEEE World Congress on Computational Intelligence (WCCI 2006), International Joint Conference on Neural Networks (IJCNN-06)*, Seiten 8325–8333.

[Gutmann und Fox, 2002] Gutmann, J.-S. und Fox, D. (2002). An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, Seiten 454–459.

[Gutmann und Schlegel, 1996] Gutmann, J.-S. und Schlegel, C. (1996). AMOS: Comparison of Scan Matching Approaches for Self-Localization in Indoor Environments. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, Seiten 61–67.

[Harris und Stephens, 1988] Harris, C. und Stephens, M. (1988). A Combined Corner and Edge Detector. In *4th ALVEY Vision Conference*, Seiten 147 – 151.

[Hart et al., 1968] Hart, P., Nilsson, N. j., und Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

[Hartley und Zisserman, 2004] Hartley, R. und Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition.

[Hähnel, 2004] Hähnel, D. (2004). *Mapping With Mobile Robots*. PhD thesis, Albert-Ludwigs-Universität Freiburg.

[Hähnel et al., 2003] Hähnel, D., Burgard, W., Fox, D., und Thrun, S. (2003). An Efficient FastSLAM Algorithm for Generating Maps of

Large-Scale Cyclic Environments from Raw Laser Range Measurements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Seiten 206–211.

[Jensen et al., 2002] Jensen, B., Philippsen, R., Froidevaux, G., und Siegwart, R. (2002). Mensch-Maschine Interaktion auf der Expo.02. In *Robotik 2002: Leistungsstand - Anwendungen - Visionen - Trends*.

[Julier und Uhlmann, 1997] Julier, S. J. und Uhlmann, J. K. (1997). A new extension of the Kalman filter to nonlinear systems. In *Proc. SPIE AeroSense Symposium 1997*, Seiten 182–193.

[Kalman, 1960] Kalman, R. (1960). A new Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

[Kanayama et al., 1991] Kanayama, Y., Kimura, Y., Miyazaki, F., und und Noguchi, T. (1991). A Stable Tracking Control Method for a Non-Holonomic Mobile Robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1991)*, Seiten 1236 – 1241.

[Kanda et al., 2004] Kanda, T., Hirano, T., Eaton, D., und Ishiguro, H. (2004). Interactive Robots as Social Partners and Peer Tutors for Children: A Field Trial. *Human-Computer Interaction*, 19(1-2):61–84.

[Keichel, 2007] Keichel, S. (2007). Online-Kartierung mittels Map-Match-SLAM. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Kennedy und Eberhart, 1995] Kennedy, J. und Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, Seiten 1942–1948.

[Klatzky, 1998] Klatzky, R. (1998). Allocentric and Egocentric Spatial Representations: Definitions, Distinctions, and Interconnections. In *Spatial Cognition*, Seiten 1–17. Springer Berlin / Heidelberg.

[Kraetzschmar et al., 2000] Kraetzschmar, G., Sablatnög, S., Enderle, S., Utz, H., Simon, S., und Palm, G. (2000). Integration of Multiple Representation and Navigation Concepts on Autonomous Mobile

Robots. In *Proceedings of the 2nd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE 2000)*, Seiten 1–13.

[Kuipers, 1978] Kuipers, B. (1978). Modelling Spatial Knowledge. *Cognitive Science*, 2(2):129–153.

[Kuipers, 2000] Kuipers, B. (2000). The Spatial Semantic Hierarchy. *AI Magazine*, 119(1-2):191–233.

[Kuipers, 2008] Kuipers, B. (2008). An Intellectual History of the Spatial Semantic Hierarchy. In *Robotics and Cognitive Approaches to Spatial Mapping*, Seiten 243–264. Springer.

[Kuipers und Beeson, 2002] Kuipers, B. und Beeson, P. (2002). Bootstrap learning for place recognition. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Seiten 174 – 180.

[Kuipers und Byun, 1991] Kuipers, B. und Byun, Y.-T. (1991). A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. *Journal of Robotics and Autonomous Systems*, 8(1-2):47–63.

[König und Bischoff, 2004] König, A. und Bischoff, M. (2004). Adaptive Reference Views for Appearance-based Localization. In *3rd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE 2004)*, Seiten 146–156.

[König und Gross, 2008] König, A. und Gross, H.-M. (2008). A Graph Matching Technique for an Appearance-based, visual SLAM-Approach using Rao-Blackwellized Particle Filters. In *IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS 2008)*, Seiten 1576 – 1581.

[König et al., 2000] König, A., Key, J., und Gross, H.-M. (2000). Visuell basierte Monte-Carlo-Lokalisation für mobile Roboter mit omnidirektionalen Kameras. In *Proc. SOAVE2000 - SelbstOrganisation von Adaptivem VErhalten*, Seiten 31–38.

[König et al., 2003a] König, A., Moser, S., und Gross, H.-M. (2003a). Farbhistogramm-basierte visuelle Monte-Carlo-Lokalisation für Mobile Roboter. In *Proc. Autonome Mobile Systeme (AMS 2003)*, Seiten 214–222.

[König et al., 2003b] König, A., Moser, S., und Gross, H.-M. (2003b). Untersuchung alternativer Farbmerkmale für die visuelle Monte-Carlo-Selbstlokalisation für mobile Roboter. In *Proc. 9. Workshop Farbbildverarbeitung*, Seiten 51–58.

[Leonard und Durrant-Whyte, 1991] Leonard, J. J. und Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the International Workshop on Intelligent Robots and Systems (IROS'91)*, Seiten 1442–1447.

[Liu und Thrun, 2003] Liu, Y. und Thrun, S. (2003). Results for Outdoor-SLAM Using Sparse Extended Information Filters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 1227–1233.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of hte Seventh International Conference on Computer Vision (ICCV'99)*, Seiten 1150–1157.

[Martin et al., 2004] Martin, C., Böhme, H.-J., und Gross, H.-M. (2004). Conception and Realization of a Multi-sensory, Interactive Mobile Office Guide. In *Proc. 2004 IEEE Int. Conf. on Systems, Man and Cybernetics (SMC 2004)*, Seiten 5368–5373.

[Martin und Gross, 2008] Martin, C. und Gross, H.-M. (2008). A Real-time Facial Expression Recognition System based on Active Appearance Models using Gray Images and Edge Images. In *IEEE Int. Conference on Face and Gesture Recognition*, Seite (paper no. 299).

[Martin et al., 2005a] Martin, C., Schaffernicht, E., Scheidig, A., und Gross, H.-M. (2005a). Sensor Fusion using a Probabilistic Aggregation Scheme for People Detection and Tracking. In *Proceedings of the 2nd European Conference on Mobile Robots (ECMR 2005)*, Seiten 176–181.

[Martin et al., 2005b] Martin, C., Scheidig, A., Wilhelm, T., Schröter, C., Böhme, H.-J., und Gross, H.-M. (2005b). A new Control Architecture for Mobile Interaction Robots. In *Proceedings of the 2nd European Conference on Mobile Robots (ECMR 2005)*, Seiten 224–229.

[Martínez Mozos et al., 2005] Martínez Mozos, O., Stachniss, C., und Burgard, W. (2005). Supervised Learning of Places from Range Data using Adaboost. In *Proeedings. of the IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 1742–1747.

[Merten, 2006] Merten, K. (2006). Robuste Selbstlokalisation für einen mobilen Roboter durch Online-Kartenadaption. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Merten und Gross, 2008] Merten, M. und Gross, H.-M. (2008). Highly adaptable hardware architecture for scientific and industrial mobile robots. In *Proceedings of the IEEE International Conference on Robotics, Automation and Mechatronics (RAM 2008)*, Seiten 1130–1135.

[MetraLabs, 2008] MetraLabs (2008). MetraLabs GmbH, Ilmenau, Deutschland. Website. http://www.metralabs.com/.

[Metropolis und Ulam, 1949] Metropolis, N. und Ulam, S. (1949). The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):333–341.

[Milstein, 2005] Milstein, A. (2005). Dynamic Maps in Monte Carlo Localization. In *Proceedings of the 18th Canadian Conference on Artificial Intelligence*, Seiten 1–12.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., und Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Seiten 593–598.

[Montemerlo et al., 2003] Montemerlo, M., Thrun, S., Koller, D., und Wegbreit, B. (2003). FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Seiten 1151–1156.

[Moravec, 1988] Moravec, H. (1988). Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2):61–77.

[Moravec, 1996] Moravec, H. (1996). Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids. Technical Report CMU-RI-TR-96-34, Carnegie Mellon University, Pittsburgh.

[Moravec und Elfes, 1985] Moravec, H. und Elfes, A. (1985). High Resolution Maps from Wide Angle Sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 116–121.

[Murphy, 1999] Murphy, K. P. (1999). Bayesian Map Learning in Dynamic Environments. In *Advances in Neural Information Processing Systems 12 (NIPS)*, Seiten 1015–1021.

[Müller et al., 2007a] Müller, S., Schaffernicht, E., Scheidig, A., Böhme, H.-J., und Gross, H.-M. (2007a). Are you still following me? In *Proceedings of the 3rd European Conference on Mobile Robots (ECMR)*, Seiten 211–216.

[Müller et al., 2007b] Müller, S., Scheidig, A., Ober, A., und Gross, H.-M. (2007b). Making Mobile Robots Smarter by Probabilistic User Modeling and Tracking. In *Proc. 52nd Int. Scientific Colloquium, Ilmenau*, Seiten 451–456.

[Nguyen et al., 2005] Nguyen, V., Martinelli, A., Tomati, N., und Siegwart, R. (2005). A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Seiten 1929–1934.

[Nourbakhsh et al., 2003] Nourbakhsh, I., Kunz, C., und Willeke, T. (2003). The Mobot Museum Robot Installations: A Five Year Experiment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Seiten 3636 – 3641.

[Oriolo et al., 1997] Oriolo, G., Ulivi, G., und Vendittelli, M. (1997). Fuzzy maps: A new tool for mobile robot perception and planning. *Journal of Robotic Systems*, 14(3):179 – 197.

[Particle Swarm Central, 2008] Particle Swarm Central (2008). Particle Swarm Central. Website. http://www.particleswarm.info/.

[Pöschl et al., 2009] Pöschl, S., Döring, N., und Böhme, H.-J. (2009). Mensch-Roboter-Interaktion im Baumarkt - Formative und summative Evaluation eines mobilen Shopping-Roboters. *Zeitschrift für Evaluation*, 8(1):27–58.

[Pöschl et al., 2008] Pöschl, S., Döring, N., Böhme, H.-J., und Martin, C. (2008). Computergestützte Artikelsuche im Baumarkt - Formative Evaluation eines Artikelsuchsystems für mobile Shopping-Roboter. *Zeitschrift für Evaluation*, 7(1):113–135.

[Ranganathan und Dellaert, 2006] Ranganathan, A. und Dellaert, F. (2006). A Rao-Blackwellized particle filter for topological mapping. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, Seiten 810 – 817.

[Rosten und Drummond, 2006] Rosten, E. und Drummond, T. (2006). Machine learning for high-speed corner detection. In *Proceedings of the 2006 European Conference on Computer Vision*, Seiten 430–443.

[Rottmann et al., 2005] Rottmann, A., Martínez Mozos, O., Stachniss, C., und Burgard, W. (2005). Place Classification of Indoor Environments with Mobile Robots using Boosting. In *Proceedings of the National Conference on Artificial Intelligence*, Seiten 1306–1311.

[Rubin, 1987] Rubin, D. (1987). The SIR algorithm. Discussion of "The calculation of posterior distributions by data augmentation"by Tanner and Wong. *Journal of the American Statistical Association*, 82(398):543–546.

[Rubin, 1988] Rubin, D. (1988). Using the SIR algorithm to simulate posterior distributions. In *Bayesian Statistics 3*, Seiten 395–402. Oxford University Press.

[Schenderlein, 2007] Schenderlein, M. (2007). Outdoor-Roboternavigation und Lokalisation in interaktiv erstellten topologischen Graphen mit Omnivision-Attributierung unter Einsatz probabilistischer Zustandsschätzer. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Schmiegel, 2004] Schmiegel, A. (2004). Pfadplanung und Bewegungssteuerung für einen mobilen Roboter unter Berücksichtigung kinema-

tischer Eigenschaften. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Schröter, 2004] Schröter, C. (2004). Robust Map Learning with Low-Cost Sensors for an Autonomous Robot. In *Proceedings of the 3rd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE 2004)*, Seiten 167–177.

[Schröter et al., 2003] Schröter, C., Böhme, H.-J., und Gross, H.-M. (2003). Extraction of Orientation from Floor Structure for Odometry Correction in Mobile Robotics. In *Proceedings of the 25th Pattern Recognition Symposium (DAGM 2003)*, Seiten 410–417.

[Schröter et al., 2004] Schröter, C., Böhme, H.-J., und Gross, H.-M. (2004). Robust Map Building for an Autonomous Robot using Low-Cost Sensors. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2004)*, Seiten 5398–5403.

[Schröter et al., 2007a] Schröter, C., Böhme, H.-J., und Gross, H.-M. (2007a). Memory-Efficient Gridmaps in Rao-Blackwellized Particle Filters for SLAM using Sonar Range Sensors. In *Proceedings of the 3rd European Conference on Mobile Robots (ECMR 2007)*, Seiten 138–143.

[Schröter und Gross, 2007] Schröter, C. und Gross, H.-M. (2007). Efficient Gridmaps for SLAM with Rao-Blackwellized Particle Filters. In *Proceedings of the 52nd International Scientific Colloquium (IWK)*, Seiten 445–450.

[Schröter und Gross, 2008] Schröter, C. und Gross, H.-M. (2008). A sensor-independent approach to RBPF SLAM - Map Match SLAM applied to Visual Mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*, Seiten 2078 – 2083.

[Schröter et al., 2007b] Schröter, C., Höchemer, M., und Gross, H.-M. (2007b). A Particle Filter for the Dynamic Window Approach to Mobile Robot Control. In *Proceedings of the 52nd International Scientific Colloquium (IWK)*, Seiten 425–430.

[Schröter et al., 2005] Schröter, C., König, A., Böhme, H.-J., und Gross, H.-M. (2005). Multi-Sensor Monte-Carlo-Localization Com-

bining Omnivision and Sonar Range Sensors. In *Proceedings of the 2nd European Conference on Mobile Robots (ECMR 2005)*, Seiten 164–169.

[Schröter et al., 2002] Schröter, C., König, A., und Gross, H.-M. (2002). Farbbild-basiertes Monte-Carlo-Lokalisationsverfahren für mobile Roboter. In *Proceedings 8. Workshop Farbbildverarbeitung*, Seiten 111–118.

[Schultz et al., 1999] Schultz, A., Adams, W., und Yamauchi, B. (1999). Integrating Exploration, Localization, Navigation and Planning with a Common Representation. *Autonomous Robots*, 6(3):293–308.

[SeeGrid, 2008] SeeGrid (2008). SeeGrid Corporation, Pittsburgh, USA. Website. http://www.seegrid.com.

[Shi und Tomasi, 1994] Shi, J. und Tomasi, C. (1994). Good Features to Track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 593 – 600.

[Shiomi et al., 2007] Shiomi, M., Kanda, T., Ishiguro, H., und Hagita, N. (2007). Interactive Humanoid Robots for a Science Museum. *IEEE Intelligent Systems*, 22(2):25–32.

[Siegwart et al., 2003] Siegwart, R., Arras, K. O., Bouabdallah, S., Burnier, D., Froidevaux, G., Greppin, X., Jensen, B., Mayor, A. L. L., Meisser, M., Philippsen, R., Piguet, R., Ramel, G., Terrien, G., und Tomatis, N. (2003). Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3-4):203–222.

[Simmons et al., 2003] Simmons, R. G., Goldberg, D., Goode, A., Montemerlo, M., Roy, N., Sellner, B., Urmson, C., Schultz, A. C., Abramson, M., Adams, W., Atrash, A., Bugajska, M. D., Coblenz, M., MacMahon, M., Perzanowski, D., Horswill, I., Zubek, R., Kortenkamp, D., Wolfe, B., Milam, T., und Maxwell, B. A. (2003). GRACE: An Autonomous Robot for the AAAI Robot Challenge. *AI Magazine*, 24(2):51–72.

[Smith und Gelfand, 1992] Smith, A. F. M. und Gelfand, A. E. (1992). Bayesian Statistics without Tears: A Sampling-Resampling Perspective. *The American Statistician*, 46(2):84–88.

[Smith et al., 1990] Smith, R., Self, M., und Cheeseman, P. (1990). Estimating Uncertain Spatial Relationships in Robotics. In *Autonomous Robot Vehicles*, Seiten 167 – 193. Springer Verlag.

[Smith und Cheeseman, 1986] Smith, R. C. und Cheeseman, P. (1986). On the Representation and Estimation of Spatial Uncertainty. *International Journal on Robotics Research*, 5(4):56–68.

[Sorenson, 1970] Sorenson, H. (1970). Least-Squares Estimation: from Gauss to Kalman. *IEEE Spectrum*, 7:63–68.

[Stachniss und Burgard, 2002] Stachniss, C. und Burgard, W. (2002). Mobile Robot Mapping and Localization in Non-Static Environments. In *Proceedings of the 20th National Conference on Artificial Intelligence*, Seiten 1324–1329.

[Stachniss und Burgard, 2005] Stachniss, C. und Burgard, W. (2005). An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments. In *IEEE/RSJ International Conference on Intelligent Robots and System (IROS 2002)*, Seiten 508–513.

[Thrun et al., 1999] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J., und Schulz, D. (1999). MINERVA: a second-generation museum tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 1999–2005.

[Thrun und Buecken, 1996] Thrun, S. und Buecken, A. (1996). Integrating Grid-Based and Topological Maps for Mobile Robot Navigation. In *Proceedings of the National Conference on Artificial Intelligence*, Seiten 944–950.

[Thrun et al., 2005] Thrun, S., Burgard, W., und Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[Thrun et al., 2002] Thrun, S., Koller, D., Ghahramani, Z., Durrant-Whyte, H., und A.Y., N. (2002). Simultaneous Mapping and Localization With Sparse Extended Information Filters. In *Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics*.

[Thrun et al., 2000] Thrun, S. B., Beetz, M., Bennewitz, M., Burgard, W. A., Cremers, A. B., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N., Schulte, J., und Schulz, D. (2000). Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *Journal of Robotic Research*, 19(11):972–999.

[Torralba et al., 2003] Torralba, A., Murphy, K., Freeman, W., und Rubin, M. (2003). Context-based vision system for place and object recognition. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, Seiten 273–280.

[Täubig, 2004] Täubig, H. (2004). VSLAM - Simultaneous Localization and Mapping - Ein probabilistischer Modellierungsansatz mit visuellen Features. Diplomarbeit, FG Neuroinformatik und Kognitive Robotik, Technische Universität Ilmenau.

[Täubig und Schröter, 2004] Täubig, H. und Schröter, C. (2004). Simultanous Localization and Mapping (SLAM) mit Hierarchischen Partikelfiltern. In *Proceedings of the 3rd Workshop on Self-Organization of AdaptiVE Behavior (SOAVE 2004)*, Seiten 157–166.

[Ulrich und Borenstein, 1998] Ulrich, I. und Borenstein, J. (1998). VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Seiten 1572 – 1577.

[Ulrich und Nourbakhsh, 2000] Ulrich, I. und Nourbakhsh, I. (2000). Appearance-based place recognition for topological localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Seiten 1023–1029.

[Videre Design, 2008] Videre Design (2008). Videre Design LLC, Menlo Park, USA. Website. http://www.videredesign.com/.

[Viola und Jones, 2001] Viola, P. und Jones, M. (2001). Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade. In *Advances in Neural Information Processing Systems 14 (Neural Information Processing Systems: Natural and Synthetic, NIPS 2001)*, Seiten 1311–1318.

[Wilhelm, 2005] Wilhelm, T. (2005). *Methoden der vision-basierten Nutzerwahrnehmung für eine natürliche Interaktion mit mobilen Servicerobotern*. PhD thesis, Technische Universität Ilmenau.

[Wilhelm et al., 2004] Wilhelm, T., Backhaus, A., Böhme, H.-J., und Gross, H.-M. (2004). Statistical and Neural Methods for Vision-based Analysis of Facial Expression and Gender. In *Proc. 2004 IEEE Int. Conf. on Systems, Man and Cybernetics (SMC 2004)*, Seiten 2203–2208.

[Wilhelm et al., 2002a] Wilhelm, T., Böhme, H.-J., und Gross, H.-M. (2002a). Sensorfusion for Visual and Sonar based People Tracking on a mobile Servicerobot. In *Proc. Int. Workshop on Dynamic Perception 2002*, Seiten 315–320.

[Wilhelm et al., 2002b] Wilhelm, T., Böhme, H.-J., und Gross, H.-M. (2002b). Wechselseitige Unterstützung von sonarbasiertem und visuellem Personentracking. In *Robotik 2002: Leistungsstand - Anwendungen - Visionen - Trends*, Seiten 575–580.

[Wilhelm et al., 2003a] Wilhelm, T., Böhme, H.-J., und Gross, H.-M. (2003a). Looking Closer. In *Proc. European Conference on Mobile Robots (ECMR 2003)*, Seiten 65–70.

[Wilhelm et al., 2003b] Wilhelm, T., Böhme, H.-J., und Gross, H.-M. (2003b). Towards an Attentive Robotic Dialog Partner. In *Proc. Fifth International Conference on Multimodal Interfaces (ICMI-PUI' 03)*, Seiten 297–300.

[Wilhelm et al., 2005] Wilhelm, T., Böhme, H.-J., und Gross, H.-M. (2005). Classification of Face Images for Gender, Age, Facial Expression, and Identity. In *Artificial Neural Networks: Biological Inspirations - ICANN 2005*, Seiten 569–574.

[World Robotics, 2008] World Robotics (2008). International Federation of Robotics (IFR) - Annual World Robotics Study. Website. http://www.worldrobotics-online.org/.

[Yang et al., 2004] Yang, S. X., Zhu, A., und Meng, M. Q.-H. (2004). Biologically Inspired Tracking Control of Mobile Robots with Bounded Accelerations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 1610 – 1615.

[Zimmer, 2000] Zimmer, U. (2000). Embedding local metrical map patches in a globally consistent topological map. In *Proceedings of the 2000 International Symposium on Underwater Technology*, Seiten 301–305.