Globale Selbstlokalisation autonomer mobiler Roboter

Ein Schlüsselproblem der Service-Robotik

Dissertation
zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians Universität
Würzburg

vorgelegt

von **Dirk Schäfer**

aus Eschwege

Würzburg im September 2003

Eingereicht am: 10.09.2003

bei der Fakultät für Mathematik und Informatik

1. Gutachter: Prof. H. Noltemeier, Würzburg

2. Gutachter: Prof. K. Schilling, Würzburg

Tag der mündlichen Prüfung: 23.12.2003

Für Heike, Helga und Wilhelm

Danksagung

Die hier vorliegende Dissertation wurde teilweise durch die Deutsche Forschungsgemeinschaft (DFG) im Rahmen des Schwerpunktprogrammes »Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen« (No 88/14-2 und No 88/14-3) gefördert.

Herzlich Bedanken möchte ich mich besonders bei den zahlreichen Studenten, die wesentlich mit ihrem Engagement durch Diplomarbeiten, Projektpraktika oder Hiwi-Tätigkeit zum Erfolg des Projektes beigetragen haben. Die langen Diskussionen und oft mühsamen praktischen Tests haben diese Arbeit erst ermöglicht!

In zeitlicher Abfolge sind dies Matthias Schwark, Boris Kluge, Martin Buck, Reinhold Almen, Florian Mayer, Thomas Heilmann, Stefan Hieß, Jürgen Roder, Simon Osiander, Bernhard Schwarz, Stefan Gillert, Matthias Maar, Andreas Pokorny, Oliver Gerber, Florian Fischer, David Brand, Fabian Müller.

Diese Arbeit wäre ohne technische Unterstützung bei den anfallenden Hardwareproblemen nicht machbar gewesen deshalb auch weiterer Dank an Yosry Morsi und Rudolf Benedikt.

Zuletzt bedanke ich mich bei Prof. Hartmut Noltemeier für die Betreuung dieser Arbeit.

Vorwort

Diese Arbeit beschäftigt sich mit autonomen mobilen Robotern. Darunter kann man sich mobile, mit Rädern ausgestattete Computer mit angeschlossenen Sensoren vorstellen. Alles was der Roboter zum Funktionieren benötigt, ist auf dem mobilen Gerät installiert. Prominentes Beispiel autonomer mobiler Roboter ist zum Beispiel der Mars-Rover, der vor einigen Jahren über Wochen hinweg in den Schlagzeilen war.

Das Problem der globalen Selbstlokalisation kann man umgangssprachlich folgendermaßen beschreiben: Ein mobiler Roboter befindet sich in einer bekannten Einsatzumgebung innerhalb eines Gebäudes und besitzt darüber eine Karte, allerdings fehlt im die tatsächliche Position, an welcher er sich befindet. Der Roboter möchte nun seine Position anhand aufgenommener Sensordaten und anhand von Bewegungen innerhalb der Einsatzumgebung bestimmen.

Dieses Problem ist Menschen nur allzu vertraut. Wenn Sie sich innerhalb einer fremden Stadt verlaufen haben, versuchen Sie Ihre Position mit Hilfe eines Stadtplanes zu bestimmen. Dabei helfen Ihnen Straßennamen. Die Einsatzumgebung des Roboters stellt jedoch keine solche Landmarken zur Verfügung, sondern verlangt einen Datenabgleich aufgenommener Daten mit denen der Umgebungskarte.

Eine Lösung dieses Problems ist für mobile Roboter wichtig, kann doch ein Positionsverlust aufgrund fehlerhafter Daten in Gebäude-Einsatzumgebungen auftreten. Mit einer Problemlösung können Fehlerzustände bisheriger mobiler Systeme behoben werden, so dass ein robusteres Roboter-System entwickelt werden kann. Erst dann, wenn alle denkbaren Fehlerfälle mobiler Roboter gelöst sind, werden Service-Roboter Einzug in unsere Haushalte finden.

Die hier in dieser Arbeit besprochene Lösung involviert fast alle in der mobilen Robotik auftretenden Probleme. So müssen effiziente Matchingalgorithmen entworfen werden, welche Echtzeit-Bedingungen genügen, um Sensordaten auf Kartenmodelle abzubilden. Kartenmodelle der Einsatzumgebung müssen erstellt werden, um hochgenaues Datenmaterial für das Matching zur Verfügung zu haben. Als Resultat tauchen meistens mehrere mögliche Positionen auf, an denen sich der Roboter befinden könnte, sogenannte Hypothesen. Um diese Mehrdeutigkeiten zu eliminieren, müssen neue Daten akquiriert werden. Dies geschieht, indem der Roboter in der Umgebung umherfährt. Dafür müssen Navigationsalgorithmen entworfen werden, um Bewegungen

innerhalb der Einsatzumgebung zu ermöglichen.

Alle diese Probleme werden in dieser Arbeit angesprochen und Lösungen aus der Literatur präsentiert, modifiziert und im Gesamtkontext präsentiert. Mit diesen Voraussetzungen wird dann der EFM-Lösungsalgorithmus in dieser Arbeit präsentiert, welcher eine immer größer werdende lokale Karte, erstellt aus Sensordaten, auf das Kartenmodell abbildet. Dabei werden Hypothesenmengen über die Zeit verfolgt und bewertet und aufgrund der Matchingergebnisse bestärkt oder vermindert, so dass am Ende des Prozesses die Position des Roboters festgestellt wird.

Mit der Lösung des globalen Selbstlokalisationsproblems wird in dieser Arbeit gleichzeitig eine Zerlegung der verschiedenen Problemstellungen in Teilprobleme erreicht, so dass als Konsequenz ein Baukasten von Lösungsalgorithmen zur Verfügung gestellt wird, mit Hilfe dessen Service-Roboter generisch aufgebaut werden können.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Grundlegende Definitionen . . . 3
1.1.1 Mathematische Systemtheorie . . . 3
1.1.2 Regelung . . . 5
1.1.3 Koordinatensysteme . . . 6
1.2 Modellierung eines Roboter-Systems . . . 6
1.2.1 Zustand und Konfiguration . . . 6
1.2.2 Objekte und Hindernisse . . . 7
1.2.3 Umweltmodellierung, Aktorik, Sensorik . . . 8
1.3 Das Problem der Globalen Selbstlokalisation . . . 9
1.4 Ziele der Arbeit und Übersicht . . . 11

I Generierung von Lokalisations-Hypothesen **13**

2 Feature-basierte Lokalisation **15**

2.1 Problemstellungen der Lokalisation . . . 15
2.1.1 Lösungsansätze des Lokalisationsproblems . . . 17
2.1.2 Dekomposition der Problemstellung . . . 19
2.2 Modellierung und Extraktion lokaler Features . . . 20
2.2.1 Modellierung von lokalen Features . . . 20
2.2.2 Feature-Extraktion aus Scan-Sensordaten . . . 25
2.2.3 Feature-Extraktion aus Kartenmodellen . . . 31
2.3 Matching lokaler Features . . . 36
2.3.1 Alignment . . . 37
2.3.2 Pose Clustering . . . 43
2.3.3 Geometrisches Hashing . . . 48
2.4 Transformationsbestimmung . . . 55
2.4.1 Transformationsbestimmung aus wenigen Zuordnungen . . . 55
2.4.2 Transformationsbestimmung aus größeren Matchings . . . 57
2.5 Verifikation durch Projektion . . . 59
2.6 Algorithmischer Ablauf der featurebasierten Lokalisation . . . 60
2.7 Bewertung von Hypothesen und Lokalisationsverfahren . . . 61
2.7.1 Güte von Hypothesen . . . 61
2.7.2 Güte von Lokalisationsverfahren . . . 62

2.8 Verfahrensvergleich in synthetischen Umgebungen 62
2.8.1 Anforderungen an die simulierten Daten 62
2.8.2 Ergebnisse mit synthetischen Daten 64
2.9 Resümee 64

3 Generierung feature-basierter Karten 67
3.1 Problemstellungen der Kartierung 68
3.1.1 Kartenrepräsentationen 68
3.1.2 Topologisch korrekte und konsistente Karten 72
3.1.3 Das Problem der feature-basierten Kartierung 74
3.1.4 Dekomposition der Problemstellung 75
3.2 Korrektur zweidimensionaler Sensordaten 78
3.2.1 Lösungsansätze in der Literatur 79
3.2.2 Modellierung eines inkrementellen probabilistischen Ansatzes . 81
3.2.3 Statische Modellierung der Umgebung 83
3.2.4 Dynamische Modellierung der Umgebung 90
3.2.5 Algorithmus der inkrementellen probabilistischen Kartierung . . 93
3.3 Experimentelle Evaluation der inkrementellen Kartierung 94
3.3.1 Testumgebung 1: Dynamische Objekte 95
3.3.2 Testumgebung 2: Große Zyklen 97
3.3.3 Testumgebung 3: Multiple Räume 99
3.3.4 Testumgebung 4: Kleine Zyklen 101
3.3.5 Vergleich der Ansätze 101
3.4 Erzeugen von Segmentkarten 103
3.4.1 Lösungsansätze in der Literatur 103
3.4.2 Feature-basierte Kartierung durch Kurvenrekonstruktion 104
3.4.3 Feature-basierte Kartierung durch Segment-Merging 109
3.5 Experimentelle Evaluation der Erzeugung von Segmentkarten 113
3.6 Resümee 118

4 Methoden zur Effizienzsteigerung 121
4.1 Reduktion der Bild- und Modellfeatures 121
4.1.1 Bewertung informativer Features 122
4.1.2 Auswahl informativer Features 123
4.1.3 Segmentierung der Featuremenge 125
4.2 Verfahren zur Effizienzsteigerung 130
4.2.1 Schnelle feature-basierte Verfahren 130
4.2.2 Kombination feature-basierten Verfahren 132
4.2.3 Integration von Vorwissen 133
4.3 Anwendung der Methoden und Performancestudien 133
4.3.1 Test 1: Schnelligkeit und Güte des neuen Verfahrens 134
4.3.2 Test 2: Auswirkungen der unterschiedlichen Matching-Verfahren 136
4.3.3 Test 3: Auswirkungen der heuristischen Auswahlverfahren . . . 140
4.4 Resümee 145

II Hypothesenelimination **147**

5 Navigation in lokalen Karten **149**
5.1 Das Problem der Navigation autonomer mobiler Roboter 150
5.1.1 Lösungsansätze in der Literatur 150
5.1.2 Dekomposition des Problems 152
5.2 Pfadplanung vom Start zum Ziel 153
5.2.1 Generierung der Anfangs- und Endparameter 154
5.2.2 Optimalitätskriterien 155
5.2.3 Pfadplanungsansätze 156
5.3 Positionsverfolgung und Positionskorrektur 164
5.3.1 Positionsverfolgung 165
5.3.2 Positionskorrektur 170
5.4 Trajektorienverfolgung und Hindernisvermeidung 171
5.4.1 Generierung von stetigen Trajektorien 171
5.4.2 Dynamische Hindernisvermeidung 173
5.5 Lösungsalgorithmus des Navigationsproblems 175
5.6 Resümee 177

6 Hypothesenelimination **179**
6.1 Definitionen und Problemstellungen 179
6.1.1 Geometrischer Baum und Overlay-Baum 179
6.1.2 Sensorpunkt und Entscheidungsstrategie 181
6.1.3 Effizienz der Entscheidungsstrategie 181
6.1.4 Definition der Problemstellung 182
6.2 Lösung des statischen Problems 182
6.2.1 Modifizierte Eliminationsstrategien 185
6.2.2 Anforderungen an die Praxis 187
6.3 Lösung des dynamischen Problems 188
6.3.1 Die Monte-Carlo-Lokalisation 188
6.3.2 Die aktive Markov-Lokalisation 188
6.3.3 Multiples Hypothesentracking 191
6.4 Autonome Exploration 193
6.4.1 Problemstellungen und Lösungsansätze 194
6.4.2 Auswahl der Sensorpunkte 195
6.4.3 Auswahl und Erreichen des besten Sensorpunktes 198
6.4.4 Ablauf der Exploration 199
6.5 Elimination durch Exploration und Feature-Matching 200
6.5.1 Idee des Verfahrens 201
6.5.2 Bewertung von Hypothesen 203
6.5.3 Verschmelzen von Hypothesenmengen 205
6.5.4 Abbruchkriterien der globalen Selbstlokalisation 206
6.6 Praktische Ergebnisse und Vergleiche 208
6.6.1 Exploration 208

6.6.2 Selbstlokalisation durch Exploration und Feature-Matching . . . 212
6.6.3 EFM im Informatikgebäude . . . 214
6.6.4 EFM im Mathematikgebäude . . . 215
6.7 Resümee . . . 216

III Entwurf von Service-Robotern in intelligenten Umgebungen 219

7 Mobile Agenten in intelligenten Umgebungen 221
7.1 Ubiquitous Computing . . . 221
7.2 Modellierung einer intelligenten Umgebung . . . 223
7.3 Modellierung einer Mensch-Maschine-Schnittstelle . . . 228
7.4 Modellierung intelligenter Objekte . . . 229
7.5 Resümee . . . 230

8 Modellierung eines Service-Roboters 231
8.1 Modellierung eines Roboter-Systems . . . 231
8.1.1 Systemanforderungen der Software . . . 232
8.1.2 Systemanforderungen der Hardware . . . 237
8.1.3 Designkriterien für ein Roboter-System . . . 237
8.2 Modellierung eines Service-Roboters . . . 238
8.3 Aufgaben und Prozesse eines Service-Roboters . . . 239
8.3.1 Aufgaben und Prozesse eines Roboter-Systems . . . 239
8.3.2 Aufgaben und Prozesse eines Service-Tasks . . . 241
8.4 Resümee . . . 242

9 Service-Roboter in intelligenten Umgebungen 245
9.1 Architektur eines Service-Roboters . . . 245
9.1.1 Architekturmodelle mobiler Roboter . . . 246
9.1.2 Ein Kompromiss: die hybride Architektur . . . 248
9.2 Dekomposition eines Service-Roboter . . . 249
9.2.1 Modellierung des intelligenten Roboter-Objektes . . . 250
9.2.2 Modellierung des Informationsraumes des Roboter-Objektes . . . 251
9.2.3 Mensch-Maschine Schnittstelle als aktives Objekt . . . 252
9.3 Resümee . . . 254

10 Resümee 255
10.1 Zusammenfassung . . . 255
10.2 Rückblick und Bewertung . . . 256
10.3 Beiträge . . . 258

A Statistische Grundlagen 261
A.1 Mittelwert, Varianz, Kovarianz, Kovarianzmatrix . . . 261
A.2 Mahalanobis-Distanz . . . 262
A.3 Dichtefunktion der Normalverteilung . . . 262

A.4 Kalman-Filter 263
A.5 Bedingte Wahrscheinlichkeiten 264
A.6 Die Bayes'sche Formel 265

B Bewertung von Algorithmen **267**

C Komplexitätsklassen **269**

D Sensorik und Aktorik **271**
D.1 Ultraschall-Sensorik 271
D.2 Odometrie-Sensorik 272
D.3 Laser-Sensorik 273
D.4 Aktorik 274
D.5 Eingesetzte mobile Roboter 275

1 Einleitung

Die Robotik ist ein interdisziplinärer Forschungsbereich, den man nicht mehr genau abgrenzen kann. Robotik stellt neben der Bioinformatik das wohl zukünftig bahnbrechendste Forschungsgebiet dar. Allgemein befasst sich die Robotik mit der Forschung an »Robotern« und deren Einsatz in vielfältigen Bereichen des täglichen Lebens.

Der Begriff »Roboter« wurde 1921 von dem tschechoslowakischen Schriftsteller Karel Čapek (tschech. Robota - » Arbeit, Zwangsarbeit, Fronarbeit«) geprägt. Unter dem Begriff Roboter kann man sich heutzutage sehr viel vorstellen, je nach Wissensstand der einzelnen Person. Das liegt daran, dass es tatsächlich keine klare Trennung zwischen verschiedenen Robotertypen gibt und man Definitionen dafür vergeblich sucht.

Eine Unterscheidung im Sinne von Experten erfolgt meist durch den Mobilitätsgrad der Roboter, sodass man die Klassen der stationären Roboter und der mobilen Roboter identifizieren kann. Stationäre Roboter werden einmal montiert und besitzen ihren festen Arbeitsbereich, mobile Roboter dagegen können sich in Einsatzumgebungen bewegen. Man kann mobile Roboter weiter anhand ihres Antriebs und ihrer Einsatzumgebungen klassifizieren. Neben Humanoiden Robotern [22, 157] mit zwei Beinen besteht das Forschungsinteresse hauptsächlich an radangetriebenen Robotern, andere Fortbewegungsmethoden spielen nur eine untergeordnete Rolle. Bei den Einsatzumgebungen kann man zwischen statischen Büroumgebungen oder strukturierten Umgebungen, unstrukturierten räumlichen Umgebungen und natürlichen freien Umgebungen unterscheiden. Das Forschungsinteresse mobiler Roboter beschränkt sich meistens auf Teilprobleme in statischen Umgebungen ohne viele bewegte Hindernisse. Diese statischen Umgebungen zeichnen sich zudem durch einen für Radantriebe günstigen ebenen Boden ohne Steigungen aus. Unstrukturierte Umgebungen haben im Unterschied zu statischen Büroumgebungen ein sehr hohes Maß an Dynamik, erzeugt durch Menschen. Zudem sind die räumlichen Gegebenheiten mit großen offenen Flächen, Fenstern etc. durchsetzt. Natürliche freie Umgebungen besitzen Bodenunebenheiten, Bepflanzungen, unterschiedliche Bodenbeläge. Roboter-Systeme in diesem Bereich, etwa autonome Fahrzeuge, Minenroboter, werden oft als Field-Roboter klassifiziert [237].

Man kann weiter zwischen Industrierobotern, Spielzeugrobotern und Service-Robotern unterscheiden. Dabei sind die Übergänge zwischen diesen Roboterklassen fließend, sogar überschneidend (siehe Abbildung 1). Unter der Klasse der Industrieroboter versteht man in der Produktion eingesetzte Greifarm-Roboter. Diese kann man auch

Abbildung 1: Beispiele für Roboter

Links sieht man einen typischen Industrie-Roboter oder Manipulator, in der Mitte einen teilautonomen mobilen Roboter und rechts einen stationären Tank-Roboter [56].

der Klasse der stationären Roboter zuordnen [51,86,184,185]. Typische Beispiele dieser Roboterklasse sind der ABB-IRB140 [1] oder der Kuka-Roboter KR30 [129].

Vertreter der stark wachsenden jungen Roboterklasse der Spielzeugroboter sind zum Beispiel Lego-Mindstorms und der Sony-Hund Aibo. Sie erfreuen Kinder und vor allem junggebliebene Menschen, haben jedoch keine spezifische Aufgabe. Ihren Bekanntheitsgrad verdanken sie dem RoboCup-Wettbewerb [176].

In dieser Arbeit interessiert vor allem der Typ der Service-Roboter. Diese sind teilautonome oder autonome Systeme, welche eine Service-Tätigkeit für Menschen erledigen. Diese Klasse von Robotern wird zumeist den mobilen Robotern zugeordnet. Typische Anwendungen von Service-Robotern erstrecken sich über alle mögliche Umgebungstypen, etwa Forschungsroboter [224], Museumsroboter [216], autonome Flugsysteme [65,226], autonome Waffensysteme [73] oder autonome Reinigungsroboter [63,135].

Der kommerzielle Erfolg ist bisher nur den Spielzeugrobotern und den Industrierobotern beschieden. Service-Robotern, konzipiert um Menschen alltägliche Aufgaben abzunehmen, ist der Durchbruch noch nicht gelungen. Das Kostenargument, Service-Roboter seien noch zu teuer, kann kaum ernst genommen werden, kosten doch Industrieroboter ein Vielfaches und auch nutzlose Toy-Robots lassen sich trotz des hohen Preises verkaufen. Bei näherer Betrachtung ergibt sich der Grund für die Misere: Die Sensorik, Sensordatenverarbeitung und autonome Energieversorgung mobiler Geräte sowie die Lösungsalgorithmen sind noch nicht in der Lage, in so komplexen Umgebungen zu agieren, dass sie von Menschen akzeptiert werden. Die Schnittstellen zum Menschen sind zu unnatürlich und viele Probleme, etwa das Treppensteigen, sind für mobile Roboter ungelöst. Durch den Komplexitätsgrad der Problemlösungsalgorithmen ergeben sich zudem Systemfehler, die das Vertrauen in solche Systeme mindern [50]. Zum Vergleich: die Mensch-Maschine-Interaktion zwischen einem einfachen stationären Computer und seinem Anwender funktioniert des öfteren auch nicht.

In diesem Kapitel soll der Leser im weiteren die elementarsten Grundlagen der Robo-

tik kennenlernen, um die in dieser Arbeit behandelte Problematik verstehen zu können. Dazu wird er zunächst in die mathematische Systemtheorie entführt, um abstrakt ein System kennenzulernen und wie man ein solches handhabt. Danach wird ein abstraktes Roboter-System und die dazugehörige Einsatzumgebung modelliert. Dem schließt sich die Erklärung der behandelten Problematik der globalen Selbstlokalisation dieser Arbeit an.

1.1 Grundlegende Definitionen

Das Verständnis dieses Kapitels ist unablässig für das Verständnis der weiteren Arbeit. Die mathematische Systemtheorie befasst sich mit der Modellierung sowohl linearer als auch nichtlinearer Systeme. Für diese Arbeit sind jedoch nur die grundlegendsten Begriffe erforderlich, die nachfolgend definiert werden.

1.1.1 Mathematische Systemtheorie

Zielsetzung dieses Kapitels ist die mathematische Beschreibung von dynamischen diskreten Roboter-Systemen, die später modelliert bzw. approximiert und geregelt werden sollen. Zunächst kann man ganz abstrakt ein technisches System beschreiben:

Definition 1.1 (System) *Ein System Σ_E wird beschrieben durch $\Sigma_E = (T, X, U, f)$. Dabei bezeichnet T ein diskretes oder kontinuierliches Zeitintervall. Mit X wird der Zustandsraum des Systems und mit U die Eingabemenge beschrieben. Die Funktion f ist dann eine Abbildung von D_θ nach X, bezeichnet als Zustandsübergangsfunktion. Es gilt dabei $D_\theta \subset \{(\tau, \sigma, x, \omega) \mid \sigma, \tau \in T, \sigma \leq \tau, x \in X, \omega \in U^{[\sigma,\tau)}\}$.*

Definition 1.2 (System mit Ausgabe) *Ein System mit Ausgabe ist gegeben durch ein System Σ_E erweitert um eine Ausgabemenge Y und eine Abbildung $h : T \times X \to Y$, bezeichnet als Beobachtungsfunktion. Das System wird dann beschrieben durch $\Sigma = (T, X, U, Y, f, h)$.*

Es seien im folgenden X,Y und U stets endlich-dimensionale Vektorräume.

Definition 1.3 (Zustand) *Ein Element $x \in X$ bezeichnet man als Zustand.*

Definition 1.4 (dynamisches System) *Ein System heißt dynamisch, falls sich sein Zustand mit der Zeit t verändert. Andernfalls heißt das System statisch oder zeitunabhängig.*

In der Anwendung wird zwischen zwei Hauptgruppen von dynamischen Systemen unterschieden: zeitdiskreten und kontinuierlichen Systemen.

Definition 1.5 (Zeitdiskretes System) *Zeitdiskrete Systeme zeichnen sich dadurch aus, dass der Eingabezeitraum in Zeitschritte unterteilt ist - es sind also nur Eingaben zu genau spezifizierten Zeitpunkten möglich. Dementsprechend existiert auch nur eine zeitdiskrete Ausgabe.*

Die Zustandsübergangsfunktion muss bei zeitdiskreter Ein- und Ausgabe keineswegs diskret sein.

Definition 1.6 (Kontinuierliche Systeme) *Bei kontinuierlichen Systemen kann jeder Zeitpunkt im definierten Zeitintervall angenommen werden. Es existiert dann auch zu jedem Zeitpunkt eine Eingabe und bei Systemen mit Ausgabe auch eine Ausgabe.*

Definition 1.7 (Modellierung kontinuierlicher dynamischer Systeme) *Ein kontinuierliches nichtlineares, dynamisches System kann durch folgende Gleichungen modelliert werden:*

$$\dot{x}_t = f(t, x_t, u_t, \theta),$$

$$y_t = h(t, x_t, \theta).$$

Hierbei bezeichnet f *die Zustandsübergangsfunktion,* h *die Beobachtungsfunktion.* θ *ist dabei ein Parametervektor,* t *bezeichnet die Zeit. Der Vektor* x *bezeichnet den Systemzustand des dynamischen Systems,* u *die Eingabe und* y *die Ausgabe.*

Definition 1.8 (Modellierung diskreter dynamischer Systeme) *Ein diskretes nichtlineares, dynamisches System kann durch folgende Gleichungen modelliert werden:*

$$x_{k+1} = f(k, x_k, u_k, \theta),$$

$$y_k = h(k, x_k, \theta).$$

Hierbei bezeichnet f *die Zustandsübergangsfunktion,* h *die Beobachtungsfunktion.* θ *ist dabei ein Parametervektor,* k *bezeichnet den Zeitindex. Der Vektor* x *bezeichnet den Systemzustand des dynamischen Systems,* u *die Eingabe und* y *die Ausgabe.*

Das Problem der Modellierung besteht darin, alle statischen und dynamischen Parameter zu erfassen, die das System beeinflussen. Oft kann man keine Funktionen explizit angeben und ist auf Ein -/ Ausgaberepräsentation beschränkt.

Definition 1.9 (Ein-/Ausgaberepräsentation eines System) *Ein System* Λ *wird beschrieben durch* $\Lambda = (T, U, Y, \lambda)$. *Dabei bezeichnet T ein diskretes oder kontinuierliches Zeitintervall. Mit* U *wird die Eingabemenge und mit* Y *die Ausgabemenge beschrieben.* λ *ist dann eine Abbildung von* D_λ *nach* Y. *Es gilt dabei* $D_\lambda \subset \{(\tau, \sigma, \omega) \mid \sigma, \tau \in T, \sigma \leq \tau, \omega \in U^{[\sigma,\tau)}\}$

Ein Ein-/ Ausgabe-System bezeichnet man auch als gedächtnisloses System.

Definition 1.10 (Zeitinvariantes System) *Ein System heißt zeitinvariant, falls für alle* t, τ, t', x' *und* u *die Bedingungen bestehen:*

1. $h(t + \tau, x, u, \theta) = h(t, x, u, \theta)$
2. $f(t + \tau, t' + \tau, x', u, \theta) = f(t, t', x', u, \theta)$

Dabei ist $t \geq t' \geq 0$ *und* $t' + \tau \geq 0$.

Eine weitere Modellierungsmöglichkeit ergibt sich dadurch, das nichtlineare System in eingeschränkten Bereichen mit linearen Systemen zu approximieren. Ein lineares System kann man folgendermaßen definieren:

Definition 1.11 (Lineares System) *Ein System heißt linear, falls die Mengen* U, X, Y *Vektorräume [140] sind und für jedes feste* $t, t' \in T$ *die Abbildungen*

$$(x', u, \theta) \to f(t, t', x', u, \theta),$$

$$(x, \theta) \to h(t, x, \theta)$$

lineare Abbildungen sind. Andernfalls heißt das System nichtlinear.

1.1.2 Regelung

Regelung ist nichts anderes, als einem realen, typischerweise nichtlinearen System ein Wunschverhalten abzuringen. Als Beispiel kann man hier eine Heizung benennen, die die Raumtemperatur regelt in Abhängigkeit der Witterungsverhältnisse oder die Regelung eines Roboters entlang eines vorberechneten Weges zu einer bestimmten vorgegebenen Position.

Definition 1.12 (Regelsystem) *Ein System* Σ *heißt Regelsystem, falls die Zustandsübergangsfunktion* f *folgende Bedingungen erfüllt:*

1. *(Konsistenz)* $f(t, t', x, u) = x$ *identisch in* t, x, u.
2. *(Kausalität) Es seien* $t_0, t_1 \in T$ *mit* $t_0 < t_1$. *Falls* $u = u'$ *für* $t_0 \leq t < t_1$ *so gilt* $f(t_1, t_0, x, u) = f(t_1, t_0, x, u')$ *für* $x \in X$.
3. *(Halbgruppeneigenschaft) Für jedes* $t_0, t_1, t_2 \in T$ *mit* $t_0 \leq t_1 \leq t_2$, *jedes* x_0 *und jedes* $u \in U$ *gilt* $f(t_2, t_1, f(t_1, t_0, x_0, u), u)) = f(t_2, t_0, x_0, u)$.

In der Kontrolltheorie [106] heißt ein solches System auch Kontrollsystem.

Die Halbgruppeneigenschaft sagt anschaulich folgendes aus: den Zustand des Systems zum Zeitpunkt t_2 kann man erreichen, indem man als Anfangszustand entweder t_0 annimmt oder aber man beginnt mit dem Anfangszustand t_1, der aber als Funktion des Anfangszustands t_0 definiert ist. Ein Zustand beinhaltet also als Vorgeschichte alle anderen Zustände zu früheren Zeitpunkten.

Definition 1.13 (Regelung) *Gegeben sei ein System* Σ. *Falls ein Regelsystem* Σ_{Regler} *existiert, das für das System* Σ *eine Eingabe* u *so generiert, das* Σ *ein gewünschtes Verhalten aufweist, nennt man* Σ_{Regler} *einen Regler. Erfolgt die Regelung für alle Zeitpunkte in vorgegebenen Toleranzgrenzen, so heißt die Regelung stabil.*

Definition 1.14 (Offener Regelkreis) *In einem offenen Regelkreis wird eine Eingabefunktion durch einen externen Prozess generiert. Diese Anordnung heißt auch Vorsteuerung.*

Definition 1.15 (Geschlossener Regelkreis) *In einem geschlossenen Regelkreis wird die Eingabe durch einen externen Prozess* **und** durch eine Rückkopplung der Systemausgabe erzeugt.

1.1.3 Koordinatensysteme

Im Laufe dieser Arbeit wird des öfteren zwischen verschiedenen Koordinatensystemen unterschieden. Man unterscheidet grundsätzlich zwischen einem lokalen Koordinatensystem für ein Objekt und einem Weltkoordinatensystem.

Definition 1.16 (Weltkoordinatensystem) *Ein Weltkoordinatensystem wird durch ein kartesisches Koordinatensystem repräsentiert. Dessen Nullpunkt ist für alle Objekte eindeutig festgelegt.*

Alle Objekte in der Einsatzumgebung können durch Weltkoordinaten beschrieben werden.

Definition 1.17 (Lokales Koordinatensystem) *Ein lokales Koordinatensystem für das Objekt* p *wird repräsentiert durch ein kartesisches Koordinatensystem, dessen Nullpunkt den Schwerpunkt des Objektes repräsentiert. Bezüglich des Weltkoordinatensystems variiert der Nullpunkt des lokalen Koordinatensystems des Objekts* p.

Meistens werden Sensordaten in lokalen Koordinaten angegeben, etwa die Entfernungsdaten eines Laserscans.

1.2 Modellierung eines Roboter-Systems

Abhängig von der obigen Systematik kann man nun ein Roboter-System folgendermaßen definieren:

Definition 1.18 (Roboter-System) *Ein Roboter-System* $\mathcal{A}$ *ist ein diskretes nichtlineares dynamisches System* (T, X, U, Y, f, h)*, wobei* $Y = X$ *und* $h = id$ *gilt. Die Eingabe* U *wird durch Sensordaten realisiert.*

Das Roboter-System $\mathcal{A}$ agiert in einer Einsatzumgebung $X \subset R^N$, wobei $N = 2, 3$, welche als **Arbeitsraum des Roboters** bezeichnet wird. In dieser Einsatzumgebung interessiert die Beschreibung des Zustands des Roboter-Systems. Im nachfolgenden werden nur noch mobile Roboter betrachtet. Für diese Klasse von Roboter-Systemen kann man die folgende Zustandsbeschreibung liefern:

1.2.1 Zustand und Konfiguration

Definition 1.19 (Zustand, Konfiguration) *Der Zustand* x_k *des Roboter-Systems* $\mathcal{A}$ *zu einem Zeitpunkt* k *wird durch einen Punkt* $p_{\mathcal{A}}$ *im euklidischen Raum* $\mathbb{E}$ *und durch eine Vorzugsrichtung* $\gamma_{\mathcal{A}}$ *beschrieben. Er kann also modelliert werden als* $x_k = (p_{\mathcal{A}}, \gamma_{\mathcal{A}})$*. Äquivalent zum Begriff Zustand wird auch der Begriff Konfiguration verwendet.*

Die Zustandsübergangsfunktion kann man also auch als Konfigurationsübergangsgleichung bezeichnen.

Definition 1.20 (Konfigurationsraum) *Die Menge aller möglichen Zustände* q *im gegebenen Arbeitsraum* X *wird als Zustandsraum oder Konfigurationsraum* $\mathcal{C}$ *bezeichnet.*

Im Arbeitsraum sollen sich nun ein mobiler Roboter $\mathcal{A}$ und viele Hindernisse $\mathcal{B}_1, \ldots \mathcal{B}_n$ befinden. Das Roboter-System $\mathcal{A}$ wird zu einem Zeitpunkt k durch eine kompakte Teilmenge $\mathcal{A} \subset \mathcal{X}$ des Arbeitsraums beschrieben, welches es überdeckt.

1.2.2 Objekte und Hindernisse

Definition 1.21 (Objekt, Hindernis) *Ein Objekt oder Hindernis des Arbeitsraums wird durch eine kompakte Teilmenge* $\mathcal{B}_i^k \subset X$ *des Arbeitsraums beschrieben.*

Durch minimale Veränderungen im Arbeitsraum soll sich auch die Konfiguration eines Objektes verändern. Durch die obige Feststellung, dass Objekte im Arbeitsraum eine Ausdehnung besitzen, ergeben sich auch nicht gültige, da von anderen Objekten überdeckte, Konfigurationen im Konfigurationsraum.

Zu beachten ist, dass die definierten Hindernisse des Arbeitsraums sowohl stationäre Hindernisse respektive Wände als auch dynamische Hindernisse respektive bewegte Objekte sein dürfen.

Definition 1.22 (Konfigurations-Hindernis) *Jedes Hindernis* $\mathcal{B}_i^k$ *in* X *wird im Konfigurationsraum zum Zeitpunkt* k *als Konfigurations-Hindernis* $\mathcal{CB}_i^k = \{q_k \in \mathcal{C} \mid \mathcal{A}(q_k) \cap \mathcal{B}_i^k \neq \emptyset\}$ *abgebildet.*

Diese Konfigurationen zum Zeitpunkt k sind aus der Sicht des Roboters verbotene Zustände. Ebenso ergibt sich, dass keine zwei Hindernisse gemeinsame Konfigurationen zu einem Zeitpunkt k annehmen dürfen.

Definition 1.23 (C-Hindernisregion) *Die Menge* $\mathcal{C}_{Obst}^k = \cup_{i=1}^q \mathcal{CB}_i^k$ *heißt C-Hindernisregion von* $\mathcal{C}$ *zum Zeitpunkt* k.

Definition 1.24 (Freier Raum) *Die Menge* $\mathcal{C}_{free}^k = \mathcal{C} \backslash \mathcal{C}_{Obst}^k$ *heißt freier Raum des Konfigurationsraums* $\mathcal{C}$ *zum Zeitpunkt* k.

Der Roboter kann in der Regel alle Konfigurationen des freien Raumes einnehmen. Falls die genaue Position des Roboters im freien Raum der Umgebung nicht bekannt ist, kann man unter Umständen aber hypothetische Konfigurationen, sogenannte Hypothesen, angeben:

Definition 1.25 (Hypothese) *Eine Hypothese* h *beschreibt eine mögliche Konfiguration* q *des Roboters im Konfigurationsraum* $\mathcal{C}$ *der Einsatzumgebung. Die Menge aller*

möglicher Konfigurationen des Roboters abhängig von Sensordaten und Umgebungsdaten zu einem Zeitpunkt k *wird als Hypothesenmenge* H^k *bezeichnet.*

1.2.3 Umweltmodellierung, Aktorik, Sensorik

Mit den obigen Definitionen kann man den Begriff einer Karte erklären, wie er umgangssprachlich verwendet wird:

Definition 1.26 (2-D Karte) *Die Konfigurationen der Hindernisse* $\mathcal{B}_1, \ldots, \mathcal{B}_n$ *in einem Arbeitsraum* X*, eingeschränkt auf ihre zweidimensionalen Positionen und die Konfigurationen des freien Raums* $\mathcal{C}_{free}$ *beschränkt auf ihre zweidimensionalen Positionen beschreiben eine 2-D Karte* m_g *des Arbeitsraums.*

Diese Karte wird in dieser Arbeit oft auch als globale oder auch statische Karte bezeichnet. Die Hindernisse und Konfigurationen des freien Raums besitzen keinen Zeitindex, da die Karte als Modell der Einsatzumgebung fungieren soll. Eine aktuelle Karte der Umgebung kann folgendermaßen definiert werden:

Definition 1.27 (Globale 2-D Karte zum Zeitpunkt k**)** *Die Konfigurationen der Hindernisse* $\mathcal{B}_1^k, \ldots, \mathcal{B}_n^k$ *in einem Arbeitsraum* X*, eingeschränkt auf ihre zweidimensionalen Positionen und die Konfigurationen des freien Raums* $\mathcal{C}_{free}^k$ *beschränkt auf ihre zweidimensionalen Positionen beschreiben eine 2-D Karte* m_g^k *des Arbeitsraums zu einem Zeitpunkt* k*.*

Um eine solche Karte zu generieren, benötigt man im Idealfall die tatsächlichen Positionen jedes einzelnen Objekts im Arbeitsraum zu einem Zeitpunkt k. Um die Positionen zu bestimmen, verwendet man Sensoren. Sensoren (siehe auch Anhang D) messen die Zustände einer begrenzten Teilmenge $\mathcal{C}_{\text{local}}$ des Konfigurationsraumes $\mathcal{C}$ zu einem Zeitpunkt k. Die Sensordaten werden mit s_k bezeichnet. Dabei kann eine Konfiguration von einem Objekt verdeckt sein oder auch frei von Objekten sein. Die Sensormessungen sind in der Regel ungenau und mit einem Messfehler behaftet. Eine globale Karte zu einem Zeitpunkt k ist also in den meisten Fällen nicht erstellbar.

Deshalb verwendet man die folgende Heuristik. Man erstellt eine (lokale) Karte zu einem Zeitpunkt k aus den bis dorthin akquirierten Sensordaten $s_1, \ldots, s_k$. Die damit zusammenhängenden Probleme werden in Kapitel 3 besprochen.

Definition 1.28 (Lokale Karte) *Eine lokale Karte* m_k *zu einem Zeitpunkt* k *wird beschrieben durch eine Sequenz von Sensordaten,* $m_k = (s_1, \ldots, s_k)$ *welche einen Teilraum* $\mathcal{C}_{local}$ *des Konfigurationsraumes* $\mathcal{C}$ *repräsentiert.*

Des weiteren möchte man den Arbeitraum des Roboters verändern, indem man Objekt-Positionen oder aber die Position des Roboters selber verändert. Dies geschieht mit sogenannten Aktoren: Aktoren versetzen den Roboter in die Lage, von einer Konfiguration x_k zu einer anderen Konfiguration zum Zeitpunkt x_{k+1} überzuwechseln und

dabei den Zustand von anderen Objekten im Arbeitsraum zu verändern.

Ein Zustandsübergang wird durch die Zustandsübergangsfunktion f des Roboter-Systems realisiert. Dabei ist zu beachten, dass dieser Funktion das sogenannte **kinematische Modell** des Roboters zugrunde liegt. Damit sind dem Roboter natürliche Beschränkungen auferlegt, sodass manche theoretisch mögliche Zustandsübergänge in der Praxis vom Roboter nicht realisiert werden können. Wie bei der Sensorik treten auch bei einem Zustandsübergang Messfehler oder Ungenauigkeiten auf, die auf die Interaktion mit der Umgebung zurückzuführen sind.

Es sei $u_k = a_k = (\Delta d_k, \Delta\theta_k)$ die Eingabe der Zustandsübergangsfunktion. Dabei soll sich das Roboter-System vom ursprünglichen Zustand mit einer kleinen Distanz Δd_k und Winkelabweichung von $\Delta\theta_k$ bewegt haben. Die Zustandsübergangsfunktion hat dann die folgende Form:

$$f(x_k, u_k) = \begin{bmatrix} x_k & + & \Delta d_k \cos\theta_k \\ y_k & + & \Delta d_k \sin\theta_k \\ \theta_k & + & \Delta\theta_k \end{bmatrix} \tag{1.1}$$

Dieser Zustandsübergang ist in der Regel nicht exakt, sodass sich der Zustandsübergang zu einem Zeitpunkt k mit Fehlern folgendermaßen ergibt:

$$f(x_k, u_k) = \begin{bmatrix} x_k & + & (\Delta d_k + e_{trans}) \cos(\theta_k + e_{rot}) \\ y_k & + & (\Delta d_k + e_{trans}) \sin(\theta_k + e_{rot}) \\ \theta_k & + & \Delta\theta_k + e_{rot} \end{bmatrix} \tag{1.2}$$

Dabei repräsentiert der Wert e_{trans} die Translationsungenauigkeit und e_{rot} die Rotationsungenauigkeit. Um diese beiden Werte zu modellieren, werden für einen mobilen Roboter Bewegungsmodelle angegeben (siehe auch Kapitel 3.2.3). Außerdem wird die Eingabe u_k um Sensordaten s_k erweitert zu $u_k = (a_k, s_k)$ um den Zustandsübergang besser abzuschätzen (siehe Kapitel 5.3.1). Die hier betrachteten Roboter und deren Sensorik sind ausführlicher in Anhang D beschrieben.

1.3 Das Problem der Globalen Selbstlokalisation

Nachdem die grundlegenden Begriffe für die Robotik erarbeitet worden sind, kann man sich dem Thema dieser Arbeit widmen. Hier soll das Problem der globalen Selbstlokalisation mobiler Roboter betrachtet und effizient gelöst werden. Man kann das zugrundeliegende Problem folgendermaßen definieren:

Problem 1.29 (Globale Selbstlokalisation) *Ein mobiler Roboter $\mathcal{A}$, operierend in einem Arbeitsraum X besitzt lediglich eine statische Karte m_g der Einsatzumgebung. Der Roboter ist mit einem Laser-Entfernungsmesser, einem Kompass und Odometrie-Sensoren ausgestattet. Über die bisherige Position ist dem Roboter-System nichts bekannt, es befindet sich an irgendeiner Konfiguration des Arbeitsraums. Gesucht ist die*

tatsächliche Konfiguration des Roboter-Systems, die mit Hilfe der vorhandenen Sensorik und Aktorik effizient bestimmt werden soll.

Das Problem wurde 1995 von Guibas et al. vorgestellt [88] und eine theoretische Lösung samt Komplexitätsabschätzung (siehe dazu auch Anhang B zur Aufwandsabschätzung für Algorithmen) angegeben. Der Ansatz beschränkt sich auf polygonale Umgebungen (siehe auch Kapitel 3). Im Kontext des hergeleiteten Formalismus gibt es dabei keine Hindernisse, der Arbeitsraum wird durch einen zusammenhängenden Rand, bestehend aus Konfigurationen des Hindernisraums, beschrieben. Der umschlossene freie Konfigurationsraum wird dann in zusammenhängende Bereiche unterteilt, welche gemeinsame Sicht besitzen, sogenannte Sichtbarkeitszellen. An jedem Punkt einer solchen Konfiguration kann man eine gemeinsame Menge von ausgezeichneten Konfigurationen sehen. Diese ausgezeichneten Konfigurationen sind bei der Guibas-Lösung die Ecken eines einfachen Polygons, welches den Rand des Arbeitsraums beschreibt.

Der Sichtbereich des Roboters, eine Teilmenge des Arbeitsraums durch einen Sensor wiedergegeben, wird ebenfalls als ein einfaches Polygon modelliert. Dabei kann man in diesem die tatsächlich gesehenen Ecken der Arbeitsumgebung (d.h. ebenfalls ausgezeichnete Konfigurationen) identifizieren und dann nach dem zusammenhängenden Bereich der Karte suchen, welcher die gleiche Menge von ausgezeichneten Konfigurationen sehen kann. Bezeichne n die Anzahl der Ecken des Polygons der Arbeitsumgebung, dann konnte Guibas eine Komplexität von $\mathcal{O}(n^5)$ zur Teillösung des Problems angeben. Die vorgestellte Problemlösung kann als Ergebnis auch mehrdeutige Konfigurationen für den Roboter zurückliefern.

Zur Elimination dieser Mehrdeutigkeiten gibt erst der Artikel von Dudek et al. [62] eine Lösung. Die Autoren stellen einen Greedy-Algorithmus vor, der das Problem der Elimination von Mehrdeutigkeiten in $\mathcal{O}(k^2n^4)$ löst, wobei k die Anzahl der Reflexecken und n die Anzahl der Ecken des einfachen Polygons des Arbeitsaums beschreibt. Der Lösungsalgorithmus geht folgendermaßen vor: Für jede hypothetische Konfiguration des Roboters betrachtet man die zweidimensionalen Roboterkoordinaten als Ursprung einer 2D-Karte und verschiebt diesen auf den Ursprung des globalen Koordinatensystems. Dieser repräsentiert dann alle möglichen Konfigurationen des Roboters eingeschränkt auf die zweidimensionale Position. Alle so entstehenden Karten werden nun übereinandergelegt und deren Sichtbarkeitszellenzerlegungen miteinander geschnitten. Diese Schnitte repräsentieren Bereiche, in denen die Sicht für zwei unterschiedliche Roboterstandorte nicht mehr übereinstimmt, sodass ein Roboter an diesen Schnitten eine oder mehrere Hypothesen eliminieren kann. Der Greedy-Algorithmus wählt nun aus diesen Kanten immer die am nächsten liegende an, bestimmt zu dieser den kürzesten Weg in der Schnittmenge der Kartenrepräsentationen und eliminiert dann mit den entsprechenden Hypothesen auch deren Kartenrepräsentation. Dudek et al. zeigen außerdem, dass eine optimale Lösung des Problems $\mathcal{NP}$-hart ist.

Hier soll ebenso wie zu den klassischen Ansätzen das Hauptproblem in zwei unabhängige zu lösende Teilprobleme zerlegt werden:

Problem 1.30 (Lokalisationsproblem) *Ein mobiler Roboter $\mathcal{A}$ steht an einer beliebigen, aber festen Konfiguration x_k innerhalb des Arbeitsraums X zu einem Zeitpunkt k. Über die Einsatzumgebung besitzt das Roboter-System eine statische Karte m_g. Der Roboter besitzt ferner einen Laser-Entfernungsmesser, um lokale Umgebungsdaten s_k aufzunehmen. Gesucht ist eine Menge von hypothetischen Positionen H^k, welche den Standort des Roboter-Systems mit einer lokalen Repräsentation der Sensordaten in der Umgebung zum Zeitpunkt k wiederspiegelt.*

Problem 1.31 (Hypothesenelimination) *Gegeben sei ein mobiler Roboter $\mathcal{A}$ mit einer einer statischen Karte m_g des Arbeitsraums X und einer Menge von hypothetischen Konfigurationen H^k zu einem Zeitpunkt k, an denen sich der Roboter befinden könnte. Gesucht sind Bewegungsaktionen des Roboter-Systems, um damit alle hypothetischen Konfigurationen aus H^k bis auf die tatsächliche Konfiguration p^* zu eliminieren.*

1.4 Ziele der Arbeit und Übersicht

In dieser Arbeit wird die obige Problematik der globalen Selbstlokalisation bei mobilen Robotern untersucht. Dabei sollen folgende Ziele erreicht werden:

1. Entwicklung eines leistungsfähigen Algorithmus zur Lösung des Problems der globalen Selbstlokalisation. Dabei sollen bisherige theoretische und praktische Resultate mit berücksichtigt werden.

2. Berücksichtigung weiterer Probleme mobiler Roboter, die im Zusammenspiel mit der Problemlösung der globalen Selbstlokalisation benötigt werden.

3. Einbettung der Problemlösung in ein allgemeines Konzept für einen Service-Roboter.

Abhängig von den obigen Aufgaben ergibt sich folgende Vorgehensweise:

Im **ersten Teil** der Arbeit wird die Idee, Bereiche gleicher Sicht zu betrachten durch Bereiche gleicher Features abstrahiert. Dafür müssen Verfahren zur Extraktion von Features und zur Berechnung von Überdeckungen von Features, also Lokalisationsverfahren hergeleitet und auf ihre Einsatzfähigkeit geprüft werden. Dafür müssen weiter feature-basierte Karten durch leistungsfähige Kartierungsverfahren aus Sensordaten erstellt werden. Mit Hilfe von verbesserten schnellen Verfahren ist es möglich, lokale und aktuelle, aus Sensordaten erstellte Karten auf die gegebene statische globale Modellkarte abzubilden und Lokalisationshypothesen für den tatsächlichen Standort des Roboters zu einem Zeitpunkt k zu berechnen.

Im **zweiten Teil** der Arbeit soll sich der Roboter intelligent in seiner Einsatzumgebung bewegen, und damit Lokalisationshypothesen verstärken oder ausschließen. Dabei werden auch die Problematiken der Trajektoriengenerierung, der Positionsverfolgung und der Zielsuche angesprochen und ein neues Lösungsverfahren vorgestellt. Praktische Tests zeigen die Leistungsfähigkeit der vorgestellten Lösung.

Im **dritten Teil** der Arbeit werden die vorgestellten Problematiken in ein Konzept für einen Service-Roboter integriert. Dabei wird eine Brücke zwischen dem Forschungsgebiet »Ubiquitous Computing« und der Robotik geschlagen. Erst mit beiden Ansätzen kann es gelingen, Service-Roboter in unstrukturierten Umgebungen erfolgreich einzusetzen. Dabei wird auch eine allgemeingültige Definition für einen Service-Roboter gegeben und aufgezeigt, welche Aufgaben und Prozesse und damit welche Einsatzmöglichkeiten ein einfaches mobiles System besitzt. Schließlich zeigt sich, das die stabile und sichere Realisierung einer Lösung des globalen Selbstlokalisationsproblems in unstrukturierten räumlichen Umgebungen einen einsatzfähigen Service-Roboter zur Folge hat. Die globale Selbstlokalisation kann somit als ein Schlüsselproblem der Service-Robotik angesehen werden.

Teil I

Generierung von Lokalisations-Hypothesen

Übersicht

Der erste Teil der vorliegenden Arbeit beschäftigt sich mit der Generierung und Korrektur feature-basierter Karten, um mit Hilfe feature-basierter Lokalisationsanfragen in strukturierten und unstrukturierten Einsatzumgebungen Lokalisationsanfragen durchzuführen. Dabei gliedert sich der Ablauf in die drei folgenden Kapitel:

Feature-basierte Lokalisation
Die Entwicklung eines Algorithmus zur Erzeugung von Lokalisations-Hypothesen unter Verwendung von Segmenten als lokale Features steht im Vordergrund des Kapitels. Dazu werden Verfahren zur Extraktion von Features aus Karte und Scan vorgestellt, bekannte Matching-Algorithmen an die Problematik angepasst und Verifikationslösungen zur Gütesicherung vorgestellt. Ein praktischer Vergleich zeigt die Einsatzfähigkeit der Algorithmen.

Erzeugung feature-basierter Karten
In diesem Kapitel werden zweidimensionale Sensordaten zu korrigierten Umgebungskarten zusammengefügt. Aus den korrigierten Sensordaten werden in einem nächsten Schritt feature-basierte Karten berechnet. Alle Verfahren werden abschließend einem Praxistest unterworfen.

Methoden zur Effizienzsteigerung
Die feature-basierten Lokalisationsverfahren werden im letzten Kapitel dieses Abschnittes um Heuristiken erweitert, sodass sehr schnelle Lokalisationsanfragen möglich werden. Dazu werden die Featuremengen auf sogenannte informative Features eingeschränkt und die Umgebungskarte geeignet partitioniert. Auch hier zeigen praktische Tests die Einsatzfähigkeit der Verfahren.

2 Feature-basierte Lokalisation

In diesem Kapitel werden bekannte Algorithmen aus der modellbasierten Objekterkennung auf das noch genauer zu definierende Lokalisationsproblem angewendet. Dabei steht die Lösung folgender Probleme im Mittelpunkt:

Modellierung von Features
: Aus Sensordaten und Umgebungsdaten müssen sogenannte Features, d.h. geometrische Primitive, modelliert werden. Dabei muss vor allem Wert darauf gelegt werden, Unsicherheiten bei der Bestimmung der Features geeignet zu beschreiben.

Realisierung von Matchingverfahren
: Die Feature-Repräsentationen der Bild- und Modelldaten müssen effizient aufeinander abgebildet werden. Dabei müssen die Unsicherheiten der betrachteten Features mit berücksichtigt werden.

Einsatzfähigkeit der Algorithmen
: Die erstellten Algorithmen müssen anhand geeigneter Experimente gegeneinander auf ihre Einsatzfähigkeit hin getestet werden. Dazu sind geeignete Tests zu entwickeln.

Um die obigen Probleme anzugehen, soll zunächst die Problemstellung dieses Kapitels formalisiert werden.

2.1 Problemstellungen der Lokalisation

Das Lokalisationsproblem ist ein Teilproblem des globalen Selbstlokalisationsproblems, welches als Problem 1.29 definiert wurde.

Problem 2.1 (Lokalisationsproblem) *Ein mobiler Roboter befindet sich zu einem Zeitpunkt* k *an einer unbekannten Konfiguration* p^* *innerhalb eines Gebäudes. Über seine Einsatzumgebung steht eine Umgebungskarte* m_g *zur Verfügung. Mit Hilfe der gegebenen Sensorik des Roboters sollen alle Konfigurationen* H^k *in der Karte bestimmt werden, die als möglicher Standort in Frage kommen.*

Dieses Problem ist nicht zu verwechseln mit dem in der Literatur als Lokalisation bezeichneten Problem der Positionverfolgung (siehe Kapitel 5), bei welchem dem Roboter eine Anfangskonfiguration bekannt und somit der Suchraum um diese Konfiguration

herum beschränkt ist. Beim Lokalisationsproblem treten eine Reihe von Problemen auf, die von der Sensorik und der Beschaffenheit der Umgebung abhängen. So kann das Sensorsystem eine beschränkte Reichweite besitzen oder nur einen bestimmten Bereich der Umgebung abtasten. Dabei spielt ferner die Genauigkeit des Sensors wie auch die Abtrastrate eine entscheidende Rolle.

Im Zusammenspiel mit den verschiedenen Umgebungstypen können sich die Probleme noch verstärken, wenn beispielsweise sehr schnelle Objekte in der Umgebung vorhanden sind und die Abtastrate des Sensors relativ gering ist. In der Literatur wurde diese Problematik mit Ultraschall-Sensoren, Laser-Sensoren oder auch Kamera-Sensoren behandelt [52, 68, 189, 203]

Da in dieser Arbeit nachfolgend ein Laser-Entfernungsmesser (siehe auch Anhang D) verwendet wird, lässt sich das Problem folgendermaßen definieren:

Problem 2.2 (Globales Karten-Scan-Matching) *Gegeben sei eine Karte m_g und Sensordaten s_k in Form eines Scans zu einem bestimmten Zeitpunkt k. Ein globales Karten-Scan-Matching berechnet eine Hypothesenmenge H^k als Menge hypothetischer Aufnahmepositionen von s_k.*

Mit der Lösung der Problematik bekommt man eine Menge von hypothetischen Konfigurationen, an denen sich der Roboter aufhalten könnte unter der Voraussetzung nur einen Sensordatensatz s_k zur Verfügung zu haben. Unter der Annahme eines zur Verfügung stehenden Kartierungsverfahrens (siehe Kapitel 3) kann man das Problem für einen beliebigen Zeitpunkt verallgemeinern:

Problem 2.3 (Globales Karten-Karten-Matching) *Gegeben seien eine Karte m_g und eine lokale Karte m_k zu einem Zeitpunkt k. Ein globales Karten-Karten-Matching berechnet eine Hypothesenmenge H^k als Menge hypothetischer Konfigurationen an denen die lokale Karte m_k ihren Ursprung in m_g hat.*

Dabei wird also eine Transformation, welche die lokale Karte m_k auf eine Konfiguration im globalen Koordinatensystem von m_g abbildet und eine Menge von hypothetischen Standorten unter Zuhilfenahme einer Menge von Sensordaten, repräsentiert durch eine lokale Karte, berechnet.

Im Kontext der feature-basierten Lokalisation muss man eine Unterscheidung in Problem 2.2 und Problem 2.3 nicht treffen. Man kann beide Probleme zu folgendem Problem zusammenfassen:

Problem 2.4 (Feature-basiertes Lokalisationsproblem) *Gegeben sei eine Bildfeature-Menge F und eine Modellfeature-Menge G. Die Bildfeature-Menge repräsentiert eine Menge von aktuellen Sensordaten s_k zu einem Zeitpunkt k, die Modellfeature-Menge repräsentiert das Umgebungsmodell, die Umgebungskarte m_g. Das Problem der feature-basierten Lokalisation besteht darin, die Bildfeature-Menge geometrisch*

konsistent auf die Modellfeature-Menge G *abzubilden. Die dabei entstehende Hypothesenmenge* H^k *repräsentiert die Menge der hypothetischen Positionen im Koordinatensystem von* m_g*, auf welche die Bildfeatures* F *abgebildet werden können.*

Bei dieser Definition der Problematik wurde schon ein Vorgriff auf noch zu definierende Termini gemacht. Der Leser sei auf Abschnitt 2.1.1 für ein genaues Verständnis verwiesen.

Mit der obigen Problemstellung 2.4 wurde eine Brücke geschlagen zwischen dem Lokalisationsproblem der Robotik und generellen Matching-Problemen, welche in der Literatur der modellbasierten Objekterkennung beschrieben werden [90, 99, 158, 236].

2.1.1 Lösungsansätze des Lokalisationsproblems

Eine frühere Lösung dieser Problematik wurde im Rahmen unserer Arbeitsgruppe mit Hilfe einer Zerlegung der Karte in Bereiche gleicher Sicht gelöst [89, 116] nach dem grundlegenden Ansatz von Guibas [87]. Eine Lokalisationsanfrage konnte durch die Berechnung eines minimalen Distanzwertes mit Hilfe von Polygonmetriken generiert werden. Nachteile dieser Strategie waren ein Preprocessingaufwand bei Polygonen mit einbeschriebenen Hindernissen von $\mathcal{O}(n^6)$, wobei n die Anzahl der Ecken des Polygons bezeichnet. Für realistische Szenarien war dieser Ansatz völlig ungeeignet.

Eine andere probabilistische Lösungsvariante, die Markov-Lokalisation, beruht auf der Generierung von Hypothesen nur durch Ausnutzung der minimalen Distanz eines Sensorwertes zum nächsten Hindernis [69]. Zwar kann dieser Ansatz erweitert werden um die Verwendung mehrerer Distanzwerte [102], jedoch steigt damit auch der Berechnungsaufwand. Nicht unerwähnt bleiben soll die erweiterte Variante dieses erfolgreichen Ansatzes, die Monte-Carlo Lokalisation [68, 209, 210], welche in Kapitel 5 nochmal aufgegriffen wird. Unerwähnt geblieben sind die Ansätze der modellbasierten Objekterkennung, die meistens mit Hilfe von Kamerabildern und abstrakten Objektbeschreibungen agieren [17, 45].

In diesem Kapitel soll die Problemstellung der Lokalisation aus der Sicht der modellbasierten Objekterkennung gelöst werden [124, 188]. Die Problemstellung 2.4 ist jedoch sehr komplex und wird im übernächsten Abschnitt in seine Teilprobleme zerlegt. Für das Verständnis des Lesers müssen zunächst einige Definitionen vorgeschaltet werden.

Modellierung lokaler Features

In der Problemvariante der modellbasierten Objekterkennung war von Features die Rede. Ein Feature kann man folgendermaßen definieren:

Definition 2.5 (Lokales Feature) *Ein lokales Feature ist ein räumlich beschränktes geometrisches Merkmal eines (exakten) Modells oder eines (ungenauen) Abbildes der Realität.*

Man unterscheidet grundsätzlich zwischen Features aus dem Modell (Modellfeatures) und Features aus dem Abbild der Realität (Bildfeatures), erzeugt durch Sensordaten.

Definition 2.6 (Feature-Zuordnung) *Eine Feature-Zuordnung* fp *ist ein Paar* $(f, g) \in F \times G$*, wobei* F *die Menge der Bildfeatures und* G *die Menge der Modellfeatures ist. Synonym wird auch der Begriff Korrespondenz verwendet.*

Definition 2.7 (Feature-Matching) *Ein Feature-Matching* M *ist eine Teilmenge von* $F \times G$*, wobei* F *die Menge der Bildfeatures und* G *die Menge der Modellfeatures ist. Ein Feature-Matching ist somit eine Menge von Feature-Zuordnungen. Die Menge* $2^{F \times G}$ *aller Matchings zwischen* F *und* G *heißt Korrespondenzenraum.*

Betrachtet wird zunächst allgemein eine Gruppe von Abbildungen, bezeichnet als $\mathcal{TPR}$, welche Bildfeatures auf Modellfeatures abbildet. Die in Frage kommenden Abbildungen werden später genauer definiert.

Definition 2.8 (Indikatorfunktion) *Es seien ein Bildfeature* $f \in F$ *und ein Modellfeature* $g \in G$ *gegeben. Dann heißt die Funktion*

$$\chi : \mathcal{TPR} \times F \times G \to \{0, 1\}$$

Indikatorfunktion. Der Wert der Funktion $\chi(\phi, f, g) = 1$*, falls* f *mittels* $\phi \in \mathcal{TPR}$ *auf* g *abgebildet werden kann, ansonsten 0.*

Die Indikatorfunktion repräsentiert ein Maß dafür, wann die Abbildung zweier Features aufeinander erlaubt ist. Eine Menge von gültigen Feature-Zuordnungen bildet ein Matching M, wobei zu klären ist, welche Matchings erlaubt sind.

Geometrische Konsistenz

Intuitiv ist ein Feature-Matching $M = \{(f_1, g_1), \ldots, (f_k, g_k)\}$ im Kontext der Objekterkennung oder der Lokalisation dann »sinnvoll«, falls eine Abbildung $\phi \in \mathcal{TPR}$ existiert, so dass $\phi(f_i)$ ähnlich zu g_i ist für $1 \leq i \leq k$. Dies drücken die im folgenden definierten Begriffe der induzierten Abbildung und geometrischen Konsistenz aus.

Definition 2.9 (Induzierte Abbildungen) *Sei* $M = \{(f_1, g_1), \ldots, (f_k, g_k)\}$ *ein Feature-Matching. Sei* χ *eine Indikatorfunktion. Die Menge* $\mathrm{Trans}(M)$ *der von* M *induzierten Abbildungen ist wie folgt definiert:*

$$\mathrm{Trans}(M) = \bigcap_{(f,g) \in M} \{\phi \in \mathcal{TPR} \mid \chi(\phi, f, g) = 1\}$$

Die Menge der vom Matching M *induzierten Abbildungen enthält also genau die Transformationen, mit denen jede einzelne Zuordnung aus* M *mit Hilfe der Indikatorfunktion* χ *als passend klassifiziert wird.*

Definition 2.10 (Geometrisch konsistent) *Ein Feature-Matching M heißt geometrisch konsistent (bezüglich einer gegebenen Indikatorfunktion), falls* $\mathrm{Trans}(M) \neq \emptyset$.

Diese Indikatorfunktion ist natürlich von dem verwendeten Featuretyp abhängig, der verwendet wird.

2.1.2 Dekomposition der Problemstellung

Die feature-basierte Lokalisationsproblematik kann man in eine Reihe von Teilproblemen zerlegen, welche in der Literatur modellbasierter Objekterkennungsstrategien vorgenommen wird.

Problem 2.11 (Modellierung und Extraktion von Features) *Gegeben seien Sensordaten s_k und Umgebungsdaten m_g. Dann besteht das Problem darin, aus der Menge der Sensordaten und aus den Umgebungsdaten geometrische Features einer Bildfeature-Menge F und einer Modellfeature-Menge M zu modellieren, die miteinander vergleichbar sind. Die Vergleichbarkeit wird durch eine Modellierung der Ähnlichkeit eines Bildfeatures $f \in F$ zu einem Modellfeature $g \in G$ erreicht.*

Problem 2.12 (Berechnung geometrisch konsistenter Matchings) *Gegeben seien eine Modellfeature-Menge G und eine Bildfeature-Menge F. Das Problem besteht darin, effizient eine Menge $\mathcal{M}$ von Feature-Matchings aus den Mengen F und G zu berechnen.*

Problem 2.13 (Bestimmung der Transformation) *Für alle $M \in \mathcal{M}$ soll aus der von einem Matching $M \in \mathcal{M}$ induzierten Abbildung effizient eine Lokalisationshypothese h^k_{nv} berechnet werden. Die Menge der Lokalisationshypothesen wird mit H^k_{nv} bezeichnet und als Ergebnis berechnet.*

Problem 2.14 (Verifikation des Ergebnisses) *Das Problem besteht darin, die Menge der nicht verifizierten Lokalisationshypothesen H^k_{nv} maximal einzuschränken, sodass möglichst wenige falsche Lokalisationshypothesen und möglichst viele richtige Lokalisationshypothesen erzeugt werden. Als Ergebnis wird eine Menge von Hypothesen H^k berechnet.*

Es ist also zu beachten, dass es einen Unterschied zwischen Lokalisationshypothesen und Hypothesen gibt. Lokalisationshypothesen entstammen der Menge H^k_{nv} und Hypothesen der Menge H^k.

In den nachfolgenden Abschnitten sollen die Teilprobleme jeweils für sich untersucht und auf die Lokalisationsproblematik der betrachteten Roboter und deren Sensoraustattung angepasst werden.

2.2 Modellierung und Extraktion lokaler Features

Die Modellierung geeigneter lokaler Features und deren Extraktion aus den gegebenen Daten ist für ein Lokalisationsverfahren die Schnittstelle zur Außenwelt. Sie bestimmt zu einem großen Teil die Genauigkeit und Anwendbarkeit des Verfahrens. Je schlechter die Extraktion und Transformation der gegebenen Daten in die modellierten Featuretypen ist, desto schlechter und ungenauer ist auch das Ergebnis der Lokalisation.

Mit Hilfe von Heuristiken versucht man den Fehler aufzufangen und geeignete Features zu extrahieren. Dabei müssen diese Verfahren zur Extraktion aufeinander abgestimmt sein, sodass sinnvolle Ergebnisse entstehen.

2.2.1 Modellierung von lokalen Features

Betrachtet man ein Modell oder ein Bild einer 2D-Szene, kann man unterschiedliche einfache oder zusammengesetzte geometrische Primitive identifizieren, die als Features geeignet sind. Im folgenden sind drei von ihnen definiert:

Definition 2.15 (Eckenfeature) *Ein Eckenfeature* f_e *ist ein Schnittpunkt* $p \in \mathbb{E}$ *zweier Kanten* s_i *und* s_j *des betrachteten Szenarios. Ein Eckenfeature wird dann beschrieben durch ein Dreitupel* $f_e = (p, s_i, s_j)$.

Definition 2.16 (Segmentfeature) *Ein Segmentfeature* f_s *beschreibt eine Kante des Modells oder des Bildes. Ein Segmentfeature wird beschrieben durch* $f_s = (p_1, p_2)$ *mit dem Anfangspunkt* $p_1 \in \mathbb{E}^2$ *und dem Endpunkt* $p_2 \in \mathbb{E}^2$.

Definition 2.17 (Rechte-Winkel-Feature) *Ein Rechter-Winkel-Feature* f_r *beschreibt zwei sich schneidende Kanten* s_i *und* s_j *des Modells oder des Bildes, deren eingeschlossener Winkel 90 Grad beträgt. Ein solches Feature wird dargestellt als* $f_r = (s_i, s_j)$.

Annahme: Als lokale Features werden hier nur Segmentfeatures betrachtet, die aus einer Karte (Modellfeature oder Kartenfeature) oder aus einem Scan (Bildfeature oder Scanfeature) extrahiert werden. Andere Features sind denkbar, können aber meistens durch Segmente repräsentiert werden.

Modellierung der Abbildungen

Annahme: Es werden Abbildungen in der Ebene $\mathbb{E}^2$, die die Längen (das heißt auch Winkel) und den Drehsinn erhalten (sogenannte rigid motions) betrachtet, um Bildfeatures auf Modellfeatures abzubilden. Diese lassen sich durch Verkettung einer Rotation um den Ursprung des Koordinatensystems mit einer nachfolgenden Translation beschreiben.

Eine Rotation eines Punktes $p \in \mathbb{C}$ um den Ursprung mit dem Winkel ϕ kann durch eine Multiplikation mit $e^{i\phi}$ dargestellt werden. Eine Verschiebung oder Translation um die Beträge u beziehungsweise v in Richtung der Achse des Real- beziehungsweise Imaginärteils entsprechend einer Addition von $u + iv$ zu p. Insgesamt haben die interessierenden Abbildungen t also folgende Gestalt:

$$t_{\phi,u,v}\ (p) = p \cdot e^{i\phi} + u + iv \text{ für } p \in \mathbb{C} \text{ und } \phi, u, v \in \mathbb{R}$$

Diese Abbildungen bilden zusammen mit der Hintereinanderausführung die Gruppe $\mathcal{TPR}$.

Ähnlichkeit von Features

Mittels einer Metrik kann man die Ähnlichkeit von Features einer Feature-Menge F beschreiben.

Definition 2.18 (Metrik) *Sei F eine Feature-Menge. Die Abbildung $d : F \times F \to \mathbb{R}$ ist ein Distanzmaß für F falls für alle Features $f, g \in F$ die folgenden Bedingungen gelten:*

1. *Positivität:*
   $d(f, g) \geq 0$ *und* $d(f, g) = 0$ *genau dann wenn* $f = g$
2. *Symmetrie:*
   $d(f, g) = d(g, f)$
3. *Dreiecksungleichung:*
   $d(f, h) \leq d(f, g) + d(g, h)$ *für alle* $f, g, h \in F$

Dabei kann man genau quantifizieren, wie ähnlich sich zwei Features sind.

Annahme: Da die Bild- oder Scandaten verrauscht sind und der Laser-Scanner nur Punktmengen liefert, aus denen Segmente extrahiert werden müssen (siehe dazu auch Kapitel 3), muss man einen Fehler beim Matching in Kauf nehmen.

Ähnlichkeiten unter diesen Bedingungen für Features zu definieren, ist schwierig. Betrachtet man zum Beispiel zwei Subsegmente s_1 und s_2 zu einem Segment s, welche untereinander disjunkt sind, so kann man beide optimal auf s abbilden. Allerdings muss man die Symmetrie-Bedingung dann entsprechend definieren, sodass auch s gefahrlos auf s_1 oder s_2 abzubilden ist. Eine einfache Lösung bietet die Verwendung der gerichteten Hausdorff-Distanz:

Definition 2.19 (Gerichtete Hausdorff-Distanz) *Es seien zwei Segment-Features f und g gegeben. Dann berechnet die gerichtete Hausdorff-Distanz d_{hd} die Ähnlichkeit zwischen f und g mittels des maximalen euklidischen Abstands zweier Punkte von f und g.*

Verwendet man das oben definierte Konzept der geometrischen Konsistenz, dann ist für zwei Segment-Features f und g eine Indikatorfunktion χ gesucht, welche determiniert, ob zwei Features ähnlich sind oder nicht. Die modellbasierte Objekterkennung kennt zur Lösung des Problems das Konzept der Kompatibilitätsbereiche, welches Hüllen erlaubter Fehler um die Features legt [7, 167]. Eine Zuordnung wird dann als geometrisch konsistent betrachtet, wenn es eine passende Distanzfunktion zwischen den Features gibt, sodass diese einen Schwellenwert unterschreitet. Mittels der obigen Distanzfunktion d_{hd} ist also eine Indikatorfunktion

$$\chi(\phi, f, g) = 1 \text{ genau dann wenn } d_{hd}(\phi(f), g) < \delta$$

gesucht.

Hier wird für jedes extrahierte Feature ein eigener Kompatibilitätsbereich oder beschränkter Fehler angenommen, da jedes extrahierte Segment mit unterschiedlicher Güte aus Sensordaten gewonnen werden kann.

Berechnung des beschränkten Fehlers

Definition 2.20 (Lokaler beschränkter Fehler) *Jedes lokale Bildfeature $f \in F$ besitzt einen lokalen beschränkten Fehler, der von einem konstanten Grundrauschen und einer - der Entfernung des gemessenen Hindernisses - proportionalen Abweichung abhängt. Der Fehler kann also durch eine lokale Konstante δ_f als beschränkt angenommen werden.*

Um δ_f zu bestimmen und damit festzustellen zu können, wann eine Feature-Zuordnung geometrisch konsistent ist, wird im folgenden eine Fehlerabschätzung für einen einzelnen Sensormesswert durchgeführt und diese dann auf ein aus n Punkten bestehendes Segment übertragen. Betrachte also im folgenden nur einen **einzelnen Messwert**, welcher aus einer unsicheren Messung resultiert. Aus dieser Messung sollen nun maximale Toleranzwerte ermittelt werden um damit tatsächlich entstandene Messwerte von fehlerbehafteten zu unterscheiden.

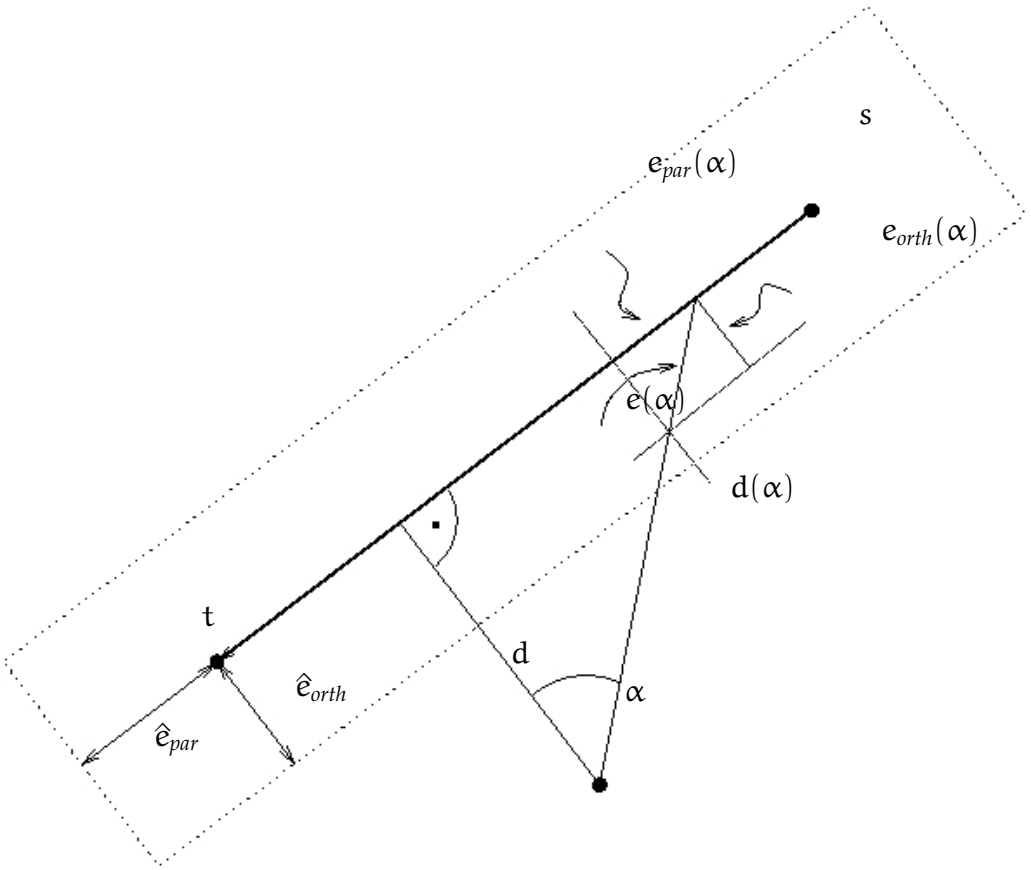
Das Grundrauschen des Sensors e_{abs} und der Rauschanteil e_{rel} sind durch feste, geräteabhängige Konstanten gegeben. Für ein Hindernis in Entfernung d wird also angenommen, dass der Sensor Messwerte aus dem Intervall $[d-(e_{abs}+d\cdot e_{rel}), d+(e_{abs}+d\cdot e_{rel})] \cap \mathbb{R}_{\geq 0}$ liefert (siehe Abbildung 2). Es soll zunächst untersucht werden, wie groß der orthogonale Abstand eines Scanpunktes zum Segment, an dem die Entfernung gemessen wurde, unter diesen Annahmen höchstens werden kann. Betrachte dazu eine Messung $(\alpha, d(\alpha))$ in Richtung α mit $-\pi < \alpha < \pi$ mit dem Abstand $d(\alpha)$. Für den Abstand $d(\alpha)$ gilt

$$d(\alpha) = \frac{d}{\cos \alpha} \tag{2.1}$$

Für Messungen in dieser Richtung ist der Betrag des Fehlers durch

$$e(\alpha) = d(\alpha) \cdot e_{rel} + e_{abs} \tag{2.2}$$

Abbildung 2: Fehlerhülle bei Kartensegment $\overline{st}$



Fehlerhülle des Kartensegments $\overline{st}$. Diese wird bestimmt durch den orthogonalen Fehler e_{orth} und den parallelen Fehler e_{par}.

beschränkt.
Für die weitere Fehlerbetrachtung ist eine Zerlegung des Fehlers in eine orthogonale Komponente e_{orth} und eine parallele Komponente e_{par} notwendig. Dabei gilt

$$e(\alpha) = \sqrt{e_{par}^2 + e_{orth}^2}. \tag{2.3}$$

Die orthogonale Komponente kann also durch

$$e_{orth}(\alpha) = e(\alpha) \cdot \cos\alpha = d \cdot e_{rel} + e_{abs} \cos\alpha \tag{2.4}$$

berechnet werden.

Der maximale Abstand eines Scanpunktes im Bezug auf die Endpunkte des gemessenen Segments parallel zu dessen Ausrichtung kann analog zum obigen Fall des senkrechten Abstands bestimmt werden. Zunächst erhält man

$$e_{par}(\alpha) = e(\alpha) \cdot \sin\alpha = d \cdot e_{rel} \cdot \tan\alpha + e_{abs} \sin\alpha \tag{2.5}$$

Die obigen Abschätzungen gelten jeweils für Messpunkte $p_i = (\alpha_i, d_i), 1 \leq i \leq n$. Da die Segmente aus Messwerten durch Ausgleichsgeraden modelliert werden und o.B.d.A. das betrachtete Segment aus n Punkten hervorging, kann man nun für dieses Segment Fehlergrenzen herleiten.

Man erhält den maximalen orthogonalen Fehler $\hat{e}_{orth}$ durch

$$\hat{e}_{orth} = \max_{p_i \in s} e_{orth}(\alpha_i) \text{ für } 1 \leq i \leq n \tag{2.6}$$

welcher vom Abstand d des Sensors zur Geraden, die das Segment trägt, sowie vom (mit diesem Segment) maximal möglichen Wert für $\cos\alpha$ abhängt. Die Werte e_{rel} und e_{abs} sind Konstanten.

Aus diesem maximalen orthogonalen Messpunktabstand kann außerdem eine Abschätzung für die Winkelabweichung des Segmentfeatures $\overline{st}$ gegenüber dem Hindernissegment abgeleitet werden. Wurde ein Segment mit der Länge l extrahiert, so befindet sich im schlimmsten Fall das eine Ende des Segments um $\hat{e}_{orth}$ vor dem Hindernis, das andere Ende entsprechend hinter dem Hindernis. Der maximale Winkel ϕ_{max} zwischen extrahiertem Segment und Hindernisabschnitt erfüllt also

$$\sin\phi_{max} = \frac{2 \cdot \hat{e}_{orth}}{l} \tag{2.7}$$

Man erhält den maximalen parallelen Fehler $\hat{e}_{par}$ durch

$$\hat{e}_{par} = \max_{p_i \in s} e_{par}(\alpha_i) \text{ für } 1 \leq i \leq n \tag{2.8}$$

Dieser wird maximal für extremale Werte von α_i, das heißt an den Endpunkten des Segmentes $\overline{st}$.

Der lokale Fehler lässt sich nun als das Maximum des maximalen parallelen und orthogonalen Fehlers beschreiben:

$$\delta_f = \max\{\hat{e}_{orth}, \hat{e}_{par}\} \tag{2.9}$$

Annahme: Besitzt man eine individuelle Fehlerabschätzung für ein beliebiges lokales Bildfeature, kann man nun zwei Fälle unterscheiden die für die Bestimmung des δ-Wertes der Indikatorfunktion wichtig sind:

- Das Modellfeature g besitzt keine Fehler
  Damit folgt für eine einzelne Zuordnung zwischen Modellfeature g und Bildfeature f, dass $\delta := \delta_f$ gelten soll.

- Das Modellfeature und das Bildfeature besitzen einen Fehler
  Verwendet man Kartenrepräsentationen, welche mit Methoden aus Kapitel 3 erstellt wurden, besitzen auch die Modellfeatures einen lokal beschränkten Fehler, da sie ja als Bildfeatures extrahiert wurden. Dann gelte $\delta := \max\{\delta_f, \delta_g\}$. Dabei sind δ_f und δ_g lokal beschränkte Fehler von f und g.

Der erste Fall ist im zweiten Fall enthalten, sodass man nun in der Lage ist zu entscheiden, ob zwei Segment-Features $f \in F$ und $g \in G$ geometrisch konsistent sind. Das Maximum zweier lokal beschränkter Fehler zu verwenden, impliziert auch den Sonderfall, dass sich Feature f innerhalb der Fehlerhülle von g befinden kann, aber Feature g außerhalb der Fehlerhülle von Feature f. Dies wurde in Kauf genommen für eine tolerantere Auslegung des Ähnlichkeitsbegriffs.

2.2.2 Feature-Extraktion aus Scan-Sensordaten

Unter idealisierten Annahmen liefert ein Laser-Scanner das Sichtbarkeitspolygon eines Standortes in einer Umgebungskarte. In der Praxis hingegen sind die Winkelauflösung auf $0.5°$ und die Reichweite des Sensors auf 50 Meter beschränkt (SICK LMS 200 Laser-Scanner, siehe Anhang D). Die Sensordaten werden durch einen Scan mittels einer angular geordneten Liste von Sensorpunkten repräsentiert, welche eine 180-Grad-Ansicht des Standorts liefern. Mit diesem Wissen kann man die folgende Problemstellung formulieren:

Problem 2.21 (Segmentfeature-Extraktion aus einem Scan) *Gegeben sei ein Scan mit angular geordneten Messpunkten in der euklidischen Ebene. Gesucht ist eine Menge von Segmenten S, welche die Sensordaten geeignet beschreibt.*

Zunächst soll der Algorithmus zur Linienextraktion vorgestellt werden [91], welcher dann einige Verbesserungen erfährt um die Wahl der Konstanten zu automatisieren. Die Hough-Transformation [61] kann als alternativer Algorithmus eingesetzt werden. Zuletzt soll noch eine Variante vorgestellt werden, mit welcher man die Genauigkeit des verwendeten Umgebungsmodells (die Modellfeatures) auf die relevantesten Features einschränken kann.

Berechnung von Ausgleichsgeraden

Die Linienextraktion (Algorithmus 2.1) teilt die Folge der Scanpunkte in eine Menge von Teilfolgen auf. Für jede Teilfolge wird mit Hilfe des Linienfilters eine Ausgleichsgerade produziert und mittels Algorithmus 2.2 verifiziert und entweder angenommen oder verworfen. Betrachte zunächst eine Folge von n Punkten in der euklidischen Ebene. Das numerische Verfahren zur Ausgleichsgeradenberechnung [199] funktioniert wie folgt. Die Ausgleichsgerade wird so gelegt, dass die Summe der Abstandsquadrate

$$E_{fit} = \sum_{i=0}^{n-1} ((x_i \cos\alpha + y_i \sin\alpha) - d)^2 \tag{2.10}$$

minimiert wird, sodass die Parameter α und d für alle Punkte auf der Geraden gelten. Die Lösungen für α und d und die damit verbundene Standardabweichung berechnet sich dann folgendermaßen (siehe auch Anhang A):

$$\alpha = \frac{1}{2}\tan^{-1}\frac{-2S_{xy}}{S_{y^2} - S_{x^2}} \tag{2.11}$$

$$d = \bar{x}\cos\alpha + \bar{y}\sin\alpha \tag{2.12}$$

$$\sigma^2 = \frac{1}{2n}(S_{x^2} + S_{y^2} - \sqrt{4S_{xy}^2 + (S_{y^2} - S_{x^2})^2}) \tag{2.13}$$

mit

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i \tag{2.14}$$

$$S_{x^2} = \sum_{i=0}^{n-1}(x_i - \bar{x})^2 \qquad S_{y^2} = \sum_{i=0}^{n-1}(y_i - \bar{y})^2 \tag{2.15}$$

$$S_{xy} = \sum_{i=0}^{n-1}(x_i - \bar{x})(y_i - \bar{y}) \tag{2.16}$$

Aufteilung der Folge von Scanpunkten

Aus der Menge der Scanpunkte muss zunächst mittels einer Heuristik eine Menge von Teilfolgen konstruiert werden. Die Idee liegt nahe, sich die Abfolge der Punkte zunutze zu machen. Intuitiv muss man zwischen aufeinanderfolgenden Punkten nach einem Distanzwert fahnden, welcher eine Begrenzung zweier Linien ausmacht und einem Durchlass der Realität entsprechen könnte. Diese Begrenzung MAX-DISTANCE ist stark umgebungsabhängig. Ist der Abstand zweier Scanpunkte zueinander größer als diese Konstante, wird an dieser Stelle die Punktfolge aufgetrennt und versucht, ein Segment durch die vorherige Punktmenge zu legen.

Verifikation der berechneten Ausgleichsgeraden

Aus einer Menge von Punkten soll entschieden werden, ob eine Ausgleichsgerade daraus hervorgehen darf. Dazu werden zwei Kriterien verwendet:

- Minimale Anzahl von Punkten
  Jede Ausgleichsgerade sollte eine ausreichende Anzahl von Scanpunkten repräsentieren. Die untere Schranke wird durch die Konstante MIN-POINTS-ON-LINE festgelegt. Je kleiner, desto mehr Linien werden erzeugt.

Algorithmus 2.1: Segmentextraktion aus Laser-Sensordaten

Aufruf: **segment-extraction()**
Input: Scan s
Output: Menge l von Liniensegmenten

```
l := empty
start := 0
for {i = 1; i < n; i++} do
  p1 = scanpoint(s, i − 1)
  p2 = scanpoint(s, i)
  if {distance(p1, p2) > MAX-DISTANCE} then
    l := l ∪ split(s, start, i − 1) // Aufruf Split-Algorithmus
    start := i
  end if
end for
l := l ∪ split(s, start, n − 1)
```

- Minimale Varianz der Geraden
  Jede zu extrahierende Gerade sollte ein festgelegtes Gütekriterium erreichen. Dazu wird eine minimale Varianz MAX-SIGMA festgelegt. Ein kleiner Wert erzeugt im Mittel mehr Linien als ein hoher Wert.

In der Implementierung haben sich brauchbare Werte für MAX-SIGMA $\in [30, 50]$ Millimeter und für MIN-POINTS-ON-LINE $\in [5, 20]$ Milimeter ergeben. Die verwendete Methode make-line berechnet die eigentliche Ausgleichsgerade.

Korollar 2.22 (Aufwand des Linienfilters) *Die Linienextraktion nach Algorithmus 2.1 ist ein typischer divide-and-conquer Algorithmus. Die Zeitkomplexität beträgt* $\mathcal{O}(n^2)$. *Dabei bezeichnet* n *die Anzahl der Scanpunkte. Die Speicherkomplexität beträgt* $\mathcal{O}(n)$.

Beweis: Die Komplexitätsabschätzung kann analog zum Quicksort-Algorithmus durchgeführt werden [91]. □

Der verbesserte Linienfilter

Der obige Algorithmus besitzt zwei entscheidende Nachteile: Er setzt voraus, das ein Benutzer den Algorithmus entsprechend an die Umgebung anpasst und demzufolge die Konstante MAX-DISTANCE adaptiert. Zum weiteren werden im obigen Algorithmus alle Messpunkte als gleich behandelt, was aber nicht der Realität entspricht. Näher liegende Messpunkte besitzen eine wesentlich höhere Genauigkeit als weiter entfernt liegende Punkte. Demzufolge ist das Qualitätsmaß, wann eine neue Ausgleichsgerade akzeptiert wird, nämlich ein minimaler Varianzwert, unzureichend.

Algorithmus 2.2: Split-Algorithmus

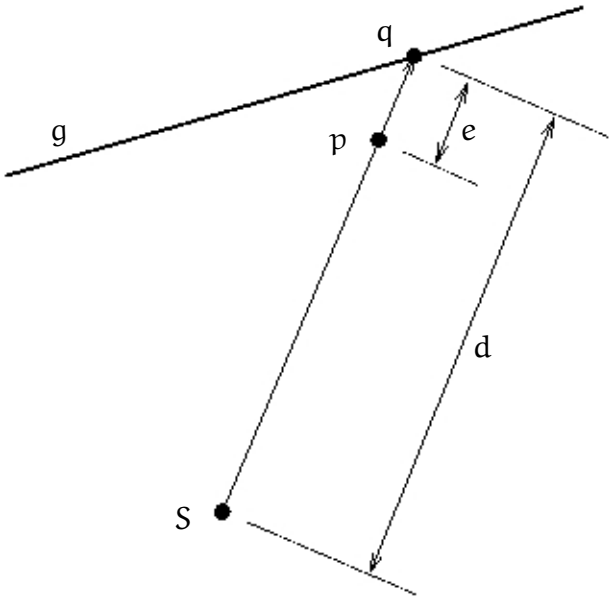
Aufruf: **split()**
Input: Scan s, Startpunkt start, Endpunkt end
Output: Menge l von Liniensegmenten

```
1  l := empty
2  line := make-line(s, start, end)
3  if {number-of-points-on-line(line) > MIN-POINTS-ON-LINE} then
4    if { σ(line) < MAX-SIGMA} then
5      l := l ∪ {line}
6    else
7      p_start := scanpoint(s, start)
8      p_end := scanpoint(s, end)
9      i_split := start
10     d := 0
11     for {i = start + 1; i <= end − 1; i++} do
12       p := scanpoint(s, i)
13       if (distance-to-line(p, p_start, p_end) > d) then
14         i_split := i
15         d := distance-to-line(p, p_start, p_end)
16       end if
17     end for
18     l := l ∪ split(s, start, i_split)
19     l := l ∪ split(s, i_split, end)
20   end if
21 end if
```

Der ursprüngliche Algorithmus wird deshalb adaptiert, sodass der Parameter MAX-DISTANCE nun durch den Algorithmus 2.3 bestimmt wird. Dazu werden die Abstände benachbarter Messpunkte aus der Folge der Scanpunkte in einem Histogramm aufgetragen und die erste Nullstelle des Histogramms als MAX-DISTANCE gewählt. Mit dieser Heuristik kann man flexibel auf jede Einsatzumgebung reagieren unter der Annahme, dass kleine Messpunktabstände sehr häufig vorkommen und Ausreißerwerte eher auf Lücken in der Umgebung zurückzuführen sind. Die Abhängigkeit des Algorithmus verlagert sich dann von der Einstellung des MAX-DISTANCE-Parameters auf die Festlegung der Histogrammgröße und damit auch der Histogrammfeinheit. Dieser Parameter wurde in allen Anwendungen auf den Wert 72 gesetzt.

Als neues Qualitätsmaß einer Ausgleichsgerade wird für jeden zu dieser Gerade zugeordneten Messpunkt überprüft, ob dieser an dieser Geraden entstanden sein könnte (siehe Abbildung 3). Dazu wird der Schnittpunkt zwischen der Ausgleichsgerade und einer Gerade vom Ursprung über den Messpunkt ermittelt. Ist dieser Schnittpunkt weiter als die Fehlerwerte des Laser-Scanners es erlauben entfernt, wird das Segment abgelehnt. Der Schnittpunkt muss also innerhalb der Fehlerhülle des extrahierten Seg-

Abbildung 3: Akzeptierte Segmente durch den Linienfilter



Die Ausgleichsgerade g wird mit der Geraden durch S und p geschnitten, der Schnittpunkt ist q. Ein Vergleich der Distanz zwischen p und q lehnt die Gerade ab oder bestätigt sie.

mentes liegen (siehe dazu auch Abschnitt 2.2.1). Wird das Segment von allen erzeugenden Punkten nicht abgelehnt, gilt das Segment als angenommen. Der Algorithmus 2.4 leistet das gewünschte.

Der Aufwand des verbesserten Linienfilters resultiert aus dem Aufwand des Linienfilters und dem zusätzlichen Aufwand zur Determinierung der Qualität und der Partitionsdiskriminante. Für die Qualität kommt pro Untersuchung ein konstanter Aufwand zur Bestimmung der Geraden, des Schnittpunktes und der Abfrage, ob die Gerade akzeptiert wird, hinzu. Die Bestimmung der Partitionsdiskriminante kann in einem Preprocessing-Schritt erfolgen und benötigt einen zusätzlichen Aufwand von $\mathcal{O}(n)$. Es ergibt sich also insgesamt:

Korollar 2.23 (Aufwand des verbesserten Linienfilters) *Der Aufwand des verbesserten Linienfilters unter Verwendung der Algorithmen 2.3 und 2.4 hat eine Komplexität von $\mathcal{O}(n^2)$ Zeitschritten. Dabei bezeichnet n die Anzahl der Messpunkte des Scans.*

Hough-Transformation

Eine weitere Möglichkeit zur Extraktion von Geraden (oder auch allgemeineren geometrischen Formen [5, 6]) aus geometrischen Daten stellt die Hough-Transformation [75, 107, 136] zur Verfügung. Die Hough-Transformation ist eine gleichberechtigte

Algorithmus 2.3: Partitionsdiskriminante

Aufruf: **compute-discriminant()**
Input: n Scanpunkte, histogram-bin-width, histogram-bin-num
Output: partition-discriminant

```
1  Bestimme min-dist der Menge der Scanpunkte
2  Bestimme max-dist der Menge der Scanpunkte
3  Berechne und speichere alle Distanzen aufeinanderfolgender Scanpunkte
4  if {max-dist > min-dist} then
5     hist-bin-width = (1.001·max-dist - min-dist) /histogram-bin-num;
6     Erzeuge Histogramm
7     while (iteriere über alle gefundenen Distanzen) do
8        Bestimme Index = int((d - min-dist) / histogram-bin-width)
9        Erhöhe Zähler im Histogramm
10    end while
11    for {i=0; i<histogram-bin-num; i++} do
12       if {Nullstelle entdeckt} then
13          partition-discriminant = min-dist + i · histogram-bin-width
14          i = histogram-bin-num;
15       end if
16    end for
17 end if
18 return partition-discriminant
```

Alternative zum Linienfilter-Algorithmus.

Jede Gerade g kann durch ein Paar $(\rho, \theta) \in [-R, R] \times [0, 2\pi[$ folgendermaßen beschrieben werden, denn es gilt:

$$x \cos\theta + y \sin\theta = \rho \tag{2.17}$$

ρ ist dabei der Abstand des Ursprungs von g und θ der Winkel zwischen der positiven x-Achse und g. Der Abstand ρ wird dabei positiv (negativ) gemessen, wenn der Ursprung links (rechts) der Geraden g liegt.

Ist ein Punkt $p \in \mathbb{R}^2$ vorgegeben, so können die Parameter aller Geraden, auf denen p liegt, leicht bestimmt werden. Für alle Punkte p des Scans werden die Parameter dieser Geraden, die p enthalten, in ein Histogramm über den Geradenparameterraum eingetragen. Darin kann anschließend nach Parameterwerten gesucht werden, deren entsprechende Geraden viele Punkte des Scans enthalten. Die Parameter aller Geraden, die einen gegebenen Punkt $p = (x_p, y_p)$ enthalten, liegen auf einer verschobenen Sinuskurve, deren Amplitude und Phasenverschiebung durch die Polarkoordinaten von p beschrieben werden. Das größte Problem der Hough-Transformation liegt darin, die Feinheit der Diskretisierung zu bestimmen. Je gröber man die Diskretisierung des Parameterraums wählt, desto ungenauer die Geraden, aber desto schneller eine

Algorithmus 2.4: Fehlerprüfung

Aufruf: **proof-segment()**
Input: Die Scanpunkte $p_1, \ldots p_m$ auf dem Segment e
Output: Boolscher Wert, ob Segment angenommen wird

```
1  RET := FALSE
2  for {p ∈ {p_1, ..., p_m} } do
3     Definiere Gerade l vom Scan-Ursprung l_s durch den betrachten Punkt p
4     Berechne Schnittpunkt q als Schnittpunkt von l und dem Segment e
5     if {es gibt einen Schnittpunkt} then
6        error := distance(p, q)
7        dist := distance(q, l_s)
8        if {error ≤ δ_f} then
9           RET := TRUE
10       end if
11    else
12       return FALSE
13    end if
14 end for
15 return RET
```

Anfrage. Je feiner man die Diskretisierung wählt, desto größer die Fehleranfälligkeit durch Ausreißer und desto langsamer die Anfragezeit.

Zudem gibt es das Problem, das Korridore oder Nischen nicht erkannt werden, da man nicht mit Segmenten, sondern mit Linien arbeitet. Ein möglicher Lösungsansatz dafür ist die simultane Clusterung im Scanbild ähnlich des Linienfilters und eine Clusterung einzelner Partitionen im Parameterraum zur Gewinnung von Segmenten.

Korollar 2.24 (Aufwand der Hough-Transformation) *Die Hough-Transformation nach Algorithmus 2.5 hat eine Zeitkomplexität von $\mathcal{O}(nm)$. Dabei bezeichnet n die Anzahl der Scanpunkte des Scans und m die Diskretisierungsfeinheit des Winkelintervalls. Die Speicherkomplexität ist aus $\mathcal{O}(mr)$. Dabei bezeichnet r die Diskretisierungsfeinheit des Abstandsintervalls.*

Beweis: Siehe [61]. □

2.2.3 Feature-Extraktion aus Kartenmodellen

Bisher ist man stillschweigend von einem Umgebungsmodell m_g ausgegangen, wobei dessen genaue Repräsentation im Ungewissen blieb. Hier wird vereinfachend davon ausgegangen, dass die gegebene Karte in Form einer geometrischen Kartenrepräsentation vorliegt (siehe dazu auch Kapitel 3.1.1). Dann kann man das folgende Problem definieren:

Algorithmus 2.5: Hough-Transformation

Aufruf: **hough-transformation()**
Input: Eine Menge M von Punkten
Output: Eine Menge von Geraden

1 Diskretisiere den Parameterraum $[-R, +R] \times [0, 2\pi[$ der Geraden. Die Abstandskomponente braucht nur von 0 bis zum maximalen Abstand eines Punktes aus M vom Ursprung betrachtet zu werden.
2 Sei H ein zweidimensionales Feld natürlicher Zahlen, das der Diskretisierung des Parameterraumes entspricht. Zunächst sei der Wert jedes Elementes gleich 0.
3 **for** (alle Punkte $p \in M$) **do**
4 **for** {alle diskretisierten Winkel $\theta \in [0, 2\pi[$} **do**
5 Sei g die Gerade, die p enthält und die Richtung θ besitzt.
6 Bestimme den Abstand ρ zwischen g und dem Ursprung.
7 Inkrementiere den Wert des Elementes von H, das dem Paar (ρ, θ) entspricht, um 1.
8 **end for**
9 **end for**
10 Suche nach Elementen mit großen Werten beziehungsweise deren Häufungen in H und gib die entsprechenden Geraden aus.

Problem 2.25 (Segmentfeature-Extraktion aus geometrischen Karten) *Gegeben sei eine zweidimensionale geometrische Kartenrepräsentation einer Einsatzumgebung. Diese soll in eine zweidimensionale Segmentkarte transformiert werden.*

Annahme: In diesem Kapitel wird davon ausgegangen, dass die Kartenrepräsentation m_g in Form einer Segmentkarte vorliegt. Die Erzeugung von Segmentkarten aus anderen in dieser Arbeit behandelten Kartenrepräsentationen wird in Kapitel 3.4 behandelt.

Die Extraktion von Segmentfeatures aus Modelldaten darf nicht unabhängig von der Extraktion von Segmentfeatures aus Bilddaten betrachtet werden. Durch den Extraktionsfehler, den man macht, wenn man Segmente aus den Scan berechnet, kann es vorkommen, dass Kartendetails einer sehr detaillierten Modellkarte der Umgebung nicht erkannt werden können bzw. aus der groben Datenmenge andere, völlig falsche Segmente extrahiert werden. Es kann jedoch auch vorkommen, dass die Modelldaten optimal auf die Bilddaten abgestimmt sind oder dass man annehmen kann, dass das der Fall ist. Für die Transformation der ursprünglichen Modelldaten in eine Segmentkartenrepräsentation ergeben sich die folgenden Varianten:

Vergröbernde Extraktion der Modellfeatures
Kleine Features werden zu neuen, größeren Features zusammengefasst.

Erweiterte Matchingmöglichkeiten
Die Zuordnung eines Features aus den Bilddaten zu **mehreren** anderen Modellfeatures kann erlaubt werden. Dazu muss dann auch zugelassen werden, dass

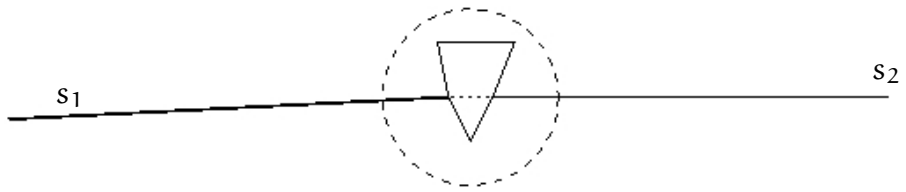
Features aus dem Bild **kürzeren** Features aus dem Modell zugeordnet werden dürfen.

Optimistische Strategie

Die Fehler, die bei der Transformation gemacht werden können, sind unbedeutend, da genug Features extrahiert werden können. Deshalb werden die Modellfeatures ohne einen Extraktionsfehler aus dem Modell übernommen.

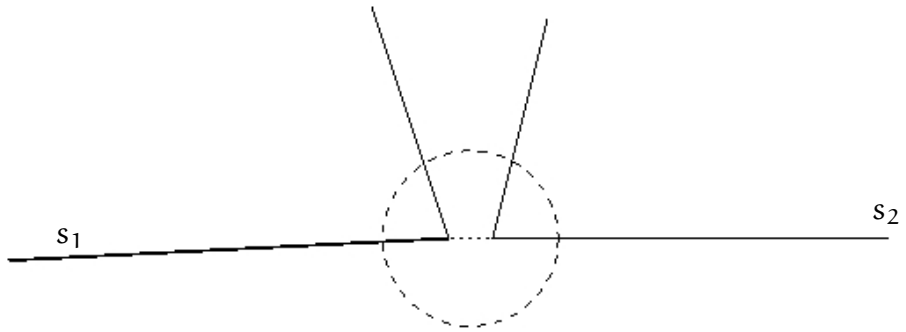
Die erweiterten Matchingmöglichkeiten führen in der Praxis nur zu einer längeren Laufzeit der feature-basierten Verfahren ohne nennenswerte Resultate zu liefern. Die optimistische Strategie, keinen Extraktionsfehler für Modellfeatures anzunehmen, kann nur für Kartenmodelle verwendet werden, welche mittels Verfahren aus Kapitel 3 erstellt wurden. In der Regel verwendet man eine vergröbernde Extraktion der Modellfeatures, weshalb diese im folgenden auch näher beleuchtet werden soll.

Abbildung 4: Kartendetails: Nischen, Vorsprünge



Beispiel für eine Nische zwischen Segment s_1 und s_2, welche durch einen Laser-Scanner nicht erkannt werden kann.

Abbildung 5: Kartendetails: Lücken



Beispiel für eine Lücke zwischen Segment s_1 und s_2, welche durch einen Laser-Scanner nicht erkannt werden kann.

Die Idee besteht darin, abhängig vom Extraktionsfehler der Scanfeatures, diesen Fehler auch auf die Kartenfeatures anzuwenden. Dazu wird eine Schranke δ festgelegt, welche den maximalen zulässigen Matchingfehler für ein Segmentfeature bezeichnet.

Algorithmus 2.6: Vergröbernde Feature-Extraktion

Aufruf: **merge-segments()**
Input: Karte m_g, Größenschranke δ
Output: Menge der Supersegmente S als Kartenfeatures

$S \leftarrow \emptyset$ {S Menge der Supersegmente der Karte}
$S' \leftarrow \{s \mid s \text{ ist Kartensegment und } \mathit{length}(s) \geq \delta\}$
{S' ist die Menge der initialen Supersegmente}
for {alle $\overline{st} \in S'$} **do**
  $f \leftarrow \overline{st}$ {Erzeuge ein neues Supersegment-Feature f.}
  $S'' \leftarrow \{\overline{uv} \in S' \backslash \{\overline{st}\} \mid \overline{uv} \text{ ist parallel zu } f\}$ {Füge Segmente aus S'' an f an.}
  repeat
    for {alle $\overline{uv} \in S''$} **do**
      if {minimaler Abstand zwischen $\overline{st}$ und $\overline{uv}$ ist kleiner als δ} **then**
        Projiziere $\overline{uv}$ auf die Gerade (st) und erhalte $\overline{u'v'}$.
        $f \leftarrow \min\{\overline{s't'} \mid \overline{s't'} \subseteq (st) \text{ und } \overline{st} \cup \overline{u'v'} \subseteq \overline{s't'}\}$
        {das heißt bzgl. Inklusion minimales, beide Segmente umfassendes Segment}
        $S'' \leftarrow S'' \backslash \{\overline{uv}\}$ {entferne das angefügte Segment}
      end if
    end for
  until {Aus S'' wurde kein Segment entnommen}
  $S \leftarrow S \cup \{f\}$ {Füge f als weiteres Supersegment zu S hinzu.}
end for
Löse Min-Set-Cover-Problem mit Algorithmus 2.7.

Alle Details, die sich innerhalb dieser Fehlerschranke befinden, werden vernachlässigt, insbesondere auch Segmente kleinerer Länge (siehe Abbildungen 4 und 5).

Für jedes Segment $\overline{st}$ der Karte wird bis zu einer Entfernung von δ in der Umgebung der Endpunkte s und t nach weiteren Segmenten $\overline{u_i v_i}$ gesucht. Diese Segmente $\overline{u_i v_i}$ sollen möglichst dieselbe Orientierung wie das Segment $\overline{st}$ besitzen. Das resultierende Segment oder Supersegment f zu $\overline{st}$ wird dann so verlängert, dass es die Projektion von $\overline{u_i v_i}$ enthält.

Set-Cover-Problematik

Um in einem weiteren Schritt noch alle Supersegmente aus der Featuremenge zu entfernen, die komplett innerhalb einer δ-Hülle eines anderen Supersegments liegen, ist es notwendig ein Überdeckungsproblem zu lösen. Gesucht ist dabei eine minimale Anzahl von Supersegmenten, welche alle anderen Segmente durch ihre δ-Hülle überdecken. Dieses in der Literatur [13] bekannte Set-Cover-Problem ist $\mathcal{NP}$-vollständig. Die bestmögliche Approximation dieser Problematik in Polynomialzeit liefert eine einfache Greedy-Heuristik.

Definition 2.26 (Min-Set-Cover) *Sei* $K = \{k_1, \ldots, k_r\}$ *die Menge der Modellfeatures der Karte. Sei* $F = \{f_1, \ldots, f_s\}$ *die Menge der durch Vergröberung aus der Karte erhaltenen Segmentfeatures. Weiter sei* $K_{f_i} \subseteq K$ *die Menge der Kanten aus* K*, die zum Segmentfeature* f_i *beitragen. Die Min-Set-Cover-Problematik besteht nun darin, ein* $F_{min} \subseteq F$ *zu bestimmen, so dass* $\cup_{f \in F_{min}} K_f = K$ *und* $| F_{min} |$ *minimal ist.*

Algorithmus 2.7: Greedy-Heuristik für MinSetCover

Aufruf: **greedy-min-set-cover()**
Input: Menge von Super-Segmenten sowie ihre Überdeckungseigenschaften
Input: Menge von Original-Segmenten, die überdeckt werden sollen
Output: Minimale Überdeckung von Super-Segmenten

```
1 while {es gibt noch zu überdeckende Segmente} do
2   for {iteriere über alle Super-Segmente} do
3     if {Super-Segment gehört noch nicht zur Überdeckung} then
4       Teste wieviele weitere Original-Segmente dieses Super-Segment überdeckt.
5       Merke das Super-Segment, wenn es maximal viele überdeckt.
6     end if
7   end for
8   Füge das maximale Super-Segment der Überdeckung hinzu.
9 end while
```

Für den Gesamtalgorithmus (Algorithmus 2.6 und Algorithmus 2.7) ergeben sich die folgenden Komplexitäten:

Korollar 2.27 (Zeit- und Speicherkomplexität) *Der Speicheraufwand liegt in* $\mathcal{O}(n)$*, der Zeitaufwand in* $\mathcal{O}(n^3)$*, wobei* n *die Anzahl der Segmentfeatures der Karte bezeichnet.*

Beweis: Betrachte zunächst Algorithmus 2.6:
Sei l die Anzahl der Kartensegmente. Dann ist die Anzahl der zu speichernden Supersegmente nicht größer, da ja Segmente zu Supersegmenten zusammengefasst und übriggebliebene Segmente gelöscht werden. Der Speicherplatzbedarf liegt also in $\mathcal{O}(l)$.

Für die Zeitkomplexität muss man durch zwei geschachtelte for-Schleifen schon über die gesamte Kartensegmentmenge iterieren. Zusätzlich wird in jedem Schritt der beiden Schleifen eine repeat-Schleife aufgerufen und mindestens ein Kartensegment entfernt. Damit ergibt sich eine Zeitkomplexität von $\mathcal{O}(l^3)$.

Betrachte nun den Algorithmus 2.7:
Zusätzlichen Speicherplatz benötigt man für die Greedy-Heuristik nur bei der Berechnung der Anzahl zugeordneter Segmente für ein Super-Segment.

Für die Zeitkomplexität muss man über alle Original-Segmente iterieren, über alle Super-Segmente und dann noch einmal über alle Original-Segmente, um festzustellen

wie viele solche ein betrachtetes Super-Segment überdeckt. Damit ergibt sich eine Zeitkomplexität von $\mathcal{O}(l^3)$.

Insgesamt ergeben sich die Komplexitäten des Korollars. □

Diese angesprochene Problematik wird nochmal in Kapitel 3 angesprochen unter dem Kontext, neue Features in eine feature-basierte Karte einzufügen und dabei ähnliche Features miteinander zu verschmelzen.

2.3 Matching lokaler Features

Im vorherigen Abschnitt wurde die Beschaffung der betrachteten Features aus den Modell- und Bilddaten unter der Annahme verrauschter Daten behandelt. Diese Features sollen nun in geeigneter Weise zugeordnet werden. Ziel ist dabei ein maximales, geometrisch konsistentes Matching zu erreichen und damit die Problemstellung 2.12 zu lösen.

In der reichhaltigen Literatur der modellbasierten Objekterkennung [29,164,167] findet man eine Klassifikation der Problematik in zwei Problemlösungsstrategien:

- **Zuordnungsbasierte Verfahren**
  Der Korrespondenzenraum, also die Menge aller möglichen Matchings, wird durchsucht mit dem Ziel, ein maximales, geometrisch konsistentes Feature-Matching zu finden aus dem sich dann eine Abbildung und damit der hypothetische Standort des Roboters berechnen lässt.

- **Transformationsbasierte Verfahren**
  Der Transformationenraum, als die Gruppe $\mathcal{TPR}$, wird durchsucht nach Teilmengen, deren Elemente die Bildfeatures unter bestimmten Aspekten gut auf die Modellfeatures abbilden.

Zu den zuordnungsbasierten Verfahren gehört die von Bolles und Cain vorgestellte Local-Focus-Feature-Methode [24], bei der besonders ausgeprägte Features ausgewählt werden und in deren Umgebung nach Zuordnungen von weiteren Features gesucht wird, um diese Zuordnung zu bestärken oder zu schwächen. Ein weiterer Vertreter dieser Methodik ist die Interpretationsbaumsuche [84]. Dabei ordnet jeder Knoten des Interpretationsbaumes ein Bildfeature einem Modellfeature zu. Bei jeder Zuordnung werden Constraints für eine erfolgreiche Zuordnung abgetestet. Ein Matching entspricht dann einem Pfad von der Wurzel zu einem Blatt. Weitere Vertreter sind das Alignment-Verfahren [105] und das geometrische Hashing [236], welche im folgenden auf die Lokalisations-Problematik angewandt werden.

Zu den transformationsbasierten Verfahren gehört das Sampling, bei welchem der Transformationenraum in diskrete Bereiche zerlegt und für diese getestet wird ob deren Abbildungen den gestellten Anforderungen genügen [43,44,115], und das Critical

Point Sampling [43, 44]. Ein weiterer Vertreter ist das Pose-Clustering [159–161], welcher ebenfalls im folgenden auf die Lokalisations-Problematik angewandt werden soll.

2.3.1 Alignment

Das Alignment ist ein zuordnungsbasiertes Verfahren, bei welchem minimale Feature-Matchings erzeugt werden. Diese liefern dann eine Transformation zwischen Bildfeatures und Modellfeatures. Da als Features hier Segmentfeatures verwendet werden, besteht ein minimales Feature-Matching aus zwei Zuordnungen (f_n, g_n) und $(f_m, g_m) \in F \times G$ nicht-paralleler Segmente. Der Winkel der Segmente muss dabei mindestens einen Schwellenwert vom Nullwinkel und vom 180 Grad Winkel abweichen.

Algorithmus 2.8: Alignment-Matching

Aufruf: **alignment()**
Input: Menge der Bildfeatures F und der Modellfeatures G
Output: Menge von Matchings $\mathcal{M}$

```
1  Preprocessing der m Kartenfeatures.
2  M = ∅.
3  for {alle n Bildfeatures aus F} do
4     for {alle m Modellfeatures aus G } do
5        Bilde minimale Feature-Zuordnung M
6        Sei eine Menge von Constraints C vorgegeben
7        for {C ∈ C} do
8           Sei die Stelligkeit k = 2.
9           Falls M mindestens k Zuordnungen enthält, prüfe, ob die letzen k Zuord-
            nungen in der Liste M die Constraits C erfüllen.
10       end for
11       if {Constrainttests erfüllt} then
12          Füge die Zuordnung M zu den gültigen Zuordnungen M hinzu.
13       end if
14    end for
15 end for
16 return M
```

Die Effizienz der Alignment-Methode hängt von folgenden Faktoren ab:

- **Effiziente Constraint-Testverfahren zur Zuordnung von Segmenten**
  Wenige Tests, welche möglichst billig falsche Zuordnungen unter Beachtung der Extraktionsfehler ausschließen, werden gefordert.
- **Effiziente Speicherung der Features des gegebenen Modells**
  Schneller Zugriff auf die Datenstruktur unter Berücksichtigung von Unsicherheiten durch die Extraktionsfehler ist wünschenswert.

Contraints auf den Zuordnungen

Die Suche nach einem geometrisch konsistenten Feature-Matching für Segment-Features kann auf Matchings beschränkt werden, die bestimmte Bedingungen oder Constraints erfüllen.

Definition 2.28 (Constraints auf Feature-Zuordnungen) *Sei F die Menge der Bildfeatures, sei G die Menge der Modellfeatures. Ein k-näres Constraint ist eine Relation $R \subseteq (F \times G)^k$.*

Ein Constraint schränkt die Menge der möglichen Zuordnungen ein. Im folgenden werden Constraints für Zuordnungen zwischen Segment-Features definiert.

Betrachtet man zwei Features $f \in F$ und $g \in G$, so kann man eine triviale Eigenschaft einer Zuordnung ausmachen. Ein Bildfeature soll nicht länger sein als ein Modellfeature. Ein Bildfeature darf aber kürzer sein, da man unter Umständen ein Modellfeature in seiner vollen Länge mit einer Sensormessung nicht erfassen kann.

Definition 2.29 (Längen-Constraint) *Es seien Bildfeatures F und Modellfeatures G gegeben. Das unäre Constraint*

$$C_{length} = \{(f_i, g_i) \mid \mathtt{length}(f_i) \leq \mathtt{length}(g_i)\}$$

erlaubt nur Zuordnungen von Bildfeatures f_i auf Modellfeatures g_i, falls f_i nicht länger als g_i ist.

Das Längen-Constraint kann durch erweiterte Matchingmöglichkeiten auch nicht aktiviert sein. Hier soll im folgenden aber immer von der Gültigkeit des Constraints ausgegangen werden.

Betrachtet man zwei Modellfeatures g_1 und g_2, so besitzen diese einen eingeschlossenen Winkel zueinander. Zwei auf diese beiden Modellfeatures abzubildende Bildfeatures f_1 und f_2 müssen demzufolge natürlich auch einen solchen Winkelwert besitzen. Dieser darf aber im Rahmen der Messungenauigkeit vom ursprünglichen Winkel differieren.

Definition 2.30 (Winkel-Constraint) *Sind (f_1, g_1) und (f_2, g_2) zwei Zuordnungen zwischen Segmentfeatures, so muss der Winkel zwischen f_1 und f_2 im Rahmen der Messgenauigkeit mit dem Winkel zwischen g_1 und g_2 übereinstimmen.*

$$C_{angle} = \{((f_1, g_1), (f_2, g_2)) \mid \angle(g_1, g_2) \in [\angle_{\min}(f_1, f_2), \angle_{\max}(f_1, f_2)]\}$$

Algorithmisch betrachtet bearbeitet man ein Winkel-Constraint folgendermaßen: Man sucht für ein Segmentfeature f_1 ein passendes Bildfeature g_1. Ist dann eine Zuordnung (f_1, g_1) gegeben, wählt man aus den verbliebenen Bildfeatures ein weiteres f_2 aus. Aus der Menge der Modellfeatures wird dann ein Feature g_2 gesucht, welches das

Winkel-Constraint erfüllt. Eine solche Anfrage kann effizient bestimmt werden, falls die Kartensegmente nach Ausrichtung sortiert vorliegen.

Neben dem Winkel kann man auch die Abstände zweier Bildfeatures betrachten. Betrachtet man zwei Modellfeatures g_1 und g_2, besitzen diese gewisse Abstände zueinander. Diese kann man durch das sogenannte Abstands-Intervall definieren:

Definition 2.31 (Abstands-Intervall) *Seien s_1 und $s_2 \subseteq \mathbb{R}^2$ Segmente in der euklidischen Ebene, sei $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ die euklidische Abstandsfunktion. Dann ist das* Abstandsintervall $[m, M]$ *von s_1 und s_2 durch*

$$m := \min_{p_1 \in s_1} \min_{p_2 \in s_2} d(p_1, p_2)$$

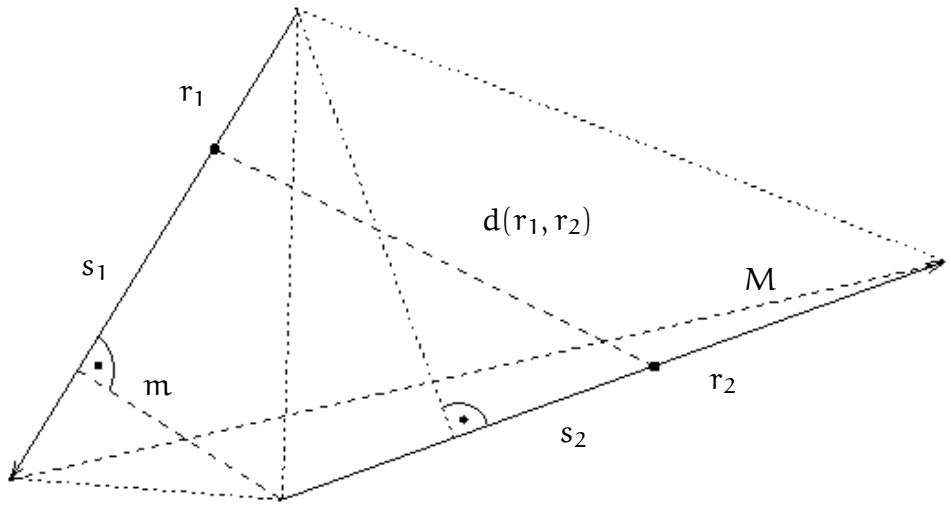
und

$$M := \max_{q_1 \in s_1} \max_{q_2 \in s_2} d(q_1, q_2)$$

definiert.

Es werden also Punktepaare (p_1, p_2), $(q_1, q_2) \in s_1 \times s_2$ betrachtet, so dass deren euklidischer Abstand $d(p_1, p_2)$ minimal beziehungsweise $d(q_1, q_2)$ maximal wird. Der maximale Abstand wird dabei zwischen Endpunkten angenommen. Zur Berechnung der minimalen Abstände sind zusätzlich die Längen der Lote von jedem Endpunkt des einen Segmentes auf das andere zu betrachten (siehe Abbildung 6). Mit dem Abstands-

Abbildung 6: Abstandsintervall von Segmenten



Für zwei Segmente s_1 und s_2 kann man den minimalen Abstand m und den maximalen Abstand M definieren. Demzufolge werden alle Abstände aus $[m, M]$ zwischen beiden Segmenten angenommen.

Intervall kann man nun das folgende Constraint definieren:

Definition 2.32 (Abstands-Constraint) *Sind* (f_1, g_1) *und* (f_2, g_2) *zwei Zuordnungen zwischen Segmentfeatures, wobei die Bildfeatures* f_1 *und* f_2 *eventuell teilweise verdeckt sind, so ist im Falle einer geometrisch konsistenten Zuordnung das Abstandsintervall* $[m_f, M_f]$ *von* f_1 *und* f_2 *im Abstandsintervall* $[m_g, M_g]$ *von* g_1 *und* g_2 *enthalten.*

$$C_{dist} = \{((f_1, g_1), (f_2, g_2)) \mid [m_f, M_f] \subset [m_g, M_g]\}$$

Das Abstands-Constraint und das Winkel-Constraint kann man auch gleichzeitig abtesten und das resultierende Constraint als das Constraint der gegenseitigen Lage bezeichnen.

Definition 2.33 (Constraint der gegenseitigen Lage) *Es seien* (f_1, g_1) *und* (f_2, g_2) *Zuordnungen zwischen Bildfeatures* $f_1, f_2 \in F$ *und Modellfeatures* $g_1, g_2 \in G$. *Das Constraint der gegenseitgen Lage ist dann definiert als*

$$C_{place} = C_{angle} \cap C_{dist}$$

Constrainttests in der Praxis

Die Zuordnung der Segmentfeatures geschieht in der Praxis folgendermaßen: Dem ersten Feature f_1 eines jeden Paares von Scanfeatures wird ein Kartenfeature g_1 zugeordnet, wobei das Constraint C_{length} über die Segmentlänge erfüllt sein muss. Durch den vorgegebenen Winkelbereich des zweiten Scanfeatures f_2 lässt sich nun aus den Kartenfeatures leicht ein passendes g_2 finden. Die Kartenfeatures werden dabei in einem Preprocessingschritt in einer effiziente Suchstruktur abgelegt [122]. Für jedes so gewonnene Paar von Zuordnungen zwischen Bild- und Modellfeatures wird das Abstands-Constraint C_{dist} geprüft und bei Akzeptanz eine vorläufige Hypothese ausgegeben.

Effizienzsteigerung durch Preprocessing

Eine effiziente Speicherung der Kartensegmente ist für die Performance des Alignment-Verfahren wichtig. Dabei soll die Datenstruktur eine effiziente, fehlertolerante Suche nach Segmenten ermöglichen, um die Constraint-Tests zu beantworten. Um das zu erreichen, wird ein Segment u durch ein Quadrupel $u = (\theta, \rho, \alpha, \omega)$ parametrisiert (siehe Abbildung 7). Die Parameter θ und ρ beschreiben die orientierte Gerade g, auf der sich das Segment $u = \overline{st}$ befindet. ρ beschreibt dabei den Abstand der Geraden g zum Ursprung und θ den Winkel mit der positiven $x-$Achse und der Normalen von g. Der Abstand ρ ist positiv, wenn der Ursprung links von g liegt, ansonsten negativ. Die beiden anderen Parameter α und ω beschreiben die Lage des Segmentes u auf der Geraden g. Anfangs- und Endpunkt werden dabei durch ihren Abstand zum Fußpunkt des Lotes des Koordinatenursprungs auf g beschrieben, wobei der Abstand in Geradenrichtung positiv, in entgegengesetzter Richtung negativ gewertet wird.

Algorithmus 2.9: Generierung einer Feature-Zuordnung

Aufruf: **compute-feature-pairing()**
Input: zwei Segmentfeatures f_1, f_2 aus dem Bild
Output: minimale Feature-Zuordnungen oder die leere Menge

```
1  if (eingeschlossenen Winkel der Segmente ≠ 0 ∧ ≠ 180) then
2    Berechne Bereichsanfrage für f1 und erhalte g1
3    if (Orientierungsvorwissen passt) then
4      if (length(f2) ≤ length(g2)) then
5        if (g2 erfüllt Winkel-Constraint C_angle) then
6          if (Kartensegmente g1 und g2 nicht parallel) then
7            if (Orientierungsvorwissen für das zweite Paar) then
8              if (Abstands-Constraint ist erfüllt) then
9                Akzeptiere die Feature-Zuordnung M = {(f1, g1), (f2, g2)}
10             end if
11           end if
12         end if
13       end if
14     end if
15   end if
16 end if
```

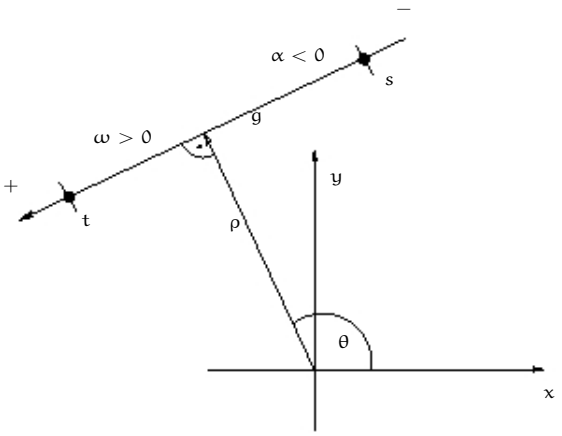
Modellierung der Ungenauigkeiten

Die Anfragesegmente werden als Punktmengen aus einem verrauschten Scan gewonnen. Jede zugeordnete Gerade besitzt dann eine individuelle Fehlerabschätzung δ. Da die Anfragesegmente nur ungefähr den Segmenten der Karte entsprechen, ja sogar nur Teilsegmente sein können, muss die Anfrage an die Datenstruktur Unsicherheiten repräsentieren. Demzufolge wird kein individuelles Anfragesegment als Anfrage an die Datenstruktur gestellt, sondern ein Hyperquader, d.h. für jeden Parameter der Anfrage ein eigenes Anfrageintervall verwendet.

Hat man also ein unsicheres Bildfeature $f = (\theta, \rho, \alpha, \omega)$ extrahiert, möchte man zumindest die Orientierungsunsicherheit und Distanzunsicherheit in Form von Intervallen quantifizieren. Die Unsicherheits-Intervalle für die Parameter α und ω seien beliebig groß, da eine Einschränkung dieser Parameter algorithmisch nicht lohnenswert ist.

Die Orientierungs-Ungenauigkeit $\Delta\theta_{max}$ bekommt man mit Hilfe der individuellen Fehlerungenauigkeit δ eines extrahierten Segments. Betrachtet man eine Feature-Zuordnung zwischen einem Bildfeature f und einem Modellfeature $g = \overline{st}$, so darf das Feature g jede Lage innerhalb der Fehlerhülle um f annehmen. Nun muss man aber das Feature f als Anfragefeature festhalten und kann g um einen Winkel $\Delta\theta_{max}$ variieren. Dieser Winkel wird berechnet (siehe Abbildung 8) durch $\sin \Delta\theta_{max} = \frac{2\delta}{|t-s|}$. Somit ergibt sich der Anfragebereich für die Orientierung zu $[\theta - \Delta\theta_{max}, \theta + \Delta\theta_{max}]$.

Abbildung 7: Parametrisierung von Segmenten



Parametrisierung eines Segments durch einen Winkel θ, eine Distanz vom Ursprung ρ und Distanzen α für den Anfangspunkt s und ω für den Endpunkt t vom Lotpunkt.

Entfernungsungenauigkeit

Die Entfernungsungenauigkeit kann man abschätzen, indem man zum Anfragesegment f und dem Messfehler δ die Geraden betrachtet, auf denen ein gültiges Modellfeature liegen kann.

Betrachtet man Abbildung 9, kann man vier extremale Geraden ausmachen, welche für einen minimalen und einen extremalen Distanzwert in Frage kommen, die zu f parallelen Geraden welche h_1 und h_2 im rechten Winkel schneiden, und die beiden diagonalen Geraden, welche h_1 und h_2 mit einer Distanz δ von s und t scheiden. Man kann also d_{min} und d_{max} ermitteln, indem man die Abstände des Ursprungs zu den Geraden ermittelt.

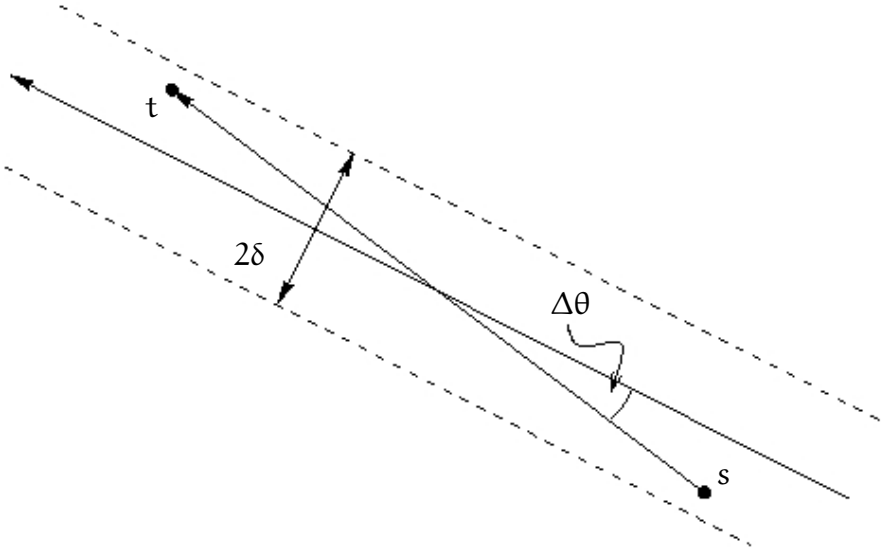
Bereichsanfrage

Sind nun die Intervalle für kompatible Segmente aus der Karte bekannt, so kann an eine geeignete Suchstruktur eine entsprechende Bereichsanfrage gestellt werden. Für ein Segment f ergibt sich eine Bereichsanfrage der Form

$$[\theta - \Delta\theta_{max}, \theta + \Delta\theta_{max}] \times [\rho - d_{min}, \rho + d_{max}] \times]-\infty, +\infty[\times]-\infty, +\infty[$$

Hier bietet sich ein 4-d-Baum [225] für n Punkte aus $\mathbb{R}^4$ an. Mit einer Zeitkomplexität aus $\mathcal{O}(n \log n)$ lässt sich dieser aufbauen und benötigt linear viel Speicherplatz. Bereichsanfragen können damit in Zeit aus $\mathcal{O}(n^{3/4} + a)$ beantwortet werden, wobei a die Größe der Antwort bezeichnet.

Abbildung 8: Segment mit Fehlerabschätzung



Bildfeature f mit Fehlerhülle δ und Modellfeature $g = \overline{st}$

Zeit- und Speicherkomplexität des Alignment-Verfahrens

Für das Alignment-Verfahren ergibt sich das folgende Komplexitätsresultat.

Korollar 2.34 (Komplexität des Alignment-Verfahrens) *Das Alignment-Verfahren nach Algorithmus 2.8 und 2.9 benötigt bei m Bildfeatures und n Modellfeatures eine Zeit- und Speicherkomplexität von $\mathcal{O}(n^2m^2)$.*

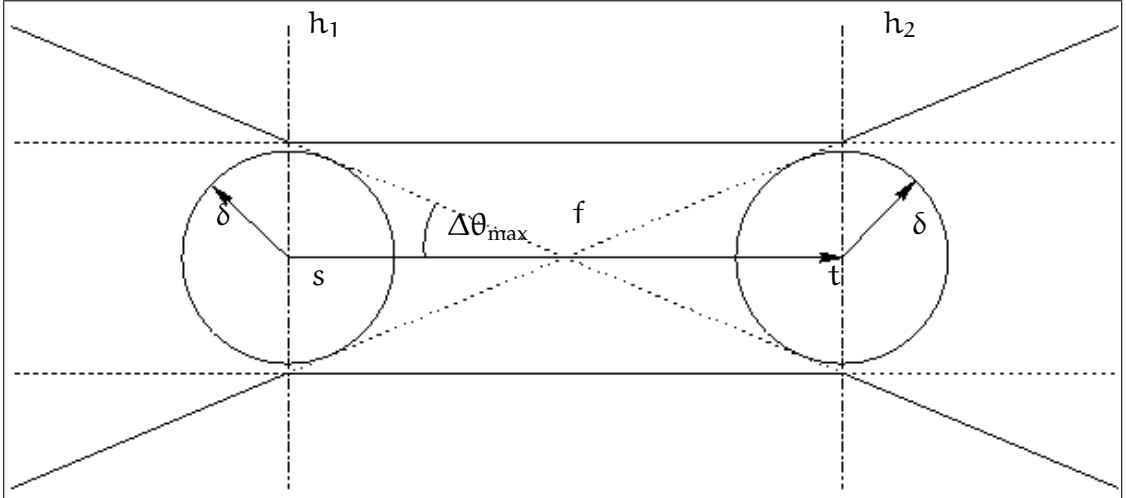
Beweis: In der Vorverarbeitungsphase muss zum Aufbau der Suchstruktur ein 4-d-Baum für die n Segmente der Karte aufgebaut werden, was sich in einer Zeit von $\mathcal{O}(n \log n)$ realisieren lässt, ebenso das Sortieren der Kartensegmente nach der Richtung. Die Segmente der Karte benötigen insgesamt einen Speicheraufwand von $\mathcal{O}(n)$.

In der Anfragephase können einem Bildfeature höchstens $\mathcal{O}(n)$ viele Modellfeatures zugeordnet werden, sodass die Anzahl minimaler Matchings aus $\mathcal{O}(m^2n^2)$ ist. Von gleicher Komplexität ist die Anzahl der zu speichernden vorläufigen Hypothesen, also $\mathcal{O}(m^2n^2)$. □

2.3.2 Pose Clustering

Das Pose-Clustering ist eine Verallgemeinerung der Hough-Transformation [159–161]. Die Hough-Transformation wurde in der Literatur hauptsächlich für die Erkennung von geometrischen Primitiven aus Pixelbildern verwendet. Hier wird das Prinzip auf den Parameterraum der Transformationen angewendet. Dieser wird in Zellen diskretisiert, wobei jede Zelle eine Menge von Transformationen enthält und mit einem zunächst leeren Matching markiert wird. Sind alle Zuordnungen abgearbeitet, werden

Abbildung 9: Kompatible Modellfeatures zum Anfragefeature f



Alle Modellfeatures, welche innerhalb des Korridors von f liegen, kommen als Zuordnungspartner für f in Frage.

Zellen mit maximalem Matchingwert gesucht und als Hypothesen ausgegeben. Bei der Anwendung des Prinzips auf die Praxis kristallisieren sich folgende Bereiche heraus, die für die Effizienz des Verfahren entscheidend sind:

- Speicherung des Histogramms des Transformationsparameterraumes
  Das Histogramm soll effizient und speichergünstig abgelegt sein.
- Markierung der Zellen unter Ungenauigkeiten
  Ein Bildfeature und dessen Ungenauigkeit muss sich entsprechend in der Markierung der Zellen niederschlagen.
- Auswertung des Matching-Histogramms
  Das Histogramm muss fehlertolerant ausgewertet werden unter Berücksichtigung von Ausreißern, die ignoriert werden sollen.

Diskretisierung und Speicherung des Histogramms

Ein zu wählender Parameter ist die Feinheit der Diskretisierung. Sie wird durch die Genauigkeit festgelegt, mit der die Features aus dem Scan extrahiert wurden.

$$\tau = \min_{f \text{ Scanfeature}} d_{max}(f)$$

legt dabei die Feinheit der Diskretisierung in den Translationsrichtungen fest. Dabei bezeichnet $d_{max}(f)$ den maximalen Sensorwert für ein Feature f. τ repräsentiert also den minimalsten der von der Feature-Menge F erreichbaren maximalen Sensorwert. Somit stellt diese Größe einen Anhaltspunkt für die zu erreichende Genauigkeit der

Algorithmus 2.10: Pose Clustering

Aufruf: **pose-clustering()**
Input: Menge F der Bildfeatures, Menge G der Modellfeatures, Schwellenwert γ
Output: Menge M möglicher Matchings

```
1  Bestimme τ und σ, die Diskretisierungsparameter.
2  Diskretisiere den Transformationenraum in Zellen.
3  // Markierungsphase
4  Initialisiere jede Zelle C.
5  for {alle möglichen Feature-Zuordnungen (f, g)} do
6     Bestimme die Menge der Zellen im Transformationenraum, die durch (f, g)
      induzierte Transformationen enthalten.
7     Für jede solche Zelle C füge die Zuordnung (f, g) zum Matching M(C)
      hinzu.
8  end for
9  // Auswertungsphase
10 for {alle Zellen C} do
11    if {Zähler von C überschreitet Schwellenwert γ } then
12       Füge Feature-Zuordnung der Menge M hinzu.
13    end if
14 end for
15 return M
```

Positionsbestimmung dar.

Äquivalent dazu legt

$$\sigma = \min_{f \text{ Scanfeature}} \Delta\theta_{max}(f)$$

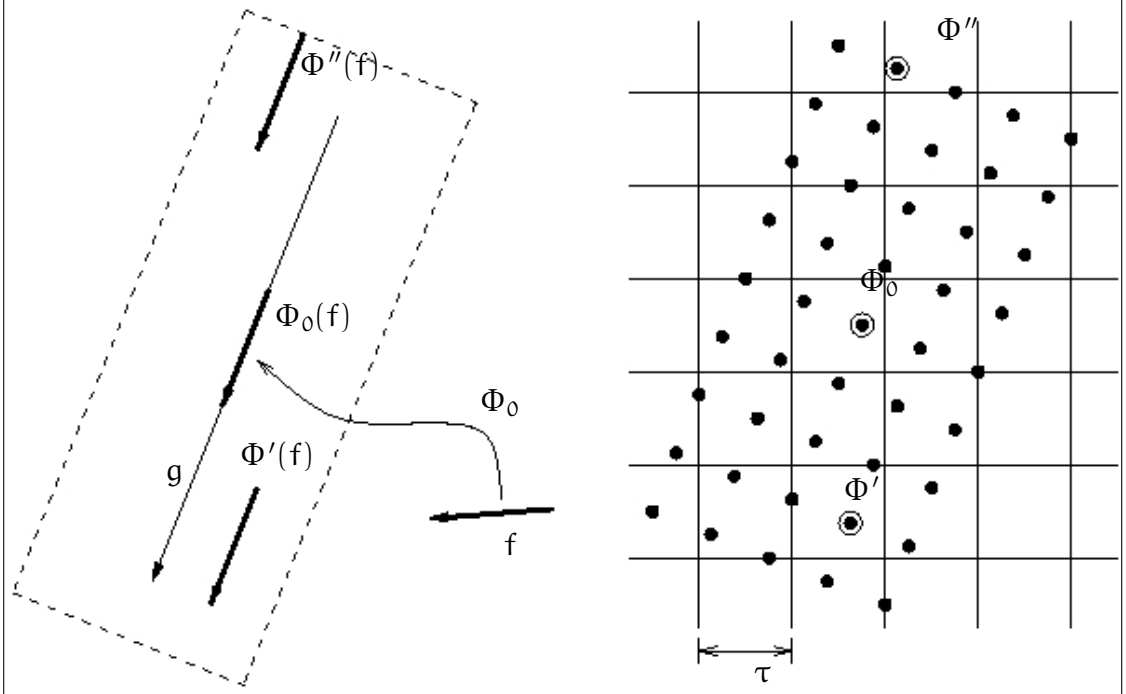
die Feinheit der Diskretisierung des Transformationsparameterraumes in θ-Richtung fest. Der Wert $\Delta\theta_{max}$ wurde in Abschnitt 2.3.1 hergeleitet.

Aus Speicherplatzgründen wird kein statisches, dreidimensionales Array zur Speicherung der Zellen verwendet, sondern ein Wörterbuch, das jeweils einem Tripel ganzer Zahlen, das die Koordinaten der Zelle in Diskretisierungseinheiten angibt, ein Feature-Matching zuordnet. Dadurch wird eine schlechtere Laufzeit gegen den Speicherplatzgewinn getauscht.

Markierung der Zellen

Es wird über die Menge aller Zuordnungen (f, g) von Bildfeatures $f \in F$ aus dem Bild- und Modellfeatures $g \in G$ iteriert. Zu jeder dieser Zuordnungen wird eine Reihe von Rotationswinkeln $\varphi_1, \ldots, \varphi_s$ mit $\varphi_{i+1} - \varphi_i = \sigma$ berechnet, so dass f nach einer Drehung um φ_i ($i \in \{1, \ldots, s\}$) in seiner Richtung höchstens um den bei der Extraktion

Abbildung 10: Berechnetes Gitter beim Pose-Clustering



Links ist das Modellfeature g mit entsprechender Fehlerhülle abgebildet. Rechts ist das Histogramm mit seiner Feinheit τ abgebildet. Entsprechend der Fehlerhülle von g werden die Abbildungen Φ_i berechnet und die Zellen im Histogramm markiert.

gemachten Winkelfehler von der Richtung von g abweicht.

Zu jeder dieser Rotationen φ_i um den Ursprung wird eine Translation u_i berechnet, die die Mittelpunkte der Segmente aufeinander abbildet. Die Verknüpfung dieser Rotation φ_i mit der Translation u_i liefert die Abbildung Φ_i (Abbildung 10).

In der Umgebung dieser Abbildung werden im Parameterraum der Transformationen orthogonale Gitterpunkte berechnet, deren Ausbreitung von der Längendifferenz zwischen den Segmenten f und g sowie von der Ungenauigkeit δ abhängt, mit der das Scanfeature f extrahiert wurde. Die Transformationen auf diesen Gitterpunkten entsprechen der Abbildung Φ_i verkettet mit einer zu g parallelen und orthogonalen Verschiebung. Das berechnete Gitter ist im allgemeinen gegen die Diskretisierung des Transformationenraumes verdreht und daher um den Faktor $\sqrt{2}$ feiner als die Diskretisierung τ zu wählen, um alle zu markierenden Zellen innerhalb des Fehlerbereichs zu treffen. Die Zähler der Zellen in denen die berechneten Gitterpunkte liegen, werden für die Zuordnung (f, g) um eins inkrementiert.

Auswertung des Histogramms

Die Auswertung der Zellen ist der wichtigste Schritt des Pose-Clusterings. Nachdem alle Zuordnungen möglicher Bildfeatures f mit Modellfeatures g berechnet und markiert wurden, muss über die Histogramm-Datenstruktur iteriert werden, um Zellen mit möglichst hohem Zähler zu finden. Dabei wird ein Schwellenwert eingeführt, ab welchem erst Matchings generiert werden, um Ausreißer und falsche Zuordnungen auszuschließen. Dieser Schwellenwert wird relativ zum größten entdeckten Matching in Prozent vom Benutzer festgelegt, und liegt üblicherweise bei 80%. Aus allen Zellen, die diesen Schwellenwert übersteigen, wird ein vorläufiges Matching berechnet, welches abschließend mit einem Verifikationsverfahren zu verifizieren ist.

Zeit- und Speicherkomplexität

Für das obige Matchingverfahren ergibt sich die folgende Abschätzung der Komplexitäten:

Korollar 2.35 (Speicher- und Zeitkomplexität) *Für das Pose-Clustering Verfahren nach Algorithmus 2.10 ergibt sich bei m Bildfeatures eine Gesamtlaufzeit der Markierungsphase aus $\mathcal{O}((m \cdot \frac{L}{\tau}) \log(m \cdot \frac{L}{\tau}))$. Dabei ist L die Summe der Längen aller Modellfeatures. Der Speicherplatzbedarf des Verfahrens wird durch die Anzahl markierter Zellen bestimmt, welcher bei insgesamt m Bildfeatures somit aus $\mathcal{O}(m \cdot \frac{L}{\tau})$ ist.*

Beweis: Der maximale Extraktionsfehler orthogonal zum entsprechenden Segment sowie der maximale Winkelfehler werden als beschränkt angenommen, das heißt es existiert eine Konstante $c \in \mathbb{N}$, so dass diese beiden Fehler durch $c\tau$ beziehungsweise durch $c\sigma$ beschränkt sind.

Sei L die Summe der Längen aller Modellfeatures, also die insgesamt modellierte Wandlänge. Dann ist für jedes Segmentfeature aus dem Bild die Anzahl der in der Markierungsphase zu berechnenden Transformationen aus $\mathcal{O}(\frac{L}{\tau})$, da die Schritte senkrecht zu einem Kartensegment g in der Rotationsdimension und der entsprechenden Translationsrichtung jeweils durch die Konstante c beschränkt sind und in der Translationsrichtung parallel zum Kartensegment g der Länge l durch $\lceil \frac{l}{\tau} \rceil$ nach oben abgeschätzt werden können. Die Anzahl markierter Zellen bei insgesamt m Scansegmenten ist somit aus $\mathcal{O}(m \cdot \frac{L}{\tau})$, da die Anzahl der berechneten Transformationen pro markierter Zelle mindestens 1 ist. Unter der in realen Szenarien erfüllten Annahme, dass für jede Position des Roboters und damit für jede Transformation nur geometrisch konsistente Matchings beschränkter Größe existieren, wird durch jede markierte Zelle konstant viel Speicherplatz belegt, so dass die genannte Abschätzung für die Anzahl markierter Zellen die Speicherkomplexität des Algorithmus wiedergibt.

Da der Wörterbuchzugriff in logarithmischer Zeit erfolgt, ergibt sich für m Scansegmente insgesamt eine Gesamtlaufzeit der Markierungsphase aus $\mathcal{O}((m \cdot \frac{L}{\tau}) \log(m \cdot \frac{L}{\tau}))$.

Die Suche nach Matchings, die relativ zum maximalen konstruierten Matching hinreichende Größe besitzen, kann durch zweimaliges Traversieren des Wörterbuches erledigt werden. Somit wird die Gesamtlaufzeit von der Markierungsphase dominiert und ist ebenfalls aus $\mathcal{O}((m \cdot \frac{L}{\tau}) \log(m \cdot \frac{L}{\tau}))$. □

2.3.3 Geometrisches Hashing

Geometrisches Hashing ist ein zuordnungsbasierter Ansatz der modellbasierten Objekterkennung [236]. Dieser basiert im Gegensatz zu den bisherigen Ansätzen auf einem Preprocessingschritt, in dem geometrisch-invariante Informationen der Modellfeatures in einer Hashtabelle indiziert werden [164, 221, 222]. In der Erkennungsphase können dann sehr schnell mit Hilfe der Vorverarbeitung Anfragen bearbeitet werden. Das allgemeine Verfahren für einen beliebigen Featuretyp wird nachfolgend beschrieben.

Preprocessing

Im Vorverarbeitungsschritt (siehe Algorithmus 2.11) bestimmt man abhängig von der Transformationsgruppe aus der Menge der Modellfeatures die Menge der Basen. Eine Basis B ist hier eine minimale Menge von Features $\{g_1, \ldots, g_k\}$ mit der Eigenschaft, dass aus den transformierten Basisfeatures $\{\phi(g_1), \ldots, \phi(g_k)\}$ zusammen mit den Urbildern auf die Transformation $\phi \in \mathcal{TPR}$ zurückgeschlossen werden kann.

Für jede Basis B werden die restlichen Modellfeatures, welche nicht zur Basis gehören im Bezugssystem dieser Basis beschrieben. Genauer werden Invarianten $I(g_i)$ der einzelnen Modellfeatures g_i bezüglich der gewählten Basis B berechnet. Diese Invarianten dienen dann als $r-$dimensionaler Schlüssel in einer Hashtabelle, in der die Basis entsprechend abgelegt wird.

Definition 2.36 (Geometrische Invariante) *Für eine Basis* B *und für ein Feature* f *beschreibt eine Menge von geometrische Invarianten* $I(f)$ *eine eindeutige Charakterisierung dieses Features bezüglich der Basis und bezüglich der betrachteten Transformationsgruppe* $\mathcal{TPR}$ *.*

Objekterkennung

In der Erkennungsphase oder Anfragephase (siehe Algorithmus 2.12) werden aus der Menge der Bildfeatures alle möglichen Basen gebildet. Dabei wird davon ausgegangen, dass die Bildfeature-Menge F sehr klein ist. Betrachte nun eine gewählte Basis B_i der Bildfeatures. Für die verbliebenen, nicht zur Basis gehörenden Bildfeatures f_i werden nun die Invarianten $I(f_i)$ berechnet und in der Hashtabelle Hash die passenden Modellbasen gesucht. Jede getroffene Modellbasis bekommt nun eine Stimme, d.h. man benötigt zusätzlich eine Datenstruktur, welche einen Zähler für jede mögliche Modellbasis enthält, sodass dieser Zähler erhöht werden kann. Nachdem alle Bildfeatures und alle Basen abgearbeitet worden sind, sucht man die besten Modellbasen heraus, d.h.

Algorithmus 2.11: Preprocessing beim Geometrischen Hashing

Aufruf: **gh-preprocessing-step()**
Input: Menge von Modellfeatures G
Output: Hashtabelle Hash mit den abgelegten Basen

```
1 Sei Hash eine Hashtabelle.
2 for {alle Basen B ⊆ G der Modellfeatures} do
3   for {alle Modellfeatures g_m ∈ G\B} do
4     Berechne die Invarianten I(g_m) von g_m relativ zu B.
5     Füge die Basis B unter dem Schlüssel I(g_m) in Hash ein.
6   end for
7 end for
```

diejenigen, welche den höchsten Zähler erreicht haben, und bestimmt für diese Basen entsprechend deren Transformationen Φ.

Modellierung der Basen und Berechnung der Invarianten

Als Features sollen hier Linien dienen. Dabei bezeichne r die Distanz der Normalen von der Linie zum Ursprung des Koordinatensystems. ϕ bezeichnet demnach dann den Winkel zwischen der Normalen und der positiven X−Achse (siehe dazu auch Abbildung 7).

Eine Basis besteht dann aus zwei Linien mit der sich die Invarianten (ϕ, r) zu einer beliebigen dritten Linie berechnen lassen. Dabei gilt die Voraussetzung, dass die Linien nicht parallel liegen und sich alle drei Linien nicht in einem gemeinsamen Punkt schneiden. Für eine Rigid-motion, die hier betrachtet wird, benötigt man drei Linien, da ansonsten die Eindeutigkeit der Transformation nicht gegeben ist.

Um zu einer Basisbeschreibung zu kommen, und damit eine Beschreibung für alle anderen Linienfeatures zu bekommen, muss man folgendermaßen vorgehen: Man sucht eine Transformation P, welche die erste Linie auf die X−Achse abbildet und die zweite Linie, welche die Basis beschreiben soll so, dass die beiden Linien sich im Ursprung des Koordinatensystems schneiden. Diese Transformation kann man dann zur Bestimmung der Invarianten einer dritten Linie anwenden.

Betrachte nun eine Rigid-motion T mit

$$T = \begin{pmatrix} R & b \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi & b1 \\ -\sin\phi & \cos\phi & b2 \\ 0 & 0 & b3 \end{pmatrix} \tag{2.18}$$

Dabei ist R eine Rotationsmatrix und b ein Translationsvektor. Dadurch ergibt sich P als

$$P = (T^{-1})^T = \begin{pmatrix} (R^{-1})^T & 0 \\ -b^T(R^{-1})^T & 1 \end{pmatrix} = \begin{pmatrix} R & 0 \\ -b^T R & 1 \end{pmatrix} \tag{2.19}$$

Algorithmus 2.12: Erkennungsschritt beim Geometrischen Hashing

Aufruf: **gh-recognition-step()**
Input: Menge von Bildfeatures F, Hashtabelle Hash
Output: Menge von Transformationen T

Sei $T = \emptyset$ eine Menge von Transformationen.
repeat
  Leere die Menge der Kandidatenbasen.
  Wähle aus der Menge F_i der Bildfeatures eine Basis $B_i \subseteq F_i$.
  for {alle Bildfeatures $f_i \in F_i \backslash B_i$} **do**
    Berechne die Invarianten $I(f_i)$ von f_i relativ zu B_i.
    Finde durch Nachschauen in der Hashtabelle diejenigen zu B passenden Basen $\{B_1, \ldots, B_k\}$ von Modellfeatures, für die Modellfeatures mit denselben Invarianten $I(f_i)$ existieren.
    for {alle Basen $B_m \in \{B_1, \ldots, B_k\}$} **do**
      Füge B_m zur Menge der Kandidatenbasen hinzu beziehungsweise erhöhe einen Zähler für B_m, falls sie schon vorhanden war.
    end for
  end for
until {es gibt noch Basen von Bildfeatures}
for {alle Kandidatenbasen B_m mit großem Zähler} **do**
  Bestimme eine Transformation ϕ, die die Features aus B_i auf die entsprechenden Features aus B_m abbildet.
  Verifiziere oder falsifiziere ϕ, und füge ϕ im verifizierten Fall an T an.
end for

Es folgt

$$P^{-1} = T^T = \begin{pmatrix} R^T & 0 \\ b^T & 1 \end{pmatrix} \tag{2.20}$$

Hat man nun allgemein zwei Linien, beschrieben durch ihre Parameter

$$a_1 = (\cos\phi_1, \sin\phi_1, -r_1)^T \tag{2.21}$$

und

$$a_2 = (\cos\phi_2, \sin\phi_2, -r_2)^T \tag{2.22}$$

sollen diese mittels P abgebildet werden auf

$$e_1 = (\cos\frac{\pi}{2}, \sin\frac{\pi}{2}, 0)^T \tag{2.23}$$

und

$$e_2 = (\cos\rho, \sin\rho, 0)^T \tag{2.24}$$

Dabei ist ρ ein fester, aber unbekannter Winkel. Es soll also gelten:

$$\lambda_1 P a_1 = e_1 \text{ und } \lambda_2 P a_2 = e_2 \tag{2.25}$$

Es folgt

$$(\lambda_1 a_1, \lambda_2 a_2) = P^{-1}(e_1, e_2) = \begin{pmatrix} -\sin\phi & \cos\phi\cos\rho - \sin\phi\sin\rho \\ \cos\phi & \sin\phi\cos\rho + \cos\phi\sin\rho \\ b_2 & b_1\cos\rho + b_2\sin\rho \end{pmatrix} \tag{2.26}$$

Damit ergibt sich

$$\lambda_1 = \frac{b_2}{-r_1} \tag{2.27}$$

und

$$\lambda_2 = \frac{b_1\cos\rho + b_2\sin\rho}{-r_2} \tag{2.28}$$

Damit kann man nun zwei Matrizen P bestimmen, welche die Transformation beschreiben.

$$P^{(1)} = \begin{pmatrix} \sin\phi_1 & -\cos\phi_1 & 0 \\ \cos\phi_1 & \sin\phi_1 & 0 \\ \csc(\phi_1-\phi_2)(r_2\sin\phi_1 - r_1\sin\phi_2) & \csc(\phi_1-\phi_2)(-r_2\cos\phi_1 + r_1\cos\phi_2) & 1 \end{pmatrix} \tag{2.29}$$

und

$$P^{(2)} = \begin{pmatrix} -\sin\phi_1 & \cos\phi_1 & 0 \\ -\cos\phi_1 & -\sin\phi_1 & 0 \\ \csc(\phi_1-\phi_2)(r_2\sin\phi_1 - r_1\sin\phi_2) & \csc(\phi_1-\phi_2)(-r_2\cos\phi_1 + r_1\cos\phi_2) & 1 \end{pmatrix} \tag{2.30}$$

Für eine abzubildende dritte Linie

$$a_3 = (\cos\phi_3, \sin\phi_3, -r_3)^T \tag{2.31}$$

ergeben sich dann unter Anwendung von $P^{(1)}$ die Berechnungsvorschriften für die Invarianten (ϕ', r')

$$\phi' = \phi_3 - \phi_1 + \frac{\phi_3}{2} \tag{2.32}$$

$$r' = r_3 + \frac{1}{\sin(\phi_1-\phi_2)}(r_2\sin(\phi_3-\phi_1) - r_1\sin(\phi_3-\phi_2)) \tag{2.33}$$

oder mit $P^{(2)}$

$$\phi' = \phi_3 - \phi_1 + \frac{\phi}{2} \tag{2.34}$$

$$r' = -r_3 - \frac{1}{\sin(\phi_1-\phi_2)}(r_2\sin(\phi_3-\phi_1) - r_1\sin(\phi_3-\phi_2)) \tag{2.35}$$

Diskretisierung und Anfrage unter Unsicherheiten

Für eine effiziente, mit Matchingfehlern arbeitende Umsetzung des Algorithmus ist es zunächst erforderlich, den Wertebereich der Invarianten zu diskretisieren, um mit Integer-Werten als Schlüssel- oder Hashwerten effizient arbeiten zu können. Diese Diskretisierung modelliert in gewisser Weise natürlich schon fehlerbehaftete Anfragen.

Um den tatsächlichen Fehler bei der Objekterkennung zu modellieren, verwendet man zwei konstante Fehlerwerte, welche das Fehlerintervall der Invarianten bestimmen. Für jede Invariante können die extrahierten Werte dann in den Intervallen $[\phi - \text{MAXR}, \phi + \text{MAXR}]$ bzw. $[r - \text{MAXD}, r + \text{MAXD}]$ liegen. In der Praxis haben sich MAXR=18 und MAD=2 bewährt.

Komplexität

Korollar 2.37 (Speicher- und Zeitkomplexität des Preprocessings) *Es bezeichne m die Anzahl der Modellfeatures. Dann benötigt das geometrische Hashing im Preprocessing-Schritt eine Laufzeit und einen Speicherplatzbedarf von $\mathcal{O}(m^{k+1})$, wobei k die Größe der Basis repräsentiert.*

Beweis: Sei k die (als konstant angenommene) Größe einer Basis. Es bezeichne weiter m die Anzahl der Modellfeatures. Dann ist die Anzahl der möglichen Modellbasen aus $\mathcal{O}(m^k)$, und im Preprocessing-Schritt (Algorithmus 2.11) ist die Anzahl der Kombinationen aus Modellfeature und Modellbasis, für die Invarianten in konstanter Zeit berechnet werden, aus $\mathcal{O}(m^{k+1})$. Also ist auch die Laufzeit und der Speicherplatzbedarf aus dieser Größenordnung, da Invarianten und Basis in einer Hashtabelle abgelegt werden.

□

Korollar 2.38 (Zeitkomplexität einer Anfrage) *Es bezeichne n die Anzahl der Bildfeatures und m die Anzahl der Modellfeatures. Dann benötigt das geometrische Hashing eine Anfrage-Laufzeit von $\mathcal{O}(n^{k+1} \cdot m^k)$*

Beweis: Betrachte zunächst den Fall für eine Bildfeaturebasis B mit Größe k. Dann sind für $n - k$ weitere Bildfeatures, welche nicht zur Basis gehören, in jeweils konstanter Zeit Invarianten zu bestimmen und entsprechende Basen aus der Hashtabelle auszulesen. Im schlechtesten Fall können dies alle gespeicherten Basen der Hashtabelle sein. Somit ergibt sich für eine gewählte Bildfeature-Basis eine Laufzeit aus $\mathcal{O}(n \cdot m^k)$.

Unter der optimistischen Annahme, bei Anfragen an die Hashtabelle jeweils eine beschränkte Anzahl an Modellbasen zu erhalten, ist die Laufzeit für eine gewählte Bildfeaturebasis linear in der Anzahl der Bildfeatures.

Werden alle möglichen Basen aus der Menge der Bildfeatures betrachtet, wie in Algorithmus 2.12 vorgesehen, müssen n^k Basen betrachtet werden.
Damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n^{k+1} \cdot m^k)$. □

Pruningmethoden zur Effizienzsteigerung

Realistische Karten, wie sie eingesetzt werden sollen, sind sehr komplex und besitzen eine große Anzahl von Features oder Segmenten. Gerade das geometrische Hashing geht hier sehr unintelligent vor und berechnet für jeweils drei beliebige Segmente der Karte alle möglichen Invarianten.

Da jedoch nur Laser-Entfernungsscans zum Einsatz kommen, kann auch deren Sichtbarkeitsbedingung verwendet werden. So können sich zwei Segmente nicht sehen, die sich mehr als 50 Meter entfernt voneinander befinden, da der Laser-Entfernungsmesser eine angenommene Reichweite von 50 Metern besitzt. Auch sind solche Segmente nicht sichtbar, die sich in unterschiedlichen Räumen befinden oder auf unterschiedlichen Seiten einer Wand.

Man kann die Sichtbarkeitsbedingung erfüllen, indem ein Vorverarbeitungsschritt vor der eigentlichen Invariantenberechnung abgearbeitet wird. Dabei wird jedem Segmentfeature der Modellfeature-Menge eine Menge von sichtbaren Segmenten zugeordnet.

Betrachte zunächst die Standarddefinition eines Graphen, bekannt aus der Graphentheorie [127]:

Definition 2.39 (Graph) *Ein gerichteter Graph (oder einfach Graph) ist ein Quadrupel* $G = (V, R, \alpha, \omega)$ *mit folgenden Eigenschaften:*

1. V *ist eine nicht leere Menge, die Eckenmenge des Graphen.*

2. R *ist eine Menge, die Pfeilmenge des Graphen.*

3. Es gilt $V \cap R = \emptyset$.

4. $\alpha : R \to V$ *und* $\omega : R \to V$ *sind Abbildungen, wobei* $\alpha(r)$ *die Anfangsecke und* $\omega(r)$ *die Entdecke des Pfeils* r *ist.*

Der Graph heißt endlich, wenn $|V \cup R| < +\infty$.

Definition 2.40 (Schlinge, parallel, invers) *Sei* $G = (V, R, \alpha, \omega)$ *ein Graph. Ein Pfeil* $r \in R$ *heißt Schlinge, wenn* $\alpha(r) = \omega(r)$. *Der Graph* G *heißt schlingenfrei, wenn er keine Schlingen enthält. Zwei Pfeile* r *und* r' *heißen parallel, wenn* $r \neq r'$ *und* $\alpha(r) = \alpha(r')$ *sowie* $\omega(r) = \omega(r')$. *Die Pfeile heißen invers, wenn* $\alpha(r) = w(r')$ *sowie* $\omega(r) = \alpha(r')$. *Ein ungerichteter Graph besitzt nur die Kanten* $e \in R$ *mit* $e = (u, v)$, *wobei* $u, v \in V$. *Man schreibt dann einfach* $G = (V, R)$.

Die im folgenden betrachteten Graphen sind alle ungerichtet. Dann kann man einen Sichtbarkeitsgraphen definieren in folgender Weise:

Definition 2.41 (Sichtbarkeit-Graph) *Ein Sichtbarkeitsgraph* $G_{vis} = (E, V)$ *besteht aus einer Eckenmenge* E, *repräsentiert durch die Mittelpunkte der Segmente und ungerichteten Kanten* V, *welche den minimalen Abstand und die Sichtbarkeit zu einem anderen*

Segment repräsentieren. Existiert keine Kante zwischen zwei Ecken, sind die Segmente auch nicht füreinander sichtbar.

Die Berechnung des kompletten Sichtbarkeitsgraphen kann jedoch nicht als Performancesteigerung gewertet werden, weshalb einfache Pruningmethoden Abhilfe schaffen sollen:

- **Nearest Neighbour Pruning**
  Die Idee des Nearest Neighbour Prunings ist es, eine fest vorgegebene Anzahl von nächsten Nachbarn um das betrachtete Segment herum zu verwenden. Vorteil dieses Verfahrens ist die Maßstabsunabhängigkeit und die Schnelligkeit.

- **Distanz Pruning**
  Die Idee des Distanz Prunings ist es, um ein betrachtetes Segment einen Kreis mit einem gewissen Durchmesser zu legen und dann diejenigen Segmente in der Karte zu verwenden, die diesen Kreis schneiden. Diese Segmente gelten als sichtbar. Vorteil dieses Verfahrens ist dessen Schnelligkeit. Nachteile ergeben sich daraus, dass u.U. zu wenig Segmente verwendet werden, wenn etwa die Karte nur aus wenigen Segmenten mit großen Abständen zwischen diesen besteht. Außerdem werden u.U. auch Segmente verwendet, die auf unterschiedlichen Seiten einer Wand liegen und damit in unterschiedlichen Räumen.

- **Sektorpruning**
  Beim Sektorpruning versucht man einen Sichtbarkeitsgraphen aufzubauen, wobei die Sichtbarkeit auf k Sektoren eines Kreises beschränkt ist. Innerhalb dieser Sektoren werden die p nächsten Nachbarn mit zum Preprocessing herangezogen. Dieses Verfahren kann die Nachteile des Distanzprunings beheben, indem wirkliche Sichtbarkeitsrelationen herangezogen werden. Jedoch ist der Aufwand zum Erstellen des reduzierten Sichtbarkeitsgraphen schon zu groß.

Für die obigen Pruningmethoden kann man folgende Komplexitätsabschätzungen herleiten:

Korollar 2.42 (Komplexität der Pruningmethoden) *Die Pruning-Verfahren haben eine Zeitkomplexität von $\mathcal{O}(m^2)$ und eine Speicherkomplexität von $\mathcal{O}(m \log m)$. Dabei ist m die Anzahl der Modellfeatures.*

Beweis: Für die Speicherung des reduzierten Sichtbarkeitsgraphen benötigt man m Ecken und im schlechtesten Fall m^2 Kanten, sodass sich die Speicherkomplexität zu $\mathcal{O}(m^2)$ ergibt.

Die Berechnung der nächsten Nachbarn kann mittels des Voronoi-Diagramms [121] erfolgen. Diese hat bei m Features eine Berechnungskomplexität von $\mathcal{O}(m \log m)$. Für jedes Segment betrachten die obigen Pruningmethoden eine konstante Anzahl von Features. Die Feststellung, ob sich zwei Segmente gegenseitig sehen können, liegt in $\mathcal{O}(m)$, da man jedes Segment mit einer festen konstanten Anzahl vergleichen muss. Insgesamt gibt sich eine Zeitkomplexität aus $\mathcal{O}(m \log m)$. □

Da sich im späteren Verlauf der Arbeit herausgestellt hat, dass das geometrische Hashing nicht die Erwartungen eines effizienten Matching-Algorithmus liefert (siehe Kapitel 2.8.2), wird dieser Ansatz nicht weiter verfolgt.

2.4 Transformationsbestimmung

Nächster Schritt auf dem Weg zur Bestimmung hypothetischer Konfigurationen für einen mobilen Roboter ist die Berechnung von Transformationen aus den im vorherigen Abschnitt berechneten Matchings. Es stellt sich also das Problem, aus einer Menge von Feature-Zuordnungen eine Konfiguration zu bestimmen, d.h. das Problem 2.13 zu lösen. Damit taucht auch die Frage nach der Genauigkeit der Transformation auf. Diese wird durch den maximalen Winkelfehler in der Orientierung und durch den maximalen räumlichen Fehler für den Ort des Roboters beschrieben. Der Fehler schlägt sich in der Definition von Lokalisationshypothesen nieder:

Definition 2.43 (Lokalisationshypothese) *Es bezeichne eine Konfiguration* p *des Konfiguratiosraumes* $\mathcal{C}$ *mit einem Unsicherheitsbereich* (e_{pos}, e_{orient}) *eine Lokalisationshypothese* $lhp = (p, e_{pos}, e_{orient})$. *Der Unsicherheitsbereich markiert die Positions- und Orientierungsungenauigkeit, mit welcher ein Lokalisationsalgorithmus diese Lokalisationshypothese generiert hat.*

2.4.1 Transformationsbestimmung aus wenigen Zuordnungen

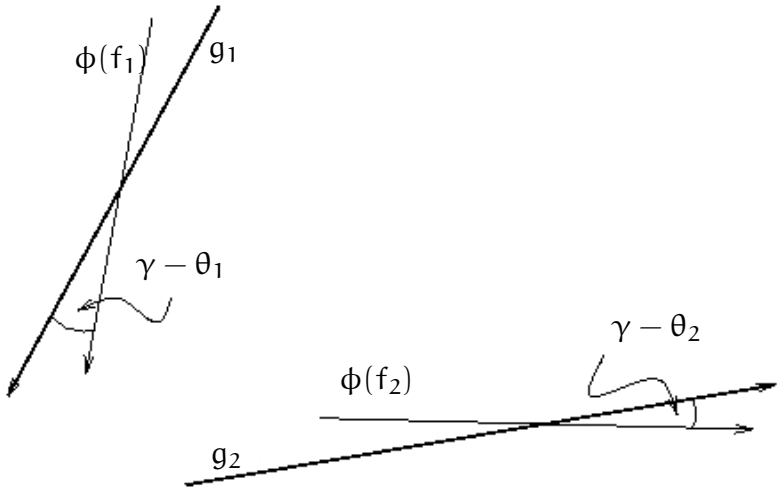
Sei nun ein Paar von Feature-Zuordnungen (f_1, g_1) und (f_2, g_2) mit $f_1, f_2 \in F$ und $g_1, g_2 \in G$ gegeben. Dabei sei vorausgesetzt, dass die beiden Modellfeatures g_1 und g_2 nicht parallel sind. Mit (g_i) sei für $i = 1, 2$ die orientierte Gerade bezeichnet, die das Kartensegment g_i trägt.

Dann hat die optimale Transformation ϕ die Eigenschaft, dass die Winkel zwischen $\phi(f_1)$ und g_1 sowie zwischen $\phi(f_2)$ und g_2 kleiner oder gleich $90°$ sind, da durch sie der Bereich erlaubter Rotationen für jede Segmentzuordnung ein abgeschlossenes Winkelintervall der Breite $180°$ ist, und der Schnitt zweier solcher Winkelintervalle nicht leer sein kann (siehe Abbildung 11).

Die Transformation ϕ soll weiter so gewählt sein, dass insbesondere $\phi(f_1)$ relativ zu (g_1) optimal liegt, d.h. der Mittelpunkt von $\phi(f_2)$ liegt auf (g_2). Außerdem soll der maximale Abstand von Punkten auf $\phi(f_1)$ zu (g_1) und von Punkten auf $\phi(f_2)$ zu (g_2) minimal sein.

Sei l_i die Länge von f_i und θ_i der Winkel zwischen f_i und g_i für $i \in \{1, 2\}$. Liegt der Mittelpunkt von $\phi(f_1)$ auf (g_1) und derjenige von $\phi(f_2)$ auf (g_2), so ist abhängig von der Rotationskomponente γ der Abbildung ϕ der maximale Abstand von Punkten

Abbildung 11: Transformationsbestimmung für Segmentfeatures



Beispiel für eine Transformation Φ, welche Bildfeatures f_i auf Modellfeatures g_i abbildet.

auf $\phi(f_1)$ zu (g_1) und von Punkten auf $\phi(f_2)$ zu (g_2) gleich

$$d_{max}(\gamma) = \max\left\{\frac{l_1}{2}\left|\sin(\gamma - \theta_1)\right|, \frac{l_2}{2}\left|\sin(\gamma - \theta_2)\right|\right\} \tag{2.36}$$

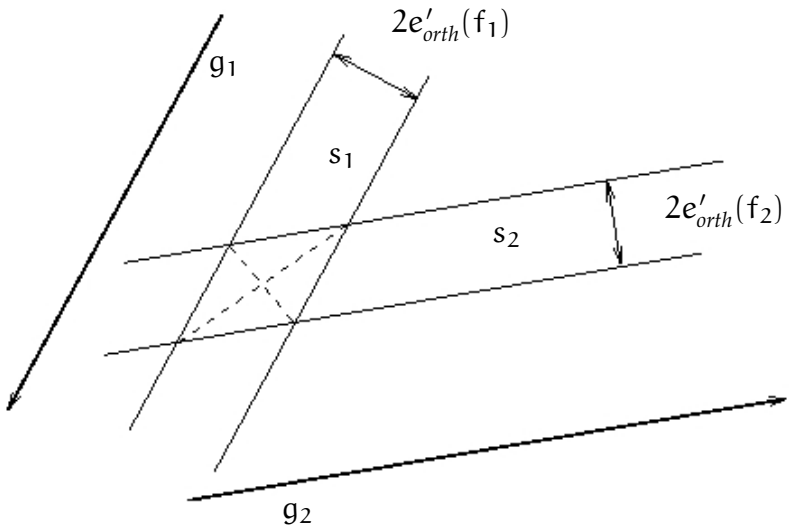
Durch Minimierung dieser Zielfunktion auf dem erlaubten Winkelbereich erhält man die Rotationskomponente γ der Abbildung. Mit der Bedingung, dass die Mittelpunkte der Bildfeatures auf die Geraden der Modellfeatures abzubilden sind, lässt sich die Translationskomponente bestimmen, so dass die komplette Transformation gewonnen werden kann.

Fehlerabschätzung der Ortes

Durch die Verwendung fehlerbehafteter Segmentfeatures kann die Bestimmung einer Transformation nicht fehlerfrei sein. Um die Güte der Transformation abzuschätzen, betrachte im folgenden eine Abbildung, welche einem Paar von Segmentfeatures (f_1, f_2) ein Paar Modellfeatures (g_1, g_2) zuordnet.
Der tatsächliche Standort des Roboters befindet sich im Schnitt zweier Streifen s_1 und s_2, die parallel zu den entsprechenden Kartensegmenten g_1 beziehungsweise g_2 verlaufen (Abbildung 12). Die Breite der Streifen entspricht den doppelten, orthogonalen Fehlern der Bildfeatures f_1 und f_2.

Als Abschätzung für die Positionsungenauigkeit e_{pos} kann die längere Diagonale des entstehenden (Schnitt-)Parallelogramms (nach Voraussetzung des vorigen Abschnittes

Abbildung 12: Fehlerabschätzung der Transformationsbestimmung



Der tatsächliche Standort des Roboters befindet sich im Schnitt der beiden Streifen s_1 und s_2.

sind g_1 und g_2 nicht parallel) dienen. Diese kann man mit Hilfe des Kosinussatzes bestimmen.

Dazu sei

$$e_i = 2 \cdot \frac{e_{\text{orth}}(f_i)}{\sin(\angle(g_1, g_2))} \tag{2.37}$$

für $1 \leq i \leq 2$. Dann kann man den Kosinussatz anwenden:

$$e_{\text{pos}} = e_1^2 + e_2^2 + 2 \cdot e_1 \cdot e_2 \cdot \cos(\angle(g_1, g_2)) \tag{2.38}$$

Als Abschätzung für die Orientierungsungenauigkeit e_{orient} wird das arithmetische Mittel der maximalen Orientierungsfehler der beiden Scansegmente verwendet.

$$e_{\text{orient}} = \frac{1}{2} \cdot (\Delta\phi_{\max}(f_1) + \Delta\phi_{\max}(f_2)) \tag{2.39}$$

Siehe dazu auch die Abschnitte 2.2.1 und 2.3.1.

2.4.2 Transformationsbestimmung aus größeren Matchings

Bei Matching-Verfahren, die mehr als eine minimale Anzahl von Zuordnungen erzeugen, möchte man alle vorhandenen Informationen nutzen, um eine möglichst genaue Abschätzung der Position zu erhalten. Betrachte also im folgenden ein Matching M

mit k Feature-Zuordnungen.

Der einfachste Weg größere Matchings zu behandeln besteht darin, Teilmengen der Zuordnungen zu verwenden und für alle diese mit oben beschriebenem Verfahren jeweils eine Transformation zu bestimmen. Anschließend verschmilzt man die erzeugten Transformationen zu einer einzigen Transformation (siehe Algorithmus 2.13). Somit wird genau eine Lokalisationshypothese für ein ausgezeichnetes Matching erzeugt. Dabei wird mit jeder weiteren Zuordnung der Fehler der Abschätzung kleiner, da

Algorithmus 2.13: Transformationsbestimmung

Aufruf: **transformation-estimation()**
Input: Feature-Matching fm
Output: Lokalisationshypothese lhp

```
1  LHP = ∅
2  Suche zulässige Segmentpaare aus dem Matching heraus, verbiete zu spitze und zu
   stumpfe Winkel.
3  while {Iteriere über alle zulässigen Feature-Zuordnungen} do
4     Hole die nächste Feature-Zuordnung fp.
5     Berechne die Konfiguration p aus der Transformation.
6     Berechne den Positionsfehler e_pos und Winkelfehler e_orient.
7     Füge lhp = (p, e_pos, e_orient) an LHP an
8  end while
9  if {LHP ≠ ∅} then
10    Berechne eine Transformation und damit EINE lhp.
11 end if
```

anschaulich die erlaubten Positionen für den Roboter geschnitten werden.

Das Verschmelzen der einzelnden Transformationen für ein Matching geschieht folgendermaßen:

Konfiguration p
: Als Roboterposition des Ergebnisses wird der Schwerpunkt aller Roboterpositionen der Eingabetransformationen verwendet. Für die Roboter-Orientierung wird die Verteilung der Orientierungen als Punkte auf dem Einheitskreis betrachtet. An der Stelle des größten Bogenabstands zwischen benachbarten Punkten wird der Kreis gewissermaßen „aufgetrennt" und auf dem nun linearen Stück das arithmetische Mittel der übrigen Orientierungen im Intervall gebildet.

Positionsfehler (e_{pos}, e_{orient})
: Als Positionsfehler wird der jeweils ermittelte minimale Positionsfehler und minmale Orientierungsfehler verwendet.

Die Komplexität zur Bestimmung aller Transformationen hängt wesentlich von der Anzahl der erzeugten Matchings ab.

Korollar 2.44 (Komplexität der Transformationsbestimmung) *Es bezeichne* p *die Anzahl der durch ein Matchingverfahren erzeugten Matchings* $\mathcal{M}$*, welche aus Zuordnungen zwischen Bildfeatures und Modellfeatures bestehen. Bezeichne* k *die größte Menge von Zuordnungen eines Matchings* M*. Dann beträgt die Zeitkomplexität* $\mathcal{O}(p \cdot k \cdot A)$*, wobei* A *den Aufwand repräsentiert, eine Transformation zu berechnen. Die Speicherkomplexität beträgt* $\mathcal{O}(p)$*.*

Beweis: Es werden durch ein Matching-Verfahren p Matchings bestimmt. Jedes Matching kann maximal k Feature-Paare tragen. Aus diesen Feature-Paaren kann im einfachen Fall für 2 Feature-Paare eine Transformation mit einem konstanten Aufwand A bestimmt werden. Für k Feature-Paare werden k Transformationen bestimmt (Algorithmus 2.13) und diese dann miteinander verschmolzen.

Man benötigt zusätzlichen Speicher für die Speicherung der berechneten Transformationen, welche von der Anzahl der Matchings abhängen, sodass sich ein Speicherplatzbedarf von $\mathcal{O}(p)$ ergibt. □

Im Falle des Alignment-Verfahrens ergeben sich $\mathcal{O}(n^2m^2)$ mögliche Matchings der Größe $k = 2$ für $\mathcal{M}$, sodass man auf $\mathcal{O}(n^2m^2)$ Zeitschritte kommt, da $k \cdot A$ als konstant angenommen werden kann.

2.5 Verifikation durch Projektion

Der Verifikationsschritt der modellbasierten Objekterkennung dient dazu, mittels der gesamten zur Verfügung stehenden Informationen die Menge der erzeugten Lokalisationshypothesen $H^k_{n\nu}$ zu verifizieren und damit Problem 2.14 zu lösen. So werden die getroffenen Entscheidungen bei der Modellierung der Unsicherheiten im Ergebnis überprüft und falsche hypothetische Positionen eliminiert.

Für die Verifikation kommen diejenigen Features in Frage, die für die Bestimmung des Matchings keine Rolle gespielt haben. Diese noch nicht zugeordneten Features des Bildes werden mittels der zur Lokalisationshypothese gehörigen Transformation auf die Modelldaten projiziert und in deren Umgebung nach passenden Zielfeatures gesucht. Dabei ist die Ausdehnung der zu durchsuchenden Umgebung von den bei der Extraktion gemachten Fehlern abhängig.

Um zu einer Entscheidung über Akzeptanz oder Ablehnung der Lokalisationshypothese zu gelangen wird ein Schwellenwert BEST-MATCH festgelegt der angibt wie groß der prozentuale Anteil zugeordneter Bildfeatures mindestens sein muss um eine Hypothese zu akzeptieren. Alternativ können auch Lokalisationshypothesen akzeptiert werden, die eine maximale Anzahl von Features zuordnen, oder diejenigen, die den von zugeordneten Bildfeatures überdeckten Winkelbereich maximieren.

Algorithmus 2.14: Verifikation durch Projektion

Aufruf: **verify-hypotheses()**
Input: Menge von nicht-verifizierten Lokalisationshyptoehsen H^k_{nv}
Output: Menge von Hypothesen H^k

```
1  Lösche alle Feature-Zuordnungen, die außerhalb der Karte liegen.
2  for {alle Feature-Zuordnungen} do
3     Hole die Abbildung für das Matching.
4     for {alle für dieses Matching nicht zugeordneten Bildfeatures} do
5        Projeziere das Feature f_i mittels der Abbildung Φ_i auf die Modelldaten.
6        Suche abhängig vom Zuordnungsfehler in Suchstruktur nach Modellfeatures.
7        Wähle ein passendes Modellfeature abhängig von Distanz aus.
8        Füge es in Feature-Zuordnung ein.
9     end for
10    Merke die größte Kardinalität einer Zuordnung.
11 end for
12 Wähle die besten Zuordnungen > BEST-MATCH Prozent der größten Feature Zu-
   ordnungen aus und füge sie an H^k an.
```

Korollar 2.45 *Es bezeichne* q *die Anzahl der erzeugten Lokalisationshypothesen* H^k_{nv} *die verifiziert werden sollen. Dann beträgt der Verifikationsaufwand nach Algorithmus 2.14* $\mathcal{O}(q \cdot m \cdot n^{3/4})$ *Zeitschritte. Dabei bezeichnet* n *die Anzahl der Modellfeatures und* m *die Anzahl der Bildfeatures.*

Beweis: Die Modellfeatures wurden effizient in einem kd-Baum abgespeichert. Eine Anfrage benötigt dann $\mathcal{O}(n^{3/4})$ Zeitschritte. Für jede Lokalisationshypothese müssen nun die Bildfeatures auf die Karte abgebildet, und für jedes Bildfeature muss ein passendes Kartenfeature gefunden werden. Pro Lokalisationshypothese hat man also einen Aufwand von $\mathcal{O}(m \cdot n^{3/4})$. Insgesamt ergibt sich ein Verifikationsaufwand von $\mathcal{O}(q \cdot m \cdot n^{3/4})$. □

Betrachtet man wiederum das Alignment-Verfahren als eingesetztes Verfahren zur Bestimmung der Menge der Lokalisationshypothesen (und damit der Kardinalität), ergibt sich ein Verifikationsaufwand von $\mathcal{O}(n^2 m^2 m \cdot n^{3/4}) = \mathcal{O}(n^{2+3/4} \cdot m^3)$.

2.6 Algorithmischer Ablauf der featurebasierten Lokalisation

In den letzten Abschnitten wurden jeweils Teilaspekte eines feature-basierten Lokalisationsverfahrens betrachtet. Das vollständige Verfahren gibt Algorithmus 2.15 wieder. Die Menge der Modellfeatures G ist in der Regel durch das Kartenmodell vorgegeben. Für die Generierung der Bildfeatures F kann man ein Verfahren zur Segmentreduktion wählen. Aus den Mengen F und G berechnet man mittels des Alignment-Verfahrens, des Pose-Clusterings oder des geometrischen Hashings zu einem Zeitpunkt k eine Menge von Matchings $\mathcal{M}_k$. Aus dieser werden hypothetische Positionen H^k_{nv} generiert,

welche zuletzt noch einen Verifikationsschritt durchlaufen müssen, um als Hypothesenmenge H^k akzeptiert zu werden.

Algorithmus 2.15: Feature-basierte Lokalisation

Aufruf: **feature-based-localisation()**
Input: Sensordaten und Umgebungsdaten
Output: Menge von Hypothesen H^k

1 Generiere Bildfeatures F aus Sensordaten.
2 Generiere Modellfeatures G aus Umgebungsdaten.
3 Berechne mit Hilfe eines Matching-Verfahrens eine Menge von Matchings $\mathcal{M}_k$.
4 Bestimme Transformationen und eine vorläufige Hypothesenmenge H^k_{nv}
5 Verifiziere die erzeugten Positionen aus H^k_{nv}, füge gute Hypothesen der Menge H^k hinzu.
6 Liefere die Menge der Hypothesen H^k zurück.

Die Zeitkomplexität einer feature-basierten Lokalisationsanfrage wird durch die verwendeten Algorithmen zur Lösung der Teilprobleme bestimmt, ebenso die Speicherkomplexität. Für das Alignment-Verfahren ergibt sich der Aufwand zu $\mathcal{O}(n^{2+3/4} \cdot m^3)$ und wird deutlich vom Verifikationsaufwand bestimmt. Die Speicherkomplexität ergibt sich aus dem Aufwand zur Speicherung der Matchings, nämlich $\mathcal{O}(n^2 \cdot m^2)$.

Somit steht ein Framework von Verfahren zur Verfügung, welches für einen Roboter mittels Sensordaten und einem Umgebungsmodell hypothetische Standorte im Umgebungsmodell berechnet, und damit Problem 2.4 löst. Die Tauglichkeit dieses Verfahrens soll in den folgenden beiden Abschnitten geklärt werden.

2.7 Bewertung von Hypothesen und Lokalisationsverfahren

Für spätere Anwendungen ist es erforderlich Gütekriterien zu definieren, um Aussagen über die Güte der Verfahren und der erzeugten Hypothesen treffen zu können.

2.7.1 Güte von Hypothesen

Wie in den beiden letzten Abschnitten vorgestellt, wird die Position einer berechneten Hypothese mit einem Positionsfehler und einem Orientierungsfehler versehen. Diese Gütekriterien sind jedoch nur lokale Aussagen über generierte Hypothesen. Was von größerem Interesse ist, ist vielmehr die Fragestellung nach der Anzahl nicht generierter Hypothesen. Dafür benötigt man jedoch mehr Informationen, welche in der Praxis nicht so einfach zu beschaffen sind, nämlich den wahren Standort an welcher die Lokalisationsanfrage gestartet wurde. Damit kann man die im folgenden verwendete Güte definieren:

Definition 2.46 (Positive Lokalisation) *Eine Lokalisationsanfrage wird positiv bewertet, falls eine der Hypothesen aus der durch ein Lokalisationsverfahren L erzeugten Hypothesenmenge H^k zu einem Zeitpunkt k höchstens $e_{pos}^{max} = 0.25$ m und $e_{orient}^{max} = 10$ Grad vom tatsächlichen Standort entfernt ist und die Kardinalität der Hypothesenmenge H^k einen fest gewählten Schwellenwert $e_{anzahl}^{max} = 10$ nicht übersteigt.*

Die Kardinalität der Hypothesenmenge ist dabei entscheidend, könnte man sich ansonsten ein Verfahren vorstellen, welches nur zufällig Hypothesen erzeugt und somit eine beliebige Güte erreicht würde.

2.7.2 Güte von Lokalisationsverfahren

Um die vorgestellten Matchingverfahren und damit Lokalisationsverfahren miteinander zu vergleichen, benötigt man ein abhängig von der gegebenen Karte definiertes Gütekriterium:

Definition 2.47 (Lokalisationsgüte, Güte eines Lokalisationsverfahrens) *Es bezeichne $n > 0$ die Anzahl von (randomisiert) generierten Konfigurationen p_i in der gegebenen Karte m_g. Es bezeichne weiterhin L ein feature-basiertes Lokalisationsverfahren sowie F und G die Bildfeature- und Modellfeature-Mengen. Die Güte eines Lokalisationsverfahrens, oder kurz Lokalisationsgüte, bezeichnet eine Funktion $g(L, F, G, p_i)$, welche dem 4-Tupel (L, F, G, p_i) einen Wert aus $[0, 1]$ zuordnet, welcher die prozentuale Anzahl der m positiv bewerteten Lokalisationsversuche an den n ausgewählten Standorten in der gegebenen Karte angibt.*

Für eine Vergleichbarkeit von Lokalisationsverfahren wird weiter vorausgesetzt, dass die Sensordaten reproduzierbar und für jedes Lokalisationsverfahren identisch sind. Diese Bedingungen sind für synthetisch erzeugte Karten sicher erfüllt. Ein Verfahrensvergleich mit realistischen Daten wird in Kapitel 4 vorgestellt, da dafür Methoden zur Kartierung in Kapitel 3 erarbeitet werden müssen.

2.8 Verfahrensvergleich in synthetischen Umgebungen

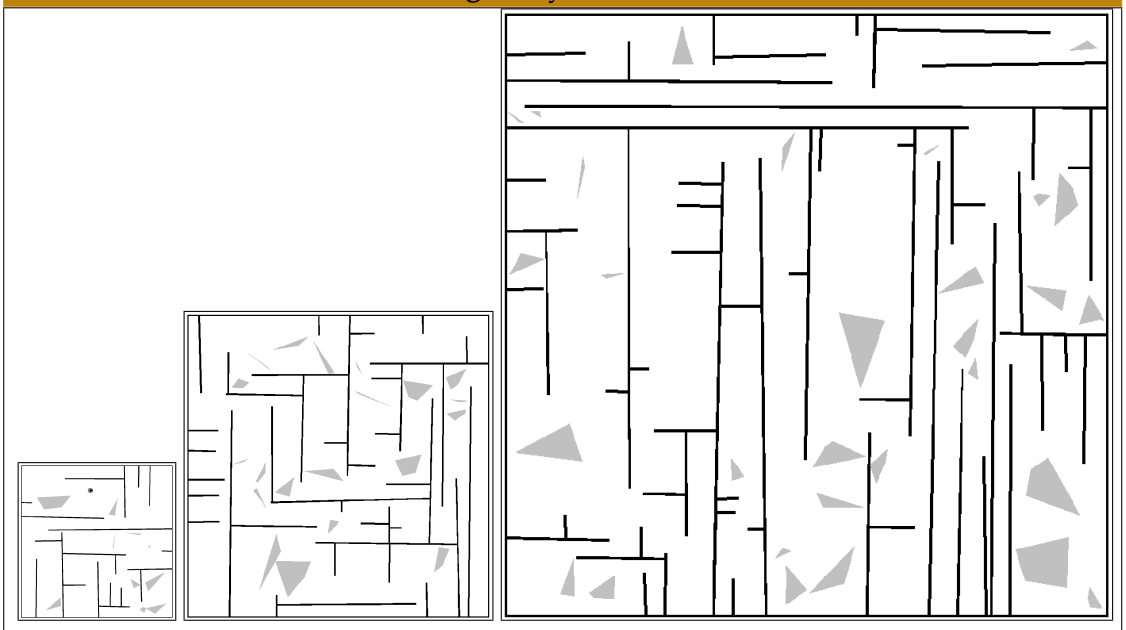
Obige vorgestellten Matchingverfahren (Alignment, Pose-Clustering, Geometrisches Hashing) sind zunächst mit synthetischen Daten getestet worden, um unter idealisierten Umweltumgebungen möglichst gute Anfragewerte zu bekommen.

2.8.1 Anforderungen an die simulierten Daten

Für die Simulation wurden einfache polygonale Karten mit einfachen einbeschriebenen polygonalen Hindernissen erzeugt (siehe Abbildung 13). Folgende Messdaten wurden in den Simulationsdaten variiert:

- Kartengröße
  Die Testläufe wurden auf quadratischen Karten durchgeführt. Ausgehend von

Abbildung 13: Synthetische Karten



Unterschiedliche Kartengrößen synthetischer Karten. Abgebildet ist jeweils eine Karte der Größe 1, Größe 2 und 3 (von links nach rechts). Ein Roboter im Vergleich dazu ist als Punkt in der kleinsten Karte (Mitte-Oben) zu erkennen.

einer Grundkarte von 300 Längeneinheiten Kantenlänge, 20 Wänden und 10 Hinderissen wurde diese mit den Faktoren 1 bis 4 multipliziert. Dieser Faktor repräsentiert im weiteren die Kartengröße. Die Karten wurden so erzeugt, dass fast rechtwinklige Segmentanordnungen vorherrschen.

- Rauschen der Scandaten
  Die simulierten Scandaten wurden mit einem Grundrauschen versehen, welches gleichverteilt war und zwischen 2 und 10 Prozent in 2 Prozent-Schritten erhöht wurde.

- Kompassungenauigkeit
  Die Kompassungenauigkeit wurde mit $\pm 5°$ angegeben.

- Lokalisationsanzahl
  Es wurden zu jeder erzeugten Karte 50 Punkte im Inneren zufällig erzeugt und diese als Ausgangspunkt für den Standort eines Roboters genommen. Insgesamt wurden für jede Kartengröße 20 Karten erzeugt.

Für jedes Lokalisationsverfahren wurden also insgesamt $4 \cdot 5 \cdot 20 \cdot 50 = 20.000$ Lokalisationsanfragen durchgeführt und ausgewertet. Eine Lokalisationsanfrage hatte dabei außerdem Zeit- und Speicherbegrenzungen zu erfüllen. Je nach Kartengröße war das Zeitintervall, in welcher die Berechnung ausgeführt werden konnte, begrenzt. So war

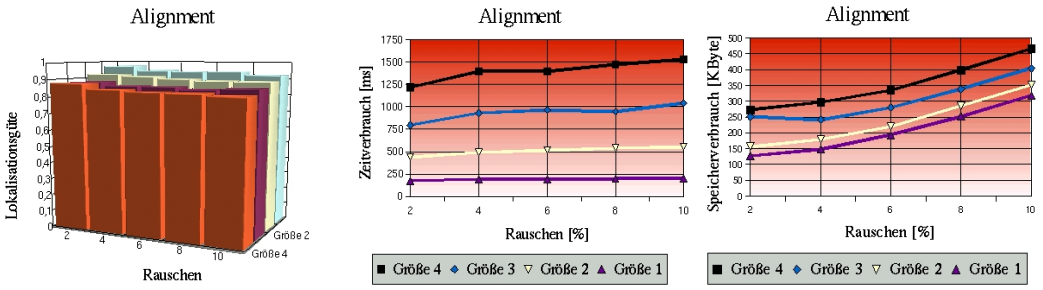
die Zeitbeschränkung 1 Minute bei Kartengröße 1. Außerdem war eine Speicherbegrenzung auf 200 Megabyte in allen Testläufen festgesetzt.

2.8.2 Ergebnisse mit synthetischen Daten

Man kann im Vergleich der Abbildungen 14, 15 und 16 erkennen, dass das Alignment die besten Ergebnisse erzielt. Die Genauigkeit der Verfahren sinkt abhängig von der Kartengröße und vom Scannerrauschen. Mit zunehmender Kartengröße steigt auch die Anzahl der abgebrochenen Lokalisationen, da nur beschränkte Ressourcen vorhanden waren. Der Speicher- und Zeitverbrauch beim Alignment ist im Vergleich mit den anderen beiden Verfahren gering. Pose-Clustering liefert außerdem bessere Ergebnisse als das geometrische Hashing und ist weniger zeit- und speicherintensiv. Insgesamt ist Alignment also Testsieger, gefolgt von Pose-Clustering. Das geometrische Hashing bildet das Schlusslicht.

Was für die Praxis noch entscheidend ist: auch Alignment erreicht mit fast optimalen Bedingungen nur ca. 90 Prozent Lokalisationsgüte. In der Praxis ist durch diesen Vergleich für alle Verfahren mit weiteren Performanceeinbußen zu rechnen.

Abbildung 14: Lokalisationsgüte Alignment

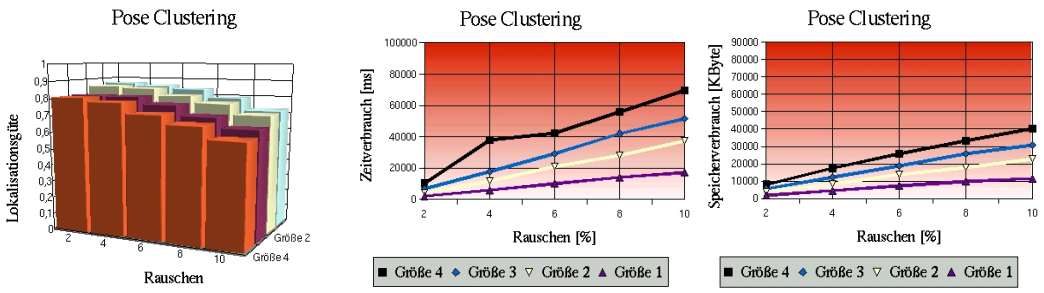


Lokalisationsgüte des Alignment-Verfahrens (links) sowie Zeit- und Speicherverbrauch des Verfahrens (mitte und rechts).

2.9 Resümee

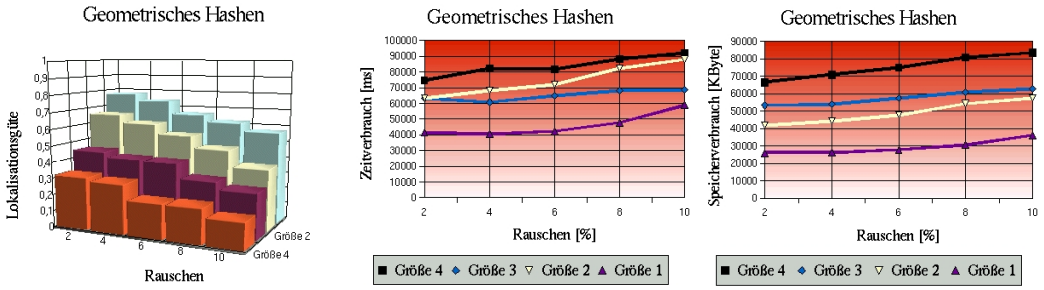
In diesem Kapitel wurde die Lösung des Lokalisationsproblems mit Hilfe eines featurebasierten Ansatzes vorgestellt. Es wurden neben dem Framework eines allgemeinen Verfahrens drei klassische unterschiedliche Matchingverfahren vorgestellt und an die Lokalisationsprobematik angepasst. Diese Verfahren wurden mit synthetischen Daten gegeneinander getestet um obere Güteschranken der Verfahren unter optimalen Bedingungen zu bekommen.

Abbildung 15: Lokalisationsgüte Pose-Clustering



Lokalisationsgüte des Pose-Clusterings (links) sowie Zeit- und Speicherverbrauch des Verfahrens (mitte und rechts).

Abbildung 16: Lokalisationsgüte Geometrisches Hashing



Lokalisationsgüte des geometrischen Hashings (links) sowie Zeit- und Speicherverbrauch des Verfahrens (mitte und rechts).

Als wichtiges Problem bleibt im nachhinein die einfache und schnelle Erzeugung realistischer Karten der Umgebung, die nicht zu detailliert sind, jedoch die Besonderheiten des Gebäudes wiederspiegeln. Grundrißkarten zu zeichnen ist zu aufwendig und geht an der Realität vorbei, insbesondere bei rasch wechselnden Umgebungen. Hier sind Verfahren zur automatischen Kartengenerierung gefragt, die dann außerdem eine genauere interne Repräsentation liefern können und so unabhängig von baulichen Abweichungen sind. Dieser Schritt wird im Kapitel 3 bewältigt.

Ansätze für weitere Forschung bietet die Möglichkeit, zusätzliche Feature-Typen in die Problematik zu integrieren [113], welche hier unberücksichtigt gelassen wurden. Zusätzlich müßten weitere Matching-Algorithmen gegen die realisierten Verfahren getestet werden um Performance-Aussagen gewinnen zu können. Eine andere offene Problematik besteht darin, die Laufgeschwindigkeit der Verfahren durch Pruningmethoden zu erhöhen ohne die Lokalisationsgüte wesentlich zu mindern. Dieser Schritt wird in Kapitel 4 behandelt.

3 Generierung feature-basierter Karten

Karten kommt beim Problem der globalen Selbstlokalisation eine große Rolle zu. Zum einen werden sie als zugrundeliegendes Umgebungsmodell für die feature-basierten Verfahren aus dem letzten Kapitel benötigt, zum anderen sollen Scans zu lokalen Karten kombiniert werden um als Anfragemodell mit dem ursprünglichen Modell gematcht zu werden (siehe Kapitel 6). In diesem Kapitel sollen deshalb die folgenden Fragestellungen geklärt werden:

Generieren von 2D-Modellen
: Wie generiert man aus Scandaten und Odometriedaten effizient überlagerte 2-D-Modelle, welche die Umwelt korrekt wiedergeben?

Segmentkarten
: Wie bekommt man aus korrigierten Sensordaten feature-basierte Karten als Modell- und Bilddaten für die feature-basierten Lokalisationsverfahren?

Kartengüte
: Wie gut sind die generierten Karten? Welcher Kartentyp liefert die besten Ergebnisse für Lokalisationsverfahren?

Um diese Fragestellungen zu klären, müssen zunächst Sensordaten generiert werden. Dies kann durch einen menschlichen Supervisor mit Hilfe eines mobilen Roboters durch eine Kartierungsfahrt in der Einsatzumgebung geschehen. Ein autonomer Einsatz des mobilen Roboters zur sogenannten Exploration wird in Kapitel 6.4 vorgestellt. Mit einem möglichst geringem Bewegungsaufwand und damit auf möglichst kürzestem Weg soll der Roboter die Einsatzumgebung explorieren.

Die aufgenommenen Sensordaten sind meistens stark verrauscht und müssen mit Hilfe von Kartierungsverfahren korrigiert werden. Dabei taucht das Problem auf, dass in größeren Umgebungen einmal explorierte Bereiche konsistent mit neuen Daten verrechnet werden müssen. Prominentes Beispiel dafür sind zyklische Umgebungen, in welchen man zu einem Punkt zurückkehrt, welchen man schon einmal entdeckt hat. In der Regel gibt es dann die Problematik, dass der Zykel in den aufgenommenen Daten nicht entdeckt wird. Dieses Erkennen und Korrigieren von Zyklen ist in der Literatur vielfach aufgegriffen worden [40, 196, 206, 218].

Die Problemstellung kann außerderm um dynamische Umgebungen erweitert werden. Damit wird landläufig die Tatsache ausgedrückt, dass die zu explorierende

Umgebung nicht menschenleer ist, sondern dass darin gearbeitet wird, während man diese exploriert. Demzufolge werden bei der Kartierung dynamische Hindernisse in den Sensordaten auftauchen, die berücksichtigt werden müssen. Des weiteren spielen dynamische Objekte eine wichtige Rolle. Damit sind Objekte der Karte gemeint, die unterschiedliche Zustände annehmen, etwa Türen die offen oder geschlossen sein können. Bisher gibt es zu diesen Problemstellungen nur wenige Lösungsansätze [169, 195, 208], welche sich auf die Objektverfolgung in aufeinanderfolgenden Scandaten spezialisieren.

Sind die Sensordaten korrekt miteinander verschmolzen worden, sollen aus diesen effizient Features extrahiert werden [79, 173]. Dazu muss beachtet werden, dass Features in einzelnen Sensordaten nur teilweise zu sehen sind. Nahe beieinander-liegende Sensordaten müssen auf geeignete Art miteinander verschmolzen werden, so dass ein feature-basiertes Lokalisationsverfahren (siehe Kapitel 2) damit gute Lokalisationsergebnisse bewerkstelligen kann.

In diesem Kapitel werden Lösungsverfahren für die angesprochenen Problematiken vorgestellt und zu einem allgemeinen Konzept zusammengefasst. Dabei wird insbesondere berücksichtigt, dass Lösungskonzepte einzelner Problemstellungen austauschbar sind.

3.1 Problemstellungen der Kartierung

Um einen Roboter in einem Gebäude operieren zu lassen, muss normalerweise eine zwei- oder dreidimensionale Karte der Umgebung generiert werden. Karten können unterschiedlichste Repräsentationen besitzen und sind für Applikationen, bei denen Roboter in Gebäuden agieren sollen, unverzichtbar.

3.1.1 Kartenrepräsentationen

Fundamental für die Funktionalität eines autonomen mobilen Roboters ist das Design von Datenstrukturen für die Speicherung von Karten der Umgebung, in denen der Roboter agieren soll. Zunächst ist zu klären, was eine Karte ist. Für die hier wesentliche Definition einer Karte benötigt man zunächst Kartenprimitive, die Objekte beschreiben, welche in der Karte vorkommen.

Definition 3.1 (Kartenobjekt, Kartenfeature) *Ein Kartenobjekt* o *ist ein einfach zu speicherndes Objekt mit einer Reihe von* k *Attributen* t_i, $o = \{t_i \mid 1 \leq i \leq k\}$ *, die es von anderen Kartenobjekten eindeutig unterscheidet.*

Üblicherweise handelt es sich bei Kartenobjekten um Hindernisse, Wände, Türen, Gegenstände oder speziell hier Segmente. Im Kontext der feature-basierten Lokalisation wird anstatt Kartenobjekt auch synonym der Begriff Kartenfeature verwendet.

Oft wird eine Karte auch über den Arbeitsraum, die Konfigurationen, die Hindernisse und den freien Raum bestimmt (siehe Kapitel 1.1). Hier wird eine allgemeinere Definition gegeben:

Definition 3.2 (Karte) *Eine Karte* $m_{gen} = \{o_i \mid 1 \leq i \leq n\}$ *ist eine diskretisierte Repräsentation einer realen Umgebung durch eine Menge von Kartenobjekten* o_i*, die effizient gespeichert sind. Die Karte erlaubt das Einfügen, Lokalisieren und Entfernen von Kartenobjekten.*

Eine **ideale** Datenstruktur zur Speicherung einer Karte sollte in der Lage sein, die Kartenobjekte so effizient wie möglich zu speichern. Außerdem sollte die Kartenstruktur eine effiziente Pfadplanung und Lokalisationsanfragen ermöglichen. Die Integration von verschiedensten Sensordaten bei der Kartierung in die Datenstruktur und die Repräsentation des Zustands eines Kartenobjektes sollten ebenfalls eine solche Datenstruktur auszeichnen. In den folgenden Abschnitten werden mehrere Kartenrepräsentationen vorgestellt. Die Datenstrukturen sind weitgehend sensorunabhängig und auf eine zweidimensionale Kartenrepräsentation beschränkt.

Topologische Karten

Topologische Repräsentationen einer Umgebung finden sich in der Robotik an vielen Stellen [16, 18, 132, 137, 183].

Definition 3.3 (topologische Karte) *Eine topologische Karte* m_{top} *speichert die Nachbarschaftsrelation zwischen den verschiedenen Kartenobjekten. Die resultierende Kartenrepräsentation ist ein Graph* $m_{top} = (V_t, R_t)$*, der in den Knoten* V_t *die Kartenobjekte und in den Kanten* R_t *die Nachbarschaftsrelation zwischen zwei Kartenobjekten speichert.*

Topologische Karten versuchen, eine abstrakte Definition der Nachbarschaftsbeziehungen der Kartenobjekte zu liefern. Metrische Informationen sollen dabei so gut es geht vermieden werden. Diese Kartenrepräsentation ist kompakt und stabil im Bezug auf Sensorungenauigkeiten, da sie jeweils einen ganzen Bereich der Karte beschreibt. Topologische Karten werden nicht direkt aus Sensordaten gewonnen, sondern meist aus Karten anderen Typs extrahiert, da die Extraktion topologischer Nachbarschaftsbeziehungen aus Sensordaten aufgrund der geringen Informationen einzelner Daten nicht möglich ist.

Die Speicherung eines (parametrisierten) Graphen erfolgt als eine doppelt verkettete Liste von Knoten und Kanten. Das Einfügen und Entfernen benötigt eine Zeitkomplexität aus $\mathcal{O}(m + n)$ wobei n die Anzahl der Ecken und m die Anzahl der Kanten repräsentiert. Nicht miteingerechnet sind dabei die Kosten für die Vergleiche zwischen den einzelnen gespeicherten Kartenobjekten in den Knoten. Der Speicherbedarf ist aus $\mathcal{O}(m + m)$.

Hier werden topologische Karten zur Speicherung ganzer Laser-Entfernungsscans verwendet. Ein Kartenobjekt besteht dann aus einem Scan, welcher durch eine Aufnahmeposition in Weltkoordinaten und seine Sensordaten an dieser Position repräsentiert wird. Die topologische Karte speichert somit die korrigierten rohen Sensordaten verlustfrei ab, wodurch Korrekturen der Sensorpositionen eine Korrektur der Karte ermöglichen.

Geometrische Karten

Der Begriff geometrische Karte ist zunächst ein Oberbegriff für eine Menge von Kartenrepräsentationen. Unter den Begriff fallen sowohl Gitterkarten als auch feature-basierte Karten.

Definition 3.4 (geometrische Karte) *Eine geometrische Karte speichert die Kartenobjekte mit ihrer absoluten Position in Weltkoordinaten ab.*

Geometrische Karten bieten durch das Speichern der metrischen Informationen die Möglichkeit, sehr genaue Repräsentationen durch unbegrenzte Vielfalt an geometrischen Primitiven zu ermöglichen. Mit der Genauigkeit der Repräsentation steigt auch der Verwaltungsaufwand der Datenstrukturen, weshalb oft nur grobe Primitive mit Größen- und Positionsunsicherheit gespeichert werden und damit eine Diskretisierung der Karte erreicht wird. Da diese Kartenrepräsentation am häufigsten anzutreffen ist, werden nachfolgend noch spezielle Klassen vorgestellt.

Polygonale Karten

Zunächst soll die Definition eines einfachen Polygons vorgeschaltet werden:

Definition 3.5 (Polygon, einfaches Polygon) *Ein Polygon* $P = (p_1, \ldots, p_n)$ *ist gegeben durch eine endliche Folge von Punkten* $p_i \in \mathbb{E}^2$*. Der Rand* P^R *des Polygons* P *ist die Menge der Punkte auf den Kanten* $\overline{p_i p_{i+1}}$ *für* $1 \leq i < n$*. Ein Polygon heißt einfach, wenn sich nur benachbarte Kanten schneiden und dies jeweils nur in dem gemeinsamen Eckpunkt.*

Polygone zerteilen die Ebene $\mathbb{E}^2$ in einen beschränkten Teil (das Innere des Polygons) und in einen unbeschränkten Teil (das Äußere des Polygons). Im folgenden wird mit dem Begriff Polygon immer nur ein einfaches Polygon gemeint.

Mit dem Begriff des Polygons kann eine polygonale Karte definiert werden, die für theoretische Betrachtungen von Bedeutung ist:

Definition 3.6 (Polygonale Karte) *Eine Karte* $m_{pol} = (P, B)$ *ist durch einen Polygonrand* P^R *eines Polygons* P *und eine Menge* $B = \{b_1, \ldots, b_k\}$ *von einfachen Polygonen gegeben. Die Menge* B *repräsentiert eine Menge von Hindernissen, welche im Inneren von* P *enthalten und paarweise disjunkt sind.*

Die Speicherung der Karte erfolgt durch Speicherung der Liste der Punkte des Randpolygons und der Liste der Punkte der Hindernispolygone. Der Test, ob ein Punkt im zweidimensionalen freien Raum der Karte liegt oder nicht, wird bei k Hinderispolygonen und einem Randpolygon mit $k+1$ Point-in-Polygon-Tests durchgeführt.

Dieser Kartentyp, meistens ohne einbeschriebene Hindernisse, wird für Algorithmen der Algorithmischen Geometrie benötigt, etwa bei der Exploration oder Hypothesenelimination (siehe Kapitel 6).

Gitterkarten oder Probability Grids

Ein Ansatz einer gitterbasierten Karte besteht darin, Sensorinformationen durch Wahrscheinlichkeitsprofile in das Gitter abzulegen, die die Sicherheit der Existenz eines Objektes in einer individuellen Gitterzelle bestimmen. Dieser Ansatz wurde von Alberto Elfes [64] in die Robotik eingeführt und erfreut sich großer Beliebtheit [38,178,189,212].

Definition 3.7 (Gitterkarte) *Eine 2-D Gitterkarte m_{grid} ist die Diskretisierung einer Umgebung in ein zweidimensionales Raster, $m_{grid} = \{g(i,j) \mid 1 \leq i \leq n, 1 \leq j \leq m\}$. Jedes Feld $g(i,j)$ repräsentiert dabei einen Zustand eines zweidimensionalen Bereiches*

$$[x_0 + i \cdot x, x_0 + (i+1) \cdot x] \quad \times \quad [y_0 + i \cdot y, y_0 + (i+1) \cdot y]$$

der Umgebungskarte. Dabei ist x die Feinheit der Diskretisierung in X-Richtung und y die Feinheit der Diskretisierung in Y-Richtung. x_0 und y_0 repräsentieren den Beginn des Gitters in Weltkoordinaten. Punkte, die durch das Feld $g(i,j)$ repräsentiert werden, besitzen eine uniforme Repräsentation durch den Inhalt des Feldes.

Um eine zweidimensionale Position in Weltkoordinaten auf die Gridkarte abzubilden, bzw. um einen Gridpunkt auf eine Weltkoordinate abzubilden, werden zwei Übergangsfunktionen definiert: Der Zugriff auf die Gitterkarte erfolgt durch die sogenannte Diskretisierungsfunktion $dis : \mathbb{R} \times \mathbb{R} \to \mathbb{N} \times \mathbb{N}$ durch

$$dis(x,y) = (round(\frac{x}{f} + x_0), round(\frac{y}{f} + y_0)) \tag{3.1}$$

und bildet einen Punkt (x,y) nach $g(i,j)$ ab.

Die inverse Diskretisierungsfunktion, $dis^{-1} : \mathbb{N} \times \mathbb{N} \to \mathbb{R} \times \mathbb{R}$, definiert durch

$$dis^{-1}(i,j) = (f \cdot (i + x_0), f \cdot (j - y_0)) \tag{3.2}$$

bildet einen Gridpunkt auf eine Koordinate in Weltkoordinaten ab.

Bei der Wahl der Zellgröße existiert ein Zusammenhang zwischen möglichst hoher Auflösung der Umgebung und dem verwendeten Speicherplatzbedarf des Gitters. Wird die Zellgröße zu gering gewählt, können unter Umständen wichtige Details der

Umwelt nicht erfasst werden. Wird die Zellgröße zu groß gewählt, verschlingt die Karte viel Speicherplatz.

Gitterkarten haben keine dynamische Kartenstruktur, da die initiale Größe des Gitters in der Regel nicht verändert wird. Zudem gehen bei der Speicherung Informationen verloren, da die Ansammlung der Informationen in Gitterpunkten zu einer gewissen Gleichförmigkeit führt, die bei der Weiterverarbeitung stören kann. Trotzdem weisen sie einige Vorteile auf, allen voran die sehr hohe Genauigkeit bei schnellem Zugriff auf das Gitter.

Die Speicherkomplexität ist abhängig von der Feinheit der Diskretisierung und ist aus $\mathcal{O}(n \cdot m)$. Dabei bezeichnet n die Feinheit in X-Richtung und m die Feinheit in Y-Richtung. Als Datenstruktur für effiziente Zugriffe kann ein Array verwendet werden. Update-Operationen im Gitter zur Manipulation der Zellen benötigen nur konstanten Zeitaufwand.

Gitterkarten eignen sich hervorragend zur Repräsentation unbekannter und karierter Bereiche, weshalb sie hier verwendet werden. Andere Kartenrepräsentationen besitzen diese Eigenschaft nicht oder nur nur eingeschränkt. Durch die Integration der Sensordaten in das Gitter erfolgt eine Informationsverschmelzung mit vorherigen Sensordaten, sodass eine Korrektur einmal integrierter Sensordaten allein mit dem Gitter nicht möglich ist.

Feature-basierte Karten

Eine weitere geometrische Kartenrepräsentation sind feature-basierte Karten:

Definition 3.8 (Feature-basierte Karte) *Eine feature-basierte Karte* m_{feat} *wird repräsentiert durch die Menge* $m_{feat} = \{f \mid f \text{ ist Feature}\}$.

Feature-basierte Karten können zum Beispiel Segmentkarten sein, wobei alle Kartenobjekte als Segmente mittels Weltkoordinaten der Anfangs- und Endpunkte repräsentiert werden.

Features werden normalerweise als Listen abgelegt. Hier können feature-basierte Karten auch in einem 4-d-Baum abgelegt sein, der einen effizienten Zugriff auf diese ermöglicht (siehe auch Kapitel 2). Die Segmente sind dann als 4-Tupel parametrisiert. Diese Datenstruktur benötigt im Aufbauschritt $\mathcal{O}(n \log n)$ Operationen, wobei n die Anzahl der Segmente ist. Die Speicherkomplexität liegt in $\mathcal{O}(n)$. Die Komplexität einer Einfüge- oder Löschoperation ist aus $\mathcal{O}(n^{3/4} + a)$, wobei a die Ausgabekomplexität ist.

3.1.2 Topologisch korrekte und konsistente Karten

Zwei wichtige Begriffe sind für die Kartierung wichtig und prägen die nachfolgende Ausführung, nämlich topologische Korrektheit und Konsistenz. Für das Verständnis

werden diese Begriffe im Kontext geometrischer Karten definiert. Dafür sei ein geometrisches Kartenobjekt definiert als:

Definition 3.9 (Geometrisches Kartenobjekt) *Ein geometrisches Kartenobjekt* o_i *ist ein* n*-Tupel aus* $\mathbb{R}^n$*.*

Nun kann man die Begriffe definieren:

Definition 3.10 (Topologisch korrekte Karte) *Eine Karte* m *heißt topologisch korrekt, wenn die in der Karte enthaltenen Kartenobjekte genau dann miteinander identifizierbar sind, wenn die zugehörigen Kartenobjekte in der realen Welt miteinander identifizierbar sind. Sei* $\tau : \mathbb{R}^n \to \mathbb{R}^m$ *eine Abbildung, welche Kartenobjekte* $o_i \in m$ *auf Objekte der Realität* p_i *abbildet.*
Es gilt also $d_O(o_i, o_j) < \epsilon \Leftrightarrow d_P(\tau(o_i), \tau(o_j)) < \delta$ *für geeignete* d_O, d_P*.*

Definition 3.11 (Konsistente Karte) *Es sei* τ *eine Abbildung* $\mathbb{R}^n \to \mathbb{R}^m$ *der Kartenobjekte* $o_j \in m$ *auf die Objekte der Realität* p_j*. Es sei weiter* $d_O(o_i, o_j)$ *eine Distanzfunktion zwischen Kartenobjekten und* $d_P(\tau(o_i), \tau(o_j))$ *eine Distanzfunktion zwischen realen Objekten.*
Eine Karte m *heißt konsistent, wenn jeder Teilbereich eines reales Objekt durch höchstens ein Kartenobjekt repräsentiert wird, d.h. wenn für alle Kartenobjekte gilt:*

$$|d_O(o_i, o_j) - d_P(\tau(p_i), \tau(p_j))| < \epsilon$$

Dabei modelliert ϵ *die vorhandene Ungenauigkeit der Kartenrepräsentation.*

Vor allem der Begriff der konsistenten Karte ist schwer zu fassen. So ist eine Karte, in welcher eine Wand mehrfach vorkommt, inkonsistent. Dabei gilt jedoch, dass zwei überlappende parallele Segmente mit kleiner Distanz, welche einen Teilbereich eines Objektes repräsentierten, konsistent sind. Erst ab einer bestimmten Distanz ϵ zwischen den Kartenobjekten wird die Karte inkonsistent.

Korollar 3.12 *Jede konsistente Karte ist topologisch korrekt. Die Umkehrung gilt nicht.*

Beweis:

(i) Es seien o_i und o_j Objekte der Karte m. Es gelte die Bedingung der Konsistenz. Für große Distanzen zwischen zwei Objekten o_i und o_j ist nichts zu zeigen, da sich der Begriff der topologischen Korrektheit nur auf kleine Distanzen bezieht. Sei deshalb o.B.d.A. $d_O(o_i, o_j) < \epsilon_1$. Dann folgt aus der Konsistenz dass $d_P(\tau(o_i), \tau(o_j)) < \delta_1$ gilt, sodass

$$|d_O(o_i, o_j) - d_P(p_i, p_j)| < |\epsilon_1 - \delta_1| < \epsilon$$

Analoges folgt für die Annahme $d_P(\tau(o_i), \tau(o_j)) < \delta_1$.

(ii) Wenn nun m topologisch korrekt ist, folgt nur für Objekte mit kleinen Distanzen ($< \epsilon$) die Konsistenz. Wenn es nun o_i, o_j gibt mit $d_O(d_i, d_j) > \epsilon$ kann keine Aussage getroffen werden, sodass insbesondere auch nicht die Konsistenz folgt.

□

Anschaulich bedeutet eine konsistente Karte nichts anderes als dass aufeinanderfolgende Sensordaten mit minimalem Fehler in die Karte integriert werden. Topologisch korrekt bedeutet nichts anderes, als dass alle in der Realität vorkommenden Korrespondenzen zwischen Sensordaten durch das Kartierungsverfahren erkannt werden.

In der Realität ist ein Test auf Konsistenz und topologische Korrektheit unmöglich. Man bedient sich deshalb abgeschwächter Begriffe der lokalen Konsistenz und lokalen topologischen Korrektheit.

Definition 3.13 (Lokal-konsistente Karte) *Eine Karte m heißt lokal-konsistent, falls für eine Teilmenge der Kartenobjekte die Bedingung der Konsistenz gilt.*

Demnach sind aufeinanderfolgende Sensordaten lokal konsistent, wenn ihr Matchingfehler gering ist. Sie sind topologisch korrekt im lokalen Sinne, da die Korrespondenz aufeinanderfolgender Sensordaten eingehalten wird.

3.1.3 Das Problem der feature-basierten Kartierung

Mit den obigen Formalismen kann man nun die anfänglich aufgeworfene Fragestellung der Generierung von 2-D-Modellen formalisieren:

Problem 3.14 (Simultaneous Localization and Mapping) *Zu einem Zeitpunkt k sei ein Netzwerk mit k Knoten $X_0, \ldots, X_{k-1}$ gegeben. Jeder Knoten X_i entspricht einem Tupel aus einer Konfiguration p_i und einer Messung von unsicheren Sensordaten s_i, d.h. $X_i = (p_i, s_i)$.*

Gesucht ist eine Menge von Konfigurationen $\overline{p_i}$, so dass das Netzwerk durch $X_0 = (\overline{p_0}, s_0), \ldots, X_{k-1} = (\overline{p_{k-1}}, s_{k-1})$ eine topologisch-korrekte und konsistente Karte repräsentiert.

In der Literatur ist die Problemstellung als **SLAM-Problematik** bekannt (=**S**imultaneous **L**ocalization **A**nd **M**apping), denn für eine gute Positionsschätzung benötigt man eine Karte der Umgebung und für eine gute Karte benötigt man gute Positionsschätzungen. Das SLAM-Problem kann man in eine Reihe von Teilprobleme und Problemvarianten zerlegen, die unten separat definiert werden.

Die Lösung des SLAM-Problems ist jedoch nur eine Lösung auf dem Weg zu featurebasierten Karten:

Problem 3.15 (Erzeugen feature-basierter Karten) *Zu einem Zeitpunkt k sei ein Netzwerk mit k Knoten $X_0, \ldots, X_{k-1}$ gegeben mit $X_i = (\overline{p_i}, s_i)$, welches eine topologisch-korrekte und konsistente Karte repräsentiert. Gesucht ist eine feature-basierte Kartenrepräsentation m_{feat}, welche effiziente Lokalisationsanfragen ermöglicht.*

Im Idealfall liefert schon das SLAM-Problem eine feature-basierte Kartendarstellung. In der Regel sind allerdings Extraktionsverfahren gesucht, welche Sensordaten effizient in die gewünschte Kartenrepräsentation transformieren. Die Frage nach der Güte der Darstellung der feature-basierten Karte lässt sich dann in experimentellen Studien ermitteln.

Das Gesamtproblem des in diesem Kapitel zu untersuchenden Gegenstandes kann man also folgendermaßen beschreiben:

Problem 3.16 (Feature-basiertes SLAM-Problem) *Zu einem Zeitpunkt k sei ein Netzwerk mit k Knoten $X_0, \ldots, X_{k-1}$ gegeben. Jeder Knoten X_i entspricht einem Tupel einer Konfiguration p_i und einer Messung unsicherer Sensordaten s_i, d.h. $X_i = (p_i, s_i)$. Gesucht ist eine topologisch-korrekte und konsistente feature-basierte Kartenrepräsentation m_{feat}, welche effiziente Lokalisationsanfragen ermöglicht.*

3.1.4 Dekomposition der Problemstellung

In diesem Kapitel sollen die zwei Hauptprobleme separat betrachtet und im folgenden in eine Reihe von Teilprobleme und Problemvarianten zerlegt werden. Beginnend mit der SLAM-Problematik kann man durch Annahmen über die Umgebung den Schwierigkeitsgrad des Problems erhöhen.

Statische und dynamische Umgebungen

Der triviale Fall, dass sich der Roboter in einer statischen Umgebung befindet und dabei an einer Konfiguration steht, soll dabei ausgeschlossen sein. Die beiden anderen Fälle, ein statisches oder dynamisches Umgebungsmodell anzunehmen, verlangen zunächst nach weiteren Definitionen.

Wenn man von Dynamik in einer Umgebung spricht, kann man zwei verschieden Arten von Dynamik ansprechen - dynamische Hindernisse und dynamische Objekte. Diese kann man jedoch klar voneinander trennen:

Definition 3.17 (Dynamisches Hindernis) *Ein Kartenobjekt heißt dynamisches Hindernis o_i^{dh}, wenn es sich mit einer Geschwindigkeit durch den Konfigurationsraum bewegt und in der Lage ist, jede freie Konfiguration zu erreichen.*

Definition 3.18 (Dynamisches Objekt) *Ein Kartenobjekt heißt dynamisches Objekt o_j^{do}, wenn es sich nur innerhalb einer kompakten Teilmenge des Konfigurationsraumes $\mathcal{C}$ bewegen kann.*

Definition 3.19 (Statisches Objekt) *Alle Kartenobjekte, welche kein dynamisches Hindernis oder dynamisches Objekt repräsentieren, werden statische Objekte* o_k^s *genannt.*

Die Begriffe topologisch korrekt und konsistent beziehen sich nur auf statische Kartenobjekte, für dynamische Hindernisse und dynamische Objekte machen sie keinen Sinn.

Mit der Klassifikation der unterschiedlichen Kartenobjekte kann man eine Unterscheidung in statische und dynamische Karten sowie in dynamische und statische Umgebungen vornehmen:

Definition 3.20 (Statische Karte) *Eine statische Karte repräsentiert alle Kartenobjekte als statische Hindernisse.*

Definition 3.21 (Dynamische Karte) *Eine dynamische Karte repräsentiert nur die statischen Kartenobjekte als statische Hindernisse.*

Ein dynamisches Umgebungsmodell repräsentiert eine realistische Einsatzumgebung, ein statisches Kartenmodell setzt nur das Vorhandensein von statischen Hindernissen voraus. Mit dieser Abgrenzung der Umgebungsmodelle vereinfacht sich das SLAM-Problem, man spricht dann von dem statischen SLAM-Problem oder dem dynamischen SLAM-Problem.

Problem 3.22 (Elimination der Dynamik, Dynamisches SLAM-Problem) *Zu einem Zeitpunkt* k *sei ein Netzwerk mit* k *Knoten* $X_0, \ldots, X_{k-1}$ *gegeben. Jeder Knoten* X_i *entspricht einem Tupel aus einer Konfiguration* p_i *und einer Messung von unsicheren Sensordaten* s_i*, d.h.* $X_i = (p_i, s_i)$*. Gesucht ist eine Menge von Konfigurationen* $\overline{p_i}$*, sodass das Netzwerk durch* $X_0 = (\overline{p_0}, s_0), \ldots, X_{k-1} = (\overline{p_{k-1}}, s_{k-1})$ *eine topologisch-korrekte und konsistente* ***dynamische*** *Karte repräsentiert.*

In der Regel kann man eine dynamische Kartenrepräsentation nicht garantieren. Oft werden dynamische Objekte und dynamische Hindernisse als statische Objekte modelliert. Eng verwandt mit der Problemstellung ist auch das Tracking-Problem. Man kann falsche statische Kartenobjekte in dynamischen Umgebungsmodellen reduzieren, indem man dynamische Hindernisse mittels Tracking-Verfahren ausschließt [123, 195]. Die Tracking-Problematik kann man folgendermaßen beschreiben:

Problem 3.23 (Tracking) *Gegeben seien Sensordaten* s_k *und* s_{k+1} *zu zwei aufeinanderfolgenden Zeitpunkten. Die Sensordaten repräsentieren sowohl statische als auch dynamische Kartenobjekte. Das Problem, nur aus den Sensordaten dynamische Hindernisse zu identifizieren und diese über mehrere Zeitschritte in den Sensordaten zu verfolgen, nennt man Tracking-Problem.*

Alle nicht als dynamische Hindernisse identifizierte wechselnde Konfigurationen werden als dynamische Objekte klassifiziert. Dieses Problem wird in dieser Arbeit nicht weiter verfolgt.

Erkennung und Korrektur von Zyklen, Recovery

Die größte Schwierigkeit bei der Lösung der SLAM-Problematik ist noch nicht zur Sprache gekommen, die Identifikation von Zyklen und die Korrektur der an den Zyklen beteiligten Sensordaten. Das Netzwerk von Knoten kann einen oder mehrere Zykel besitzen. Ein Zykel sei folgendermaßen definiert:

Definition 3.24 (Zykel) *Seien* $X_1, \ldots, X_N$ *mit* $X_i = (s_i, p_i)$ *eine Folge von nacheinander aufgenommenen Sensordaten mit Konfigurationen* $(p_1, \ldots, p_n)$ *in der Karte und* $(p_1^r, \ldots, p_n^r)$ *die realen Konfigurationen mit einer Distanzfunktion* d_P*. Eine Teilfolge von Positionen* $\{p_i^r, \ldots p_j^r\}$ *heißt Zykel, falls gilt:* $i < j$ *und* $d_P(p_i^r, p_j^r) < \delta$ *und für alle* p_k^r, p_l^r *mit* $i < k, l < j$ *mit* $k \neq l$ *gilt* $d_P(p_k, p_l) \geq \delta$*.*

Aus der Existenz von Zyklen ergeben sich die beiden nachfolgenden Probleme:

Problem 3.25 (Zykel-Detektion) *Das Problem der Zykel-Detektion besteht darin, alle in der Realität vorkommenden Korrespondenzen zwischen Sensordaten und damit zwischen Kartenobjekten aufzuspüren, welche die topologische Korrektheit der Karte* m *verletzen.*

Problem 3.26 (Zykel-Korrektur) *Das Problem der Zykel-Korrektur besteht darin, alle detektierten Zykel in* m *zu korrigieren, sodass eine topologisch korrekte und konsistente Karte entsteht.*

Diese beiden Teilprobleme machen die SLAM-Problematik so schwierig, dass eine Lösung noch aussteht obwohl es vielversprechende Ansätze in der Literatur gibt, welche in Kapitel 3.2 vorgestellt werden.

Obwohl die meisten erfolgreichen Heuristiken zur Lösung der SLAM-Problematik davon ausgehen, dass Zykel in den Umgebungskarten nur soweit auftauchen, als dass sie ohne zusätzliche Erkennungsverfahren verarbeitet werden können, ist das Recovery-Problem immer präsent:

Problem 3.27 (Recovery) *Das Recovery-Problem besteht darin, das Kartierungsverfahren gegenüber Sensorfehlern und Matchingfehlern abzusichern und es in die Lage zu versetzen jederzeit eine Korrektur der bestehenden Kartenrepräsentation vorzunehmen.*

Zum Recovery-Problem kann man auch die Erkennung und Korrektur von Zyklen zählen [177].

Erzeugung feature-basierter Karten

Nachdem in den obigen Abschnitten die Korrektur von Sensordaten beschrieben wurde, benötigt man weiter ein Verfahren, welches aus den korrigierten Sensordaten ei-

ne feature-basierte Repräsentation rekonstruiert. Das Problem kann man wie folgt beschreiben:

Problem 3.28 (Erzeugung feature-basierter Karten) *Es sei ein Netzwerk mit* k *Knoten* $X_0, \ldots, X_{k-1}$ *einer Umgebung gegeben. Jeder Knoten* X_i *entspricht einem Tupel aus korrigierten Konfigurationen* $\overline{p_i}$ *und einer Messung von unsicheren Sensordaten* s_i*, d.h.* $X_i = (\overline{p_i}, s_i)$*. Jedes* s_i *besteht aus einer Menge von Sensorpunkten. Gesucht ist eine minimale Menge von Segmentfeatures maximaler Länge repräsentiert durch* m_{feat} *so dass diese die gemessene Umgebung approximiert.*

Man kann sich das Problem auch auf eine andere Art definieren, wenn man die Punktzugehörigkeit zu einem Sensor-Datensatz aufgibt:

Problem 3.29 (Rekonstruktion der Umgebung) *Gegeben sei eine Menge* P *von Punkten, welche aus* k *Sensordaten erzeugt wurde. Gesucht ist eine Menge von Segmentfeatures maximaler Länge repräsentiert durch* m_{feat}*, welche diese Punkte geeignet verbindet, sodass daraus die Umgebung rekonstruiert wird.*

Die beiden obigen Problemstellungen betrachten das Problem als global zu lösendes Problem. In der Praxis ist eine inkrementelle Problemstellung und -lösung gefordert, weshalb man für die obigen Probleme jeweils eine inkrementelle Entsprechung formulieren kann:

Problem 3.30 (Inkrementelle Erzeugung feature-basierter Karten) *Es sei ein Netzwerk mit* $k-1$ *Knoten* $X_0, \ldots, X_{k-2}$ *aus einer Umgebung und ein Knoten* X_{k-1} *zu einem Zeitpunkt* k *gegeben. Jeder Knoten* X_i *entspricht einem Tupel aus einer korrigierten Konfiguration* $\overline{p_i}$ *und einer Messung von unsicheren Sensordaten* s_i*, d.h.* $X_i = (\overline{p_i}, s_i)$*. Aus diesem Netzwerk mit* $k-1$ *Knoten resultiert eine Featuremenge* G^{k-1}*. Gesucht ist eine Feature-Menge* G^k *durch Erweiterung des Netzwerks um den Knoten* X_{k-1} *mit der zugehörigen Featuremenge* F*, sodass die Feature-Menge* $G^k = G^{k-1} \cup F$ *eine minimale Menge von Segmenten maximaler Länge durch die Karte* m^k_{feat} *beschreibt, welche die Umgebung approximiert.*

Problem 3.31 (Inkrementelle Rekonstruktion der Umgebung) *Gegeben sei eine Menge* P^{k-1} *von Punkten aus* $k-1$ *Zeitschritten mit einer Feature-Menge* G^{k-1} *und eine neue Menge von Punkten* P *aus den Sensordaten* s_k *zum Zeitschritt* k*. Gesucht ist eine Feature-Menge* G^k *aus der Punktmenge* $P^k = P^{k-1} \cup P$*, sodass daraus das Umgebungsmodell repräsentiert durch die Feature-Karte* m_{feat} *rekonstruiert wird.*

Den Lösungsansätzen für die Probleme 3.28 - 3.31 wird in Abschnitt 3.4 nachgegangen.

3.2 Korrektur zweidimensionaler Sensordaten

In diesem Abschnitt sollen zunächst die wichtigsten Lösungsansätze des Problems 3.14 der Literatur vorgestellt werden und dann ein modifizierter inkrementeller probabili-

stischer Ansatz für eine statische Kartenrepräsentation hergeleitet werden. Anschließend wird eine dynamische Kartenrepräsentation eingeführt und der heuristische Lösungsalgorithmus vorgestellt.

3.2.1 Lösungsansätze in der Literatur

Die Lösung der SLAM-Problematik 3.14 mit all ihren Teilproblemen ist bisher nicht gelungen. Es gibt jedoch Ansätze, welche interessante Ideen zur Lösung der Teilprobleme verfolgen, jedoch keinen Ansatz, welcher alle Problematiken berücksichtigt. Betrachtet man allgemein Lösungsansätze ohne dynamische Implikationen für das SLAM-Problem aber unter Einbeziehung der Zyklen-Problematik (Probleme 3.25 und 3.26), kann man die Ansätze der Literatur zunächst grob in zwei Klassen von Verfahren einteilen:

Geometrische Ansätze
: Geometrische Relationen werden zwischen den Sensordaten aufgebaut und nach einem geometrischen Optimierungskriterium adaptiert.

Probabilistische Ansätze
: Bewertung und Adaption von Positions-Relationen mittels eines probabilistischen Optimalitätskriteriums.

Beide Klassen kann man nochmals in die folgenden Verfahrenstypen einteilen:

Offline-Verfahren
: Daten werden in einer separaten Kartierungsfahrt aufgenommen. Das Zusammenfügen zu einer Karte geschieht hinterher und ist nicht an Echtzeit-Bedingungen geknüpft.

Online-Verfahren oder **Echtzeit-Verfahren**
: Während der Datenaufnahme wird praktisch ohne Zeitverzögerung eine Karte erstellt, welche zu jedem Zeitpunkt aktuell ist. Man bezeichnet diese Verfahren auch als inkrementelle Verfahren. Eine Zyklendetektion und Korrektur ist bei diesen Verfahren bisher nicht möglich.

Recovery-Verfahren
: Verfahren welche versuchen, zusätzlich zur Echtzeit-Bedingung Kartierungsfehler und damit auch Zyklen zu erkennen (Problem 3.27) und die Karte entsprechend zu korrigieren.

Sichere Verfahren
: Verfahren, welche zu jedem Zeitpunkt eine topologisch korrekte und konsistente Karte erstellen. Bisher ist kein Verfahren bekannt.

Die wichtigsten Verfahren sollen anschließend kurz vorgestellt werden.

Geometrische Kartierung

Unter geometrischen Verfahren zur Kartierung versteht man Kartierungsverfahren, welche geometrische Wurzeln haben. So arbeiten die klassischen Scanmatching-Verfahren mit geometrischen Bewertungen (siehe Kapitel 5.3.1). Ebenso zählen dazu Verfahren, welche klassische Optimierungsfunktionen verwenden.

Das bekannteste Offline-Verfahren zur Korrektur der Sensordaten ist das Verfahren von Feng Lu [141–143], welches 1997 präsentiert wurde. Dieses bestimmt mittels der Formulierung des SLAM-Problems 3.14 als geometrisches Optimierungsproblem eine optimale Schätzung der betrachteten Sensordaten. Die konsistente Positionsschätzung liefert eine Lösung des SLAM-Problems und ist in der Lage, Zyklen zu erkennen und zu korrigieren, und liefert augenscheinlich eine topologisch-korrekte und konsistente Karte. Eine Lösung unter Echtzeit-Bedingungen ist nicht möglich.

Als inkrementelles geometrisches Kartierungsverfahren könnte das Verfahren von Cox [52, 91] gelten, ist es doch in der Lage, einen Scan mit einem Kartenmodell zu überdecken. Allerding bleibt dabei das Problem 3.28 bestehen, eine feature-basierte Repräsentation aus integrierten Scan- und Kartendaten zu berechnen.

Eine Variante der konsistenten Positionsschätzung liefert das LRGC (Local Registration and Global Correlation) Verfahren von Gutman und Konolige [92, 94]. Dabei kommen mehrere interessante Ideen zum Tragen. Erste Idee ist die Verwendung lokaler Karten, welche aus den aktuellsten Sensordaten generiert werden. In einer bestehenden Karte wird nun in einem durch das Bewegungsmodell spezifizierten Bereich nach Möglichkeiten gesucht, die lokale Karte zu matchen. Die Bewertung des Matchings erfolgt durch die Verwendung eines Korrelationsoperators. Diese zweite Idee realisiert in jedem Zeitschritt ein Karten-Karten-Matching (siehe Problem 2.3) in einem beschränkten Bereich, allerdings nicht mehr unter Echtzeit-Bedingungen. Das Verfahren ist abhängig vom verwendeten Bewegungsmodell und von der Güte der verwendeten Scanmatching-Verfahren, so dass unerwartete Fehler nicht detektiert werden können. Das Verfahren liefert für kleine zyklische Umgebungen eine topologisch-korrekte und konsistente Karte.

Probabilistische Kartierung

Das probabilistische Äquivalent zur konsistenten Positionsschätzung ist der Expectation-Maximization (EM)-Algorithmus zur Kartierung [40, 211, 217]. Das Verfahren benötigt zu Beginn Sensordaten, die topologisch-korrekt vorliegen. Das Verfahren arbeitet nach dem folgenden Prinzip: Im E-Schritt wird die aktuell beste Karte als statische Karte angenommen, so dass mit dieser die Positionen der Sensordaten bestimmt werden. Im M-Schritt werden die Sensor-Aufnahmepositionen nicht verändert und daraus die beste Karte generiert. Mittels iterierter Anwendung wird eine im günstigsten Fall immer besser werdende Karte erstellt. Durch den Hill-Climbing Cha-

rakter des Verfahrens ist eine beste Lösung nicht immer möglich. Das Verfahren ist ein Offline-Verfahren und liefert topologisch-korrekte Karten.

Das Maximum-Likelihood-Mapping integriert inkrementell Sensordaten in eine aktuelle Karte. Die neuen Sensordaten werden dann an der am besten bewerteten Position in die Karte eingefügt und diese als neue Karte für den darauffolgenden Zeitschritt angenommen. Das Verfahren ist echtzeitfähig und garantiert lediglich lokal-konsistente Karten. Zyklen können weder detektiert noch korrigiert werden. Bekanntester Vertreter für diesen Lösungsansatz ist das Verfahren von Thrun [206, 213].

Dieses liefert zusätzlich eine Recovery-Idee: Zunächst wird eine inkrementelle Karte mittels der Maximum Likelihood Methode aufgebaut. Zusätzlich wird als Zyklendetektor ein Partikelverfahren zur Positionsverfolgung verwendet (siehe Kapitel 5.3.1) um die volle Wahrscheinlichkeitsverteilung über alle Positionen zu berechnen. Inkonsistenzen werden detektiert, sobald die berechneten Standorte des Roboters differieren. Das Verfahren ist allerdings nur so gut wie das zugrundeliegende Bewegungsmodell des Roboters und kann unerwartete Fehler nicht auffangen. Es ist in der Lage, das Problem 3.14 für kleine zyklische Umgebungen in Echtzeit zu lösen und liefert dann eine topologisch-korrekte und konsistente Karte.

3.2.2 Modellierung eines inkrementellen probabilistischen Ansatzes

Die Idee der inkrementellen Kartierung ist es Sensordaten s_k an einer Aufnahmekonfiguration p_k zu akquirieren, diese durch eine lokale Entscheidung in das bestehende Kartenmodell m_{k-1} einzufügen und dieses dabei zu erweitern. Beim Einfügen muss die Aufnahmeposition mit den vorhandenen Daten korrigiert werden, sodass anschließend das Kartenmodell m_k mit Hilfe der Karte m_{k-1} und den Sensordaten s_k an der korrigierten Position $\overline{p_k}$ erstellt werden kann.

Dieser Ansatz hat seine Idee von Ansätzen der Positionskorrektur (siehe dazu auch Kapitel 5.3.1), bei denen aufeinanderfolgende Sensordaten miteinander verglichen und so der Positionsfehler korrigiert wird, und ist in der Literatur als Maximum-Likelihood-Methode bekannt.

Vorteil eines inkrementellen Algorithmus ist die Schnelligkeit mit welcher Sensordaten in die Umgebungskarten integriert werden können. Hier soll als eine Variante des Algorithmus von Thrun [206] zur inkrementellen Kartierung hergeleitet werden. Dieser Ansatz verwendet zunächst ein statisches Sensormodell unter der Annahme statischer Umgebungen und wird anschließend um ein dynamisches Sensormodell unter der Annahme dynamischer Umgebungen erweitert.

Statische Herleitung einer Berechnungsformel

Der Bayes-Filter (siehe Anhang A) ist das Herz des Algorithmus. Er löst das Problem, den Zustand eines zeit-varianten dynamischen Systems, d.h. eines mobilen Roboters, vorherzusagen.

Es bezeichne x_t den Zustand des kontinuierlichen zeit-varianten dynamischen Systems zum Zeitpunkt t. Sensordaten zum Zeitpunkt t werden mit s_t bezeichnet. Es bezeichne weiterhin a_t die Änderung des dynamischen Systems im Zeitintervall $]t, t+1]$. Die Menge aller Daten (Messwerte und Zustandsänderung) wird mit $d_t = \{s_0, a_0, \ldots, s_t, a_t\}$ bezeichnet.

Annahme: Es gelte die Markov-Annahme, dass die Wahrscheinlichkeit $P(x_t \mid d_t)$ eine hinreichende Beschreibung vergangener Daten ist.

Es gelte dann:

$$P(x_t \mid d_t) = P(x_t \mid s_t, a_{t-1}, d_{t-1}) \tag{3.3}$$

Unter Anwendung der Bayes'schen Regel erhält man

$$P(x_t \mid d_t) = \eta \cdot P(s_t \mid x_t, a_{t-1}, d_{t-1}) \cdot P(x_t \mid a_{t-1}, d_{t-1}) \tag{3.4}$$

Dabei ist η ein aus der Bayes'schen Regel resultierender Normalisierungsfaktor. Mit Hilfe der Markov-Annahme erhält man

$$P(x_t \mid d_t) = \eta \cdot P(s_t \mid x_t) \cdot P(x_t \mid a_{t-1}, d_{t-1}) \tag{3.5}$$

Den Term $P(x_t \mid a_{t-1}, d_{t-1})$ kann man durch einen Integrationsschritt ersetzen und erhält

$$P(x_t \mid d_t) = \eta \cdot P(s_t \mid x_t) \cdot \int P(x_t \mid a_{t-1}, d_{t-1}, x_{t-1}) \cdot P(x_t - 1) \mid a_t - 1, d_{t-1}) dx_{t-1} \tag{3.6}$$

welchen man zusammenfassen kann zu

$$P(x_t \mid d_t) = \eta \cdot P(s_t \mid x_t) \cdot \int P(x_t \mid a_{t-1} x_{t-1}) \cdot P(x_t - 1) \mid d_{t-1}) dx_{t-1} \tag{3.7}$$

Anwendung auf das statische SLAM-Problem

Bezeichne nun einen Zustand x durch eine Karte m und den Standort p des Roboters.

$$x = (m, p) \tag{3.8}$$

Dabei bezeichnet der Standort des Roboters auch den Punkt, an welchem neue Sensordaten akquiriert werden. Dann kann man (analog zu oben) die folgende Formel betrachten:

$$\begin{aligned} P(p_t, m_t \mid d_t) \quad = \quad & P(s_t \mid p_t, m_t) \cdot \\ & \int\int P(p_t, m_t \mid a_{t-1}, p_{t-1}, m_{t-1}) \cdot P(p_{t-1}, m_{t-1} \mid d_{t-1}) dp_{t-1} dm_{t-1} \end{aligned} \tag{3.9}$$

Mit der obigen Annahme gilt nun, $m = m_{t-1}$, da sich unter der Annahme, dass die Welt statisch ist, nichts ändert. Es folgt:

$$P(p_t, m \mid d_t) = P(s_t \mid p_t, m) \cdot \int P(p_t \mid a_{t-1}, p_{t-1}, m) \cdot P(p_{t-1}, m \mid d_{t-1}) dp_{t-1} \quad (3.10)$$

Annahme: Die Bewegung des Roboters hat keinen Einfluss auf die Karte.

Somit bekommt man die rekursive Gleichung

$$P(p_t, m \mid d_t) = P(s_t \mid p_t, m) \cdot \int P(p_t \mid a_{t-1}, p_{t-1}) \cdot P(p_{t-1}, m \mid d_{t-1}) dp_{t-1} \quad (3.11)$$

Diese Gleichung taucht in der Robotik häufiger auf [69] und beschreibt den inkrementellen Positionsupdate. Der Term $P(s_t \mid p_t, m)$ beschreibt das Sensormodell, der Term $P(p_t \mid a_{t-1}, p_{t-1})$ das Bewegungsmodell des Roboters, sodass man mit einer initialen Wahrscheinlichkeitsverteilung $p(s_0, m)$ rekursiv die Formel berechnen kann.

3.2.3 Statische Modellierung der Umgebung

In diesem Abschnitt soll zunächst die statische Repräsentation des Konfigurationsraumes durch eine Gitterkarte repräsentiert werden. Anschließend werden Bewegungsmodell und Sensormodell vorgestellt, welche an das Verfahren von Thrun [206, 213] angelehnt sind. Durch die diskretisierte Darstellung des Umgebungsmodells und die tatsächliche iterative Berechnung der Umgebungsmodelle zu diskreten Zeitpunkten werden die kontinuierlichen Zeitpunkte t im folgenden durch eine diskretisierte Darstellung k ersetzt.

Modellierung einer Gitterkarte

Die hier verwendete Gitterkarte entspricht der aus Definition 3.7. Jedes Feld $g(i, j)$ innerhalb der Gitterkarte wird durch eine Abbildung $E : \mathbb{N} \times \mathbb{N} \rightarrow \{\text{Unbekannt, Belegt, Frei}\}$ mit genau einer der folgenden Eigenschaften identifiziert:

Unbekannt: Über das Feld $g(i, j)$ liegen keine Informationen vor. Zu Beginn wird das gesamte Gitter mit diesem Zustand belegt.

Belegt: In dem Feld $g(i, j)$ befindet sich mindestens ein statisches Objekt.

Frei: Das Feld $g(i, j)$ wurde bereits exploriert und festgestellt, dass sich dort kein statisches Objekt befindet.

Diese Repräsentation ist eine stark vereinfachte aber effiziente Repräsentation des von Elfes vorgestellten Ansatzes des Occupancy-Grids [64], jedoch dadurch in der Lage, sehr große Karten zu repräsentieren. Die Nachteile der Repräsentation liegen darin, dass einmal als statisch deklarierte Objekte, etwa dynamische Objekte, in der Karte verzeichnet bleiben.

Bewegungsmodell

Das Bewegungsmodell beschreibt die Bewegung des Roboters, d.h. es versucht die bedingte Wahrscheinlichkeitsverteilung $P(p_k \mid a_{k-1}, p_{k-1})$ zu modellieren. Diese Wahrscheinlichkeitsverteilung gibt an, wie groß die Wahrscheinlichkeit ist, dass sich der Roboter an der Position p_k befindet, wenn er einen Zeitschritt vorher von der Position p_{k-1} aus die Aktion a_{k-1} ausgeführt hat.

Es wird angenommen, dass sich der mobile Roboter mit Translations- und Rotationsfehler in seiner Umgebung fortbewegt. Für einen kleinen Bewegungsschritt kann dieser Fehler durch drei unabhängige normalverteilte Zufallsvariablen modelliert werden. In der vorliegenden Implementierung wird die Wahrscheinlichkeitsverteilung im Unterschied zur Implementierung von Thrun über die Konfigurationen des Roboters durch eine Partikelmenge approximiert.

Algorithmus 3.1: Berechnung des Bewegungsmodells

Aufruf: **motion-model()**
Input: Position p_{k-1}, Aktion a_{k-1}, Partikelanzahl n
Output: Partikelmenge $Par_n = \{par_1, \ldots, par_n\}$

```
1 Par_0 = ∅.
2 Berechnet Position ohne Fehler p = p_{k-1} + a_{k-1}
3 for { i = 1 bis i ≤ n} do
4    par_i = p + ( N(0, σ_x), N(0, σ_y), N(0, σ_θ) )
5    Par_n = Par_{n-1} ∪ par_i
6    i++
7 end for
8 return Par_n
```

Definition 3.32 (Partikel) *Ein Partikel par_i wird durch ein Zwei-Tupel (p, w) repräsentiert. Dabei beschreibt p_i eine Konfiguration des Konfigurationsraums $\mathcal{C}$ und w_i das Gewicht des Partikels aus $[0, 1]$.*

Damit die Approximation der Wahrscheinlichkeitsverteilung durch eine Menge von n Partikeln erfolgen kann, müssen Translations- und Rotationsfehler einer Aktion $a_{k-1} = (\Delta d, \Delta\theta)$ beschrieben werden. Dies geschieht hier mittels dreier unabhängiger Normalverteilungen, deren Standardabweichungen durch relative und absolute Fehlerwerte berechnet werden. Seien e_x^{rel}, e_y^{rel} und e_θ^{rel} die relativen und e_x^{abs}, e_y^{abs} und e_θ^{abs} die absoluten Fehler für die Translation und Rotation. Dann ergeben sich die Standardabweichungen zu

$$\sigma_x = \Delta d \cdot e_x^{rel} + e_x^{abs} \tag{3.12}$$

$$\sigma_y = \Delta d \cdot e_y^{rel} + e_y^{abs} \tag{3.13}$$

$$\sigma_\theta = \Delta\theta \cdot e_\theta^{rel} + e_\theta^{abs} \tag{3.14}$$

Korollar 3.33 (Aufwand der Partikelberechnung) *Sei n die Anzahl der zu generierenden Partikel, dann besitzt der Algorithmus 3.1 die Zeitkomplexität $\mathcal{O}(n)$.*

Beweis: In Algorithmus 3.1 werden n Partikel erzeugt. Die Erzeugung eines Partikel erfolgt in konstanter Zeit, indem für die neu berechnete Position p jeweils ein Partikel mit einer neuen zufälligen Konfiguration erzeugt wird, welches innerhalb eines möglichen Fehlerbereiches um die Konfiguration p liegt. Somit ergibt sich der obige Aufwand. □

Statisches Sensormodell

Das Sensormodell, beschrieben durch den Term $P(s_k \mid p_k, m_{k-1})$, modelliert die Sensorik des Roboters. Es wird eine Wahrscheinlichkeitsverteilung spezifiziert, die angibt, wie groß die Wahrscheinlichkeit ist den Scan s_k aufzunehmen, unter den Bedingungen, dass p_k die Aufnahmeposition und m_{k-1} die aktuelle Karte ist.

Als Kartenrepräsentation wird auf die Repräsentation als Gitterkarte zurückgegriffen. Sei s_k ein Scan mit Aufnahmeposition p_k. Dann wird für jeden Messwert $s_k^{[i]}$ des Scans bestimmt, welchen Typ der Gitterpunkt repräsentiert in den der aktuelle Sensorwert hineinfällt.

Dabei liegt folgende Idee zugrunde: Ein Sensorpunkt $s_k^{[i]}$ überstreicht mit seiner Aufnahme auch einen Bereich der Karte, der bis zum Aufnahmepunkt frei war und damit auch bereits besucht. Dieser Bereich sollte als frei markiert werden und repräsentiert den Sichtbarkeitsbereich des Roboters. Mit abnehmender Distanz vom Scanmittelpunkt wird die Wahrscheinlichkeit geringer, dass dort ein Scanpunkt liegen kann. Falls $s_k^{[i]}$ in einen freien Gitterpunkt fällt, wird die Distanz d zum nächsten Hindernis in der Karte m_{k-1} berechnet und entsprechend einer Normalverteilung gewichtet.

Algorithmus 3.2: Berechnung des Sensormodells

Aufruf: **sensor-model()**
Input: Scan s_k, Position p_k, Karte m_k
Output: $P(s_k \mid p_k, m_{k-1})$
1 Initialisiere $w = 1$.
2 **for all** {Messwerte $s_k^{[i]}$ des Scans s_k} **do**
3 Verschiebe $s_k^{[i]}$ um die Aufnahmeposition p_k.
4 Berechne die Distanz zum nächsten Hindernis in m_{k-1} mittels Spiralsuche.
5 $w = w \cdot P(s_k^{[i]} \mid p_k, m_{k-1})$
6 **end for**
7 Setze $P(s_k \mid p_k, m_{k-1}) = w$.

Fällt ein Messpunkt in einen als belegt gekennzeichneten Bereich, soll dieser bestärkt werden. Fällt ein Messpunkt in einen unbekannten Bereich, soll dieser ebenfalls stark

Algorithmus 3.3: Spiralsuche in einer Gitterkarte

Aufruf: **spiral-search()**
Input: Punkt p in Weltkoordinaten
Output: Minimale Distanz d zu einem Hindernis

```
1  Identifiziere den Startpunkt der Spiralsuche (i, j) = dis(p) im Gitter.
2  d = 0; count= 1; sign= 1
3  while (d == 0) do
4     //Suche in X−Richtung
5     for k=0; k<count; k++ do
6        i+= sign;
7        if E(i, j) = Belegt then
8           p_2 = dis^{-1}(i, j)
9           return | p − p_2 |
10       end if
11    end for
12    //Suche in Y−Richtung
13    for l=0; l<count; l++ do
14       j+= sign;
15       if E(i, j)= Belegt then
16          p_2 = dis^{-1}(i, j)
17          return | p − p_2 |
18       end if
19    end for
20    Ändere die Suchrichtung, sign = - sign;
21    Erhöhe den Suchradius, count++;
22 end while
```

bewertet werden. Hier werden insbesondere die schon belegten Bereiche etwas bevorzugt, was sich wesentlich auf den Erfolg des Algorithmus auswirkt.

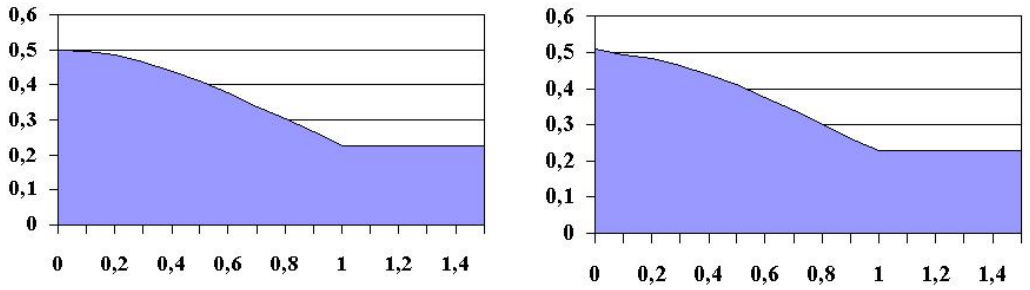
Die folgende Formel spezifiziert die Wahrscheinlichkeit eines Sensorwertes $s_k^{[i]}$ unter der Bedingung, dass die Aufnahmekonfiguration des Scans p_k und die aktuelle Karte m_{k-1} ist.

$$P(k_t^{[i]} \mid p_k, m_{k-1}) = \begin{cases} N(0,\, 0.8,\, 0) & , \; s_k^{[i]} \text{fällt in } g(i,j) \text{ mit } E(i,j) = \textit{unbekannt} \\ N(0,\, 0.8,\, 0) + 0.01 & , \; s_k^{[i]} \text{fällt in } g(i,j) \text{ mit } E(i,j) = \textit{belegt} \\ N(0,\, 0.8,\, d) & , \; s_k^{[i]} \text{fällt in } g(i,j) \text{ mit } E(i,j) = \textit{frei und } d < 1 \\ N(0,\, 0.8,\, 1) & , \; \text{sonst.} \end{cases} \tag{3.15}$$

Dabei ist $N : \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R} \rightarrow [0, 1]$ mit $N(\mu, \sigma, d)$ die Dichtefunktion an der Stelle d einer normalverteilten Zufallsvariable mit Erwartungswert μ und Standardabweichung σ.

$$N(\mu, \sigma, d) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{d-\mu}{\sigma}\right)^2} \tag{3.16}$$

Abbildung 17: Bewertungsfunktion eines Messpunktes



Bewertungsfunktion für einen Messpunkt abhängig von der Distanz d zu einem Hindernis. Links ist diese für freien und unbekannten Bereich, rechts für freien und belegten Bereich dargestellt.

Die Abbildung 17 visualisiert die Wahrscheinlichkeitsfunktion $P(s_k^{[i]} \mid p_k, m_{k-1})$ für verschiedene Distanzen d. Als Erwartungswert wurde 0.8 und Standardabweichung 0.0 gewählt. Die Wahrscheinlichkeitsbewertung eines kompletten Scans ergibt sich aus dem Produkt aller Einzelwahrscheinlichkeiten jedes einzelnen Messwertes des Scans.

$$P(s_k \mid p_k, m_{k-1}) = \prod_{\forall i} P(s_k^{[i]} \mid p_k, m_{k-1}) \tag{3.17}$$

Die Bewertungsfunktion ist gegenüber des Originalansatzes modifiziert worden, um belegte Bereiche stärker zu gewichten.

Korollar 3.34 (Aufwand der Sensormodell-Berechnung) *Die Zeitkomplexität zur Berechnung des Sensormodells nach Algorithmus 3.2 beträgt $\mathcal{O}(n \cdot m^2)$, wobei n die Anzahl der Messpunkte des Scans und m die maximale Anzahl der Gridpunkte einer Dimension repräsentiert.*

Beweis: Die Entscheidung, ob sich ein Punkt $p = (x, y)$ mit $x, y \in \mathbb{R}$ in der freien Fläche befindet, d.h. dass dort kein Objekt erkannt wurde, kann in der Gitterkartenrepräsentation mit dem Aufwand $\mathcal{O}(1)$ gefällt werden. Dazu wird der Punkt mittels der Diskretisierungsfunktion $dis(p) = (i, j)$ in das Gitter abgebildet.

Durch eine Spiralsuche (siehe Algorithmus 3.3) wird das nächste Feld ermittelt, dessen Eintrag als *Belegt* gekennzeichnet ist, und die Distanz des übergebenden Punktes zu diesem Gitterpunkt ermittelt. Das Feld $g(i, j)$ ist der Startpunkt der Spiralsuche. Im schlechtesten Fall muss damit das gesamte Gitter abgesucht werden, was einen Aufwand von $\mathcal{O}(m^2)$ benötigt, wobei m die maximale Anzahl der Gitterpunkte einer Dimension ist.

Sei $g(i', j')$ das gefundene belegte Feld, so kann der Abstand in Weltkoordinaten zwischen $g(i, j)$ und $g(i', j')$ ermittelt werden durch $d = | \, dis^{-1}(i, j) - dis^{-1}(i', j') \, |$. Dieses kostet nur konstanten Zeitaufwand.

Insgesamt ergibt sich die Zeitkomplexität von $\mathcal{O}(n \cdot m^2)$ aufgrund von n Messpunkten. □

In der Realität dauert eine Berechnung des Sensormodells mit einem 180-Grad-Scan bei einer Winkelauflösung von 0.5 Grad, d.h. 360 Messwerten, ca. eine Millisekunde auf einem Pentium III mit 900 MHz.

Integration von Sensordaten

Ist die korrigierte Konfiguration $\overline{p_k}$ der Sensordaten s_k zum Zeitpunkt k berechnet, müssen die Sensordaten in die Gitterkarte integriert werden.

Dafür kann eine lokale Gitterkarte g_{local} verwendet werden. Zu Beginn sei diese lokale Gridkarte ebenso wie die globale Gitterkarte g_{grid} für jeden Gitterpunkt mit der Eigenschaft *Unbekannt* initialisiert. Die Dimension der Gitterkarte ist dabei abhängig von der Diskretisierung der maximalen Sensorreichweite. In die lokale Gridkarte werden zunächst die Messpunkte $s_k^{[i]}$ eingetragen und markiert.

Desweiteren muss für je zwei benachbarte Messpunkte der kürzeste Weg in der Gitterkarte berechnet werden. Alle Gitterpunkte des kürzesten Weges erhalten eine Markierung. Das Ziel ist dabei, einen umschlossenen Bereich der Gitterkarte zu erhalten, welcher durch die Markierungen den aktuellen sichtbaren Bereich der Sensordaten repräsentiert. Das Innere der Markierung wird nun mit der Eigenschaft *Frei* deklariert.

Anschließend wird der Koordinatenursprung der lokalen Gitterkarte auf die Konfiguration $\overline{p_k}$ der globalen Gridkarte transformiert. Schließlich wird die lokale Gitterkarte in die globale Gitterkarte integriert. Algorithmisch geschieht das dadurch, dass für jeden als *Frei* oder *Belegt* markierten Gitterpunkt aus der lokalen Gridkarte die Position in Weltkoordinaten berechnet wird und aus dieser die Gridposition des globalen Gitters. Es gilt also

$$(i, j) = dis_{grid}(dis_{local}^{-1}(s, t)) \tag{3.18}$$

und

$$(s, t) = dis_{local}(dis_{grid}^{-1}(i, j)) \tag{3.19}$$

Hat man zwei korrespondierende Gitterpunkte identifiziert, muss man eine Fallunterscheidung treffen, um den Zustand des Gitterpunktes im globalen Gitter zu setzen:

1. Fall: $E_{local}(s, t) =$ *Frei* und $E_{grid}(i, j) =$ *Belegt*
   Dann ändert sich die Belegung von $E_{grid}(i, j)$ nicht, da einmal belegte Punkte belegt bleiben.

2. Fall: $E_{local}(s,t) = \textit{Belegt}$ und $E_{grid}(i,j) = \textit{Frei}$
   Dann ändert sich die Eigenschaft von $E_{grid}(i,j)$ auf *Belegt*.
3. Fall: $E_{local}(s,t) = \textit{Belegt}$ und $E_{grid}(i,j) = \textit{Unbekannt}$
   Dann ändert sich die Eigenschaft von $E_{grid}(i,j)$ auf *Belegt*, da noch keine Belegung vorhanden war.
4. Fall: $E_{local}(s,t) = \textit{Frei}$ und $E_{grid}(i,j) = \textit{Unbekannt}$
   Dann ändert sich die Eigenschaft von $E_{grid}(i,j)$ auf *Frei*, da noch keine Belegung vorhanden war.
5. Fall: Ansonsten ergeben sich keine Zustandsänderungen der Eigenschaften.

Algorithmus 3.4: Integration von Sensordaten in Gitterkarte

Aufruf: **add-scan()**
Input: Scan s_k, Position $\overline{p_k}$, Karte m_{k-1} durch g_{grid}
Output: Neue Karte m_k

```
1  Erzeuge neues Gitter g_local mit Größe m × m.
2  Berechne dis_local(s_0, k_0) für s_k^[0] und setze E_local(s_0, k_0) = Belegt.
3  for all {s_k^[i] aus s_k, i > 0} do
4      Berechne dis_local(s_i, k_i) für s_k^[i] und setze E_local(s_i, k_i) = Belegt.
5      Suche Weg von dis_local(s_i, k_i) nach dis_local(s_{i-1}, k_{i-1}) und markiere den gefundenen Weg.
6  end for
7  Suche Startpunkt innerhalb des lokalen Grids.
8  Transformiere rekursiv bis zu den erzeugten Markierungsgrenzen alle unbesuchten Gitterpunkte als freie Gitterpunkte.
9  for all {Markierte Gitterpunkte der lokalen Karte g_local} do
10     Transformiere g_local nach g_grid.
11 end for
```

Korollar 3.35 (Komplexität des Einfügens) *Das Einfügen eines Scans s_k in eine Gitterkarte m_{k-1} nach Algorithmus 3.4 benötigt eine Zeitkomplexität von $\mathcal{O}(n \cdot m^2)$. Dabei bezeichnet m die Anzahl der Gridpunkte, die benötigt werden um die maximale Sensorreichweite mittels eines lokalen Gitters zu diskretisieren. Die Variable n gibt die Anzahl der Meßpunkte des Scans an. Die Speicherkomplexität beträgt $\mathcal{O}(m^2)$.*

Beweis: Für jeden Meßpunkt muss untersucht werden, ob der vorangegangene Meßpunkt benachbart ist oder nicht. Dann erfolgt eine kürzeste Wege-Suche über das Grid, wobei im schlechtesten Fall alle Gitterpunkte untersucht werden. Für n Meßpunkte und eine Gittergröße von $m \times m$ ergibt sich eine Zeitkomplexität von $\mathcal{O}(n \cdot m^2)$. Die Suche nach einem Startpunkt innerhalb des markierten Bereiches benötigt im schlechtesten Fall ebenfalls $\mathcal{O}(m^2)$ Zeitschritte, ebenso die Markierung des sichtbaren Gitterbereichs. Der anschließende Einfüge-Schritt benötigt ebenfalls $\mathcal{O}(m^2)$ Zeitschritte, so dass

sich insgesamt die angegebenen Zeitkomplexität ergibt. Speicherplatz wird nur für das zusätzliche lokale Gitter benötigt, so dass sich dieser zu $\mathcal{O}(m^2)$ ergibt. □

3.2.4 Dynamische Modellierung der Umgebung

Für die Betrachtung des dynamischen SLAM-Problems 3.22 setzt man eine dynamische Umgebung voraus. Dabei können sich Zustände dynamischer Objekte und dynamischer Hindernisse verändern, so dass die Umgebung zu zwei unterschiedlichen Zeitpunkten nicht identisch ist.

Die Modellierung eines Lösungsansatzes des dynamischen SLAM-Problems geschieht normalerweise über die Betrachtung aller möglichen Kartenrepräsentationen [214].

In diesem Ansatz soll ein anderer Weg gegangen werden. Die Idee dabei ist es, neben einem freien und einem belegten Zustand der Gitterkarte auch noch einen dynamischen Zustand zuzulassen. Erlaubt man dann zunächst nur dynamische Hindernisse in der Karte, ändert sich die Karte zu unterschiedlichen Zeitpunkten nicht.

Nimmt man weiter an, dass dynamische Objekte nicht sehr häufig vorkommen und sich die Umgebungskarte nur minimal verändert, kann man immer noch eine statische Umgebung voraussetzen. Das diese Annahmen in der Praxis sehr erfolgreich sind, zeigen die Resultate in Kapitel 3.3.

Dynamische Gitterkarte

Die dynamische Gitterkarte soll zusätzlich zur statischen Repräsentation einer Umgebung auch dynamische Hindernisse repräsentieren. Dynamische Objekte dagegen sollen mit Hilfe der Gitterkarte eliminiert werden. Diese Repräsentation ist eine echte Erweiterung der Gitterkartenrepräsentation von Elfes [64], da zusätzlich dynamische Zustände für einen Gitterpunkt eingeführt werden.

Ein Gridpunkt $g(i,j)$ einer dynamischen Gitterkarte wird durch zwei Zustände repräsentiert:

Watched: Gibt an, wie oft neue Sensorinformationen in den betrachteten Gitterpunkt integriert wurden. Zu Beginn ist watched(i,j) mit 0 belegt.

Occupied: Gibt an, wie oft während der Kartierung in dem betrachteten Gitterpunkt ein Hindernis detektiert wurde. Zu Beginn ist occupied(i,j) mit 0 belegt.

Im Gegensatz zur statischen Repräsentation wird der Status eines Gitterpunktes aus diesen beiden Werten berechnet und entsprechend interpretiert. Die Berechnung der Belegtheitswahrscheinlichkeit erfolgt durch

$$P(i,j) = \begin{cases} 0 & \text{falls watched}(i,j) = 0 \\ \frac{\text{occupied}(i,j)}{\text{watched}(i,j)} & \text{sonst} \end{cases} \tag{3.20}$$

Die Eigenschaft eines Gitterpunktes $g(i,j)$ lässt sich nun durch die Abbildung $E : \mathbb{N} \times \mathbb{N} \rightarrow \{\text{Unbekannt, Frei, Dynamisch, Belegt}\}$ folgendermaßen definieren:

$$E(i,j) = \begin{cases} \text{Unbekannt} & \text{falls } P(i,j) = 0 \\ \text{Frei} & \text{falls } P(i,j) \in]0, P_{free}[\\ \text{Dynamisch} & \text{falls } P(i,j) \in [P_{free}, P_{occ}[\\ \text{Belegt} & \text{falls } P(i,j) \geq P_{occ} \end{cases} \tag{3.21}$$

Dabei sind P_{free} und P_{occ} Schwellenwerte aus $[0,1]$ mit der Bedingung $P_{free} \leq P_{occ}$. Sie determinieren die Grenze, ab wann ein Gitterpunkt als frei beziehungsweise als belegt angesehen wird.

Dynamisches Sensormodell

Bei der dynamischen Modellierung muss das Bewegungsmodell, d.h. die Berechnung von $P(p_k \mid a_{k-1}, p_{k-1})$ nicht adaptiert werden.

Die Berechnung des Sensormodells $P(s_k \mid p_k, m_{k-1})$ bedarf jedoch einer Korrektur, da die dynamische Gitterkarte für jeden Gitterpunkt eine dynamische Eigenschaft bereitstellt. Dieser dynamische Zustand kann also ein freier oder ein belegter Zustand sein. Aus diesem Grunde wird er etwas schlechter bewertet als ein Messpunkt welcher in einen belegten Gitterpunkt fällt.

Die folgende Formel spezifiziert die Wahrscheinlichkeit eines Sensorwertes $s_k^{[i]}$ unter der Bedingung, dass die Aufnahmekonfiguration des Scans p_k und die aktuelle Karte m_{k-1} ist.

$$P(s_k^{[i]} \mid p_k, m_{k-1}) = \begin{cases} N(\mu,\, \sigma,\, 0) & , \; s_k^{[i]} \text{fällt in g(i,j) mit E(i,j) = } \textit{Unbekannt} \\ N(\mu,\, \sigma,\, 0) + 0.005 & , \; s_k^{[i]} \text{fällt in g(i,j) mit E(i,j) = } \textit{Dynamisch} \\ N(\mu,\, \sigma,\, 0) + 0.01 & , \; s_k^{[i]} \text{fällt in g(i,j) mit E(i,j) = } \textit{Belegt} \\ N(\mu,\, \sigma,\, d) & , \; s_k^{[i]} \text{fällt in g(i,j) mit E(i,j) =} \textit{Frei} \text{ und } d < 1 \\ N(\mu,\, \sigma,\, 1) & , \; \text{sonst.} \end{cases} \tag{3.22}$$

Dabei gilt wie beim statischen Sensormodell $\mu = 0$ und $\sigma = 0.8$.

Korollar 3.36 (Aufwand der Sensormodell-Berechnung) *Die Zeitkomplexität zur Berechnung des Sensormodells nach Algorithmus 3.2 beträgt $\mathcal{O}(n \cdot m^2)$, wobei n die Anzahl der Datenpunkte des Scans und m die maximale Anzahl der Gridpunkte einer Dimension repräsentiert.*

Beweis: Die Entscheidung, ob sich ein Punkt $p = (x, y)$ mit $x, y \in \mathbb{R}$ in der freien Fläche, d.h. dass dort kein Objekt erkannt wurde, befindet, kann in der Gitterkartenrepräsentation mit dem Aufwand $\mathcal{O}(1)$ gefällt werden. Dazu wird der Punkt mittels der

Diskretisierungsfunktion $dis(p) = (i, j)$ in das Gitter abgebildet.

Durch eine Spiralsuche (siehe Algorithmus 3.3) wird das nächste Feld ermittelt, dessen Eintrag *Belegt* oder **Dynamisch** ist und die Distanz des übergebenden Punktes zu diesem Gitterpunkt ermittelt. Das Feld $g(i, j)$ ist der Startpunkt der Spiralsuche. Im schlechtesten Fall muss damit das gesamte Gitter abgesucht werden, was einen Aufwand von $\mathcal{O}(m^2)$ benötigt mit der maximalen Anzahl m der Gitterpunkte einer Dimension.

Sei $g(i', j')$ das gefundene belegte Feld, so kann der Abstand in Weltkoordinaten zwischen $g(i, j)$ und $g(i', j')$ ermittelt werden durch $d = \mid dis^{-1}(i, j) - dis^{-1}(i', j') \mid$. Dieses kostet nur konstanten Zeitaufwand.

Insgesamt ergibt sich die Zeitkomplexität von $\mathcal{O}(n \cdot m^2)$. □

Integration neuer Sensordaten

Die Integration von Sensordaten s_k in die dynamische Gitterkarte muss noch genauer beleuchtet werden. Algorithmisch bildet man dabei zwei verschiedene Gitterkarten aufeinander ab. Für die Sensordaten wird ein lokales Gitter g_{local} erzeugt, welches neben den Sensorpunkten auch die freien Konfigurationen repräsentieren kann. Das lokale Gitter des Scans hat allerdings nicht die Möglichkeit dynamische Konfigurationen zu entdecken.

Sollen Sensordaten in die dynamische Gitterkarte g_{dyn} eingefügt werden, müssen die beiden Gitter an der einzufügenden Konfiguration $\overline{p_k}$ übereinandergelegt werden. Dies geschieht wie oben nach Algorithmus 3.4. Ein neuer Sensorpunkt wird entsprechend der folgenden Fallunterscheidung in die Gitterkarte eingefügt:

1. Fall: $E_{local}(s, t) = \textit{Unbekannt}$
   In der globalen Karte g_{dyn} verändert sich nichts.

2. Fall: $E_{local}(s, t) = \textit{Frei}$
   Die Variable watched(i, j) der Gitterkarte g_{dyn} wird um 1 erhöht, da der Gridpunkt mit einem gültigen Wert belegt war. Die Variable occupied(i, j) wird nicht verändert. Die resultierende Belegtheitswahrscheinlichkeit $P(i, j)$ verringert sich dadurch.

3. Fall: $E_{local}(s, t) = \textit{Belegt}$
   Die Variable watched(i, j) der Gitterkarte g_{dyn} wird um 1 erhöht, da der Gridpunkt mit einem gültigen Wert belegt war. Die Variable occupied(i, j) wird ebenfalls um eins erhöht, da ein Hindernis detektiert wurde. Die resultierende Belegtheitswahrscheinlichkeit $P(i, j)$ erhöht sich dadurch.

Durch diesen Ansatz zur Integration von Sensordaten in eine dynamische Gitterkarte ist es möglich, dass sich die Eigenschaft eines Gitterpunktes während der Kartierung verändern kann.

Modellierung des befahrenen Pfades

Zusätzlich zur Information der Sensordaten besitzt man zu jedem Zeitpunkt k die Information, dass sich der Roboter von Konfiguration $\overline{p_{k-1}}$ zur Konfiguration $\overline{p_l}$ bewegt hat. Auf diesem Weg können also garantiert keine Hindernisse vorhanden sein, die überdeckten Konfigurationen repräsentieren einwandfrei freie Konfigurationen.

Diese Information kann man nutzen, indem man abhängig von der Ausdehnung des Roboters eine zweidimensionale Region in Weltkoordinaten modelliert und dann alle Gitterpunkte identifiziert, welche eine Position im Inneren der befahrenen Region repräsentieren. Um ein Feld $g(i,j)$ als frei zu deklarieren, werden die Variablen $\text{occupied}(i,j) = 0$ und $\text{watched}(i,j) = 15$ gesetzt unter der Voraussetzung einer vorzeichenlosen 4-Bit-Repräsentation der Variablen.

3.2.5 Algorithmus der inkrementellen probabilistischen Kartierung

Das Verfahren funktioniert sowohl für die statische als auch für die dynamische Modellierung des Umgebungsmodells in seiner Gesamtheit folgendermaßen: Ausgehend von einer initialen Belegung des Gitters wird der erste Scan eingefügt, alle weiteren Scans sukzessive. Dazu wird für jeden eingefügten Scan das Bewegungsmodell des Roboters und demzufolge auch der Unsicherheitsbereich des Roboters berechnet und Partikel darin erzeugt. Jeder erzeugte Partikel wird mittels des Sensormodells bewertet und schließlich der Partikel mit maximalem Wert als neuer Roboterstandort in die Gitterkarten-Repräsentation integriert. Dieser Kartierungsalgorithmus wird in der Praxis in einem separaten Prozess modelliert. Für einen Zeitschritt nach Algorithmus 3.5 kann man die folgende Komplexitätsabschätzung angeben.

Algorithmus 3.5: Inkrementelle probabilistische Kartierung

Aufruf: **mapping-step()**
Input: Sensordaten s_k, Odometriedaten p_k und Odometriedaten p_{k-1}
Input: Karte m_{k-1} zum Zeitpunkt $k-1$
Output: Karte m_k

1 Berechne mit Bewegungsmodell Partikelmenge Par.
2 Bewerte die Partikelmenge Par mit dem Sensormodell.
3 Berechne beste Partikel par_{max} welches Konfiguration $\overline{p_k}$ repräsentiert.
4 Füge den Scan an die Konfiguration $\overline{p_k}$ in Gitterkarte ein.
5 Liefere die resultierende Karte m_k zurück.

Korollar 3.37 (Aufwand des Iterationsschritt der Kartierung) *Ein Zeitschritt der inkrementellen probabilistischen Kartierung nach Algorithmus 3.5 hat eine Zeitkomple-*

xität von $\mathcal{O}(q \cdot n \cdot m^2)$ *und eine Speicherkomplexität von* $\mathcal{O}(n+m^2+l^2)$. *Dabei bezeichnet* q *die Anzahl der Partikel,* m *die Anzahl der Gitterpunkte zur Diskretisierung der maximalen Reichweite des Sensors,* n *die Anzahl der Messpunkte des Sensors und* l *die maximale Anzahl der Gitterpunkte in einer Gitterdimension der Karte* m_{grid}.

Beweis: Die Partikelerzeugung und -verschiebung benötigt $\mathcal{O}(q)$ Aufwand. Jedes Partikel muss mit dem Sensormodell bewertet werden, was pro Partikel $\mathcal{O}(n \cdot m^2)$ kostet. Dabei wird das lokale Gitter verwendet. Die Bestimmung des maximalen Partikels benötigt eine Zeitkomplexität von $\mathcal{O}(q)$. Die Integration der Sensordaten an die berechnete Position in das globale Gitter benötigt einen Aufwand von $\mathcal{O}(n \cdot m^2)$. Insgesamt ergibt sich eine Zeitkomplexität von $\mathcal{O}(q \cdot nm^2)$.

Man benötigt ein lokales Gitter mit m^2 Gitterpunkten und ein globales Gitter mit l^2 Gitterpunkten. Zusätzlich werden noch die n Partikel benötigt. Insgesamt ergibt sich eine Speicherkomplexität von $\mathcal{O}(n + m^2 + l^2)$. □

3.3 Experimentelle Evaluation der inkrementellen Kartierung

In diesem Abschnitt soll das obige Verfahren zur Korrektur zweidimensionaler Daten durch Experimente verifiziert werden. Es soll zwischen verschiedenen Ansätzen zur Umgebungsrepräsentation unterschieden werden:

Einfaches statisches Umgebungsmodell
: Inkrementelle Kartierung ohne Einbeziehung der Dynamik. Dabei wird lediglich ein einfaches Gitter zur Kartenrepräsentation verwendet.

Statisches Umgebungsmodell
: Inkrementelle Kartierung ohne Einbeziehung der Dynamik. Dabei wird das dynamische Umgebungsmodell verwendet. Dabei werden aber keine dynamischen Zustände angenommen. Der Schwellenwert zwischen einem freien und einem belegten Zustand wird auf 0.5 gesetzt, sodass eine Bewertung nach Elfes [64] möglich wird.

Dynamisches Umgebungsmodell
: Inkrementelle Kartierung unter Einbeziehung der Dynamik.

Die Verfahren werden jeweils mit vier Sensordatensätzen aus vier Umgebungen getestet. Die Umgebungen sind das Erdgeschoss des Informatikgebäudes, der Naturwissenschaftliche Hörsaalbau, das Hanggeschoss des Informatikggebäudes und das Erdgeschoss des Mathematikgebäudes, auch als Testumgebungen 1 bis 4 bezeichnet. Die Sensordaten, aus denen die Umgebungsrepräsentation berechnet werden soll, wurden jeweils mit einer geführten Joystickfahrt unter Verwendung eines Pioneer II DX Roboters mit einem SICK LMS 200 Laser-Sensorsystem erzeugt.

Für die Experimente wurde bei augenscheinlicher schlechter Korrektur das Bewegungsmodell des Roboters adaptiert, indem ein größerer Rotationsfehler zugelassen und außerdem die Anzahl der Partikel erhöht wurde.

3.3.1 Testumgebung 1: Dynamische Objekte

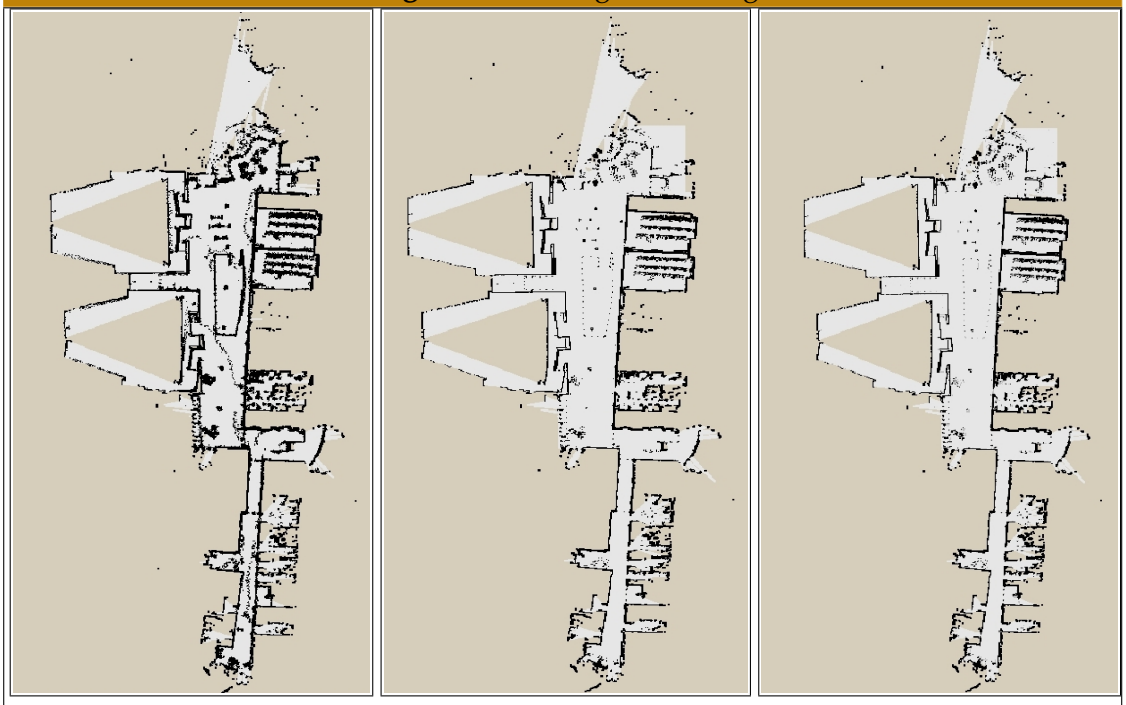
Abbildung 18: Kartierungstest 1

Die Umgebungsrepräsentation des ersten Tests besitzt 1826 Scans. Links ist der unkorrigierte Datensatz mit Fehlern, rechts die korrigierte Messpunktmenge mit Aufnahmepositionen dargestellt.

Die Testumgebung zeichnet sich dadurch aus, dass zwei Hörsäle in der hintersten Reihe durchfahren wurden und außerdem zwei Seminarräume. In der Umgebung waren oben und links (siehe Abbildung 18) große Glasflächen vorhanden. Zudem gab es einen nicht befahrbaren Innenraum in der Mitte. Im unteren Bereich der Karte waren viele offene Türen zu anderen, nicht kartierten Räumen vorhanden.

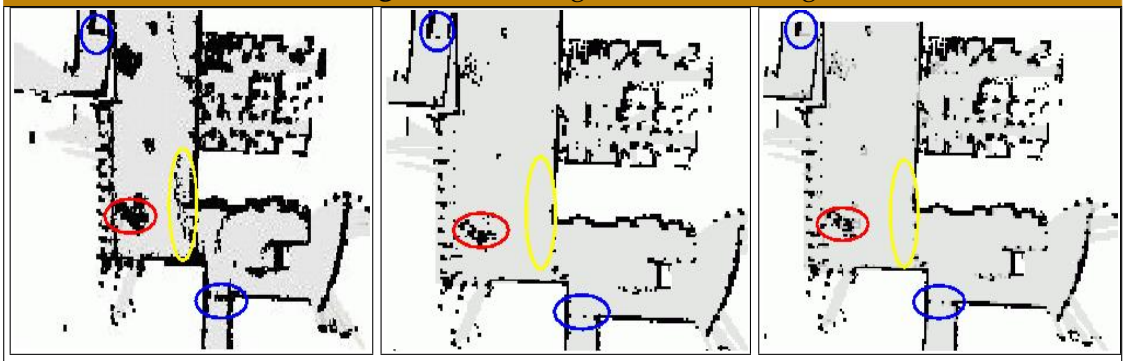
Die Karte wurde durch alle verwendeten Kartierungsverfahren gut korrigiert (siehe

Abbildung 19: Kartierungstest 1 - Ergebnisse



In der linken Karte ist das Ergebnis mit einer einfachen Gitterrepräsentation zu erkennen. In der mittleren Karte ist eine dynamische Repräsentation mit statischen Einstellungen und rechts mit dynamischen Einstellungen zu sehen. Die hellen Bereiche der Karten beschreiben den vom Sensor „gesehenen" Bereich.

Abbildung 20: Kartierungstest 1- Detailvergleich



In der linken Vergrößerung erkennt man, dass einzelne Punkte im Vergleich zur rechten Vergrößerung als statisch modelliert sind. Auch freie Punkte sind rechts dynamisch gekennzeichnet (grau).

Abbildung 19). Dabei kann man zwischen der einfachen Gitterrepräsentation und den anderen beiden Varianten auch bei kleiner Darstellung große Unterschiede ausmachen. Bei der einfachen Repräsentation werden einmal integrierte Hindernisse nicht mehr durch die Kartenrepräsentation korrigiert.

Im Detail ergeben sich sogar noch größere Unterschiede. In Abbildung 20 kann man deutlich erkennen, dass das zweite Verfahren dynamische Hindernisse wie offene und geschlossene Türen unter Umständen als Türen modelliert. Das dritte Verfahren hingegen erkennt diese Türen korrekt als dynamische Hindernisse. Zudem gibt es einige Konfigurationen im zweiten Verfahren, die als freie Konfigurationen modelliert wurden, im dritten Verfahren jedoch als dynamisch.

3.3.2 Testumgebung 2: Große Zyklen

Die zweite Testumgebung zeichnet sich durch große leere Bereiche aus. Der Hörsaalbau der Universität besitzt zudem Ankündigungstafeln, deren Beine für den Roboter sichtbar sind. Viele Säulen, drei Treppenaufgänge, eine Rampe und viele Schränke bevölkern die Umgebung. Zum Zeitpunkt der Kartierung waren nur die Versuchsleiter anwesend, so dass sich dynamische Implikationen in dieser Umgebung in Grenzen hielten.

Abbildung 21: Kartierungstest 2

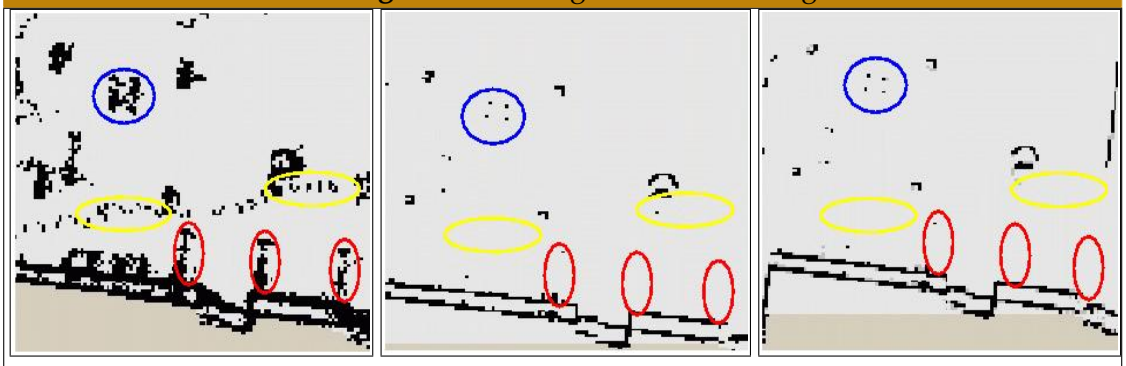
Unkorrigierte Scanliste (links) und korrigierte Scanliste (rechts) einer Explorationsfahrt im Hörsaalbau der Universität mit insgesamt 1759 Scans.

Abbildung 22: Kartierungstest 2 - Ergebnisse

Ergebnis der einfachen statischen Modellierung (links) und das Ergebnis der dynamischen Modellierung (rechts). Die Karte ist in beiden Versionen nicht vollständig korrekt und konsistent korrigiert.

Die Schwierigkeit in dieser Umgebung ergab sich durch die zyklische Explorationsfahrt (Abbildung 21) mit einem relativ großen Zykel von etwa 30 mal 30 Metern. Die Korrektur der Sensordaten erwies sich als erwartet schwierig, sodass in mehreren Kartierungversuchen mit angepasstem Odometriefehler ein adäquates Kartierungsergebnis präsentiert werden kann (Abbildung 22).

Abbildung 23: Kartierungstest 2- Detailvergleich



Mehrere vergrößerte Ausschnitte der obigen Testumgebung 2 zeigen die feinen Unterschiede der Ansätze. Links die einfache statische Modellierung, in der Mitte die dynamische Repräsentation mit statischer Bewertung und rechts die dynamische Modellierung.

Die Unterschiede lassen sich in dieser Umgebung nur anhand der reinen statischen und der dynamischen Modellierung messen. Während ein Versuchsleiter sich mit Hilfe seiner Beine in der statischen Karte verewigt hat, ist er in der dynamischen Modellierung nicht mehr vorhanden.

Im Detail gibt es jedoch für die einfache Modellierung auch positive Ergebnisse. So werden Hindernisse, die nur in einem oder zwei Scans aufgenommen wurden, in der dynamischen Gitterkarten-Repräsentation als freier Zustand modelliert. Damit fallen auch die meisten vorhandenen Schautafeln der Umgebung aus dieser Karte heraus. In der statischen Gitterkarte sind sie jedoch vorhanden. Siehe dazu auch die Abbildung 23.

3.3.3 Testumgebung 3: Multiple Räume

Die Testumgebung 3 zeichnet sich dadurch aus, dass sehr viele verschiedene Räume angefahren worden sind. Dadurch mussten viele Rotationsbefehle durch das Kartierungsverfahren ausgeglichen werden. Dabei ist zu beachten, dass bei zu schneller Rotation durch den Einsatz eines 180 Grad-Laserscanners benachbarte Scans im schlechtesten Fall keinen gemeinsamen Messpunkt aufweisen. Durch den Besuch vieler Räume kommt es zudem zu einem Verstärkungseffekt: Durch eine Sensoraufnahe in einem Raum, der anschließenden Durchfahrt durch eine Tür und der darauffolgenden neuen Sensoraufnahe in einem neuen Raum ergeben sich ebenfalls keine gemeinsamen Messpunkte.

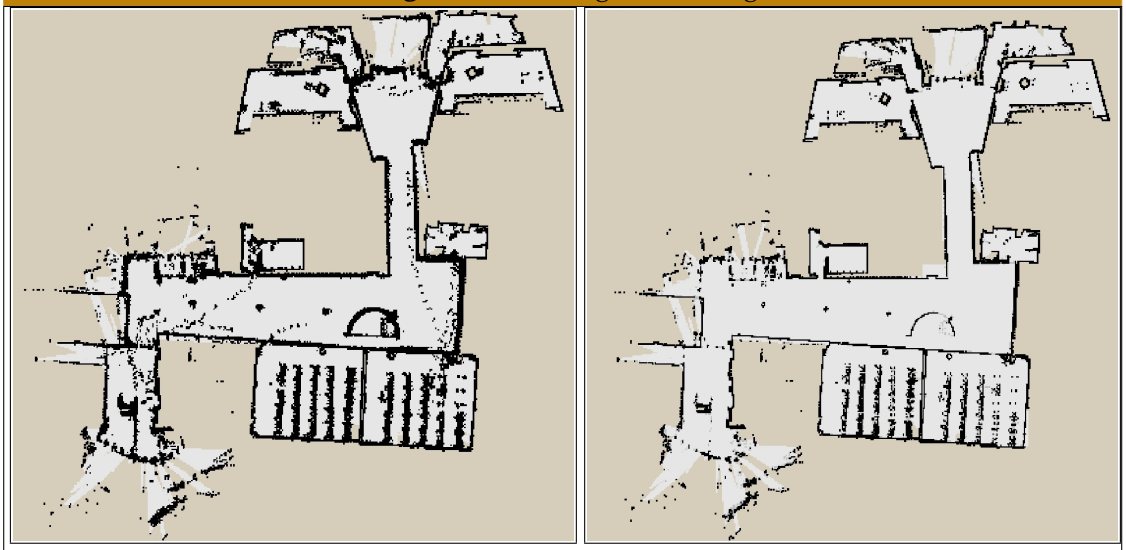
Abbildung 24: Kartierungstest 3

Unkorrigierte Scanliste (links) und korrigierte Scanliste (rechts) einer Explorationsfahrt im Hanggeschoss des Informatikgebäudes mit 1574 Scans.

In der Testumgebung, dargestellt in Abbildung 24, waren zusätzlich zum Versuchsleiter weitere dynamische Objekte anwesend. Die Räume waren durch Stühle und Tische möbliert, was sich auch durch relativ unstrukturierte Sensordaten zeigte. Die Umgebung war ansonsten durch große Flächen relativ einfach zu kartieren.

Abbildung 25 liefert wiederum die bekannten Unterschiede zwischen statischer und dynamischer Modellierung. Zur Erzeugung der Ergebniskarten waren ebenfalls Anpassungen an den Odometriewerten und den Partikelwerten notwendig. Dabei zeigt sich trotzdem, dass kleine Kartierungsfehler durch einzelne falsch integrierte Scans nicht vermieden werden konnten.

Abbildung 25: Kartierungstest 3 - Ergebnisse

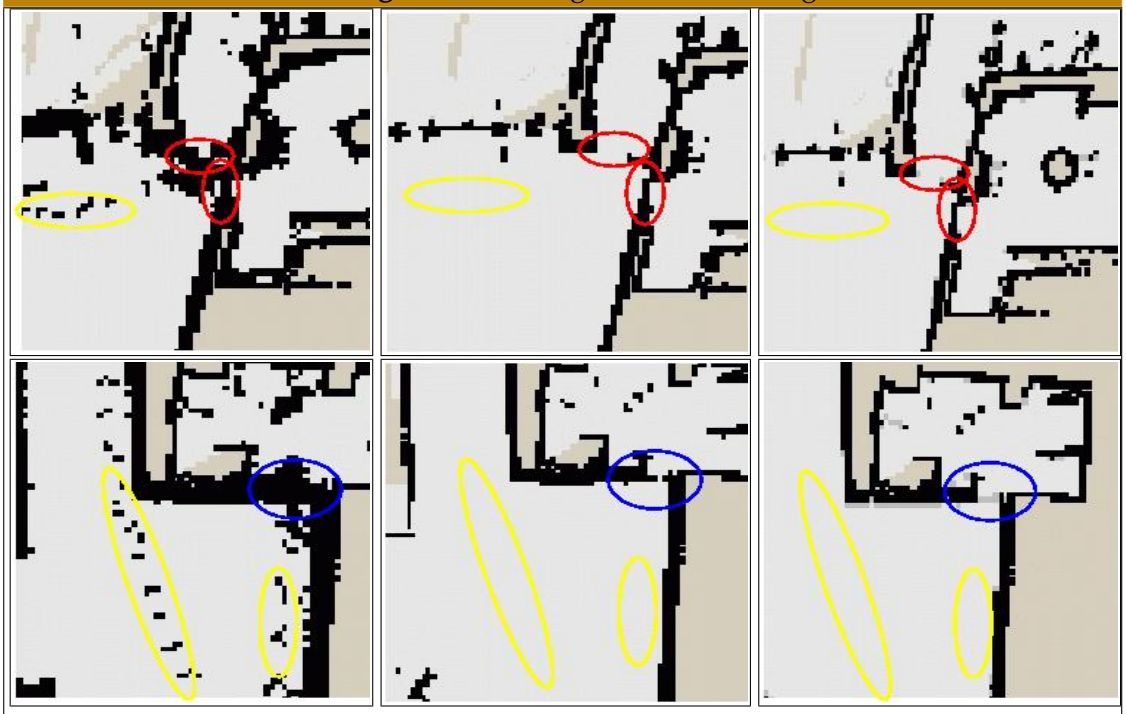


Links ist die korrigierte statische Repräsentation mit den bekannten Modellierungsfehlern abgebildet, rechts die dynamische Repräsentation.

Betrachtet man die Kartierungsergebnisse im Detail an zwei verschiedenen Positionen der Karten, dargestellt in Abbildung 26, erkennt man eine unterschiedliche Modellierungsgüte von Türen. Während der dynamische Ansatz zwei von drei Türen als dynamisch modelliert, wird im dynamischen Modell mit statischer Modellierung nur eine von drei Türen erkannt, in der einfachsten Repräsentation gar keine.

An dieser Stelle besteht für das dynamische Verfahren noch Modellierungsbedarf. Eine Tür wurde dabei von einer Mehrzahl von Scans als tatsächlich belegt modelliert, obwohl deren Status einmal als frei deklariert war, da der Roboter den dahinterliegenden Raum ja kartiert hatte. Ein erweiterter Kartierungsansatz durch Tracking-Methoden könnte an dieser Stelle Abhilfe schaffen. Könnte mit einem Tracking-Algorithmus die Menge der dynamischen Objekte detektiert und für die Kartierung ausgeschlossen wer-

Abbildung 26: Kartierungstest 3- Detailvergleich



Mehrere vergrößerte Ausschnitte zeigen die feinen Unterschiede der Ansätze.

den, könnte man eine Zustandsänderung eines Gitterpunktes von *Dynamisch* zu *Frei* dann als eindeutiges Indiz für ein dynamisches Hindernis ansehen und entsprechend modellieren.

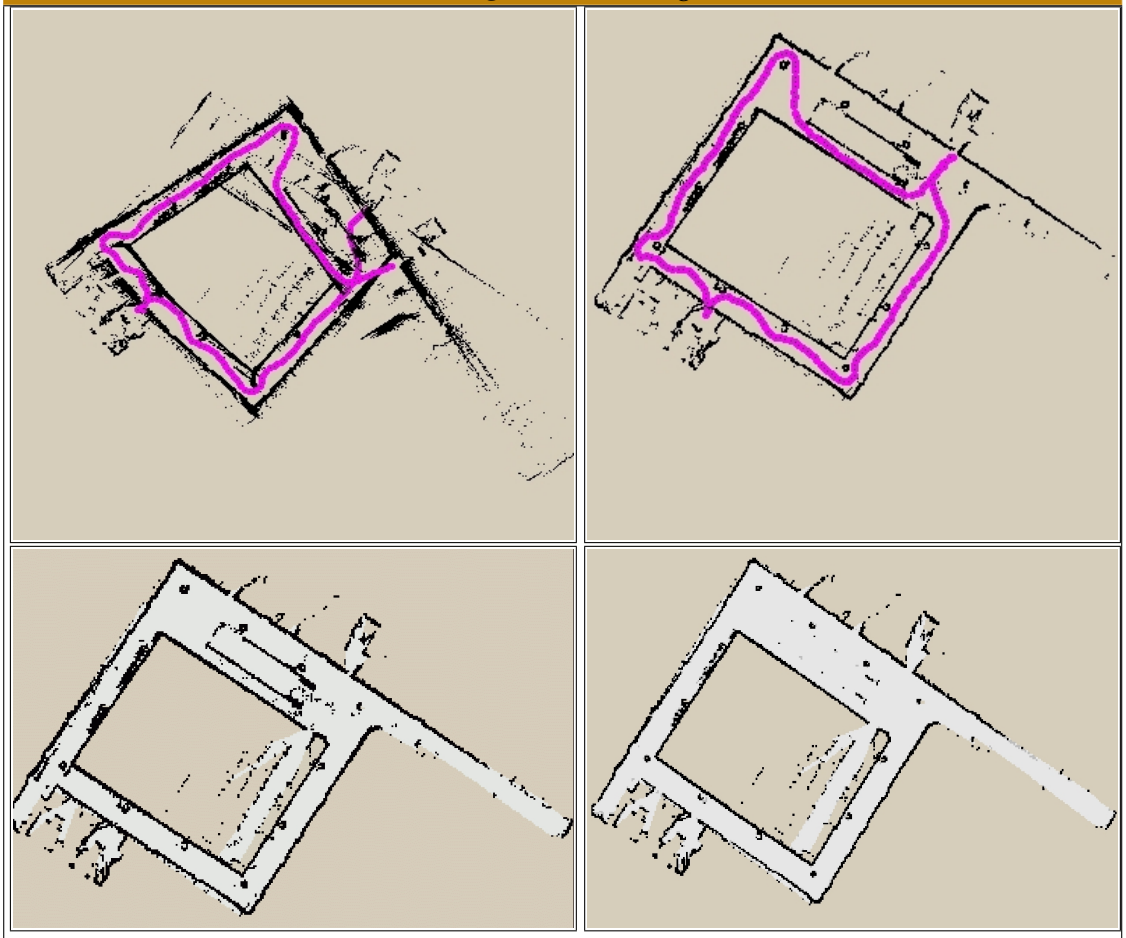
3.3.4 Testumgebung 4: Kleine Zyklen

Die letzte Testumgebung repräsentiert einen Teil des Erdgeschosses des Mathematik-Gebäudes der Universität Würzburg (Abbildung 27). Diese Umgebung besitzt einen recht kleinen Zykel, der mit dem Kartierungsverfahren problemlos erkannt wird wenn die Odometriedaten gut genug sind. Diese Umgebung zeichnet sich weiter durch ihre langen Gänge aus, welche unter Umständen schief und nicht größenerhaltend kartiert werden können. Die Kartierungsergebnisse in Abbildung 27 bestätigen die Ergebnisse der vorherigen Testumgebungen und zeigen die Anwendbarkeit der Algorithmen.

3.3.5 Vergleich der Ansätze

In den vier Testumgebungen konnte die Leistungsfähigkeit des inkrementellen probabilistischen Kartierungsverfahrens verdeutlicht werden. Die einfache statische Modellierung hat große Schwächen, modelliert jedoch alle jemals auftretenden Objek-

Abbildung 27: Kartierungstest 4



Links oben ist die unkorrigierte Menge der Sensordaten, rechts oben die korrigierte statische Repräsentation mit gefahrenem Weg abgebildet. Links unten erkennt man die statische und rechts unten die dynamische Korrektur.

te. Zudem ist diese Repräsentation in der Praxis sehr schnell und benötigt weniger Speicherplatz. Die vorgestellte dynamische Modellierung ist in der vorliegenden Implementierung deutlich langsamer, besitzt jedoch einen klaren Modellierungsvorteil. Dynamische Hindernisse werden nicht immer, aber häufiger im Vergleich mit dem dynamischen Verfahren mit statischer Modellierung nach Elfes erkannt.

Die vorgestellten Verfahren bieten eine adäquate Lösung des Problems 3.14 an und sind in der Lage, mit Hilfe eines Versuchsleiters in kürzester Zeit korrigierte Gitterkarten-Modelle der Einsatzumgebung zu liefern.

3.4 Erzeugen von Segmentkarten

Nachdem in den obigen Abschnitten eine Heuristik zur Lösung des Problems 3.14 der Korrektur von Sensordaten beschrieben wurde, benötigt man weiter ein Verfahren, welches aus den korrigierten Sensordaten Segmente rekonstruiert und das Problem 3.15 löst.

3.4.1 Lösungsansätze in der Literatur

Das Erstellen geometrischer Karten wird in der Literatur durch die Extraktion von Segmenten aus Sensordaten bewerkstelligt (siehe dazu auch Kapitel 2.2.2). Aktuelle Ansätze lösen dabei neben dem inkrementellen Problem 3.30 gleichzeitig das SLAM-Problem 3.14 mit feature-basierten Karten unter Verwendung eines Kalman-Filters [14, 53, 109, 219, 220].

Hauptproblem, welches dabei auftaucht, ist neben der Berechnung der optimalen Schätzung der Roboter-Position das damit verbundene Verschmelzen zugeordneter Segmente. Bewegt sich der Roboter in seiner Einsatzumgebung, tauchen Wände oft zunächst nur als kleines Segment auf. Im weiteren Verlauf kann sich dieses Segment ausdehnen, je nachdem, wieviel man von der zugehörigen Wand schon gesehen hat.

Bewegen sich dynamische Objekte in der Einsatzumgebung, verstärkt sich der Effekt. Eine lange Wand, repräsentiert durch ein Segment, wird plötzlich durch die Sichtbarkeit eines dynamisches Objektes in den Sensordaten in mehrere Teile fragmentiert.

Ein Ansatz zur Zusammenfassung von Segmenten liefert Gonzales [79], in dem zu erzeugten Linien ihre Extraktions- und Ungenauigkeitsparameter gespeichert werden. Zusätzlich wird ein sogenannter Viewing-Sector definiert, welcher den sichtbaren Bereich zwischen Sensor und entstandenem Segment repräsentiert. Die extrahierten Linien des lokalen Scans werden dann mit denen der aktuellen Karte verglichen, indem überprüft wird, ob ein solches neues Segment in den Viewing-Sector fällt. Für hereinragende Segmente wird nur dieser Teil betrachtet und die Distanz mit dem Kartensegment berechnet. Liegen die Liniensegmente nahe beisammen, werden sie zu einem neuen Segment verlängert mit entsprechendem Viewing-Sector.

Das größte Problem, das beim Verschmelzen von Segmenten auftritt, ist die zunehmende Ungenauigkeit der Kartenrepräsentation. Dies rührt daher, dass die Darstellung mittels Segmenten eine Approximation der Messdaten ist. Durch einen Verschmelzungsschritt wird die Approximation der zugrundeliegenden Punktmenge verschlechtert. Außerdem ist zunächst nicht ersichtlich, ob und wann ein Verschmelzen zweier Segmente erlaubt werden kann und wann nicht. Ein weiteres Problem des Ansatzes ist die enorm ansteigende Menge von geometrischen Features, so dass adäquate Verschmelzungsoperationen von Segmenten nur mit entsprechenden Datenstrukturen in Echtzeit möglich sind.

Eine andere, interessante Möglichkeit bietet sich in der Betrachtung der gesamten zu einem Zeitpunkt vorhandenen Messwerte der Sensordaten. Das Problem besteht dann darin, mit Hilfe dieser Messwerte die Umgebungskarte zu rekonstruieren (Problem 3.29 und Problem 3.31). Lösungsansätze zur effizienten Rekonstruktion von Objekten aus Sensordaten liefert ein Teilgebiet der Algorithmischen Geometrie, das Teilgebiet der Kurven-Rekonstruktion. Bekannte Algorithmen wie der Crust-Algorithmus [11] versuchen Punktmengen, welche aus stetigen, differenzierbaren und geschlossenen Kurven oder Konturen entstanden sind, zu rekonstruieren. Mit Hilfe der Anforderungen kann man mögliche benachbarte Punkte eines betrachteten Punktes in geeigneter Weise geometrisch bestimmen und verbinden. Reduziert man die Anforderungen an die Punktmenge kommt man zu weiteren Algorithmen, etwa dem NN-Crust [57] oder dem Conservative-Crust [58]. Die realistischste Variante ist der Gathan-Algorithmus [58], welcher nur noch planare, einfache nicht-schneidende Kurven voraussetzt, welche aus mehreren Teilen bestehen, Endpunkte und auch Ecken besitzen, und damit nichtstetige Kurven sein dürfen. Algorithmen zur Kurven-Rekonstruktion werden in der Computer-Vision dazu verwendet, die Kontur von 3-D Objekten, aufgenommen durch 3D-Laserscanner, zu rekonstruieren [10, 11].

Bei der Rekonstruktion von Segmenten aus den 2D-Sensordaten kann man auf das Wissen zurückgreifen, dass das Innere einer Umgebung im wesentlichen aus Polygonzügen besteht. Dabei ist eine Stetigkeit der Polygonzüge ausgeschlossen. Eine ausführliche Darstellung liefert [9].

Nachfolgend werden die in der Literatur vorkommenden Problemlösungen aufgegriffen, um zwei Verfahren zu entwickeln, welche aus korrigierten Sensordaten feature-basierte Kartenrepräsentationen erzeugen. Diese liefern teilweise unter Echtzeitbedingungen feature-basierte Kartenrepräsentationen, deren Güte mit Hilfe feature-basierter Lokalisationsanfragen in Kapitel 4 evaluiert wird.

3.4.2 Feature-basierte Kartierung durch Kurvenrekonstruktion

Ein wichtiger Punkt des Problems 3.29 im Vergleich mit dem Kurven-Rekonstruktionsproblem liegt darin, möglichst lange Segmente zu erhalten. Es reicht also nicht, nur Punkte zu verbinden, sondern diese Verbindungen sind dann zu maximal-langen Segmenten auszudehnen.

Hier soll deshalb eine Variante der Kurven-Rekonstruktionsverfahren verwendet werden, welches aus der Gesamtmenge der generierten Messpunkte lange Segmente extrahiert. Allerdings erkauft man sich dieses mit zwei entscheidenden Nachteilen: Es müssen Segmente aus einer riesigen Anzahl von Messpunkten generiert werden. Bei einer Karte mit 1500 Scans ergeben sich $360 \cdot 1500 = 540.000$ Messpunkte. Außerdem besitzen die Punkte keine vorher vorhandene lokale Ordnung bezüglich anderer benachbarter Messpunkte mehr und liegen beliebig in der euklidischen Ebene.

Diese Nachteile werden durch zwei zusätzliche Algorithmen ausgeglichen. Zum einen durch einen verallgemeinerten Reduktionsfilter und zum anderen durch das Zusammenfassen von Segmentketten zu einzelnen Segmenten.

Im folgenden soll der hier verwendete Algorithmus zur Kurven-Rekonstruktion vorgestellt werden. Dazu wird zunächst eine allgemeine Heuristik zur Reduktion von Punkten vorgestellt. Anschließend soll die Heuristik zur Erzeugung der Segmente beschrieben werden. Abschließend wird eine allgemeine Heuristik zur Reduktion und zum Zusammenfassen von Segmenten erläutert.

Reduktion der Punktmenge

Die Reduktion der Punktmenge ist ein wesentlicher Aspekt in der Anwendung des Algorithmus, erkauft er doch eine bessere Laufzeit durch viel weniger zu betrachtende Punkte. Die Reduktion ist eine Generalisierung des Reduktionsfilters [92] auf allgemeine Punktmengen. Innerhalb eines gewissen Radius um einen frei gewählten Punkt, der sogenannten Punktwolke, werden alle Punkte zu einem neuen Punkt zusammengefasst. Die Punktwolke wird durch den Schwerpunkt aller Punkte ersetzt (siehe Algorithmus 3.6).

Algorithmus 3.6: Reduktion der Punkte

Aufruf: **reduce-points()**
Input: Minimale erlaubte Anzahl von Punkten MIN-POINTS, Reduktionsradius r, Punktmenge S
Output: Menge S' reduzierter Messpunkte.

```
1  S' = ∅ {Initialisiere S' mit der leeren Folge}
2  while {S nicht leer} do
3     p = p_0 {hole ersten Punkt aus der Folge}
4     R=range-search(p,r,S) {suche alle Punkte im Umkreis r von p}
5     if {|R| ≥ MIN-POINTS} then
6        p_mid = point-of-gravity(R) {Berechne Schwerpunkt}
7        S' = S' ∪ p_mid
8        S = S − R
9     else if  then
10       S = S − p
11    end if
12 end while
13 return S'
```

Man kann anstatt eines Reduktionsansatzes auch die berechneten belegten Konfigurationen der Gitterkarten als Reduktionsmechanismus verwenden. Dabei bekommt man jedoch eine sehr homogene Präsentation der Punktmenge, so dass dem ein Reduktionsansatz vorgezogen werden sollte. Ein Filtern der Sensordaten unter der Prämisse

dynamischer Umgebungen bestärkt den Einsatz der Konfigurationen der Gitterkarte, wurde aber nicht weiter verfolgt.

Korollar 3.38 (Komplexität der Reduktion der Punkte) *Es sei S eine Menge von n Meßpunkten in der euklidischen Ebene. Dann berechnet der Algorithmus 3.6 eine reduzierte Punktmenge S' mit m Meßpunkten in $\mathcal{O}(m(k + \log^2 n))$ Zeitschritten und einem zusätzlichen Speicherplatzbedarf von $\mathcal{O}(n \log n)$. Dabei ist k die maximale Anzahl von Punkten, die bei einer Bereichsanfrage zurückgeliefert werden kann.*

Beweis: Zur Speicherung der Meßpunkte wird ein Point-Dictionary [153] als Datenstruktur verwendet, dessen Aufbau $\mathcal{O}(\log^2 n)$ Zeitschritte und zusätzlichen Speicherplatz von $\mathcal{O}(n \log n)$ benötigt. Das Point-Dictionary ist als Kombination eines zweidimensionalen Range-Trees und eines Voronoi-Diagramms implementiert. Für jeden der insgesamt m neuen Meßpunkte muss bei der Reduktion eine Bereichsanfrage an die Datenstruktur gestellt werden, was jeweils einen Aufwand von $\mathcal{O}(k+\log^2 n)$ bedeutet. Zur Reduktion aller n Scanpunkte ergibt sich ein Zeitaufwand von $\mathcal{O}(m(k + \log^2 n))$ und ein Speicheraufwand von $\mathcal{O}(n \log n)$. □

Erzeugen von Segmenten

Nächster Schritt ist die Erzeugung von Segmenten aus der reduzierten Menge der Meßpunkte. Da die lokalen Nachbarschaften der Punkte durch eine globale Betrachtungsweise verlorengehen, benötigt man eine Heuristik um geeignete benachbarte Punkte eines resultiertenden Punktes p der Menge S' zu bestimmen. Hier wird eine Variante des NN-Crust-Algorithmus verwendet, welcher als einfache Heuristik den nächsten Nachbarn zu p bestimmt, welcher innerhalb eines bestimmten Radius r_{max} liegen muss. Ein experimentell-bestimmter Wert von $r_{max} \in [0.1, 0.5]$ hat sich bewährt. Die algorithmische Bestimmung erledigt Algorithmus 3.7.

Korollar 3.39 (Rekonstruktion der Kurve) *Sei S' eine Menge von n Meßpunkten. Dann benötigt der Algorithmus 3.7 $\mathcal{O}(n(k + \log^2 n))$ Zeitschritte und $\mathcal{O}(l)$ Speicherplätze zur Berechnung von F mit l Elementen. Dabei ist k die maximale Anzahl von Punkten, die bei einer Bereichsanfrage zurückgeliefert werden kann.*

Beweis: Für jeden der n Meßpunkte muss eine Bereichsanfrage an die verwendete Datenstruktur gestellt werden. Wie oben wird ein Point-Dictionary verwendet, so dass dieses $\mathcal{O}(k + \log^2 n)$ Zeitschritte benötigt. Zusätzlich werden für jeden Punkt $\mathcal{O}(k)$ Berechnungsschritte zur Bestimmung der minimalen Distanz benötigt. Das Löschen eines Punktes benötigt $\mathcal{O}(\log^2 n)$ zusätzliche Schritte. Insgesamt ergibt sich die obige angegebene Zeitkomplexität. Zusätzlichen Speicherplatz benötigt man für die Erzeugung von l Segmenten aus der Menge der Meßpunkte. □

Algorithmus 3.7: Heuristik zur Kurven-Rekonstruktion

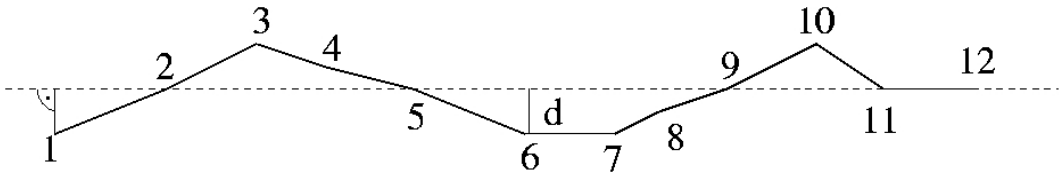
Input: reconstructing-step()
Input: Reduzierte Menge von Punkten $S' = (p_1, \ldots, p_n)$, Distanz r_{max}
Output: Menge von Segmenten F

```
1  while {noch nicht alle Punkte betrachtet} do
2    Hole den aktuellen Punkt p.
3    Suche aus der Punktmenge den nächsten Nachbarn q ≠ p.
4    if d(p, q) < r_max then
5      Generiere ein Segment pq und füge es der Menge F hinzu.
6      Lösche p und setze p := q.
7    else
8      Lösche p und setze p := q.
9    end if
10 end while
11 return F.
```

Reduktion der Segmente

Da durch Algorithmus 3.7 immer noch sehr kurze Segmente entstehen, sollen diese zu größeren Segmenten zusammengefasst werden. Eine Methode dazu liefern Kurozumi und Davis [130, 165, 223] mit der sogenannten Minimax-Methode, die Segmentketten innerhalb eines bestimmten Bereichs zusammenfasst. Hier wird eine vereinfachte Methode (siehe Abbildung 28) verwendet, indem durch die Anfangs- und Endpunkte der ersten Segmente einer Segmentkette eine Ausgleichsgerade gelegt wird und für weitere Punkte der Abstand zu dieser Geraden berechnet wird. Dieser darf den maximalen zulässigen Abstand d_{max} nicht überschreiten, da ansonsten das Segment nicht zum neuen Segment hinzugenommen wird. In diesem Fall wird das alte entstandene Segment der Menge F zugeschlagen und über den Anfang der verbliebenen Segmentkette eine neue Ausgleichsgerade gelegt und der Prozeß wiederholt.

Abbildung 28: Vereinfachte Minimax-Methde



Wird für einen Punkt der Folge die Distanz d_{max} nicht überschritten, wird das resultierende Segment akzeptiert. Andernfalls wird das Segment mit allen vorherigen Punkten erzeugt und ein neues Segment begonnen.

Dieser maximal zulässige Abstand d_{max} ist ein globaler einzustellender Parameter. Ein Wert aus $[0.1, 0.5]$ hat sich in der Praxis bewährt. Die Schnittpunkte zwischen dem

Lot auf die Ausgleichsgerade vom ersten betrachteten Punkt bis zum letzten gültigen Punkt bilden dann das neue Segment.

Algorithmus 3.8: Reduktion der Segmente

Aufruf: **simple-minimax()**
Input: Folge von benachbarten Segmenten $F = (p_1, \ldots, p_{l+1})$,Distanz d_{max}
Output: Liste von Segmenten F_e.

```
1  l :=empty
2  F_e :=empty
3  p_0=n-th-point(P, 0)
4  p_1=n-th-point(P, 1)
5  l = l ∪ p_0p_1
6  for {i=2 to numpoints(P) − 1} do
7     line:=make-line(l)
8     p=n-th-point(P, i)
9     if {distance-to-line(p,line) < d_max} then
10       l = l ∪ p
11    else
12       s=create-segment(line)
13       F_e = F_e ∪ s
14       l:=empty
15       l := l ∪ p
16       i:=i+1
17       p=n-th-point(P, i)
18       l := l ∪ p
19    end if
20 end for
21 return F_e
```

Korollar 3.40 (Reduktion der Segmente) *Der Algorithmus 3.8 hat eine Zeitkomplexität von $\mathcal{O}(l)$, wobei l die Anzahl der Features aus F_e ist.*

Beweis: Die Folge P der Punkte repräsentiert eine Kontur der Umgebung, wobei benachbarte Segmente durch Algorithmus 3.7 erzeugt wurden. Somit benötigt man linearen Aufwand in der Anzahl der Segmente zur Anwendung des Algorithmus. Zusätzlicher Speicheraufwand wird nicht benötigt. □

Feature-basierte Kartierung durch Kurvenrekonstruktion

Der allgemeine Ansatz zur Erzeugung feature-basierter Karten durch Kurvenrekonstruktions-Algorithmen und damit ein Lösungsansatz für Problem 3.29 liefert Algorithmus 3.9. Es können dabei insbesondere auch andere Algorithmen, etwa der NN-Crust-Algorithmus damit eingesetzt werden.

Algorithmus 3.9: Feature-basierte Kartierung durch Kurvenrekonstruktionn

Aufruf: **feature-based-curve-reconstruction()**
Input: Punktmenge $S = (p_1, \ldots, p_n)$
Output: Menge F_e von Segmentfeatures
1 Reduziere die Punktmenge S durch Punktreduktion zu S'.
2 Anwendung eines Kurven-Rekonstruktions-Algorithmus auf S' mit Ergebnis F.
3 Reduktion der Segmente von F.
4 **return** F_e.

Die Komplexität des Algorithmus 3.9 mit der in Algorithmus 3.7 betrachteten Heuristik zur Verbindung von Punkten ergibt sich folgendermaßen:

Korollar 3.41 (Komplexität der Kurvenrekonstruktion mit Punktreduktion) *Es sei S eine Menge von n Meßpunkten. Dann benötigt der Algorithmus 3.9 $\mathcal{O}(m(\log^2 n + \log^2 m) + l)$ Zeitschritte, um die Menge S auf m Punkte zu reduzieren und eine Menge F_e mit l Features zu berechnen. Dazu wird ein Speicherbedarf von $\mathcal{O}(n \log n + l)$ fällig.*

Beweis: Addieren der Komplexitäten der vorgestellten Algorithmen liefert das gewünschte Resultat. □

Eine Modifikation des obigen Algorithmus zu einem inkrementellen Verfahren erweist sich schwierig. Abhängig von der Eingabe der neuen Sensordaten muss man Regionen in der Karte identifizieren, in welche die neuen Sensordaten fallen. Alle Segmente welche aus dieser Region entstanden sind, müssen zerstört werden und für diese Region eine neue Reduktion der Punktmenge berechnet werden. Der inkrementelle Ansatz ist in dieser Arbeit unberücksichtigt gelassen worden.

3.4.3 Feature-basierte Kartierung durch Segment-Merging

Als zweiter Ansatz soll hier ein dem Verfahren von Gonzales [79] ähnlicher Algorithmus realisiert werden. Betrachte dazu die Probleme 3.28 und 3.30. Dort werden $k-1$ Sensordaten betrachtet, welche im folgenden schon korrigiert und zusammengefasst sein sollen. Diese Daten repräsentieren die Featuremenge G^{k-1} der Modelldaten zu einem Zeitpunkt $k-1$. Aus den neuen Sensordaten s_k können mit den Verfahren aus Kapitel 2.2.2 Segmente extrahiert werden, so dass eine Feature-Menge F zur Verfügung steht.

Segmentreduktion

Für eine Modellfeature-Menge G^{k-1} aller vorheriger Sensordaten betrachtet man zunächst ein einzelnes Segment g. Zu diesem werden alle Features f aus der Bildfeature-Menge F gesucht, deren Abstand $d(f, g)$ kleiner als eine vorgegebene Schwellenwertdistanz e_{trans} und deren gemeinsamer Innenwinkel kleiner einem festgelegten

Schwellenwinkel e_{rot} ist. Dieses unterscheidet sich von dem im Kapitel 2.2.3 vorgestellten Verfahren, da hier Segmente zugelassen werden, welche die Fehlerhülle eines Segmentes berühren und nicht darin liegen.

Wird ein solches Segment f gefunden, dann wird eine Ausgleichsgerade durch die Menge der Endpunkte der Segmente f und g gelegt und das Lot der Endpunkte auf die Ausgleichsgerade berechnet. Die Schnittpunkte mit der größten euklidischen Distanz auf der Ausgleichsgeraden bilden die neuen Endpunkte des sogenannten Supersegments $\overline{fg}$.

Die beiden zur Entstehung des Supersegments beitragenen Segmente f und g werden gelöscht und das Supersegment $\overline{fg}$ an die Segmentmenge G^k angehängt. Dieser Vorgang wird solange iteriert, wie für ein Segment g in der Menge F ein passendes Gegenstück gefunden wird.

Plausibilität des Zusammenfassens

In diesem Abschnitt soll geklärt werden, wann es plausibel ist zwei Segmente s_1 und s_2 nach obiger Heuristik zusammenzufassen. Die maximale Fehlerhülle von s_1 und s_2 sei o.B.d.A. gegeben durch den Parameter δ_f für das Segment s_1.

Das Segment $s_2 = (p_1, p_2)$ sei nun um einen Abstand e_{trans} von s_1 entfernt und besitze eine Winkelabweichung von s_1 von e_{rot}. Zur vereinfachten Betrachtung sei die X-Achse durch das Segment s_1 gelegt, wie in Abbildung 29 dargestellt.

Der Punkt p_1 markiert den Punkt mit minimalem Abstand zu s_1 und besitzt demnach die Y-Koordinate e_{trans}. Das Segment s_2 besitze die Länge $l(s_2)$. Damit kann man für p_2 die Y-Koordinate $l(s_2) \cdot \sin e_{rot}$ bestimmen.

Damit p_2 noch innerhalb der Fehlerhülle von s_1 liegt, wenn man s_1 und s_2 als Supersegment betrachtet und die Fehlerhülle von s_1 entsprechend erweitert, muss demzufolge die folgende Bedingung gelten:

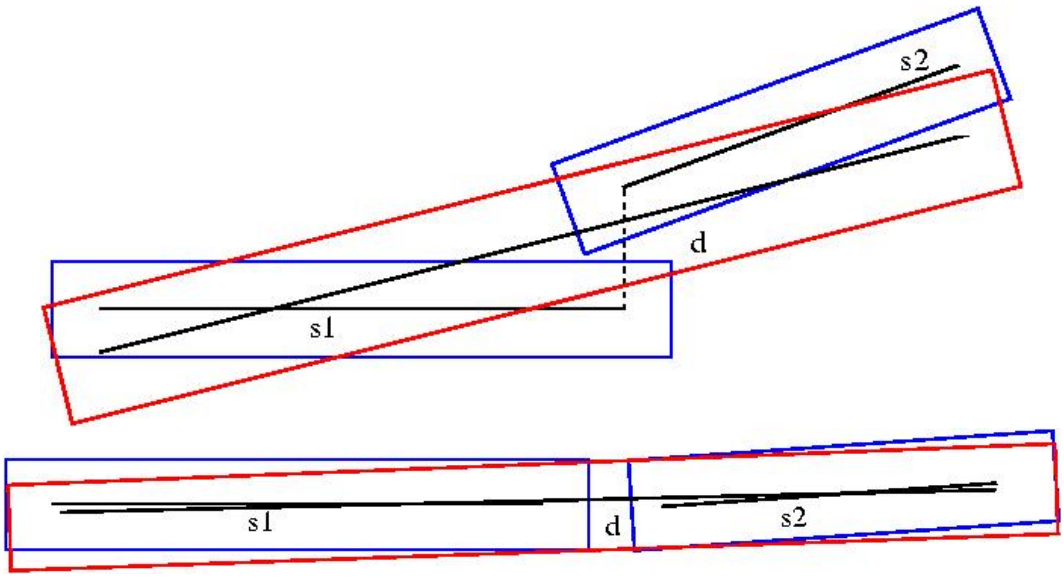
$$l(s_2) \cdot \sin e_{rot} + e_{trans} \leq \delta_f \tag{3.23}$$

Somit hat man ein Berechnungskriterium, welches entscheidet ob eine Verschmelzung zweier Segmente zulässig ist oder nicht.

Für eine sukzessive Anwendung des Entscheidungskriteriums ergibt sich aber das Problem, dass die Fehlerhülle abhängig von der Lage des Segments in der Karte ist. Durch ein einmaliges Verschmelzen zweier Segmente verändert sich die Lage der Fehlerhülle. Je öfter ein Segment mit anderen Segmenten in einen solchen Vorgang verwickelt ist, desto ungenauer kann die Repräsentation des entsprechenden Segments werden. Ob solche Vorgänge die tatsächliche Kartenrepräsentation beeinträchtigen, wird in expe-

rimentellen Vergleichen in Kapitel 4 bei der Anwendung von Lokalisationsverfahren untersucht.

Abbildung 29: Zusammenfassen zweier Segmente



Zwei Segmente s_1 und s_2 mit gegebener Fehlerhülle (blau). In der oberen Abbildung besäße das zusammengefasste Segment die rote Fehlerhülle. In diesem Fall ist eine Verschmelzung nicht zulässig. Im unteren Fall ist eine Verschmelzung zulässig, da das resultierende Segment noch in der verlängerten Fehlerhülle von s_1 läge.

Löschen schlechter Segmente

Bei der Integration und dem Zusammenfassen von Segmenten kann man folgende Beobachtung machen: Ein Feature wird in der Regel mehrmals in einer Sequenz von Bewegungsschritten vom Roboter gesehen. Demzufolge müssen diese Features zu verschiedenen Zeitschritten auch miteinander verschmolzen werden. Dabei tauchen auch Segmente auf, die nur einmal in der Umgebung gesehen wurden. Solche Segmente repräsentieren unter Umständen dynamische Objekte der Karte. Eine zusätzliche Heuristik kann diese Features entfernen.

Es sei $g : G \to \mathbb{N}_{>0}$ eine Funktion, welche die Anzahl der Mergingschritte als Bewertung der Featuremenge G^k verwendet. Die Bewertung $g(h)$ eines Features $h \in G^k$, welches aus $f_1 \in G^{k-1}$ und $f_2 \in F$ entstanden ist, enthält dann eine um 1 höhere Bewertung als die maximale Bewertung von $g(f_1)$ oder $g(f_2)$.

Dann kann man für eine feature-basierte Karte eine Kartengüte definieren:

Definition 3.42 (Kartengüte) *Die Kartengüte einer feature-basierten Karte* m^k_{feat} *zu einem Zeitpunkt* k *legt einen Schwellenwert* $minseg \in \mathbb{N}$ *für die Gütebewertung* $g(f)$ *eines jeden Features* f *der erzeugenden Featuremenge* G^k *der Karte fest. Für alle Features gilt* $g(f) >= minseg$. *Der Wert* $minseg$ *bezeichnet dann die Kartengüte der Karte* m^k_{feat}.

Nachteil dieser Heuristik ist die Tatsache, dass tatsächliche Segmente, die nicht zusammengefasst werden konnten, so ebenfalls verschwinden können. Diese Problematik kann man (insbesondere für die inkrementelle Variante) umgehen, indem Features aktiviert sind, wenn sie die Schwelle $minseg$ erreicht haben. Sodann tauchen in der tatsächlichen Kartenrepräsentation m^k_{feat} zu einem Zeitpunkt k nur die aktiven Features auf.

Feature-basierte Kartierung durch Segment-Merging

Algorithmus 3.10: Feature-basierte Kartierung durch Segment-Merging

Aufruf: **segment-merging()**
Input: Featuremengen G^{k-1} und F, Größenschranken e_{trans}, e_{rot}, $minseg$
Output: Featuremenge G^k

```
1  for { i=1; i< |G^{k-1}|, i++ } do
2     g:= n-th-segment(G^{k-1}, i)
3     changed=FALSE
4     for {j:=i+1 to |F|} do
5        f:=n-th-segment(F, j)
6        angle:=min-angle-between(f, g)
7        dist:=min-distance-between(f, g)
8        if { f erfüllt die Bedingung für das Zusammenfassen} then
9           supersegment=combine-segments(f, g)
10          Markiere das kombinierte Feature.
11          changed=true
12          Füge supersegment an G^{k-1} an.
13       end if
14    end for
15    if {changed==true} then
16       Lösche alle markierten Features.
17       i=0
18    end if
19 end for
20 return G^k = G^{k-1}.
```

Der Algorithmus 3.10 verschmilzt inkrementell Segmentmengen miteinander und löst dabei das Problem 3.30. Deshalb ist dieser Algorithmus, zusammen mit der Korrektur

von Sensordaten nach Algorithmus 3.5, in der Lage die Problemstellung 3.16 zu lösen.

Für die Komplexität ergibt sich für Algorithmus 3.10:

Korollar 3.43 (Komplexität des Mergings) *Es sei* G^{k-1} *eine Feature-Menge mit* n *Features und* F *eine Feature-Menge mit* m *Features. Der Algorithmus 3.10 zur Berechnung von* G^k *besitzt eine Zeitkomplexität von* $\mathcal{O}(n \cdot m^2)$ *und eine Speicherkomplexität von* $\mathcal{O}(n + m)$.

Beweis: Der Aufwand hängt von der Anzahl der noch zu untersuchenden Segmente m aus F und der tatsächlichen Zahl n von Features in G^{k-1} ab. Dabei kann sich die Position eines neuen Supersegments maximal m^2 mal verändern, wobei eine Zeitkomplexität von $\mathcal{O}(n \cdot m^2)$ erreicht wird. Diese Komplexität kann man nochmals verringern, indem man die Segmente in einer Suchstruktur ablegt, welche Bereichsanfragen zulässt.
Es wird Speicherplatz für die Featuremenge G^k und die neue Featuremenge F benötigt, also Speicherplatz aus $\mathcal{O}(n + m)$. □

Der obige Algorithmus lässt sich mit $minseg = 1$ auch zur Reduktion von Segmentmengen einsetzen, die mittels eines Kurvenrekonstruktions-Algorithmus erstellt wurden.

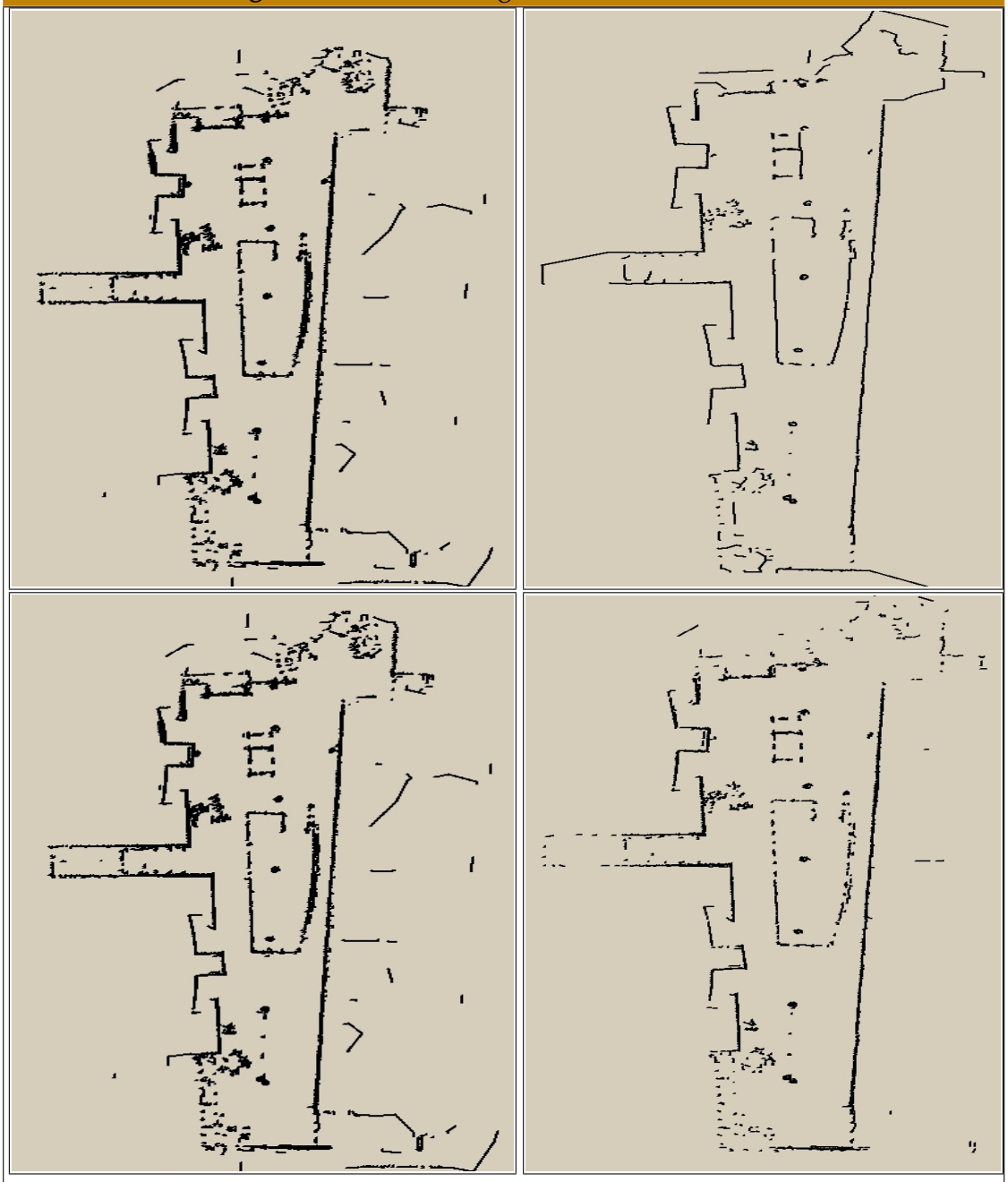
3.5 Experimentelle Evaluation der Erzeugung von Segmentkarten

Als Testumgebung dient hier ein vereinfachter Datensatz aus Abbildung 18. Aus diesem werden mit der Heuristik durch Punktreduktion und dem Verfahren des Segment-Mergings eine Reihe von Instanzen erzeugt. Zum Vergleich werden außerdem die in einer LEDA-Erweiterung verfügbaren Crust-Algorithmen auf den Datensatz angewandt. Dabei beizeichnet ABE-Crust den Crust-Algorithmus [11] und DK-Crust den NN-Crust-Algorithmus [57].

Für jedes genannte Verfahren werden die obigen Heuristiken miteinander kombiniert, wenn es sinnvoll erscheint. Zunächst wird jedes Verfahren für sich ohne Reduktionsmechanismen ausgeführt. Zusätzliche Mechanismen sind die Min-Max-Heuristik nach Algorithmus 3.8, die Merging-Heuristik mit Algorithmus 3.10, sowie die Min-Max-Heuristik kombiniert mit einem vorgeschalteten Reduktionsfilter durch Algorithmus 3.6. Für das Segment-Merging macht die Min-Max-Heuristik keinen Sinn, da extrahierte Segmente höchstens zufällig eine Segmentkette bilden.

Als zusätzliches Vergleichsverfahren wird eine inkrementelle Version des Segment-Mergings verwendet. Dieses kommt aufgrund seiner Effizienz auch in den weiteren Kapiteln zum Einsatz. Eine inkrementelle Version der obigen Heuristik mittels Punktreduktion wurde nicht berücksichtigt.

Abbildung 30: Unterschiedlich generierte Karten mit ABE-Crust



Der ABE-Crust-Algorithmus liefert die für das Auge besten Ergebnisse einer rekonstruierten Einsatzumgebung. Dargestellt sind die vier Heuristiken von links oben nach rechts unten in der beschriebenen Reihenfolge.

Abbildung 31: Unterschiedlich generierte Karten mit DK-Crust

Der DK-Crust-Algorithmus liefert viele falsche Features im freien Raum der Karte. Zudem ist die Anzahl der Features enorm hoch.

Tabelle 1: Laufzeiten der Verfahren zur feature-basierten Kartenerzeugung

Verfahren	Reduktionsverfahren	Extraktionsdauer	Anzahl Segmente	Abbildung
ABE-Crust	-	7.73s	19353	30
ABE-Crust	MinMax	1m 25.24s	2487	30
ABE-Crust	MinMax+Reduktion	28.93s	18905	30
ABE-Crust	Merging	7m 35.32s	3419	30
DK-Crust	-	5.66s	27849	31
DK-Crust	MinMax	1m 25.80s	2989	31
DK-Crust	MinMax+Reduktion	51.00s	27283	31
Dk-Crust	Merging	14m 45.84s	2882	31
Punktreduktion	-	30.79s	761	-
Punktreduktion	MinMax	31.02s	760	-
Punktreduktion	MinMax+Reduktion	1m 26.10s	81	33
Punktreduktion	Merging	31.81s	106	-
Segment-Merging	-	1.03s	2613	-
Segment-Merging	Merging	5.62s	844	34
Iter. Seg.-Mg.	Merging	2.07s	938	32

Abbildung 32: Generierte Instanzen mittels des Segment-Mergings

Zwei mit unterschiedlicher Güte mittels eines inkrementellen Verfahrens des Segment-Mergings erstellte Karten. Die linke Karte besitzt noch alle Details bei Güte 1, die rechte Karte wurde um alle Features reduziert, welche nicht mindestens Güte 9 besitzen.

Mittels der Ergebnisse in Tabelle 1 bestätigt sich die Vermutung, dass die Betrachtung der rohen Messpunkte Performanceverluste bei der Rekonstruktion der Karte mit sich bringt. Die Reduktion der Punktmenge kostet viel zusätzliche Zeit und verbessert die Performance der Heuristik der Punktreduktion nicht. Die aus der Literatur der Kurvenrekonstruktion stammenden Verfahren ABE-Crust und DK-Crust können ebenfalls keine Echtzeit-Kriterien erfüllen. In Kombination mit den Reduktionsverfahren verschlechtert sich deren Laufzeit beträchtlich. Eine Verwendung ohne Reduktion wäre aber aussichtslos, da bei der kleinen Testinstanz mit 145 Scans schon etwa 20000 Segmente generiert werden, was im Hinblick auf den Einsatz featurebasierter Lokalisationsverfahren nicht vertretbar ist. Außerdem muss man feststellen, dass die Punktreduktions-Heuristik sehr parameterabhängig ist. Mit falsch gewählten Parametern reduziert sich die Segmentmenge drastisch, sodass keine verwertbaren Informationen übrigbleiben.

In Abbildung 33 sind mehrere Instanzen für die Punktreduktions-Heuristik mit unterschiedlichen Parametereinstellungen für die Variablen $r_{max} \in [0.1, 0.4]$ und $d_{max} \in [0.1, 0.4]$ gewählt worden. Eine Güte abhängig von der Anzahl zusammengefasster Segmente ist für Kurvenrekonstruktions-Algorithmen sinnlos, da meistens nur zwei Nachbarn eines Segmentes angenommen werden. In Abbildung 34 sind für das Segment-Merging-Verfahren kombiniert mit dem Matching-Verfahren Karten mit Segmenten generiert worden, die mindestens eine entsprechende Güte besitzen. Segmente, welche das Gütekriterium nicht erfüllten, wurden gelöscht. Dabei erkennt man, dass das Gütekriterium von einer detaillierten Sicht bei einer Güte von 1 die Karte sukzessive vergröbert bis zu einer Güte von 9.

Betrachtet man alle mittels der Verfahren extrahierten Karten, dann entsprechen vor allem die Karten des ABE-Crust-Algorithmus und der Punktreduktions-Heuristik dem Detaillierungsgrad, den sich das menschliche Auge wünscht. Das DK-Crust-Verfahren ist nicht geeignet, da es zuviele falsche Segmente in den freien Raum hineinlegt. Das Verfahren des Segment-Mergings liefert auch sehr grobe Ergebnisse. Da jedoch letzteres Verfahren das effizienteste Verfahren ist, werden die anderen Verfahren im weiteren Verlauf nicht mehr beachtet.

Für die Gütebetrachtung, welche Verfahren mit den im vorigen Kapitel vorgestellten feature-basierten Verfahren harmonieren, werden letztendlich nur unterschiedliche Instanzen des Segment-Mergings herangezogen. Lokalisationsergebnisse mit den anderen Verfahren werden nicht im Detail aufgezeigt, doch liefern sie ohne Ausnahme zu viele und zu kleine Segmente für feature-basierte Anfragen. Dadurch sind die Lokalisationsergebnisse vielfach zu ungenau und den resultierenden Karten des Segment-Mergings mit Lokalisationsgüten von höchstens 40 Prozent weit unterlegen (siehe dazu auch Kapitel 4.

3.6 Resümee

In diesem Kapitel wurde das Problem der Korrektur von zweidimensionalen Sensordaten sowie das Erzeugen von feature-basierten Karten bearbeitet.

Dazu wurden zunächst verschiedene Kartenrepräsentationen beschrieben und anschließend die Problemstellungen und ihre offenen Teilprobleme dargestellt. Für die Kartierung wurden die bekanntesten Verfahren aus der Literatur erläutert und anschließend eine effiziente Heuristik implementiert und adaptiert. Diese kann Karten mit kleinen Zyklen in Echtzeit korrigieren was für die meisten Einsatzumgebungen ausreicht. Die erstellte Kartenrepräsentation ist dann eine Gitterkarte, welche mit neuen, verbesserten Bewertungskriterien in der Lage ist, Dynamik im begrenzten Umfang zu repräsentieren. Gitterkarten bieten außerdem den Vorteil, den tatsächlich eingesehenen Raum zu repräsentieren, weshalb sie für die spätere Arbeit unverzichtbar sind.

Aus den korrigierten Sensordaten muss eine weitere Kartenrepräsentation für feature-basierte Lokalisationsverfahren gewonnen werden. Dazu wurden zwei unterschiedliche Ansätze der Literatur verfolgt und für jeden ein Ansatz implementiert. Der Ansatz der Kurven-Rekonstruktion liefert augenscheinlich die besten Karten, kann aber nicht die benötigte Performance bieten. Das Verfahren der Segmentverschmelzung ist in einer iterativen Version in der Lage, parallel zur Gitterkartenrepräsentation eine feature-basierte Repräsentation in Echtzeit aufzubauen.

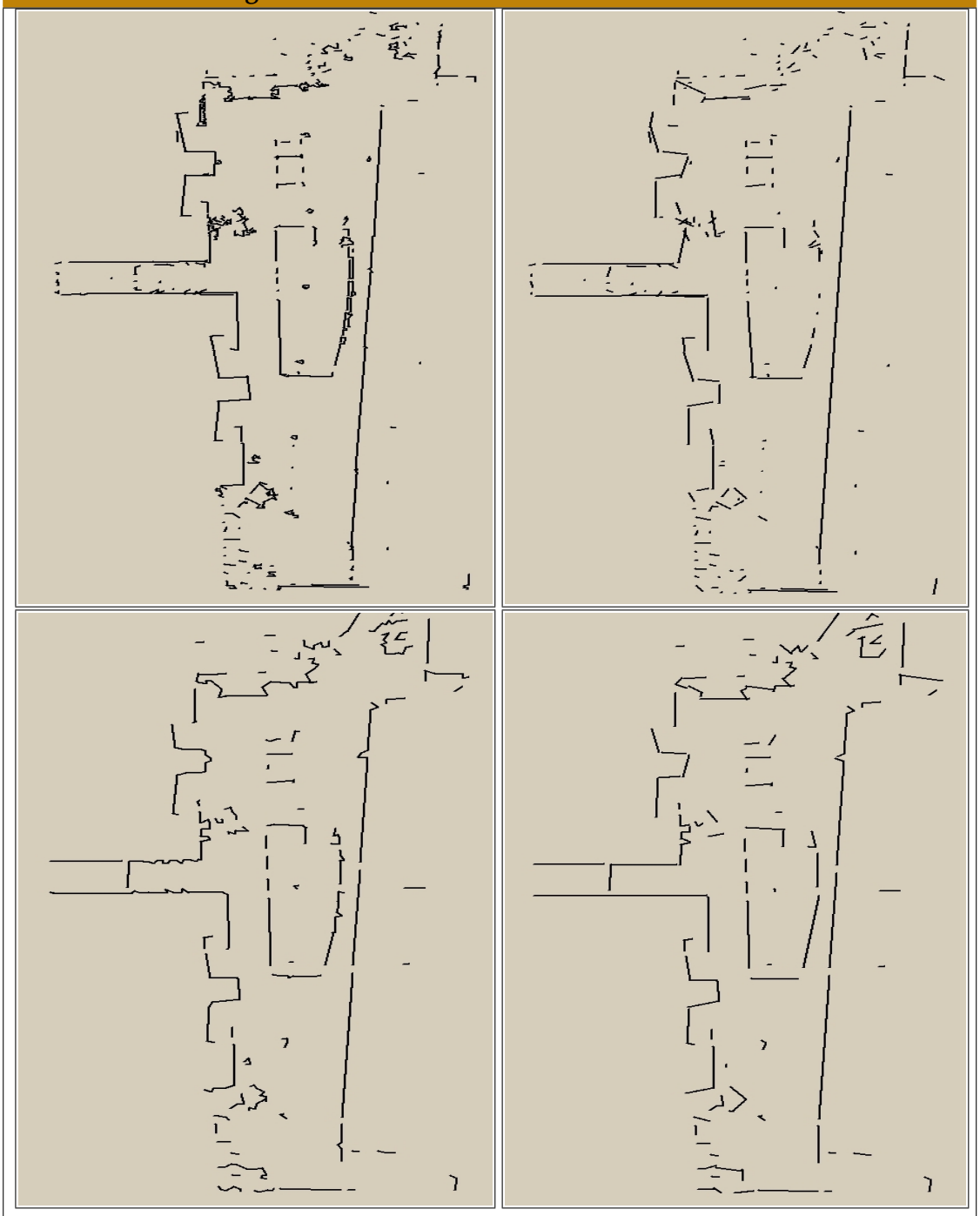
Dabei kann man mittels des letzten Verfahrens eine Reihe von Karten unterschiedlicher Güte erzeugen. Welche Instanz tatsächlich für feature-basierte Verfahren ideal ist, wird im nächsten Kapitel beschrieben.

Bei der Bearbeitung der Probleme wurden viele Ideen unberücksichtigt gelassen. So ist für die Lösung der Zykel-Problematik eine Erweiterung der dynamischen Gitterkarte in eine Menge von Teilkarten möglich. Diese Teilkarten müssen sich gegenseitig überlappen. Damit bekommt man eine Generalisierung der Idee von Lu, indem man Teilkarten anstatt Scans für einen Algorithmus zur konsistenten Positionsschätzung verwendet und deren initiale Position iterativ korrigiert.

Weiter kann man die Idee der inkrementellen Kurvenrekonstruktion aufnehmen und einen zu entwickelnden Algorithmus mit den vorhandenen Algorithmen vergleichen.

Der feature-basierte Kartierungsansatz berücksichtigt nicht die repräsentierte Dynamik in der Gitterkarte. Eine Integration von Tracking-Verfahren in die Kartierungsproblematik könnte dynamische Hindernisse in der Feature-Kartenrepräsentation eliminieren und die Möglichkeit schaffen, nur dynamische Objekte in einer Gitterkarte modellieren zu müssen.

Abbildung 33: Generierte Instanzen mittels der Punktreduktion



Mittels unterschiedlicher Parametervariationen der Punktreduktions-Heuristik erstellte Karten.

Abbildung 34: Generierte Instanzen mittels des Segment-Mergings

Mittels unterschiedlicher Gütevariationen des Segment-Merging-Algorithmus erstellte Karten. Die Güte variiert von 1 (links oben), über Güte 3, Güte 5 und Güte 9 (rechts unten).

4 Methoden zur Effizienzsteigerung

Die vorgestellten feature-basierten Verfahren aus Kapitel 2 haben den Nachteil, zu zeit- und speicherplatzintensiv zu sein, was die Tests in Abschnitt 2.8.2 zeigen. Erste Ergebnisse mit dem Alignment-Verfahren verschärften den Eindruck, der bei den Tests mit künstlichen Karten schon gewonnen werden konnte: In generierten Karten realistischer Umgebungen ist die Featuremenge erheblich größer als in synthetischen Umgebungen und damit der Berechnungsaufwand unrealistisch hoch. Eine Untersuchung mit speicherintensiveren Verfahren würde also schon im Ansatz scheitern.

In diesem Kapitel sollen deshalb Möglichkeiten untersucht und entwickelt werden, wie man die Effizienz der Verfahren spürbar steigern kann. Folgende Ziele sollen dabei verfolgt werden:

Geschwindigkeit
: Eine möglichst konstante und akzeptable Laufzeit nahe der Echtzeitfähigkeit soll für feature-basierte Verfahren erreicht werden.

Speicherverbrauch
: Der Speicherverbrauch für die Verfahren sollte im erträglichen Rahmen bleiben. Lokalisationsanfragen sollten nicht aufgrund von Speicherplatzproblemen abbrechen.

Lokalisationsgüte
: Die Güte der Verfahren sollte in akzeptablen Bereichen bleiben, eine hohe Geschwindigkeit soll auf Kosten eines geringen Güteverlusts in Kauf genommen werden.

Zunächst sollen dazu Methoden vorgestellt werden, die Geschwindigkeit zu erhöhen und den Speicherbedarf zu senken. Anschließend soll die Güte der Lokalisationsanfrage durch die Kombination der vorgestellten Verfahren verbessert werden. Eine abschließende Parameterstudie zeigt dann den Erfolg der erarbeiteten Methoden.

4.1 Reduktion der Bild- und Modellfeatures

Für eine feature-basierte Anfrage spielt die Anzahl der verwendeten Features eine große Rolle - bestimmt sie doch (wie in Kapitel 2 gezeigt) die Zeit- und Speicherkomplexität. Eine Reduktion sowohl von Modell- als auch von Bildfeatures auf eine

konstante Menge würde die Komplexität der verwendeten Algorithmen auf ein vorhersagbares und erträgliches Maß reduzieren.

In der Literatur finden sich vielfältige Beiträge zur Verbesserung der Performance. Viele verwenden ihre Mühe darauf, die Matchingqualität zu verbessern [168], den Fehler beim Matching entsprechend genau abzuschätzen [85, 227] oder aber falsche Hypothesen zu entdecken [164].

Einen der wichtigsten Beiträge liefern Bolles und Chain mit der Klassifizierung von Features durch ihre Local-Feature-Focus-Methode [24], was nichts anderes bedeutet als die Einteilung der Features in wichtige und unwichtige oder beschreibenende und weniger beschreibende Features. Dabei werden die besonders ausgezeichneten Features mit speziellen Attributen beschrieben und nach den ausgezeichneten Features in der Featuremenge gesucht. Dieser Ansatz wird in den meisten heute verwendeten Vision-Systemen in Kombination mit einer Objektdatenbank verfolgt, in welcher beschreibende Features des Objekts spezifiziert sind [168].

Einen weiteren Beitrag im Bereich der Robotik liefert Jensfeld [113] durch die Verwendung mehrerer Feature-Typen, darunter auch der Feature-Typ einer Tür. Es werden also spezielle Konstellationen von Features gesucht, welche für das Matching eine enorme Zeitersparnis liefern. Dabei wird explizit vorausgesetzt, dass Türen in der Umgebung existieren müssen.

Wichtige, beschreibende Features liefern die Idee, welche diesen Abschnitt beherrschen wird: Die Suche und das Auffinden von informativen Features, jedoch abweichend von den obigen Ansätzen ausschließlich aus der Menge der Segmentfeatures. Dabei wird bewusst in Kauf genommen, dass ein Geschwindigkeitsgewinn zu einem Verlust der Genauigkeit der berechneten Matchings führt.

4.1.1 Bewertung informativer Features

Hier soll der Ansatz von Bolles und Chain aufgegriffen werden, informative Features zu finden. Dazu muss zunächst geklärt werden, was hier unter einem informativen Features verstanden wird:

Definition 4.1 (Informatives Feature, Redundantes Feature) *Es seien zwei Features* f_1 *und* f_2 *gegeben. Das Feature* f_1 *heißt informativer als* f_2*, falls eine Lokalisationsanfrage, in welcher die Featuremenge* $F \cup f_1$ *mit* $(F \cup f_1) \cap f_2 = \emptyset$ *eingesetzt wird, eine bessere Bewertung liefert als eine Lokalisationsanfrage mit* $F \cup f_2$ *mit* $(F \cup f_2) \cap f_1 = \emptyset$ *bei gleicher Modellfeature-Menge.* f_1 *heißt dann Informatives Feature. Falls sich die Lokalisationsgüte mit einem Feature* f_2 *nicht verbessert, heißt* f_2 *redundantes Feature.*

Leider sieht man einem Feature vor einer Lokalisationanfrage nicht an, ob es informativer oder weniger informativ als ein anderes Feature ist. Was man jedoch konsequenterweise verwenden kann, ist eine Bewertung einzelner Features.

Definition 4.2 (Bewertung eines Features) *Ein Feature* $f \in F$ *kann durch eine Abbildung* $g : F \to \mathbb{R}$ *bewertet werden.*

Gesucht ist dann eine Bewertung aller Features einer Featuremenge und ein Auswahlkriterium, wie man aus der bewerteten Featuremenge eine informative Featuremenge auswählt. Folgende Bewertungen für Segmentfeatures sind vorstellbar:

Länge des Features
: Eine triviale Bewertungsfunktion, die aber Sinn macht. Können doch einzelne vorkommende Längen in der Featuremenge sehr aussagekräftig und damit informativ sein.

Extraktionsfehler
: Wahl eines Vertrauenswertes aus der Extraktion der Features. Schlecht extrahierte Features sind weniger vertrauenswürdig als sehr gut extrahierte Features.

Abhängig von diesen Kriterien können die Features der Bewertung entsprechend sortiert und dann durch ein Auswahlkriterium ausgewählt werden. Betrachtet man direkt aus einem Scan extrahierte Segmentfeatures (und damit Bildfeatures), kann man den Extraktionsfehler bei der Berechnung zur Bewertung heranziehen (siehe Kapitel 2.2). Natürlich sind auch andere Bewertungen denkbar, hier aber nicht betrachtet worden.

4.1.2 Auswahl informativer Features

Eine Bewertung der Features alleine ist nicht ausreichend für eine Reduktion der Featuremenge. Ein Auswahlkriterium liefert erst die gewünschte reduzierte Featuremenge. Als Auswahlkriterien werden die drei Möglichkeiten angeboten:

k-besten Auswahl
: Aus der Menge der betrachteten Features werden die k besten Features ausgewählt, welche die Bewertungsfunktion g liefert.

Probabilistische Auswahl
: Aus der Menge der Features wird eine konstante Menge von n Features zufällig ausgewählt. Eine Bewertungsfunktion spielt keine Rolle.

Histogrammreduktion
: Die Menge der bewerteten Features wird abhängig von der Bewertungsfunktion g durch eine Histogrammbildung über den Wertebereich von g klassifiziert und die im Histogramm am seltensten vorkommenden Features ausgewählt.

Dabei muss man bei der Wahl des Auswahlverfahrens darauf achten, dass mit der Auswahl der Features auch eine konsistente Wahl des Matchingverfahrens einhergeht. So ist das Geometrische Hashing darauf spezialisiert, dass Features sich gegenseitig sehen müssen, um als Basis in Frage zu kommen. Bei ungeschickter Auswahl der Features wird so viel Rechenzeit für ein absehbar schlechtes Ergebnis verschwendet

(siehe dazu auch die praktischen Ergebnisse in Abschnitt 4.3).

Jedes unten beschriebene Auswahlverfahren erreicht, dass eine Menge $F_{red} \subset F$ generiert wird mit $| F_{red} | = k$. Die Verfahren erreichen also eine Reduktion auf eine konstante Anzahl von Features. Dabei spielt es keine Rolle, ob es sich dabei um Modellfeatures oder um Bildfeatures handelt.

k-besten-Auswahl

Eine Auswahl der besten Features macht nicht immer Sinn und ist auch abhängig von der Bewertungsfunktion. Verwendet man nur die besten Features, können diese die am häufigsten vorkommenden Features sein, sodass ein Matching mit diesen Features nur unnötige Rechenzeit vergeudet, wenn unzählige Hypothesen erzeugt werden.

Vielmehr ist es erforderlich, Bewertungen zu suchen, welche spezielle Eigenschaften der Features am besten bewerten. Da die Suche nach Bewertungen meist schwierig und umgebungsabhängig ist, wird die k-besten-Auswahl mit den anderen Auswahlverfahren verknüpft.

Korollar 4.3 (Aufwand der k-besten-Auswahl) *Der Algorithmus benötigt $\mathcal{O}(n)$ Speicherplätze bei n Features der betrachteten Featuremenge und hat eine Zeitkomplexität von $\mathcal{O}(n \log n + k)$, welche vom Sortieraufwand des Sortierverfahrens abhängig ist.*

Probabilistische Auswahl

Die Anwendung probabilistischer Verfahren bei der Suche in großen Datenbeständen ist oft erfolgversprechend. Genauso kann man dieses Paradigma auf diese Problematik übertragen. Man wählt aus der Featuremenge eine Anzahl von k Features randomisiert aus und versucht diese Teilmenge gegen die andere Featuremenge zu matchen (siehe Algorithmus 4.1). Im Idealfall bekommt man randomisiert eine Menge Features geliefert, welche exakt die Featuremenge darstellt, die für die tatsächliche Hypothese erforderlich war. In der Regel ist der Erfolg beim Einsatz probabilistischer Methoden eingeschränkt, wenn die Featuremengen größere Karten repräsentieren.

Korollar 4.4 (Aufwand der probabilistischen Reduktion) *Der Algorithmus benötigt abhängig von der Anzahl der n Features der Featuremenge $\mathcal{O}(n)$ Speicherplätze und hat eine Zeitkomplexität von $\mathcal{O}(k)$.*

Histogrammreduktion

Die obigen beiden Verfahren wählen abhängig von der Bewertungsfunktion entweder irgendwelche Features aus, oder aber die besten k einer Featuremenge F. Dabei ist bisher aber die Struktur der Bewertungsfunktion nicht berücksichtigt worden. So kann es, wie oben schon angedeutet, sinnlos sein die besten Features zu wählen, denn man

Algorithmus 4.1: Probabilistische Auswahl

Aufruf: **probabilistic-selection()**
Input: Menge von Features F, Anzahl gewünschter Features k
Output: Menge von Features F_{red} mit $| F_{red} | = k$

```
1 F_red = ∅
2 while {| F_red | < k ∧ F ≠ ∅} do
3    Bestimme zufällig ein Feature f_i ∈ F.
4    Füge f_i der Menge F_red hinzu.
5    Lösche f_i aus F.
6 end while
```

hätte sich nur die Bewertung der Features anschauen müssen um dann festzustellen, dass man mit einem Feature, was eine sehr seltene Bewertung hat, einen Volltreffer sicher hat.

Die Histogrammreduktion versucht, dieses Manko der obigen Auswahlverfahren zu beheben. Die Bewertungsfunktion g wird demnach in verschiedene Teilmengen klassifiziert, abhängig von der Größe des Histogramms und dem Wertebereich der Bewertungsfunktion. Je mehr Einträge das Histogramm in einem Feld besitzt, desdo öfter kommt dieser Wert vor. Je geringer die Histogrammeinträge, desdo besser kann die Eindeutigkeit eines Features erkannt werden. Somit werden die Features alsdann nach ihrer Histogrammbewertung aufsteigend sortiert und die besten k Features und damit seltensten Features ausgewählt.

Als triviale Bewertungsfunktion dient im weiteren die Länge eines Segmentes als Bewertung g. Die Komplexität des Algorithmus ergibt sich folgendermaßen:

Korollar 4.5 (Aufwand der Histogram-Reduktion) *Der Algorithmus 4.2, operierend auf einer Featuremenge F mit n Features benötigt $\mathcal{O}(n + l)$ Speicherplätze und hat eine Zeitkomplexität von $\mathcal{O}(n \log n)$. Dabei bezeichnet l die Größe des Histogramms.*

Beweis: Alle n Features werden entsprechend ihres Wertes ins Histogramm abgelegt. Da in jedem Eintrag des Histogramms nur ein Wert für ein Feature erhöht wird, benötigt man für jedes Feature noch den zugehörigen Histogrammplatz. Nach der Bewertung wird für jedes Feature der Histogrammwert bestimmt, d.h. n Operationen sind nötig und n zusätzliche Speicherplätze. Die Features werden anschließend entsprechend dem Histogrammwert sortiert, so dass sich der Gesamtaufwand zu $\mathcal{O}(n \log n)$ ergibt mit $\mathcal{O}(n + l)$ benötigten Speicherplätzen. □

4.1.3 Segmentierung der Featuremenge

Was durch die Reduktion der Bild- und Modellfeatures erreicht wird ist eine starke Informationsreduktion, gleichgültig wie komplex die dahinterstehenden Objekte sind. Dies mag für geringe Objekte, oder besser, kleine Umgebungskarten ausreichend sein,

Algorithmus 4.2: Histogrammreduktion

Aufruf: **histogram-seletion()**
Input: Menge von Features F, Anzahl gewünschter Features k
Output: Menge von Features F_{red} mit $| F_{red} | = k$

```
1  F_red = ∅
2  Definiere ein Histogramm mit l Werten.
3  Berechne die Segmentlänge für einen Histogrammeintrag.
4  for all {f ∈ F} do
5     Bestimme den Eintrag des Histogramms abhängig von der Segmentlänge.
6     Erhöhe den Zähler im Histogrammfeld.
7  end for
8  for all {f ∈ F} do
9     Bewerte alle Segmente mit dem entsprechenden Histogrammscore.
10 end for
11 Sortiere die Bewertung der Segmente aufsteigend.
12 while {| F_red |< k ∧ F ≠ ∅} do
13    Hole erstes Element f_i aus der sortierten Folge.
14    Füge f_i der Menge F_red hinzu.
15    Lösche f_i aus F.
16 end while
```

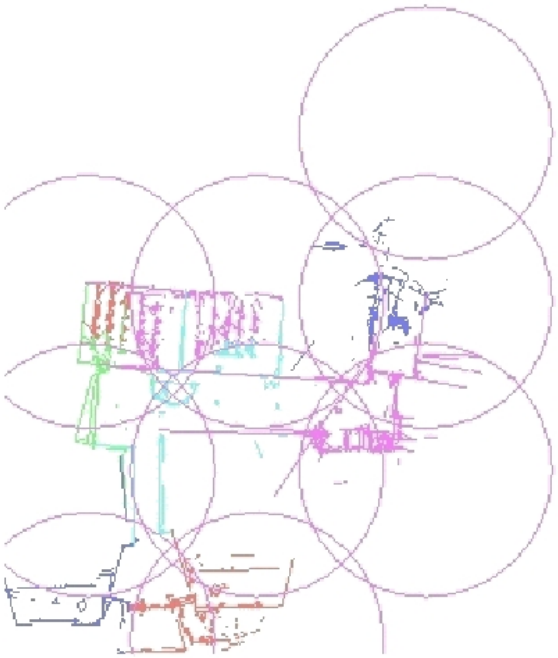
für große Karten muss man andere Strategien anwenden.

Eine Idee, kommend aus der Kartierungsliteratur, ist die Segmentierung einer Karte in eine Hierarchie von Karten [113, 177]. Jedoch soll hier ein Konzept für allgemeine Featuremengen angewandt werden, so dass sowohl Bildfeatures- als auch Modellfeatures segmentiert werden können. Weiter kann man die Beobachtung machen, dass die grundlegenden Matchingverfahren Alignment, Pose-Clustering und Geometrisches Hashing immer nur eine kleine Region der zu matchenden Objekte betrachten und dann eine Hypothese erzeugen, die in einem späteren Schritt verifiziert wird. Man kann sich überlegen, dass ein Scan ebenfalls eine lokal-beschränkte Bildfeature-Menge ist. Somit ergibt sich das Verfahren automatisch: Wenn man sicherstellt, dass man Modellfeatures einer Region für eine Anfrage extrahiert, kann man auf die bekannten Matchingverfahren zurückgreifen.

Eine Aufteilung einer Karte m in Teilkarten ist also erforderlich. Dies macht zusätzlich Sinn, kann durch die Erzeugung von Segmentkarten aus dem vorherigen Kapitel eine Repräsentation in lokale Teilkarten die Merging-Operationen verbessern. Eine Teilkarte kann folgedermaßen definiert werden.

Definition 4.6 (Teilkarte einer Segmentkarte) *Eine Teilkarte m^{ij} ist eine Karte, die aus einer Teilmenge der Featuremenge einer Karte m besteht. Der Index ij, ein Kreisdurchmesser d und eine Anzahl n von Features bestimmen eine Teilkarte wie folgt:*

Abbildung 35: Homogene Segmentierung einer Karte

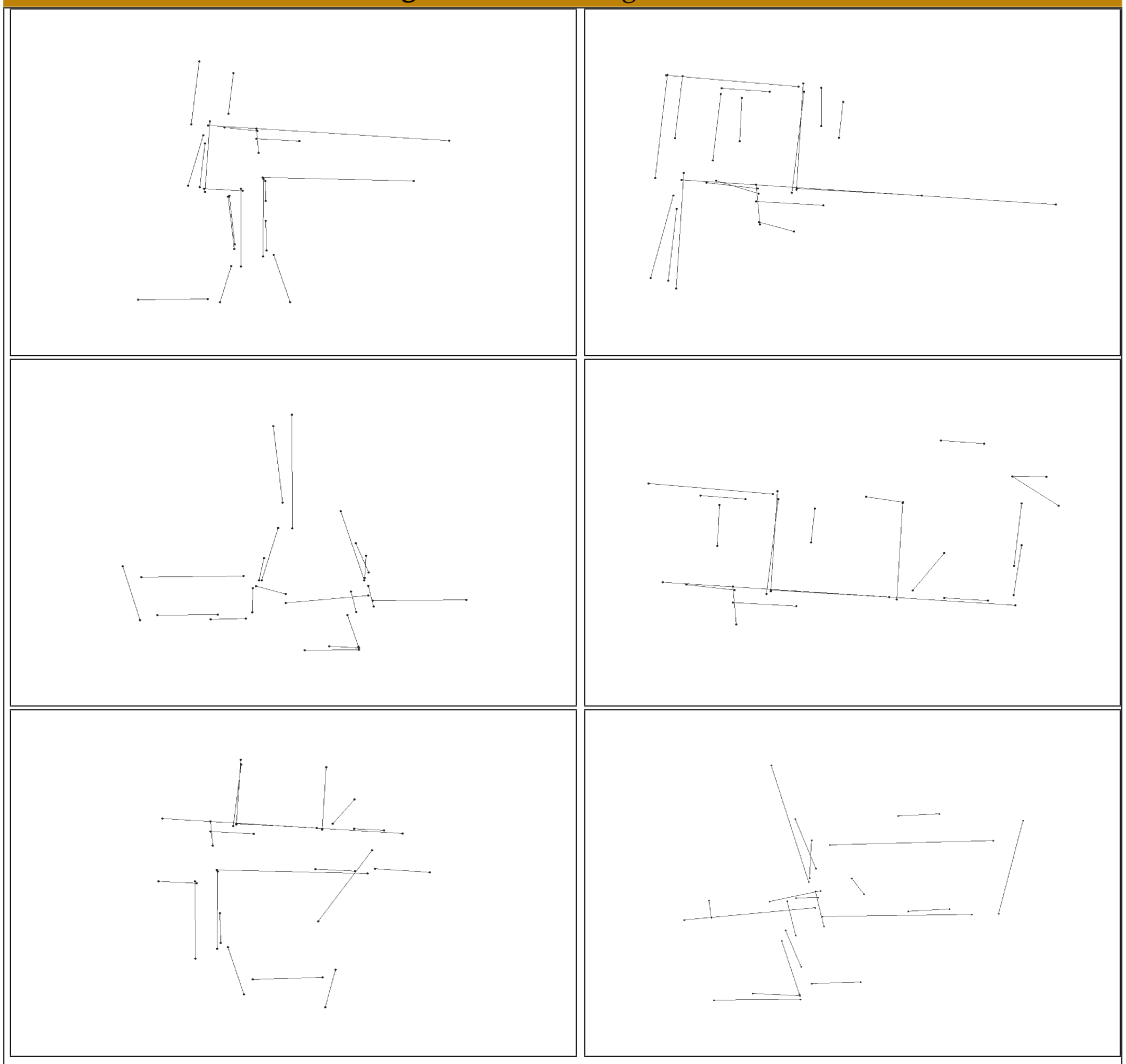


Segmentierung der Hanggeschosskarte des Informatikgebäudes in 9 einzelne Teilkarten, welche jeweils durch einen Kreis dargestellt ist. Die Gesamtkarte hat insgesamt 2224 Segment-Features.

Es sei B *das die Karte* m *umgebende Reckteck mit den Startkoordinaten* (B_x, B_y) *und der Größe* Δ_x *in* X*-Richtung und* Δ_y *in* Y*-Richtung. Die Karte* m *wird in ein gleichmäßiges Gitter mit* $n_x = \frac{\Delta_x}{d}$ *Spalten und* $n_y = \frac{\Delta_y}{d}$ *Zeilen aufgeteilt. Dabei ist* d *der Abstand zweier aufeinanderfolgender Zeilen oder Spalten.*
Eine Teilkarte m^{ij} *mit* $0 \leq i \leq n_x$ *und* $0 \leq j \leq n_y$ *mit* $i, j \in \mathbb{N}$ *besteht aus allen Segmentfeatures, die vom Kreis mit dem Durchmesser* $\overline{d} = \epsilon \cdot d$ *und dem Mittelpunkt* (x_i, y_i) *berührt werden, wobei* $x_i = B_x + i \cdot d + \frac{1}{2}d$ *und* $y_i = B_y + j \cdot d + \frac{1}{2}d$. *Der Parameter* $\epsilon \geq 0$ *ist ein Maß für die Überlappung zweier benachbarter Kartenteile.*
Für jede Teilkarte werden n *informative Features gemäß einem der oben beschriebenen Verfahren zur Featureauswahl ausgewählt. Dabei besitzt jede Teilkarte mindestens* $n_{min} < n$ *Features.*

Das Konzept der überlappenden Kartenbereiche ist wichtig, da sich der Roboter auch in Grenzbereichen, abhängig von der Aufteilung von m in Teilkarten, befinden kann. Wenn der Roboter in Richtung des benachbarten Kartenbereiches ausgerichtet ist und in diesem Kartenbereich keine erfolgreiche Anfrage machen kann, dann kann er dafür (durch die Überlappung) im benachbarten Kartenbereich diese Anfrage erfolgreich durchführen, da dort die entsprechenden Features zu finden sind.

Abbildung 36: Featuremengen der Teilkarten



Ausgewählte Teilkartendarstellung der Hanggeschosskarte mit der jeweiligen Feature-Menge eines Bereichs.

Die Wahl der Überlappung ϵ und die Einstellung des Kreisdurchmessers d sind wichtige heuristisch-einstellbare Parameter, die von der Beschaffenheit der Karte abhängen. Wählt man d zu klein, gibt es pro Kartenteil eventuell zu wenig Features, so dass dieser Kartenteil aufgrund einer unteren Schranke für die Mindestzahl Features n_{min} nicht berücksichtigt wird. Wählt man d zu groß, kann die Featuremenge pro Kartenteil zu groß werden. Die Abbildung 35 zeigt eine Karte mit einer Segmentierung in Teilkarten. Die Repräsentationen der einzelnen Teilkarten sind in Abbildung 36 dargestellt.

Ein ähnliches Problem tritt auf, wenn die in der Karte vorkommenden Features gehäuft in einigen kleinen Teilbereichen auftreten. In anderen Bereichen wiederum gibt es fast überhaupt keine Features. Dann ist es sinnvoll, die Segmentierung der Karte adaptiv zu gestalten, abhängig von einem entsprechenden Qualitätsmaß, wieviele Features in einer Teilkarte vorhanden sein sollen. Diese Problematik wird im weiteren nicht weiter verfolgt.

Ein weiteres nicht zu vernachlässigendes Problem ist die Redundanz, die durch längere Features auftritt, welche sich über mehrere Kartenbereiche erstrecken. Diese Features tauchen demzufolge in allen beteiligten Kartenbereichen auf. Wählt man den Überlappungsbereich zu groß und d zu klein, erreicht man damit im schlimmsten Fall eine Vervielfachung der gesamten Featuremenge.

Algorithmus 4.3: Segmentierung der Karte

Aufruf: **map-decomposition()**
Input: Karte m mit Featuremenge F, Reichweite d, Überlappung ϵ
Output: Menge von k Kartenteilen m^{ij}

```
1  Bestimme die Anzahl der Kartenteile abhängig von d und ε.
2  Berechne die Mittelpunkte (x_i, x_j) der Kartenteile m^ij.
3  for all {k Mittelpunkte} do
4     for all {Features f_i in F do
5        Bestimme die minimale Distanz dist von f_i zum Mittelpunkt von m^ij.
6        if {dist <= d̄} then
7           Füge f_i an Kartenteil m^ij an.
8        end if
9     end for
10 end for
11 Lösche Kartenbereiche mit n < n_min.
```

Korollar 4.7 (Komplexität des Zerlegungsalgorithmus) *Die Zerlegung einer Karte m mit n Features in eine Menge von k Teilkarten nach Algorithmus 4.3 benötigt $\mathcal{O}(k \cdot n)$ Zeitschritte und $\mathcal{O}(k \cdot n)$ Speicherplätze.*

Beweis: Zunächst muss die Karte in mehrere Bereiche unterteilt werden. Dies erfordert die Berechnung der Mittelpunkte der neuen Karten und benötigt bei k Teilkarten $\mathcal{O}(k)$ Operationen. Für jedes der n Features muss anschließend geprüft werden, ob es innerhalb der Teilkarte ist. Dazu wird die minimale Distanz des Features zum Mittelpunkt der jeweiligen Teilkarte berechnet, was in konstanter Zeit geschehen kann. Somit ergibt sich für ein Feature ein Aufwand von $\mathcal{O}(k)$ Operationen. Abschließend benötigt man k Schritte um zu testen, ob die Karten gültig sind. Insgesamt benötigt man also $\mathcal{O}(kn)$ Operationen.

Für die Speicherung der n Features werden im schlechtesten Fall alle Features dupliziert, so dass man auf einen Speicheraufwand von $\mathcal{O}(kn)$ kommt. □

Die Wahl der Parameter, deren Eigenschaften oben schon diskutiert wurden, wird experimentell festgelegt. Siehe dazu Abschnitt 4.3.

4.2 Verfahren zur Effizienzsteigerung

Wurde im letzten Abschnitt geklärt, wie Feature-Mengen reduziert werden können, soll sich dieser Abschnitt mit dem Einsatz dieser Verfahren in einem Lokalisationsverfahren befassen. Dazu wird ein schnelles, feature-basiertes Lokalisationsverfahren vorgestellt. Zum anderen wird der Einsatz eines kombinierten Verfahrens angedacht, welches die vorgestellten Matchingverfahren kombiniert. Beide Ansätze sollen dazu dienen den Einsatz solcher Verfahren in der Realität sicherzustellen und zu verbessern.

4.2.1 Schnelle feature-basierte Verfahren

Der traditionelle Ansatz, mit einer Bildfeature-Menge und einer Modellfeature-Menge eine Lokalisationsanfrage anzustoßen, welche ein in Kapitel 2.3 vorgestelltes Verfahren einsetzt, soll nun revidiert werden.

Durch die obigen Verfahren werden die Bildfeatures F in k Teilmengen zerlegt, so dass $F = F_1 \cup F_2 \cup \cdots \cup F_k$. Ebenso werden die Modellfeatures G in l Teilmengen zerlegt, $G = G_1 \cup \cdots \cup G_l$. Da die Matchingverfahren die zur Verfügung stehenden einzelnen Features verwenden um ein Matching zu bestimmen, braucht man die Kardinalitäten der einzelnen miteinander zu matchenden Mengen nicht zu beachten. Man muss lediglich sicherstellen, dass eine minimale Anzahl Features für ein Matching zur Verfügung steht.

Durch diese Zerlegung der Mengen F und G muss man nun, um alle hypothetischen Zuordnungen zu bekommen, jede Bildfeature-Teilmenge mit jeder Modellfeature-Teilmenge matchen. Eine Zerlegung erkauft man sich mit einer Menge von $k \cdot l$ Lokalisationsanfragen.

Nun kann man sich noch weitere Heuristiken überlegen, diese Menge von Lokalisationanfragen einzuschränken. Folgende Möglichkeiten kommen dabei in Betracht:

Konstante Iterationszahl
Die Menge der Anfragen kann durch eine konstante Zahl I_{max} begrenzt werden. Dann stellt sich die Wahl, eine „informative Teilkarte" auszuwählen.

Einschränkung der Hypothesenmenge
Berechnete Hypothesen bekommen in der Regel mit der Verifikation eine Gütebewertung mitgeliefert. Man kann nun eine maximale Anzahl H_{max} an Hypothesen festlegen, die erreicht werden darf. Da eine Anfrage (im ursprünglichen

Sinne) mit Sensordaten eine ungenaue Hypothesenmenge liefert und nicht garantiert werden kann, dass wirklich alle Hypthesen generiert wurden, kann man diese Tatsache hier positiv nutzen. Somit können alle weiteren Lokalisationsanfragen verworfen werden. Die Einschränkung der Hypothesenmenge stellt damit ein Pruningkriterium für die Menge der Anfragen dar.

Wahl der Anfrage-Reihenfolge
Besitzt man eventuelles Vorwissen, wo sich der Roboter in der Regel aufhält, oder wo sich der Roboter befinden müsste, kann man die Anfragereihenfolge der Teilkarten entsprechend festlegen. Diese kann man fest vorgeben oder aber durch eine probabilistische Auswahl festlegen.

Der mit den obigen Eigenschaften ausgestattete Gesamtalgorithmus 4.4 wird in den folgenden experimentellen Studien ausführlich getestet. Die Komplexität des Algorithmus wird ohne die zusätzlichen Abbruchbedingungen bestimmt. Es wird davon ausgegangen, dass alle möglichen Teilmengen-Anfragen durchgeführt werden.

Algorithmus 4.4: Schnelle iterative Lokalisation

Aufruf: **fast-localization()**
Input: Bildfeatures F, Modellfeatures G
Input: Maximale Hypothesenanzahl H_{max}, Maximale Iterationszahl I_{max}
Output: Hypothesenmenge H

```
1  Zerlege die Modellfeatures G in k Teilmengen.
2  Zerlege die Bildfeatures F in l Teilmengen.
3  Lege eine Iterationsreihenfolge der Teilkarten fest.
4  CANCEL:=true; counter:=0;
5  while {CANCEL} do
6     counter++
7     Wähle Bildfeature-Menge F_i aus, berechne informative Feature-Menge F_i_quer.
8     Wähle Modellfeature-Menge G_j aus, berechne informative Feature-Menge G_j_quer.
9     Generiere eine Lokalisationsanfrage H_ij = localize(F_i_quer, G_j_quer)
10    H = H ∪ H_ij
11    if {| H |> H_max} then
12       CANCEL:=false
13    end if
14    if { counter > I_max } then
15       CANCEL:= false
16    end if
17    if {counter > k · l} then
18       CANCEL:=false
19    end if
20 end while
21 return H
```

Korollar 4.8 (Komplexität des Matchings der iterativen Lokalisation) *Das Verfahren 4.4 zur schnellen feature-basierten Lokalisation unter Einsatz von Alignment als Matching-Verfahren zwischen einer Bildfeature-Menge* F *und einer Modellfeature-Menge* G *hat eine Zeitkomplexität von* $\mathcal{O}(k \cdot l \cdot p^{2+3/4} \cdot q^3)$. *Dabei bezeichnen* p *und* q *die Konstanten, auf welche die Bild- und Modellfeatures reduziert wurden.* k *und* l *repräsentieren die Anzahl der Teilmengen von* F *und* G. *Die Speicherkomplexität ergibt sich zu* $\mathcal{O}(k \cdot l \cdot p^2 \cdot q^2)$.

Beweis: Das Alignment-Verfahren hat eine Gesamtkomplexität von $\mathcal{O}(n^{2+3/4} \cdot m^3)$. Die Menge F wird in k Teilmengen segmentiert, die Menge G in l Teilmengen. Dann ergeben sich bei einer optimalen Zerlegung $k \cdot l$ Lokalisationsanfragen mit jeweiliger Anfragekomplexität aus $\mathcal{O}((\frac{n}{k})^{2+3/4} \cdot (\frac{m}{l})^3)$. Wendet man jetzt Methoden zur Reduktion der jeweiligen Featuremengen an und betrachtet man pro Bildfeature-Menge eine konstante Auswahl von p Features und analog für Modellfeature-Mengen eine konstante Auswahl von q Features, so ergibt sich der Aufwand zu $\mathcal{O}(k \cdot l \cdot p^{2+3/4} \cdot q^3)$.

Analog benötigt man Speicherplatz für Hypothesen, erzeugt aus $k \cdot l$ Lokalisationsanfragen. Pro Anfrage können $\mathcal{O}(p^2 \cdot q^2)$ Hypothesen erzeugt werden, also bei $k \cdot l$ Verfahren ergibt sich die Speicherkomplexität zu $\mathcal{O}(k \cdot l \cdot p^2 \cdot q^2)$. □

4.2.2 Kombination feature-basierten Verfahren

Im Laufe dieser Arbeit sind verschiedene feature-basierte Matchingverfahren vorgestellt worden. Es liegt auf der Hand, diese Verfahren gleichzeitig für Lokalisationsverfahren einzusetzen. Zum einen besteht die Möglichkeit, dass sich die Verfahren gegenseitig ergänzen, zum anderen wird durch redundante Systeme die Fehleranfälligkeit des Gesamtverfahrens verringert.

Algorithmus 4.5: Kombinierte feature-basierte Lokalisation

Aufruf: **comined-localization()**
Input: Menge von Bildfeatures F und Modellfeatures G, l Lokalisationsverfahren
Output: Menge von Hypothesen H^k

```
1 for {i=0; i < l; i++} do
2    H_i^k = localize(i, F, G)
3 end for
4 Verschmelze die Hypothesenmengen H_i^k geeignet zur Hypothesenmenge H^k.
5 return H^k
```

Die Kombination von l verschiedenen Lokalisationsverfahren summiert deren Zeit- und Speicherkomplexität entsprechend auf.

Einzige Problematik, die bei Algorithmus 4.5 zu betrachten ist, ist das Zusammenfassen der generierten Lokalisationshypothesen aus den einzelnen Verfahren zu einer Gesamthypothesenmenge. Hierbei kann man mehrere Möglichkeiten betrachten:

Einfaches Zusammenfassen
Die Hypothesen werden so zusammengefasst, dass eine Hypothese etwa 0,25 m abdeckt. Hypothesen, die eine geringere Distanz haben, werden zu einer Hypothese mit gemeinsamen Schwerpunkt zusammengefasst. Diese Zusammenfassung ist jedoch reihenfolgeabhängig.

Gewichtetes Verschmelzen
Die Hypothesen werden nach der ersten Methde verschmolzen, jedoch erhält jede Hypothese noch eine Bewertung durch das erzeugende Lokalisationsverfahren.

Best-Score Verschmelzen
Eine Hypothese wird nur dann akzeptiert, wenn die Mehrzahl der eingesetzen Lokalisationsverfahren diese Hypothese erzeugt hat. Eine Hypothese gilt dabei als erzeugt, wenn sie nicht mehr als 0,25 m von einer anderen Hypothese abweicht.

In den nachfolgenden Tests wurde die Methode des einfachen Zusammenfassens verwendet, welche in Kapitel 6 nochmal aufgegriffen wird.

4.2.3 Integration von Vorwissen

Vorwissen kann bei der Anwendung feature-basierter Verfahren eine große Rolle spielen und die Komplexität der verwendeten Verfahren drastisch einschränken.

Beispiele für Vorwissen wurden in vergangenen Kapiteln schon genannt, etwa Vorwissen über die Orientierung des Roboters, über mögliche Regionen, in denen er sich befinden kann, oder aber auch Vorwissen, wie gut ein Verfahren in einer Umgebung abschneidet.

Überlegungen zur Integration von Vorwissen sind angestellt und auch Experimente in dieser Richtung unternommen worden, diese sind jedoch nicht in die Arbeit eingeflossen. Dabei wurde versucht, aus den Sensordaten Situationen zu klassifizieren, welche in früheren Zeitpunkten bei Lokalisationsverfahren aufgetaucht sind. Allerdings hat sich dieser Ansatz als zu aufwändig [188] herausgestellt und zum anderen als wenig aussagekräftig und sehr fehleranfällig [149].

4.3 Anwendung der Methoden und Performancestudien

In diesem Abschnitt sollen die obigen Verfahren in realistischen Einsatzumgebungen verifiziert werden. Dazu wurden realistisch generierte Kartenrepräsentationen für

jedes Lokalisationsverfahren zur Verfügung gestellt, so dass jedes untersuchte Verfahren die gleiche Eingabe an Modellfeatures bekam. Dabei wurden drei verschiedene Karten unterschiedlicher Ausprägung untersucht. Die drei Karten repräsentierten das Erdgeschoss und das Hanggeschoss des Informatikgebäudes sowie das Erdgeschoss des Mathematikgebäudes der Universität Würzburg. Die Ausprägungen der Karten resultieren aus Verfahren zur Erzeugung von Segmentkarten (siehe Kapitel 3). Auf die vierte Repräsentation aus Kapitel 3.3 wurde auf Zeitgründen verzichtet.

Als Bildfeatures dienten die im Kartierungsprozess verwendeten Scans, deren tatsächliche Konfiguration in Weltkoordinaten durch die Kartierung exakt berechnet wurde. Diese Konfigurationen wurden später als Verifikationsinstrument benötigt, um die berechneten Hypothesen zu vergleichen. Die tatsächlichen Konfigurationen waren den Lokalisationsverfahren nicht zugänglich.

Eine klare Trennung zwischen Trainings- und Testdaten wurde nicht vorgenommen, da dies einen zu großen Einfluß darunterliegender Kartierungsverfahren auf die Ergebnisse zur Folge gehabt hätte.

Nachfolgend wurden verschiedene Tests durchgeführt mit den folgenden Fragestellungen:

Test 1: Schnelligkeit und Güte der neuen Verfahren
: Wie gut schneidet das iterative Verfahren mit Standardeinstellungen, Kartensegmentierung und Histogrammauswahl gegenüber dem ursprünglichen Alignment-Verfahren und gegenüber einer weiteren Variante nur mit konstanter Featurereduktion ab?

Test 2: Auswirkungen der unterschiedlichen Verfahren
: Wie gut sind die iterativen Verfahren mit Standardparametern, Kartensegmentierung und Histogrammauswahl in Umgebungen unterschiedlicher Ausprägung? Wie gut ist eine Kombination der Verfahren?

Test 3: Auswirkungen der heuristischen Auswahlverfahren
: Wie verhalten sich die Auswahlverfahren beim iterativen Verfahren mit Standardparametern und Kartensegmentierung? Wie wirkt sich der Parameter der Hypothesenbeschränkung aus?

Für alle Tests wurden insgesamt 139.644 Lokalisationsanfragen durchgeführt, wobei jede einzelne Anfrage mehrere Anfragen an Teilkarten implizieren konnte.

4.3.1 Test 1: Schnelligkeit und Güte des neuen Verfahrens

In diesem Abschnitt werden die folgenden Verfahren gegeneinander getestet:

Alignment
: Das Standard-Alignment-Verfahren wird eingesetzt.

Iteratives Alignment mit konstanter Feature-Auswahl
Ein iteratives Lokalisationsverfahren mit Alignment als Matching-Algorithmus kommt zum Einsatz. Die Bildfeature-Menge wird auf 10 Features begrenzt, abhängig von der Qualität der Segmentextraktion. Die Modellfeature-Menge wird auf 50 Features begrenzt. Diese werden mit der Histogrammreduktion aus der Menge der Modellfeatures ausgewählt. Eine Segmentierung der Karte findet nicht statt. Weiter sind als Parameter $I_{max} = 10$ und $H_{max} = 10$ eingestellt.

Iteratives Alignment mit Segmentierung
Den obigen Verfahren wird das iterative Alignment mit einer Segmentierung der Karte mit $d = 50$ und $\epsilon = \frac{3}{4}$ gegenübergestellt. Die Auswahl der Features aus den Teilkarten der Modellfeatures erfolgt durch Histogrammauswahl. Die Bildfeatures werden wie oben eingeschränkt.

Getestet wurden die Verfahren anhand der drei Instanzen mit unterschiedlichen Ausprägungen. Es sollte mit diesem Test die Fragestellung geklärt werden, wie gut das Verfahren gegenüber dem ursprünglichen Verfahren ist.
Es zeigt sich, dass die neuen iterativen Verfahren dem ursprünglichen Verfahren klar überlegen sind (siehe die Abbildungen 37, 38 und 39). Es werden hohe Lokalisationsgüten zwischen 70 und 80 Prozent aller Anfragen erreicht und die geforderten Echtzeit-Qualitäten, da die Berechnungszeiten mehrerer Anfragen selbst in der Hanggeschoss-Umgebung insgesamt bei maximal 300 Milisekunden lagen.

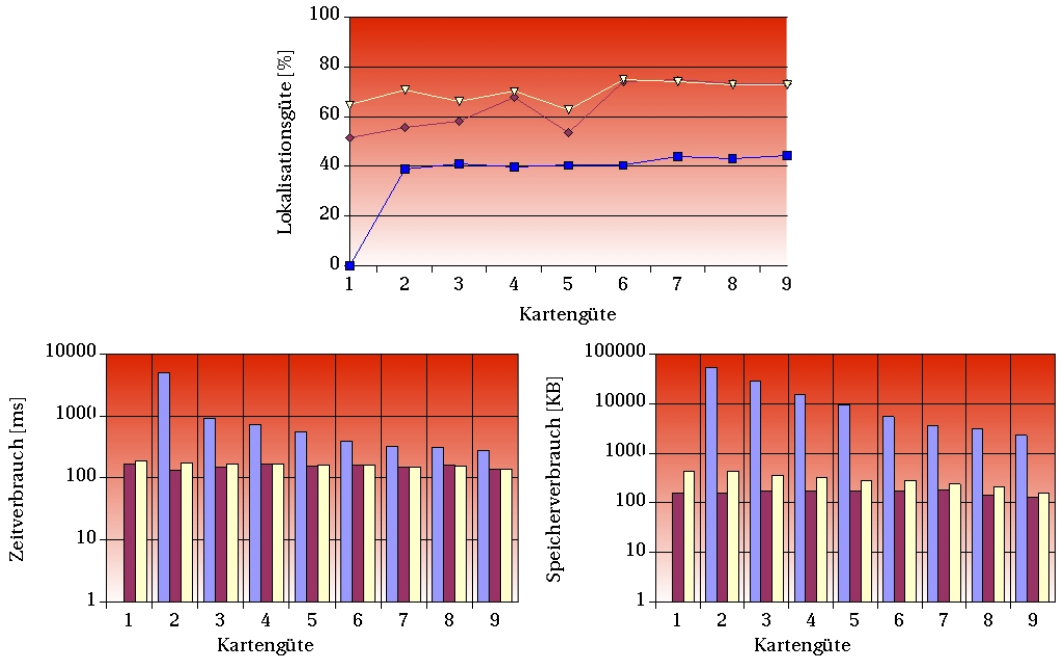
Das ursprüngliche Alignment-Verfahren besitzt eine viel geringere Lokalisationsgüte als die beiden iterativen Verfahren. In der Hanggeschosskarte konnten außerdem keine Ergebnisse erzielt werden. Diese schlechten Ergebnisse sind auf den Abbruch vieler Lokalisationsverfahren aufgrund Speicherplatzmangel oder auf die maximale Lokalisationszeit zurückzuführen.

Weiter zeigt sich, dass eine konstante Featureauswahl zwar Sinn macht, aber die Diskrepanz der Lokalisationsgüte zwischen der konstanten Auswahl an Features und der Kartensegmentierung wächst, je größer die Karten werden.

Weiteres Resultat ist, dass durch die zunehmende Kartengüte (Kartenausprägungen) die Verfahrensergebnisse gegen eine Lokalisationsgüte konvergieren. Dies geschieht deshalb, weil die getesteten Kartenrepräsentationen der Erdgeschosskarten wenige Features besitzen, sodass ab einer bestimmen Kartengüte eine konstante Zahl Features übriggeblieben ist. Dagegen konvergieren die Kurven bei der Hanggeschosskarte nicht, da die Featureanzahl zu hoch ist.

Betrachtet man den Zeit- und Speicherverbrauch, so ist die konstante Featureauswahl das schnellste Lokalisationsverfahren. Das Alignment-Verfahren reagiert auf Umgebungsvergrößerungen mit enormen Zuwächsen an Speicherverbrauch und Laufzeit, und ist deshalb für realistische Anwendungen zu langsam.

Abbildung 37: Test 1-Erste Testumgebung



Abgebildet sind die Lokalisationsgüte (oben) sowie das Laufzeitverhalten (unten links) und das Speicherverhalten (unten rechts) abhängig von der gewählten Kartengüte im Informatik Erdgeschoss. Das Alignment-Verfahren (schwarz) ist das deutlich schlechtere Verfahren im Vergleich mit der konstanten Auswahl (gelb) oder dem schnellen Feature-Verfahren (blau). Das letztgenannte Verfahren liefert die besten Resultate und kann auch im Performance-Vergleich punkten. Die konstante Auswahl hat die beste Performance an Laufzeit und Speicherplatzverbrauch.

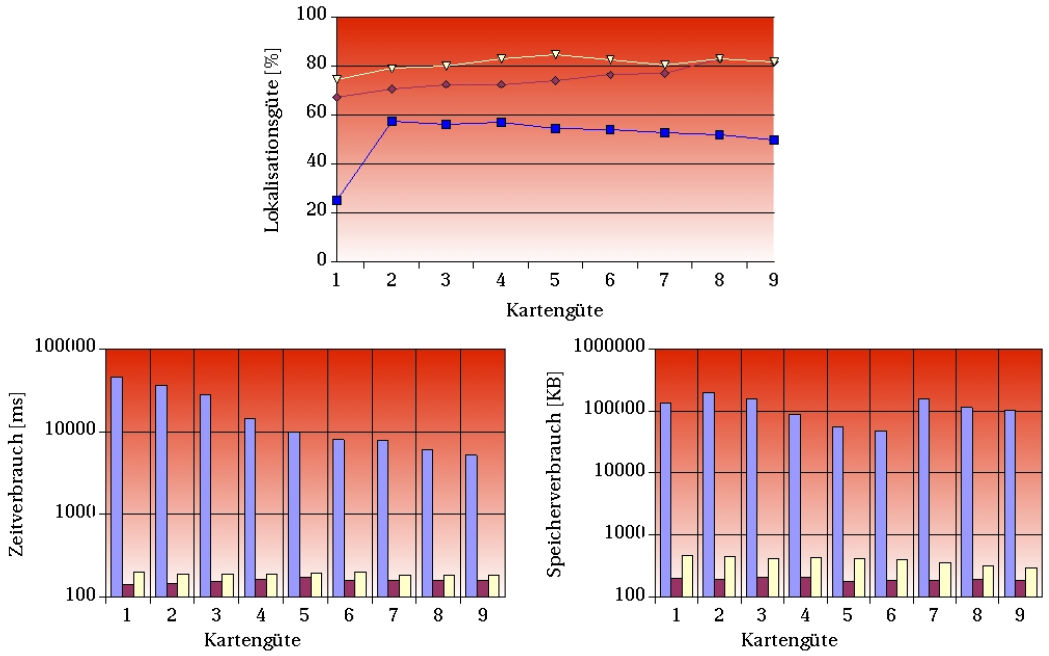
4.3.2 Test 2: Auswirkungen der unterschiedlichen Matching-Verfahren

In diesem Abschnitt wird der Fragestellung nachgegangen, wie gut die iterativen Verfahren mit unterschiedlichem Matchingverfahren in Umgebungen unterschiedlicher Ausprägung sind. Weiter soll der Problematik nachgegangen werden, ob eine Kombination der Verfahren erfolgversprechend ist.

Dabei werden die folgenden Verfahren gegeneinander getestet:

Iteratives Alignment mit Segmentierung
Das iterative Alignment mit einer Segmentierung der Karte wird mit den Parametern $d = 50$ und $\epsilon = \frac{3}{4}$ verwendet. Die Auswahl der Features aus den Teilkarten der Modellfeatures erfolgt durch Histogrammauswahl. Die Bildfeatures werden auf 10 Features eingeschränkt.

Abbildung 38: Test 1-Zweite Testumgebung



Abgebildet sind die Lokalisationsgüte (oben) sowie das Laufzeitverhalten (unten links) und das Speicherverhalten (unten rechts) abhängig von der gewählten Kartengüte im Mathematik Erdgeschoss. Wie vorher ist das Alignment-Verfahren (schwarz) das deutlich schlechtere Verfahren verglichen mit der konstanten Auswahl (gelb) oder dem schnellen Feature-Verfahren (blau).

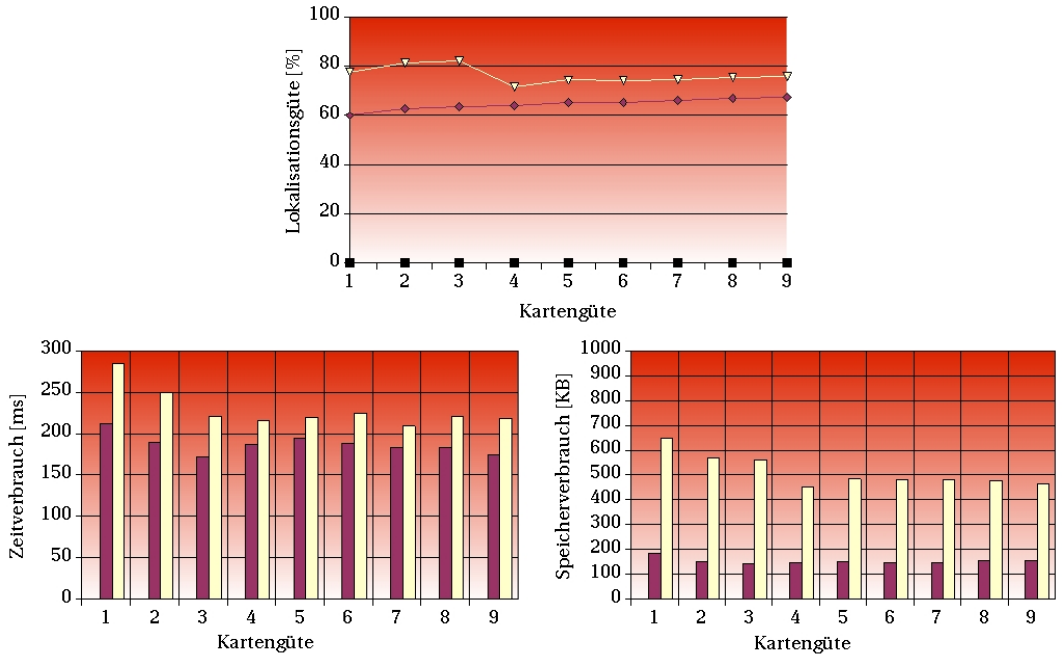
Iteratives Pose-Clustering mit Segmentierung
: Das iterative Pose-Clustering besitzt dieselben Parameter wie das iterative Alignment-Verfahren.

Iteratives Geometrische Hashing mit Segmentierung
: Das iterative geometrische Hashing besitzt dieselben Parameter wie das iterative Alignment-Verfahren.

Kombiniertes Verfahren
: Das kombinierte Verfahren verbindet die drei obigen Verfahren miteinander und führt sie sequentiell hintereinander aus.

Nun ist man in der Lage, die verschiedenen in Kapitel 2 vorgestellten Verfahren in ihrer effizienten Variante gegeneinander zu testen. Wir tun dies in drei verschiedenen Umgebungen mit Standardparametersätzen.

Abbildung 39: Test 1-Dritte Testumgebung

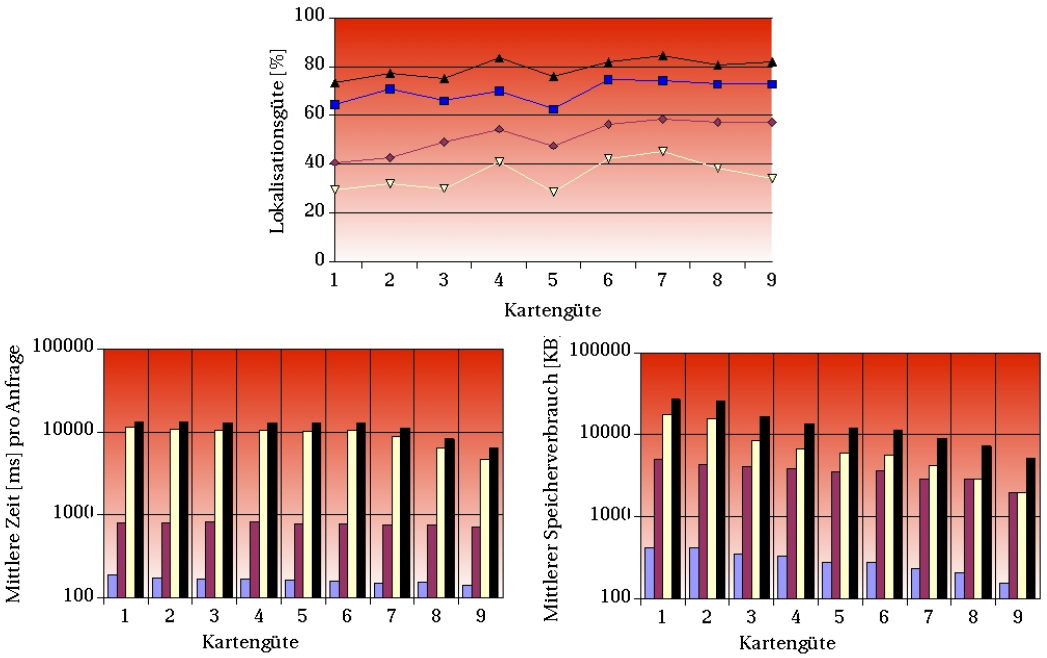


Abgebildet sind die Lokalisationsgüte (oben) sowie das Laufzeitverhalten (unten links) und das Speicherverhalten (unten rechts) abhängig von der gewählten Kartengüte im Hanggeschoss der Informatik. Das Alignment-Verfahren liefert hier keine Ergebnisse mehr, da zuviele Segmente vorhanden sind. Die Gütedifferenz zwischen konstanter Auswahl (gelb) und schnellem Feature-Verfahren (blau) wächst deutlich an.

Es zeigt sich dabei in den Abbildungen 40 - 42 folgendes Bild. Das geometrische Hashing ist wie in den synthetischen Umgebungen das schlechteste Verfahren bezüglich der Lokalisationsgüte. Es kommt auf knapp 50 Prozent Lokalisationsgüte in der Erdgeschosskarte der Mathematik-Umgebung. Das Pose-Clustering-Verfahren erreicht in letztgenannter Umgebung ihr bestes Ergebnis mit ca. 60 Prozent Lokalisationsgüte. In den beiden anderen Umgebungen liegt die Güte allerdings nur bei ca. 50 Prozent. Das Verfahren erweist sich außerdem gegenüber den unterschiedlichen Kartenausprägungen als sehr konstant. Das Alignment-Verfahren ist hier Testsieger, vergleicht man nur die Matching-Verfahren miteinander. Es kommt mindestens auf 70 Prozent der Lokalisationsgüte, in der Erdgeschoss-Mathematik-Umgebung sogar über 80 Prozent.

Es zeigt sich insgesamt, dass die Umgebung des Informatik-Erdgeschosses für die Verfahren eine sehr schlechte Umgebung darstellt, dagegen das Mathematik-Erdgeschoss eine sehr gute Umgebung zu sein scheint.

Abbildung 40: Test 2-Erste Testumgebung

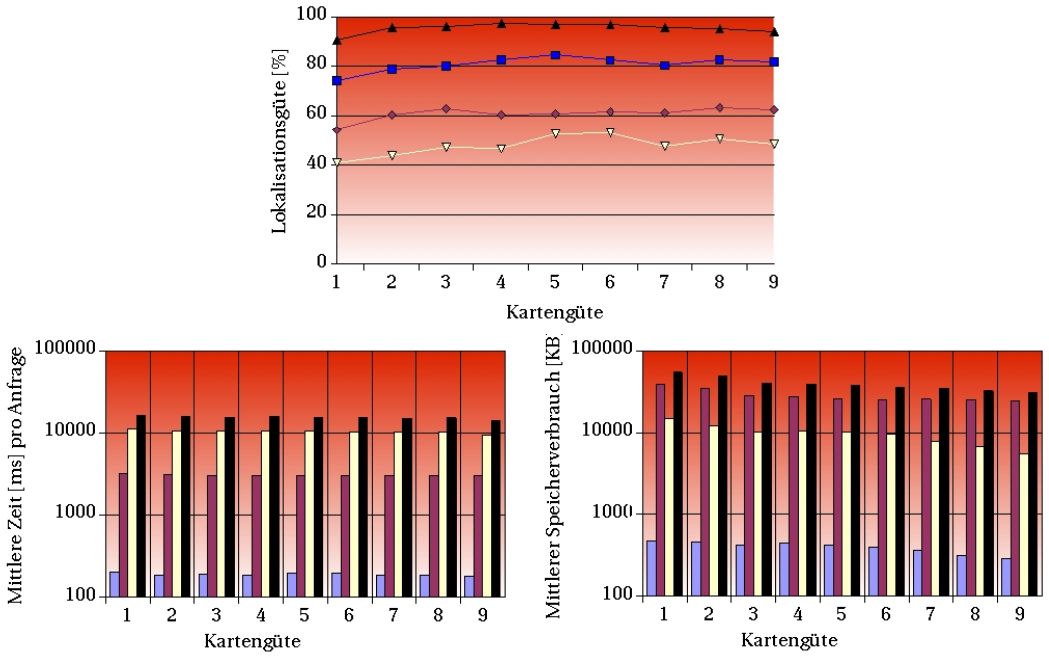


Vergleich der iterativen Lokalisationsverfahren im Erdgeschoss der Informatik. Das geometrische Hashing (gelb) wird gegen das Pose-Clustering-Verfahren (rot), das Alignment-Verfahren (blau) und das kombinierte Verfahren (schwarz) getestet. Man erkennt nahezu konstante Ergebnisse aller Verfahren über alle Kartenausprägungen hinweg. Alle Lokalisationsergebnisse deuten darauf hin, dass diese Testumgebung für die Verfahren eine Herausforderung darstellt, liegt die kombinierte Güte doch nur bei ca. 80 Prozent. Außerdem erkennt man, dass Alignment die beste Performance bietet.

Betrachtet man die unterschiedlichen Kartenausprägungen, stellt man für die Verfahren keine sonderlich großen Abweichungen fest. Leichte Schwankungen nach oben und auch nach unten bestimmen dabei das Bild. Was allerdings nicht verwundert, ist die Tatsache, dass der Speicherplatzbedarf und der Zeitbedarf mit den Kartenausprägungen abnimmt, da die Featuremenge von Kartengüte 1 bis Kartengüte 9 stark sinkt. Da sich kaum Unterschiede bei der Lokalisationsgüte bemerkbar gemacht haben, ist dieses eine weitere erfreuliche mögliche Reduktion der Komplexität.

Es zeigt sich, dass die Matchingverfahren sich in der Kombination der Verfahren sehr gut gegenseitig ergänzen, bekommt man doch sehr gute Lokalisationsergebnisse. In der Mathematik-Erdgeschoss-Umgebung liegt die Lokalisationsgüte sogar über 95 Prozent. Der Nachteil dabei ist, dass das kombinierte Verfahren die einzelnen Verfahren nicht schneller macht, lediglich die Güte steigt drastisch an. So benötigt das kombinierte Verfahren in der Hanggeschoss-Umgebung weit über 10 Sekunden für eine Lokalisatio-

Abbildung 41: Test 2-Zweite Testumgebung



Die vier Verfahren werden in der Mathematik-Erdgeschoss Umgebung getestet. Diese Umgebung scheint feature-basierte Verfahren zu belohnen, liegt doch die kombinierte Güte weit über 95 Prozent. Der Speicher- und Zeitaufwand ist leicht erhöht im Vergleich mit der ersten Testumgebung, da diese Umgebung mehr Features besitzt.

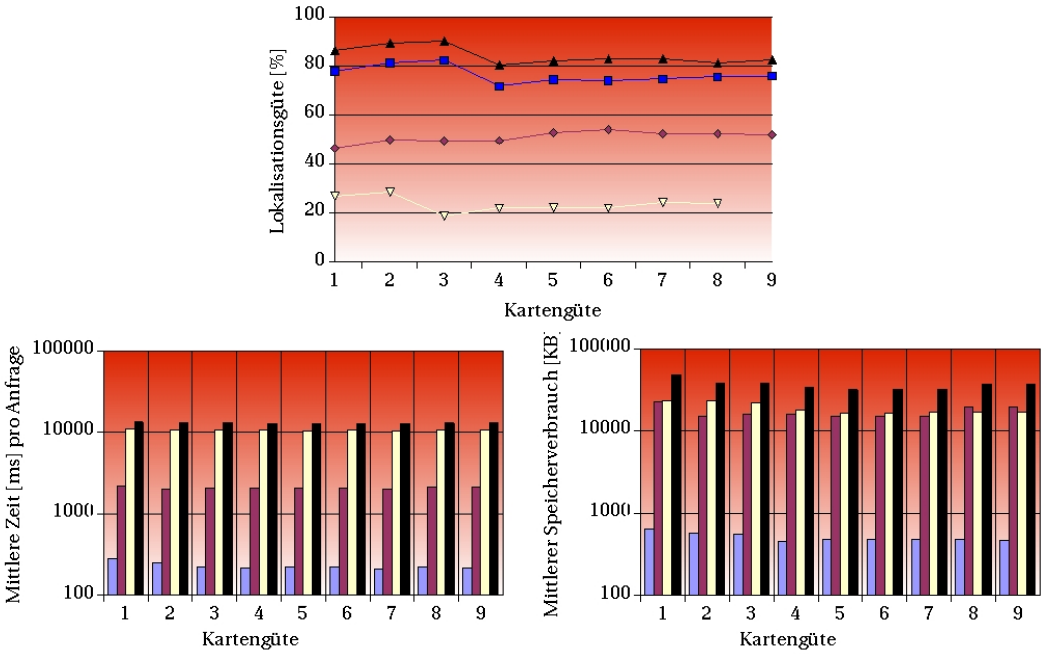
nanfrage. Deshalb wird in späteren Kapiteln nur das Alignment-Verfahren angewandt, da es vom Zeit- und Speicheraufwand einen guten Performancewert aufweist, obwohl eine Kombination der Verfahren bessere Ergebnisse bietet.

4.3.3 Test 3: Auswirkungen der heuristischen Auswahlverfahren

Als nächstes werden die verschiedenen Auswahlkriterien gegeneinander getestet, die in diesem Kapitel besprochen wurden. Es werden nun die folgenden Verfahren gegeneinander getestet:

Iteratives Alignment mit Histogrammauswahl ohne Hypothesenbeschränkung
Das iterative Alignment mit einer Segmentierung der Karte wird mit den Parametern $d = 50$ und $\epsilon = \frac{3}{4}$ verwendet. Die Auswahl der Features aus den Teilkarten der Modellfeatures erfolgt durch Histogrammauswahl mit 50 Features. Die Bildfeatures werden auf 10 Features eingeschränkt. Es erfolgt jedoch kein frühzeitiger Abbruch der Berechnung durch eine Beschränkung der Hypothesenmenge.

Abbildung 42: Test 2-Dritte Testumgebung



In der Umgebung Informatik-Hanngeschoss bietet sich wie in den beiden vorherigen Testszenarien das gleiche Bild. Das kombinierte Verfahren ist das beste, gefolgt von Alignment. Das Alignment-Verfahren kann seine Güte als einziges Verfahren in etwa halten, die anderen Verfahren liefern im Vergleich die schlechtesten Ergebnisse in dieser Umgebung.

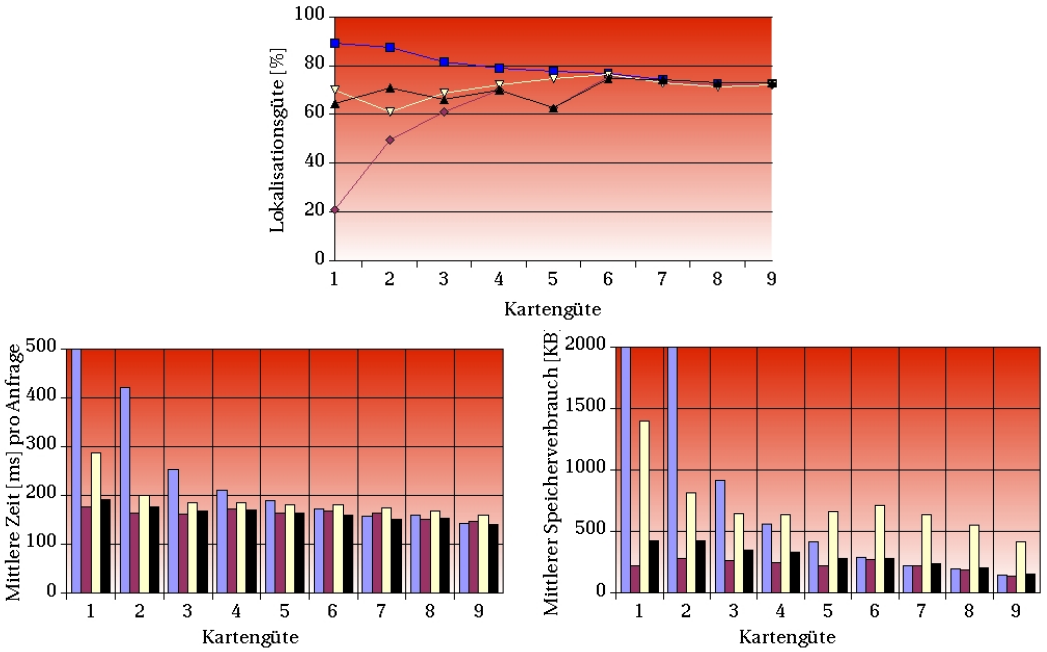
Iteratives Alignment mit Hypothesenbeschränkung und Standardparametern
Verfahren wie oben, jedoch mit einer Hypothesenbeschränkung auf 10 Hypothesen. Dieses Verfahren ist schon bei früheren Tests eingesetzt worden.

Iteratives Alignment mit einmaliger probabilistischer Auswahl
Iteratives Alignment mit einer Segmentierung der Karte und obigen Parametern. Es wird jedoch keine Histogrammauswahl vorgenommen, sondern eine probabilistische Auswahl von 50 Features.

Iteratives Alignment mit zweimaliger probabilistischer Auswahl
Iteratives Alignment mit probabilistischer Auswahl wie oben. Jedoch werden pro Iteration zweimal jeweils 50 Features (u.U. dieselben Features) probabilistisch gezogen und Hypothesen berechnet.

Dabei soll den Fragestellungen nachgegangen werden, wie sich die Auswahlverfahren mit Standardparametern und Kartensegmentierung gegeneinander verhalten und wie

Abbildung 43: Test 3-Erste Testumgebung

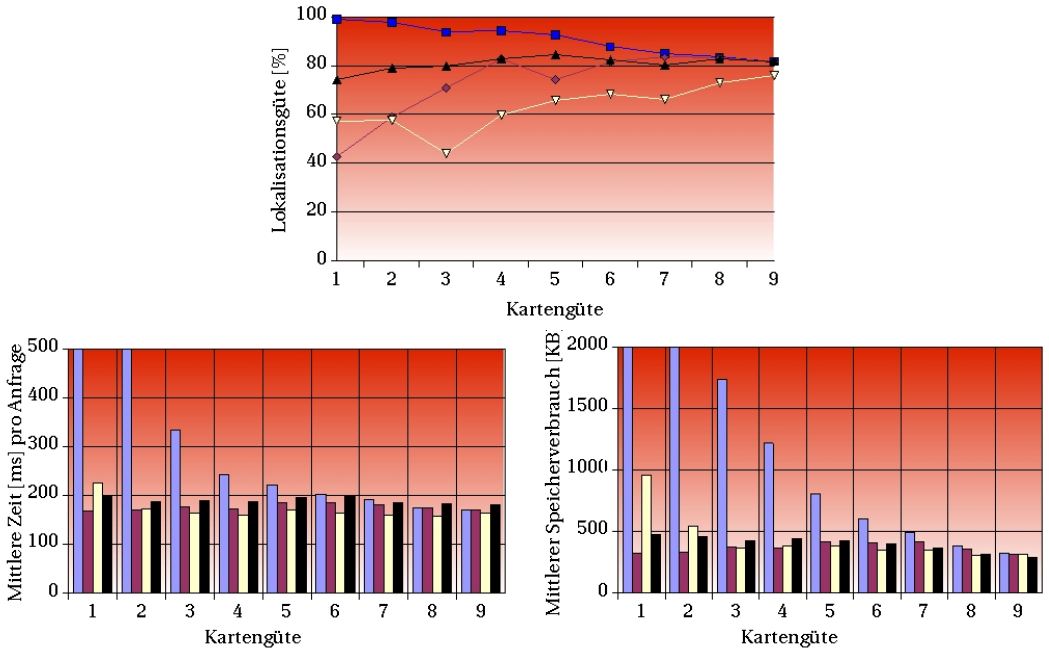


Die Abbildung zeigt die Testergebnisse in der Erdgeschoss-Informatik-Umgebung. Das probabilistische Verfahren mit einmaliger Auswahl (rot) wird gegen das mit zweimaliger Auswahl (gelb), die Histogrammauswahl (schwarz) und das Standardverfahren ohne Hypothesenbeschränkung getestet. Dabei stellt man eine starke Schwankungsbreite der probabilistischen Verfahren fest. Besonders in den Kartenausprägungen mit großer Featurezahl ist eine zufällige Featurewahl nicht das richtige Rezept. Bei mehrmaliger Anwendung der zufälligen Auswahl können sich die zufälligen Effekte verringern, so dass ein stabileres Verhalten in der Lokalisationsgüte zu beobachten ist. Das Verfahren ohne Hypothesenbeschränkung liefert deutlich bessere Ergebnisse in größeren Karten, benötigt aber auch eine längere Laufzeit. Ansonsten sind die probabilistischen Verfahren die schnellsten.

sich insbesondere das Pruningkriterium zur Hypothesenbeschränkung auswirkt.

Die Tests sind jeweils wieder in den drei schon bekannten Umgebungen durchgeführt worden, welche die Abbildungen 43, 44 und 45 zeigen. Betrachtet man zunächst die Fragestellung, ob das Pruning durch eine feste Hypothesenanzahl große Effekte zu verzeichnen hat, so muss man dieses bestätigen. In den kleineren Testumgebungen besteht eine Qualitätsdifferenz von fast 20 Prozent bei Kartenausprägungen mit großer Featurezahl. Dieser Unterschied wird jedoch durch die Kartengüte aufgehoben, so dass sich die Lokalisationsgüte auf hohem Niveau stabilisiert. In der Hanggeschosskarte bleibt die Lokalisationsdifferenz sehr hoch, da die Featuremenge trotz der Effekte der

Abbildung 44: Test 3-Zweite Testumgebung



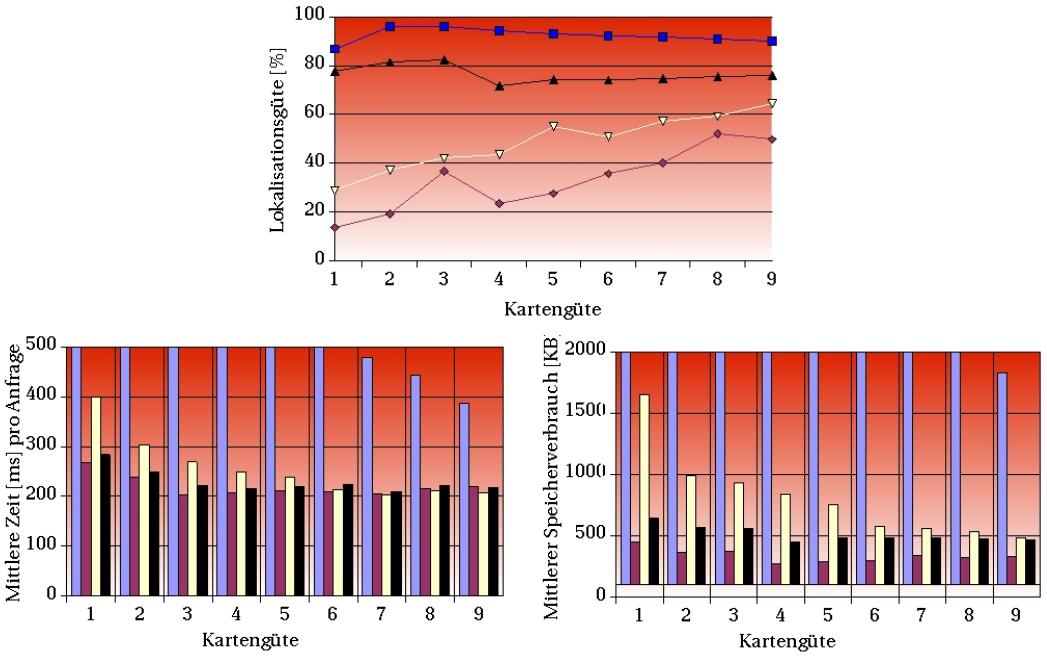
Die Abbildung zeigt die Testergebnisse in der Erdgeschoss-Mathematik Umgebung. Die Farbverteilung ist wie in Abbildung 43 angegeben. Auffallend ist hier, dass die einmalige probabilistische Auswahl die zweimalige Auswahl in der Lokalisationgüte klar schlägt. Außerdem kann man feststellen, dass die Wahl der Kartengüte die maximal erreichbare Lokalisationsgüte ebenfalls klar einschränkt.

Kartengüte nicht stark genug verringert wird.

Was die Kartengüte ohne eine Hypothesenbeschränkung verbessert, wird durch den dramatischen Anstieg des Speicher- und Zeitverbrauchs verschlimmert. Die Anfragezeiten liegen dann mit über 2000 Millisekunden nicht mehr im Echtzeitbereich. Damit zeigt sich, dass ein vorzeitiges Abbruchkriterium die Lokalisationsgüte zwar entscheidend beschneidet, jedoch damit sehr schnelle Anfragen zur Verfügung stellt, selbst in großen Karten.

In der zweiten Fragestellung gibt es ein ähnliches Dilemma. Die Histogrammauswahl ist deutlich besser als die probabilistischen Auswahlverfahren in großen Umgebungen wie der Hanggeschosskarte. Dort gibt es eine Performancedifferenz in der Lokalisationsgüte von 50 Prozent. In kleineren Umgebungen negiert sich jedoch teilweise das Bild. Allerdings muss man die Testläufe der probabilistischen Verfahren mit einiger Vorsicht betrachten, scheint die Schwankungsbreite doch sehr hoch zu sein. So schlägt

Abbildung 45: Test 3-Dritte Testumgebung



Die Abbildung zeigt die Testergebnisse in der Hanggeschoss-Informatik Umgebung. Die Farbverteilung wurde in Abbildung 43 angegeben. Deutlich kann man feststellen, dass die probabilistischen Verfahren unterlegen sind. Zudem konvergieren die Verfahren mit zunehmender Segmentmenge der Karten nicht mehr alle gegen eine Lokalisationsgüte, sondern es bleibt eine Gütedifferenz bestehen.

die einmalige zufällige Auswahl in der Erdgeschoss-Mathematik-Umgebung die zweimalige Auswahl klar.

Betrachtet man sich die Performance-Werte der Verfahren, haben die probabilistischen Verfahren klar die Nase vorn. Insgesamt kann man außerdem verzeichnen, dass die Lokalisationszeit bei größeren Featuremengen in allen Verfahren ansteigt.

Betrachtet man zum Schluß dieses Tests die verschiedenen Kartenausprägungen der Verfahren, kann man bei kleinen Karten für die Histogrammauswahl ein eher konstantes Verhalten bewerten, so dass man eine Karte von der Kartengüte 9 bevorzugen müßte. Die Lokalisationszeit nimmt bei allen Verfahren mit dem Anstieg der Kartengüte ab. Für die nachfolgenden Kapitel werden deshalb Ausprägungen der Kartengüte 9 verwendet.

4.4 Resümee

In diesem Abschnitt wurden Heuristiken zur Effizienzsteigerung feature-basierter Lokalisationsstrategien vorgestellt. Diese besitzen nun dank schneller Berechnungszeit und guter Lokalisationsergebnisse eine realistische Perspektive für einen Einsatz in der Robotik.

Die Verfahren wurden in mehreren experimentellen Testreihen in verschiedenen realistischen Einsatzumgebungen auf ihre Tauglichkeit getestet. Dabei zeigen sich die probabilistischen Verfahren zeitlich sehr überlegen mit geringer Lokalisationsgüte. Die Histogrammauswahl brilliert durch gute Anfrageergebnisse und stärkeren Berechnungsaufwand. Ein kombiniertes Verfahren verwendet alle vorgestellten Verfahren und erreicht unerwartet hohe Lokalisationsergebnisse, hat aber fast unbrauchbaren Zeitkonsum. Dem Anwender bleibt es schließlich überlassen zu entscheiden, welches Verfahren zum Einsatz kommt.

Gleichzeitig haben sich weitere Probleme ergeben, die durch die Tests offen gelassen werden mussten. Große Karten schaffen neue Probleme, die noch zu untersuchen sind. Dabei stößt man leicht an die Leistungsgrenze heutiger Computer. Eine automatische Clusterung großer Karten in heterogene Teile scheint ein richtiger Weg zu sein, um kleinen homogenen Teilkarten entgegenzuwirken, die sehr viele Features besitzen.

Ebenso scheint der Ansatz tauglich, die Reihenfolge der iterativen Verfahren geeignet zu wählen und die Wahl mit Vorwissen zu verknüpfen. Damit kann man iterierte Lokalisationsanfragen auf die besten Teilkarten beschränken. Kann man etwa Teilkarten ebenso wie Features bewerten und dann eine Reihenfolge determinieren, welche Karten in welcher Reihenfolge ausgewählt werden? Ein Ansatz wäre zum Beispiel, die Features im jeweiligen Kartenteil zu bewerten und daraus eine Gesamtbewertung zu berechnen.

Teil II

Hypothesenelimination

Übersicht

Im ersten Teil der Arbeit wurde die effiziente Generierung von hypothetischen Roboterstandorten mit Hilfe feature-basierten Verfahren unter Zuhilfenahme generierter feature-basierter Umweltmodelle erarbeitet. Nun besteht eine Problemlösung des globalen Selbstlokalisationsproblems im zweiten Teil aus einem Algorithmus zur effizienten Elimination der überzähligen Lokalisationshypothesen. Die vorzustellende Problemlösung gliedert sich in zwei Phasen:

Navigation in lokalen Karten
Zunächst wird die Problematik behandelt, wie man in erstellten Umweltmodellen Pfade plant und mit einem mobilen Roboter mit unsicherem Sensorwissen entlang dieser Pfade zu anvisierten Zielpunkten gelangt und dabei dynamischen Hindernissen ausweicht. Dazu wird das Navigationsproblem in die Teilprobleme Pfadplanung, Positionsverfolgung, dynamische Hindernisvermeidung und Positionskorrektur aufgeteilt und verwendete Lösungen der Literatur vorgestellt. Abschließend wird ein Gesamtkonzept für die Navigation als Grundlage zur Lösung der Hypothesenelimination präsentiert.

Hypothesenelimination
Nach der Lösung grundlegender Probleme zum sicheren Erreichen eines Sensorpunktes besteht das Problem der Hypothesenelimination darin, geeignete Sensorpunkte oder Ziele auszuwählen, an denen mehrdeutige Positionen eliminiert werden können, und diese sodann in einer geeigneten Reihenfolge abzufahren. Dazu wird zunächst die theoretische Problemstellung und deren beste Lösung vorgestellt. Anschließend wird die Aufgabenstellung der selbstständigen Kartierung mit der Problematik des Feature-Matchings kombiniert, um Sensorpunkte gewinnen und verifizieren zu können. Ein effizienter Problemlösungsalgorithmus wird vorgestellt und anschließend experimentell bestätigt.

5 Navigation in lokalen Karten

Ein Roboter kann nur Objekte manipulieren, wenn er sie erreichen kann. Die meisten industriellen Roboter sind stationäre Roboter, die sich innerhalb einer begrenzten Reichweite (abhängig vom Manipulatorarm) ihr Arbeitsobjekt holen und manipulieren. Zukünftige mobile Roboter sollen die Reichweitenbeschränkung aufheben, autonom und unabhängig arbeiten und den Transport von Material zu den stationären Robotern übernehmen. Die Vision ist eine Roboterfabrik fast ohne menschliche Mitarbeiter [82, 101] mit autonomen interagierenden Produktionshelfern.

Wichtigste Aufgabe, um einen Teil dieser Vision zu verwirklichen, ist dabei die Navigation eines Roboters. Navigation bedeutet soviel wie die Gesamtheit der Maßnahmen zur Planung eines Bewegungsablaufs von einem Start- zu einem Zielpunkt, zur Einhaltung des gewählten Kurses und zur Positionierung am Zielpunkt.

Navigation – wie sie hier verstanden werden soll – ist eine Aufgabe, die an einen mobilen Roboter mit der Voraussetzung gestellt werden kann, dass dem Roboter sein aktueller Standort oder seine aktuelle Konfiguration in seiner Einsatzumgebung bekannt ist und er Wissen über seine Umgebung in Form einer Kartenrepräsentation (siehe Kapitel 3) besitzt. Ferner sei eine Menge von Zielen in der Karte durch eine Menge von Konfigurationen bekannt, die nacheinander abgefahren werden sollen.

Die Aufgabe ist es dann, falls es einen gangbaren Weg von der initialen Konfiguration des Roboters zur Zielkonfiguration gibt, einen solchen unter bestimmten Optimalitätsbedingungen zu planen, diesen Weg auch unter dem Einfluss dynamischer Hindernisse abzufahren und die tatsächliche Zielkonfiguration zu erreichen. Im Kontext der Hypothesenelimination wird die Zielsuche verwendet, um in einer lokalen Karte geplante Zielpunkte anzufahren und an diesen neue Informationen zu akquirieren (siehe dazu auch Kapitel 6).

In diesem Kapitel wird das Navigationsproblem - wie in der Literatur üblich - zunächst in Teilprobleme unterteilt und für jedes solche die aktuellsten Problemlösungen aus der Literatur präsentiert. Anschließend wird eine Gesamtproblemlösung angegeben, wie sie im weiteren Verlauf der vorliegenden Arbeit verwendet wird.

5.1 Das Problem der Navigation autonomer mobiler Roboter

Die formale Problemstellung der Navigation kann folgendermaßen beschreiben:

Problem 5.1 (Navigation, Zielsuche) *Gegeben sei ein mobiler Roboter mit einer Startkonfiguration q_{start}, einer Endkonfiguration als Ziel q_{end} und eine (2D-)Roboterkarte m_g der Umgebung. Aufgabe ist es, den besten Weg von q_{start} nach q_{end} zu berechnen, diesen erfolgreich zu traversieren um abschließend den Zielpunkt zu erreichen. Dabei wird ein bester Weg abhängig von einem Optimalitätskriterium ausgewählt.*

Dieser einfachen Problematik ist nicht anzusehen, dass die einzelnen Teilprobleme so komplex sind. Das Problem, eine Karte der Umgebung zu generieren, wurde schon in Kapitel 3 behandelt. Geht man blauäugig an die Problematik heran, bräuchte man zur Problemlösung nur einen Pfad vom Startpunkt zum Zielpunkt zu planen und dann exakt-berechnete Bewegungsbefehle auszuführen. Tatsächlich scheitert man schon bei der einfachen Aufgabe, die exakte initiale Position des Roboters zu bestimmen. Der berechnete Pfad des Roboters muss den physikalischen Abmessungen des Roboters und dem kinematischen Modell entsprechen, so dass sichergestellt ist, dass der Roboter nicht steckenbleibt und den Pfad traversieren kann. Ist der Roboter einmal angefahren, muss zu jedem Zeitpunkt die exakte Position zur Positionskorrektur mit dem vorberechneten Pfad abgeglichen werden. Zusätzlich muss zu jedem Zeitpunkt ein gültiger Geschwindigkeitsvektor gesetzt sein, so dass das mobile Gerät auf zukünftige Hindernisse entsprechend reagieren kann. Außerdem muss während der Bewegungsphase die Betriebssicherheit gewährleistet sein, d.h. der Roboter muss mobilen Hindernissen ausweichen, rechtzeitig bremsen und in ausweglosen Situationen stehenbleiben.

5.1.1 Lösungsansätze in der Literatur

Betrachtet man sich die Problematik genauer, kann man drei Fragen formulieren, welche ein erfolgreicher Navigations-Algorithmus beantworten muss:

Wahl des Umgebungsmodells
Wie wird die Umgebung modelliert, in der sich der Roboter bewegt?

Bestimmung der Roboter-Position
Wie bestimmt man die Position des Roboter zu jedem Zeitpunkt, wenn sich der Roboter dabei bewegt?

Bewegungsverhalten des Roboters
Wie modelliert man das Bewegungsverhalten des Roboters so, dass ein Pfad geplant und gleichzeitig Hindernissen effizient ausgewichen werden kann?

Wahl des Umgebungsmodells

Bei der Umgebungsmodellierung kann man zwischen zwei Fällen unterscheiden:

Statisches Modell
Der Roboter ist das einzige bewegte Objekt innerhalb der Umgebung [132].

Dynamisches Modell
Der Roboter bewegt sich in einer hochgradig dynamischen Umgebung mit einer Vielzahl dynamischer Objekte [128].

Das statische Umgebungsmodell hat den Vorteil, dass sich das Navigationsproblem nur auf die Aufgabe beschränkt von der Start- zur Zielkonfiguration zu fahren. Zudem können die meisten Kartierungsverfahren nur statische Modelle modellieren. Dynamische Umgebungsmodelle hingegen modellieren durch Langzeit-Beobachtungen die Dynamik jeder Konfiguration, so dass eine Pfadplanung aufgrund dieser Informationen dem statischen Ansatz überlegen ist (siehe dazu auch Kapitel 3.2.4).

Bestimmung der Roboter-Position

Man kann in der Literatur zunächst grundsätzlich zwischen zwei Klassen von Ansätzen zur Bestimmung der Roboterposition während der Roboterbewegung unterscheiden:

Landmarken-basierte-Verfahren
Die Umgebung wird mit Landmarken markiert, so dass zu jedem Zeitpunkt die Position des Roboters genau bestimmt werden kann [25].

Sensor-basierte Verfahren
Aufgrund von Sensorinformationen und einer gegebenen Karte der Umgebung soll der Roboter seine Position feststellen. Bauliche Veränderungen der Umgebung sind ausgeschlossen [69, 92].

Man kann dabei weiter zwischen aktiven und passiven Landmarken unterscheiden [25]. Aktive Landmarken-Systeme [146] sind die gebräuchlichsten Navigationshilfen und werden recht häufig in der Industrie bei Navigationsmanövern eingesetzt, zum Beispiel bei der Navigation von Schiffen. Sie erlauben eine sehr genaue Positionierung. Aktive Landmarken senden ein eindeutiges Signal aus, mit welchem man mit Hilfe der Triangulations-Technik den exakten Standort des Geräts bestimmen kann. Typische Beispiele solcher Landmarken sind Infrarot-Landmarken, Lichtschranken oder aber auch das GPS-System.

Passive Landmarken-Systeme [66] repräsentieren eindeutige Markierungen in der Einsatzumgebung eines Roboters. Diese besitzen eine feste Position und müssen vom Roboter im schlechtesten Fall erst entdeckt werden. Typische Beispiele solcher Landmarken sind Markierungslinien auf dem Boden, Barcodes an Wänden [25].

Sensor-basierte Verfahren [23, 37, 133, 134] können dagegen schnell ohne bauliche Veränderungen in beliebigen Umgebungen eingesetzt werden. Solche versuchen ihre Position aufgrund der Startposition und der bis zum aktuellen Zeitpunkt aufgenommenen

Sensorinformationen zu bestimmen. Dabei sind diese Verfahren fehleranfällig in der Weise, dass korrupte Sensordaten zu falschen Ergebnissen führen können. Der Vorteil solcher Verfahren liegt in der uneingeschränkten Bewegungsfreiheit der Roboter und dem Wegfallen des Markierungsaufwandes in der Einsatzumgebung. Beispiele funktionierender Systeme sind die Museumsroboter Rhino [69] und Minerva [207].

Bewegungsverhalten des Roboters

Weiter kann man grundsätzlich unterscheiden, wie der Roboter einen Pfad von einer Start- zu einer Zielkonfiguration findet.

Planungs-basierte Verfahren
Ein Pfad wird in einem Umgebungsmodell von der Startkonfiguration zur Zielkonfiguration komplett geplant [132].

Reaktive Verfahren
Zu jedem Zeitpunkt wird der lokal beste Pfad ausgewählt, welchen der Roboter entsprechend seinem Optimalitätskriterium auswählt [33].

Der Unterschied besteht darin, dass bei planungs-basierten Verfahren die Umgebung bekannt ist, während man bei reaktiven Verfahren zusätzlich die lokalen Sensorinformationen in die Entscheidung einbezieht. Dabei ist zu beachten, dass man einen Pfad auch in einem dynamischen Umgebungsmodell mit dem ersten Ansatz planen kann [128]. Reaktive Verfahren zeichnen sich dadurch aus, dass sie aktuell und schnell auf Änderungen der Umgebung reagieren können, jedoch nicht den Zielpunkt erreichen müssen. Eine lokale Bewertungsfunktion kann bekanntermaßen in lokalen Minima gefangen sein. Ein planungs-basiertes Verfahren dagegen findet immer einen besten Weg, so er existiert, jedoch sind solche Verfahren langsam und können nicht auf aktuelle Entwicklungen reagieren.

5.1.2 Dekomposition des Problems

Hier soll im folgenden das Problem der Navigation in vier verschiedene Teilprobleme zerlegt werden. Dabei wird ein dynamisches Umgebungsmodell vorausgesetzt. Die Positionen des Roboters sollen mittels sensor-basierter Verfahren ermittelt werden. Außerdem wird ein hybrider Ansatz verfolgt, welcher einen Pfad nach einem planungsbasiertem Schema in eine Menge von Teilziele zerlegt und während der Pfadverfolgung reaktiv diese Teilziele abfährt. Im Überblick ergeben sich dann die folgenden zu lösenden Teilprobleme:

Pfadplanung vom Start zum Ziel
Berechne in der lokalen Karte einen besten Weg zwischen beiden Konfigurationen. Dieser Ansatz ist planungs-basiert und verwendet ein statisches Weltmodell zu einem Zeitpunkt t. Dieses Teilproblem liefert eine Menge von Teilzielen.

Positionsverfolgung
Die Positionsverfolgungs-Algorithmen berechnen aus Odometriedaten, Karte und aktuellen Sensorwerten eine neue Roboterkonfiguration in Weltkoordinaten.

Dynamische Hindernisvermeidung
Abarbeiten und Anfahren der lokalen Ziele und Vermeidung von Kollisionen mit der Umgebung und mit Hindernissen.

Positionskorrektur-Problem
Positionsverfolgungsansätze sind rechenintensiv, so dass diese Lösungsalgorithmen extern berechnet werden. Durch ein solches verteiltes System (siehe Kapitel 8.2) müssen Kommunikationszeiten mit einbezogen werden, die die Roboterposition mit der korrigierten Positionsschätzung synchronisieren.

Dieser Ansatz zur Lösung des Navigationsproblems ist in der aktuellen Literatur sehr erfolgreich [96, 151, 166, 180, 219]. Der Unterschied des hier beschriebenen Ansatzes im Vergleich beruht darauf, dass die Positionsverfolgung nicht direkt auf dem Roboter-System berechnet wird. Dadurch weiß man erst zu späteren Zeitpunkten, wo sich der Roboter befunden hat und wie groß der Positionsfehler war. Dieser Fehler muss dem Roboter in geeigneter Weise übermittelt werden. Das Update-Problem taucht ursprünglich in der Literatur bei der Überwachung von Raumsonden auf. Dort werden zu vorbestimmten Zeitpunkten sogenannte TCM-Korrekurmanöver ausgeführt, um diese auf den entsprechenden Trajektorien zu halten [228].

5.2 Pfadplanung vom Start zum Ziel

Das Pfadplanungsproblem alleine erfreut sich in der Literatur großer Beliebtheit [42, 60, 133, 144, 179, 182]. Dies ist nicht verwunderlich, ist dieses Problem doch schon in einer sehr schwachen Ausprägung ein $\mathcal{NP}$-hartes Problem, wenn man einen Pfad in einer Umgebung mit Objekten konstanter Geschwindigkeit planen möchte [41, 125, 172]. Deshalb muss man sich mit Heuristiken zur Lösung der Problemstellung beschäftigen.

Hier soll eine 2-dimensionale Karte der Einsatzumgebung durch m_{feat} oder m_{grid} vorausgesetzt werden. Im Falle einer Gitterkarte repräsentieren dynamische Konfigurationen zunächst freie Konfigurationen, da deren Zustand ja auch frei sein kann. Dies macht Sinn, denn so kann man auch durch Türen Pfade planen, die in diesem optimistischen Fall als offen vorausgesetzt werden. Im Falle einer Segmentkarte werden Hindernisse zum aktuellen Zeitpunkt als belegte Konfigurationen bewertet. Alle anderen Konfigurationen repräsentieren freie Konfigurationen. Dann kann man einen Pfad in $\mathcal{C}_{free}$ definieren:

Definition 5.2 (Pfad) *Ein Pfad in $\mathcal{C}_{free}$ eines Roboters $\mathcal{A}$ von der initialen Konfiguration q_{start} zur Konfiguration q_{end} ist eine stetige Abbildung $\tau : [0, 1] \rightarrow \mathcal{C}$ mit $\tau(0) = q_{start}$ und $\tau(1) = q_{end}$.*

Ein Pfad ist regulär, wenn der Roboter diesen traversieren und somit auch das Ziel vom Startpunkt aus erreichen kann. Insbesondere ist ein Roboter mit einer realistischen geometrischen Ausdehnung in der Lage, den regulären Pfad zu traversieren. Mit den obigen Definitionen kann man das Problem folgendermaßen definieren:

Problem 5.3 (Pfadplanung) *Gegeben ist eine Startkonfiguration* q_{start} *und eine Zielkonfiguration* q_{end}*. Gesucht ist ein* ***regulärer Pfad*** *von der Startkonfiguration zur Zielkonfiguration, d.h. eine Abbildung* $\tau : [0,1] \rightarrow \mathcal{C}_{free}$*, wobei* $\tau(0) = q_{start}$ *und* $\tau(1) = q_{end}$*.*

Die Problematik, dynamische Hindernisse in der Karte zu berücksichtigen, wird durch reaktive Algorithmen zur dynamischen Hindernisvermeidung realisiert (siehe Kapitel 5.4). Somit wird die schwierige Problematik der Pfadplanung mit bewegten Objekten durch eine auch vom Menschen verwendeten Heuristik beschrieben, indem zunächst der Pfad optimal geplant wird und erst beim Abfahren entschieden wird, welche Modifikationen und Umwege gefahren werden müssen.

5.2.1 Generierung der Anfangs- und Endparameter

Für einen Pfadplanungsalgorithmus sind die Bestimmung von Anfangs- und Endpunkt von Wichtigkeit, und sollen im folgenden betrachtet werden.

Verifikation der initialen Position

Die Modellierung der Positionsunsicherheit wird durch eine Konfiguration und durch einen Positionsfehler beschrieben. Für einen probabilistischen Ansatz wird dies durch eine Kovarianzmatrix und für einen geometrischen Ansatz durch eine geometrische Fehlerregion modelliert. In beiden Fällen wird eine Unsicherheitsregion definiert, welche man durch eine Menge von Partikeln mit unterschiedlichen Konfigurationen in der Region repräsentieren kann (siehe dazu auch Kapitel 3.2.3).

Es reicht prinzipiell, wenn die initiale Position vorgegeben ist, so dass ein Pfadplaner von diesem Startpunkt aus planen kann. In der Realität erzeugt man solche initialen Positionen für den Roboter bei der Generierung des Umweltmodells. Man beginnt die Kartierung an einer ausgezeichneten Stelle und verwendet das danach entstandene Kartenmodell als Referenzkarte, so dass der Nullpunkt der Karte einer Startposition entspricht. Weitere Startpositionen kann man dann sehr genau durch Markierung der Positionen relativ zur Startposition erhalten.

Besitzt man kein Vorwissen über die initiale Position, muss man auf die globale Selbstlokalisation zurückgreifen (siehe Kapitel 6).

Als Alternative kann man, wenn Karte und Unsicherheitsregion gegeben sind, auch einen nahezu exakten Standort bestimmen, indem man jeden in der angenommenen Fehlerregion erzeugten Partikel als hypothetischen Standort des Roboters annimmt

und dann mittels eines Sensormodells diesen Standort bewertet. Der Partikel mit der besten Bewertung repräsentiert dann den wahrscheinlichsten Standort.

Wahl des Zielpunktes

Die Wahl des Zielpunktes kann vom Benutzer durch eine Mensch-Maschine-Schnittstelle (siehe auch Kapitel 9.2.3) erfolgen. Dabei ergibt sich die Problematik, dass man bei oder nach einem Kartierungsschritt in der Karte spezifische Konfigurationen festlegen muss, welche die Ziele identifizieren. Diese Festlegung ist manuell sehr zeitintensiv und fehlerhaft und sollte im Rahmen einer automatischen Kartierung erfolgen.

Im Zuge der globalen Selbstlokalisation erfolgt die Auswahl der Zielpunkte durch den Roboter automatisch aufgrund der aktuell explorierten Karte und speziellen Konfigurationen zwischen exploriertem und unexploriertem Gebiet. Siehe dazu die ausführliche Darstellung in Kapitel 6.4.

5.2.2 Optimalitätskriterien

Die Generierung eines Pfades von der Start- zur Zielkonfiguration unterliegt Optimalitätskriterien. Je nach Wahl dieses Kriteriums ergibt sich ein anderer bester Pfad. Optimalitätskriterien können die folgenden sein:

Gesamtlänge des Pfades
: Minimiere die vom Roboter traversierte Pfad- oder Weglänge vom Startpunkt zum Zielpunkt.

Fahrzeit
: Minimiere die Fahrzeit des Roboters, abhängig von Hindernissen. Minimale Fahrzeit heißt nicht immer kürzester Pfad vom Startpunkt zum Zielpunkt.

Gesamtrotation
: Berechne einen rotations-minimalen Pfad vom Start- zum Zielpunkt wobei die Summe der Rotationen auf dem Pfad minimiert wird. Dies ist sinnvoll, da Rotationen die Hauptfehlerquelle bei der Bewegung sind.

Abstand zu Hindernissen
: Berechne einen regulären Weg vom Startpunkt zum Zielpunkt mit einer minimalen Anzahl von Hindernissen bzw. einem maximalen Abstand von allen Hindernissen.

Im folgenden wird immer die Gesamtlänge des Pfades als Optimalitätskriterium angenommen.

5.2.3 Pfadplanungsansätze

Dynamische Eigenschaften und zeitliche Abfolgen zur Planung eines Pfades sind unberücksichtigt gelassen. Eine vorhandene Karte wird zu einem Zeitpunkt als aktuell angesehen und mit ihr Pfade geplant. Der Roboter stößt außerdem nicht mit anderen Hindernissen zusammen, so dass keine mechanische Interaktion berücksichtigt werden muss. Es werden auch keine kinematischen Beschränkungen für den Roboter $\mathcal{A}$ angenommen, so dass er sich im Arbeitsraum frei bewegen kann. In der Literatur [132] kann man drei klassische Vertreter von globalen Pfadplanungsalgorithmen ausmachen, welche im folgenden näher erklärt werden.

Roadmaps
In der Karte wird ein Netzwerk oder ein Graph möglicher befahrbarer Wege berechnet und ähnelt dem Konzept der Straßenkarten.

Zellzerlegung
Die Karte wird in eine Menge von Zellen unterteilt, so dass man aufgrund der Nachbarschaft der Zellen einen Graphen modellieren kann.

Potentialfelder
Eine Bewertungsfunktion wird über die Kartenrepräsentation gelegt mit dem Startpunkt als Maximum und dem Zielpunkt als Minimum der Bewertung. Anschließend wird ein Weg aus Konfigurationen mit absteigenden Bewertungen bestimmt.

Allen hier vorgestellten Pfadplanern ist gemeinsam, dass sie eine Menge von Teilzielen berechnen, welche dem Roboter übergeben werden. Kinematische Constraints werden dadurch in die dynamische Hindernisvermeidung oder in die Regelung zum Erreichen von Teilzielen [71, 113, 174] verlagert.

Roadmap-Methoden

Betrachte nun den Ansatz zur Pfadplanung mittels Roadmaps. Dazu benötigt man zunächst grundlegende Definitionen:

Definition 5.4 (Geometrischer Graph) *Ein geom. Graph ist ein Graph* $G = (\mathcal{V}, \mathcal{E})$ *mit einer Eckenmenge* $\mathcal{V}$ *und einer Kantenmenge* $\mathcal{E}$*. Dabei gilt: jedes* $v \in \mathcal{V}$ *ist ein Punkt der euklidischen Ebene* $\mathbb{E}^2$ *und jedes* $e \in \mathcal{E}$ *ist ein Drei-Tupel* $e = (v_1, v_2, c)$ *mit* $v_1, v_2 \in \mathcal{V}$ *und einer Kurve* $c \subset \mathbb{R}^2$ *mit den Endpunkten* v_1, v_2 *und der Länge* $l(c)$*.*

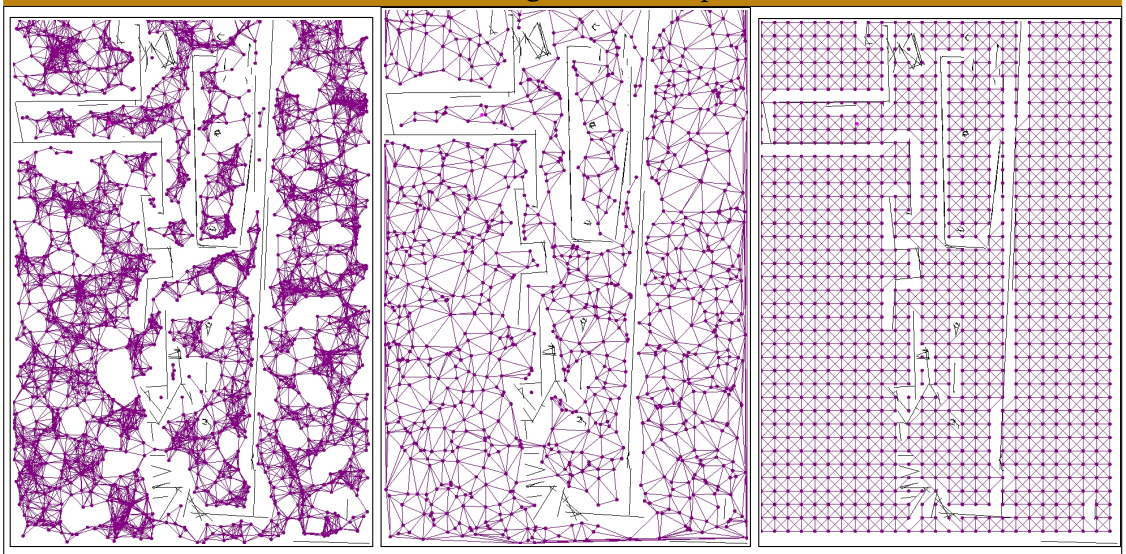
Definition 5.5 (Pfad, Weg) *Ein Pfad oder Weg* $W \subset G$ *eines geometrischen Graphen ist eine Sequenz von* w *Kurven* $(c_1, \ldots c_w)$ *und für jede* $c_i = (v_{i,1}, v_{i,2})$ *gilt* $v_{i,2} = v_{i+1,1}$*.*

Ein Pfad W kann mit einem Bewertungskriterium bewertet werden. Besteht das Bewertungskriterium aus der geometrischen Länge eines Pfades, kann man die Länge $l(W)$ eines Pfades $W = (c_1, \ldots c_w)$ durch die Gleichung $l(W) = \sum_{i=1}^{w} l(c_i)$ bestimmen.

Definition 5.6 (Roadmap) *Eine Roadmap* $\mathbb{R}$ *ist ein Netzwerk eindimensionaler Kurven repräsentiert durch einen geometrischen Graph, welcher zweidimensionale Positionen des freien Raums* $\mathcal{C}_{\text{free}}$ *des Arbeitsraums des Roboters* $\mathcal{A}$ *repräsentiert.*

Die Pfadplanung wird durch Roadmaps auf das Planen von Pfaden auf der Roadmap reduziert. Wegstücke werden durch ihre Länge begrenzt. Ist ein Wegstück länger als erlaubt, wird es in zwei Wegstücke zerlegt. Dies soll die Sicherheit erhöhen mit der sich der Roboter auf dem Weg fortbewegt und stellt eine Obergrenze in der Distanz der Fahrstrecken dar.

Abbildung 46: Roadmaps



Verwendete Roadmaps - randomisierte Roadmap (links), Delaunay-Roadmap (Mitte) und Grid-Roadmap (rechts)

Normalerweise müssen Roadmaps bestimmten Stetigkeits-Bedingungen genügen, da diese durch mobile Roboter traversiert werden sollen, was mittels Regelung geschieht. Da jedoch dem mobilen Roboter eine gewissen Freiheit zum Ausweichen von Hindernissen gegeben wird, können hier im ersten Schritt mehrere einfache Algorithmen angewendet werden um Roadmaps zu erzeugen, welche die Stetigkeit außer acht lassen. Ein Beispiel dafür liefert Abbildung 46.

Randomisierte Roadmap

In den freien Raum $\mathcal{C}_{\text{free}}$ der Umgebung werden randomisiert zweidimensionale Punkte gelegt und danach alle Punkte innerhalb eines zu wählenden Radius r miteinander verbunden [117–119, 131], wenn das Verbindungssegment kein Hindernis schneidet.

Delaunay-Roadmap
Wiederum werden randomisiert Punkte in den freien Raum $\mathcal{C}_{\text{free}}$ der Karte gelegt, jedoch mittels eines Delaunay- Graphen [121] verbunden.

Voronoi-Roadmap
Alternativ zur Delaunay-Roadmap kann man auch ein verallgemeinertes Voronoi-Diagramm für die Umgebungsrepräsentation berechnen [121].

Grid-basierte Roadmap
Die Umgebung wird durch ein gleichförmiges Gitter modelliert als Graph repräsentiert. Jede Ecke des Gitters ist mit anderen benachbarten Ecken verbunden.

Visibility-Graph
Ein Graph bestehend aus der Start- und Zielkonfiguration, den Ecken der Hindernisse in der Karte als Eckenmenge und den Kanten, welche die Sichtbarkeit zweier Ecken repräsentieren, wird als Roadmap-Repräsentation verwendet [83].

Es sind natürlich noch weitere Ansätze zur Generierung von Roadmaps denkbar, etwa der generalisierte Voronoi-Graph [47–49]. Die Pfadplanung hat die Aufgabe, einen regulären Weg von der Start- zur Zielposition zu planen. Unter der Annahme, dass alle Kurven der Roadmap aus Segmenten bestehen. d.h. Kurven durch eine Menge von Segmenten approximiert wurden, kann man für einen gültigen aber noch nicht regulären Weg eine **Fehlerhülle** spezifizieren. Diese hängt vom Roboterradius r_{robot} und dem Sicherheitsdistanz d_{save} ab.

Definition 5.7 (sicherer Weg) *Ein sicherer und damit regulärer Weg W_s ist ein Weg, bei welchem jedes Segment w_i einen Abstand von mindestens $d_{min} = r_{robot} + d_{save}$ zu einem Hindernis hat, welches in der Karte verzeichnet ist.*

Definition 5.8 (Fehlerhülle) *Eine Fehlerhülle F zu einem Weg $W = \{c_0, \ldots, c_n\}$ ist eine Liste von n Rechtecken $F = \{f_0, f_1, \ldots, f_n\}$, wobei um jedes Wegstück c_i ein Rechteck f_i mittels d_{max} gelegt wird.*

Exemplarisch wird im folgenden für eine feature-basierte Karte der Aufwand zur Bestimmung eines sicheren Weges von einem Anfangs- zu einem Endpunkt hergeleitet. Im ersten Schritt wird dazu die Berechnung einer probabilistischen Roadmap beschrieben.

Korollar 5.9 (Aufwand zur Berechnung einer Roadmap) *Gegeben sei eine feature-basierte Karte m_{feat} der Einsatzumgebung mit m Kartenfeatures. Dann beträgt die Zeitkomplexität zur Erzeugung einer randomisierten Roadmap nach Algorithmus 5.1 $\mathcal{O}(n \cdot (\log^2 n + \cdot k \cdot m))$ und die Speicherkomplexität $\mathcal{O}(n \cdot (k + \log n))$. Dabei bezeichnet n die Anzahl der erzeugten Ecken der Roadmap und k die maximale Anzahl benachbarter Ecken.*

Algorithmus 5.1: Probabilistische Roadmap-Berechnung

Aufruf: **probabilistic-roadmap()**
Input: Anzahl von n Punkten, Radius r, Karte m_{feat}
Output: Roadmap R

```
1  V = ∅, E = ∅, nn = 0, n_max = 5 · n.
2  while {nn < n AND nn < n_max} do
3     Generiere zufällig einen Punkt p.
4     for all {g ∈ m_feat} do
5        if {p liegt im freien Bereich} then
6           if d(p, g) > d_min then
7              Füge p in V ein.
8              nn++
9           end if
10       end if
11    end for
12 end while
13 Erzeuge effiziente Anfragestruktur für V.
14 for all v_i ∈ V do
15    Hole k Nachbarn von v_i.
16    for all {k Nachbarn von v_i} do
17       Erzeuge Segment s = v_i v_j (über) für einen Nachbarn v_j.
18       for all g ∈ m_feat do
19          if { ∩(s, g) = ∅ AND d(s, g) > d_save} then
20             Füge s in E ein.
21          end if
22       end for
23    end for
24 end for
```

Beweis: Zunächst müssen n Ecken der Roadmap erzeugt werden. Dazu wird angenommen, dass eine feature-basierte Kartenrepräsentation m_{feat} vorhanden ist. Beim Test, ob der erzeugte Punkt innerhalb des freien Raumes liegt, wird zusätzlich ein Constraint über den Abstand d_{min} dieses Punktes von den Kartenobjekten berücksichtigt. Ein Test, ob ein Punkt innerhalb eines freien Bereiches und d_{min} von einem Hindernis entfernt ist, kann in $\mathcal{O}(m)$ entschieden werden, da jedes Kartenobjekt betrachtet werden muss. Die Erzeugung der Punkte benötigt also $\mathcal{O}(n \cdot m)$ Zeitschritte. Der Algorithmus terminiert, falls nach einer maximalen Anzahl von n_{max} Versuchen nicht die geforderte Anzahl von n Punkten ermittelt werden konnte, die obige Bedingungen erfüllt.

Mit Hilfe eines Point-Dictionaries [150, 153] mit Aufbaukosten $\mathcal{O}(\log^2 n)$ kann die Anfrage für jeden Punkt nach den nächsten Nachbarn in einem bestimmten Radius r in $\mathcal{O}(k + \log^2 n)$ entschieden werden. Sei dabei k die maximale Anzahl von Nachbarn

zu einem Punkt p. Für jede mögliche der k Verbindungen muss zusätzlich untersucht werden, ob diese ein Hindernis aus m_{feat} schneidet.

Betrachte dazu ein einzufügendes Segmentfeature s. Für dieses wird in $\mathcal{O}(m)$ Zeitschritten getestet, ob es ein Kartenfeature schneidet und ob es gleichzeitig eine Distanz größer d_{min} von jedem Kartenfeature besitzt. Die Tests lassen sich in konstantem Aufwand bewältigen. Für k zu testende Verbindungen eines Punktes benötigt man also $\mathcal{O}(k \cdot m)$ Zeitschritte.

Insgesamt ergeben sich also $\mathcal{O}(n \cdot m) + \mathcal{O}(\log^2 n) + \mathcal{O}(n \cdot (k + \log^2 n + \cdot k \cdot m))$ Zeitschritte, also $\mathcal{O}(n \cdot (\log^2 n + \cdot k \cdot m))$ Zeitschritte.

Speicheraufwand von $\mathcal{O}(n) + \mathcal{O}(n \log n)$ wird für die Verwaltung der Ecken des Graphen benötigt. Weiter werden maximal $\mathcal{O}(nk)$ Kanten erzeugt, wodurch sich ein Speicheraufwand von $\mathcal{O}(n \cdot (k + \log n))$ ergibt. □

Der zweite Schritt besteht in der Berechnung der kürzesten Wege oder Trajektorien innerhalb einer Roadmap. Da nicht alle möglichen Positionen in der Karte auf der Roadmap liegen, ist es erforderlich vom aktuellen Startpunkt des Roboters zum nächsten Knoten v der Roadmap einen Weg zu planen, ebenso vom Zielpunkt zu einem nächsten Knoten w der Roadmap. Schließlich besteht die Wegeplanung nur noch in der Berechnung eines kürzesten Weges von v nach w innerhalb der Roadmap. Wichtigster Suchalgorithmus der dabei zum Einsatz kommt, ist der A^*-Algorithmus [100]. Man erhält als Ausgabe eine Menge von Teilzielen, welche durch die Endpunkte der Wegstücke der Trajektorie repräsentiert werden.

Algorithmus 5.2: Pfadplanung auf Roadmaps

Aufruf: **compute-roadmap()**
Input: Startposition q_{start} und Zielposition q_{end}, m_{feat}
Output: Pfad W_s vom Start zum Ziel

1 Berechne Roadmap R.
2 Berechne kürzesten Weg s_1 von q_{start} zu einem Punkt v auf R.
3 Berechne kürzesten Weg s_2 von q_{end} zu einem Punkt w auf R.
4 **if** {s_1 und s_2 sind befahrbar} **then**
5 Berechne Pfad von v nach w in der Roadmap R.
6 Ausgabe des Gesamtpfades $W_s = \overline{q_{start}v} \cup \overline{vw} \cup \overline{wq_{end}}$.
7 **else**
8 Ausgabe von $W_s = \emptyset$.
9 **end if**

Korollar 5.10 (Komplexität der Pfadplanung auf Roadmaps) *Die Zeitkomplexität zur Bestimmung eines regulären Weges W im Roadmap-Algorithmus 5.2 bei einer geplan-*

ten Roadmap R *mit* n *Knoten einer Segmentkarten-Repräsentation* m_{feat} *mit* m *Features beträgt* $\mathcal{O}(n^2 \cdot m)$. *Die Speicherkomplexität beträgt* $\mathcal{O}(n)$.

Beweis: Die Bestimmung eines zu q_{start} nächsten Knotens v des Graphen benötigt $\mathcal{O}(n)$ Aufwand, ebenso die Bestimmung von w. Zusätzlich muss für alle $g \in m_{feat}$ getestet werden, ob $\overline{vq_{start}}$ bzw. $\overline{wq_{end}}$ sich nicht schneiden bzw. ob die minimale Distanz d_{save} eingehalten wird. Dies verursacht einen Aufwand von $\mathcal{O}(m)$ Zeitschritten.

Die Anwendung des A*-Algorithmus zur kürzesten Wege Suche auf dem Graphen $G = (\mathcal{V}, \mathcal{E})$ mit Startecke v und Endecke w berechnet sich hier in $\mathcal{O}(n^2)$. Die Zeitkomplexität summiert sich mit der Komplexität zur Berechnung der zusätzlichen Segmente zu $\mathcal{O}(n^2 + m)$.

Die Speicherkomplexität des berechneten Weges W bestimmt die Roadmap R und ist aus $\mathcal{O}(n)$. □

Zellzerlegungs-Ansätze

Eine populäre Methode der Pfadplanung ist eine Zerlegung der Karte in Zellen und damit in einen Verbindungs-Graphen und anschließender Suche nach einem Pfad in diesem Graphen [87, 116, 132].

Definition 5.11 (Zellzerlegung) *Eine Zellzerlegung* $\mathcal{Z}$ *zerlegt den freien Raum des Konfigurationsraumes* $\mathcal{C}_{free}$ *in eine Menge von* n *Regionen, die Zellen genannt werden.*

Definition 5.12 (Verbindungs-Graph) *Der Verbindungs-Graph* $G_{Con} = (V_{Con}, E_{Con})$ *repräsentiert die Karte* m_g. *Dabei stehen die Knoten des Graphen für einzelne, durch eine Zerlegung entstandene, Zellen. Die Kanten zwischen zwei Knoten repräsentieren eine direkte Nachbarschaft der Zellen. Existiert keine Kante zwischen zwei Knoten, sind die entsprechenden Zellen auch nicht benachbart.*

Einen Verbindungsgraphen kann man in zwei Schritten erzeugen. Im ersten Schritt berechnet man eine Zellzerlegung der Umgebungskarte. Man unterscheidet bei Zerlegungen zwischen exakten Zerlegungen (z.B. Sichtbarkeitszellenzerlegung) und approximativen Zerlegungen (z.B. Quadtree) der Karte. Bei einer **approximativen Zerlegung** wird die Karte mit dem Verbindungs-Graph durch eine Teilmenge der Karte repräsentiert, währenddessen bei einer **exakten Zerlegung** der Verbindungs-Graph die ganze Karte repräsentiert. Typische Vertreter von Zellzerlegungen sind die folgenden:

Trapezoidzerlegung
Eine vertikale Sweepline fügt vertikale Kanten von jeder Ecke eines Hindernisses zu einem Hindernis oder dem Kartenrand an [132].

Sichtbarkeitszellenzerlegung
Die Karte wird in Bereiche gleicher Sicht zerlegt, indem Stahlen von jeder Ecke

über jede Reflexecke in die Karte eingefügt werden. Jede dann entstandene Sichtbarkeitszelle repräsentiert eine Menge von Konfigurationen, welche eine gemeinsame Anzahl von Ecken sehen (siehe auch Kapitel 6) [87,114,116].

Grid-Zerlegung
Der freie Raum der Karte wird durch ein Gitter repräsentiert, zum Beispiel durch einen Quadtree [132]. Die Gitterzellen sind dabei unterschiedlich groß und repräsentieren freie zusammenhängende Konfigurationen.

Im zweiten Schritt bestimmt man für die Zellenzerlegung einen Verbindungsgraphen. Jede Zelle wird durch ihren Mittelpunkt als Knoten eines Verbindungsgraphen angesehen. Eine Kante zwischen zwei Knoten bekommt man dadurch, dass man für jeden Knoten die entsprechende Zelle bestimmt und dann benachbarten Zellen durch Segmente zwischen den Mittelpunkten der Zellen verbindet. Diese Verknüpfungsheuristik funktioniert nur für konvexe Zellen, für andere Zelltypen muss man weitere Hilfspunkte einfügen. Dabei wird diese Anzahl durch die Lösung des Art-Gallery-Problems für jede Zelle determiniert [162]. Analog zur Roadmap-Methode hat man nun einen Graphen zur Verfügung, auf welchem man analog Pfade von Start- zu Zielkonfigurationen berechnen kann. Der Berechnungsaufwand hängt von der Zellzerlegungsvariante ab.

Algorithmus 5.3: Pfadplanung mittels Zellzerlegungen

Aufruf: **cell-decomposition()**
Input: Start- und Zielposition, $\mathcal{C}_{\text{free}}$
Output: Pfad path=$(p_1, \ldots, p_n)$ als Menge von Konfigurationen vom Start zum Ziel

1 Berechne die gewählte Zerlegung $\mathcal{Z}$.
2 Bestimme den Verbindungsgraphen G_{con}.
3 Bestimme Anfangsknoten v und Endknoten w.
4 Berechne Pfad $\overline{path}$ von v nach w in G_{con}.
5 Berechne den Pfad $path$ ohne die extremalen Knoten von $\overline{path}$ durch Einfügen von Segmenten zwischen den Zellmittelpunkten.
6 Verbinde Start- und Zielpunkt mit den zugehörigen Rändern der berechneten Nachbarzelle.
7 Ausgabe des Gesamtpfades $path$.

Für eine detaillierte Auskunft sei der Leser auf die entsprechende Literatur verwiesen.

Potentialfeld-Metode

Die Potentialfeld-Methode wurde von Andrews, Hogan und Khatib erstmals vorgestellt [12, 120]. Die Idee der Methode besteht darin, ein künstliches Potentialfeld U anzunehmen. Der Roboter steht dann unter dem Einfluß dieses Potentialfeldes und soll vom maximalen Potential (dem Startpunkt) zum minimalen Potential (dem Zielpunkt) bewegt werden. Ein Zustandsübergang findet dann nur von Konfigurationen q mit $U(q)$ zu Konfigurationen r mit $U(r)$ statt, wenn $U(r) \leq U(q)$ ist. Das Potentialfeld wird

über den gesamten Konfigurationsraum $\mathcal{C}$ definiert und setzt sich aus mehreren Potentialen zusammen.

Definition 5.13 (Repulsives Potential) *Ein repulsives oder abstoßendes Potential U_{rep_i} stößt den Roboter vom Hindernissen B_i weg.*

Demzufolge werden um Hindernisse repulsive Potentiale gelegt. Die Idee, ein repulsives Potential zu benutzen liegt darin, dass man um ein erkanntes Hindernis eine Potentialbarriere legt, so dass der Roboter gewisse Konfigurationen aufgrund des Potentials nicht annehmen kann. Außerdem soll ein repulsives Potential keinen Effekt bei größerer Entfernung um das Hindernis haben, d.h. das Potential wirkt nur ab einer gewissen Distanz.

Definition 5.14 (Attraktives Potential) *Die attraktiven Potentiale U_{att} ziehen den Roboter in Richtung Ziel.*

Mit zunehmender Distanz zum Zielpunkt wird das Potential geringer. Demzufolge kann man das Gesamtpotential definieren als

Definition 5.15 (Potentialfunktion) *Die Potentialfunktion $U : \mathcal{C} \rightarrow \mathbb{R}$ bewertet die Konfigurationen des Konfigurationsraumes $\mathcal{C}$ als Summe von attraktiven und repulsiven Potentialen. Für eine einzelne Konfiguration q gilt*

$$U(q) = U_{att}(q) + \sum_{i=1}^{m} U_{rep_i}(q)$$

Nach Algorithmus 5.4 kann exemplarisch für eine Gitterkarten-Repräsentation der Umgebung ein Weg von q_{start} nach q_{end} berechnet werden. Die Zeit- und Speicherkomplexität hängt dabei linear von der Anzahl der Gitterzellen ab.

Algorithmus 5.4: Potentialfeld-Algorithmus

Aufruf: **potential-field()**
Input: Start- und Zielkonfiguration, Konfigurationsraum $\mathcal{C}$.
Input: Potentialfunktion $U(q)$ für $\mathcal{C}$
Output: path=$(p_1, \ldots, p_n)$ repräsentiert durch eine Menge von Konfigurationen.
1 o.b.d.A. diskretisiere den Konfigurationsraum in ein Gitter G.
2 Bestimme für jeden Gitterpunkt g_{ij} das Potential u_{ij}.
3 Heuristische Suche im Gitter vom Start- zum Zielpunkt, wobei in jedem Schritt nach einem gleichen oder geringerem Potential gesucht wird.
4 Rückgabe von `path`

Diese Methode hat einen entscheidenden Nachteile. Problematisch ist die Suche nach einer entsprechenden Bewertungsfunktion für den Konfigurationsraum. In der Regel sind Bewertungsfunktionen mit lokalen Minima ausgestattet, so dass man Heuristiken entwerfen muss aus diesen zu entkommen [132].

5.3 Positionsverfolgung und Positionskorrektur

Hat man im letzten Abschnitt einen Pfad von einer Start- zu einer Zielkonfiguration berechnet, geht es in diesem Abschnitt darum, mittels der gegebenen Karte und der generierten Sensorinformation den traversierten Weg [4,38] des Roboters zu verfolgen, wenn sich der Roboter dazu schon geeignet bewegt.

Verfolgt man die Position des Roboters und vergleicht die Position mittels des berechneten optimalen Pfades, kann man dann eingreifen, wenn sich Bewegungsfehler ereignet haben, so dass man unter Umständen gezwungen ist, den Pfad zur Zielposition neu zu planen. Die Planung kann jedoch in der Regel nicht in Echtzeit erfolgen. Dies spiegelt auch das Verhalten eines Menschen in fremden Umgebungen wieder, plant man doch einen Weg, versucht den Weg einzuhalten und korrigiert durch Straßenkarten oder Hinweise diesen, wenn man davon abgekommen ist.

Die Problematik soll hier differenziert werden in das Positionsverfolgungs-Problem und das Positionskorrektur- oder Update-Problem. Das Problem der Positionsverfolgung lässt sich folgendermaßen definieren:

Problem 5.16 (Positionsverfolgung, Positionsschätzung) *Ein Roboter mit einer Konfiguration* $\overline{x_{k-1}} = (x, y, \theta)^T$ *bewegt sich durch eine gemessene Bewegung* $a_k = (\Delta d_k, \Delta\theta_k)^T$ *zu einem Zeitpunkt* k *an eine neue Konfiguration* x_k*. Diese repräsentiert aufgrund von Bewegungsfehlern nicht die tatsächliche Position des Roboters. Mit Hilfe von Sensordaten* s_{k-1} *aufgenommen an Position* $\overline{x_{k-1}}$ *und Sensordaten* s_k *an Position* x_k *soll eine Positionsschätzung* $\overline{x_k}$ *den Bewegungsfehler eliminieren.*

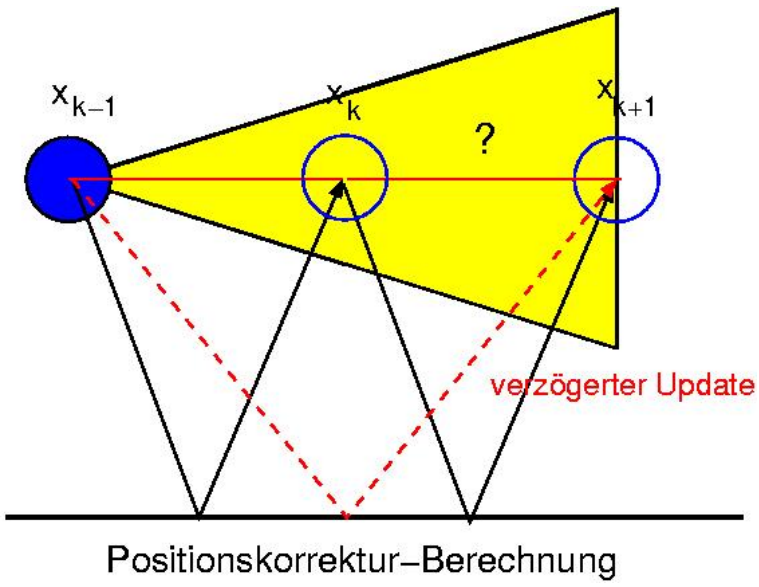
Diese Positionsschätzung berechnet einen Fehlervektor $e_k = \mid x_k - \overline{x_k} \mid$ zu einem Zeitpunkt k. Unter idealen Bedingungen kann diese Berechnung mittels eines Algorithmus in Echtzeit geschehen, so dass dem Roboter ohne Zeitverzögerung die neue Position $\overline{x_k}$ zur Verfügung steht und diese für die Korrektur der Position des nächsten Zeitschrittes herhalten kann.

Unter nicht-idealen Bedingungen ist diese Information nicht sofort für den Roboter zugänglich, so dass sich das folgende Problem (siehe Abbildung 47) ergibt:

Problem 5.17 (Positionskorrektur, Update-Problem) *Ein mobiler Roboter traversiert einen gegebenen Pfad und besitzt zu einem bestimmten Zeitpunkt* k *einen Fehlervektor* e_{k-1} *, welcher die Differenz der geglaubten und der ermittelten Position zum Zeitpunkt* $k-1$ *repräsentiert und welcher den Roboter erst zum Zeitpunkt* $k+1$ *erreicht. Das Problem besteht darin, dem Roboter zum Zeitpunkt* $k+1$ *seine abweichende Position mitzuteilen, so dass dieser stabil seinen geplanten Weg zum Zielpunkt fortsetzen kann.*

Es wird zunächst die Fragestellung der Positionsverfolgung behandelt. Anschließend wird auf das Problem der Positionskorrektur eingegangen.

Abbildung 47: Verzögerte Positionskorrektur



In einer verteilten Anwendung kann es zu Verzögerungen der Positionskorrektur kommen, so dass der Roboter erst zum Zeitpunkt $k+1$ die Korrekturdaten des Zeitpunktes $k-1$ erhält. Wie soll sich dann der Roboter zum Zeitpunkt $k+1$ verhalten?

5.3.1 Positionsverfolgung

Die Verwendung reiner Odometriedaten reicht normalerweise nicht aus, um die Roboterposition über einen längeren Zeitraum richtig zu bestimmen, da sich die Fehler im obigen Differenzvektor akkumulieren. Deshalb werden in der Literatur vielfältige Verfahren vorgeschlagen, welche man in die beiden folgenden Klassen einteilen kann:

Inkrementelle sensor-basierte Ansätze
: Die neue Position $\overline{x_k}$ des Roboter wird aufgrund der alten Position $\overline{x_{k-1}}$ mit Sensordaten s_{k-1} und den neuen Sensorinformationen s_k berechnet. Zugehörige Ansätze sind die Koppelnavigation oder auch Scan-Scan-Matching-Verfahren [91, 92, 95, 234].

Modell-basierte Ansätze
: Die neue Position des Roboter wird aufgrund der alten Position $\overline{x_{k-1}}$, den neuen Sensordaten s_k und der Karte m_g bestimmt. Zugehörige Ansätze sind die Markov-Lokalisation [69], Monte-Carlo-Lokalisation [68, 70, 209, 210].

Inkrementeller sensor-basierter Ansatz zur Positionsverfolgung

Sensor-basierte Verfahren zeichnen sich dadurch aus, dass kein Modell der Umgebung verwendet werden muss, welches unter Umständen eine weitere Fehlerquelle bei der Bestimmung der wahren Roboterposition sein kann. Das Problem wird in der Literatur als Scan-Matching-Problem beschrieben, soll hier aus Übersichtlichkeitsgründen als Scan-Scan-Matching Problem firmieren:

Problem 5.18 (Scan-Scan-Matching) *Gegeben seien zwei Scans s_{k-1} und s_k und deren geschätzte Aufnahmepositionen $\overline{x_{k-1}}$ und x_k. Das Problem besteht darin, eine Position $\overline{x_k}$ zu bestimmen, so dass $\{(s_{k-1}, \overline{x_{k-1}}), (s_k, \overline{x_k})\}$ im Sinne der Kartierung lokal konsistent ist. Gesucht ist also die beste Schätzung der Aufnahmeposition $\overline{x_k}$ für den Scan s_k.*

Die Problemlösung wird dabei in drei Schritten realisiert. Die Konfiguration wird als Ausgangspunkt für zwei verschiedene Positionsschätzungen o_k und p_k verwendet. Diese werden dann zu einer Gesamtschätzung $\overline{x_k}$ verschmolzen.

Im ersten Schritt werden nur Sensordaten mit ihren geschätzten Aufnahmepositionen $o_k = x_k$ betrachtet und eine Positionskorrektur mittels der Sensordaten errechnet. Dazu wird die lokale Positionsdifferenz der Odometriedaten mit der zuletzt bekannten Position des Roboters als Schätzung für die Roboterposition berechnet und mittels eines Scan-Scan-Matching-Ansatzes die beste Schätzung $\overline{o_k}$ mit Positionsfehler $\Sigma_{\overline{o_k}}$ berechnet.

Der Nachteil der Lösung, würde man nur den ersten Schritt verwenden, besteht darin, dass die Schätzung aufgrund der Odometriedaten schon sehr schlecht sein kann. Zudem kann die Lösung eines Scan-Matching-Verfahrens stark von der Schätzung differieren und unter Umständen fehlerhaft sein.

Im zweiten Schritt wird deshalb mittels Koppelnavigation zusätzlich ein probabilistisches Bewegungsmodell aufgrund der lokalen Positionsdifferenzen der Odometriedaten verwendet um den absoluten und relative Fehler des Zustandsübergangs zu modellieren. Die Koppelnavigation (engl.: dead-reckoning) akkumuliert die Positionsdaten aller bisheriger Zeitpunkte.

Würde man nur diesen zweiten Schritt zur Positionsschätzung verwenden, würden die Fehler sich die Fehler stark akkumulieren, was zu unbrauchbaren Ergebnissen führen wurde [92].

Bei der Koppelnavigation wird die Berechnung der Roboterposition und Orientierung aus den Bewegungsdaten der Räder (auch Odometrie) vollzogen. Dabei können abhängig vom kinematischen Modell des Roboters ein oder mehrere Räder in die Berechnung einfließen. Bewegt sich das Fahrzeug, so wird in regelmäßigen Abständen die Fahrzeugposition aktualisiert. Hierzu wird der **zurückgelegte Weg** Δd und die

Änderung in der Orientierung $\Delta\theta$ seit dem letzten Zeitpunkt gemessen und mit der aktuellen Fahrzeugposition verrechnet.

Die Roboterkonfiguration $\overline{x_{k-1}}$ zum diskreten Zeitpunkt $k-1$ wird durch die Eingabe $a_k = (\Delta d_k, \Delta\theta_k)^T$ allgemein mittels des kinematischen Modells aktualisiert zu

$$p_k = f(\overline{x_{k-1}}, a_k) \tag{5.1}$$

Für die Modellierung des Positionsfehlers wird angenommen, dass die Fehler in der Positionsbestimmung durch Koppelnavigation normalverteilt sind, ebenso die gemessene Entfernung Δd und Rotation $\Delta\theta$. Dabei geht man davon aus, dass der zurückgelegte Weg Δd nicht mit der Änderung in der Orientierung $\Delta\theta$ korreliert, also ist $\sigma_{\Delta d\Delta\theta} = 0$. Bewegt sich der Roboter um a_k, so wird die neue geschätzte Roboterposition mittels Koppelnavigation berechnet durch

$$\overline{p_k} = f(\overline{x_{k-1}}, \overline{a_k}) \tag{5.2}$$

mit einer resultierenden Kovarianzmatrix $\Sigma_{\overline{p_k}}$.

Dritter Schritt zur Lösung des Scan-Scan-Matchings besteht darin, die beiden berechneten Positionsschätzungen $\overline{p_k}$ und $\overline{o_k}$ mittels eines Kalman-Filters [148, 235] zu fusionieren. und es ergibt sich:

$$\overline{x_k} = (\Sigma_{\overline{o_k}}^{-1} + \Sigma_{\overline{p_k}}^{-1})^{-1}(\Sigma_{\overline{o_k}}^{-1}\overline{o_k} + \Sigma_{\overline{p_k}}^{-1}\overline{p_k}) \tag{5.3}$$

$$\Sigma_{\overline{x_k}} = (\Sigma_{\overline{o_k}}^{-1} + \Sigma_{\overline{p_k}}^{-1})^{-1} \tag{5.4}$$

Als Scan-Scan-Matching-Verfahren kommen folgende Verfahren in Betracht:

Cox-Algorithmus
Das Verfahren von Cox [52] überdeckt einen Linienmodell mit dem aktuellen Scan. Hierzu wird jedem Scanpunkt eine Linie des Modells zugeordnet. Aus dieser Zuordnung lässt sich dann die Verschiebung und Verdrehung des Scans s_k gegenüber des Linienmodells des Scans s_{k-1} bestimmen.

IDC-Algorithmus
Die Idee des Verfahrens von Lu [141–143] ist es, Scanpunkte des aktuellen Scans s_k den Punkten des Referenzscans s_{k-1} zuzuordnen. Dabei wird dann eine Fehlersumme über die Zuordnungen minimiert, um so die Verschiebung und Verdrehung der Scans zu bestimmen. Kern des IDC-Algorithmus ist die Anwendung der Heuristiken „nächster Punkt" und „gleiche Entfernung".

Kombiniertes Verfahren CSM
Das Verfahren von Gutmann [92] nutzt in polygonalen Umgebungen die bessere Laufzeit des Cox und in nicht-polygonalen Umgebungen die Universalität des IDC-Algorithmus.

IDC-Sektor

Eine Variante des IDC-Algorithmus ist der IDC-Sektor [20], bei welchem einzelne Sektoren zugeordnet werden. Der Scan s_k wird dabei in mehrere überlappende Sektoren unterteilt. Für jeden Sektor i wird ein Fehlerwert e_i berechnet und der Sektor gelöscht, falls der Fehlerwert einen Schwellwert übersteigt. Nur die übriggebliebenen Sektoren werden für das Matching mit dem Referenzscan s_{k-1} verwendet.

Probabilistisches Scanmatching

Das in Kapitel 3.2 vorgestellte Verfahren zur inkrementellen Kartierung kann auch als Scan-Scan-Matching verwendet werden, indem der Referenzscan s_{k-1} mit einem Grid belegt wird und die Bewertung des aktuellen Scans äquivalent zu der Bewertung des Kartierungsverfahrens geschieht. Anstatt eines Gradientenabstiegsverfahrens wird allerdings der beste Partikel einer bewerteten Partikelmenge verwendet.

Der resultiertende Algorithmus 5.5 zur sensor-basierten Positionsverfolgung hat einen Aufwand, der vom jeweiligen Scan-Scan-Matchingverfahren abhängt, so dass der Leser auf die Resultate der Literatur verwiesen wird. Positionskorrektur durch Kalman-

Algorithmus 5.5: Inkrementelle Sensor-basierte Positionsverfolgung

Aufruf: **sensorbased-position-estimation()**
Input: Scan s_{k-1} und s_k, Position $\overline{x_{k-1}}$ mit Fehler $\Sigma_{\overline{x_{k-1}}}$ und Vektor a_k
Output: Positionsschätzung $\overline{x_k}$

1 Berechne die normalverteilte Eingabe $\overline{a_k}$ und Σ_{a_k}.
2 Berechne mittels Koppelnavigation $\overline{p_k}$ und $\Sigma_{\overline{p_k}}$.
3 Berechne mittels Scan-Scan-Matching aus s_{k-1} und s_k Position $\overline{o_k}$ und $\Sigma_{\overline{o_k}}$.
4 Fusioniere die Schätzungen mittels Kalman-Filter.
5 Rückgabe von $\overline{x_k}$ und Positionsunsicherheit $\Sigma_{\overline{x_k}}$.

Filterung kann effizient implementiert werden, solange die Fehlermodelle genau sind. Der Erfolg des Kalman-Filters hängt auch stark vom Resultat der Scanüberdeckung ab. Liefert Scan-Scan-Matching eine falsche Positionsschätzung, berechnet der Kalman-Filter ebenfalls eine falsche, fusionierte Schätzung. Dies kann zu katastrophalen Fehlern führen, so dass die Roboterposition verloren geht.

Modellbasierter Ansatz zur Positionsverfolgung

Im Gegensatz zu den obigen inkrementellen Verfahren, arbeiten modellbasierte Verfahren zusätzlich mit einer Karte m_g der Umgebung. Diesen Vorteil an Informationen erkauft man sich hingegen durch eine verminderte Berechnungszeit, so dass man geeignete Heuristiken entwickeln muss, die Laufzeit zu verringern. In der Regel wird damit der Vorteil, Sensordaten mit Kartendaten zu matchen, weitgehend zunichte gemacht.

Problem 5.19 (Modellbasierte Positionsverfolgung, Lokalisation) *Gegeben sein eine Positionsschätzung $\overline{x_{k-1}}$ mit einer Positionsunsicherheit $\Sigma_{\overline{x_{k-1}}}$ der Roboterposition zum Zeitpunkt $k-1$ in einer Karte m_g. Gesucht ist eine Positionsschätzung $\overline{x_k}$ zum Zeitpunkt k mit neuer Positionsunsicherheit $\Sigma_{\overline{x_k}}$ nach Ausführung einer Bewegung a_k und Generierung der Sensordaten s_k.*

Dieses Problem taucht in leicht veränderter Form schon in Kapitel 2 als globales Karten-Scan-Matching auf (Problem 2.2). Hier wird jedoch nur die optimale Position in einem eingeschränkten Bereich der Karte gesucht, der durch die Positionsschätzung $\overline{x_{k-1}}$ und der Positionsunsicherheit festgelegt ist.

Für gewöhnlich wird die Problemstellung in zwei Schritten gelöst: Im ersten Schritt wird eine neue Roboterposition x_k mit zugehöriger Unsicherheit durch die vorherige Positionsschätzung und der Odometriedifferenz der Positionen a_k mit Hilfe der Karte m_g und eines Bewegungsmodells abhängig von der kinematischen Modellierung bestimmt. Dabei wird die Unsicherheit in der Regel vergrößert, da die Odometriedaten fehlerbehaftet sind. In der Literatur wird dieser Schritt oft auch als Vorhersageschritt bezeichnet.

Im zweiten Schritt wird die neue Positionsschätzung $\overline{x_k}$ bestimmt. Das geschieht dadurch, dass mittels der Sensordaten s_k und der Kartenrepräsentation m_g die Positionen innerhalb des möglichen Fehlerbereichs mit einem Sensormodell bewertet werden und die am besten bewertete Position als neue Schätzung ausgewählt wird. Dieser Schritt wird als Aktualisierungsschritt bezeichnet.

In der Literatur findet sich zu dieser Problematik eine Unmenge von Artikeln, die sich auf drei probabilistische Verfahren mittels Algorithmus 5.6 zusammenfassen lassen:

Markov-Lokalisation
: Dieser Ansatz berechnet eine diskrete Approximation der Wahrscheinlichkeitsfunktion des Roboterstandorts [69]. Dazu wird ein dreidimensionales Wahrscheinlichkeitsgitter $\mathcal{L}$ verwendet, wobei jede Zelle eine Konfiguration repräsentiert. In jeder Zelle dieses Gitters befindet sich die Posteriori-Wahrscheinlichkeit dafür, dass die Zelle die aktuelle Position des Roboters ist. Im Vorhersageschritt wird das Bewegungsmodell wird durch eine abgeschnittene dreidimensionale Gauß-Verteilung beschrieben, welches wesentlich von der kinematischen Modellierung des Fehlermodells abhängt. In einem Aktualisierungsschritt wird aus dem neuen Scan die minimale Distanz zu einem Hindernis berechnet und abhängig davon die Wahrscheinlichkeit in $\mathcal{L}$ verändert. Hat sich der Roboter bewegt, wird die Wahrscheinlichkeitsverteilung in Richtung der Aktion a_k verschoben.

Monte-Carlo Lokalisation
: Diese partikel-basierte Variante der Markov-Lokalisation verwendet anstatt einem Grid eine Menge von Partikeln, um die Wahrscheinlichkeitsverteilung zu

approximieren [68, 70, 113, 209, 210]. Sensor- und Bewegungsmodell verändern sich dabei nicht. Siehe dazu auch die inkrementelle Kartierung in Kapitel 3.2.2.

CBL-Algorithmus
Die Korrelation basierte Markov-Lokalisation (CBML) erlaubt die Lokalisation mittels Integration dichter Sensordaten und stellt weiter einen dynamischen Kartenupdate und ein präziseres Sensormodell zur Verfügung [126].

Algorithmus 5.6: Modellbasierte Positionsverfolgung

Aufruf: **modelbased-position-estimation()**
Input: Positionsschätzung $\overline{x_{k-1}}$ mit Unsicherheit $\Sigma_{\overline{x_{k-1}}}$, Aktion a_k, Karte m und Scan s_k
Output: $\overline{x_k}$ mit Unsicherheit $\Sigma_{\overline{x_k}}$
1 Roboter befindet sich an Position $\overline{x_{k-1}}$
2 Berechnung mit dem Bewegungsmodell und Bewegung a_k neue Positionsschätzung x_k.
3 Bewerte die möglichen Positionen mit dem Sensormodell und Karte m
4 Rückgabe von $\overline{x_k}$ mit Unsicherheit $\Sigma_{\overline{x_k}}$.

In der Literatur sind Positionsverfolgungsansätze (auch solche, die hier nicht erwähnt wurden) extensiv studiert und miteinander verglichen worden [93,97,189]. Dabei bleibt trotzdem die Möglichkeit bestehen, dass Positionsverfolgungs-Ansätze die Roboterposition verlieren können oder aber das Problem, dass eine falsche Roboterposition berechnet wurde. Man kann letztere Problematik entschärfen, wenn man mehrere Ansätze gegeneinander laufen lässt und bei der Auswahl der tatsächlichen Roboterposition einen Mehrheitsentscheid der Verfahren zulässt. Dieses impliziert jedoch, dass man mehr Rechenkapazität zur Verfügung hat und somit auf externe Rechner ausweicht. Diese eventuelle Genauigkeit und Sicherheit erkauft man sich mit einem weiteren Problem, dem Update-Problem, welches im folgenden behandelt wird.

5.3.2 Positionskorrektur

In den meisten Ansätzen der Literatur wird davon ausgegangen, dass die obigen Positionsverfolgungs-Algorithmen in Echtzeit eine neue Positionsschätzung liefern, welche als neue Roboterposition für die Positionsregelung angesehen wird. Wenn der mobile Roboter aber nicht genügend Rechenkapazitäten zur Verfügung stellt, muss die Positionsverfolgung extern berechnet werden. Die Positionskorrektur kann dann aber aufgrund von Übertragungszeiten nicht mehr in Echtzeit ausgeführt werden. Für die Problematik gibt es eine Reihe von Lösungsmöglichkeiten:

Update während der Fahrt
Ein Korrekturvektor wird kontinuierlich zum Roboter gesandt. Wird dieser dann auf die Roboterposition addiert, ergeben sich Unstetigkeitsstellen, welches für eine Positionsregelung unter Umständen nicht handhabbar ist. Zudem weiß die

Positionsverfolgungseite nicht, wann die korrigierte Position in die Odometriedaten integriert wurde.

Zyklische Korrektur bei Stillstand
Der Roboter befindet sich nicht immer im Fahrtzustand. So ist es möglich, in den Ruhepausen oder in zyklischen Zeitintervallen ein Positionsupdate vorzunehmen. Nachteilig dabei ist, dass der Roboter ungewünschte Ruhepausen einlegt oder aber die Ruhepausen nicht schnell genug kommen.

Korrektur der Zielpositionen
Eine weitere Möglichkeit ist es, nicht die Roboterposition, sondern die Positionen der noch anzufahrenden Ziele abhängig vom gerade berechneten Korrekturvektor zu verändern. Somit bleiben die Odometriedaten des Roboters stetig.

In dieser Arbeit ist immer die dritte Variante zum Einsatz gekommen, welche sich durch die erfolgreichen Kartierung als auch die Hypothesenelimination im nächsten Kapitel bestätigt hat.

5.4 Trajektorienverfolgung und Hindernisvermeidung

Der dritte Teil um das Navigations-Problem effizient zu lösen, besteht in der Abarbeitung der Teilziele, welche durch die statische Pfadplanung generiert wurden.

Problem 5.20 (Trajektorienverfolgung) *Gegeben sei eine Menge von Konfigurationen* $p_1, \ldots, p_n$*, welche einen Pfad path von einer Start- zu einer Zielkonfiguration in der Einsatzumgebung repräsentieren. Dann soll der Roboter den Pfad stetig und jeweils mit bester Geschwindigkeit abfahren. Taucht ein Hindernis auf, soll der Roboter sicher vor dem Hindernis zum stehen kommen.*

Man kann das Problem noch erweitern, indem dynamischen Hindernissen ausgewichen werden soll:

Problem 5.21 (Dynamische Hindernisvermeidung) *Gegeben sei eine Menge von Konfigurationen* $p_1, \ldots, p_n$*, welche einen Pfad path von einer Start- zu einer Zielkonfiguration in einer Einsatzumgebung repräsentieren. Dann soll der Roboter den Pfad stetig und jeweils mit bester Geschwindigkeit abfahren. Taucht ein Hindernis auf, soll der Roboter dem Hindernis ausweichen und die nächste erreichbare Konfiguration* p_i *anfahren.*

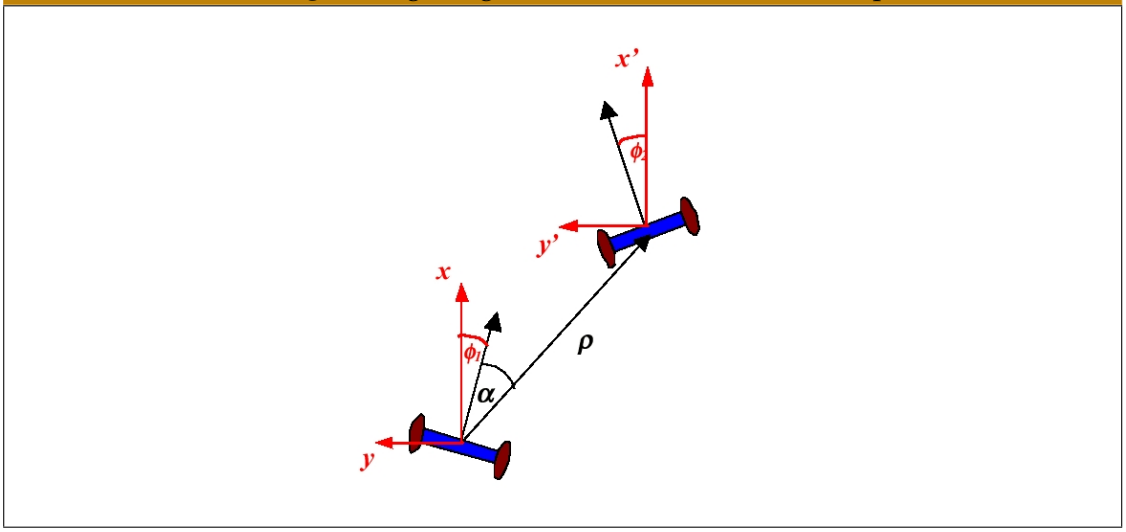
5.4.1 Generierung von stetigen Trajektorien

Man kann zunächst das einfache Problem 5.20 unter der Annahme einer stetigen Umgebung annehmen. Betrachtet man eine Startkonfiguration (x, y, θ) und eine Zielkonfiguration (x', y', θ'), wobei die Orientierung des Roboters keine Rolle spielen soll, ergibt sich ein regelungstechnisches Problem, welches mittels des Unicyle-Modells

gelöst wird [108].

Der Roboter besitzt den Systemzustand (x, y, θ) und zeitliche Ableitungen d.h. Translations- und Rotationsgeschwindigkeit (v_{trans}, v_{rot}).

Abbildung 48: Regelung eines Roboters zu einem Zielpunkt



Zur einfacheren Beschreibung des Problems wird ein Polarkoordinatensystem (ρ, α) eingeführt, das seinen Ursprung im gedachten Mittelpunkt des Roboter hat und mitbewegt wird. Die Koordinate ρ bezeichnet den Abstand zum erreichenden Zielpunkt (x', y', θ') und α den entsprechende Winkel, um den der Koordinatenursprung des lokalen Koordinatensystems des Roboters vom Ziel abweicht. Die aktuelle Position des Ziels wird damit zur Regelabweichung (siehe Abbildung 48). Mathematisch ergibt sich folgende Darstellung eines Differentialgleichungssystems für das einfache Unicycle-Modell

$$\dot{\rho} = v_x \cos\alpha \tag{5.5}$$

$$\dot{\alpha} = v_{trans}\frac{\sin\alpha}{\rho} - v_{rot} \tag{5.6}$$

$$\dot{\theta} = -v_{rot} \tag{5.7}$$

Gesucht sind die Translations- und Rotationsgeschwindigkeit des Roboters, die als Stellgrößen (v_{trans}, v_{rot}) berechnet werden.

Nach Linearisierung ergibt sich die Berechnungsvorschrift für die Stellgröße $(v_{trans}, v_{rot})^T$ mit den Bedingungen $v_{trans} < v_{trans}^{max}$ und $v_{rot} < v_{rot}^{max}$ zu

$$v_{trans} = K_\rho \rho \tag{5.8}$$

$$v_{rot} = K_\alpha \alpha + K_\theta \theta \tag{5.9}$$

Ein stabiles Verhalten wird für $K_\rho = 1, K_\theta = -1$ und $K_\alpha = 3$ erreicht. Nachdem ein Teilziel erreicht ist, kann das nächste Teilziel anvisiert werden und so eine stetige Bewegung des Roboters erreicht werden.

Unter Berücksichtigung von Hindernissen muss ein zusätzlicher Detektionsmechanismus eingebaut werden, dass der Roboter in diesem Fall seine Geschwindigkeit verringert und zum Stillstand kommt. Der Algorithmus 5.7 leistet das gewünschte.

Algorithmus 5.7: Regelung entlang vorgegebenen Trajektorie

Aufruf: **static-control()**
Input: Zielpunkt p, Genauigkeit ϵ, v_{rot}^{max}, v_{trans}^{max}
Output: Status status des Roboters, ob Teilziel erreicht wurde

```
1  stop := false; status = OK;
2  Berechne (ρ, α) zum Zielpunkt.
3  while {ρ > ε AND !stop} do
4     Berechne (ρ, α) zum Zielpunkt.
5     if {ρ <= ε} then
6        v_rot = 0, v_trans = 0
7     else
8        Berechne neue Translationsgeschwindigkeit.
9        Berechne neue Rotationsgeschwindigkeit.
10       if {Nahes Hindernis in Sensordaten detektiert} then
11          Setze stop := true; status = BLOCKED;
12       end if
13    end if
14 end while
15 if status != BLOCKED then
16    status = REACHED;
17 end if
18 return status;
```

5.4.2 Dynamische Hindernisvermeidung

Dynamische Hindernisvermeidung nimmt in der Literatur einen breiten Rahmen ein [26, 69, 83, 125, 170]. Dabei wird bei allen Ansätzen davon ausgegangen, dass man die aktuellen Sensorinformationen zu einem reaktiven Verhalten des Roboters heranzieht, da das Ausweichen vor einem Hindernis schnell erfolgen muss.

Bei allen Ansätzen besteht als Nebeneffekt die Gefahr, dass das Teilziel nicht erreicht werden kann, so dass ein Navigationsverfahren Möglichkeiten bieten muss, diese Gefahr zu erkennen und durch eine Neuplanung eines Weges diese Gefahr bannt.
Die wichtigsten in der Literatur beschriebenen Ansätze sind im folgenden kurz aufgestellt:

Künstliche Potentialfelder
Die Potentialfeld-Methoden [132], welche in Kapitel 5.2.3 vorgestellt wurde, wird nur auf lokale Sensorinformationen angewendet um damit jeweils ein Teilziel zu erreichen. Damit können mit Hilfe des Potentialfeldes dynamische Hindernisse umgangen werden.

Vektor-Feld-Histogramm
Dieser Ansatz wurde von Borenstein vorgestellt [26–28]. Dabei wird ein Occupancy-Grid [64] für die aktuellen Sensordaten verwendet um daraus Hindernis-Informationen zu extrahieren. Die Umgebung des Roboters wird dabei in mehrere Sektoren unterteilt und für jeden Sektor eine Bewertung über die Hindernisdichte berechnet. Unter Berücksichtigung des zu erreichenden Zieles wird ein Sektor ausgewählt welcher einen möglichst freien Raum in Richtung des Ziels repräsentiert. Aus diesem werden Bewegungsrichtung und Geschwindigkeit des Roboters berechnet.

Elastic-Band Ansatz
Ein elastisches Band [30, 170, 171] ist ein deformierbarer, kollisionsfreier Pfad repräsentiert als eine Menge von miteinander verbundenen Blasen (engl. bubbles). Die Blasen eines elastischen Bandes repräsentieren alle Konfigurationen p des Roboters, welche durch vorgegebene kinematische Restriktionen von der Konfiguration q ohne Kollisionen erreichbar sind. Der Radius r des Bubbles is dabei die minimale Distanz zu dem am nächsten liegenden Hindernis. Die Generierung der Blasen wird durch Untersuchung der lokalen Freiheit des Roboters auf Konfigurationen entlang des Weges bewerkstelligt. Jede Blase wird dabei durch repulsive und attraktive Kräfte wie beim Potentialfeld-Ansatz modifiziert. Zusätzlich wird verlangt, dass der Ursprung einer Blase immer in einer vorherigen Blase liegen muss. Somit wird ein Pfad zu einem Teilziel berechnet und mittels neuen Sensorinformationen inkrementell adaptiert.

Dynamic-Window
Das Dynamic Window [31, 69, 71, 72, 198] ist ein lokaler Suchraum im Raum aller möglichen Translations- und Rotationsgeschwindigkeiten. Es sei Δt das Zeitintervall mit den Beschleunigungen $\dot{v}, \dot{\omega}$. Weiter sei (v_a, ω_a) der aktuelle Geschwindigkeitsvektor. Dann ist das Dynamic Window definiert als $V_d = \{(v, w) \mid v \in [v_a - \dot{v}\Delta t, v_a + \dot{v}\Delta t] \text{ AND } \omega \in [\omega_a - \dot{\omega}\Delta t, \omega_a + \dot{\omega}\Delta t]\}$ und beschreibt ein Rechteck im Geschwindigkeitsraum um die aktuellen Geschwindigkeiten v und w. Alle Geschwindigkeiten außerhalb des Fensters können durch den Roboter bis zum nächsten Zeitschritt nicht erreicht werden. Damit können auch die zu den Geschwindigkeiten gehörigen Trajektorien nicht erreicht oder angenommen werden. Durch die Aufstellung einer Optimierungsfunktion wird unter den Constraints der Distanz zum Ziel, der Ausrichtung am Ziel und der maximalen Geschwindigkeit die Menge der erlaubten Geschwindigkeiten bewertet und die Geschwindigkeit mit der maximalen Bewertung bestimmt.

Der allgemeine Algorithmus 5.8 bindet die obigen Algorithmen in die Problematik der Navigation ein.

Der große Nachteil dieser Verfahren ist die Tatsache, dass ein stabiles Verhalten des Roboters nicht vorausgesagt werden kann im Gegensatz zur einfachen Trajektorienverfolgung. Auch wenn die meisten der obigen Verfahren in der Praxis evaluiert worden sind, ist eine industrielle Nutzung aus Sicherheits- und Genauigkeitsgründen bisher nicht erfolgt.

Algorithmus 5.8: Dynamische Hindernisvermeidung

Aufruf: **dynamic-obstacle-avoidance()**
Input: Aktuelle Position $\overline{x_k}$, Scan s_k, aktuelle Zielliste path
Output: Melde einen Status zurück

```
1  Setze Status auf OK.
2  while {Es gibt noch Teilziele in path} do
3     if {Ist das aktuelle Ziel p_i auch noch das beste Teilziel?} then
4        Hole das neue beste Ziel p_j.
5     end if
6     Werte s_k aus und identifiziere Hindernisse.
7     Teste die möglichen Geschwindigkeitspaare (v, w) und wähle das beste Paar.
8     if {Keine Geschwindigkeit gefunden} then
9        Setze (0, 0)
10       return Status BLOCKED.
11    end if
12 end while
13 return Status = REACHED.
```

5.5 Lösungsalgorithmus des Navigationsproblems

In den obigen Abschnitten sind die einzelnen hier interessierenden Teilprobleme des Navigationsproblems besprochen und aktuelle, in der Literatur vorkommende Lösungsansätze vorgestellt worden. Diese Ansätze kommen in Teilen hier zum Einsatz. Die Navigation wird nun unabhängig von der obigen Zerlegung in Teilprobleme in drei unterschiedliche Prozesse zerlegt. Damit ist es möglich, eine auf mehrere Rechner verteilte Berechnung der Problematik durchzuführen.

Der erste Prozess beschreibt den Navigationsalgorithmus 5.9. Dieser kommuniziert mit zwei weiteren Prozessen und reagiert auf deren Ergebnisse. Der Navigationsalgorithmus stellt die Pfadplanung von einem Start- zu einem Zielpunkt sicher. Die tatsächliche Pfadausführung wird an die dynamische Hindernisvermeidung (Algorithmus 5.8) oder die statische Regelung (Algorithmus 5.7) weitergegeben. In regelmäßigen Abständen wird vom Navigationsprozess der Status der Bewegung abgefragt. Wird der Pfad

Algorithmus 5.9: Navigation oder Zielsuche

Aufruf: **navigation-step()**
Input: Startposition p_{start}, Zielposition p_{end}
Output: Status, ob das Ziel erreicht wurde oder nicht

```
1  basicState = OK, returnState = OK
2  Verifiziere den initialen Standort des Roboters.
3  Berechne Pfad path von Start- zur Zielkonfiguration.
4  if {path != ∅} then
5     Übergebe Pfad der dynamischen Hindernisvermeidung.
6     while {basisState != REACHED} do
7        // Abfrage der Positionsschätzung
8        Hole basicStatus der dynamischen Hindernisvermeidung.
9        if {basicState = BLOCKED} then
10          Berechne neuen Pfad path vom Standort zum Ziel
11          if { path = ∅} then
12             basicState = OK; returnStatus = FALSE;
13          else
14             basicState = OK;
15             Übergebe Pfad der dynamischen Hindernisvermeidung.
16          end if
17       end if
18       // Abfrage der Positionsschätzung
19       Hole aktuelle Positionsschätzung der Positionsverfolgung.
20       if {Position verloren} then
21          basicState = REACHED; returnState = FALSE;
22       end if
23    end while
24 end if
25 return returnState;
```

des Roboters wirksam blockiert, wird versucht, einen neuen Pfad vom Ausgangspunkt zum Zielpunkt zu planen. Scheitert auch das, ist die Navigation fehlgeschlagen. Der grundlegende Bewegungsprozess liefert nach Anfrage einen *OK*-Status. Ist die Menge der Teilziele abgearbeitet, wird der Status *REACHED* zurückgegeben, bei einem ausweglosen Hindernis ein Status *BLOCKED*.

Die Positionsverfolgung mit Algorithmus 5.6 oder 5.5 bestimmt als dritter Prozess iterativ eine tatsächliche Positionsschätzung für den Roboterstandort. Dabei wird die mögliche Positionsungenauigkeit berechnet.

Der Navigationsprozess reagiert auf zu große Positionsungenauigkeiten indem die Position als verloren angegeben wird, so dass eine globale Selbstlokalisation dann Abhilfe schaffen muss. Durch diesen modularen Aufbau ist sichergestellt, dass einzelne Kom-

ponenten (siehe auch Kapitel 9.1.2) einfach ausgetauscht werden können. Eine praktische Evaluation der Zielsuche wird nicht präsentiert, wird doch dieser Ansatz in der praktischen Lösung des Hypotheseneliminationsproblems verwendet.

5.6 Resümee

In diesem Kapitel wurde ein grundlegendes Verfahren beschrieben, einen Roboter effizient von einem beliebigen Startpunkt in einer Umgebung zu einem beliebigen anderen Zielpunkt fahren zu lassen. Dabei wurden bekannte Standardverfahren der Robotik eingesetzt, kombiniert und zu einem Gesamtsystem zusammengefügt. Dieses ist eine Kombination aus planungs-basierten und reaktiven Komponenten, kann in mehrere Prozesse aufgegliedert werden und insbesondere auch räumlich verteilt arbeiten.

Dieses Kapitel hat nicht alle Fragestellungen abschließend behandelt. So wurde die Problematik von sich bewegenden dynamischen Hindernissen ausgespart, erhöht sie doch noch einmal die Komplexität eines Navigationssystems. Ansätze in der Literatur findet man in [67, 123].

Der vorgestellte Navigationsalgorithmus ist noch anfällig gegen unerwartete Störungen durch Fehlerquellen, die nicht immer reproduzierbar sind. Redundante Algorithmen (siehe auch Kapitel 8.2) sollten solche Störungen eliminieren. Langzeit-Testverfahren sind notwendig, die Güte der Navigationsalgorithmen zu erhöhen. Diese können in der Regel in Forschungsprojekten nicht durchgeführt werden, da der wissenschaftliche Fortschritt dabei sehr gering ist. Es ist bezeichnend, dass trotz vieler Jahre intensiver Robotik-Forschung bisher nur ein Navigationssystem für mobile Roboter käuflich zu erwerben ist [135].

6 Hypothesenelimination

Im ersten Teil der Arbeit wurde geschildert, wie ein mobiler Roboter Hypothesen erzeugt, wenn er seinen Standort nicht kennt. In diesem Kapitel sollen Algorithmen vorgestellt werden, um aus der Menge der Hypothesen die tatsächliche Position des Roboters zu bestimmen.

Dazu wird zunächst auf die klassische Lösungsmethode eingegangen, die für das Verständnis weiterer Algorithmen notwendig ist. Dieser Lösungsansatz wird im weiteren auch statische Lösung bezeichnet. Danach wird die für den praktischen Gebrauch relevante Problemstellung vorgestellt, wie auch die bisher bekannten Lösungsansätze. Schließlich wird ein neuer Lösungsansatz präsentiert, welcher sich an der klassischen Lösung orientiert. Dazu wird der Algorithmus im praktischen Einsatz getestet.

6.1 Definitionen und Problemstellungen

Beginnen soll dieser Abschnitt mit einer Erweiterung der Formalismen, welche in Kapitel 5 ausgeführt wurden.

6.1.1 Geometrischer Baum und Overlay-Baum

Betrachte im weiteren einen geometrischen Graphen, welcher eine Roadmap einer Einsatzumgebung repräsentiert.

Definition 6.1 (Geometrischer Baum) *Ein zusammenhängender, zyklenfreier, ungerichteter geometrischer Graph, der damit die Eigenschaften eines Baumes erfüllt, wird als geometrischer Baum bezeichnet.*

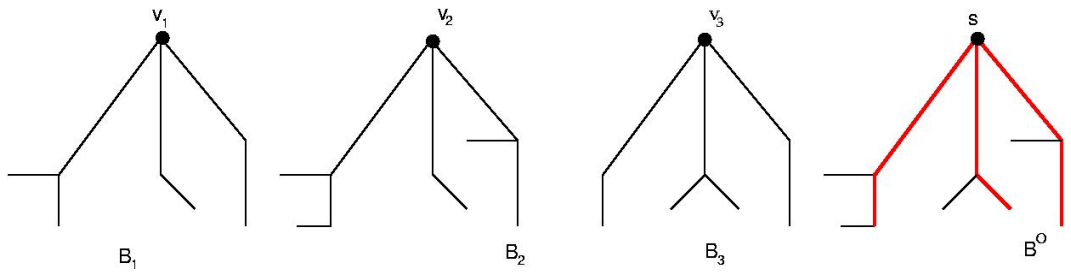
Definition 6.2 (Overlay Baum) *Ein Overlay-Baum ist ein geometrischer Baum der durch Vereinigung mehrerer geometrischer Bäume entsteht. Er wird definiert als ein 5-Tupel* $B^O = (O, \mathcal{B}, \mathcal{S}, \kappa, \lambda)$ *mit*

- $O = (\mathcal{V}_O, \mathcal{E}_O)$ ist ein geometrischer Baum
- $\mathcal{B} = \{B_1, \ldots, B_n\}$ ist eine Menge geometrischer Bäume.
- $\mathcal{S}$ ist eine Eckenmenge mit der Eigenschaft: es gibt eine bijektive Abbildung $\phi : \mathcal{S} \to \mathcal{B}$ und $\phi(s) = \mathcal{B} \Rightarrow s \in \mathcal{B}$. s heißt Startecke.

- κ ist eine Funktion $\kappa : \mathcal{V}_O \to 2^{\mathcal{B}}$, für jede Ecke $v \in O$ des Ursprungsbaumes.
- λ ist eine Funktion $\lambda : \mathcal{E}_O \to 2^{\mathcal{B}}$, für jede Kurve $e \in O$ des Ursprungsbaumes.

Alle Bäume $B_i \in \mathcal{B}$ werden dabei durch die Startecke s auf den Ursprung abgebildet. Alle überlappenden Kurven werden vereinigt. Falls zwei Äste differieren, werden sie in den Overlay Baum eingefügt. Spätere Überschneidungen werden nicht mehr berücksichtigt. Ein Beispiel für einen Overlay-Baum liefert Abbildung 49. Einige Ecken im Overlay Baum haben besondere Eigenschaften:

Abbildung 49: Beispiel eines Overlay-Baums



Drei Bäume (links) werden zu einem Overlay-Baum (rechts) überlagert.

Definition 6.3 (Unvollständige Ecke) *Eine Ecke v mit $\kappa(v) \neq \mathcal{B}$ heißt unvollständige Ecke. Die Menge $\mathcal{V}^u = \{v \in \mathcal{V}_O : \kappa(v) \neq \mathcal{B}\}$ ist die Menge der unvollständigen Ecken.*

Definition 6.4 (nächstgelegene unvollständige Ecke) *Für die Menge der unvollständigen Ecken $\mathcal{V}^u$ im B^O ist v^{nu} eine* nächstgelegene unvollständige Ecke*, wenn gilt:*

$$\forall v \in \mathcal{V}^u : l(W_{v_{nu}}) \leq l(W_v)$$

Eine unvollständige Ecke markiert eine Konfiguration, an welcher sich alle Bäume, die durch einen Overlay-Baum repräsentiert werden, unterscheiden. Interessant ist dabei vor allem die von der Startecke s aus am nächsten gelegene unvollständige Ecke des Overlay-Baumes.

Definition 6.5 (Kürzeste Wege Baum) *Ein Baum $B(p) = (\mathcal{V}_B, \mathcal{E}_B)$ ist ein* kürzeste-Wege Baum *zu einem geometrischen Graphen $G = (\mathcal{V}_G, \mathcal{E}_G)$, wenn gilt:*

- $B(p)$ ist ein geometrischer Baum mit der Wurzel $p \in \mathcal{V}_B$.
- $\mathcal{V}_B = \mathcal{V}_G$,
- $\forall v \in \mathcal{V}_B$: der Pfad $W(p, v) \in B$ von p nach v existiert und $W(p, v)$ ist ein kürzester Pfad von p nach v in G, d.h. $l(W(p, v))$ ist minimal bezüglich aller Pfade in G.

6.1.2 Sensorpunkt und Entscheidungsstrategie

Im Kontext der Hypothesenelimination sind ausgezeichnete Konfigurationen wichtig, benötigt man doch solche um ununterscheidbare Hypothesen dadurch zu unterscheiden.

Definition 6.6 (Sensorpunkt) *Ein Sensorpunkt* sp *ist eine Konfiguration innerhalb der Einsatzumgebung* m*, an der durch neue Sensorinformationen Hypothesen ausgeschlossen werden können, d.h.* $sp \in \mathcal{C}_{free}$*. Die Menge der Sensorpunkte wird mit* SP *bezeichnet.*

Betrachtet man die vorherige theoretische Systematik des Overlay-Baumes, dann ist ein Sensorpunkt dann eine unvollständige Ecke des Overlay-Baumes, da die Wegeinformation von s bis zu einer unvollständigen Ecke aller Bäume identisch war und ab der unvollständigen Ecke differiert. Im späteren Verlauf der Arbeit wird der Begriff Sensorpunkt etwas schwächer interpretiert. Ein Sensorpunkt muss dann nicht mehr eine unvollständige Ecke des Overlay-Baumes sein. Siehe dazu auch die Abbildung 54.
Dabei taucht die Problematik auf, welche nächste unvollständige Ecke man von s aus anfahren soll. Die Auswahl einer solchen Ecke realisiert eine sogenannte Entscheidungsstrategie:

Definition 6.7 (Entscheidungsstrategie) *Eine Entscheidungsstrategie* N *determiniert die Reihenfolge in der die Sensorpunkte der Menge* SP *besucht werden, und damit, in welcher Reihenfolge die Hypothesen eliminiert werden.*

Definition 6.8 (Verifikationspfad) *Ein Verifikationspfad* $W_{L,N}$*, abhängig vom Algorithmus* L *zur Hypothesengenerierung und von der Entscheidungsstrategie* N*, bezeichnet die Gesamtheit aller Teilpfade, die der Roboter nach Abarbeitung des Hypothesenelimination-Algorithmus befahren hat. Der optimale und damit effizienteste Verifikationspfad wird mit* $W^*_{L,N}$ *bezeichnet.*

6.1.3 Effizienz der Entscheidungsstrategie

Die Effizienz einer solchen Entscheidungsstrategie wird in der Theorie [121] durch den Kompetitivitätsfaktor bestimmt. Dieser repräsentiert das Verhältnis zwischen der tatsächlichen Lösung und der besten Lösung und wird bestimmt als das Maximum aus allen möglichen Fällen.

Definition 6.9 (Kompetitivitätsfaktor) *Der* Kompetitivitätsfaktor $C_{L,N}$ *abhängig vom Algorithmus* L *zur Hypothesengenerierung und der Entscheidungsstrategie* N *ist definiert als*

$$C_{L,N} = \max_{m \in \mathcal{M}, p \in \mathcal{C}} \frac{d(W_{L,N}(p,m))}{d(W^*_{L,N}(p,m))}$$

Dabei bezeichnet $\mathcal{M}$ *die Menge aller möglichen Karten und* p *Konfigurationen aus* $\mathcal{C}$*. Eine approximative Lösung des Problems hat einen* Kompetitivitätsfaktor *von* $C_{L,N}$ *mal der optimalen Lösung.*

In der Praxis ist eine solche Bewertung fraglich, in der Anwendung ist vielmehr eine optimale Lösung für eine Einsatzumgebung verlangt.

6.1.4 Definition der Problemstellung

Nachdem die notwendigsten Termini definiert wurden, kann man das Problem genauer spezifizieren:

Problem 6.10 (Statische Hypothesenelimination) *Der Roboter steht an einer ihm unbekannten Konfiguration p innerhalb der Karte m_g. Er verwendet einen Lokalisationsalgorithmus L, um eine Menge von hypothetischen Positionen H_s zu generieren. Um die wahre Position zu determinieren, muss er alle anderen Hypothesen eliminieren. Dazu verwendet er für die aktuelle Hypothesenmenge einen iterativen Entscheidungsalgorithmus N, um entlang eines Verifikationspfades die tatsächliche Konfiguration p^* zu bestimmen.*

Wie schon im ersten Teil der Arbeit aufgezeigt, ist die Generierung von Hypothesen mit vielen Fehlern behaftet. Es ist in der Praxis unrealistisch, auf genau **eine** Lokalisationsanfrage eines Algorithmus L zu bauen um dann zu versuchen, diese generierten Hypothesen zu eliminieren (siehe Kapitel 2). Deshalb wird das Problem hier folgendermaßen erweitert:

Problem 6.11 (Dynamische Hypothesenelimination) *Der Roboter steht zu einem Zeitpunkt $k > 1$ an einer ihm unbekannten Konfiguration p innerhalb der Karte m_g. Er verwendet einen Lokalisationsalgorithmus L, um zum aktuellen Zeitpunkt eine Menge von hypothetischen Positionen H^k_{loc} zu generieren. Die Menge H^k repräsentiert eine bewertete Menge von hypothetischen Standorten des Roboters, die seit dem Positionsverlust des Roboters generiert wurden.*
Das Problem besteht nun darin, solange einen Entscheidungsalgorithmus N anzuwenden, welcher den Roboter entlang eines Verifikationspfades $W_{L,N}(p, m_k)$ zu ausgewählten Sensorpunkten bringt, bis eine Hypothese h aus H^k eine akzeptable sichere Bewertung besitzt, so dass der Roboter diese als seinen Standort p^* akzeptieren kann.

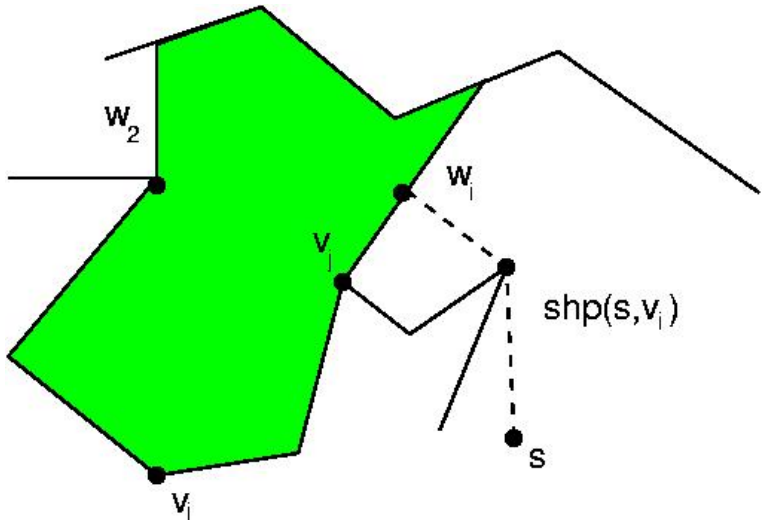
Beiden Problemstellungen ist gemein, dass die tatsächliche Position des Roboters unbekannt ist. So ist zunächst völlig unklar, wie ein Verifikationspfad zu generieren ist.

6.2 Lösung des statischen Problems

Der zweite Teil des globalen Selbstlokalisationsproblems war als erstes von Dudek et al. beschrieben und gelöst worden [62]. Sie zeigten, dass das Problem in einfachen polygonalen Karten (Kapitel 3) ohne einbeschriebene Hindernisse unter Verwendung eines punktförmigen Roboters ohne Sensor- und Odometriefehler $\mathcal{NP}$-vollständig ist

durch Reduktion auf das Minimale Entscheidungsbaum-Problem. Die angegebene approximative Lösung verwendet sogenannte Overlay Arrangements von Sichtbarkeitszellenzerlegungen der gegebenen Karte, d.h. es wird zunächst eine Sichtbarkeitszellenzerlegung berechnet. Für alle berechneten k Hypothesen aus der Menge H_s einer Anfrage werden k Sichtbarkeitszellenzerlegungen übereinanderlegt und deren Schnitt bestimmt, so dass dann die Zellen bestimmt werden können, welche durch Schnitt entstanden sind. Positionen auf dem Rand einer Sichtbarkeitszelle, berechenbar durch kürzeste-Wege vom Ursprung aus, repräsentieren dann die Menge der Sensorpunkte. Durch Anfahren der jeweils am nächsten zum Ursprung und lokalem Standort liegenden Zelle und Elimination hypothetischer Standorte können so die Zerlegungen eliminierter Standorte entfernt werden, sodass man dadurch Sensorpunkte eliminiert. Durch sukzessives Abarbeiten der Hypothesen bekommt man zum Schluss den tatsächlichen Standort des Roboters. Der Algorithmus hat dabei eine Zeitkomplexität von $\mathcal{O}(k^2n^4)$, wobei k die Anzahl der Hypothesen bezeichnet und n die Anzahl der Ecken der Karte.

Abbildung 50: Fenster

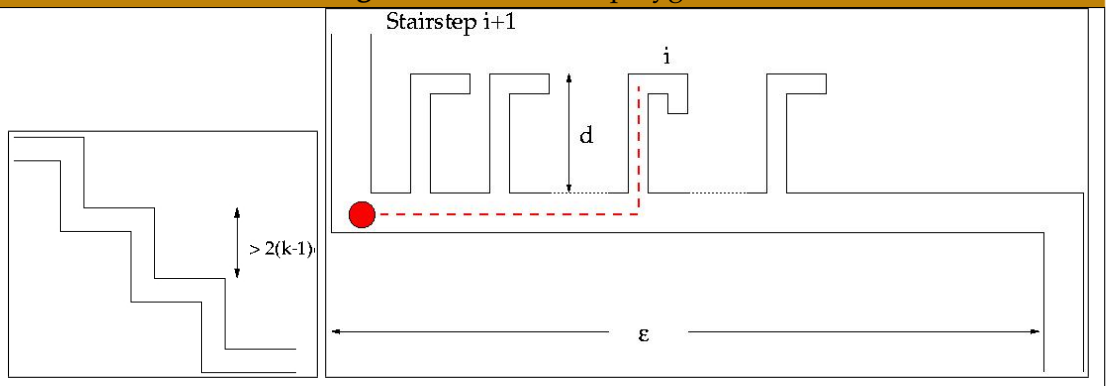


Die Ecke v_i kann auf dem kürzesten Weg von s nach v_i erst auf dem Window W_i zum ersten mal gesehen werden. Dieses Window wird von v_i und der Reflexecke v_j erzeugt.

1996 gibt Schuierer eine bessere Lösung für das Problem an mit einer Zeitkomplexität von $\mathcal{O}(kn^2)$ [194]. Anstatt einer Sichtbarkeitszellenzerlegung betrachtet er nur sogenannte Fenster innerhalb der Karte auf denen Sensorpunkte liegen. Dazu berechnet er zunächst kürzeste-Wege-Bäume von allen hypothetischen Positionen innerhalb der Karte zu allen Ecken des Polygons. Die Sensorpunkte, an denen falsche Hypothesen ausgeschlossen werden können, sind die Schnittpunkte des kürzeste-Wege-Baumes von der Startecke s zu ausgewählten Ecken v_i mit zugehörigem Fenster w_i. An die-

sem Schnittpunkt kann die zugehörige Ecke zum ersten Mal gesehen werden. Diese modifizierten kürzeste-Wege-Bäume werden in einen Overlay-Baum eingefügt. Eine Entscheidungsstrategie sucht dann unvollständige Knoten innerhalb des Overlay Baums, welche Sensorpunkte für den Overlay-Baum repräsentieren.

Abbildung 51: Worst-Case in polygonalen Karten



Worst-Case Beispiel für punktförmige Roboter in einfachen Polygonen mit einem Kompetitivitätsfaktor von $2(k-1)$. Links ist ein Treppenstufenpolygon im Ausschnitt dargestellt. Die Verbindungen zwischen den Treppenstufen sind exorbitant hoch und sind für die Lösung nicht wichtig. Auf jeder dieser k Treppenstufen des Polygons gibt es k Kamine, die eine Länge von d aufweisen. Die Entfernung ϵ im Bild rechts ist vernachlässigbar, da man die Kamine auch sternförmig anordnen kann. Jede Treppenstufe unterscheidet sich von den anderen Treppenstufen in genau einem Kamin, so dass der Roboter diesen Kamin finden muss, um zu entscheiden auf welcher Treppenstufe er sich befindet. Der optimale Weg würde zu Beginn den optimalen Kamin nehmen und hätte eine Distanz von d zurückgelegt. Die worst-case Strategie hingegen muss alle Kamine abfahren und kann erst mit dem Absuchen des vorletzten Kamin entscheiden, in welcher Treppenstufe er sich befindet. Somit ergibt sich ein Kompetitivitätsfaktor von $2(k-1)$.

Wie vorher schon bei Dudek verwendet Schuierer die Entscheidungsstrategie *minimum-distance-localization (MDL) und wählt jeweils den am nächsten liegenden Sensorpunkt*. Diese Greedy-Strategie hat einen Kompetitivitätsfaktor von $2(k-1)$, wobei der Roboter in jedem Eliminationsschritt zur Ausgangsposition zurückkehrt. D.h. in jedem Schritt wird im worst-case genau eine Hypothese eliminiert, sodass insgesamt $(k-1)$ Schritte unternommen werden müssen. Der Faktor 2 repräsentiert die jeweilige Rückkehr zum Ausgangspunkt (siehe dazu auch das Beispiel in Abbildung 51). Der Lösungsalgorithmus von Schuierer wird durch Algorithmus 6.1 repräsentiert.

Algorithmus 6.1: Algorithmus von Schuierer

Aufruf: **schuierer-mdl-strategy()**
Input: Menge H_s hypothetischer Roboterpositionen
Output: Der wahre Standort p^* des Roboters

1 Berechne für jede Ecke v in m und für jede Hypothese $h \in H_s$ einen Sensorpunkt v_h^*. v_h^* ist der am nächsten liegende Sensorpunkt zu h welcher auf dem am nächsten liegenden Fenster liegt, welches von v aus sichtbar ist.
2 Berechne für jede Hypothese $h \in H_s$ einen kürzeste-Wege-Baum B_h von h zu allen Sensorpunkten v_h^*. $\mathcal{B} = \{B_h \mid h \in H_s\}$.
3 Berechne den Overlay-Baum $B^O = (O, \mathcal{B}, H_s, \kappa, \lambda)$.
4 **while** $\mid H_s \mid > 1$ **do**
5 b^* ist die nächste unvollständige Ecke in B^O und damit Sensorpunkt von B^O.
6 Berechne den kürzesten Weg W_{b^*} vom Ursprung zu b^*.
7 Traversiere den Pfad W_{b^*}, generiere neue Sensorinformationen am Endpunkt und kehre zum Ausgangspunkt zurück.
8 Lösche die Hypothesen $\bar{h}$ welche nicht konsistent mit den Sensorinformationen sind und führe ein Update des Overlay-Baums aus.
9 **end while**
10 Liefere die verbliebene Hypothese aus H_s als wahren Standort p^* zurück.

6.2.1 Modifizierte Eliminationsstrategien

Bisher wurde nur die Strategie MDL präsentiert und analysiert. Diese Strategie hat den beweisbar besten Kompetitivitätsfaktor für das obige Problem in einfachen Polygonen. Für die Praxis kann es jedoch andere Entscheidungsstrategien geben, die eine bessere Performance für einzelne Karten liefern mit einem vielleicht schlechteren Kompetitivitätsfaktor.

Entscheidungsstrategie RWL

Die Entscheidungsstrategie Restweglänge (RWL) verwendet die erwartete Größe der Hypothesenmenge, welche eliminiert wird, als Kriterium. Dabei wird angenommen, dass alle Hypothesen gleich wahrscheinlich sind. Dann wird eine Gewichtsfunktion definiert, welche die Länge des Pfades zum nächsten Sensorpunkt mit dem Erwartungswert der Anzahl der Hypothesen nach dem Eliminationsschritt multipliziert.

$$\forall i : \overline{l(W)}_i = l(W_i) \cdot \frac{1}{k} \cdot \overline{|H_s|}_i \tag{6.1}$$

Dabei ist $\overline{|H_s|}_i$ die erwartete Größe der Hypothesenmenge. Diese kann beschrieben werden als

$$\overline{|H_s|}_i \leq \frac{|H_{s_{1,i}}|}{|H_s|} |H_{s_{1,i}}| + \frac{|H_s| - |H_{s_{1,i}}|}{|H_s|} (|H_s| - |H_{s_{1,i}}|) \tag{6.2}$$

Entscheidungsstrategie IEE

Ist die Menge der generierten Hypothesen nicht gleichverteilt, kann man eine andere Entscheidungsstrategie Inverse Erwartete Entropie (IEE) verwenden. Dabei wird angenommen, dass ein Lokalisationsalgorithmus L eine Menge von bewerteten Hypothesen zurückliefert. Dann haben diese Hypothesen unterschiedliche Wahrscheinlichkeiten bezüglich der Bewertung und der Gewichtsfunktion.

Demzufolge wurde die Gleichung für den Erwartungswert der Hypothesenmenge durch eine Gleichung für den Informationsgehalt der Hypothesen ersetzt.

$$\overline{|H_s|}_i \leq \frac{\sum p(h_j)}{\sum p(h_g)} \cdot |H_{s_{1,i}}| + \frac{\sum p(h_k)}{\sum p(h_g)} \cdot (|H_s| - |H_{s_{1,i}}|) \tag{6.3}$$

Dabei ist $h_j \in H_{s_{1,i}}, h_k \in (H_s - H_{s_{i,1}}), h_g \in H_s$.

Beweis des Kompetitivitätsfaktors

In diesem Abschnitt wird der Beweis für einen Kompetitivitätsfaktor einer Entscheidungsstrategie bewiesen, welche dieselbe Menge der Sensorpunkte wie die Strategie MDL verwendet.

Satz 6.12 (Kompetitivität einer gewichteten Hypothesenmenge) *Man nimmt an, dass die Entscheidungsstrategie* N *zur Hypothesenelimination in jedem Schritt den minimalen Wert der Gewichtsfunktion* $s_i = h_i \cdot l_i$ *für ein passendes* i *wählt und in jedem Schritt mindestens eine Hypothese eliminiert. Dabei bezeichnet* h_i *den Strategiewert und* l_i *die Länge des kürzesten Pfades zu einem Sensorpunkt. Dann hat diese Strategie einen Kompetitivitätsfaktor von* $2(k-1)\frac{h_{max}}{h_{min}}$ *bei Rückkehr zum Startpunkt in jedem Eliminationsschritt in einfachen Polygonen.*

Beweis: Die Sensorpunkte oder unvollständigen Knoten des Overlay-Baumes werden durch kürzeste-Wege-Bäume berechnet. Deshalb gibt es auch keine Umwege um die Punkte zu erreichen.
Annahme: Die richtige Hypothese wird als letztes verarbeitet. Dann werden $(k-2)$ Hypothesen zuvor betrachtet. Man bekommt also

$$s_1 \leq s_2 \cdots \leq s_{k-2} \leq s_{k-1} \tag{6.4}$$

Diese Ungleichung kann man schreiben als

$$l_1 \cdot h_1 \leq \cdots \leq l_{k-1} \cdot h_{k-1} \tag{6.5}$$

Wenn l_{k-1} der optimale kürzeste Weg ist mit Distanz d und h_{opt} der Strategiewert für diesen, dann bekommt man die Ungleichungen

$$l_i \cdot h_i \leq d \cdot h_{opt} \Rightarrow l_i \leq \frac{d \cdot h_{opt}}{h_i} \tag{6.6}$$

für $1 \leq i \leq k-2$.
Für die Summe des Verifikationspfades bekommt man

$$\sum_{i=1}^{k-1} l_i \leq (k-1)\frac{d \cdot h_{opt}}{h_{min}} \leq (k-1)\frac{d \cdot h_{max}}{h_{min}} \tag{6.7}$$

Daraus folgt der Kompetitivitätsfaktor

$$C_{L,N} = \max_{m \in \mathcal{M}, p \in m} \frac{l(W_{L,N}(p,m))}{l(W^*_{L,N}(p,m))} = (k-1)\frac{h_{max}}{h_{min}} \tag{6.8}$$

Also kann der Kompetitivitätsfaktor beliebig schlecht werden, da er abhängig von der Wahrscheinlichkeitsverteilung und der Gewichtsfunktion ist. Für $h_i = 1$ für alle i bekommt man den Kompetitivitätsfaktor $2(k-1)$. Die Strategie MDL ist also ein Spezialfall dieser Abschätzung. □

Für die oben vorgestellten Strategien bekommt man schlechtere Kompetitivitätsfaktoren, da MDL ja bewiesenermaßen den besten Kompetitivitätsfaktor besitzt. Stellvertretend für die anderen Strategien soll die Kompetitivität für die Strategie RPL hergeleitet werden:

Satz 6.13 (Kompetitivitätsfaktor der Strategie RPL) *Strategie RPL hat einen Kompetitivitätsfaktor von* $4(k-1)$.

Beweis: Unter Verwendung des obigen Theorems hat man nur nach dem Minimum und dem Maximum der Gewichtsfunktion zu suchen.

Aus Gleichung (6.1) und (6.2) bekommt man einen minimalen Wert $\frac{k}{2}$ und einen maximalen Wert $(k-1)$. Eingefügt in die Gleichung bekommt man einen minimalen Wert der Gewichtsfunktion von $h_{min} = \frac{k}{2}$ und einen maximalen Wert von $h_{max} = k$.

Damit bekommt man einen Kompetitivitätsfaktor von $4(k-1)$, wobei man nach jedem Eliminationsschritt zum Ausgangspunkt zurückkehrt. □

6.2.2 Anforderungen an die Praxis

In der Praxis ist das Verfahren von Schuierer nicht einsetzbar. Der theoretische Ansatz geht davon aus, dass in Bereichen mit gleicher lokaler Sicht die Berechnung der Trajektorien mit Anfangs- und Endkonfigurationen einzelner Kurven identisch ist, sodass diese aufeinander abgebildet werden können. Aber schon kleine Störungen, hervorgerufen durch ein dynamisches Hindernis, kann diese Trajektorien stören.

Ein Versuch, diese Problematik mit Voronoi-Diagrammen als Roadmaps zu lösen, scheiterte an eben diesen kleinen Störungen, die aber in alltäglichen Einsatzumgebungen vorhanden sind. Verwendet man randomisierte Roadmaps, kann man gleiche

Trajektorien bei gleicher Sicht nicht mehr garantieren.

Zudem ist die Annahme, durch eine einzige Lokalisationsanfrage eine Hypothesenmenge zu bekommen, die tatsächlich alle möglichen Roboterstandorte garantiert, nicht zu halten, wie auch oben schon erwähnt wurde.

6.3 Lösung des dynamischen Problems

In diesem Abschnitt werden zunächst die bekannten Lösungen des dynamischen Problems aus der Literatur detaillierter vorgestellt. Erfolgreiche Ansätze und Literatur sind für die Problematik sehr schwer zu finden, bindet dieses Problem zuviele ungelöste Teilprobleme ein. Neben der Monte-Carlo-Lokalisation werden die Aktive-Markov-Lokalisation [69] und das Multi-Hypothesen-Tracking [113] vorgestellt.

6.3.1 Die Monte-Carlo-Lokalisation

Der einfachste Lösungsalgorithmus kommt aus der Problemstellung der Positionsverfolgung. Die Monte-Carlo-Lokalisation, eine Erweiterung der Markov-Lokalisation, kann ebenfalls zur Elimination von Hypothesen genutzt werden [68].

Der Algorithmus arbeitet nach einem iterativen Schema: Zu Beginn wird im Kartenmodell m eine Menge von n Partikeln normalverteilt erzeugt. Abhängig von randomisierten Bewegungen a_k wird die Partikelmenge in jedem Schritt entsprechend des Bewegungsmodells des Roboters verschoben. Partikel, die auf Hindernisse oder aus der Karte verschoben werden, werden aus der Menge gelöscht. Nach einem Aktualisierungsschritt durch Bewertung mit einem Sensormodell durch neue Sensordaten werden die k am schlechtesten bewerteten Partikel gelöscht und abhängig von einer Heuristik neue Partikel an die Menge angefügt. Gibt es nach einer Reihe von Bewegungen einen besten Partikel, abhängig von einem einzustellenden Schwellwert, wird diese Position als aktuelle Roboterposition p^* angenommen.

Der Algorithmus 6.2 ist echtzeitfähig und die einfachste Heuristik zur Bestimmung der aktuellen Roboterposition. Er ist sehr anfällig für Umgebungen in denen ein nichtnormalverteilter Bewegungsfehler auftreten kann. Zu diesem Ansatz finden sich viele weitere Heuristiken zur Verbesserung [113].

6.3.2 Die aktive Markov-Lokalisation

Die aktive Markov-Lokalisation unterscheidet sich zum Monte-Carlo Ansatz und auch zur Markov-Lokalisation in einem entscheidenen Punkt. Während die Monte-Carlo-Lokalisation nur zufällige Aktionen ausführt, wählt die aktive Markov-Lokalisation

Algorithmus 6.2: Monte-Carlo-Lokalisation

Aufruf: **monte-carlo-localization()**
Input: Karte m der Umgebung, Anzahl von initialen Partikeln n
Output: Tatsächliche Position p^* des Roboters

```
1  Berechne n normalverteilte Partikel in der Karte.
2  ITERATE:=true, k = 0
3  while {ITERATE} do
4     k++;
5     Bestimme randomisiert eine Bewegungsaktion a_k für den Roboter.
6     Führe die Aktion aus.
7     Generiere neue Sensordaten s_k.
8     for {alle Partikel p_i} do
9        Transformiere das Partikel p_i entsprechend des Bewegungsmodells und der
         Bewegung a_k.
10       if {p_i stößt auf Hindernis oder liegt außerhalb der Karte} then
11          Lösche Partikel p_i.
12       else
13          Bewerte das Partikel p_i mit dem Sensormodell neu.
14          if {Partikel p_i erreicht eine beste Bewertung} then
15             p* := p_i
16             ITERATE:=false
17          end if
18       end if
19    end for
20    Lösche die schlechtesten k < n Partikel.
21    Füge normalverteilt neue Partikel hinzu.
22 end while
```

die jeweils beste berechnete Aktion a_k basierend auf der Minimierung der zukünftigen erwarteten Unsicherheit aus.

Betrachte dazu zunächst eine beliebige Bewegungsaktion a_k. Für diese Aktion kann man erwartete Kosten $C(a_k)$ und erwarteten Gewinn $U(a_k)$ berechnen. Der erwartete Gewinn $U(a_k)$ wird determiniert durch den Anstieg der Wahrscheinlichkeit beim Ausführen von a_k und Betrachtung einer Menge von $|s_k|$ Sensormesswerten an der neu berechneten Position. Die erwarteten Kosten $C(a_k)$ werden repräsentiert durch die Zeit und die Pfadlänge für Aktion a_k. Damit bekommt man eine Bewertung $B(a_k) = U(a_k) - C(a_k)$ für a_k.

Die aktive Markov-Lokalisation funktioniert nun folgendermaßen (siehe Algorithmus 6.3): Zunächst wird solange die Markov-Lokalisation ausgeführt, bis sich die Menge der hypothetischen Standorte H_m auf eine kleine Anzahl reduziert hat. Bezeichnet wird diese Menge von Konfigurationen mit H_m. Danach werden alle möglichen Aktionen

a_k betrachtet, die mittels der diskretisierten Karte m_g möglich sind. Abhängig von der Anzahl der möglichen hypothetischen Positionen und abhängig von der Anzahl der betrachteten Sensormeßwerte wird nun eine Bewertung für jede Aktion a_k berechnet, bestehend aus der Summe von $U(a)$ und $C(a)$ (die Details lassen sich in der Dissertation von Fox [69] nachlesen). Aus allen bewerteten Aktionen wird dann die beste Aktion a_k^* ausgewählt und ausgeführt. Mittels der Markov-Lokalisation wird dann eine neue Bewertung der Menge H_m erstellt, die u.U. weniger Hypothesen besitzt.

Algorithmus 6.3: Aktive Markov-Lokalisation

Aufruf: **active-markov-localization()**
Input: Gitterkarte m_g
Output: Tatsächliche Position p^* des Roboters

```
1  ITERATE:= true; STATUS:=RANDOM; k=0;
2  while {ITERATE} do
3     k++;
4     if {STATUS = RANDOM} then
5        Markov-Lokalisation solange bis Anzahl der guten Positionen gering ist.
6        if {Anzahl der möglichen Positionen | H_m |< 10} then
7           STATUS = ACTIVE
8        end if
9     end if
10    if {STATUS = ACTIVE} then
11       if {| H_m |> 1} then
12          for {alle möglichen Aktionen a_k der Karte m_g } do
13             Berechne den erwarteten Gewinn U(a) abhängig von |s_k| und | H_m |.
14             Berechne Kosten C(a_k) für die Aktion a_k.
15             Berechne Bewertung der Aktion a_k durch B(a_k) := U(a_k) − C(a_k).
16          end for
17          Wähle die beste Aktion a* aus.
18          Führe die Aktion a*.
19          Bilde mittels Markov-Lokalisation neue Zustandsbewertung H_m.
20       else
21          ITERATE:=false;
22          p* gefunden
23       end if
24    end if
25 end while
26 return p*
```

Für den Algorithmus gibt Fox die folgende Komplexität an:

Korollar 6.14 (Komplexität der Aktiven-Markov-Lokalisation) *Die Zeitkomplexität der Aktiven-Markov-Lokalisation liegt in* $\mathcal{O}(|\ L\ | \cdot |\ H_m\ | \cdot |s_k|)$ *für einen Iterati-*

onsschritt. Dabei bezeichnet $|L|$ *die Anzahl der diskretisierten Konfigurationen der Karte* m_g, $|H_m|$ *die Anzahl der hypothetischen Positionen und* $|S|$ *die Anzahl der betrachteten Sensormesswerte. Die Speicherkomplexität liegt in* $\mathcal{O}(L)$.

Die aktive Markov-Lokalisation ist eine probabilistische Variante des Algorithmus von Schuierer. Allerdings verwendet dieser Algorithmus weniger Intelligenz. Würde man eine geometrische Version dieses Algorithmus betrachten, hätte man ebenfalls eine Diskretisierung der Karte zur Verfügung. Man würde alle möglichen Aktionen betrachten, dafür die möglichen Wege und eine Bewertung berechnen, welche die Übereinstimmung von Wegen unter Einbeziehung der Kosten der Wege determiniert. Somit würde man ebenfalls einen Overlay-Baum mit eingeschränkter Sensorinformation berechnen. Der Algorithmus hat den Nachteil, für große Karten völlig unpraktikabel zu werden, da man alle möglichen Aktionen a_k der Karte betrachtet. Die intelligente Auswahl der Sensorpunkte und damit die beschränkte Menge von Aktionen a_k bei Schuierer wird durch einen informationslosen Vorgang des Austestens aller Möglichkeiten ersetzt.

6.3.3 Multiples Hypothesentracking

Der Ansatz von Jensfeld [113] ist ein feature-basierter Ansatz, und gliedert sich dabei in eine einfache Explorationsstrategie (Algorithmus 6.5) und in eine Hypothesenverfolgung- und verschmelzung (Algorithmus 6.4).

Betrachte zunächst die Hypothesenverfolgung. Mit einem feature-basierten Ansatz werden mehrere Feature-Typen verwendet: Linien, Türen und Punkte. Feature-Detektoren berechnen dabei aus den Sensordaten automatisch die entsprechenden Features. Dabei unterscheidet man zwei Klassen von Features: Erzeugende Features sind seltener in der Karte verteilt, unerstützende Features hingegen sind sehr oft in der Kartenrepräsentation zu finden.

Im Matching-Schritt werden die Bildfeatures auf die Modellfeatures der Umgebungskarte gematcht. Dabei werden Kandidatenmatchings erzeugt, d.h. neue hypothetische Standorte des Roboters, bezeichnet mit C_k^j für den j-ten Standort zum Zeitpunkt k. Mittels eines Tests wird für jede schon erzeugte Hypothese aus vorherigen Zeitschritten H_k^i getestet, ob das Kandidatenmatching zur Hypothese passt oder nicht. Jede Hypothese hat dabei eine probabilistische Bewertung. Die Hypothese wird dementsprechend bestärkt im positiven Fall. Falls die Hypothesen unterschiedlich sind, wird eine neue Hypothese zur Hypothesenmenge hinzugenommen, wenn das Kandidatenmatching durch eine erzeugende Hypothese hervorgegangen ist.

Nach dem Bewertungsschritt wird die Hypothesenmenge darauf untersucht, ob alle Hypothesen gültig sind und ob sie eine gültige Bewertung besitzen. Zu schlecht bewertete Hypothesen, d.h. Hypothesen, welche unter eine Schwellenwertwahrscheinlichkeit fallen, werden gelöscht. Hypothesen, die ausserhalb der Umgebung liegen und solche die in Hindernissen liegen, werden ebenfalls gelöscht. Das Pruning

bringt dementsprechend auch eine Normalisierung der Wahrscheinlichkeitsverteilung mit sich.

Algorithmus 6.4: Multiples Hypothesentracking

Aufruf: **multi-hypotracking()**
Input: Karte m_g mit Modellfeatures G, Hypothesenmenge H_k
Output: Bewertete Hypothesenmenge H_{k+1}

```
1  Generiere Feature-Menge F aus den Sensordaten.
2  Generiere mögliche Kandidatenmatchings C_k aus F und G.
3  Berechne abhängig von der Bewegung neue Position aller Hypothesen.
4  for { C_k^j ∈ C_k} do
5     for { H_k^i ∈ H_k} do
6        if {C_k^j kann mit H_k^i gematcht werden} then
7           Update H_k^i
8        else
9           if {Das betrachete Feature ist ein erzeugendes Feature} then
10             Initialisiere neue Hypothese und füge sie H_k an.
11          end if
12       end if
13    end for
14 end for
15 H_{k+1}^i = H_k^i
16 Normalisiere die Bewertung.
17 Pruning des Hypothesenbaumes und erneute Normalisierung der Bewertung.
```

Die Explorationsstrategie wird folgendermaßen realisiert. Zu Beginn wird eine randomisierte Strategie verfolgt, bis eine Hypothese eine akzeptable Bewertung besitzt. Danach wird immer die beste Hypothese aus der Hypothesenmenge für diese eine Aktion gewählt.

Die Berechnung der nächsten Aktion wird wie folgt erreicht: Über die Karte wird eine topologische Karte gelegt, welche die Roadmap des Roboters repräsentiert. Jeder Knoten bekommt eine entsprechende Bewertung abhängig davon, wie viele und welche Features er sieht. Kanten zwischen Knoten bekommen ebenfalls eine Bewertung abhängig von ihrer Länge.

Die ausgezeichnete beste Hypothese wird als wahre Position angenommen. Dann wird zu dieser Position der nächste Knoten in der Karte ausfindig gemacht. Da diese eine Roadmap repräsentiert und der nächste Knoten angefahren werden soll, werden alle benachbarten Knoten des zur Hypothese nächsten Knotens abgesucht und bewertet. Kann der Knoten nicht angefahren werden, wird für die entsprechende Hypothese deren Bewertung herabgesetzt und die nächste beste bewertete Hypothese ausgewählt.

Algorithmus 6.5: Explorationsstrategie nach Jensfeld

Aufruf: **jensfeld-explore()**
Input: Karte m der Umgebung
Output: Standort p^* des Roboters

```
1  RANDOM:=true; ABBRUCH=true;
2  while {RANDOM} do
3     Führe randomisierte Bewegung durch.
4     Führe das multiple Hypothesentracking durch.
5     if {Eine Hypothese erreicht eine gute Bewertung} then
6        RANDOM:=false
7     end if
8  end while
9  while {ABBRUCH} do
10    Hole die am besten bewertete Hypothese h.
11    Berechne für diese die nächste beste Aktion.
12    Führe das multiple Hypothesentracking durch.
13    if {Eine Hypothese h_best erreicht eine gute Bewertung} then
14       ABBRUCH:=false
15    end if
16 end while
17 return p* = h_best
```

Dieser Ansatz versucht durch geschickten Einsatz unterschiedlicher Feature-Typen die Lokalisationszeit eines feature-basierten Verfahrens zu reduzieren. Außerdem ist der Ansatz des multiplen Hypothesentrackings erwähnenswert. Der Ansatz zur Elimination der Hypothesen hat eine entscheidene Schwäche: Es gibt keine Aussagen darüber, was mit wieder aktivierten Hypothesen passiert, die aus der Hypothesenmenge gefallen sind aber wieder auftauchen können. So ist ein Abbruch des Algorithmus nicht gewährleistet. Komplexitätsaussagen zu dem Algorithmus sind keine getroffen worden.

6.4 Autonome Exploration

Autonome Exploration bezeichnet die Problematik eines mobilen Roboters, autonom eine Umgebung abzufahren und dabei eine Karte zu erstellen [80,103,104]. Als Kartenrepräsentationen werden dabei sowohl Gitterkarten [64] als auch Segmentkarten [163] der Umgebung erstellt.

Die Lösung des Explorationsproblems 6.15 liefert gleichzeitig eine Lösung der globalen Selbstlokalisation, denn nach der Exploration kann der Roboter seine bisherige Umgebungskarte durch die lokale, explorierte Umgebungskarte ersetzen. In dieser lokalen Karte ist die Position des Roboters bekannt.

6.4.1 Problemstellungen und Lösungsansätze

Das Explorationsproblem kann man formal folgendermaßen beschreiben:

Problem 6.15 (Exploration) *Finde einen möglichst optimalen Weg* $W = (p_1, \ldots, p_t)$ *durch eine Umgebung, so dass jede Konfiguration der Umgebung mindestens einmal von Sensoren erfasst worden ist. Dabei soll nach Traversieren des Pfades eine Umgebungskarte* m_k *der Einsatzumgebung erstellt worden sein. Der optimale Weg sei dabei abhängig von einem Optimierungskriterium. Die Exploration ist* ***vollständig****, wenn zu einem Zeitpunkt* k *keine erreichbaren Konfigurationen der Umgebung mehr existieren, die eingesehen werden können.*

Als Optimalitätskriterien wird in der Literatur meistens die Pfadlänge angegeben, die zu minimieren ist [46]. Es sind aber auch andere Kriterien vorstellbar, etwa die Pfadlänge des maximal sichersten Pfades.

Das Explorationsproblem ist eine Erweiterung des Kartierungsproblems 3.14 um die folgenden Teilprobleme:

Auswahl der Sensorpunkten
Determiniere die Menge SP der Sensorpunkte in der lokalen Karte, an denen neue Informationen anfallen.

Auswahl einer neuen Aktion
Bestimme aus der Menge SP der Sensorpunkte eine neue Aktion a_k durch Auswahl eines Sensorpunktes sp.

Bewegungsplanung einer neuen Aktion
Plane die Bewegungsaktion a_k in der lokalen Karte und bewege den Roboter entlang einer geplanten Trajektorie (siehe Kapitel 5).

In der Literatur gibt es zu dieser Problematik zwei unterschiedliche Lösungsmethoden. Zum einen gibt es Ansätze der Algorithmischen Geometrie, die sich mit dem Entwurf von Algorithmen beschäftigen, welche eine möglichst kurze Rundtour in einer polygonalen Welt ohne Hindernisse berechnen [8, 98, 103, 104, 204]. Dabei geht es darum, alle Ecken eines Polygons mindestens einmal gesehen zu haben. Der Roboter wird als punktförmig angenommen und bewegt sich ohne Bewegungsfehler. Ferner besitzt er eine optimale, unbeschränkte und fehlerfreie Sicht im Polygon, so dass auch die Kartierung kein Problem bereitet. Auf der heuristischen Auswahl der Sensorpunkte und der Reihenfolge, wie das Polygon abgefahren wird, liegt dann der Augenmerk der Forschung.

Die andere Sichtweise dieser Problematik widmet sich der realistischen Kartierung. Dabei spielt die autonome Kartierung, also Exploration, eine eher untergeordnete Rolle, ist doch das Kartierungsproblem selber Bestandteil der Forschung (siehe Kapitel 3). Nennenswerte Ansätze zur Exploration sind daher rar, etwa [80, 213].

In dieser Arbeit wird eine realistische Variante des Problems betrachtet. Da die Kartierung (Kapitel 3) und die Bewegungsplanung (Kapitel 5) schon besprochen wurden, bleibt nur die Problematik offen, Sensorpunkte zu bestimmen und daraus jeweils einen Sensorpunkt auszuwählen.

6.4.2 Auswahl der Sensorpunkte

An einem Sensorpunkt sp kann der Roboter Bereiche einsehen, die noch nicht im Kartenmodell festgehalten wurden. An diesen Positionen werden neue Informationen akquiriert. Die Menge der Sensorpunkte, welche die aktuelle Karte m_k liefert, wird mit SP_k bezeichnet. Das Problem der Generierung von Sensorpunkten kann man folgendermaßen beschreiben:

Problem 6.16 (Generierung von Sensorpunkten) *Gegeben sei eine lokale Karte m_k zu einem Zeitpunkt k. Dann besteht das Problem der Generierung von Sensorpunkten darin, mit Hilfe der Kartenrepräsentation alle möglichen Sensorpunkte zum Zeitpunkt k aufzuspüren.*

Das Problem ist sehr stark von der Kartenrepräsentation abhängig. Für eine Gitterkarte sind solche Gitterpunkte Sensorpunkte, welche schon vom Roboter besucht wurden, die Eigenschaft *Frei* oder *Dynamisch* besitzen und in deren Nachbarschaft Gitterpunkte mit der Eigenschaft *Unbekannt* liegen.

Für eine Segmentkarten-Repräsentation wird die Berechnung schon viel schwieriger. Betrachte zunächst nur einen einzelnen Scan. Dann kann man das Scanpolygon definieren:

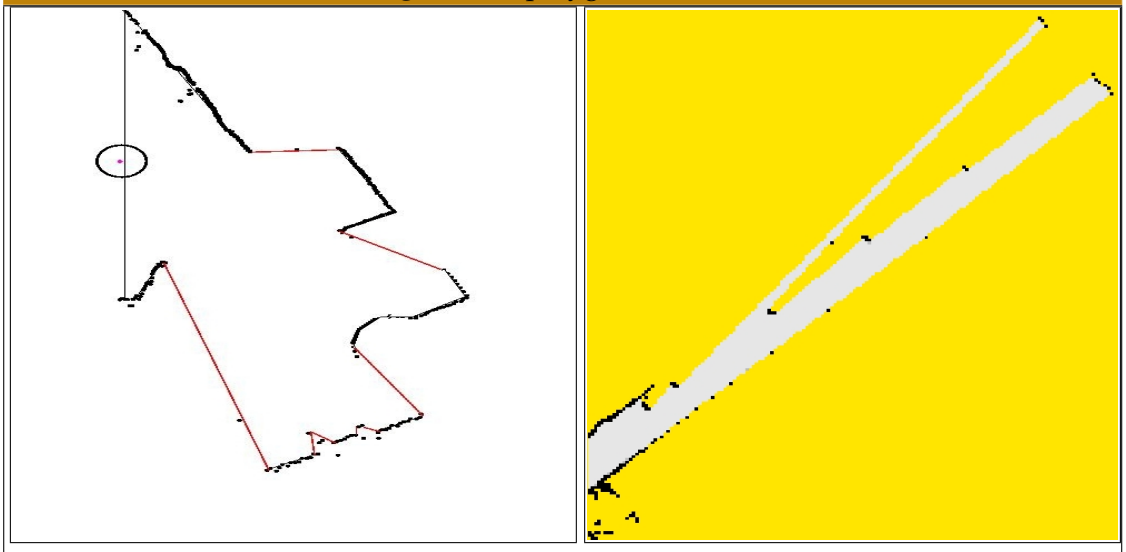
Definition 6.17 (Scanpolygon) *Ein Scanpolygon P_s aus dem Scan $s = \{s^{[0]}, \ldots, s^{[n]}\}$(in angularer Ordnung) besitzt alle Messpunkte $s^{[s_i]}$ als Ecken V und die Kantenmenge $E = \{(s^{[s_i]}, s^{[i+1]}) \mid 0 \leq i \leq n\} \cup \{(n, 0)\}$.*

Im Scanpolygon gibt es ausgezeichnete Kanten, bezeichnet als freie Kanten:

Definition 6.18 (Freie Kante) *Eine Kante $e \in E$ eines Scanpolygons P_s heißt freie Kante, wenn e keinem Objekt zugeordnet werden kann, d.h. die Länge von e ist größer als ein Schwellwert ϵ.*

Alle Punkte auf einer freien Kante sind Sensorpunkte. Eine freie Kante markiert den Übergang von einem explorierten freien Bereich zu einem unbekannten Bereich (siehe Abbildung 52). Da bei der Kartierung jedoch Scans iterativ in die Kartenrepräsentation eingefügt werden, müsste man bei einer Segmentkartenrepräsentation für jede freie Kante bestimmen, ob der aktuelle Scan diese Kante verändert hat, sodass sie wegfällt oder in mehrere Teile zerschnitten wurde, da der Roboter die Karte in dieser Richtung exploriert hat. Das hat den Effekt, dass der Berechnungsaufwand nicht mehr vertretbar wird.

Abbildung 52: Scanpolygon und freie Kanten



Im linken Bild kann man einen aufgenommenen Scan sehen, welcher durch ein Scanpolygon approximiert wird. Die roten Linien bezeichnen die freien Kanten des Scanpolygons. Im rechten Bild ist eine Gitterkartenrepräsentation zu sehen. Graue Gitterpunkte, welche an den gelben Bereich grenzen, sind potentielle Sensorpunkte.

Demzufolge ist eine Gitterkartenrepräsentation als einzige in der Lage, effizient die aktuelle Sensorpunktmenge zu berechnen.

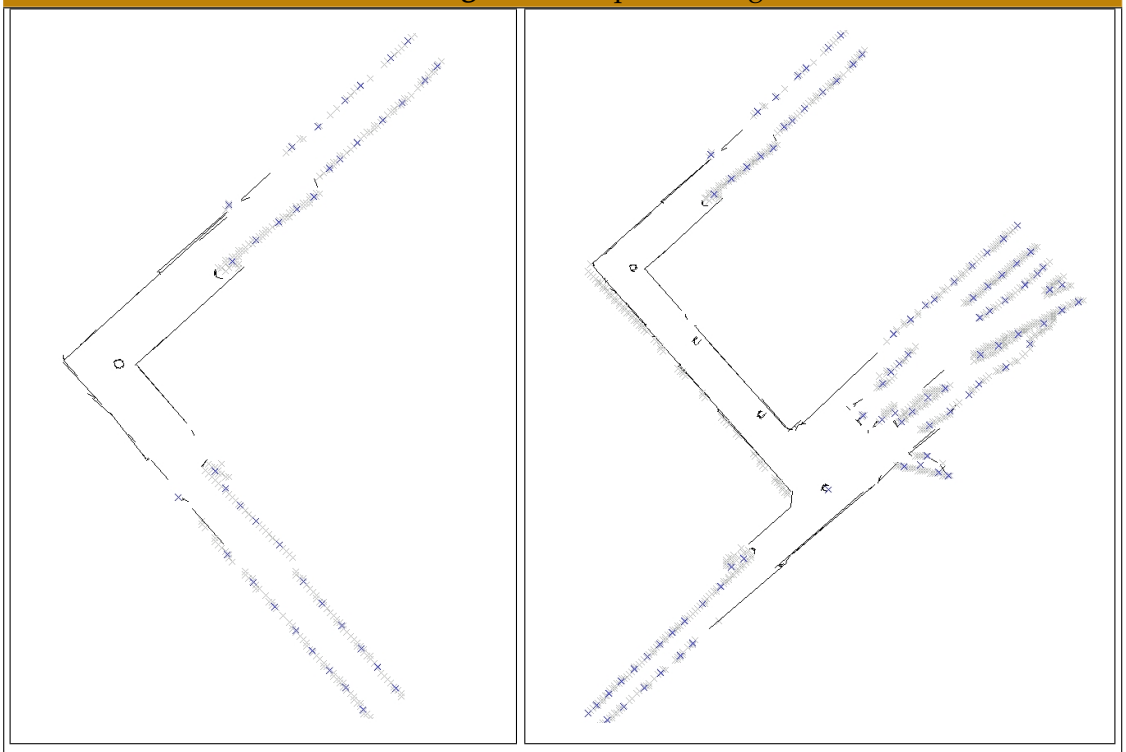
Da die aus Gitterkarten extrahierte Menge der Sensorpunkte SP_k sehr groß werden kann, ist es sinnvoll vor der Wahl der Bewegungsaktion zu einem ausgezeichneten Sensorpunkt diese Menge einzuschränken zu $\overline{SP_k}$. Die Reduktion kann nach einer der folgenden Heuristiken erfolgen:

Bereichsauswahl
Betrachte einen Sensorpunkt sp. Alle Sensorpunkte werden zusammengefasst, welche in einen Umkreis von sp mit dem Radius r liegen. Diese Sensorpunkt-Wolke wird dann durch ihren Schwerpunkt ersetzt. Damit werden auch Bereiche berücksichtigt, welche nur wenige Sensorpunkte besitzen.

Winkelauswahl
Alle Sensorpunkte werden nach ihren **Winkeln bezüglich des aktuellen Standortes** geordnet. Der Winkelbereich vom aktuellen Standort aus wird diskretisiert. und aus der geordneten Liste insgesamt n Sensorpunkte ausgewählt. Dies geschieht durch die Auswahl der Elemente mit dem Index $\lceil \frac{|SP_k|}{n} \cdot i \rceil$ für $0 \leq i < n$.

Distanzauswahl
Alle Sensorpunkte werden nach ihren **Distanzen in Bezug zum aktuellen Stand-**

Abbildung 53: Sensorpunktmengen



Zwei lokale Karten zu unterschiedlichen Zeitpunkten jeweils mit einer Sensorpunktmenge (graue Kreuze) und einer reduzierten Sensorpunktmenge (blaue Kreuze).

ort geordnet. Anschließend werden aus der geordneten Liste n Sensorpunkte verwendet, indem die Elemente mit dem Index $\lceil \frac{|S_t|}{n} \cdot i \rceil$ für $0 \leq i < n$ ausgewählt werden. Ein Beispiel findet sich in Abbildung 53.

Exemplarisch soll für die Bereichsauswahl die Zeit- und Speicherkomplexität hergeleitet werden. Dabei wird deutlich, wie stark die einzelnen in dieser Arbeit vorgestellten Probleme miteinander involviert sind.

Korollar 6.19 (Komplexität der Sensorpunktauswahl) *Der Algorithmus 6.6 zur Berechnung einer reduzierten Sensorpunktliste $\overline{SP}_k$ durch Bereichsauswahl hat eine Zeitkomplexität von $\mathcal{O}(P^2 + K \cdot C)$ und eine Speicherkomplexität von $\mathcal{O}(P^2 + D)$, wobei P die Anzahl der Sensorpunkte bezeichnet, K die Anzahl der Segmente der Kartenrepräsentation, C den Zeitaufwand zur Pfadplanung und D den Speicheraufwand.*

Beweis: Es sei eine Gitterkarte m^k_{grid} gegeben. Diese stellt in $\mathcal{O}(m^2)$ die Menge der Sensorpunkte zur Verfügung, wenn m die maximale Anzahl von Gitterpunkten in einer Dimension bezeichnet. Die eigentliche Berechnug erfolgt bei der Integration neuer Sensordaten und wird hier außer acht gelassen.

Algorithmus 6.6: Sensorpunktauswahl

Aufruf: **sensorpoint-selection()**
Input: Lokale Gridkarte m^k_{grid} zum Zeitpunkt k, Roboterkonfiguration p_k
Input: Radius r
Output: Reduzierte Menge von Sensorpunkte $\overline{SP}_k$

```
1  Berechne aus der Gridkarte die Menge SP_k.
2  while {Iteriere über alle sp ∈ SP_k} do
3     Eliminiere alle Sensorpunkte sp , die nicht von p_k aus erreichbar sind.
4  end while
5  while {Iteriere solange, bis SP = ∅} do
6     Hole den aktuellen Sensorpunkt sp_i aus SP_k.
7     area = ∅
8     while {Iteriere über alle restlichen Sensorpunkte aus SP_k} do
9        Sei sp_j der gerade aktuelle Punkt.
10       if { d(sp_i, sp_j) < r } then
11          Füge sp_j der Menge area an.
12          Lösche sp_j aus SP_k.
13       end if
14    end while
15    while {area! = ∅} do
16       Berechne den Schwerpunkt sp_new aller Punkte aus area.
17       Füge sp_new der Menge SP̄_k an.
18    end while
19 end while
20 return SP̄_k
```

Man benötigt weiter für die Pfadplanung $\mathcal{O}(C)$ Zeitschritte um für einen Sensorpunkt zu determinieren, ob er vom aktuellen Standort des Roboters aus erreichbar ist. Also ergibt sich bei P Sensorpunkten ein Aufwand von $\mathcal{O}(P \cdot C)$.

Die Sensorpunktauswahl durch Bereichsauswahl benötigt bei P Sensorpunkten $\mathcal{O}(P^2)$ Zeitschritte und genausoviele Speicherplätze. Mit dem Aufwand der Speicherung für die Pfadplanung ergeben sich die obigen Komplexitäten. □

6.4.3 Auswahl und Erreichen des besten Sensorpunktes

Die Auswahl des besten Sensorpunktes aus der Menge $\overline{SP}_k$ entscheidet eine Auswahlstrategie, abhängig von einer Bewertungsfunktion (siehe dazu auch Abschnitt 4.1). Als Bewertungsfunktionen kann man sich die folgenden vorstellen:

Distanz zur Roboterposition
Bestimme für jeden Sensorpunkt sp die euklidische Distanz zur aktuellen Roboterposition.

Weglänge zur Roboterposition
Bestimme für jeden Sensorpunkt sp die Pfadlänge zur aktuellen Roboterposition.

Gewichtung abhängig von der Sensorpunktreduktion
Gewichte die Sensorpunkte abhängig von der Anzahl der Sensorpunkte, aus welchen sie entstanden sind.

Informationsgewinn
Bewerte das Umfeld eines Sensorpunktes mit. Dabei kann man in der Gitterkarte die freien, unbesuchten Felder in einer Umgebung um einen Sensorpunkt zählen.

Die Auswahlstrategie unabhängig von einer Bewertungsfunktion kann folgendermaßen aussehen:

Beste Auswahl
Der Sensorpunkt mit der besten Bewertung wird ausgewählt.

Probabilistische Auswahl
Aus den Sensorpunkten wird ein Sensorpunkt abhängig von einer Wahrscheinlichkeitsverteilung gezogen. Dann ist jedoch eine Bewertung der Sensorpunkte überflüssig.

Weitere Bewertungen und Auswahlstrategien sind denkbar. Hier wird im weiteren die Distanz zu einem Sensorpunkt als Bewertungsfunktion und die minimale Distanz (d.h. beste Auswahl) als Auswahlstrategie verwendet.

6.4.4 Ablauf der Exploration

Aus den obigen Abschnitten ergibt sich nun der Algorithmus 6.7. Dabei ist zu berücksichtigen, dass die Exploration durch einen Prozess beschrieben wird (siehe auch Kapitel 8.2). Die Exploration benötigt zusätzlich, wie oben erwähnt, einen Navigationsprozess und einen Kartierungsprozess.

Der Explorationsprozess wird nach einer kurzen Bewegungssequenz mit einer initialen Karte der Umgebung m_k und der aktuellen Roboterposition in dieser Karte p_k gestartet. Abhängig von dieser werden Sensorpunkte produziert, ausgewählt und als Ziel einem Navigationsalgorithmus 5.9 übergeben. Dieser versucht den Sensorpunkt zu erreichen, wobei dabei gleichzeitig das Kartemodell nach Algorithmus 3.5 und Algorithmus 3.10 durch neue Sensordaten ausgebaut wird. Kann der Zielpunkt nicht erreicht werden, wird er als nicht erreichbar eingestuft und ein neuer Sensorpunkt ausgewählt. Ist ein Sensorpunkt erreicht worden, wird die gerade aktuelle Karte verwendet um eine neue Sensorpunktmenge zu gewinnen und einen neuen Sensorpunkt als Ziel auszuwählen. Dieser Prozess stoppt, wenn kein Sensorpunkt mehr erreicht werden kann oder wenn alle Sensorpunkte erreicht wurden und damit die Einsatzumgebung vollständig exploriert wurde. Algorithmus 6.7 repräsentiert einen einzigen Explorationsschritt.

Algorithmus 6.7: Explorationsschritt

Aufruf: **exploration-step()**
Input: Karte m_{k-1}, aktuelle Position p_{k-1}
Output: Karte m_k, Position p_k

```
1  Berechne Roadmap R
2  Berechne Sensorpunkt-Auswahl SP_k aus Sensorpunktmenge SP von m_{k-1}
3  Berechne die kürzesten Wege von p_k zu allen sp ∈ SP_k.
4  Suche das Minimum w_min über alle kürzesten Wege.
5  if { w_min exisistiert} then
6     // Bewegungsphase des Roboters
7     Lege neuen Sensorpunkt von w_min als Ziel fest.
8     while {Roboter bewegt sich} do
9        Bewege Roboter mit Navigationsalgorithmus von p_{k-1} nach p_k.
10       Kartiere während der Bewegungsphase die Umgebung.
11    end while
12    Liefere die neue Karte m_k und Endkonfiguration der Bewegung p_k zurück.
13 else
14    // Kein Pfad gefunden
15    Liefere die aktuellste Karte m_k = m_{k-1} und p_k = p_{k-1} zurück.
16 end if
```

Korollar 6.20 (Komplexität eines Explorationschritts) *Der Zeit- und Speicheraufwand für einen Explorationsschritt nach Algorithmus 6.7 wird durch den Aufwand für die Sensorpunktauswahl bestimmt.*

Beweis: Zusätzlicher Aufwand besteht offensichtlich darin, die kürzesten Wege zu allen Sensorpunkten zu bestimmen. Dieser Aufwand ist aber schon in der Sensorpunktsuche enthalten. Daneben muss über alle kürzesten Wege iteriert und der optimalste Weg bestimmt werden. Dieser Aufwand ist aber linear in der Anzahl der Wege. □

Der Aufwand für den kompletten Explorationsalgorithmus ist abhängig von der Einsatzumgebung und der Anzahl der benötigten Explorationsschritte.

6.5 Elimination durch Exploration und Feature-Matching

In diesem Abschnitt wird ein neues Verfahren zur Lösung des Hypothesenliminationsproblems durch Exploration und Feature-Matching (EFM) vorgestellt [187]. Grundbaustein des Verfahrens ist die autonome Exploration einer Karte. Damit besitzt man zu jedem Zeitpunkt k ein lokales Modell m_k der Umgebung. Die Informationen (Features) aus der lokalen und globalen Karte werden aus Effizienzgründen durch die Featureauswahl (siehe Kapitel 4) reduziert und dienen als Eingaben für ein Lokalisationsverfahren, das aus n Lokalisationsanfragen an n Teilkarten eine Hypothesenmenge H_{loc}^{k} erzeugt.

6.5.1 Idee des Verfahrens

Betrachtet man das statische Verfahren (Algorithmus 6.1) und verwendet als Lokalisationsverfahren L ein im ersten Teil dieser Arbeit hergeleiteten Algorithmus, so macht man folgende Beobachtung:

Das Lokalisationsverfahren L ist nicht mehr in der Lage, eine exakte Hypothesenmenge in einem Anfrageschritt aus Sensordaten mit Karte m_g zu generieren.

Daraus ergeben sich eine Reihe von Konsequenzen. Würde man die erzeugten Hypothesen des Lokalisationsverfahrens L verwenden, könnte der Eliminationsalgorithmus unter Verwendung eines Overlay-Baumes beliebig lange Verifikationspfade generieren, da die Hypothesenmenge nicht vollständig ist.

Könnte man in jedem Schritt genau eine Hypothese eliminieren und hätte am Schluß eventuell sogar noch eine Hypothese übrig, wäre die Berechnung der tatsächlichen Konfiguration des Roboters nicht sichergestellt, denn auch hier fehlt die Vollständigkeit der initialen Hypothesenmenge um zu entscheiden, ob dies tatsächlich die Roboterkonfiguration ist. Damit ist es notwendig den ursprünglichen Algorithmus umzustellen und auch andere Abbruchkriterien herzuleiten.

Betrachtet man die lokale Karte m_k , die durch ein Kartierungsverfahren erzeugt wird, kann man einige Schlußfolgerungen bezüglich der Hypothesenelimination ziehen. Diese repräsentiert die lokale Karte, die der Roboter unabhängig von anderen Informationen kennt. Sie modelliert die Sichtbarkeit, die alle hypothetischen Standorte des Roboters besitzen müssen.

Vergleicht man diese Sichtbarkeit mit dem ursprünglichen Eliminationsverfahren, so repräsentiert die lokale Karte nur einen Teil des Overlay-Baumes, da dieser aus der gegebenen Umgebungskarte m_g berechnet wurde und somit zusätzliche sichtbare Bereiche enthalten kann, welche der Roboter nicht gesehen hat.

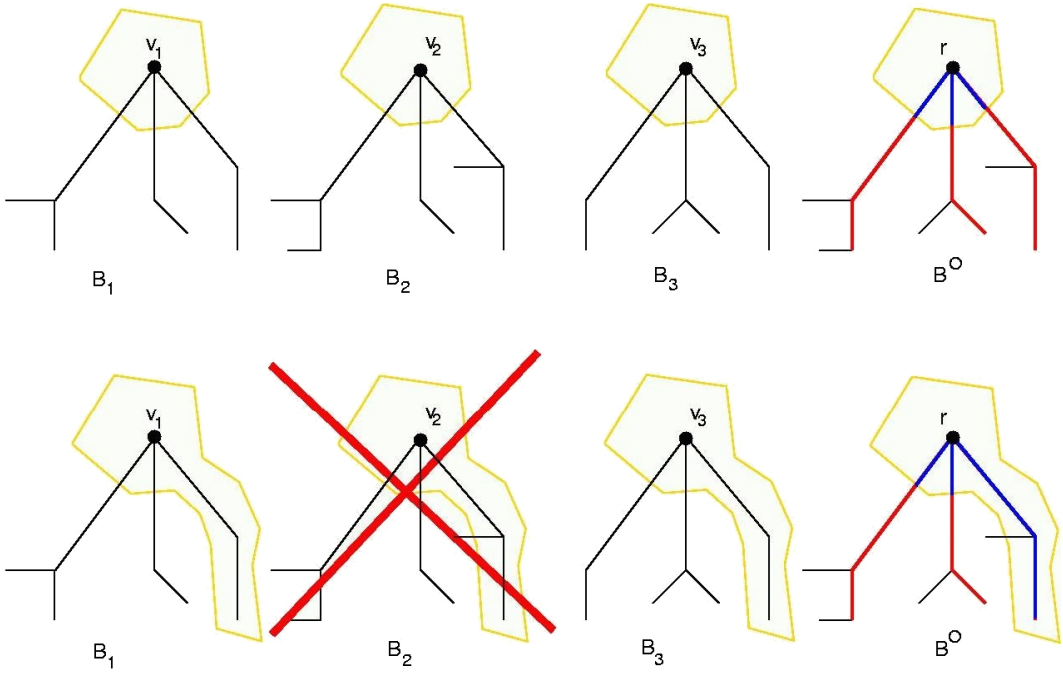
Allerdings gibt es keine Möglichkeit, den genauen Overlay-Baum zu berechnen, wenn nicht alle Hypthesen berechnet werden können. Demzufolge hat man die Möglichkeit, aus den vorhandenen, unsicheren und eventuell falschen Hypothesen einen Overlay-Baum zu generieren, bei welchem Teile des Baumes nicht im tatsächlichen Overlay-Baum vorkommen. Dann muss man in weiteren Schritten die falschen Hypothesen identifizieren und zusätzliche Sensorinformationen in den Overlay-Baum integrieren. Dieser Ansatz wurde durch die aktive Monte-Carlo-Lokalisation verfolgt, allerdings ohne einen Overlay-Baum auch tatsächlich zu berechnen. Bei der tatsächlichen Berechnung [34,36] ergeben sich enorme Probleme. Man muss dabei sicherstellen, dass Trajektorien in Kartenbereichen gleicher Sicht auch entsprechend gleich berechnet werden, was aber durch kleinste Störungen in der Umgebung nicht sichergestellt werden kann. Man kann einen zweiten Ansatz verfolgen, indem man lokale Karten

und damit einen tatsächlich sicheren Teil des Overlay-Baumes verwendet, der aber eingeschränkt ist, und mit diesem Hypothesen erzeugen und bewerten.

Der folgende Ansatz repräsentiert diese Möglichkeit. Die Berechnung des Overlay-Baumes geschieht dabei allerdings nicht direkt. Da die Kartierung schon eine elegante Möglichkeit bietet, die konsistente Sicht aller Hypothesen zu repräsentieren, kann man in der entstandenen lokalen Karte beliebige Trajektorien zu entsprechenden Sensorpunkten generieren.

Für die Auswahl und die Berechnung von kürzesten Wegen zu den Sensorpunkten hilft der im vorherigen Kapitel beschriebene Ansatz der autonomen Exploration weiter. Damit hat man nun eine Berechnungsvorschrift für einen sogenannten lokalen Overlay-Baum.

Abbildung 54: Beispiel eines lokalen Overlay-Baums



Der Sichtbereich des Roboters ist markiert dargestellt. Über die drei hypothetischen Positionen besitzt der Roboter keine Informationen, er kennt lediglich seinen Sichtbereich. Der lokale Overlay-Baum ist der blaue Bereich des rot-markierten unbekannten Overlay-Baums (oben rechts). Nachdem sich der Roboter bewegt und damit seine Sicht vergrößert hat, ist damit auch der lokale Overlay-Baum gewachsen. Eine hypothetische Position konnte außerdem schon ausgeschlossen werden.

Definition 6.21 (Lokaler Overlay-Baum) *Ein lokaler Overlay-Baum ist ein geometrischer Baum, welcher ein Teilbaum des Overlay-Baums für eine Hypothesenmenge* $H^k \subseteq H^s$ *einer unbekannten Hypothesenmenge* H^s *ist.*

Die verwendete lokale Karte, und damit der lokale Overlay-Baum, besteht nun nicht aus mehreren geometrischen Bäumen, die übereinandergelegt wurden, sondern theoretisch aus einem berechneten Baum, welcher eine Teilmenge des tatsächlichen Overlay-Baumes repräsentiert. Außerdem beschreibt die lokale Karte nur eine Hypothesenmenge und deren lokale Sicht, aber nicht wie diese Menge zu erzeugen ist. Ein Beispiel für einen lokalen Overlay-Baum liefert Abbildung 54.

Die Erzeugung der Hypothesen geschieht bei Erreichen eines Sensorpunkts durch eine feature-basierte Lokalisationsanfrage mit den Features der lokalen Karte als Bildfeatures und den Features der globalen Karte als Modellfeatures (siehe auch Algorithmus 4.4). Damit erzeugt man in jedem Eliminationsschritt eine neue Hypothesenmenge, welche mit den Hypothesen der vorherigen Zeitschritte bewertet werden muss (siehe Kapitel 6.5.2).

Die Menge der Hypothesen bezieht sich dann auf den initialen Standort der lokalen Karte in Weltkoordinaten. Die Konfiguration des Roboters p^* kann anhand der bekannten lokalen Konfiguration p_m in der Karte m_k und der initialen Konfiguration der lokalen Karte bestimmt werden.

Dieser Ansatz verwendet also in jedem Zeitschritt alle bisherigen Sensorinformationen (reduziert auf informative Features), um die hypothetische Position des Roboters zu generieren, im Unterschied zu bisherigen Ansätzen (Kapitel 6.3.1 und 6.3.2), welche ihre Entscheidungen aufgrund weniger Sensormeßwerte an einer Konfiguration treffen. Die offen gebliebenen Fragen der Hypothesenbewertung und besonders auch des Abbruchkriteriums des Algorithmus werden in den beiden nächsten Abschnitten geklärt.

6.5.2 Bewertung von Hypothesen

Die aus einer Lokalisationsanfrage erzeugten Hypothesen sollen gewichtet werden und auf diese Weise in jeder Iteration die beste Hypothese identifizieren. Das Verfahren soll abbrechen, wenn diese Hypothese gut genug ist. Mittels einer Bewertungsfunktion sollen nun die Hypothesen aus $H = H_1 \dots H_n$, die aus einer Lokalisationanfrage mit reduzierter Featuremenge (siehe Kapitel 4) entstanden sind, mit allen Informationen aus den Modell- und Bildfeatures überprüft werden. Dazu wird zunächst der Begriff der bewerteten Hypothese eingeführt, welcher den Begriff der Hypothese nach Definition 1.25 erweitert.

Definition 6.22 (Bewertete Hypothese) *Eine bewertete Hypothese sei definiert als* $h = (h_p, h_q, h_n, h_w)$. *Dabei sei* h_p *die Konfiguration der erzeugten Hypothese,* h_q *die noch zu definierende Qualität und* h_n *die Anzahl der Hypothesen aus der* h *hervorgegangen ist. Schließlich bezeichnet* h_w *die Wahrscheinlichkeitsbewertung der Hypothese.*

Zunächst sollen die erzeugten Hypothesen solchermaßen eingeschränkt werden, dass nur diejenigen für weitere Berechnungen zugelassen werden, welche ein adäquates Matching mit einem geringen Matchingfehler liefern. Betrachte dazu nun eine einzelne Hypothese $h \in H$, welche durch eine Lokalisationsanfrage entstanden ist. Es sei F eine nach h transformierte Bildfeature-Menge und G eine Modellfeature-Menge bei gegebener Hypothese h. Dann bezeichnet $\delta_{f_i} = \min_j d(f_i, g_j)$ die minimale Distanz dieses Features f_i von G. Damit ergibt sich $\sum_h = \sum_i \delta_{f_i}$ die minimale Distanz der Feature-Mengen F und G.

Demzufolge kann man die mittlere Distanz $\overline{d_h} = \frac{\sum_h}{|F|}$ definieren. Somit hat man zunächst eine meßbare Differenz für eine erzeugte Hypothese h, welche mittels eines Qualitätskriteriums verwendet werden kann:

Definition 6.23 (Qualität einer erzeugten Hypothese) *Für eine Hypothese $h \in H$ bezeichnet $h_q = 1 - \frac{\overline{d_h}}{\gamma}$ die Qualität der Hypothese h.*

Ist $h_q \leq 0$, so wird h nicht zur bewerteten Hypothesenmenge H^k_{loc} hinzugenommen. Somit gilt für alle Hypothesen $h \in H^k_{loc}$, dass ihre Qualität $h_q \in]0, 1]$ liegt. Der Parameter γ ist ein empirischer Faktor und repräsentiert einen Schwellenwert, ab welchem eine Hypothese verworfen wird. Es hat sich ein Faktor von $\gamma = 0.8$ bewährt.

Algorithmus 6.8: Verifkation einer Hypothesenmenge

Aufruf: **global-verify()**
Input: Hypothesenmenge H, Bildfeature-Menge F, Modellfeature-Menge G
Output: Hypothesenmenge, bestehend aus bewerteten Hypothesen H^k_{loc}

```
1  H^k_loc = ∅
2  for {h ∈ H} do
3     Transformiere F nach h
4     Initialisiere Σ_h = 0
5     for {f_i ∈ F} do
6        Berechne δ_{f_i}.
7        Σ_h+ = δ_{f_i}
8     end for
9     Berechne mittlere Distanz d̄_h = Σ_h / |F|.
10    Berechne die Qualität h_q = 1 − d̄_h / γ.
11    if { h_q > 0} then
12       H^k_loc = H^k_loc ∪ h̄ := (h, h_q, 1, 0)
13    end if
14 end for
```

Korollar 6.24 (Komplexität der Verfikation) *Der Zeitaufwand zur Verifikation einer Hypothesenmenge nach Algorithmus 6.8 mit M Hypothesen liegt bei n Bildfeatures und m Modellfeatures bei $\mathcal{O}(M \cdot m \cdot n^{3/4})$ und unterscheidet sich nicht vom Aufwand*

der Verifikation feature-basierter Verfahren. Zusätzlicher Speicheraufwand von $\mathcal{O}(M)$ wird für die Speicherung der Hypothesenmenge benötigt.

Der obige Algorithmus unterscheidet sich von feature-basierten Verifikationsmethoden darin, dass dort die Verifikation über die Eingabe des Lokalisationsverfahrens geschieht. Dadurch kann unter Umständen bei Verwendung von Pruningmethoden zur Effizienzsteigerung nur ein Teil der Modellfeature-Menge in den Verifikationsprozess einfließen. Hier wird garantiert, dass immer die komplette zur Verfügung stehende Menge der Modellfeatures verwendet wird. Für die Komplexitätsbetrachtung ergibt sich dadurch aber kein Unterschied.

6.5.3 Verschmelzen von Hypothesenmengen

Für die Bewertung der Hypothesen ist desweiteren eine Betrachtung der Historie einer Hypothese sinnvoll. Dafür benötigt man die relative Häufigkeit als zweites Bewertungskriterium einer Hypothese mittels

$$h_h = \frac{h_n}{\sum_{\forall h' \in H} h'_n} \tag{6.9}$$

Mit der Qualität und relativen Häufigkeit wird eine Wahrscheinlichkeitsverteilung über die Güte aller Hypothesen berechnet. Somit gibt es für jeden Zeitpunkt k des Verfahrens eine bewertete Hypothesenmenge, die alle bis zum Zeitpunkt k erzeugten Hypothesen beschreibt. Daraus ergibt sich eine inkrementelle Berechnungsvorschrift für die Hypothesenelimination:

$$H^k = \Phi(H^{k-1} \cup H^k_{loc}) \tag{6.10}$$

Die Menge H^k_{loc} repräsentiert dabei die bewertete, zum aktuellen Zeitpunkt durch ein Lokalisationsverfahren L erzeugten Hypothesenmenge.

Die Menge H^k bestünde bei einer einfachen Vereinigung der Hypothesenmengen nun aus einer Vielzahl von Hypothesen, wobei viele Hypothesen redundante Informationen besäßen, könnte man sie doch zu einer Hypothese zusammenfassen. Bei der Vereinigung dieser Menge muss man also die Unsicherheiten bei der Erzeugung der Hypothesen berücksichtigen. Dabei ist eine Reduzierung der Menge durch Zusammenfassen ähnlicher Hypothesen notwendig, welches die Abbildung Φ erledigen soll.

Dazu benötigt man den Begriff der Ähnlichkeit einer Hypothese. Dieser kann wie folgt definiert werden

Definition 6.25 (Ähnliche Hypothesen) *Zwei Hypothesen h_i und h_j heißen ähnlich, falls ihre euklidische Distanz $d(h_i, h_j)$ kleiner als ein Schwellenwert e^{max}_{pos} und ihre Winkeldistanz θ kleiner als ein Schwellenwert e^{max}_{orient} ist.*

Siehe dazu auch Kapitel 2.7. Demzufolge kann man auch ganze Mengen von ähnlichen Hypothesen definieren:

Definition 6.26 (Ähnliche Hypothesenmengen) *Eine Hypothesenmenge* H *heißt ähnliche Hypothesenmenge, wenn gilt:* $\exists h \in H\ \forall \hat{h} \in H : h$ *ist ähnlich zu* $\hat{h}$.

Mit den obigen Definitionen kann man nun eine Hypothesenmenge zu einer Hypothese verschmelzen:

Definition 6.27 (Hypothesenverschmelzung) *Die Abbildung* $\phi : H^n \to H$ *heißt Hypothesenverschmelzung einer Hypothesenmenge* H *zu einer Hypothese* h. ϕ *ist definiert durch* $\phi(h^0, \ldots, h^n) = \overline{h}$ *mit* $\overline{h_p}$ *als dem Schwerpunkt der Positionen* $h^0, \ldots, h^n$, $\overline{h_n} = \sum_{i=0}^{n} h_n^i$ *die Summe aller Häufigkeiten und* $\overline{h_q}$ *die Qualität der zuletzt erzeugten Hypothese.*

Jetzt kann die Abbildung Φ definiert werden:

Definition 6.28 (Ähnliche Hypothesenverschmelzung) *Die Abbildung* Φ *heißt ähnliche Hypothesenverschmelzung einer Hypothesenmenge* $\mathcal{H} = H^{k-1} \cup H^k_{loc}$ *zu einer Hypothesenmenge* H^k *mit* $\Phi(\mathcal{H}) = \{\phi(\mathcal{H}_0) \ldots \phi(\mathcal{H}_n)\} = H^k$. *Dabei ist* $\mathcal{H}_0 \ldots \mathcal{H}_n$ *eine disjunkte Aufteilung von* $\mathcal{H}$ *in ähnliche Hypothesenmengen.*

Aus der so entstandenen Hypothesenmenge H^k lässt sich nun die Wahrscheinlichkeit h_w jeder einzelnen Hypothese $h \in H^k$ bestimmen. Dazu wird die bereits beschriebene Qualität h_q und die relative Häufigkeit h_h verwendet:

$$h_w = h_q \cdot h_h^2 \tag{6.11}$$

Die Werte h_w müssen anschließend für die gesamte Menge normalisiert werden, damit sich die Gesamtwahrscheinlichkeit zu 1 aufsummiert.

Nun soll eine Hypothese als für den Roboter akzeptabel gelten, wenn die Bewertung h_w über einem bestimmten Schwellwert γ_{loc} liegt. Dieser Parameter ist experimentell zu bestimmen.

6.5.4 Abbruchkriterien der globalen Selbstlokalisation

In den obigen Abschnitten wurden verschiedene Algorithmenteile der Hypothesenelimination vorgestellt, wie sie hier praktiziert werden soll: durch Exploration und durch Feature-Matching. Zusammenfassend gibt es nun die folgenden Abbruchkriterien:

Exploration bricht ab
: Dies ist dann der Fall, wenn keine erreichbaren Sensorpunkte, generiert durch die autonome Exploration, mehr vorhanden sind. Dann ist die Einsatzumgebung vollständig exploriert und damit trivialerweise die Roboterkonfiguration p^* bekannt.

Beste Hypothese gefunden
: Eine Hypothese erreicht nach einer Reihe von Iterationsschritten eine Bewertung,

welche über einem einzustellenden Schwellenwert liegt. Dann akzeptiert der Roboter diese Hypothese als seinen tatsächlichen Standort. Als Schwellenwert hat sich $\gamma_{loc} = 0.65$ als zuverlässig bewährt. Erfüllt mehr als eine Hypothese das Kriterium, wird die Hypothese mit der besten Bewertung ausgewählt.

Algorithmus 6.9: Globale Selbstlokalisation durch EFM

Aufruf: **exploration-feature-matching()**
Input: Globale Karte m_g
Output: Position des Roboters p^*

1 Berechne Feature-Menge G_{m_g} der globalen Karte m_g.
2 m_0 = first-scan; $p_0 = (0, 0, 0)$, $H^0 = \emptyset$, $t = 0$, iterate=true;
3 **while** iterate **do**
4 t++;
5 $(m_k, p_k) =$ autonomous-exploration(m_{k-1}, p_{k-1})
6 **if** { $m_k = m_{k-1}$ AND $t! = 1$} **then**
7 $p_m = \max_{h_w}\{h \in H^{k-1}\}$; iterate = false;
8 **else**
9 Berechne lokale Segmentkarte mit Feature-Menge F_{m_k}.
10 Berechne Lokalisationsanfrage H^k_{loc} =localize(F_{m_k}, G_{m_g}).
11 Verifiziere Hypothesen, berechne h_q $\forall h \in H^k_{loc}$.
12 Verschmelze Hypothesen, berechne H^k.
13 **if** es existiert $h \in H^k$ mit $h_w > \gamma_{loc}$ **then**
14 iterate := false;
15 $p_m := \max_{h_w}\{h \in H^k \mid h_w > 0.65\}$.
16 **end if**
17 **end if**
18 **end while**
19 Berechne $p^* = p_m + p_k$.

Algorithmus 6.9 liefert dann eine Lösung des globalen Selbstlokalisationsproblems durch Exploration und Feature-Matching. Der Algorithmus arbeitet folgendermaßen: Zu Beginn benötigt der Algorithmus eine kurze Startbewegung bzw. einen initialen Scan, um eine erste lokale Karte für eine Anfrage zu verwenden. Dann wird im Explorationsteil durch Algorithmus 6.7 ein Sensorpunkt und ein Weg zu diesem bestimmt. An einem Sensorpunkt angekommen wird die bis dorthin explorierte Karte in Segmentkartenrepräsentation zur Featuremenge F_{m_k} berechnet. Damit und mit einer Featurerepräsentation der globalen Karte G_{m_g} wird eine feature-basierte Anfrage localize(F_{m_k}, G_{m_g}) durchgeführt. Die resultierenden Hypothesen der Anfrage werden verifiziert und bewertet, sodass die Menge H^k_{loc} bewerteter Hypothesen resultiert. Anschließend werden ähnliche Hypothesen zusammengefasst. Eine probabilistische Bewertung testet daraufhin, ob es mindestens eine Hypothese gibt, die als hypothetischer Standort vom Roboter akzeptiert werden kann. Ist das nicht der Fall, müssen neue Informationen akquiriert werden, indem ein weiterer Verifikationspfad in der

lokalen Karte m_k berechnet wird. Gibt es keine erweiterbaren Kartenteile mehr, d.h. keine Sensorpunkte, bricht der Algorithmus ebenfalls ab.

Der Speicher- und Zeitaufwand wird durch die Verifikation der Hypothesen, die Auswahl der Sensorpunkte und durch die feature-basierten Lokalisationsanfrage bestimmt, sowie durch die Anzahl der benötigten Iterationen und ist von den jeweiligen eingesetzten Algorithmen abhängig.

6.6 Praktische Ergebnisse und Vergleiche

Die in diesem Kapitel vorgestellten Verfahren werden in zwei verschiedenen Studien vorgestellt, die die obigen Algorithmen im praktischen Einsatz experimentell bestätigen. Zum Einsatz kam dabei der in Anhang D vorgestellte Pioneer-Roboter mit entsprechender Sensorik. Dieser wurde in zwei Umgebungen, der schon bekannten Informatik-Hanggeschoss-Umgebung und der Mathematik-Erdgeschoss-Umgebung getestet.

6.6.1 Exploration

Exploration ist eine Möglichkeit, das Problem der globalen Selbstlokalisation zu lösen, wenn auch nicht die effektivste. Man muss die komplette Umgebung explorieren, um dann die globale Karte durch die lokal-explorierte Karte mit dem Roboterstandort ersetzen zu können. In jeder Einsatzumgebung wurden 10 Versuche durchgeführt und zusätzlich eine mit minimalem Aufwand durch den Anwender generierte Joystick-Fahrt, um ein Vergleichskriterium für die Güte der Exploration zu haben. In jedem Versuch wurden die folgende Parameter gemessen:

Zurückgelegte Weg
: Der zurückgelegte Weg anhand der abgefahrenen Teilziele in der aufgenommenen Karte wurde gemessen.

Dauer des Explorationsvorgangs
: Die Dauer des Explorationsvorgangs vom Start bis zur Ausgabe des Endergebnisses wurde gemessen.

Explorierte Fläche
: Die explorierte Fläche sollte als Maß dienen, wieviel Fläche der Roboter exploriert hat. Da aber Glasflächen und Reflexionen mal mehr, mal weniger zusätzliche Fläche bedeuteten, war dieser Parameter wenig aussagekräftig.

Anzahl der Explorationsschritte
: Die Anzahl der Explorationsschritte und damit die Anzahl der ausgewählten Sensorpunkte wurde gezählt.

Komplexität der Karten
Die Komplexität der Resultatkarte wurde gemessen durch die Anzahl der in die Karte integrierten Scans und die Anzahl der extrahierten Segmente der Segmentkarte.

Als Testverfahren diente der Explorationsalgorithmus 6.7 mit angeschlossener iterativer Kartierung (Algorithmus 3.5) und angeschlossenem Navigationsalgorithmus 5.9 unter Verwendung probabilistischer Roadmaps. Als Sensorpunktauswahl diente die Bereichsauswahl. Die Startpunkte des Roboters wurden für jeden Explorationstest vom Versuchsleiter zufällig bestimmt.

Die jeweilige Einsatzumgebung benötigte zum Einsatz als Voraussetzung eine Funküberdeckung. Teilweise mussten sicherheitskritische Bereiche (Treppenabgänge) durch Hindernisse versperrt werden, da die Sensorik diese nicht erkennen konnte. Ansonsten wurden den Testumgebungen keinerlei Beschränkungen auferlegt. Insbesondere waren dynamische Objekte während der Testzeit zugegen.

Exploration im Informatikgebäude

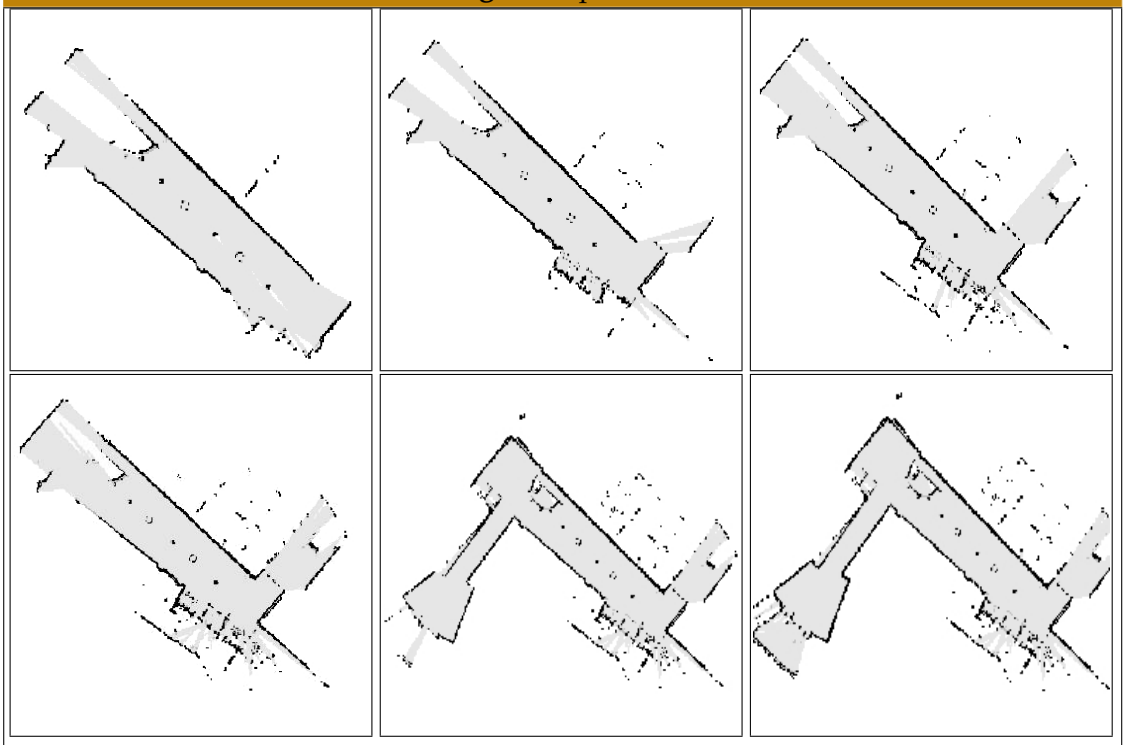
Die Informatikumgebung besitzt eine relativ große freie Explorationsfläche, mit Stühlen und Säulen durchsetzt und viel Publikumsverkehr. An zwei Seiten sind große Fensterflächen vorhanden, sodass der Roboter viele Sensorpunkte wahrnehmen kann, die nicht erreichbar sind. Zudem gibt es Sichtfenster zu angeschlossenen Seminarräumen, die einen Einblick für das mobile Gerät ermöglichen.

Tabelle 2: Explorationsergebnis in Testumgebung 1

Name	Weg [m]	Dauer [min]	expl. Fläche [m^2]	Anzahl der Schritte	Scananzahl	Segmentanzahl
Joystick-Fahrt	111,04	6	398,72	-	228	459
1	102,4	11	324,74	19	486	450
2	100	12	437,25	21	576	400
3	140	18	388,5	32	480	600
4	111,01	11	394,16	19	420	526
5	113,8	11	407,09	14	383	619
6	167	16	388,78	31	685	580
7	96,6	13	409,6	15	375	584
8	110,3	10	425	18	449	541
9	103,8	12	388,46	18	262	731
10	101,8	17	344,43	30	300	739
Durchschnitt	114,67	13,1	390,8	21,7	441,6	577

Die 10 Versuchsfahrten in der Informatik-Umgebung benötigten bei einer durchschnittlichen Weglänge von 114 Metern 13 Minuten Explorationszeit (siehe Tabelle 2). Bei den 10 Versuchen fällt auf, dass die explorierte Fläche in manchen Versuchen (2, 10)

Abbildung 55: Explorationstest 1



In dieser Abbildung ist die erzeugte explorierte Karte des Roboters zu verschiedenen Zeitschritten zu sehen. Zu Beginn (oben links) wird der rechte Bereich exploriert. Sodann fährt der Roboter zur linken Seite des Raumes bis die Exploration beendet ist (unten rechts).

sehr gering ausfällt. Das lässt darauf schließen, das diese Versuche durch unerreichbare Sensorpunkte frühzeitig beendet wurden. Weiter kann man schließen, dass die Exploration per Hand mit Hilfe eines Joysticks immer noch die schnellste Variante ist, die Umgebungskarte zu erstellen. Die besten Versuche brauchen etwa doppelt so lange wie die Joystickfahrt. Die Explorationsgeschwindigkeit des Roboters war auf 150 cm pro Sekunde festgelegt um gute Explorationsergebnisse zu gewährleisten. Diese langsame Geschwindigkeit resultiert aus der langsamen Scanfrequenz (3 Hz) des Laser-Entfernungsmessers und der Echtzeit-Fähigkeit des Systems bei der Integration der Sensordaten. Somit waren alle Versuche in der Lage, eine adäquate kartierte Umgebungskarte zu liefern. Abbildung 55 zeigt einen Explorationsversuch zu verschiedenen Zeitpunkten.

Exploration im Mathematikgebäude

Die Mathematik-Einsatzumgebung unterscheidet sich grundlegend von der vorherigen Einsatzumgebung. Die Gänge sind schmal und lang, es gibt eine Vielzahl von ange-

schlossenen Türen. Säulen liefern noch engere Gänge und Spalten, die für den Roboter nicht befahrbar sind. Zudem ist die Einsatzumgebung zyklisch, sodass die Anzahl der Partikel und die Fehlerradien erhöht werden müssen, um mittels des inkrementellen Verfahrens diesen Zykel bewältigen zu können.

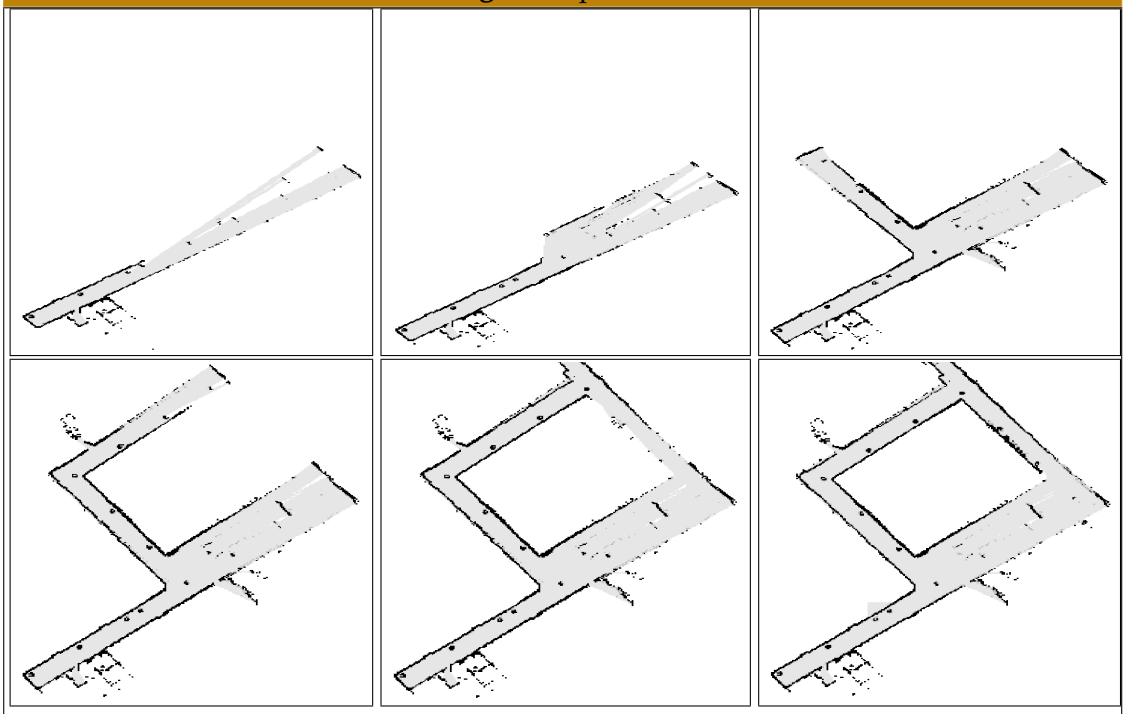
Tabelle 3: Explorationsergebnis in Testumgebung 2

Name	Weg [m]	Dauer [min]	expl. Fläche [m^2]	Anzahl der Schritte	Scananzahl	Segmentanzahl
Joystick-Fahrt	110,8	4	274,32	-	250	560
1	119,7	7	323,66	22	614	427
2	140	17	280,94	35	750	527
3	102,6	20	224,73	44	600	348
4	98,6	12	327,78	31	565	434
5	109,19	13	259,67	32	600	521
6	74	11	265,07	28	350	525
7	89,6	11	266,96	31	454	566
8	122,5	15	285,89	38	711	467
9	126,9	17	264,62	42	728	489
10	104,8	14	270,72	30	593	479
Durchschnitt	108,79	13,7	277	33,3	596,5	478,3

Die 10 Versuchsfahrten in der Mathematik-Umgebung benötigten bei einer durchschnittlichen Weglänge von 109 Metern insgesamt 13 Minuten, siehe Tabelle 3. Wiederum schwankt die Anzahl der explorierten Fläche gewaltig, was darauf zurückzuführen ist, dass Türen offen oder geschlossen waren und somit mehr mögliche Umgebung exploriert werden konnte.

In dieser Umgebung zeigt sich deutlich, dass der Roboter Schwierigkeiten mit den schmalen Gängen hat. Trotz der niedrigen Geschwindigkeit muss ein Sicherheitsbereich für den Roboter deklariert werden, welcher in schmalen Gängen fast die gesamte Korridorbreite ausmacht. Ein flüssiges Navigieren des Roboters war also nicht möglich. Zum anderen ist der „Intelligenzlevel" des Systems in der Wahl der nächsten Sensorpunkte, gerade bei sehr ausweglosen Situationen noch sehr gering. Durch meistens sehr kurze Bewegungsschritte wurde deshalb sehr viel Zeit verschwendet und im Durchschnitt die dreifache Explorationszeit im Vergleich mit der Joystick-Fahrt benötigt. Ein besseres, schnelleres Sensorsystem mit erhöhten Reaktionszeiten und erhöhter Geschwindigkeit wäre als Vergleich, ob mobile Roboter mit engen Räumlichkeiten umgehen können, wünschenswert. Insgesamt konnten trotzdem adäquate Kartierungsergebnisse erzielt werden. Abbildung 56 zeigt einen Explorationsversuch zu verschiedenen Zeitpunkten.

Abbildung 56: Explorationstest 2



Dieser Explorationstest findet in einer zyklischen Umgebung statt. Wiederum sind verschiedene Zeitschritte einer Explorationsfahrt abgebildet. Vom Start (oben links) fährt der Roboter nach rechts, anschließend links den Gang hinunter um t dann schließlich den Zykel (unten rechts) zu schließen.

6.6.2 Selbstlokalisation durch Exploration und Feature-Matching

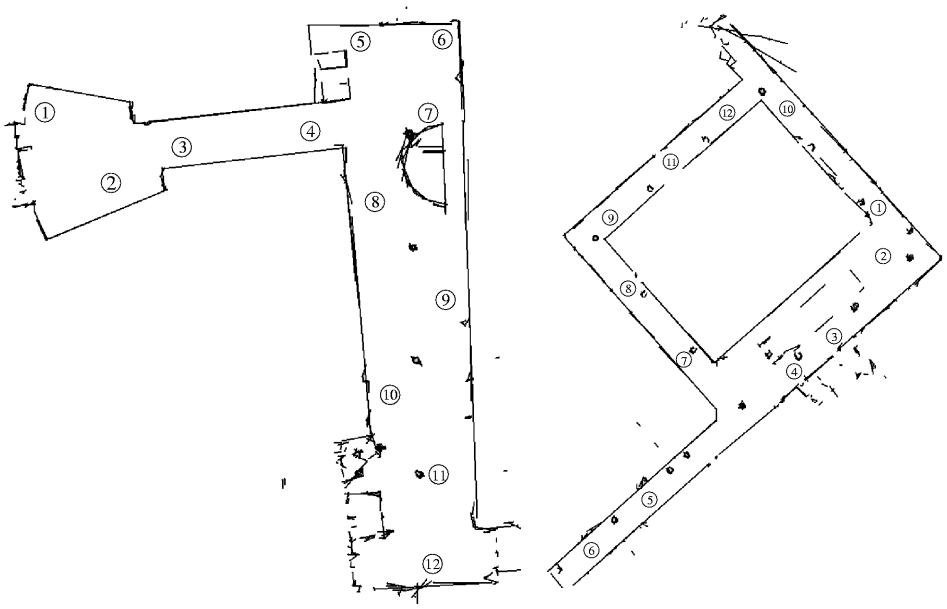
Neben der Exploration sollte vor allem das neu entwickelte Verfahren mittels Exploration und Feature-Matching zum Einsatz kommen. Es wurden in jeder Einsatzumgebung 12 Versuche in zwei Versuchsreihen durchgeführt (siehe Abbildung 57). In jedem Versuch wurden die folgende Parameter gemessen:

Startposition
Die Startposition musste für jeden Testlauf gesondert berücksichtigt werden, war dieser doch das Qualitätskriterium, ob der Algorithmus diesen Punkt lokalisieren konnte oder nicht.

Zurückgelegte Weg
Der zurückgelegte Weg wurde gemessen anhand der abgefahrenen Teilziele in der aufgenommenen Karte.

Dauer der globalen Selbstlokalisation
Die Dauer des Vorgangs vom Start bis zur Ausgabe des Endergebnisses wurde gemessen.

Abbildung 57: Einsatzumgebungen des EFM-Verfahrens



Startpositionen mit den jeweiligen Modellkarten der Einsatzumgebungen. Die Informatik-Umgebung ist links dargestellt, die Mathematik-Umgebung rechts.

Anzahl der Iterationsschritte
Die Anzahl der Iterationsschritte und damit die Anzahl der ausgewählten Sensorpunkte und der Anzahl durchgeführter Matchings wurde gezählt.

Anzahl und Güte der Hypothesen
Die Anzahl aller bewerteten Hypothesen sowie die Wahrscheinlichkeit der am besten bewerteten Hypothese wurde gemessen.

Als Testverfahren 1 diente die Hypothesenelimination mittels Exploration und Feature-Matching. Angeschlossene Verfahren waren eine statische iterative Kartierung, ein Navigationsalgorithmus unter Verwendung probabilistischer Roadmaps und eine iterative feature-basierte Lokalisation. Als Reduktionsheuristik diente im ersten Test eine Auswahl von 70 Features mittels Histogrammwahl ohne Segmentierung der Modellfeatures und eine zweifache zufällige Auswahl von 10 Features bei den Bildfeatures. Diese Auswahl diente der schnellen Ausführung des Verfahrens. Als Testverfahren 2 wurde nur die iterative Lokalisation modifiziert um eine Histogrammauswahl mit 25 Features, Segmentierung der Modellfeatures und eine zweifache zufällige Auswahl der Bildfeatures.

6.6.3 EFM im Informatikgebäude

Die erste Versuchsreihe (siehe Tabelle 4) in dieser Testumgebung konnte sich bis auf einen Versuch korrekt lokalisieren. Dabei wurden jeweils sehr oft nur wenige Iterationsschritte benötigt. Diese Umgebung zeichnet sich also nicht durch große Mehrdeutigkeiten aus. Im Schnitt benötigte man eine Fahrt von 3.94 Metern um mit etwa 3 Iterations- und Anfrageschritten in 0.84 Minuten die korrekte Roboterposition zu finden. Dabei wurde als Vergleich die tatsächliche Position vom Versuchsleiter mit der berechneten Position verglichen. Im nicht erfolgreichen Fall war das Kartierungsergebnis aufgrund neuer dynamischer Hindernisse fehlerhaft, sodass die richtige Position nicht errechnet werden konnte.

Tabelle 4: Ergebnisse 1. Versuchsreihe in Testumgebung 1

Startposition	erfolgreiche Lokalisation	Wahrschein-lichkeit	mittlerer Fehler	Hypoth.-anzahl	Weg [m]	Schritte	Dauer [min]
1	ja	1	0,34	1	0,95	2	0,27
2	ja	1	0,02	1	0,55	2	0,24
3	ja	1	0,03	1	0,65	2	0,22
4	ja	0,87	0,05	6	3,13	3	0,58
5	ja	1	0,1	1	0,76	2	0,37
6	ja	1	0,03	1	0,6	2	0,25
7	ja	0,69	0,2	2	0,91	2	0,3
8	ja	0,75	0,29	4	4,17	3	0,66
9	ja	1	0,46	1	3,91	3	0,73
10	nein	0,72	0,42	3	0,61	2	0,36
11	ja	0,73	0,21	11	6,15	4	2
12	ja	0,75	0,21	8	24,94	8	4,11
Durchschnitt	-	0,87	0,2	3,33	3,94	2,92	0,84

In der zweiten Versuchsreihe (siehe Tabelle 5) wurde die Modellkarte nach Kapitel 4 segmentiert. Durch diese wurde eine erhöhte Anzahl an Lokalisationsanfragen pro Iterationsschritt produziert, was sich in den Ergebnissen widerspiegelt. Die durchschnittliche Lokalisationszeit hat sich fast verdoppelt, obwohl dabei die Anzahl der Iterationen nur um 1 gestiegen ist. Ebenso hat sich der Verifikationsweg verlängert. In allen Versuchen gab es mehr Hypothesen zu bewerten und alle Versuche konnten erfolgreich abgeschlossen werden.

Betrachtet man abschließend die Ergebnisse in dieser Testumgebung, kann man dem EFM-Verfahren bescheinigen, taugliche Ergebnisse zu liefern. Allerdings macht diese Umgebung es dem Verfahren sehr einfach. Kritikpunkt am Verfahren ist lediglich die Schnelligkeit, mit der die Lokalisationsanfragen durchgeführt wurden. Weitere Tests zur Vergleichbarkeit mit allen in Kapitel 4 vorgestellten Verfahren wären wünschenswert, sind aber nicht mehr berücksichtigt worden.

Tabelle 5: Ergebnisse 2. Versuchsreihe in Testumgebung 1

Startposition	erfolgreiche Lokalisation	Wahrschein-lichkeit	mittlerer Fehler	Hypoth.-anzahl	Weg [m]	Schritte	Dauer [min]
1	ja	0,67	0,02	5	0,39	2	0,4
2	ja	0,91	0,18	4	0,68	2	0,39
3	ja	0,8	0,02	3	0,71	2	0,3
4	ja	0,73	0,09	6	4,3	3	0,84
5	ja	0,92	0,12	4	0,73	2	0,5
6	ja	0,69	0,22	5	5,32	4	1,39
7	ja	0,72	0,08	6	1,33	3	0,71
8	ja	0,73	0,24	5	0,77	2	0,66
9	ja	0,75	0,36	8	22,49	7	3,09
10	ja	0,67	0,26	5	20,19	9	4,42
11	ja	0,83	0,24	7	8,78	5	2,81
12	ja	0,77	0,28	6	8,61	4	1,61
Durchschnitt	-	0,77	0,18	5,33	6,19	3,75	1,43

6.6.4 EFM im Mathematikgebäude

Die Ergebnisse der zweiten Testumgebung fallen etwas anders aus als die der ersten Testumgebung. In der ersten Versuchsreihe (Tabelle 6) wurde durchschnittlich ein Weg von 16.24 Metern zurückgelegt, um die tatsächliche Roboterposition zu bestimmen. Dabei wurden 5.42 Hypothesen erzeugt mit 6.83 Iterationsschritten in einer Zeit von 2.76 Minuten. Diese Umgebung ist ein „richtiger" Testfall für das Verfahren. Durch den zyklischen Charakter tauchen häufig Mehrdeutigkeiten auf. In allen Testfällen konnte trotzdem die richtige Position bestimmt werden.

Tabelle 6: Ergebnisse 1. Versuchsreihe in Testumgebung 2

Startposition	erfolgreiche Lokalisation	Wahrschein-lichkeit	mittler Fehler	Hypoth.-anzahl	Weg [m]	Schritte	Dauer [min]
1	ja	0,73	0,12	5	37,06	11	5,9
2	ja	0,87	0,03	3	2,06	2	0,47
3	ja	0,88	0,1	2	0,69	2	0,37
4	ja	1	0,12	1	0,73	2	0,38
5	ja	0,68	0,13	7	28,26	11	4,17
6	ja	0,65	0,12	8	32,26	13	4,96
7	ja	0,67	0,06	9	2,47	3	0,59
8	ja	0,65	0,17	4	34,89	12	5,8
9	ja	0,72	0,16	4	43,35	14	6,99
10	ja	0,67	0,04	10	5,45	6	1,49
11	ja	0,88	0,15	4	5,02	3	1,16
12	ja	0,72	0,06	8	2,59	3	0,82
Durchschnitt	-	0,75	0,11	5,42	16,24	6,83	2,76

Im Gegensatz zur zweiten Versuchsreihe (siehe Tabelle 7) der ersten Umgebung bringt

hier eine Segmentierung der Umgebung in Teilkarten einen Performancegewinn. Bei etwa gleicher durchschnittlicher Wegstrecke und gleicher Anzahl von Iterationsschritten generiert diese Variante des Algorithmus weniger Hypothesen und verbraucht weniger Zeit. In allen Testfällen konnte dabei die richtige Position bestimmt werden. Abbildung 58 zeigt exemplarisch einen zusätzlichen Testlauf in dieser Umgebung.

Tabelle 7: Ergebnisse 2. Versuchsreihe in Testumgebung 2

Startposition	erfolgreiche Lokalisation	Wahrschein-lichkeit	mittlerer Fehler	Hypoth.-anzahl	Weg [m]	Schritte	Dauer [min]
1	ja	0,67	0,11	3	38,64	9	5,37
2	ja	0,7	0,05	3	1,04	2	0,48
3	ja	0,74	0,17	3	4,66	4	1,4
4	ja	1	0,16	1	0,71	2	0,46
5	ja	0,87	0,05	2	19,4	7	2,4
6	ja	0,94	0,07	2	17,6	6	2,3
7	ja	0,68	0,25	4	14,53	6	3,43
8	ja	0,75	0,16	3	13,4	7	2,48
9	ja	0,67	0,11	3	46,04	13	7,2
10	ja	0,76	0,06	4	18,4	8	3,1
11	ja	0,82	0,06	3	5,2	4	1,1
12	ja	0,78	0,08	3	5,7	4	1,3
Durchschnitt	-	0,77	0,11	2,83	15,44	6	2,59

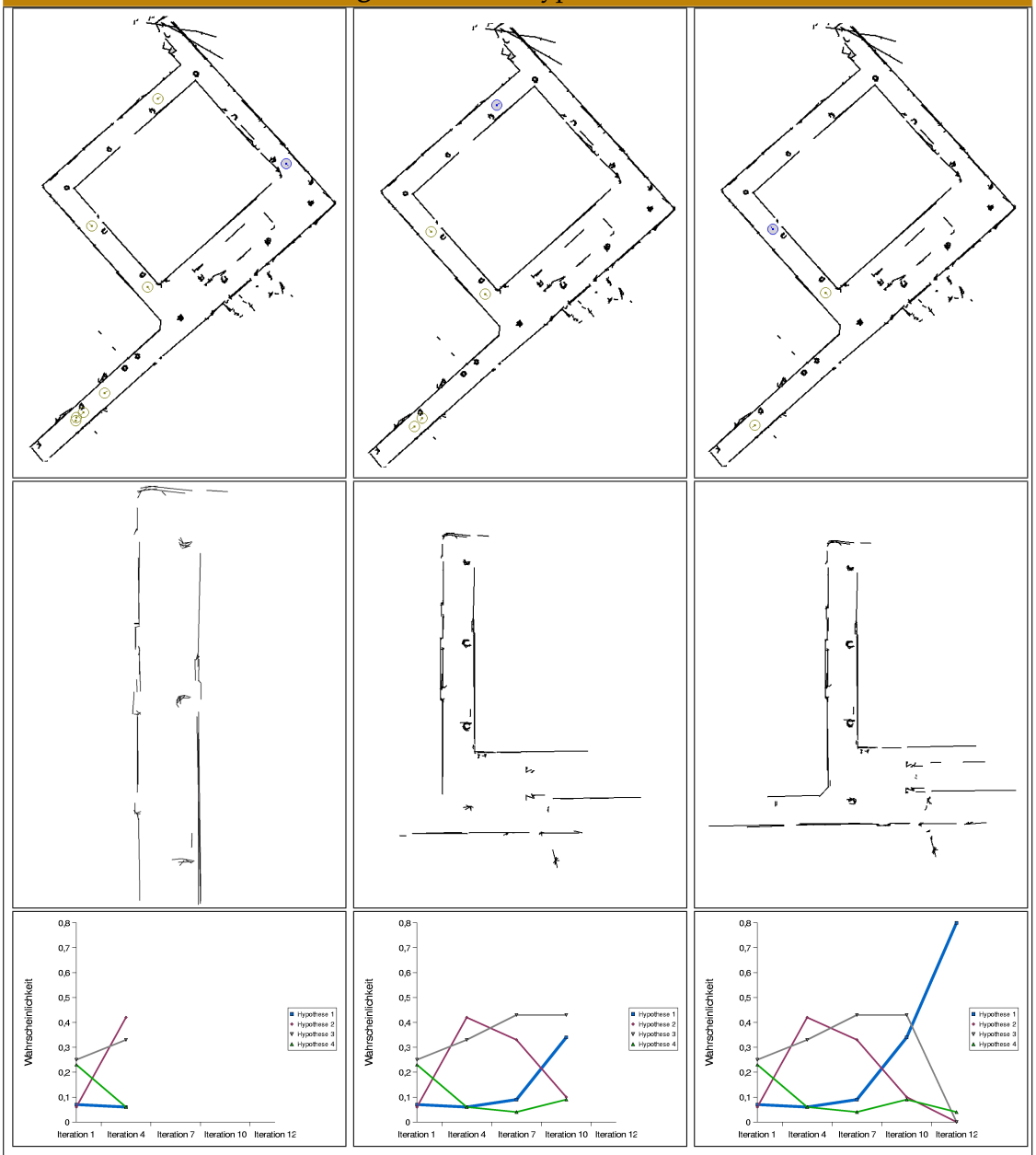
Betrachtet man sich abschließend die Ergebnisse in dieser Testumgebung, kann man dem EFM-Verfahren auch in dieser Umgebung bescheinigen, taugliche Ergebnisse zu liefern.

6.7 Resümee

In diesem Kapitel wurde das Hypotheseneliminationsverfahren in zwei Ausprägungen vorgestellt. Das ursprüngliche, theoretische Problem der Hypothesenelimination wurde rekapituliert und dessen Schwächen bei der Umsetzung in die Praxis aufgezeigt. Weiter wurde eine dynamisches Problemstellung formuliert und die bisherigen Lösungsansätze der Literatur vorgestellt.

Im weiteren Verlauf des Kapitels wurde das Problem der Exploration aufgegriffen und ein realistischer Ansatz präsentiert, der in der Praxis evaluiert wurde. Die autonome Exploration liefert die Grundlage für das EFM-Verfahren. Vergrößerung des Sichtbarkeitsbereichs des Roboters und Matching des Sichtbarkeitsbereichs gegen die vorhandene Umgebungskarte sind dabei die Kernideen des Verfahrens. Dabei kommen alle in dieser Arbeit vorgestellten Algorithmen zum Einsatz und bestätigen damit die Komplexität der Problematik. Im realistischen Einsatz konnte das Verfahren seine

Abbildung 58: Testlauf Hypothesenelimination



4-te, 10-te und 12-te Iteration eines Testlaufs. Auf den oberen Karten erkennt man hypothetische Konfigurationen des Roboters. Die Kreise in der globalen Karte repräsentieren jeweils die initiale Position der lokalen Karte des Roboters. Der farbig-markierte Kreis markiert die aktuelle beste Hypothese. Die mittleren Bilder zeigen die gerade aktuelle lokale Karte des Roboters. Auf den unteren Bildern sind die Hypothesen des Testlaufs mit ihrer Bewertung h_w dargestellt. Man erkennt, dass der tatsächliche Standort des Roboters erst sehr spät eine gute Bewertung > 0.65 erhält und der Algorithmus sodann terminiert.

Brauchbarkeit in zwei Testumgebungen unter Beweis stellen.

Das Verfahren steht und fällt allerdings mit dem Erfolg der Exploration und der feature-basierten Lokalisation. Liefert das Kartierungsverfahren keine vernünftige Karte, können keine Ergebnisse produziert werden. Heuristiken, um solche Fehlerzustände abzufangen, sind wünschenswert um die Stabilität des Systems zu erhöhen.

In diesem Kapitel haben sich eine Reihe von offenen Fragen ergeben. So liegen die theoretischen Strategien zur verbesserten Steuerung des Roboters noch nahezu ungenutzt brach und warten auf einen Einsatz in realistischen Explorationsstrategien. Zusätzliche Bewertungen der Sensorpunkte aufgrund der generierten Hypothesen wurden in dieser Arbeit nicht behandelt, können aber erfolgversprechend sein. Hauptproblem beim Einsatz alternativer Verfahren ist dabei nicht die Implementierung der Strategien, sondern die Nachvollziehbarkeit und Vergleichbarkeit. So kann ein Versuch in der Praxis insbesondere bei vorhandenen dynamischen Hindernissen schwerlich für eine andere Strategie nachvollzogen werden. Die beste Möglichkeit, aussagekräftige Tests zu bekommen liegt in der Generierung von sehr vielen Testversuchen.

Die Generierung von zusätzlichen Tests wird auch weitere Fehlersituationen ans Tageslicht bringen, die in den vorgestellten Versuchen nicht auftraten, oder nur geringfügig waren ohne Auswirkungen zu zeigen. Insbesondere ist dabei die Einbeziehung weiterer, dynamischerer Umgebungen wünschenswert.

Weitere nicht mehr realisierte Idee ist ein Vergleich des beschriebenen Verfahrens mit der Aktiven-Markov-Lokalisation oder dem Verfahren von Jensfeld.

Teil III

Entwurf von Service-Robotern in intelligenten Umgebungen

Übersicht

Der dritte Teil der Arbeit beschäftigt sich mit der Integration der vorgestellten Algorithmen, welche zur globalen Selbstlokalisation beitragen, in ein allgemeines Schema für einen Service-Roboter. Dazu wird folgende Vorgehensweise betrachtet.

Mobile Agenten in intelligenten Umgebungen
Zunächst wird beschrieben, wie eine Einsatzumgebung zukünftig auszusehen hat, nämlich eine durch ein Netz von Sensoren und externes Wissen geprägte intelligente Umgebung. In dieser tummeln sich Anwender und intelligente Helfer. Solche Helfer besitzen Anwenderschnittstellen und können auch in Form von Service-Robotern mit den Anwendern physikalisch interagieren.

Modellierung eines Service-Roboters
Dieses Kapitel beschreibt den Zusammenhang zwischen einem mobilen Roboter und seiner Serviceaufgabe, identifiziert grundlegende Aufgaben eines mobilen Systems und klassifiziert demzufolge die in den letzten beiden Teilen der Arbeit behandelten Algorithmen.

Entwurf eines Service-Roboters in intelligenten Umgebungen
Dieses Kapitel beschreibt die Architektur eines allgemeinen Service-Roboters interagierend mit einer intelligenten Umgebung und die verzahnte Anwendung der verschiedenen Algorithmen. An einem Service-Roboter-Beispiel wird exemplarisch ein solcher Einsatz beschrieben.

7 Mobile Agenten in intelligenten Umgebungen

Ein Trend wird sich in der Zukunft verstärken, welcher unter dem Schlagwort »Ubiquitous Computing« in der Literatur genannt wird. Der Begriff kann etwa als allgegenwärtiger Computer übersetzt werden. Der Begriff wurde von Mark Weiser geprägt [233] und umschreibt die Idee, den Computer immer kleiner, leichter und einfacher benutzbar zu machen, so dass dieser in den Hintergrund tritt und uns in der alltäglichen Arbeit überall unterstützen kann.

In diesem Kapitel soll ein kurzer Einstieg in die Problematiken des »Ubiquitous Computing« gegeben werden. Anschließend soll aus diesem abstrakt eine intelligente Umgebung mit intelligenten Objekten darin modelliert werden. Die intelligente Umgebung interagiert mit dem Benutzer durch kontext-basierte Mensch-Maschine-Schnittstellen [82, 112, 201] in Gestalt von intelligenten Objekten. Mit dieser Modellierung wird ein Framework geschaffen, welches der Schlüssel für das Design von Service-Robotern ist.

7.1 Ubiquitous Computing

Die grundlegenden Idee des »Ubiquitous Computing« - Ansatzes ist die Feststellung, dass gute Technologien unsichtbar sind, etwa Elektrizität, Wasserversorgung, Abwasserversorgung, Radio oder Fernsehen. Die Nutzung des Computers soll in die reale Welt integriert sein und den Benutzer in allen Lebenslagen unterstützen.

Die aktive Forschung in dieser Richtung beginnt mit dem Projekt Pandora 1988 [233], welches ein intelligentes Telefonnetz realisierte, dass Telefonanrufe zu dem entsprechenden Benutzer innerhalb des Gebäudes leitete. Das Personal Server Projekt von Intel [232] beschäftigt sich damit, die lokale vorhandene Computer-Infrastruktur zu verwenden, um von überall mit Hilfe eines kleinen, tragbaren sogenannten Personal Server Zugriff zu seinen Daten zu besitzen. Die Kommunikation soll dabei über Infrarot-Sensoren in einer modifizierten Umgebung erfolgen. Diese Forschungsrichtung, der Entwicklung einer intelligente Umgebung, wird oft auch unter dem Stichwort »Intelligentes Haus« geführt [138, 139, 229].

Ein weiterer Trend sind intelligente Geräte, etwa ein intelligenter Stift [231], intelligente Kaffeetassen [19, 77, 175], die mit Hilfe von kleinen Sensoren in einer intel-

ligenten Umgebung den Benutzer unterstützen. Mobile tragbare Systeme, etwa ein Headset [77, 181], sollen dem Benutzer Werkzeuge zur Informationsbeschaffung zur Verfügung stellen und sind unter dem Schlagwort »Embodied Virtuality« bekannt. Sie verknüpfen die Forschungsrichtung der »Virtual Reality« als auch der Robotik mit »Ubiquitous Computing«.

Das Projekt Ambiente des Frauenhofer-Instituts für Integrierte Publikations- und Informationssysteme beschäftigt sich mit der Entwicklung einer intelligenten Kommunikations- und Kooperationsinfrastruktur. Dazu gehört die sogenannte Roomware [200, 201, 205], mit Sensorik ausgestattete Kommunikationsmittel, sowie interaktive Projektionsflächen.

Die Probleme, die bei allen Ansätzen auftreten, sind nachfolgend zusammengefasst, wobei kein Anspruch auf Vollständigkeit erhoben wird. Interessant ist dabei, dass einige Probleme stark mit dem Entwurf von Service-Robotern (siehe Kapitel 8.2) verwandt sind.

Limitierte Ressourcen
: Mobile Geräte dürfen nur wenig Strom verbrauchen. Dieses wirkt sich insbesondere auch auf die Sensorik aus, die deshalb auf präzise, teure Sensorik verzichten muss. Es gibt außerdem erste Ansätze für die Stromgewinnung aus der Bewegung des Benutzers.

Sensorfusion
: Viele Sensoren in der intelligenten Umgebung sollen das Leben erleichtern. Je mehr Sensoren, desto schwieriger die Auswertung, da mehr Daten zu fusionieren sind [21].

Navigation, Positionsbestimmung
: Tracking von Benutzern durch Sensoren innerhalb eines Gebäudes, Identifikation damit diese durch mobile Helfer geleitet werden können. Objekte müssen zu jeder Zeit ihre Positionen kennen [54, 59].

Multimodalität der Ausgabe
: Verschiedene Möglichkeiten stehen zur Ausgabe bereit und müssen das gesammelte Hintergrundwissen im Kontext des Ausgabesystems präsentieren können [110].

Verteilte Systeme
: Eine Vielzahl von eigenständigen Agenten kommuniziert und interagiert (in einer intelligenten Umgebung) miteinander [139].

Intelligenz, Semantik von Sensordaten
: Planen von Aktionen, Data-Mining aus verschiedenen Wissensquellen sind große Probleme beim Einsatz.

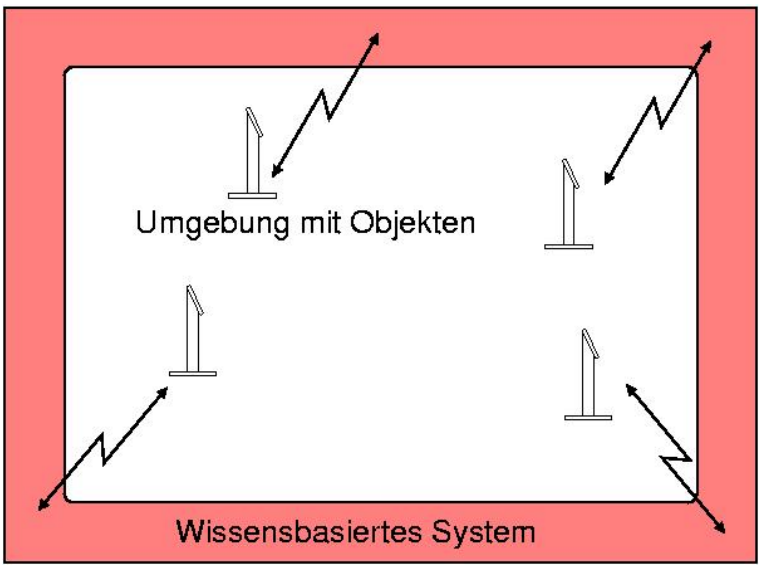
Sicherheitsaspekte
Anonymität vs. Individualität. Jeder soll fast überall an Informationen für Jedermann kommen können, aber auf der anderen Seite sollen private Informationen nur dem jeweiligen Anwender zur Verfügung stehen.

Intrabody-Kommunikation
Stromversorgung, Reichweite und Verkehrsaufkommen der erzeugten Daten [238] sind Teilprobleme für „hautnahe" Sensorik.

7.2 Modellierung einer intelligenten Umgebung

In diesem Abschnitt soll eine intelligente Umgebung [139, 147, 200] modelliert werden. Diese soll mit Hilfe von Informationen, die über Sensoren zur Verfügung gestellt werden, neue Informationen für den Benutzer in der Umgebung bereitstellen. Eine solche Umgebung wird aktive Umgebung genannt (Abbildung 59).

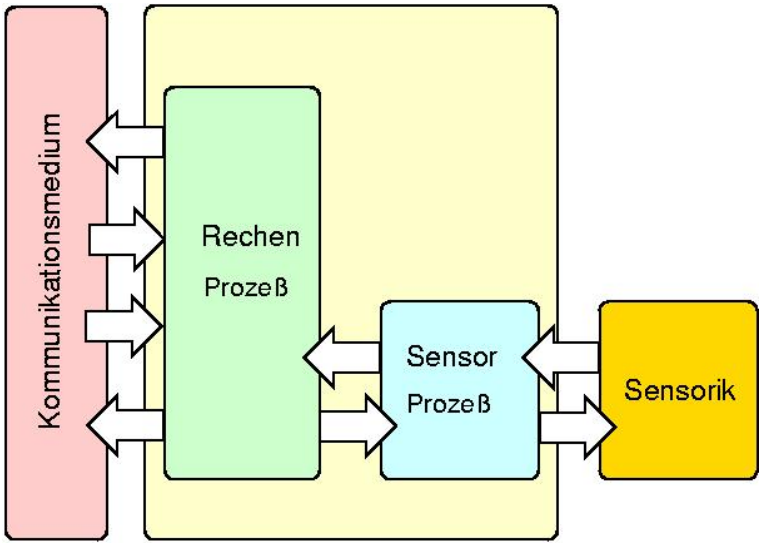
Abbildung 59: Aktive Umgebung



In einer aktiven Umgebung befinden sich nur Objekte, welche die Umgebung wahrnehmen können. Eine Interaktion findet in Form eines Informationsaustauschs mit dem Benutzer der Umgebung statt.

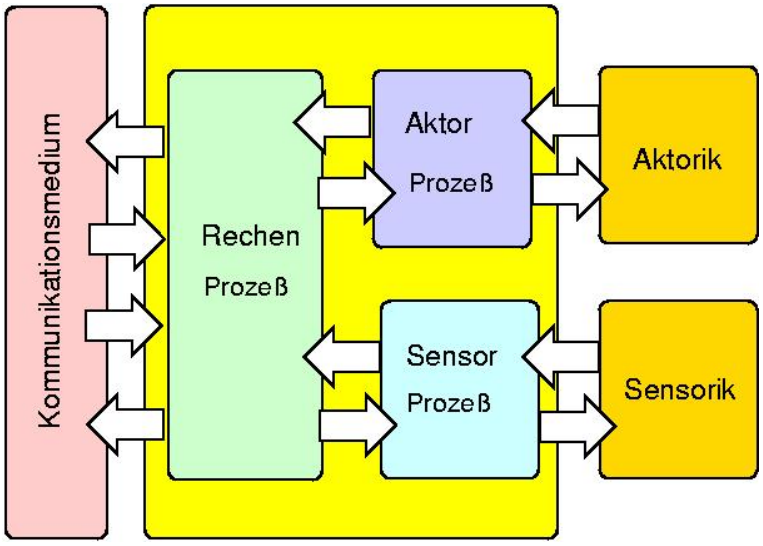
Definition 7.1 (Aktive Umgebung) *Eine aktive Umgebung ist ein physikalischer Raum, welcher mit den Benutzern der Umgebung durch Objekte interagieren kann, insbesondere kann er auf die Aktivitäten der Benutzer mit Hilfe eines Wissensbasierten Systems reagieren.*

Abbildung 60: Aktives Objekt



Ein aktives Objekt besitzt lediglich Sensoren um Umgebungsdaten aufzunehmen. Das Objekt ist nicht in der Lage, den Umgebungszustand zu verändern.

Abbildung 61: Intelligentes Objekt



Ein intelligentes Objekt kann die Umgebung wahrnehmen und diese physisch beeinflussen.

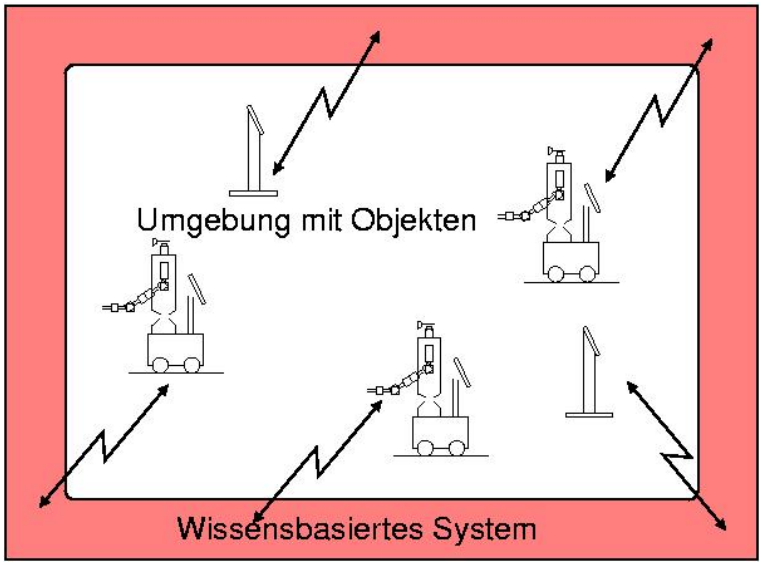
Ein Objekt kann dabei folgendermaßen definiert werden.

Definition 7.2 (Aktives Objekt) *Ein aktives Objekt (Abbildung 60) besteht aus Sensoren, einer unabhängigen Stromversorgung und einer Anbindung an ein Kommunikationsmedium mit Hilfe eines Sensor- und eines Rechenprozesses.*

Nun spiegelt dies die abstrahierte Sichtweise des »Ubiquitous Computing« wieder. Hier soll diese erweitert werden, indem auch Objekte in der Umgebung zugelassen werden, welche von der Umgebung selber kontrolliert werden. Solche Objekte sollen intelligente Objekte genannt werden (Abbildung 61).

Definition 7.3 (Intelligentes Objekt) *Ein intelligentes Objekt besitzt zusätzliche Aktoren und zugehörige Aktorprozesse, um die Umgebung zu verändern oder sich in der Umgebung zu bewegen und mit dem Benutzer zu kommunizieren. Allerdings ist ein intelligentes Objekt an die Umgebung gebunden.*

Abbildung 62: Intelligente Umgebung



In einer intelligenten Umgebung können sowohl aktive als auch intelligente Objekte vorhanden sein, die mit dem wissensbasierten System kommunizieren.

Demzufolge hat sich nun der Umgebungsbegriff erweitert, von einer aktiven zu einer intelligenten Umgebung (siehe Abbildung 62), d.h. aus einem Raum mit aktiven Sensoren wird ein Raum mit aktiven durch die Umgebung gesteuerten Helfern.

Definition 7.4 (Intelligente Umgebung) *Eine intelligente Umgebung ist ein physikalischer Raum, welcher eine Menge von aktiven und intelligenten Objekten über ein*

Kommunikationsmedium verwaltet und steuert, und ein wissensbasiertes System, um den Objekten notwendige Informationen zur Verfügung zu stellen. Insbesondere können sich die Objekte selbstständig in der Umgebung bewegen.

Die intelligente Umgebung benötigt ein wissensbasiertes System für die interagierenden Objekte. Eine allgemeine Definition für ein wissensbasiertes System kann folgendermaßen lauten:

Definition 7.5 (Wissensbasiertes System (allgemein)) *Ein Wissensbasiertes System ist ein Programm, das in einer Wissensbasis Informationen zu einer bestimmten Domäne speichert und aufgrund dieser Informationen in begrenztem Rahmen Probleme löst und Entscheidungen fällt.*

Die Hauptaufgaben Wissensbasierter Systeme liegen im Erkennen von Beziehungen, dem Ziehen von Schlussfolgerungen und dem Steuern der Anwendung von Wissen. Ein Wissensbasiertes System benötigt hierzu Techniken der Wissensrepräsentation, der Wissensverarbeitung und des Wissenserwerbs. Das Wissen wird formalisiert in einer Datenbank abgelegt, etwa in objektorientierte oder deduktive Datenbanken. Typisches Beispiel Wissensbasierter Systeme sind Expertensysteme [202].

Hier soll jedoch die folgende, etwas speziellere Definition verwendet werden (siehe Abbildung 63):

Definition 7.6 (Wissensbasiertes System) *Ein wissensbasiertes System besteht aus einer Menge von Informationsräumen sowie einer Menge von Wissensbasen, welche den Informationsräumen zur Problemlösung zur Verfügung steht.*

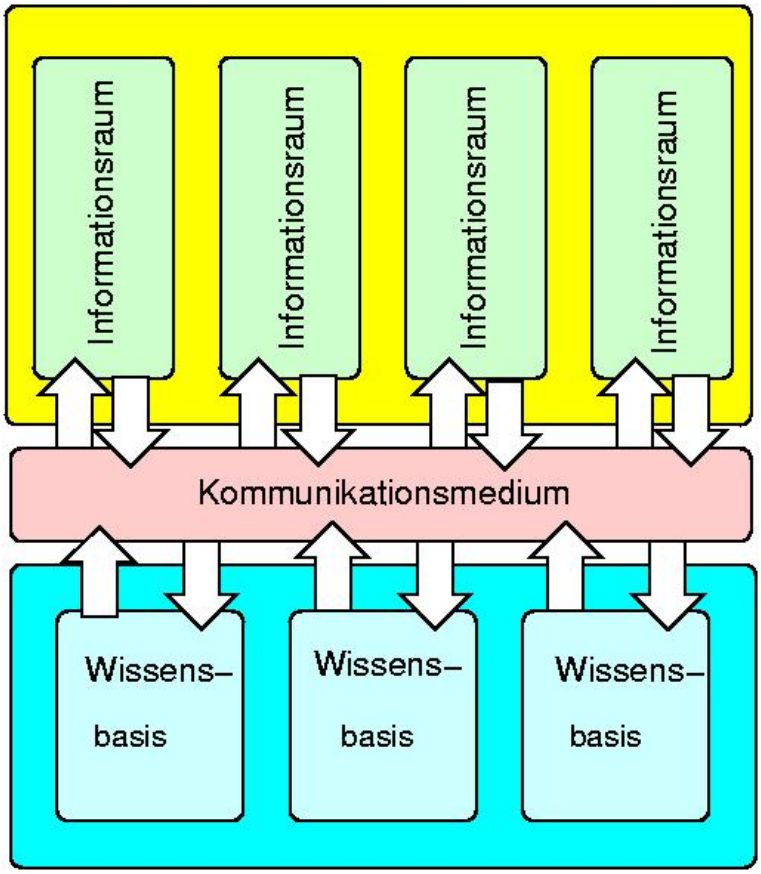
Dabei muss der Begriff Informationsraum [200] (siehe Abbildung 64) noch spezifiziert werden:

Definition 7.7 (Informationsraum) *Ein Informationsraum stellt für eine Menge intelligenter Objekte Sensorauswertungsprozesse, Rechenprozesse sowie Aktorsteuerungsprozesse durch einen Objekt-Server zur Verfügung.*

Somit liefert also eine intelligente Umgebung in Form eines wissensbasierten Systems für jedes intelligente Objekt die Möglichkeit, mit dem zur Verfügung stehenden Wissen und durch eigenes, mittels Sensoren akquiriertes Wissen (Sensorprozesse) einen internen Zustand zu bilden (Rechenprozesse) und daraus eine Handlung für die Aktoren des Objekts (Aktorprozesse) auszubilden. Für eine intelligente Umgebung kann man folgende Anforderungen festlegen, welche für eine Service-Anwendung zu beachten sind.

Intelligenz der Objekte
: Jedes intelligente Objekt der Einsatzumgebung wird durch einen endlichen Automaten repräsentiert. Einzelne Zustände des endlichen Automaten werden mit den zur Verfügung stehenden Tätigkeiten des intelligenten Objektes verknüpft.

Abbildung 63: Wissensbasiertes System



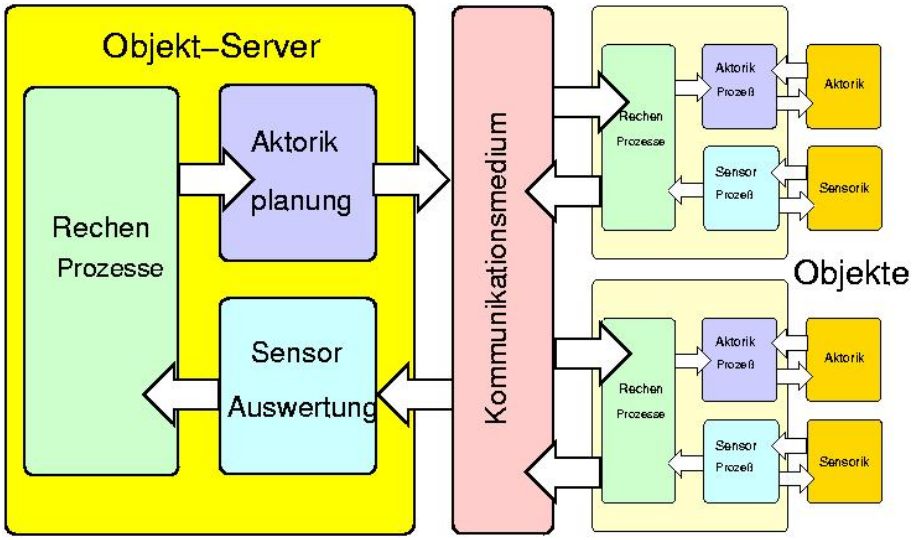
Jeder Informationsraum kann mit jedem anderen Informationsraum und auch mit den verschiedenen Wissensbasen kommunizieren.

Intelligenz der Informationsräume
Jeder Informationsraum muss zusätzlich eigene Sensor-, Aktor- und Rechenprozesse zur Verfügung stellen, um die Aktivitäten der Umgebung intelligent zu gestalten.

Integration intelligenter Objekte
Eine Erweiterung um intelligente Objekte, insbesondere auch die Möglichkeit spontan neue Objekte anderer Benutzer zu integrieren, sollte möglich sein. Dabei tauchen Sicherheitsprobleme auf, die hier nicht weiter beachtet werden.

Kontextwissen der Einsatzumgebung
Das Wissensbasierte System muss die Möglichkeit bieten, Wissen aus unterschiedlichsten Wissensquellen als Wissensbasis zur Verfügung zu stellen.

Abbildung 64: Informationsraum



Im Informationsraum steht ein Objekt-Server zur Verfügung, der die intelligenten Objekte im Informationsraum steuert, deren Daten auswertet. Die intelligenten Objekte haben trotzdem die Möglichkeit, auch ohne einen Objekt-Server miteinander zu kommunizieren.

7.3 Modellierung einer Mensch-Maschine-Schnittstelle

Eine Mensch-Maschine-Schnittstelle (Human Machine Interface, HMI) taucht in der Definition einer intelligenten Umgebung nicht auf, versteckt sie sich doch ebenfalls in einem intelligenten Objekt. Betrachtet man zum Beispiel ein Touch-Panel-PC als aktives Objekt in einer Umgebung, so besitzt dieses Panel Sensoren, welche auf Display-Druck reagieren.

Mensch-Maschine-Schnittstellen sind fundamental für den Erfolg einer intelligenten Umgebung. Beispiele für solche Schnittstellen gibt es in der Literatur reichlich. Angefangen bei der Gestaltung von Benutzeroberflächen, der Definition von Entwurfskriterien [39, 78, 112, 154, 181], der Darstellung von Informationen in unterschiedlichen Medien [110,111] oder der Auswertung von Gesten [54]. Eine Definition für eine solche Schnittstelle kann folgendermaßen lauten:

Definition 7.8 (Mensch-Maschine-Schnittstelle) *Eine Mensch-Maschine-Schnittstelle gibt dem Menschen die Möglichkeit angemessen mit der Maschine zu interagieren und der Maschine die Verpflichtung Informationen angemessen mit den Möglichkeiten der Maschine dem Menschen mitzuteilen.*

Eine solche Schnittstelle ist in dieser Architektur vom aktiven Objekt abhängig und ge-

sondert anzufertigen. Man kann folgende, allgemein gültigen Anforderungen an HMIs formulieren.

Sicherheit
Die Schnittstelle soll die sichere Bedienbarkeit des Systems sicherstellen.

Bedienbarkeit
Die Schnittstelle soll allgemeinen Richtlinien zur Bedienbarkeit entsprechen und dem Begriff Benutzerfreundlichkeit nahe kommen.

7.4 Modellierung intelligenter Objekte

Die Definition eines intelligenten Objektes wurde schon oben geliefert (siehe Abbildung 61). Man kann zwischen Objekten, die den Anwender unterstützen sollen und Objekten, die aktiv die Umwelt beeinflussen, unterscheiden.

Heutzutage besteht eine intelligente Umgebung aus einem Gebäude, in welchem ein Funknetz etabliert ist. In diesem Funknetz tummeln sich Anwender mit Hilfe ihrer Laptops und haben anonym Zugang zum Internet. Weiter gibt es agierende Forschungsroboter, die über das Funknetz mit Hilfe von Benutzerschnittstellen das mobile Gerät steuern.

Zukünftig sollen sich in einer intelligente Fabrik mobile Helfer in den Produktionsablauf einschalten, wo sie benötigt werden [82]. Im Haushalt soll uns ein intelligenter Haushaltsassistent unnötige Arbeiten abnehmen [81]. Ferner soll die Hausautomation weiter voranschreiten und die Haushaltsgeräte intelligent ihnen zugedachte Aufgaben managen [138, 139].

Aus heutiger Sicht kann man folgende Anforderungen für intelligente Objekte definieren:

Kommunikation
Das intelligente Objekt sollte in der Lage sein, angemessen mit der intelligenten Umgebung kommunizieren zu können. Das bedeutet auch, dass nur Daten gesendet und empfangen werden, die unbedingt notwendig sind.

Stromversorgung
Das intelligente Objekt sollte eine angemessene Stromversorgung besitzen, welche durch einfache Verrichtungen aufgeladen werden kann oder sich selber auflädt.

Tätigkeit
Jedes intelligente Objekt ist für eine bestimmte Tätigkeit in der Einsatzumgebung konzipiert. Diese Tätigkeit sollte für die Einsatzumgebung nützlich sein und den Anwender mit Hilfe der intelligenten Umgebung unterstützen.

Sicherheit
Das System soll stabil und sicher seine Anwendung erfüllen. Ansonsten ist die Akzeptanz der Benutzer nicht gegeben.

Aktorik und Sensorik
Die Aktorik und Sensorik sollen die Tätigkeit des Objekts unterstützen und außerdem für die intelligente Umgebung nützlich sein.

7.5 Resümee

In diesem Kapitel wurde ein Framework für einen Service-Roboter geschaffen. Dazu wurden ganz abstrakt intelligente Objekte in einer intelligenten Umgebung definiert. Zusätzlich wurden die Anforderungen an solche Entitäten gestellt, wenn realistische Anwendungen entwickelt werden sollen.

Das Framework ist eine Erweiterung der Vorstellungen des »Ubiquitous Computing«-Ansatzes. Dort wurden bisher nur kleine mobile Geräte propagiert, die in einer aktiven Umgebung eingesetzt werden können. Dieses Framework erweitert den Ansatz um die gestaltende Interaktionsmöglichkeit der Umgebung durch intelligente, darin operierende mobile Objekte. Stellt man sich Service-Roboter als solche Objekte vor, ist es möglich, einfachere mit weniger Sensorik und Rechenleistung ausgestattete Service-Roboter in intelligenten Umgebungen agieren zu lassen, da diese Rechnerkapazitäten der Umgebung und Sensorik anderer Objekten verwenden. Damit kann man für Service-Roboter realistische Einsatzszenarien entwickeln.

8 Modellierung eines Service-Roboters

Der Begriff Service-Roboter ist in der Robotik-Literatur weit verbreitet. Ein Service soll dabei von einem mobilen Roboter angeboten werden. Eine allgemeine Definition sucht man vergebens, gibt es doch keine Definition dafür. In der Literatur findet man eine Vielzahl von selbstbezeichnenden Service-Robotern:

Die Museumsroboter Rhino und Minerva agierten in Museen und hatten die Service-Aufgabe, Besuchern die Ausstellungsstücke zu erklären [215, 216]. Besucher konnten dabei sowohl Web-Besucher als auch Besucher des Museums sein. Das MOPS-System [224] hatte die Service-Aufgabe, in einem Institut Post zu verteilen. Das Morpha-Projekt hatte als Ziel, intelligente anthropomorphe Service-Roboter zu entwickeln [82]. Dazu gehört ein mobiler Haushaltsassistent als auch ein Produktionshelfer. Der Haushaltsassistent soll häusliche Tätigkeiten verrichten und so behinderte Menschen unterstützen [81]. Der Produktionshelfer soll beim Transport und bei der Verteilung der Ressourcen tätig werden [101].

Allen Systemen ist gemeinsam, dass sie eine mobile Plattform voraussetzen, welche mit Sensoren und weiteren Aktoren ausgestattet ist und eine Software zur Erledigung und Steuerung der Tätigkeit verwenden. Dabei wird bei diesen Systemen und bei allen in Entwicklung befindlichen Systemen sozusagen „das Rad ständig neu erfunden". Software wird nicht wiederverwendet, sodass kaum funktionierende Systeme anzutreffen sind, welche über den akademischen Einsatz hinausgekommen sind [135]. Zudem sind durchdachte realistische Anwendungsszenarios selten.

Im folgenden sollen die Begriffe Roboter-System und Service-Roboter formalisiert und deren Aufgaben sowie deren grundlegenden Prozesse identifiziert werden.

8.1 Modellierung eines Roboter-Systems

In den ersten beiden Teilen dieser Arbeit war immer nur von einem Roboter die Rede, welcher in einer Einsatzumgebung operierte. Dabei war höchstens die Geometrie des Roboters von Bedeutung, ansonsten war er ein abstraktes Objekt mit einer Konfiguration welches sich, von Algorithmen gesteuert, in der Einsatzumgebung bewegte und dabei Sensordaten akquirierte.

Man kann auch eine andere Betrachtungsweise für ein Roboter-System besitzen, da der Zustandsübergang, repräsentiert durch eine Funktion, ein komplexer Vorgang ist, bei welchem Sensoren ausgewertet und zusätzliche interne Repräsentationen gebildet werden müssen. Deshalb wird im folgenden ein, aus der Sicht des **Software-Entwurfs** gebräuchlicher Blickwinkel auf ein System gebraucht [197].

8.1.1 Systemanforderungen der Software

Definition 8.1 (System, Definition des Software-Entwurfs) *Ein System ist ein sinnvoller Satz von miteinander verbundenen Komponenten, die zusammenarbeiten um ein bestimmtes Ziel zu erreichen.*

Ein System kann aus Software- und Hardware-Komponenten bestehen. Diese kann man zusammenfassend auch Systemkomponenten bezeichnen. Ein System kann man in eine Hierarchie von Systemen, sogenannte Subsysteme, aufteilen. Im nachfolgenden interessiert eine wichtige Klasse von Systemen, die Echtzeit-Systeme.

Echtzeitsysteme

Definition 8.2 (Echtzeitsystem) *Ein Echtzeitsystem ist ein (Software-)System, bei dem die einwandfreie Funktionsweise des Systems abhängt von den Ergebnissen, die vom System erzeugt werden, sowie von der Zeit zu der diese Ergebnisse erzeugt werden.*

Man unterscheidet dementsprechend grundlegend zwischen weichen und harten Echtzeitsystemen. Bei einem weichen Echtzeitsystem wird dessen Funktion schlechter, wenn die Ergebnisse nicht entsprechend den spezifizierten zeitlichen Anforderungen erzeugt werden. Ein hartes Echtzeitsystem ist ein System, dessen Funktion bei Abweichungen von den zeitlichen Vorgaben inkorrekt ist.

Das Echtzeitsystem wird durch ein Stimulus-/Response-System beschrieben. Nachdem das System ein bestimmtes Eingangsereignis (Stimulus) erhalten hat, muss es eine entsprechende Reaktion (Response) erzeugen. Das Verhalten kann deshalb durch Auflistung der vom System empfangenen Stimuli, der zugeordneten Reaktionen und der Zeit definiert werden, zu der die Reaktion erfolgen muss. Der Stimulus wird von Sensoren erzeugt, die Response wird durch Aktoren vollzogen.

Ein Echtzeitsystem wird durch eine Menge von Prozessen, die nebenläufig arbeiten, modelliert. Im nachfolgenden wird dafür auch die Bezeichnung Thread verwendet. Dabei unterscheidet man nach drei Prozessarten:

Sensorverwaltungsprozess
: Für jeden Sensor existiert ein solcher Prozess. Der Prozess kann jeweils durch eine endliche Maschine mit einer Menge von Zuständen beschrieben werden.

Rechenprozesse
Berechnen die erforderliche Response aus dem Aktorzustand und den Sensordaten, welche an Aktorsteuerungsprozesse weitergegeben werden.

Aktorsteuerungsprozesse
Verwalten den Aktorbetrieb und liefern den aktuellen Aktorzustand den Rechenprozessen zur Weiterverarbeitung.

Dieses Modell ermöglicht die schnelle Datenerfassung vom Sensor und gestattet, Verarbeitung und zugeordnete Aktor-Response später auszuführen.

Ein Echtzeitsystem kann man in die folgenden Subsysteme klassifizieren:

Überwachungssysteme
Treten in Aktion, wenn ein außergewöhnlicher Messwert auftritt.

Datenerfassungssysteme
Erfassen Daten und speichern diese zwischen für langsamere, nicht in Echtzeit arbeitende Prozesse.

Steuer- und Regelungssysteme
Prüfen Sensoren und handeln abhängig von den von den Sensoren gelieferten Messwerten.

Jedes solche System besitzt dabei eine Menge von Prozessen aus den jeweiligen Prozessarten.

Verteilte Systeme

Ein komplexes System, welches aus vielen Sensoren und Aktoren besteht, wird nicht monolithisch aufgebaut, sondern modular. Dieses führt zwangsläufig zum Entwurf von verteilten Systemen.

Ein allgemeiner Ansatz für ein verteiltes System besteht darin, die Systemarchitektur als verteilte Objektarchitektur zu gestalten. Dabei werden grundlegende Systemkomponenten durch Objekte realisiert, welche eine Schnittstelle für die von ihnen angebotenen Dienste bereitstellen. Andere Objekte rufen diese Dienste ab, ohne dass ein logischer Unterschied zwischen den Benutzern und den Anbietern der Dienste besteht.

Die Objekte können auf eine Anzahl von Computern innerhalb eines Netzwerks verteilt werden und kommunizieren über eine sogenannte Middleware [192, 193, 230]. Diese agiert als Softwarebus, stellt eine Reihe von Diensten bereit und ermöglicht den Objekten die Kommunikation sowie das einfache Hinzufügen bzw. Entfernen zum System. Oft wird der Begriff Object Request Broker (ORB) für diese Schnittstelle verwendet.

Definition 8.3 (Verteiltes System) *Ein verteiltes System besteht aus einer verteilten Objektarchitektur. Die Objekte kommunizieren über eine ORB-Schnittstelle miteinander.*

Für den Einsatz solcher Architekturen hat sich der CORBA-Standard etabliert. Dieser kann in Form von frei zugänglichen Tools wie TAO für die Software-Entwicklung verwendet werden [192, 193].

Wiederverwendbare Systeme

Beim Software-Entwurf sollte man Wert auf die Wiederverwendbarkeit von Software legen [76]. Gerade für den mobilen Robotik-Bereich jedoch sind Code-Bibliotheken noch nicht verfügbar oder gerade erst im Entstehen begriffen [190, 191]. Wiederverwendbare Software-Entwicklung ist in der Regel objektorientiert und verwendet Design-Patterns [76]. Dabei verwendet man die folgenden Begrifflichkeiten:

Definition 8.4 (Interface, Schnittstelle) *Ein Interface ist eine abstrakte Klasse, welche eine wohldefinierte Schnittstelle repräsentiert.*

Definition 8.5 (Dienst, Service) *Ein Dienst oder Service repräsentiert eine durch ein Interface spezifizierte Aufgabe des Systems.*

Definition 8.6 (Komponente) *Eine Komponente ist eine Instanz eines oder mehrerer Dienste.*

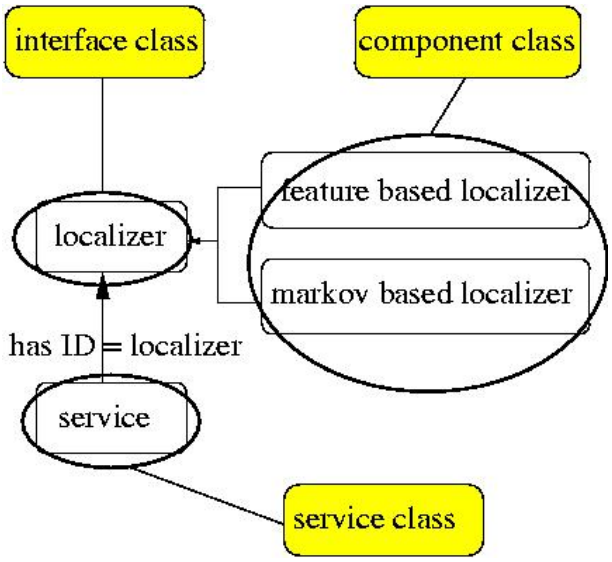
Definition 8.7 (Wiederverwendbares System) *Ein wiederverwendbares System besteht aus einer Menge von Komponenten, welche in Service-Klassen unterteilt sind. Jede Service-Klasse wird durch eine abstrakte Schnittstelle und deren Komponenten repräsentiert.*

Ein Beispiel für den Zusammenhang zwischen Dienst, Interface und Komponente ist in Abbildung 65 dargestellt. Die komponenten-basierte Software-Entwicklung hat zum Ziel abstrakte Objektklassen zu definieren, welche wiederverwendbar sind. Wenn ein System einen Dienst benötigt, ruft es eine Komponente auf, welche diesen Dienst bereitstellt. Komponenten werden durch ihre Schnittstellen jeweiligen Diensten zugeordnet. Auf diese Art ist es möglich komplexe Algorithmen durch Hierarchien von Services zu zerlegen (siehe Abbildung 66).

Kritische Systeme

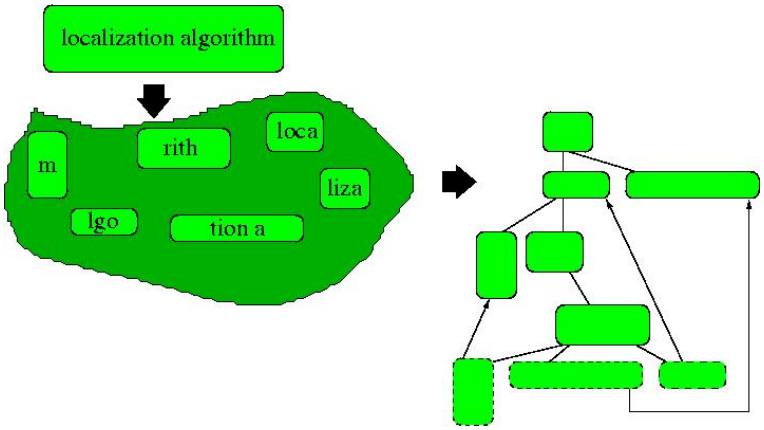
Ein wichtiger Aspekt, den große Softwarefirmen in der Vergangenheit lernen mussten, sind fehlertolerante Systeme. So wurde der Flughafen in Dallas erst nach mehreren Monaten seiner Eröffnung in Betrieb genommen, da die Steuersoftware zur Gepäckabfertigung fehlerhaft war [55, 197]. Anderes prominentes Beispiel ist der Verlust einer Marssonde der NASA aufgrund eines einfachen Umrechnungsfehlers [152]. Ein fehlertolerantes System kann man folgendermaßen definieren:

Abbildung 65: Dienste und Komponenten



Beispiel wie Interface, Service und Komponente zusammenhängen. Im Beispiel wird ein Lokalisations-Service betrachtet. Dieser besitzt das Interface localizer. Zwei verschiedene Lokalisationsalgorithmen oder Komponenten erben von der Interface-Klasse und füllen die Schnittstelle aus.

Abbildung 66: Dekomposition von Algorithmen



Beispiel wie ein Lokalisationsalgorithmus in einzelne Service-Klassen unterteilt werden kann. Diese bilden in der Ausführung eine Hierarchie von Services mit zugehörigen Komponenten als Instanzen. Mehrfach verwendete Komponenten können durch virtuelle Komponenten bereitgestellt werden.

Definition 8.8 (Fehlertolerantes System) *Ein fehlertolerantes System ist ein System, das seine Arbeit nach dem Auftreten von Systemfehlern fortsetzen kann. Die Fehlertoleranz stellt sicher, dass Systemfehler nicht zu einem Systemausfall führen.*

Fehlertolerante Systeme werden in kritischen Einsatzumgebungen eingesetzt in denen ein Systemausfall einen katastrophalen Unfall verursachen könnte, bei dem u.U. Menschen gefährdet werden. Vier Bestandteile der Fehlertoleranz von Programmen sollen den sehr hohen Anspruch an Zuverlässigkeit und Verfügbarkeit sicherstellen:

Fehlererkennung
Das System muss erkennen, dass eine spezielle Zustandskombination aufgetreten ist, die zu einem Systemausfall führen kann.

Schadensbeurteilung
Die von dem Fehler betroffenen Teile des Systemzustands müssen entdeckt werden.

Wiederherstellung nach einem Fehler
Das System muss in einen bekannten sicheren Zustand zurückkehren.

Fehlerkorrektur
Modifikation des Systems auf die Weise, dass der Fehler nicht erneut auftreten kann. Dabei ist zu beachten, dass manche Fehler nur zeitweise als Folge unglücklicher Eingaben auftreten und nicht korrigiert werden können.

Die **defensive Programmierung** ist ein wirksames Verfahren zur Implementierung fehlertoleranter Systeme. Dabei wird redundanter Code erzeugt, um den Systemzustand nach einer Änderung zu überprüfen und sicherzustellen, dass die Zustandsänderung konsistent ist. Ist das nicht der Fall, wird die Änderung rückgängig gemacht. Ein weiteres wirksames Verfahren sind fehlertolerante Architekturen. Um dabei Fehlertoleranz zu erreichen, sind folgende Konzepte sinnvoll:

- Diversitäre Programmierung
  Das Teilsystem wird in N-facher Redundanz realisiert. Die Ausgaben eines jeden Systems werden miteinander verglichen. Nach einem Vergleich mittels eines Auswahlsystems sollen immer mindestens k Aussagen verglichen werden und davon mindestens 2 identisch sein.

- Wiederherstellungsblöcke
  Jede Programmkomponente besitzt einen Test auf erfolgreiche Ausführung. Außerdem enthält sie alternativen Code, der dem System die Herstellung einer Sicherungskopie und die Wiederholung der Verarbeitung für den Fehlerfall gestattet. Wenn alle Algorithmen fehl schlagen, wird ein Fehler gemeldet.

Fehlertolerante Software muss unter Verwendung einer fehlertoleranten Steuereinheit ausgeführt werden, die sicherstellt, dass die Schritte zum Ausgleichen eines Fehlers verwendet werden.

8.1.2 Systemanforderungen der Hardware

Die Hardware wird durch ein Computersystem sowie Sensorik und Aktorik repräsentiert. Für diese kann man die folgenden Forderungen aufstellen:

Sensorausstattung

Die Sensorausstattung eines Roboter-System ist nötig, um mit der Umwelt interagieren zu können. So müssen Hindernisse, seien sie statisch oder dynamisch, erkannt werden. Demzufolge benötigt das Roboter-System eine grundlegenden Sensorausstattung.

Definition 8.9 (Grundlegende Sensorausstattung) *Das Roboter-System sollte durch eine Anzahl von Sensoren ausgestattet sein, so dass das System alle für das System kritische Situationen erfassen und darauf reagieren kann. Dann spricht man von einer grundlegenden Sensorausstattung.*

Ausfallsicherheit und Wartungsfreiheit

Analog zur Fehlervermeidung beim Software-Entwurf gibt es auch die Möglichkeit, Hardware redundant auszustatten:

Definition 8.10 (Redundante Sensorausstattung) *Das Roboter-System sollte durch den Ausfall einer begrenzten Anzahl von Sensoren mindestens die Funktionalität der grundlegenden Sensorausstattung erreichen. Dann spricht man von redundanter Sensorausstattung.*

Weiterer kritischer Punkt ist die Energieversorgung eines Roboter-Systems. Es muss gewährleistet sein, dass die Energieversorgung für eine längere Zeitspanne ausreicht und diese überwacht wird. Der Energieträger sollte einfach austauschbar sein und das komplette System bedienen. Neben der einfachen Handhabung der Energieversorgung sollte das gesamte Roboter-System nahezu wartungsfrei sein. So sollte das System einfach An- und Abzuschalten sein und keinerlei Kenntnisse über das System voraussetzen. Wartezeiten beim Herunter- oder Hochfahren sind im besten Fall nicht akzeptabel.

8.1.3 Designkriterien für ein Roboter-System

Aus den obigen Abschnitten ergibt sich nun eine Definition aus den Anforderungen für ein Roboter-System:

Definition 8.11 (Roboter-System) *Ein Roboter-System soll durch eine verteilte, wiederverwendbare, ausfallsichere Echtzeit-Software und wartungsfreie, stromsparende, redundante, grundlegende Sensorik und Aktorik repräsentiert werden.*

Diese Forderungen eines Roboter-Systems können die meisten Systeme nicht erfüllen, da die Ausfallsicherheit und Wartungsfreiheit nur unzureichend erfüllt ist, was für Industrieanwendungen oder Anwendungen in Interaktion mit Benutzern gefordert

wird. Die im Rahmen dieser Arbeit verwendeten Systeme besitzen diese Anforderungen nicht, da eine ausreichende Sensorik und Aktorik nicht gegeben ist.

8.2 Modellierung eines Service-Roboters

Die obige Anforderungs-Modellierung eines Roboter-Systems dient als Grundlage eines Service-Roboters. Ein Service-Roboter kann nun folgendermaßen definiert werden:

Definition 8.12 (Service-Roboter) *Ein Service-Roboter ist ein Roboter-System, welches zudem noch eine ausgezeichnete Service-Tätigkeit besitzt, die Menschen bei speziellen Aufgaben (in intelligenten Umgebungen) unterstützen oder spezielle Aufgaben erledigen soll. Eine Kommunikation mit dem Menschen geschieht dabei falls notwendig über Mensch-Maschine-Schnittstellen.*

Natürlich kann ein Service-Roboter auch ohne den Kontext einer intelligenten Umgebung eingesetzt werden, jedoch sind die meisten bisherigen Anwendungen nicht alltagstauglich. Putzroboter etwa werden völlig autonom konzipiert, jedoch würden intelligente Objekte oder eine intelligente Umgebung den Einsatz eines solchen Roboters erheblich vereinfachen. So muss ein Putzroboter mit einem vereinfachtes Sensorsystem möglichst intelligent seine Umgebung abfahren. Roboter-Rasenmäher können nur mit Hilfe einer abgesteckten Umrandung der zu mähenden Umgebung eingesetzt werden [63,74]. Zukünftige Produktionshelfer wie auch die schon erfolgreich eingesetzten Roboter Rhino und Minerva benötigen Kontextwissen der Einsatzumgebung.

Im Kontext einer intelligenten Umgebung sind Service-Roboter als verlängerter Arm der Umgebung denkbar, um die Nutzer der Einsatzumgebung zu unterstützen. So setzt ein Museumsroboter etwa ein Kontextwissen über Ausstellungsgegenstände voraus, ein Auskunftsroboter Kontextwissen über die arbeitenden Personen und Tätigkeiten, ein Fertigungsassistent Spezialwissen über irgendwelche Produktionsabläufe.

Ein Service-Roboter aber auch ein Roboter-System soll mit Menschen interagieren. Diese Menschen besitzen außerdem unterschiedliches Wissen. Für einen Service-Roboter kann man folgende Anforderungen formulieren:

Einfache Wartung
: Das System sollte im dauerhaften Einsatz robust sein und eine einfache Wartung zulassen.

Niedrige Kosten
: Die Serviceapplikation sollte einen erträglichen Kosten-Nutzen Faktor besitzen.

Akzeptanz der Benutzer
: Der Einsatz des Service-Roboters im täglichen Leben sollte gegeben sein. Die Einsatzumgebung sollte so gestaltet sein, dass dem Service-Roboter seine Aufgabe auch zugetraut wird.

Erfüllung der Aufgaben, Sicherheit
Der Service-Roboter soll funktionell und sicher seiner Service-Tätigkeit nachgehen.

Diese Anforderungen decken sich weitgehend mit denen eines intelligenten Objektes aus dem vorherigen Kapitel.

Analysiert man die heutige Marktsituation bestehender Service-Roboter, kommt man zu folgendem Bild: Der Siemens-Hefter Service-Roboter beispielsweise ist sehr robust im täglichen Einsatz, hat allerdings nicht unbedingt niedrige Kosten, eine Akzeptanz der Benutzer ist bisher nicht gegeben. Andere Anwendungen wie kleine Putzroboter sind noch einfacher zu warten, besitzen niedrige Kosten sind aber in den Augen der Anwender ungeeignet.

Das Beispiel Aibo zeigt, dass es auch anders geht. Dieser sogenannte Toy-Roboter ist relativ einfach aufgebaut, erfüllt nicht die Kriterien eines Roboter-Systems, besitzt auch nicht unbedingt niedrige Kosten, hat aber eine sehr hohe Akzeptanz obwohl dessen Tätigkeit scheinbar sinnlos ist. Deshalb ist er eine erfolgreiche Anwendung eines Roboter-Systems (wenn auch kein Service-Roboter).

8.3 Aufgaben und Prozesse eines Service-Roboters

In den ersten beiden Teilen dieser Arbeit wurden eine Reihe von Problemen und Problemlösungen für Roboter-Systeme vorgestellt. Dabei wurden für viele Algorithmen Eingaben vorgegeben, welche unter Umständen aus parallel-laufenden Prozessen erzeugt wurden. Da ein Roboter-System aus vielen unterschiedlichen Rechenprozessen besteht, ist es an der Zeit, diese zu unterscheiden in allgemeine Prozesse und in tatsächliche Aufgaben, die ein mobiles System erledigen soll:

Definition 8.13 (Prozess des Roboter-Systems) *Ein Prozess eines Roboter-Systems ist ein eigenständiger Thread der Softwareschicht.*

Definition 8.14 (Aufgabe des Roboter-Systems) *Eine Aufgabe des Roboter-Systems wird durch das Zusammenspiel von mehreren Prozessen des Roboter-Systems erledigt, wobei ein zusätzlicher Prozess die Ausführung der Aufgabe und Koordination der beteiligten Threads übernimmt.*

Sensorsteuerungs- und Aktorsteuerungsprozesse sind besondere Klassen von Prozessen und spielen bei der nachfolgenden Klassifizierung keine Rolle.

8.3.1 Aufgaben und Prozesse eines Roboter-Systems

Für ein mobiles Roboter-System kann man abhängig von vorherigen Kapiteln allgemein die folgenden Aufgaben spezifizieren:

Exploration
Exploration der Umgebung mit und ohne menschlichen Begleiter. Generierung einer fortlaufenden lokalen Karte mittels Bewegungs- und Sensordaten.

Globale Selbstlokalisation
Feststellen der globalen Position des Roboters mit und ohne menschlichen Begleiter.

Anfahren eines Ziels, Navigation
Steuerung zu einem Ziel mit und ohne menschlichen Begleiter.

Kommunikation, Befehlsverarbeitung
Kommunikation des Roboters mit dem Benutzer, Verarbeiten von Benutzereingaben.

Personenverfolgung oder Tracking
Begleiten von Personen durch den Roboter.

Neben den Aufgaben, modelliert als Prozesse, kann man die folgenden grundlegenden **Prozesse** identifizieren:

Kartierung
Generierung einer fortlaufenden lokalen Karte mittels Bewegungs- und Sensordaten.

Hypothesenerzeugung
Erzeugen von Lokalisationshypothesen durch Matchingverfahren der Sensordaten mit den Umweltdaten.

Bewegungsprozess
Grundlegende Bewegungsüberwachung des Systems.

Positionsverfolgung
Verfolgung der aktuellen Position des Roboters mittels Bewegungs- und Sensordaten.

Pfadplanung
Pfadplanung vom Start- zum Zielpunkt und Neuplanung bei Hindernisdetektion.

Sensordatenauswertung
Überwachen von Sensordaten, Filterung von Sensordaten.

Sensordatenverschmelzung
Verschmelzen von Sensordaten aus unterschiedlichen Sensorquellen.

Hindernisvermeidung
Erkennung und Ausweichen von Hindernissen, die den Roboter bedrohen.

Recovery
Überwachung aller Prozesse und deren Fehler und Einleitung geeigneter Gegenmaßnahmen.

Die Aufgaben besitzen dabei die Möglichkeit, mit Hilfe der Daten der unterschiedlichen Prozesse eine Aktionsplanung für den Roboter zu bewerkstelligen, weshalb kein separater Planungsprozess definiert wird.

Betrachtet man sich die Implikationen zwischen den verschiedenen Prozessen, wie in Abbildung 67 dargestellt, erkennt man die Wichtigkeit der Aufgaben-Prozesse des Roboter-Systems. Eine effiziente Lösung der globalen Selbstlokalisation würde eine Lösung der Explorations-Aufgabe und der Navigations-Aufgabe implizieren. Eine Lösung allein für die Navigationsaufgabe würde das Explorationsproblem nicht lösen und eine effiziente Exploration löst nicht das Problem der globalen Selbstlokalisation.

Die Problematik, ein möglichst „menschliches" Kommunikationsmittel zu erstellen, ist völlig unabhängig vom globalen Selbstlokalisationsproblem, da eine grundlegende Kommunikation zwischen Mensch und Maschine vorhanden ist. Eine effiziente Lösung einer Positionsverfolgung beeinflusst die Problematik der globalen Selbstlokalisation nicht, ebenso verhält es sich umgekehrt. Da aus einem effizienten Tracking-Algorithmus ein effizienter Navigationsalgorithmus resultiert, aber kein effizienter Explorationsalgorithmus, kann man das globale Selbstlokalisationsproblem als wichtigeres Problem klassifizieren. Somit kann man mit einer effizienten Lösung des globalen Selbstlokalisationsproblems die meisten Probleme eines Roboter-Systems lösen. Da ein Roboter-System Hauptbestandteil eines Service-Roboters ist, kann man zurecht die globale Selbstlokalisation als ein Schlüsselproblem der Robotik identifizieren.

8.3.2 Aufgaben und Prozesse eines Service-Tasks

Analog zu einem Roboter-System kann man auch für Service-Roboter die Menge der Rechenprozesse entsprechend in die zwei Gruppen Aufgaben und Prozesse klassifizieren. Es ergeben sich demnach die speziellen Aufgaben für einen Service-Roboter.

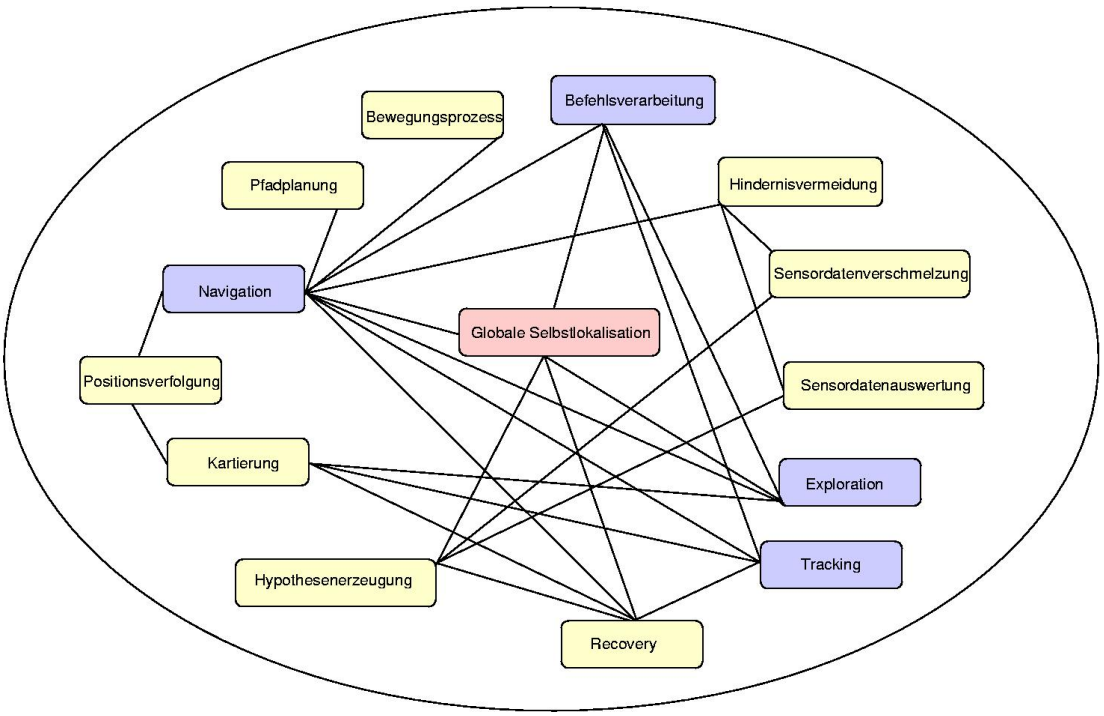
Erledigen der Service-Tätigkeit
Je nach Roboter und Einsatzgebiet soll der Roboter mit und ohne menschlichen Begleiter seine Service-Aufgabe ausführen.

Benutzereingaben
Kommunikation mit den Benutzern über eine Mensch-Maschine-Schnittstelle.

Die Aufgaben werden mit Hilfe der Prozesse und Aufgaben des Roboter-Systems gelöst. Dabei wird die Service-Tätigkeit mit Hilfe einer eines endlichen Automaten modelliert. Jedem Zustand des Automaten sind dabei ein oder mehrere Prozesse zugeordnet.

Man kann für einen Service-Task noch den folgenden Prozess zuordnen:

Abbildung 67: Prozessstruktur eines Roboter-Systems



In der obigen Abbildung kann man deutlich erkennen, welche zentrale Rolle die globale Selbstlokalisation einnimmt. Es sind dabei für jeden Prozess alle benötigten Prozesse markiert. Die dunkel markierten Prozesse repräsentieren die Aufgaben des Roboter-Systems.

Anfragen an Umgebung
: Kommunikationsprozess mit der intelligenten Umgebung, um Daten für den Service-Prozess zu akquirieren.

8.4 Resümee

In diesem Kapitel wurden Kriterien für das Design eines allgemeinen Roboter-Systems und eines Service-Roboters hergeleitet. Insbesondere wird dadurch klar, dass nur mit Hilfe der globalen Selbstlokalisation die Kriterien für ein Roboter-System erfüllt sind, sodass dieses sicher in der Einsatzumgebung operieren kann.

Weiter wurden Aufgaben und Prozesse und damit mögliche Dienste eines verteilten Systems klar zwischen Roboter-System und Service-Task getrennt. Die Spezifizierung der grundlegenden Aufgaben eines mobilen Roboters schränkt die möglichen Applikationen eines Service-Roboters wesentlich ein. Erst zusätzliche Hardware für die

Service-Tätigkeit, etwa ein Manipulatorarm, erweitert die Anzahl der Möglichkeiten.

Mit der Definition der Aufgaben eines Service-Roboters und dem Wissen über die Schwierigkeiten der einzelnen Problemstellungen aus den ersten beiden Teilen der Arbeit wird ersichtlich, dass noch ein weiter Weg zu tatsächlichen Service-Applikationen zurückgelegt werden muss.

9 Service-Roboter in intelligenten Umgebungen

Wurden in den vorherigen Kapiteln Aufgaben und Prozesse von Service-Robotern unterschieden und deren Anforderungen spezifiziert, bleibt in diesem Abschnitt die Aufgabe bestehen, eine durchdachte Architektur für einen Service-Roboter in intelligenten Umgebungen zu modellieren und exemplarisch den Einsatz eines solchen Systems darzulegen.

Dazu wird im folgenden der einfachste denkbare mobile Service-Roboter betrachtet: Dieser Service-Roboter besitzt eine Mensch-Maschine-Schnittstelle zur Interaktion mit dem Benutzer und hat die Aufgabe, vom Benutzer angewählte Zielpunkte in der Einsatzumgebung abzufahren. Die Service-Tätigkeit, das Abfahren ausgewählter Punkte, deckt sich mit der Spezifikation der Aufgaben eines mobilen Roboter-Systems - Kommunikation mit dem Benutzer und Anfahren eines Ziels. Nicht berücksichtigte Aufgaben sind dabei das Begleiten von Personen. Kartierung und globale Selbstlokalisation sind „versteckte" Aufgaben des Service-Roboters. Die Kartierungsaufgabe benötigt man zu Beginn, um eine Umgebungskarte zu akquirieren. Die globale Selbstlokalisation kommt zum Einsatz, wenn die interne Position im Rahmen der Navigation vom Roboter verloren wurde.

Zunächst werden allgemeine Architekturkonzepte mobiler Roboter aus der Literatur vorgestellt und die hier verwendete hybride Architektur erläutert. Anschließend wird die Einbettung des mobilen Roboters als Objekt in einer intelligenten Umgebung am Beispiel des einfachen Service-Roboters beschrieben.

9.1 Architektur eines Service-Roboters

Wie beim Software-Entwurf ist auch bei Service-Robotern der Entwurf des Systems entscheidend für den Erfolg. Der Entwickler muss entscheiden, welche Prozesse auf welche Komponenten verteilt werden, ob und wie Zustandsbildungen gewollt sind, wie zeit- und sicherheitskritische Probleme gelöst werden und wie der Informationsfluss im System funktioniert.

9.1.1 Architekturmodelle mobiler Roboter

In der Literatur kann man zwischen vier grundlegenden Architekturkonzepten für mobile Roboter unterscheiden:

Klassische Architektur
Hierarchische Architektur mit Weltmodell.

Subsumptions-Architektur
Agenten, welche durch Verhaltensweisen gesteuert werden.

Multi-Agenten-Architekturen
Miteinander interagierende einfache Agenten, die durch Kooperation mehr Intelligenz als ein Agent der Subsumptionsarchitektur erlangen.

Hybride Architekturen
Mischarchitekturen, die Teile der anderen Architekturen übernehmen.

Jedes in der Literatur beschriebene Roboter-System, jeder beschriebene Service-Roboter, lässt sich auf diese Architekturtypen zurückführen, sodass die einzelnen Architekturen im folgenden konkret vorgestellt werden.

Klassische Architektur

Die klassische Architektur eines Roboter-Systems, welche schon im Roboter Shakey [132, 155, 156, 202] zum Einsatz kam, besteht aus mehreren Komponenten. Ein zentralisiertes Weltmodell dient zum Planen der Aktionen des Roboters. Die Planung übernimmt das Planungssystem. Dafür stehen dann weitere spezialisierte Komponenten zur Verfügung. Die Systemkontrolle übernimmt die Ausführung der geplanten Aktionen, die zusätzlich überwacht werden, um auf unerwartete Situationen zu reagieren.

Bei den spezialisierten Komponenten zum Planen wird eine hierarchische Architektur der Komponenten gebildet. Sie unterteilen sich in Aktionen (sogenannte intermediate level actions oder ILAs) und grundlegende Aktionen (low level actions oder LLAs). Das Hauptaugenmerk liegt dabei darauf, eine kompakte und ausreichende Menge von Aktionen für das Planungssystem zu beschreiben.

Das Zusammenspiel aller Systemkomponenten wird durch den sogenannten Sense-Think-Act-Zyklus beschrieben. Die Systemkontrolle nimmt dabei in einem Iterationsschritt Sensordaten auf und liefert sie an das Planungssystem. Aus dem generierten bzw. überarbeitetem Weltmodell werden eine Reihe von ILAs generiert. Diese wiederum spalten sich in eine Reihe von LLAs auf, die einzeln abgearbeitet werden. Nach der Abarbeitung aller ILAs wird durch neue Sensorinformationen geprüft, ob der geplante Prozess wie gewünscht ausgeführt wurde.

Die Vorteile dieser Architektur liegen in der einfachen Durchschaubarkeit und der einfachen Programmierung der Architektur. Der Nachteil liegt in der Behäbigkeit der Architektur. Ein Iterationsschritt kann sehr lange dauern, sodass der Roboter nicht schnell genug auf gefährliche Situationen reagieren kann. Das System funktioniert also nur, wenn sich die Welt in einem Zyklus nicht ändert. Ein Welt-Update des Systems ist in der Regel nicht in Echtzeit möglich.

Subsumptions-Architektur

Die Subsumptionsarchitektur wurde 1986 von Rodney Brooks [32, 33] in die Robotik eingeführt. Dabei wird angenommen, das intelligentes Verhalten auch ohne explizite Wissensrepräsentation und abstrakte Schlußfolgerungsmöglichkeiten möglich ist. Die Intelligenz ist implizit in komplexen Systemen enthalten, bzw. entsteht erst durch Interaktion der einzelnen Agenten mit der Umwelt. Ein interner Zustand ist nur notwendig für Aspekte in der Umgebung, die nicht durch Sensoren gemessen werden können und dazu benötigt werden, Verhaltensweisen zu entscheiden.

Im Unterschied zum klassischen Modell wird ein globales Weltmodell verhindert. Anstatt eines Sense-Think-Act-Zyklus wird für jedes Verhalten ein reflexartiges Stimulus-Response Reaktionsmuster realisiert. Der monolithische Block der Modellbildung wird durch eine verteilte Verhaltensarchitektur ersetzt.

Dem Roboter werden mehrere Verhaltensweisen (sogenannte Behaviours, Instinkte) zugeordnet. Eine Hierarche von Verhaltensweisen repräsentieren zum Beispiel die Verhaltensweisen Hindernisvermeidung, Wandverfolgung und Korridorverfolgung.

Jedes obige Modul ist für eine bestimmte Verhaltensweise zuständig. Jedes verhaltensbasierte Modul kontaktiert die Sensoren (Stimulus) und sendet autonom die Effektorsignale (Response). Die Verhaltensweisen oder Kompetenzmodule werden in einzelne Hierarchien unterteilt. Höherwertige Verhaltensweisen können die niederen Kontaktieren und deren Verhalten ändern. Niedrige Verhaltensweisen sind für grundlegende, primitive Aufgaben zuständig, während höherwertige Verhaltensweisen komplexe Aufgaben repräsentieren. Verhaltensweisen können beliebig zugeschaltet werden und steuern so die Aktionen des Roboters. Ein separates Kontrollmodul kombiniert die verschiedenen Verhaltensweisen.

Die Vorteile dieses Ansatzes liegen in den schnellen Reaktionszeiten des Roboters. Dies ist mit einer Reihe von Nachteilen verbunden. Es ist zum einen sehr schwierig nachzuvollziehen, wie durch Kombination von mehreren Verhaltensweisen komplexere Verhaltensweisen entstehen können. Zum anderen benötigt man einen hohen Aufwand für Synchronisations- und Managementaufgaben.

Multi-Agenten-Architektur

Die Multi-Agenten-Architektur ist eine Erweiterung der Subsumptionsarchitektur [230]. Man geht dabei davon aus, dass mehrere, mit weniger Sensorik ausgestattete Agenten durch Interaktion Probleme lösen können, die mit einem Agenten nicht möglich gewesen wären.

Die normalen Agenten werden dabei um Kommunikationsfähigkeit und Interaktion erweitert. Hauptprobleme dieses Ansatzes sind die Konflikterkennung und Konfliktauflösung bei der Kommunikation mit anderen Agenten.

Zudem ist es schwierig, komplexe Probleme in eine Multi-Agenten-Repräsentation zu zerlegen. Es ist dabei problemabhängig, wie man widerspruchsfreie Zielsysteme erzeugen kann und wie das strategische Verhalten der einzelnen Agenten auszusehen hat.

9.1.2 Ein Kompromiss: die hybride Architektur

Eine hybride Architektur ist meistens ein Kompromiss der obigen Architekturkonzepte. Sie verbindet reaktive Ansätze mit Konzepten zur Bildung eines zentralen Weltmodells. Da auch die verwendete Architektur auf einem solchen Kompromiss beruht, soll sie im folgenden vorgestellt werden.

Diese geht aus der Analyse verschiedenster Hard- und Softwarekonzepte hervor. Grundlage dieses hybriden Entwurfes ist die Aufspaltung der Aufgaben und Prozesse eines Roboter-Systems. Die dem Pioneer-Roboter beigelegte Software der Firma ActivMedia zerlegt die Menge der Prozesse in zwei Teilmengen. Zum einen in reaktive Steuerungsprozesse, welche direkt auf dem Roboter durch die Software Aria [3] angewendet werden und zum anderen in Prozesse zur Visualisierung und Modellbildung auf einem externen Rechner durch ActivMedia Basic Suite [2].

Der Roboter Rhino [69, 207, 215], Benchmarkproblem für mobile Roboter-Systeme, besitzt eine verteilte Kommunikationsstruktur, welche über ein lokales Netzwerk im RWI 21 Roboter mit vier Prozessoren verfügt. Allerdings wurde kein Wert auf wiederverwendbare Software unter Einbeziehung von Entwurfsmustern gelegt, zudem sind Rechnerkapazitäten auf einem mobilen Roboter beschränkt. Rhino besitzt ein eigenes Benutzerinterface, sowohl auf dem Roboter als auch als Webinterface, eine weitere klare Trennung der internen Struktur wurde nicht vorgenommen.

Am FAW Ulm wird die Software SmartSoft verwendet, welche mittlerweile Teil des OROCOS-Projekts ist [190, 191]. Diese identifiziert Prozesse und Aufgaben eines mobilen Roboters und kapselt diese in einzelne Dienste. Zusätzlich werden diesen Diensten Kommunikationsmuster zugeordnet, so dass in einer verteilten Software-Architektur diese Dienste einfach angebunden werden können.

Die hier verwendete Architektur [35, 186] für das Roboter-System ist in zwei eigenständige Module aufgeteilt. Dabei modelliert das RobotBase-Modul die Funktionalität eines Base-Servers in SmartSoft, die Software RoLoPro modelliert alle Algorithmen des Roboter-Systems. Die Software RobotBase stellt ein Interface für ein generisches mobiles Roboter-System bereit. So ist man in der Lage, auf andere Robotertypen und deren Eigenschaften zu reagieren sodass die Hardware plattformunabhängig durch ein Programm angesteuert werden kann, welches reaktive Züge trägt und echtzeitfähige Reaktionen ermöglicht. Vom jeweiligen Robotertyp müssen dann auf der Hardware-Ebene nur die einfachsten Bewegungsbefehle umgesetzt werden, sodass man ein uniformes Verhalten des Roboters erwarten kann. RobotBase besitzt zeitkritische, reaktive Prozesse zur Sensor- und Aktorsteuerung, um auf sicherheitskritische Situationen reagieren zu können. Eine Ein-/Ausgabe-Schnittstelle erlaubt die Kommunikation mit anderen Softwaremodulen.

Die Software RoLoPro modelliert die Algorithmen-Schicht des Systems, so dass rechenintensive Prozesse auf schnelle Rechner ausgelagert werden können. Außerdem wird ein Sense-Think-Act-Zyklus realisiert. Dabei werden Aktionen geplant, aber an den Roboter zur Ausführung weitergegeben und auf ein Ergebnis vom Roboter gewartet. Währenddessen wird das Umgebungsmodell mittels neuer Sensordaten aktualisiert. Für die externen Prozesse ist nicht unbedingt eine Echtzeit-Fähigkeit verlangt.

Eine Funkverbindung dient als generische Schnittstelle zwischen den Softwareteilen des Systems. Damit verbunden sind Probleme aufgrund langsamer Übertragungswege, beschränkter Bandbreite, sowie unsicherer und instabiler Übertragungswege.

9.2 Dekomposition eines Service-Roboter

Das Architekturkonzept zerlegt ein Roboter-System in zwei unabhängige Software-Module. Diese Zerlegung wird auch von intelligenten Objekten erwartet, sodass ein Roboter-System nichts anderes ist als ein intelligentes Objekt, welches in einem Informationsraum einer intelligenten Umgebung mit einem Objekt-Server kommuniziert. Der Informationsraum bildet dann zusammen mit dem Objekt den Service-Roboter. Dieser Entwurf unterscheidet sich von bisherigen Ansätzen, da eine strikte Autonomie nicht mehr vorausgesetzt wird. Damit geht nicht wirklich etwas verloren, waren mobile Roboter schon immer auf ihre Einsatzumgebung beschränkt. Zukünftig wird mit diesem Verlust an Autonomie und der steigenden Anzahl der Sensoren das globale Selbstlokalisationsproblem an Bedeutung verlieren, was durchaus gewollt ist.

9.2.1 Modellierung des intelligenten Roboter-Objektes

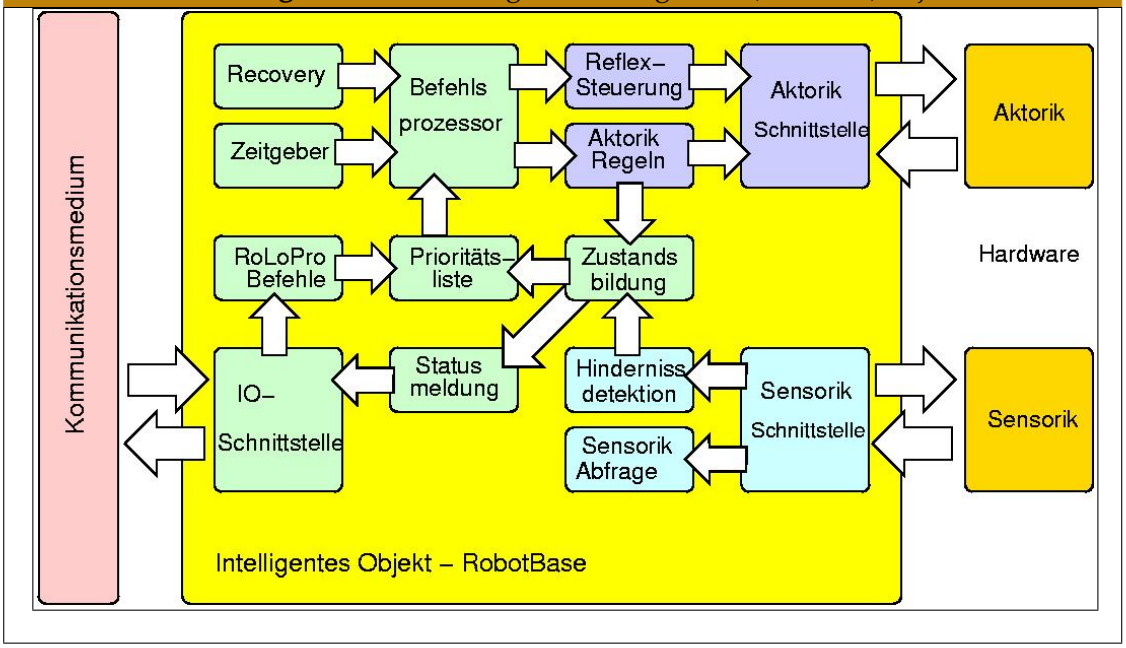
Das intelligente Objekt, welches durch die Hardware des mobilen Roboters und die Software RobotBase repräsentiert wird, umfasst nur Teile des Roboter-Systems, wie es in Kapitel 8.2 vorgestellt wurde.

Das intelligente Objekt wird durch Befehle des Objekt-Servers gesteuert, soll aber auch reaktives Verhalten besitzen. Eine Schnittstelle muss dementsprechend Kommandos empfangen und gleichzeitig Sensordaten versenden. Das intelligente Objekt muss jedoch auch selber aufgrund der ihm zur Verfügung stehenden Daten durch verhaltensbasierte Ansätze auf Umwelteinflüsse schnell reagieren und sicherheitskritische Aspekte erfüllen (Abbildung 68).

Das hier modellierte intelligente Objekt trennt die Aufgabe des Roboter-Systems der Navigation auf. Das Objekt ist nur noch Befehlsempfänger für anzufahrende Ziele, kann aber den Weg zu den Zielen durch eine lokale Hindernisvermeidung frei gestalten. Die Planungskomponente ist also in Prozesse der intelligenten Umgebung ausgelagert. Betrachtet man die Struktur der Navigation (siehe Kapitel 5), dann verbleibt nur die Hindernisvermeidung und Aktorsteuerung auf dem intelligenten Objekt.

Weiter gibt es Sensorauswertungsprozesse für die eingesetzten Sensoren (siehe auch Anhang D). Ein Rechenprozess ist für die Sicherheit des Systems zuständig. Ein Kommunikationsprozess versenden schließlich in kontinuierlichen Zeitintervallen angefallene ausgewählte oder angeforderte Sensordaten an den Objekt-Server.

Abbildung 68: Modellierung des intelligenten (Roboter-)Objekts



9.2.2 Modellierung des Informationsraumes des Roboter-Objektes

Der Informationsraum des Roboter-Objektes wird durch den Roboter-Server RoLoPro repräsentiert und besteht aus einer komplexen Sammlung von Algorithmen für mobile Roboter. Diese sind in verschiedene Services oder Dienste aufgeteilt, sodass Algorithmen eines Dienstes untereinander ausgetauscht werden können. Ein Dienst stellt eine fest definierte Schnittstelle dar, welche von anderen Algorithmen genutzt werden kann (siehe dazu auch Kapitel 8.2).

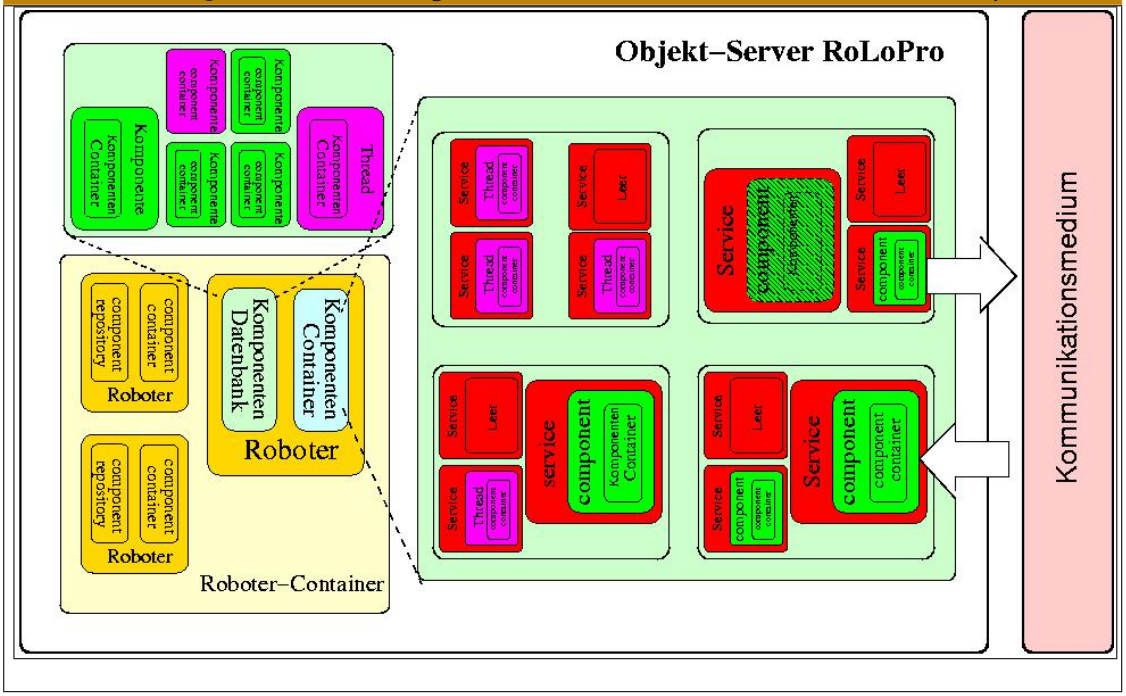
Zu jedem Dienst gibt es also mehrere Ausprägungen, die Komponenten repräsentieren. Komponenten sind Algorithmenmodule, welche neben einer generischen Ausprägung auch zusätzliche Dienst-Schnittstellen anbieten. Alle möglichen Algorithmen-Module/Komponenten sind in einem Komponenten-Container abgelegt. Für jede Service-Anwendung ist man nun in der Lage, einen entsprechenden Roboter aus generischen Diensten zusammenzubauen. Der zusammengestellte Roboter befindet sich anschließend in der Komponenten-Datenbank, welche aktive Komponenten markiert. Die abstrakte Architektur ist in Abbildung 69 dargestellt. In Abschnitt 8.2 hat man zwischen Prozessen und Aufgaben eines Robotersystems unterschieden. Diese finden sich als eigenständige Komponenten mit Prozessen in der Architektur wieder. Die Unterscheidung nach Aufgabe oder Prozess findet man nur in der Zuordnung der Komponenten zu den jeweiligen Services.

Abhängig von der Service-Tätigkeit des Roboters gibt es einen ausgezeichneten Prozess, welcher diese im Kontext der anderen Prozesse steuert. Dieser zu modellierende Prozess beschreibt die „Intelligenz" des Roboter-Systems. Die Kommunikation des Informationsraumes mit dem dazugehörigen Objekt geschieht über einen einzelnen Kommunikationsprozess. Entwürfe, welche Design-Patterns für Kommunikationsprimitive verwenden [190, 191], können einfach in das Konzept integriert werden. Damit würde der zentrale Kommunikationsprozess zugunsten lokaler, in die Services integrierte Kommunikationsprimitive wegfallen.

Betrachtet man nun das Beispiel des einfachen Service-Roboters, benötigt man für die entsprechende Robotersteuerung einen Service-Prozess, welcher auf die Navigationsbefehle des HMI reagieren kann. Als Wissensbasis wird eine 2-D Karte der Umgebung mit einbeschriebenen Zielen erwartet. Abhängig vom ankommenden Ziel und abhängig vom Standort des Roboters wird mittels der Pfadplanung ein Weg geplant und an das intelligente Objekt versendet. Ein zusätzlicher Kartierungs-Prozess bildet mit der globalen Selbstlokalisation und der Positionsverfolgung die Grundlage zur sicheren Positionsbestimmung.

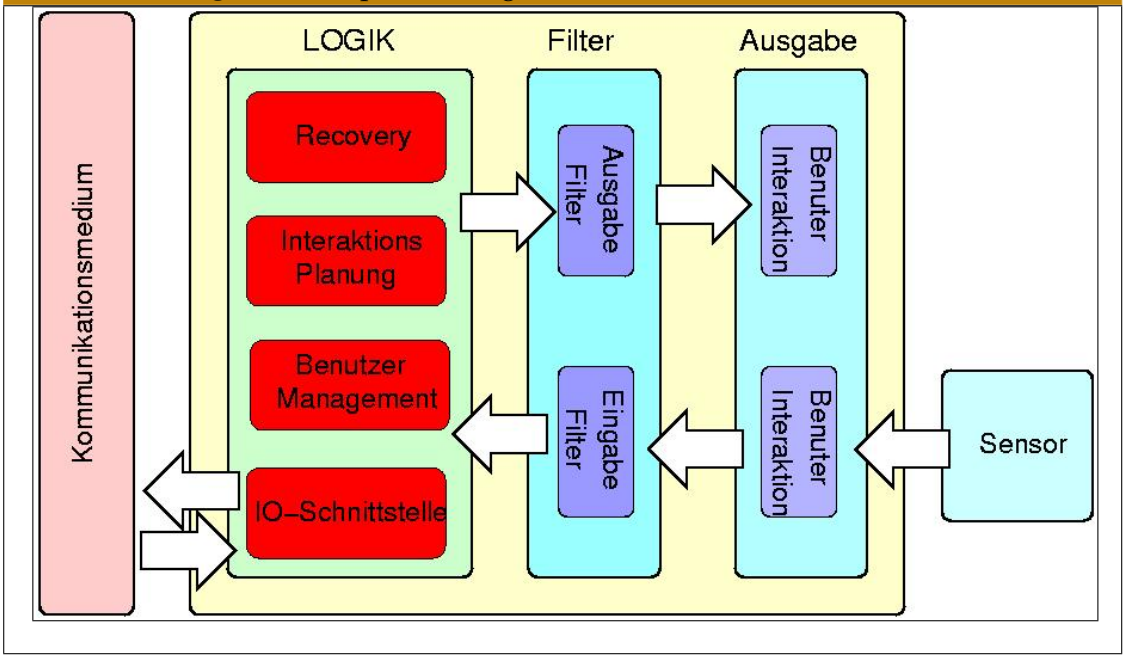
Mittels der obigen Services ist es sehr einfach möglich, weitere intelligente Objekt in die Umgebung zu integrieren oder Algorithmen auszutauschen. Damit sind Verfahrensvergleiche mit anderen Ansätzen der Literatur möglich.

Abbildung 69: Modellierung des Informationsraumes des (Roboter-)Objekts



9.2.3 Mensch-Maschine Schnittstelle als aktives Objekt

Abbildung 70: Konzept für eine generische Mensch-Maschine-Schnittstelle



Für weitere Service-Aufgaben muss man das Konzept einer Mensch-Maschine-Schnittstelle abstrakt formulieren. Die in Abbildung 70 dargestellte allgemeine Schnittstelle leistet dies. Die Schnittstelle besitzt mehrere Interaktionsmöglichkeiten mit dem Benutzer und spiegelt dabei auch verschiedene Eingabemedien wieder, wie Interaktion mittels Sprache oder Touch-Panel. Eine Filterschicht versucht die teilweise redundanten Informationen des Benutzers zu einer Intention zusammenzufassen. In einer Logik-Schicht wird auf diese Intention, abhängig vom gerade modellierten Benutzermodell, durch ein Planungsmodul reagiert. Das Planungsmodul greift dabei auf verschiedene heterogene Wissensquellen der intelligenten Umgebung zu und bildet einen Zustand. Dieser ermöglicht es, durch die Ein-und Ausgabeschicht mit dem Roboter-System zu interagieren. Alternativ kann das Planungsmodul durch einen Eingabefilter den Benutzer auf vielfältige Art und Weise nach weiteren Daten fragen, weitere Auswahlmöglichkeiten anbieten oder Statusmeldungen ausgeben. Ein Recovery-Mechanismus überwacht den Zustand des HMIs und stellt gegebenenfalls eine Notfall-Prozedur bereit.

Im Kontext des zu modellierenden einfachen Service-Roboters stellt die intelligente Umgebung ein Kartenmodell sowie Kontextinformationen über wichtige Ziele innerhalb der Umgebung zur Verfügung. Das HMI stellt diese Informationen dar und erlaubt dem Benutzer, ein Ziel auszuwählen. Da nur ein Eingabemedium zur Verfügung steht, wird die Filterfunktion nicht benötigt. Das ausgewählte Ziel wird an die intelligente Umgebung weitergeleitet, d.h. an den Objekt-Server, welcher eine Trajektorie vom Roboterstandort zum ausgewählten Ziel plant. Der geplante Weg wird dem intelligenten Objekt des Roboter-Systems übergeben.

Die obige Modellierung setzt einen Objekt-Server voraus. Es ist auch möglich, mittels des HMIs mit dem intelligenten Objekt des Roboter-Systems zu interagieren. Allerdings verlangt diese direkte Kommunikation den Preis, dass nur noch einfache Interaktionsbefehle übertragen werden, da die Steuerungsprozesse weggefallen sind. Dies spiegelt sich auch in anderen denkbaren direkten Kommunikationsvarianten zwischen aktiven Objekten wieder. Die Intelligenz ist beschränkt auf die Ressourcen der Objekte.

Eine weitere Möglichkeit zur Modellierung ist denkbar: Man kann das HMI in einem separaten Informationsraum mittels eines Content-Management-Systems modellieren [145]. Dieses sammelt Informationen der Benutzer in der Einsatzumgebung und speichert diese in einer Wissensbasis ab. Das HMI greift auf diese Wissensbasis zu und stellt dessen Daten dar. Somit muss in mindestens einem Informationsraum - wenn man sich das Roboter-System in einem weiteren Informationsraum vorstellt - ein Service-Prozess modelliert werden, welcher zwischen den Informationsräumen vermittelt und die Service-Tätigkeit bereitstellt.

9.3 Resümee

In diesem Abschnitt wurde am Beispiel einer einfachen Navigations-Service-Tätigkeit gezeigt, wie das komplette System eines Service-Roboters in verschiedene Teile zerlegt werden kann. Dabei wurden generische Konzepte für die Modellierung einer Mensch-Maschine-Schnittstelle, für eine mobile Roboter-Plattform und für eine komponentenbasierte Software zur Steuerung mobiler Objekt aufgezeigt.

Mit Hilfe des Konzepts der intelligenten Objekte in intelligenten Umgebungen und der generischen Modellierung eines Service-Roboters besitzt man ein Framework, mit welchem sich in Zukunft Anwendungen mobiler Roboter entwickeln lassen können. Die mit diesem Framework realisierten Service-Roboter sind dabei nicht mehr völlig autonom.

10 Resümee

Abschließend soll in diesem Kapitel die vorliegende Arbeit zusammengefasst werden und dabei neben den wichtigsten Errungenschaften auch Beiträge für andere Forschungsgebiete aufzeigen.

10.1 Zusammenfassung

In dieser Arbeit wurde die Problemstellung der globalen Selbstlokalisation autonomer mobiler Roboter in typischen Gebäude-Einsatzumgebungen bearbeitet. Dabei wurde das Problem zu Beginn in mehrere Teilprobleme zerlegt. Zum einen waren dies die Erzeugung von hypothetischen Roboterpositionen und zum anderen die Elimination dieser durch intelligente Steuerung des Roboters innerhalb der Einsatzumgebung. Beide Probleme sind jedoch so komplex, dass sie eine Reihe von weiteren Teilproblemen nach sich ziehen.

Im ersten Teil der Arbeit musste deshalb zunächst ein Framework geschaffen werden. Dieses bestand aus effizienten Verfahren, um Features aus Sensordaten zu extrahieren und auf eine Featurerepräsentation der Einsatzumgebung abzubilden. Die Einsatzumgebung musste außerdem mit Hilfe von Sensordaten erzeugt und in eine Feature-Repräsentation gebracht werden. Zudem mussten die feature-basierten Verfahren um Heuristiken erweitert werden, um schnelle Anfrageergebnisse der Sensordaten an die Kartenrepräsentation zu verarbeiten. Die verwendeten Teillösungen wurden abschließend getestet.

Im zweiten Teil der Arbeit wurden zunächst Navigationsstrategien für einen mobilen Roboter vorgestellt um mit Hilfe von Karten der Einsatzumgebung Pfade zu angewählten Punkten zu planen und diese sicher abzufahren. Mit dieser Grundlage konnte ein effizientes Verfahren zur Lösung der globalen Selbstlokalisation realisiert werden. Dazu wurden mehrere Kartenrepräsentationen der Umgebung verwendet. Mit einer Gitterkartenrepräsentation konnten Sensorpunkte identifiziert werden, die vom Roboter im globalen Selbstlokalisationsprozess noch nicht abgefahren worden sind. Mit Hilfe der Navigationsalgorithmen kann man zu diesen Punkten gelangen. Gleichzeitig erstellten Kartierungsverfahren ein lokales Umgebungsmodell, welches für feature-basierte Lokalisationsanfragen als Bildfeature-Menge dient. Mit Hilfe einer immer sicherer werdenden Informationsmenge war man in der Lage, das gestellte Problem zu lösen. Der EFM-Algorithmus wurde abschließend in mehreren Testumge-

bungen auf seine Einsatzfähigkeit erfolgreich getestet.

Im dritten Teil der Arbeit wurde schließlich darauf eingegangen, welche Anforderungen ein mobiler Roboter besitzen muss, um erfolgreich eine Service-Tätigkeit auszuführen. Gleichzeitig wurden die grundlegenden Aufgaben eines mobilen Roboters aufgezeigt, wodurch deutlich wurde, dass man mit dem globalen Selbstlokalisationsproblem alle anderen Problemstellungen der Robotik gleichzeitig lösen muss. Abschließend wurde ein generisches Architekturkonzept aufgezeigt, welches sich an den Ansatz des „Ubiquitous Computing" anlehnt und mobile Roboter als intelligente Objekte in einer intelligenten Umgebung entwirft.

10.2 Rückblick und Bewertung

Zu Beginn der Arbeit wurden die Ziele der Arbeit definiert, welche nochmalig aufgegriffen und deren Resultate diskutiert werden sollen.

Erstes Ziel war die Entwicklung eines leistungsfähigen Algorithmus zur Lösung des Problems der globalen Selbstlokalisation. Dieses Ziel wurde mit der Beschreibung des EFM-Algorithmus erreicht. Dieser benötigt jedoch zur erfolgreichen Ausführung mehrere komplexe Teilalgorithmen, für welche jeweils ein wiederverwendbares Framework geschaffen wurde. Die Problemstellungen der feature-basierten Lokalisation, der effizienten Kartenrepräsentationen und Kartierungsverfahren sowie der Navigation wurden dabei in Teilprobleme zerlegt und für jedes solche konkurrierende Algorithmen entworfen.

Dabei wurden mehrere innovative Lösungen entworfen. So wurde insbesondere der inkrementelle feature-basierte Lokalisationsalgorithmus mit entsprechenden Heuristiken vorgestellt. Dieser zeigt sich im Vergleich mit herkömmlichen Ansätzen für diese Problematik überlegen in Hinsicht des Zeitverbrauchs und der erreichten Lokalisationsgüten. Mehrere Kartenrepräsentationen wurden erstmals nebeneinander verwendet im Unterschied zu bisherigen Ansätzen der Literatur. Damit gelingt es, die Vorteile dieser Ansätze für nachfolgende Verfahren zu kombinieren. Erst damit kann der Einsatz des EFM-Algorithmus gelingen. Zusätzlich wurde die Gitterkartenrepräsentation um die Integration dynamischer Zustände erweitert und damit ein modifizierter inkrementeller probabilistischer Ansatz zur Kartierung vorgestellt. Weiter wurde ein Navigationsalgorithmus beschrieben, welcher bekannte Ansätze erstmals in ein wiederverwendbares Framework integriert. Schließlich wurde ein Explorationsalgorithmus vorgestellt, welcher effizient Sensorpunkte extrahiert, eine MDL-Strategie verwendet und stark an die theoretische Problemlösung angelehnt ist. Die Vorstellung des EFM-Algorithmus bildet den Höhepunkt des Algorithmen-Entwurfs.

Das Ziel, einen leistungsfähigen Algorithmus zu realisieren, wurde anhand von prak-

tischen Tests in zwei Umgebungen bestätigt. Wie leistungsfähig der Algorithmus wirklich ist, lässt sich aber erst in einer vergleichenden Studie mit einem effizienten aktiven Markov-Ansatzes realisieren, welche hier nicht realisiert wurde. Man kann jedoch schon konstatieren, dass die Markov-Lokalisation der einfachere zu implementierende Algorithmus bleibt.

Im praktischen Einsatz der globalen Selbstlokalisation sind für einen Service-Roboter noch mehrere Probleme zu lösen. So war in den Tests eine Funküberdeckung der Einsatzumgebung gefordert, die es real nur in vorzeigbaren Gebäudekomplexen gibt. Die Systemsicherheit war in den Tests nicht immer von allein sichergestellt, da die Sensorik für das System unzureichend war. Treppenabgänge etwa wurden gesondert gesichert und sollten zukünftig von weiteren Sensoren erkannt werden. Insgesamt gilt es weiter, die Intelligenz des Systems zu verbessern. Dies reicht von Detailproblemen, etwa wie man schnell aus einer Ecke wieder herauskommt, bis hin zu allgemeinen Problemen, zum Beispiel einen Punkt anzufahren, welchen man aus menschlicher Sicht aus auch angefahren hätte.

Zweites Ziel war die Berücksichtigung weiterer Probleme mobiler Roboter, die für die Lösung benötigt wurden. Dieses Ziel wurde insoweit erreicht, als das jedes behandelte Teilproblem, welches für die globale Selbstlokalisation nötig war, separat besprochen wurde. Für jede Teilaufgabe wurden aktuelle Lösungen der Literatur präsentiert und dann eine Lösung realisiert, die für das globale Ziel notwendig war. Außerdem ist die globale Problemlösung so konzipiert, dass nach dem Baukastenprinzip eine Teillösung durch eine andere ausgetauscht werden kann und so insgesamt zu einer Performanceverbesserung beitragen kann. Damit können auch einfach Algorithmenvergleiche untersucht werden. Mit der Dekomposition in Teilprobleme und einem generellen Ansatz hat man erstmals ein Framework zur Verfügung, welches Austauschbarkeit von Algorithmen möglich macht. Zudem liefert diese Arbeit dadurch einen Überblick über das Zusammenwirken unterschiedlichster Problemlösungen mobiler Roboter. Nachteile dieses Entwurfs können aber auch Perfomanceeinbussen aufgrund generischer Schnittstellen der einzelnen Dienste sein, die ein speziell-entwickeltes System nicht aufweist.

Letztes Ziel war die Einbettung der Problemlösung in ein allgemeines Konzept für einen Service-Roboter. Dazu wurden im letzten Teil der Arbeit Anstrengungen unternommen, zunächst eine andere Sicht der Dinge auf den Einsatz von Service-Robotern zu bekommen. Aus der Sicht des „Ubiquitous Computing" ist ein Service-Roboter ein intelligentes Objekt in einer Umgebung, welcher die Umgebung sogar manipulieren kann. Damit wird diese andere Sichtweise um manipulierende Objekte erweitert.

Für eine Integration von Service-Robotern in Einsatzumgebungen wurde dann ein abstraktes Framework geschaffen, welches diesen in mehrere Teile als Roboter-Objekt, Objekt-Server und Wissensbasen aufteilt. Diese Aufteilung wurde in dem in dieser Arbeit verwendeten Testsystem realisiert und hat sich bewährt. Es ist durch sei-

ne generische Schnittstelle auf andere Roboter-Objekte portierbar, etwa auf einen Nomadic-Roboter. Der Objekt-Server repräsentiert die wesentlichen Algorithmen agierend über spezifizierte Schnittstellen, so dass diese einfach austauschbar sind und der Objekt-Server selber auf den entsprechenden Service-Task einstellbar ist. Eine Vereinfachung des gesamten Handlings, etwa ein eigenes komfortables „Roboterbetriebssystem" auf der Grundlage standardisierter Problemlösungsalgorithmen für einzelne Probleme, wäre ein weiterer Schritt in die richtige Richtung. Erste Ansätze dazu sind mit dem RoLoPro-Projekt [186] gemacht worden, weitere Projekte wie OROCOS-Smartsoft [190] könnten diesen ergänzen und verbessern und zu einem generischen Framework für Service-Roboter mutieren. Mit dieser Perspektive wäre es möglich, einen Quantensprung in der Robotik zu vollziehen, da unzählige Ressourcen für tatsächliche Robotik-Forschung frei würden.

10.3 Beiträge

Nach dem Rückblick und der Bewertung der geleisteten Ergebnisse aufgrund der formulierten Ziele dieser Arbeit sollen an dieser Stelle die Beiträge zur wissenschaftlichen Forschung und Lehre herausgearbeitet werden, inwieweit sie für andere Nutzen und Übertragbarkeit bringen können.

Anwendung in der Lehre

Diese Arbeit liefert durch die komplexe Problemstellung eine sehr aktuelle Übersicht über die Probleme der mobilen Robotik in seiner Gesamtheit. Sie resultiert nicht zuletzt aus Übersichtsvorlesungen bzw. Einführungsvorlesungen in diese Problematik. Eine standardisierte Betrachtung der Schnittstellen der Algorithmen und deren Zusammenspiel, der Aufbau von Algorithmen-Bibliotheken könnte dem Forschungsgebiet der mobilen Robotik zu neuem Schwung verhelfen und die Sicht auf grundlegende, noch ungelöste Probleme freimachen.

Implementierung von Service-Robotern

Mit dem Resultat dieser Arbeit, dem EFM-Algorithmus, ist ein Schritt in Richtung komplexer, kombinierter Problemlösungsstrategien der Robotik erreicht worden. Der Algorithmus behandelt außerdem eine durchaus alltägliche Fehlersituation bei der Navigation mobiler Roboter. Mittels der kombinierten Betrachtungsweise, Roboter als intelligente Objekte in intelligenten Umgebungen anzusehen, könnte man schon bald Service-Roboter in Einsatzumgebungen wiederfinden. Die heutige Betrachtungsweise, Service-Roboter möglichst autonom in einer Umgebung einzusetzen, ist bisher gescheitert. Wenn man eine intelligente Umgebung voraussetzt, braucht man nicht viel um einen intelligenten Agenten in der Einsatzumgebung operieren zu lassen. Die Voraussetzungen werden durch zunehmende Funküberdeckung öffentlicher Gebäude und im eigenen Heim geschaffen, man muss sie nur nutzen.

Technisches Software-Engineering

Der Ansatz, Entwurfsmethoden des Software-Engineerings in technische Entwurfsprozesse zu bringen, sollte in allen Robotik-Labs Einzug halten. Das RoLoPro-Softwaresystem ist ein Schritt in die richtige Richtung durch Schnittstellendefinitionen einzelner Dienste und Ausprägungen von zugehörigen Komponenten. Neben der Anwendung solcher Konzepte müssen weiter Konzepte der Verifikation und Validierung der einzelnen Komponenten eines Dienstes angewandt werden, um sicherheitskritischen Aspekten eines solchen Systems gerecht zu werden. Solche Ansätze werden bisher weitestgehend ignoriert.

Ubiquitous Computing

Das Ubiquitous-Computing-Konzept um mobile Roboter zu erweitern, könnte beiden Forschungsrichtungen neue Impulse liefern. Die grundsätzlich informationstechnische Grundlage des Ubiquitous Computing-Ansatzes verknüpft mit dem technischgeprägten Ansatz der Robotik könnte eine ideale Ergänzung darstellen.

Algorithmische Robotik

In dieser Arbeit wurden eine Reihe von Algorithmen und Datenstrukturen verwendet, die eventuell noch besser mit noch besserer Laufzeit implementiert werden könnten. Für jede verwendete Algorithmenklasse wurden entsprechendes Interfaces im Rahmen dieser Arbeit etabliert. Halten sich andere Entwickler an solche zu diskutierende Schnittstellen, könnte man endlich wiederverwendbare und vergleichbare Software entwickeln. Die Diskussion solcher Schnittstellen, das Design effizienter Algorithmen für die unterschiedlichen Probleme könnte das Forschungsgebiet der Algorithmischen Robotik leisten und den hohen Reengineeringprozess in der Robotik reduzieren.

Modellbasierte Objekterkennung

Der Ansatz des EFM-Algorithmus, mittels einer immer größer werdenden Bildrepräsentation diese in der Modellrepräsentation zu suchen, könnte für komplexe Matchingprobleme der modellbasierten Objekterkennung vielversprechend sein. Aufgrund der allgemeinen Repräsentation mittels Feature-Mengen sind die Methoden aus Kapitel 4 leicht auf andere Objekterkennungsprobleme einsetzbar.

Medizinische Robotik

Die Problematik der globalen Selbstlokalisation taucht auch in der medizinischen Robotik auf. Dort muss ein Robo-Doc selbstständig sein Einsatzgebiet aufgrund von Sensoren oder Sonden im menschlichen Körper erkennen. Mit Hilfe des EFM-Algorithmus wäre man in der Lage aufgrund der Historie der Sensorinformationen eine Standorthypothese einer eingeführten Sonde zu berechnen.

A Statistische Grundlagen

Dieser Abschnitt wiederholt und erläutert grundlegende Begriffe aus der multivarianten Statistik.

A.1 Mittelwert, Varianz, Kovarianz, Kovarianzmatrix

Der Mittelwert oder Erwartungswert $\overline{x}$ einer Zufallsvariablen $X : E \to \mathbb{R}$ kann anschaulich als das „Massezentrum" einer Verteilung angesehen werden. Nimmt X nur endlich viele Werte $x_1, \ldots, x_n$ an, und setzt man $p_i = P(X = x_i)$ dann ist der Mittelwert $\overline{x}$ definiert durch:

$$\overline{x} = \sum_{i=1}^{n} x_i p_i \tag{A.1}$$

Fallen die Messwerte zu einem „Haufen" zusammen, so stellt man sich den Mittelwert mehr oder weniger im Zentrum des „Haufens" vor.

Die Varianz σ_x^2 ist ein Maß für die Größe dieses „Haufens", wieviel Abweichung vom typischen Wert vorliegt. Sie wird durch das arithmetische Mittel über das Quadrat der Standardabweichung oder Streuung σ_x vom Mittelwert definiert:

$$\sigma_x^2 = \overline{(X - \overline{x})} = \sum_{i=1}^{n} (x_i - \overline{x})^2 p_i \tag{A.2}$$

Man betrachte zunächst zwei Merkmale mit Zufallsvariablen X und Y. Dann ist die Kovarianz von (X, Y) definiert durch:

$$\mathrm{Cov}(X, Y) = \overline{(X - \overline{x})(Y - \overline{y})} \tag{A.3}$$

Die Kovarianz $\mathrm{Cov}(X, Y)$ besitzt folgende wichtige Eigenschaften.

1. Wenn Merkmal X und Merkmal Y gemeinsam zunehmen, dann ist $\mathrm{Cov}(X, Y) > 0$, die Funktionen X und Y sind also sozusagen „gleichläufig".

2. Nimmt Merkmal X tendenziell ab wenn Y zunimmt, dann ist $\mathrm{Cov}(X, Y) < 0$, die Funktionen X und Y sind sozusagen „gegenläufig".

3. Wenn Merkmal X von Merkmal Y unabhängig ist, d.h. X und Y sind unabhängige Zufallsvariablen, dann ist $\mathrm{Cov}(X, Y) = 0$.

4. Es gelten die Bedingungen $|\operatorname{Cov}(X,Y)| \leq \sigma_x \sigma_y$ und $\operatorname{Cov}(X,X) = \sigma_{x^2}$.

Die Kovarianzmatrix misst die Tendenz zweier Merkmale, gemeinsam zu variieren.

Seien nun $X_1, \ldots, X_d$ die Zufallsvariablen d verschiedener Merkmale und die Einträge der $(d \times d)$-Matrix $\Sigma = (\sigma_{ij})$ die Zahlen

$$\sigma_{ij} = \operatorname{Cov}(X_i, X_j), i, j = 1, \ldots, d \tag{A.4}$$

Dann bezeichnet man Σ als die Kovarianzmatrix des Zufallsvektors oder Merkmalsvektors $(X_1, \ldots, X_d)^T$.

A.2 Mahalanobis-Distanz

Die Kovarianzmatrix ermöglicht, die Distanz zwischen zwei Merkmalsvektoren zu messen und zwar invariant bezüglich linearer Transformationen auf den Messwerten. Sei beispielsweise x ein d-dimensionaler Zufallsvektor mit einem dazugehörigen Mittelwertvektor $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_d)^T$, wobei $\overline{x}_i$ der Mittelwert der Zufallsvariable X_i ist.

Dann kann die $(d \times d)$-Matrix A benutzt werden, um x in einen Vektor y zu transformieren mittels $y = Ax$. Den Mittelwertvektor erhält man durch $\overline{y} = A\overline{x}$ und die Kovarianzmatrix für y durch $\Sigma_y = A\Sigma_x A^T$.

Will man nun die Distanz von x zu $\overline{x}$ messen, verwendet man eine Norm $\|x\|$ bezüglich $\overline{x}$. Betrachte dabei zunächst nur den eindimensionalen Fall. Für ein einzelnes Merkmal lässt sich die Distanz durch den skalaren Ausdruck

$$r^2 = \left(\frac{x - \overline{x}}{\sigma_x}\right)^2 = (x - \overline{x})\frac{1}{\sigma_x^2}(x - \overline{x}) \tag{A.5}$$

normalisieren. Die Verallgemeinerung auf Matrixform im n-dimensionalen Fall ergibt dann:

$$r^2 = (x - \overline{x})^T \Sigma_x^{-1} (x - \overline{x}) \tag{A.6}$$

Die auftretende Größe r heißt die Mahalanobis-Distanz vom Merkmalsvektor x zum Mittelwertvektor $\overline{x}$, wobei Σ_x die Kovarianzmatrix für x ist.

Bei der Anwendung in der Mustererkennung könnte man ein Muster anhand eines Merkmalsvektors klassifizieren, indem man den Merkmalsvektor der Klasse zuordnet, zu welcher die Mahalanobis-Distanz minimal wird.

A.3 Dichtefunktion der Normalverteilung

Ein Zufallsvektor x heißt normalverteilt, wenn die Dichtefunktion f folgendermaßen definiert ist:

Im skalaren Fall mit Erwartungswert $\overline{x}$ und Varianz σ_x^2 gilt

$$f(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{1}{2}(\frac{x-\overline{x}}{\sigma_x})^2}$$

also

$$x \sim N(\overline{x}, \sigma_x^2)$$

Im n-dimensionalen Fall ist dies

$$f(x) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\Sigma_x)}} \exp -\frac{1}{2}(x-\overline{x})^T \Sigma_x^{-1} (x-\overline{x})$$

also

$$x \sim N(\overline{x}, \Sigma_x)$$

Dabei ist $\overline{x}$ ein n-dimensionaler Erwartungswert und Σ_x eine $n \times n$ dimensionale Kovarianzmatrix.

A.4 Kalman-Filter

Der Kalman-Filter repräsentiert eine Menge von Gleichungen, die eine effiziente rekursive Lösung des Problems der linearen Datenfilterung beschreibt, d.h. den Zustand eines zeitdiskreten evtl. nichtlinearen Prozesses vorhersagt [148, 235]. Dazu liegen Messwerte aus verschiedenen Messungen über den gleichen Sachverhalt vor, die durch den Filter zu einer, mit statistisch minimalem Fehler behafteten, Schätzung des Systemzustands führt. Jede dieser Messungen liefert eine normalverteilte Beobachtung mit Erwartungswert und Varianz.

Voraussetzung für den Einsatz des Filters ist die Möglichkeit einer linearen Modellbeschreibung des Systems und eine Normalverteilung des Fehlers der Schätzungen (also mindestens 2 Schätzungen). Der Kalman-Filter fusioniert die Schätzungen dann zu einer neuen Schätzung mit neuem Fehler, der wiederum normalverteilt ist.

Als Optimalitätskriterium für die Wahl der neuen Schätzung dient das Kriterium der minimalen Varianz d.h. die neue Schätzung wird so gewählt, dass der resultierende Fehler minimiert wird.

Angenommen, zwei Sensoren zur Entfernungsmessung seien so aufgestellt, dass sie den gleichen Abstand zu dem vor ihnen liegenden Objekt besitzen. Die Sensoren müssten also den gleichen Entfernungswert messen. Die Messungen seien mit einem Fehler behaftet. Die Messungen l_1 vom ersten Sensor und l_2 vom zweiten Sensor seien normalverteilt.

$$l_1 \sim N(\mu_1, \sigma_1^2) \tag{A.7}$$

$$l_2 \sim N(\mu_2, \sigma_2^2) \tag{A.8}$$

Betrachte zunächst den eindimensionalen Fall. Dabei entsprechen μ_1 und μ_2 den realen Entfernungen und σ_1 und σ_2 sind die jeweiligen Standardabweichungen, die man durch eine Messreihe bestimmen kann. Es ist nun naheliegend, ein gewichtetes Mittel der beiden Entfernungsmessungen zu verwenden, um sie zu verknüpfen. Die jeweiligen Gewichte sind dabei umgekehrt proportional zu der zugehörigen Varianz.

Seien l_1 und l_2 die Messungen der beiden Sensoren, dann berechnet sich der durch Kalman-Filterung bestimmte neue Abstandswert l mit Varianz σ^2 zu

$$l = \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \left(\frac{1}{\sigma_1^2} l_1 + \frac{1}{\sigma_2^2} l_2\right) \tag{A.9}$$

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \tag{A.10}$$

Falls nun gilt $\sigma_1 = \sigma_2$, dann reduzieren sich die obigen Gleichungen zu

$$l = 0.5 \cdot (l_1 + l_2) \tag{A.11}$$

$$\sigma^2 = \frac{\sigma_1^2}{2} \tag{A.12}$$

Die resultierende Varianz ist immer kleiner als jede der beiden Varianzen.

Im n-dimensionalen Fall liegen zwei n-dimensionale Vektoren m_1 und m_2 vor mit

$$m_1 \sim N(\mu_1, \Sigma_1) \tag{A.13}$$

$$m_2 \sim N(\mu_2, \Sigma_2) \tag{A.14}$$

Die durch Kalman-Filterung berechnete verbesserte Schätzung m mit Kovarianzmatrix Σ ergibt sich dann zu

$$m = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} (\Sigma_1^{-1} m_1 + \Sigma_2^{-1} m_2) \tag{A.15}$$

$$\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \tag{A.16}$$

A.5 Bedingte Wahrscheinlichkeiten

Häufig verfügt man bei der Durchführung von Experimenten über Vorinformationen, die bei der Berechnung von Wahrscheinlichkeiten interessierender Ereignisse berücksichtigt werden sollen. Bei manchen Untersuchungen wird jedoch lediglich (hypothetisch) angenommen, dass eine bestimmte Vorinformation vorliegt, wobei dann unter dieser hypothetischen Annahme gerechnet wird. Diese sogenannte Bayes'sche Methodik wird genauer diskutiert.

Definition A.1 (Bedingte Wahrscheinlichkeit) *Sei* $(\Omega, \mathcal{F}, P)$ *ein beliebiger Wahrscheinlichkeitsraum mit der Grundmenge* Ω*,* $\mathcal{F}$ *einer Familie von Teilmengen von* Ω *und* P *eine Abbildung* $P : \mathcal{F} \to [0, 1]$*.*
Seien $A, B \in \mathcal{F}$ *beliebige Ereignisse mit* $P(B) > 0$*. Dann heißt*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{A.17}$$

die bedingte Wahrscheinlichkeit von A *unter der Bedingung* B*.*

Satz A.2 (Multiplikationssatz) *Seien* $A_1, A_2, \ldots, A_n \in \mathcal{F}$ *Ereignisse mit* $P(A_1 \cap A_2 \cap \ldots \cap A_{n-1}) > 0$*. Dann gilt:*

$$P(A_1 \cap A_2 \cap \ldots \cap A_n) = P(A_1)\, P(A_2 \mid A_1)\, P(A_3 \mid A_1 \cap A_2)\, \ldots\, P(A_n \mid A_1 \cap A_2 \cap \ldots \cap A_{n-1}) \tag{A.18}$$

A.6 Die Bayes'sche Formel

Bei der Berechnung der Wahrscheinlichkeit $P(A)$ eines Ereignisses $A \in \mathcal{F}$ ist es manchmal nützlich, die (unbedingte) Wahrscheinlichkeit $P(A)$ als gewichtete Summe von bedingten Wahrscheinlichkeiten darzustellen. Hierfür ist es erforderlich, den Grundraum Ω wie folgt in (messbare) Teilmengen zu zerlegen.

Definition A.3 (Messbare Zerlegung) *Sei* $n \in \mathbb{N}$ *eine beliebige natürliche Zahl und* $B_1, B_2, \ldots, B_n \in \mathcal{F}$ *eine (endliche) Folge von Ereignissen mit den Eigenschaften*
(Z1) $B_i \cap B_j = \emptyset$ *für* $i \neq j$*,*
(Z2) $\bigcup_{i=1}^{n} B_i = \Omega$*,*
(Z3) $P(B_i) > 0$ *für alle* $i = 1, \ldots, n$*.*
Dann heißt $B_1, B_2, \ldots, B_n$ *messbare Zerlegung von* Ω*.*

Satz A.4 (Totale Wahrscheinlichkeit und Bayes'sche Formel) *Sei* $A \in \mathcal{F}$ *ein beliebiges Ereignis und* $B_1, B_2, \ldots, B_n$ *eine messbare Zerlegung von* Ω*. Dann gilt die Formel der totalen Wahrscheinlichkeit:*

$$P(A) = \sum_{j=1}^{n} P(B_j) P(A \mid B_j)\,, \tag{A.19}$$

und die Bayes'sche Formel

$$P(B_i \mid A) = \frac{P(B_i) P(A \mid B_i)}{\sum_{j=1}^{n} P(B_j) P(A \mid B_j)} \tag{A.20}$$

für jedes $i = 1, \ldots, n$*, wobei vorausgesetzt wird, dass* $P(A) > 0$*.*

Die Bayes'sche Formel wird in dieser Arbeit genutzt, um aus gegebenen Daten unter Zuhilfenahme von Wissen, das unabhängig von den vorliegenden Daten gewonnen wurde, Parameterschätzungen vorzunehmen.

B Bewertung von Algorithmen

Die in dieser Arbeit vorgestellten Algorithmen besitzen alle eine gewisse Laufzeit und einen Speicherverbrauch. Diese Größen sollen auf eine geeignete Weise charakterisiert werden. Hier werden Laufzeit und Speicherverbrauch durch die worst-case-Analyse abgeschätzt, d.h. es wird der schlechteste Fall betrachtet, der für den Algorithmus auftreten kann. Zunächst soll allgemein von der Komplexität des Algorithmus die Rede sein.

Die Komplexität sei spezifiziert durch eine Funktion f wobei der Wert $f(n)$ dann eine obere Grenze der Komplexität des Algorithmus für jede Instanz mit Größe n ist.

Die Komplexität wird durch die $\mathcal{O}$-Notation bestimmt, die wie folgt definiert werden kann: Es bezeichne $F = \{f : \mathbb{N} \to \mathbb{R}\}$ die Menge aller reelwertigen Funktionen auf $\mathbb{N}$. Für ein $f \in F$ sei

$$\mathcal{O}(f) = \{g \in F \mid \exists a \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N}, n > n_0 :\mid g(n) \mid \leq a \cdot f(n)\} \qquad \text{(B.1)}$$

Die $\mathcal{O}$-Notation hat die Eigenschaft, dass die Ergebnisse unabhängig von konstanten Faktoren sind, die durch Unterschiede aufgrund von Programmiersprachen oder aufgrund unterschiedlicher Maschinenmodelle entstehen.

Möchte man nun die **Laufzeit** abschätzen, betrachtet man die Menge der elementaren Operationen einer Instanz mit der Größe n. Jede elementare Operation kostet dann eine Zeiteinheit. Zu den elementaren Operationen gehören Zuweisungen, Vergleiche und elementare Rechenoperationen.

Möchte man den **Speicherplatz** abschätzen, betrachten man die Menge der benötigten Speicherplätze einer Instanz mit der Größe n. Jeder Speicherplatz einer Instanz, welcher erzeugt wird, kostet dann eine Speichereinheit.

C Komplexitätsklassen

In der Informatik taucht häufig das Konzept auf, ein Problem in eine Komplexitätsklasse einzustufen. Damit wird für ein Problem π ausgedrückt, wie „schwer" es zu berechnen ist. Um dieses Konzept zu formalisieren, betrachte zunächst die folgende Definition:

Definition C.1 (Entscheidungsproblem) *Ein Entscheidungsproblem π ist ein Problem, für welches jede Instanz eine Lösungsmöglichkeit aus der Menge {Ja, Nein} besitzt.*

Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ sei $\mathrm{DTIME}(f(n))$ eine Menge, welche alle Entscheidungsprobleme enthält, die auf einer deterministischen Turing-Maschine in $\mathcal{O}(f)$ entschieden werden können. Dementsprechend sei $\mathrm{NTIME}(g(n))$ für eine Funktion $g : \mathbb{N} \to \mathbb{N}$ die Menge, welche alle Entscheidungsprobleme enthält, die auf einer nichtdeterministischen Turing-Maschine in $\mathcal{O}(g)$ entschieden werden können.

Daraus resultieren die in der Literatur [15] wichtigen Klassen $\mathcal{P}$ und $\mathcal{NP}$ definiert durch

$$\mathcal{P} = \cup_{k=1}^{\infty} \mathrm{DTIME}(n^k) \tag{C.1}$$

und

$$\mathcal{NP} = \cup_{k=1}^{\infty} \mathrm{NTIME}(n^k) \tag{C.2}$$

Mittels dieser Klassen kann man für ein Entscheidungsproblem eine fundamentale Aussage machen: Ein Entscheidungsproblem π heißt $\mathcal{NP}$-vollständig, wenn es in der Menge $\mathcal{NP}$ liegt und jedes andere Problem $\gamma \in \mathcal{NP}$ in polynomieller Zeit auf π reduziert werden kann.

Bei den betrachteten Klassen taucht das ungelöste Problem auf, ob sich jedes Problem, das sich nichtdeterministisch in polynomieller Zeit lösen lässt, auch deterministisch in Polynomialzeit lösen lässt. Es gilt sicher die Inklusion $\mathcal{P} \subseteq \mathcal{P}$. In der Literatur geht man von der Vermutung $\mathcal{P} \neq \mathcal{NP}$ aus, die aber nicht bewiesen ist.

D Sensorik und Aktorik

Im Laufe der Arbeit werden die Begriffe Sensorik und Aktorik oft verwendet. Eine Definition kann hier gegeben werden.

Definition D.1 (Sensorik) *Die Sensorik eines Roboters misst Veränderungen des Roboters und der Umgebung des Roboters. Sensoren nehmen keinen Einfluss auf die Umgebung des Roboters.*

Definition D.2 (Aktorik) *Die Aktorik eines Roboters lässt den Roboter mit seiner Umgebung interagieren. Damit ist der Roboter in der Lage, seinen Standort oder sogar die Umwelt selber zu verändern.*

Die Sensorik überwacht also eine Einsatzumgebung und die Aktorik manipuliert die Einsatzumgebung.

Nachfolgend sollen die in dieser Arbeit verwendete Sensorik und Aktorik kurz vorgestellt werden, ohne allzusehr auf technische Details einzugehen.

D.1 Ultraschall-Sensorik

Navigation mittels Ultraschall wird in der Natur bei Delfinen und bei Fledermäusen praktiziert. In beiden Fällen ist die Lichtintensität im Lebensraum des jeweiligen Tieres sehr beschränkt, weshalb akustische Navigation sich durchgesetzt hat. Vor allem Fledermäuse haben die Sonartechnik (sound navigation and ranging) sehr weit entwickelt. Es gibt Unterarten, die mit schmalen sogenannten Beams oder Ultraschallimpulsen die Landschaft abtasten, andere verwenden breite Beams mit kurzen Impulsen.

Ein Ultraschallsensor versucht die Funktionsweise der Tiere nachzubilden in folgender weise: Ein Energieumwandler wird als Sender und Empfänger verwendet. Er besteht im wesentlichen aus einer beschichteten Goldfolie, die mit einer geriffelten Aluminiumwand einen Kondensator bildet. Wenn der Kondensator geladen wird, wird eine elektrostatische Kraft auf die Goldfolie ausgeübt, die das akustische Signal aussendet. Im Empfangsmodus bewirkt der reflektierte und einkommende Ultraschallimpuls einen Druck auf die Folie, welches die Kapazität des Kondensators ändert.

In der Regel wird alle 200 Millisekunden ein Ultraschallimpuls erzeugt. Es werden mehrere Frequenzen verwendet, da manche Materialien die akustische Energie absorbieren und kein Echo liefern.

Ultraschallsensoren haben den Vorteil, sehr billig zu sein. Jedoch benötigen diese Sensoren aufgrund der langsamen Schallgeschwindigkeit lange Laufzeiten (0.06 Sekunden für 10 Meter). Dies ist problematisch bei mobilen Robotern mit hohen Geschwindigkeiten. Aufgrund von Reflexionen können viele Fehler auftreten, wenn sich der Roboter in ungünstigen Positionen relativ zur Umgebung befindet. Außerdem sind Ultraschallsensoren sehr ungenau. Ein Ultraschallsensor wird folgendermaßen modelliert:

Definition D.3 (Ultraschall- oder Sonar-Sensor) *Ein* **Ultraschall-Sensor** $\Theta = (R, \theta, \beta)$ *ist gegeben durch eine Reichweite* $R = [d_{min}, d_{max}] \subset \mathbb{R}$ *und eine Funktion* $\theta : \{1, \cdots n\} \to [d_{min}, d_{max}]$ *mit einem endlichen Definitionsbereich. Dabei bezeichnet* n *die Anzahl der zur Verfügung stehenden Ultraschallsensoren, die Distanzwerte zwischen der minimalen Distanz* d_{min} *und der maximalen Distanz* d_{max} *annehmen können. Jeder Zahl aus* $\{1, \cdots, n\}$ *kann dabei ein fester Winkel aus dem Intervall* $[0, 2\pi]$ *mittels einer Funktion* β *zugeordnet werden, die den festgegebenen Abstrahlwinkel am Roboter widerspiegelt.*

D.2 Odometrie-Sensorik

Odometriesensoren sind interne Sensoren des mobilen Roboters und messen die Position des Roboters relativ zu einem lokalen Koordinatensystem. Diese Sensorik ist auch unter dem Namen Radencoder bekannt. Dabei werden die Radumdrehungen der einzelnen Räder des mobilen Roboters gemessen. Dafür steht eine unterschiedlich kodierte Codescheibe pro Rad zur Verfügung. Die Abtastung der Codescheibe geschieht durch einen Sensor.

Man unterscheidet dabei zwischen inkrementeller Kodierung und absoluten Codierung. Bei der inkrementellen Kodierung existiert für jede vollständig ausgeführte Umdrehung der Welle eine feste Anzahl von Impulsen. Die Impulszahl ist abhängig von der Unterteilung der verwendeten Codescheibe. Moderne Drehimpulsgeber kleiner Bauart haben bis zu 6.000 Segmente, aufwendige Drehimpulsgeber bis zu 72.0000 Impulse pro Umdrehung. Bei der absoluten Codierung wird die mechanische Position eindeutig einer bestimmten Codekombination zugeordnet. Diese ist z.B. durch 12 Bit codiert. Odometriewerte werden folgendermaßen modelliert.

Definition D.4 (Odometrie) *Die Konfiguration* $p = (x, y, \theta)^T$ *des Roboters relativ zu seinem lokalen Koordinatensystem zum Zeitpunkt* t *wird Odometrie zum Zeitpunkt* t *genannt.*

D.3 Laser-Sensorik

Wesentlich schneller als Schallwellen sind Lichtstrahlen, genauer: Laserstrahlen. Mit Hilfe eines ausgesendeten Lichtimpulses in einer genau spezifizierten Richtung wird dieser an einem Objekt in einiger Entfernung reflektiert und die Reflektion des Lasers und die genaue Zeit der Reflektion bestimmt. Aus dem Wert der Laufzeit des Lichts kann dann die genaue Entfernung des Objektes gemessen werden.

Laser-Entfernungssensoren werden heutzutage zum Beispiel zur Überwachung von Räumen eingesetzt. In der Robotik kommen sie häufig als Echtzeit-Abstandssensoren zum Einsatz. Aber auch zur Erkennung von Hindernissen, bei der Selbstlokalisation, Navigation, Objekterkennung werden sie (auch zusammenwirkend) eingesetzt. Sie dienen in der Industrie auch zur Vermessung von Fahrzeugen, Lageerkennung von Gütern, etc.

Laser-Entfernungsmesser sind wesentlich genauer als Ultraschall-Sensoren und berührungsfreie Sensoren. Sie liefern zuverlässige Abstandsdaten über große Bereiche durch hohe Reichweite mit hohen Tastzeiten und Geschwindigkeiten. Weiterer Vorteil ist die Tatsache, dass keine Objektausleuchtung notwendig ist.

Die Nachteile von Laser-Entfernungsmessern liegen darin, dass Reflexionen an glatten Objekten möglich sind und das Signal dann verrauscht wird. Insbesondere werden Glasflächen nicht erkannt. Zudem ist beim Einsatz von Lasern die Augensicherheit problematisch.

Man unterscheidet in der Literatur zwischen drei verschiedenen Verfahrensarten von Laser-Entfernungsmessern.

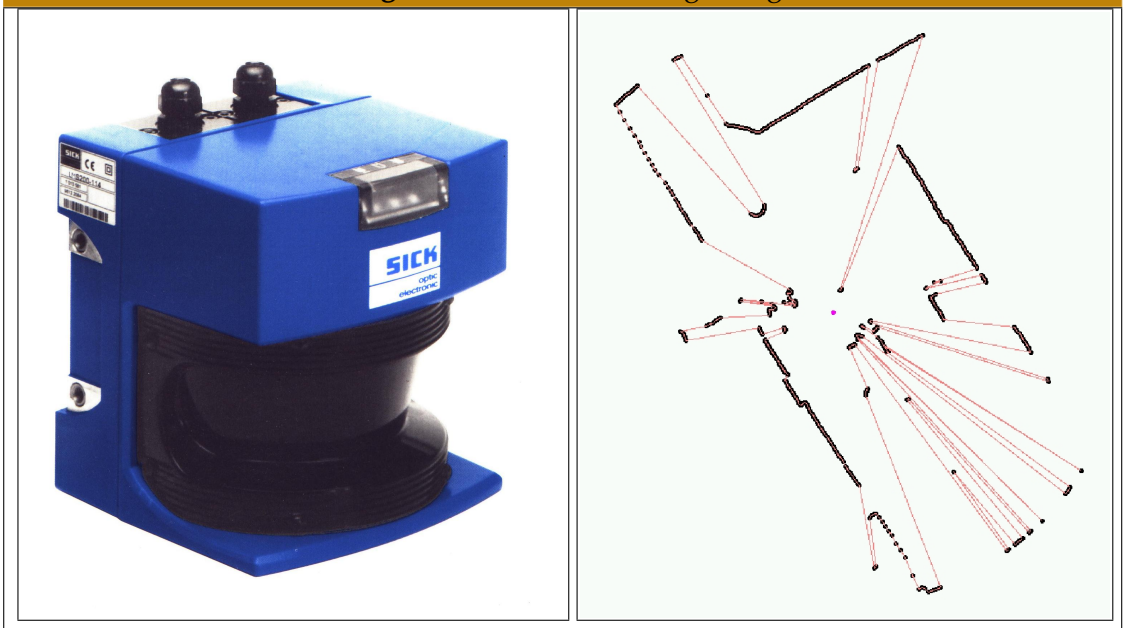
Laufzeitverfahren oder Time-Of-Flight (TOF)
Die Laufzeit eines kurzen Laser-Impulses wird gemessen.

Phasen-Verfahren
Die Phasenverschiebung einer kontinuierlichen Welle wird übertragen und die Phasendifferenz der reflektierten Welle mit der ausgesendeten verglichen.

Triangulations-Verfahren
Durch Triangulation wird der Abstand gemessen.

Der hier verwendete Laserscanner ist ein Modell des Typs LMS 200 der Firma SICK (siehe Abbildung 71, welches ein Laufzeitverfahren einsetzt. Der Laser-Entfernungsmesser kann so per Software konfiguriert werden, dass er mit einer Abtastrate von 0.25 bis 1.0 Grad einen Bereich von 100 Grad scannen kann oder aber mit einer Abtastrate von 0.5 oder 1.0 Grad einen Bereich von 180 Grad. Die Messwerte können in einem kontinuierlichen Strom ausgegeben werden oder auf Anfrage. Der in Betrieb befindliche Laser-Entfernungsmesser wird mit 38400 Baud mit einer Auflösung von 0.5 Grad und

Abbildung 71: SICK-LMS und zugehöriger Scan



Im linken Bild ist der SICK LMS 200 Sensor abgebildet. Im rechten Bild ist ein 360-Grad-Scan Scan zu sehen, welcher aus drei Scans zusammengestellt wurde.

einem Scanbereich von 180 Grad betrieben. Die Daten eines Laser-Scanners kann man folgendermaßen modellieren:

Definition D.5 ((Laser)-Scan) *Ein* **(180-Grad-)Scan** $\sum = (R, \sigma)$ *ist durch eine Reichweite* $R \in \mathbb{R}$ *und eine Funktion* $\sigma : \{\gamma_i \mid \gamma_i = \frac{i\pi}{n}; 1 \leq i \leq n\} \rightarrow [0, R]$ *mit endlichem Definitionsbereich gegeben. Dabei bezeichnet* $n \in \mathbb{N}$ *die Anzahl der Scanpunkte, die der Laser-Entfernungsmesser aufgenommen hat.*

D.4 Aktorik

In dieser Arbeit liegt das Hauptaugenmerk auf mobilen Robotern bzw. Robotern, die sich mit Rädern bewegen. Räder besitzen den Vorteil, dass die Radtechnologie weitgehend perfektioniert ist. Der Antrieb setzt jedoch eine ebene Fläche voraus. Unebenheiten in der Umgebung stellen größere Probleme dar. Durch die Radtechnologie besitzt das Fahrzeug und insbesondere die Sensorplattform eine große Stabilität. Der Einsatz von Rädern birgt jedoch einige Nachteile. Räder schränken die Manövrierbarkeit des Roboters zumeist ein, eine Drehung ist dann nur durch Vorwärtsbewegung möglich.

Man unterscheidet zwischen den zwei Antriebssystemen Differentialantrieb und

Synchro-Drive-Antrieb, andere Antriebsarten seien unberücksichtigt gelassen. Der Differentialantrieb besitzt zwei unabhängige Antriebsräder. Daneben existieren meist ein oder zwei Stützräder. Die Lenkung des Roboters geschieht durch die Geschwindigkeitsdifferenz der beiden Antriebsräder. Beispiele dazu sind ein Rollstuhl oder aber der Pioneer II-Roboter (siehe 72). Der Synchro-Drive-Antrieb zeichnet sich dadurch aus, dass die zumeist drei Räder einen synchronen Antrieb haben. Alle drei Räder sind zugleich Lenk- und Antriebsräder. Dadurch ist der Roboter in der Lage, auf der Stelle zu drehen. Die kinematische Modellierung beider Antriebe ist in [174] nachzulesen und wird hier übergangen.

D.5 Eingesetzte mobile Roboter

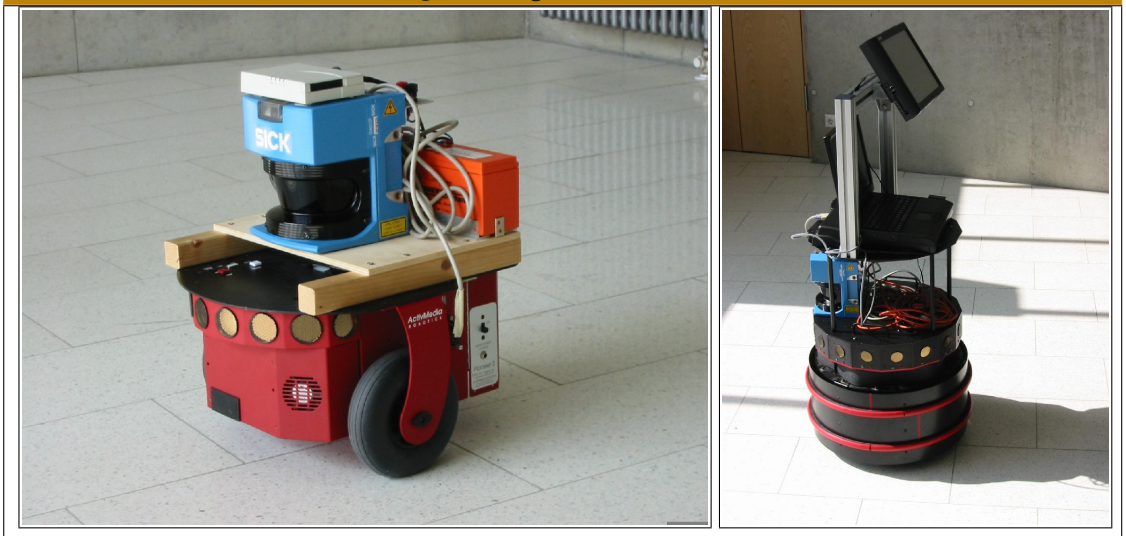
Im Rahmen dieser Arbeit sind zwei Roboter zum Einsatz gekommen, ein Pioneer- und ein Nomad-Roboter, wobei letztendlich die Experimente mit dem Pioneer-Roboter durchgeführt wurden (Abbildung 72).

Der Nomad 150 Roboter ist mit einem zusätzlichen 200 MHz PII On-Board Laptop ausgestattet. Die mobile Plattform besitzt 16 Sonar-Sensoren in ca. 50 cm Bodenhöhe. Weiter wurde er um einen Sick LMS 200 Indoor-Laserscanner als Sensor ergänzt. Als Antrieb fungiert ein Synchro-Antrieb. Zusätzlich besitzt der Nomad-Roboter einen Touch-Panel-PC als Eingabekonsole für Benutzereingaben und kann damit per Hand gesteuert werden. Die Kommunikation mit dem Roboter erlauben zwei Funkethernet-Karten.

Der Pioneer DX2 Roboter ist ausgestattet mit einem 200 MHz PII On-Board Rechner. Die mobile Plattform besitzt 8 Front-Sonar-Sensoren in ca. 30 cm Bodenhöhe. Dieser Roboter besitzt ebenfalls einen Sick LMS 200 Laserscanner als zusätzlichen Sensor. Der Roboter besitzt einen Differentialantrieb und kann über einen USB-Joystick manuell bedient werden. Der Roboter besitzt ansonsten keine Mensch-Maschine-Schnittstelle. Die Kommunikation mit dem mobilen Gerät geschieht über eine Funkethernet-Karte.

Beide Roboter wurden mit unterschiedlichen Steuersoftware-Systemen ausgeliefert, welche unter einem SuSE-Linux Betriebssystem eingesetzt werden.

Abbildung 72: Eingesetzte mobile Roboter



Im linken Bild ist der mobile Pioneer-Forschungsroboter abgebildet. Das rechte Bild zeigt den Nomad 150 Roboter mit einem speziellen Aufbau zur Mensch-Maschine-Kommunikation.

Literatur

[1] ABB. *ABB-IRB 140 Industrie-Roboter Spezifikation*, 2003.
`http://www.abb.com`

[2] ActivMedia Inc. *ActivMedia Robotics Basic-Suite*, 2002.
`http://www.amigobot.com/amigo/amigsfwe.html`

[3] ActivMedia Robotics Inc. *Aria-Manual*, 2002.

[4] J.K. Aggarwal und Raj Talluri. Position Estimation Techniques for an Autonomous Mobile Robot – A Review. In *Handbook of Pattern Recognition & Computer Vision*, Kapitel 4.4, Seiten 769–801. World Scientific, 1993.

[5] A. S. Aguado, M. E. Montiel und M. S. Nixon. Extracting arbitrary geometric primitives represented by Fourier descriptors. In *Proc. International Conference on Pattern Recognition ICPR '96*, Seiten 547–551. IEEE, 1996.

[6] A. S. Aguado, M. E. Montiel und M. S. Nixon. Parameterising arbitrary shapes via Fourier descriptors for evidence-gathering extraction. *Computer Vision and Image Understanding CVIU*, 69[2]:202–221, 1998.

[7] Alireza Ahmadyfard und J. Kittler. Region-Based Object Recognition: Pruning Multiple Representations and Hypotheses. In *11th British Machine Vision Conference*, 2000.
`http://www.bmva.ac.uk/bmvc/2000/papers/p75.pdf`

[8] S. Albers, K. Kursawe und S. Schuierer. Exploring unknown environments with obstacles. In *10th ACM-SIAM Symp. on Discrete Algorithms, Baltimore, Maryland*, Seiten 842–843, 1999.

[9] Ernst Althaus, Kurt Mehlhorn, Stefan Näher und Stefan Schirra. Experiments on Curve Reconstruction. In *Second Workshop on Algorithm Engineering and Experiments, ALENEX*, 2000.

[10] Nina Amenta und Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1999.

[11] Nina Amenta, Marshall Bern und David Eppstein. The Crust and the β-Skeleton: Combinatorial Curve Reconstruction. In *Graphical Models and Image Processing*, Band 60, 1998.
`http://www.cs.utexas.edu/users/amenta/pubs/crust.ps.gz`

[12] J. R. Andrews und N. Hogan. Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator. In *Control of Manufacturing Processes and Robotics Systems, ASME Boston*, Seiten 243–251, 1983.

[13] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan und Mario Szegedy. Proof Verification and Hardness of Approximation Problems. In *IEEE Symposium on Foundations of Computer Science*, Seiten 14–23, 1992.
`http://citeseer.nj.nec.com/lund92proof.html`

[14] K.O. Arras, N. Tomatis und R. Siegwart. Multisensor On-the-Fly Localization Using Laser and Vision. In *Proc. of IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, 2000.
`http://dmtwww.epfl.ch/isr/asl/publications.html#arras`

[15] Alexander Asteroth und Christel Baier. *Theoretische Informatik*. Pearson Education, 2002.

[16] Tim Bailey und Eduardo Nebot. Localization in Large-Scale Environments. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2000.

[17] Dana H. Ballard und Christopher M. Brown. *Computer Vision*. Prentice Hall, 1982.

[18] Antonio Banderas, Cristina Urdiales und Francisco Sandoval. An Hierarchical approach to grid-based and topological maps integration for autonomous indoor navigation. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Seiten 883–888, 2001.

[19] Michael Beigl, Hans-Werner Gellersen und Albrecht Schmidt. MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. *Computer Networks, Special Issue on Pervasive Computing*, 35[4]:401–409, März 2001.
`http://ubicomp.teco.edu/publ.html`

[20] Ola Bentsson und Albert-Jan Baerveldt. Localization in Changing Environments - Estinamtion of a Covariance Matrix for the IDC Algorithm. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Seiten 1931–1937, 2001.

[21] Gerald Bieber und Malte Korten. User tracking by sensorfusion for situation aware systems. In *ACRS 2001, Singapur*, 2001.

[22] R. Bischoff und V. Graefe. Demonstrating the Humanoid Robot HERMESat an Exhibition: A Long-Term Dependability Test. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '02; Workshop on Robots at Exhibitions. Lausanne*, 2002.

[23] B. Bolles, H. Bunke, H. Christensen und H. Noltemeier (Hrsg.). *Modelling and Planning for Sensor-Based Intelligent Robot Systems*. Nummer 224 in Machine Perception Artificial Intelligence. Elesevier, Februar 1999.

[24] Robert C. Bolles und Ronald A. Cain. Recognizing and Locating Partially Visible Objects: the Local-Feature-Focus-Method. *International Journal of Robotics Research*, 1[3]:57–82, 1982.

[25] J. Borenstein, H. R. Everett und L. Feng. *"Where am I?" – Systems and Methods for Mobile Robot Positioning*. The University of Michigan, März 1996.

`http://www-personal.engin.umich.edu/~johannb/position.htm`

[26] J. Borenstein und Y. Koren. Real-time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 194[5]:1179–1187, 1989.

[27] J. Borenstein und Y. Koren. The Vector Field Histogram -Fast Obstacle Avoidance for Mobile Robots. *IEEE Journal of Robotics and Automation*, 7[3]:278–288, 1991.

[28] Johann Borenstein. Real-time Obstacle Avoidance for Fast Mobile Robots. In *IEEE Transactions on Robotics and Automation*, 1991.

[29] Thomas M. Breuel. Geometric Matching in Computer Vision–Algorithms and Open Problems. IDIAP Memo 93-07, IDIAP, Juni 1993.
`http://www.idiap.ch/publications/breuel-93.03.bib.abs.html`

[30] Oliver Brock und Oussama Khatib. Elastic Strips: Real-Time Path Modification for Mobile Manipulators. In *Eighths International Symposium of Robotics Research, Hayama, Japan*, 1998.

[31] Oliver Brock und Oussama Khatib. High-Speed Navigation Using the Global Dynamic Window Approach. In *IEEE International Conference on Robotics and Automation*, 1999.

[32] R. Brooks. A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[33] R. Brooks. Intelligence Without Representation. *Artificial Intelligence Journal*, 47:139–159, 1991.

[34] Martin Buck. Effiziente Strategien zur Hypothesenreduktion bei der Lokalisation und Navigation autonomer Roboter. Diplomarbeit, Universität Würzburg, Juli 1999.

[35] Martin Buck, Boris Kluge, Hartmut Noltemeier und Dirk Schäfer. RoLoPro - Simulationssoftware für die Selbstlokalisation eines autonomen mobilen Roboters. In *Autonome Mobile Systeme, AMS 99, München*, Seiten 118–127, 1999.

[36] Martin Buck, Hartmut Noltemeier und Dirk Schäfer. Practical Strategies for Hypotheses Elimination on the Self-Localization Problem. Technischer Bericht 236, Informatik, Universität Würzburg, August 1999.

[37] Horst Bunke, Takeo Kanade und Hartmut Noltemeier (Hrsg.). *Modelling and Planning for Sensor Based Intelligent Robot Systems*. World Scientific, 1995.

[38] W. Burgard, D. Fox und Daniel Henning. Fast Grid-based Position Tracking for Mobile Robots, 1997.

[39] J. Riemann C. Lewis. Task-Centered User Interface Design: A practical introduction. In *Task-Centered User Interface Design*, 1993.
`http://hcibib.org/tcuid/`

[40] Jose M. Canas und María C. García-Alegre. Real time EM segmentation of occupancy grid for robots navigation. In *Proceedings of IJCAI 1999 Workshop in Ulm*, 1999.

[41] J. F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award: 1987. The MIT Press, 1988.

[42] John F. Canny und Ming C. Lin. An Oppertunistic Global Path Planner. *Algorithmica*, 10[2 - 4]:102 – 120, 1993.
`http://www.cs.unc.edu/~lin/papers.html`

[43] Todd Anthony Cass. Robust 2D Model-Based Object Recognition. Diplomarbeit, Massachusetts Institute of Technology, Mai 1988.
`ftp://publications.ai.mit.edu/ai-publications/1000-1499/AITR-1132.ps`

[44] Todd Anthony Cass. Feature Matching for Object Classification in the Presence of Uncertainty. *A.I. Memo No. 1133, Massachusetts Institute of Technology, Artifical Intelligence Laboratory*, 1990.

[45] C. H. Chen, L.F. Pau und P.S. P. Wang. *Handbook of Pattern Recognition & Computer Vision*. World Scientific, 1993.

[46] Wei-Pang Chin und Simeon Ntafos. Optimum Watchman Routes in Simple Polygons. In *Proceedings of the 2nt ACE Symposium on Computational Geometry*, Seiten 24–33, 1986.

[47] Howie Choset und Joel Burdick. Sensor based Planning and Nonsmooth Analysis. In *IEEE/ICAR, San Diego CA*, 1994.

[48] Howie Choset und Joel Burdick. Sensor based Planning, Part II: Incremental Construction of the Generalized Voronoi Graph. In *IEEE/ICAR, Nagoya Japan*, 1995.

[49] Howie Choset, Ilhan Konukseven und Alfred Rizzi. Sensor based Planning: A Control Law for Generating the Generalized Voronoi Graph. In *IEEE/ICAR, Montery, CA*, 1997.

[50] Herik I. Christensen. Summary of EU-EURON Brainstorming Meeting on Robotic, 2002, Brüssel, Belgien, 2002.
`htpp://www.euron.org`

[51] Peter I. Corke. Safety of advanced robots in human environments A discussion paper for IARP, 1999.
`http://www.international-robotics.org/wg/dependability/safety.pdf`

[52] Ingemar J. Cox. Blanche: Position Estimation for an Autonomous Robot Vehicle. In *Autonomous robot vehicles*, Seiten 221–228. Springer Verlag, 1990.

[53] J.L. Crowley. World Modelling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging. In *IEEE International Conference on Robotics and Automation*, 1989.

[54] T. Darrell, G. Gordon, J. Woodfill und V. Lesser. Integrated person tracking using stereo, color, and pattern detection. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*, Seiten 44–49, 1998.

[55] Tom DeMarco und Timothy Lister. *Mit Risikomanagement Projekte zum Erfolg führen*. Carl Hanser Verlag, 2003.

[56] Spektrum der Wissenschaft. Dossier: Roboter erobern den Alltag. Spektrum Verlag, 1998.

[57] T. K. Dey und P. Kumar. A simple provable algorithm for curve reconstruction. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 99)*, Seiten 893–894, 1999.

[58] T. K. Dey, K. Mehlhorn und E. Ramos. Curve reconstruction: connecting dosts with good reason. In *Computational Geom. Theory and Application*, Band 15, Seiten 229–244, 2000.

[59] Svetlana Domnitcheva. Location Modeling: State of the Art and Challanges. In *Workshop on Location Modeling for Ubiquitous Computing, Atlanta, Georgia*, 2001.

[60] Leo Dorst, Indur Mandhyan und Karen Trovato. The Geometrical Representation of Path Planning Problems. *Robotics and Autonomous Systems*, 7:181 – 195, 1991. `http://carol.wins.uva.nl/~leo/papers/ias.html`

[61] Richard O. Duda und Peter E. Hart. Use of Hough Transformation To Detect Lines and Curves in Pictures. *Communications of the ACM*, 15[1], 1972.

[62] Gregory Dudek, Kathleen Romanik und Sue Whitesides. Localizing a Robot with Minimum Travel. In *Proceedings of the 6th Annual ACM–SIAM Symposium on Discrete Algorithms*, Seiten 437–446, 1995.
`http://www.cim.mcgill.ca/~dudek/`

[63] Electrolux Corp. *Electrolux Trilobite*, 2003.
`http://www.electrolux.com/node613.asp`

[64] Alberto Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *IEEE Computer*, 22[6]:46–57, 1989.

[65] EMT Ingeniergesellschaft, Penzberg. *Spezifikation der Luna X 2000*, 2002.
`http://www.emt-penzberg.de/de/index.htm`

[66] H. R. Everett. *Sensors for mobile Robots - Theory and Application*. AK Peters, 1995.

[67] Paolo Fiorini. *Robot Motion Planning among Moving Obstacles*. Dissertation, Universität Californien, Los Angeles, 1995.

[68] D. Fox, W. Burgard, F. Dellaert und S. Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proc. of the National Conference on Artificial Intelligence*, 1999.
`http://www.informatik.uni-freiburg.de/~burgard/abstracts/sampling-aaa%i-99.abstract.html`

[69] Dieter Fox. *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. Dissertation, Universität Bonn, 1998.

[70] Dieter Fox, Wolfram Burgard, Hannes Kruppa und Sebastian Thrun. A Monte Carlo Algorithm for Multi-Robot Localization. *Künstliche Intelligenz*, Seiten 255–266, 1999.
`citeseer.nj.nec.com/fox99collaborative.html`

[71] Dieter Fox, Wolfram Burgard und Sebastian Thrun. Controlling Synchro-Drive Robots with the Dynamic Window Approach to Collision Avoidance. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1996.

[72] Dieter Fox, Wolfram Burgard und Sebastian Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics and Automation Magazine*, 4[1], 1997.
`http://www.informatik.uni-freiburg.de`

[73] R. A. Frederick. *Unmanned Air/Ground Vehicle (UAGV)*. Universität Utah, 2001.
`http://www.eb.uah.edu/ipt/files/2001`

[74] Friendly Robotics. *Friendly Robotics Robomower RL800*, 2003.
`http://www.onrobo.com/reviews/At_Home/Mowers/on00rl5001mofdr/`

[75] Ch. Galambos, J. Kittler und J. Matas. Progressive Probabilistic Hough Transform. In M. S. Nixon (Hrsg.), *Proc British Machine Vision Conference BMVC98*, Seiten 256–265, 1998.

[76] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Entwurfsmuster*. Addison-Wesley, 1996.

[77] H.-W. Gellersen, A. Schmidt und M. Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. *Journal on Mobile Networks and Applications, Special Issue on Mobility of Systems, Users, Data and Computing in Mobile Networks and Applications (MONET)*, 7[5]:341–351, Oktober 2002.
`http://ubicomp.teco.edu/publ.html`

[78] Stefan Gillert. Human Machine Interface für einen autonomen mobilen Auskunfts- und Begleitroboter. Diplomarbeit, Informatik, Universität Würzburg, 2002.

[79] J. Gonzales, A. Ollero und A. Reina. Map Building for a Mobile Robot equipped with a 2D Laser Rangefinder. In *IEEE International Conference on Robotics and Automation*, 1994.

[80] Hector H. Gonzalez-Banos und Jean-Claude Latombe. Navigation Strategies for Exploring Indoor Environments. In *International Journal on Robotics Research*, 2001.
`http://robotics.stanford.edu/~latombe/pub.html`

[81] B. Graf, M. Hans, J. Kubacki und R.D Schraft. Robotic Home Assistant Care-O-bot II. In *Second Joint Meeting of the IEEE Engineering in Medicine and Biology Society and the Biomedical Engineering Society, Houston, Texas, USA,*, 2002.

[82] B. Graf, E. Helms, V. Lakshmana, B. Rohrmoser und R.D. Schraft. Anthropomorphic Robot Assistants - Giving the Human a Helping Hand. In *Second IARP IEEE/RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, Toulouse, Frankreich*, Seiten 20–24, 2002.

[83] B. Graf, Jose Manuel Hostalet Wandosell und Chrstoph Schaeffer. Flexible Path Planning for Nonholonomic Mobile Robots. In *Proc. of 3rd European Workshop on Advanced Mobile Robots (EUROBOT)*, 2001.

[84] W. E. L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, 1990.

[85] W. E. L. Grimson, Daniel P. Huttenlocher und David W. Jacobs. A Study of Affine Matching With Bounded Sensor Error. In G. Sandini (Hrsg.), *Lecture Notes in Computer Science*, Band 588, Seiten 291–306. Springer, 1992.

[86] Mikell P. Groover. *Fundamentals of Modern Manufacturing*. John Wiley & Sons, New York, 2000.

[87] Leonidas J. Guibas, Rajeev Motwani und Prabhakar Raghavan. The Robot Localization Problem in Two Dimensions. In *Proceedings of the 3rd Annual ACM–SIAM Symposium on Discrete Algorithms*, Seiten 259–268, 1992.

[88] Leonidas J. Guibas, Rajeev Motwani und Prabhakar Raghavan. The Robot Localization Problem. *Algorithmics Foundation of Robotics*, Seiten 269–282, 1995. `http://theory.stanford.edu/people/motwani/papers.html`

[89] Leonidas J. Guibas, Rajeev Motwani und Prabhakar Raghavan. The Robot Localization Problem. *SIAM Journal on Computing*, 26[4]:1120–1138, August 1997.

[90] J-S. Gutmann, W. Hatzack, I. Herrmann, B. Nebel, F. Rittinger, A. Topor, T. Weigel und B. Welsch. The CS Freiburg Team. In *RoboCup 98*, 1998. `http://www.ki.informatik.hu-berlin.de/AKRoboCup/RoboCupWS-KI98/theCSF%reiburgTeam.ps.gz`

[91] Jens-Steffen Gutmann. Vergleich von Algorithmen zur Selbstlokalisation eines mobilen Roboters. Diplomarbeit, Universität Ulm, April 1996.

[92] Jens-Steffen Gutmann. *Robuste Navigation autonomer mobiler Roboter*. Dissertation, Universität Freiburg, 2000.

[93] Jens-Steffen Gutmann und D. Fox. An Experimental Comparison of Localization Methods Continued. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[94] Jens-Steffen Gutmann und K. Konolige. Incremental Mapping of Large Cyclic Environments. In *International Symposium on Computational Intelligence in Robotics and Automation (CIRA'99), Monterey*, 1999.

[95] Jens-Steffen Gutmann und Bernhard Nebel. Navigation mobiler Roboter mit Laserscans. In *Autonome Mobile Systeme 1997 (AMS'97)*, Seiten 36–47. Springer, 1997.

[96] Jens-Steffen Gutmann, Bernhard Nebel und Christian Reetz. CS Freiburg: Architektur und Aktionsauswahl im Roboterfussball. In *Autonome Mobile Systeme 2000 (AMS'2000)*. Springer, 2000.

[97] Jens-Steffen Gutmann und Christian Schlegel. AMOS: Comparision of Scan Matching Approaches for Self-Localization in Indoor Environments. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots*, 1996.

[98] Mikael Hammar und Bengt J. Nielsson. Concerning the Time Bounds of Existing Shortest Watchman Route Algorithms. *Fundamentals of Computation Theory*, Seiten 210–221, 1997.

[99] Robert Hanek, Thorsten Schmitt, Sebastian Buck und Michael Beetz. Fast Image-based Object Localization in Natural Scenes. In *IEEE International Conference on Robots and Systems (IROS)*, 2002.
`http://www9.in.tum.de/publications/2002/IROS-2002-Hanek-etal.pdf`

[100] P. E. Hart, N.J. Nilsson und B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, Oktober 1968.

[101] E. Helms, R.D. Schraft und M Haegele. rob@work: Robot Assistant in Industrial Environments. In *11th IEEE Int. Workshop on Robot and Human interactive Communication, ROMAN2002, Berlin, Germany*, 2002.

[102] Stefan Hieß und Jürgen Roder. Markov-basierte Lokalisation. Projektpraktikum 2001, Informatik, Universität Würzburg, 2001.

[103] Frank Hoffmann, Christian Icking, Rolf Klein und Klaus Kriegel. The Polygon Exploration Problem I: A Competitive Strategy. Technischer Bericht 241, Fern-Universität Hagen, 1998.

[104] Frank Hoffmann, Christian Icking, Rolf Klein und Klaus Kriegel. The Polygon Exploration Problem II: The Angle Hull. Technischer Bericht 245, Fern-Universität Hagen, 1998.

[105] D. P. Huttenlocher und S. Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5[2]:195–212, 1990.

[106] A. Isidori H.W. Knobloch und D. Flockerzi. *Topics in Control Theory*. Birkhäuser, Basel, 1993.

[107] J. Illingworth und J. Kittler. A Survey of the Hough Transform. *Computer Vision, Graphics and Image Processing*, 44:87–116, 1988.

[108] Giovanni Indiveri. On the motion control of a nonholonomic soccer playing robot. In *5th International Symposium on RoboCup 2001, Seattle, USA*, 2001.
`http://www.ais.fraunhofer.de/BE/2002/indiveri2002_1.pdf`

[109] J.D. Tardos J. A. Castellanos und G. Schmidt. Building a Global Map of the Environment of a mobile Robot: The Importance of Correlations. In *IEEE International Conference on Robotics and Automation*, 1997.

[110] A. Kobsa J. Fink und A. Nill. User-oriented Adaptivity and Adaptability in the AVANTI-Project. In *Proceedings of Conference Designing for the Web: Empirical Studies*, 1996.

[111] A. Kobsa J. Fink und A. Nill. Adaptable and Adaptive Information Provision for All Users, Including Disabled and Elderly People. In *The New Review of Hypermedia and Multimedia 4*, Seiten 163–188, 1998.

[112] R. Molich J. Nielsen. Heurstic evaluation of user interfaces. In *Proceedings of CHI'90: Empowering People*, 1990.

[113] Patric Jensfeld. *Approaches to Mobile Robot Localization in Indoor Environments*. Dissertation, Universität Stockholm, Schweden, 2001.

[114] O. Karch, H. Noltemeier und Th. Wahl. Location and Robotics. In *Location Theory: Applications in Real World Problems*, 2001.

[115] Oliver Karch, Hartmut Noltemeier und Thomas Wahl. Using Polygon Distances for Localization. Technischer Bericht SS-005/4, Universität Würzburg, 1998.

[116] Oliver Karch und Thomas Wahl. Relocalization – Theory and Practice. In Hartmut Noltemeier und Kokichi Sugihara (Hrsg.), *Special Issue of Discrete Applied Mathematics on Computational Geometry*, Band 93, Nr. 1. Elsevier, 1999.
`http://www-info1.informatik.uni-wuerzburg.de/publications/karch/publi%cations.html`

[117] Lydia E. Kavraki, Mihail N. Kolountzakis und Jean-Claude Latobe. Analysis of Probabilistic Roadmaps for Path Planning. *IEEE Transactions on Robotics and Automation*, 14[1]:166–171, 1998.
`http://robotics.stanford.edu/~latombe/pub.html`

[118] Lydia E. Kavraki und Jean-Claude Latobe. Probabilistic Roadmaps for Robot Path Planning. In *Practical Motion Planning in Robotics: Current Approaches and Future Dicections*, Seiten 33–53. John Wiley, 1998.
`http://robotics.stanford.edu/~latombe/pub.html`

[119] Lydia E. Kavraki, Jean-Claude Latobe, Rajeev Motwani und Prabhakar Raghavan. Randomized Query Processing in Robot Path Planning. *Journal of Computer and System Science*, 57[1]:50–60, 1998.
`http://robotics.stanford.edu/~latombe/pub.html`

[120] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *IEEE International Conference on Robotics and Automation, St. Lois, USA*, Seiten 500–505, 1985.

[121] Rolf Klein. *Algorithmische Geometrie*. Addison-Wesley, 1997.

[122] Boris Kluge. Lokale Features – Erkennung und Matching in Lokalisationsszenarien. Diplomarbeit, Universität Würzburg, Juni 1999.

[123] Boris Kluge. *Navigation in dynamischen Umgebungen*. Dissertation, FAW Ulm, 2003.

[124] Boris Kluge, Hartmut Noltemeier und Dirk Schäfer. Feature-based Localization of an Autonomous Mobile Robot: An Experimetal Case Study. In *Proc. of the International Conference on Intelligent Autonomous Systems (IAS6) Venice, Italy*, 2000.

[125] Bob Kohout. Challenges in Real-Time Obstacle Avoidance. In *AAAI Spring Symposium on Real-Time Autonomous Systems, Palo Alto, Ca*, 2000.

[126] K. Konolige und K. Chou. Markov Localization using Coorelation. In *International Joint Conference on Artificial Intelligence Stockholm, Schweden*, 1999.

[127] Sven Oliver Krumke. Graphentheoretische Konzepte und Algorithmen. Vorlesungsskript, Universität Würzburg, Wintersemester 1997/1998, März 1998. Skript.

[128] E. Kruse, R. Gutsche und F. Wahl. Acquisition of Obstacle Motion Patterns to Improve Mobile Robot Motion Planning. *Advanced Robotics*, 12[5]:565–578, 1998.

[129] Kuka-Roboter. *KR30 L15/2 Industrie-Roboter Spezifikation*. Kuka-Roboter, 2003. `http://www.kuka-roboter.de`

[130] Y. Kurozumi und W.A. Davis. Polygonal Approximation by the Minimax Method. *Computer Graphics and Image Processing*, 19:248–264, 1982.

[131] Jean-Claude Latobe. Motion Planning: A Journey of Robots, Molecules, Digital Actors, and other Artifacts. *Int. J. of Robotics Research, Special Issue on Robotics at the Millennium – Part I*, Seiten 1119–1128, November 1999.

[132] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[133] S. L. Laubach und J. W. Burdick. An Autonomous Sensor-Based Path-Planner for Planetary Microrovers. In *IEEE International Conference in Robotics and Automation*, 1999.
`http://robby.caltech.edu/papers/rover.icra99.pdf`

[134] Sharon Laubach, Clark Olson, Joel Burdick und Samad Hayati. Long Range Navigation for Mars Rovers Using Sensor-Based Path Planning and Visual Localization. In *5th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 1999.
`http://telerobotics.jpl.nasa.gov/people/olson/papers.html`

[135] Gisbert Lawitzky. SINAS, A Navigation System for Service Robots. In *Robotics 2000*, 2000.

[136] V. F. Leavers. Which Hough Transform? *Computer Vision, Graphics and Image Processing*, 58[2]:250–264, September 1993.

[137] David Lee. Quantitative Evaluation of the Exploration Strategies of a Mobile Robot. *International Journal of Robotics Research*, 16[4]:413–447, 1997.

[138] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Xuan Ping und Shelley ZQ Zhang. The Intelligent Home Testbed. In *Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, Seattle, Januar 1999.
`http://mas.cs.umass.edu/pub/paper_detail.php/134`

[139] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan und Shelly XQ Zhang. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, USA, Januar 1999.
`http://mas.cs.umass.edu/pub/paper_detail.php/120`

[140] F. Lorenz. *Lineare Algebra I*. Spektrum Akademischer Verlag, 1996.

[141] Feng Lu. *Shape Registration using optimization for mobile robot navigation*. Dissertation, Department of Computer Science, Universität Toronto, Kanada, 1995.

[142] Feng Lu und Evangelos Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4:333–349, 1997.

[143] Feng Lu und Evangelos Milios. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1997.

[144] Vladirmir Lumelsky und Alexander Stepanov. Path-Planning Strategies for a Point Mobile Automation Moving Amidst Unknown Obstacles of Arbitrary Shape. *Algorithmica*, 2[4]:403 – 430, 1987.

[145] Matthias Maar. Modellierung und Generierung einer Wissensbasis für ein mobiles Auskunftssystem. Diplomarbeit, Informatik, Universität Würzburg, 2002.

[146] J. Maddox. Smart Navigation Sensors for Automatic Guided Vehicles, 1994.

[147] C. Magerkurth und P. Tandler. Interactive Walls and Handheld Devices-Applications for a Smart Environment, 2002.

[148] Peter S. Maybeck. The Kalman Filter: An Introduction to Concepts.

[149] Florian Mayer. Generierung einer assoziativen Wissensbasis. Studienarbeit, Informatik, Universität Würzburg, 2002.

[150] Kurt Mehlhorn, Stefan Näher, Michael Seel und Christian Uhrig. The LEDA User Manual Version 3.6, 1997.

[151] Michigan State University. *DRS: Integrated Hybrid Software Framework for Autonomous Mobile Robot*, 2003.
`http://www.egr.msu.edu`

[152] NASA. Mars Climate Orbiter - Official Web Site, 1999.
`http://mars.jpl.nasa.gov/msp98/orbiter/`

[153] Stefan Näher, Michael Seel und Christian Uhrig. The LEDA User Manual, 2000.

[154] J. Nielsen. Heuristic evaluation. In *Usability Inspection Methods*. John Wiley & Sons, 1994.

[155] N. J. Nilson. Shakey the Robot. Technischer Bericht TN 323, SRI International, Menlo Park, California, 1984.

[156] N.J. Nilson. A Mobile Automaton: An Application of Artificial Intelligence Techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Seiten 509–520, 1969.

[157] P. Nordin und M. Nordahl. An Evolutionary Architecture For A Humanoid Robot. In *4th International Symposium on Artificial Life and Robotics*, 2002.

[158] E. Colle O. Ait Aider, P. Hoppenot. A Model to Image Straight Line Matching Method for Vision-Based Indoor Mobile Robot Self-Location. In *IEEE International Conference on Robots and Systems (IROS)*, 2002.
`http://lsc.cemif.univ-evry.fr:8080/~hoppenot/recherche/publications/a%rticles.pdf/conf2002iros.pdf`

[159] Clark F. Olson. Probabilistic Indexing for Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17[5]:518–522, Mai 1995.
`http://robotics.jpl.nasa.gov/people/olson/papers.html`

[160] Clark F. Olson. Efficient Pose Clustering Using a Randomized Algorithm. *International Journal of Computer Vision*, 23[2]:131–147, Juni 1997.
`http://robotics.jpl.nasa.gov/people/olson/papers.html`

[161] Clark F. Olson. Mobile Robot Self-Localization by Iconic Matching of Range Maps. In *8th International Conference on Advanced Robotics*, Seiten 447–452, 1997.
`http://telerobotics.jpl.nasa.gov/people/olson/papers.html`

[162] Joseph O´Rourke. *Art Gallery Theorems and Algorithms*. Oxford Press, 1987.

[163] Simon Osiander. 2-D Kartierung für autonome mobile Roboter und Auswirkungen auf die Lokalisationsgüte. Diplomarbeit, Universität Würzburg, 2002.

[164] Xavier Pennec. Toward a Generic Framework for Recognition Based on Uncertain Geometric Features. *Videre: Journal of Computer Vision Research*, 1[2]:58–87, 1998.
`http://mitpress.mit.edu/journal-issue-abstracts.tcl?issn=10892788&vol%ume=1&issue=2`

[165] Jarno Peschier. Polygon Simplification: An Overview. *via WWW*, 1994.
`http://www.jarno.demon.nl/polygon.html`

[166] Lars Petersson, David Austin und Henrik Christensen. DCA: A Distributed Control Architecture for Robotics. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Seiten 2361–2368, 2001.

[167] Arthur R. Pope. Model-Based Object Recognition–A Survey. Technischer Bericht 94–04, University of British Columbia Department of Computer Science, Januar 1994.
`http://www.cs.ubc.ca/spider/pope/home.html`

[168] Arthur R. Pope und David G. Lowe. Modeling Positional Uncertainity in Object Recognition. Technischer Bericht 94–32, University of British Columbia Department of Computer Science, November 1994.
`http://www.cs.ubc.ca/spider/pope/home.html`

[169] E. Prassler, D. Bank und B. Kluge. Motion Coordination between a Human and a Mobile Robot. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[170] Sean Quinlan und Oussama Khatib. Towards Real-Time Execution of Motion Tasks. In *Experimental Robotics 2, ed. R. Chatila und G. Hirzinger. Springer-Verlag*, 1991.

[171] Sean Quinlan und Oussama Khatib. Elastic Bands: Connecting Path Planning and Control. In *IEEE International Conference on Robotics and Automation*, Seiten 28–35, 1993.

[172] J.H. Reif. Complexity of the movers problem and generalizations. In *IEEE 20th Annual Symposium on Foundations of Computing Science*, Seiten 421–427, 1979.

[173] Antonio Reina und Javier Gonzalez. A two-stage mobile robot localization method by overlapping segment-based maps. *Robotics and Autonomous Systems*, 31[4]:213–225, 2000.

[174] Maria Isabel Ribeiro und Pedro Lima. Kinematics Models of Mobile Robots, 2002.
`www.isr.ist.utl.pt/~mir/cadeiras/robmovel/Kinematics.pdf`

[175] Philip Robinson und Michael Beigl. Trust Context Spaces: An Infrastructure for Pervasive Security in Context-Aware Environments. In *First International Conference on Security in Pervasive Computing, Boppard*, März 2003.
`http://ubicomp.teco.edu/publ.html`

[176] RoboCup. *RoboCup Junior*. RoboCup Initiative, 2003.
`http://www.robocup.org`

[177] Jürgen Roder. Recovery-Methoden bei der Echtzeit-Kartierung in Dynamischen Umgebungen. Diplomarbeit, Informatik, Universität Würzburg, 2003.

[178] Leonardo Romero, Eduardo Morales und Enrique Sucar. Learning Probabilistic Grid-Based Maps for Indoor Mobile Robots Using Ultrasonic and Laser Range Sensors. In *Lecture Notes in Computer Science*, Band 1793, Seiten 158–169, 2000.

[179] Jan Rosell, Raul Suarez und Luis Basanez. Path Validation in Constrained Motion with Uncertainty. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Seiten 2270–2274, 2001.

[180] Julio K. Rosenblatt. Utility Fusion: Map-Based Planning in a Behaviour-Based System. In *In Proceedings of FSR '97 International Conference on Field and Service Robotics*, 1997.

[181] E. Ross. Intelligent User Interfaces: Survey and Research Directions. Technischer Bericht CSTR-00-004, Department of Computer Science, Universität Bristol, März 2000.

[182] Neil C. Rowe und Robert S. Alexander. Finding Optimal-Path Maps for Path Planning Across Weighted Regions. Technischer Bericht 0, U. S. Naval Postgraduate School, Januar 1997.

[183] Byeong-Soon Ryu und Hyun S. Yang. An enhanced topological map for efficient and reliable mobile robot navigation with imprecise sensors. *Robotics and Computer-Integrated Manufacturing*, 14:185–197, 1998.

[184] Dirk Schäfer. Wissensbasierte Systeme - Einführung in die Robotik. Vorlesungsskript Universität Würzburg, Wintersemester 2000/2001, 2001.

[185] Dirk Schäfer. Wissensbasierte Systeme: Schwerpunkt Mobile Robotik. Vorlesungsskript Universität Würzburg, Wintersemester 2002/2003, 2003.

[186] Dirk Schäfer, Martin Buck, Boris Kluge, Thomas Heilmann und Stefan Hieß. RoLoPro - Open Source Software for Mobile Robot Localization. In *4th European Workshop on Advanced Mobile Robots, Lund Sweden*, Seiten 185–192, 2001.

[187] Dirk Schäfer und Stefan Hieß. Combining Exploration and Feature-Matching for Solving the Global Self-Localization Problem. In *1st European Conference on Mobile Robots (ECMR), Warschau*, 2003.

[188] Dirk Schäfer, Thomas Heilmann und Hartmut Noltemeier. Adaptive feature-basierte Selbstlokalisation autonomer mobiler Roboter. In *Robotik 2002*. VDI-Verlag, 2002.

[189] B. Schiele und J. L. Crowley. A Comparison of Position Estimation Techniques Using Occupancy Grids. In *Proc. of Int. Conf. on Robotics and Automation*, Seiten 1628–1634, 1994.
`http://www-white.media.mit.edu/people/bernt/Pubs/icra94.ps.gz`

[190] Christian Schlegel. OROCOS::SmartSoft Introduction, 2003.
`http://www1.faw.uni-ulm.de/orocos`

[191] Christian Schlegel und Robert Wörz. SmartSoft Introduction, 1996.

[192] D.C. Schmidt und S. D. Huston. *C++ Network Programming: Mastering Complexity Using ACE and Patterns*. Addison-Wesley Longman, 2002.

[193] D.C. Schmidt und S. D. Huston. *C++ Network Programming: Systematic Reuse with ACE and Frameworks*. Addison-Wesley Longman, 2003.

[194] Sven Schuierer. Efficient Robot Self-localization in simple polygons. In *Intelligent Robots*, Seiten 129–146, 1996.

[195] D. Schulz, W. Burgard, D. Fox und A.B. Cremers. Tracking Multiple Moving Targetswith a Mobile Robot using Particle Filters and Statistical Data Association. In *International Conference on Robotics & Automation (ICRA-01)*, 2001.

[196] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun und Younes H. Coordination for Multi-Robot Exploration and Mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.

`http://www.informatik.uni-freiburg.de/~burgard/abstracts/exploration-%AAAI2000.abstract.html`

[197] I. Sommerville. *Software-Engineering*. Pearson Education, 2002.

[198] Cyrill Stachniss und Wolfram Burgard. An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[199] J. Stoer. *Numerische Mathematik II*. Springer Verlag, 1993.

[200] N. A. Streitz, J. Geißler und T. Holmer. Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces. In *LNCS Vol. 1370. Cooperative Buildings - Integrating Information, Organization, and Architecture. Proceedings of CoBuild '98, Darmstadt*. Springer Verlag, 1998.

[201] N.A. Streitz und P. Tandler. Roomware: Towards the Next Generation of Human-Computer Interaction Based on an Integrated Design of Real and Virtual Worlds. In *Human-Computer Interaction in the New Millenium*, Seiten 553–578, 2001.

[202] Peter Norvig Stuart Russel. *Artificial Intellgence*. Prentice-Hall, 1995.

[203] Raj Talluri und J. K. Aggarwal. Position Estimation Techniques for an Autonomous Mobile Robot – A Review. In *Handbook of Pattern Recognition & Computer Vision*, Kapitel 4.4, Seiten 769–801. World Scientific, 1993.

[204] Xuehou Tan. Fast computation of shortest watchman routes in simple polygons. *Information Processing Letters*, 77:27–33, 2001.

[205] P. Tandler, N.A. Streitz und Th. Prante. Roomware - Moving Toward Ubiquitous Computers. In *IEEE Micro*, Seiten 36–47, November 2002.

[206] S. Thrun. A Probabilistic Online Mapping Algorithm for Teams of Mobile Robots. *International Journal of Robotics Research*, 20[5]:335–363, 2001.
`http://www.cs.cmu.edu/~thrun/papersall.html`

[207] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haenel, C. Rosenberg, N. Roy, J. Schulte und Schulz D. Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *Journal of Robotics Research*, 2001.

[208] S. Thrun, W. Burgard und D. Fox. A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2000.

[209] S. Thrun, D. Fox und W. Burgard. Monte Carlo Localization With Mixture Proposal Distribution. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[210] S. Thrun, D. Fox, W. Burgard und Dellaert. F. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128[1-2], 2001.

`http://www.informatik.uni-freiburg.de/~burgard/abstracts/monte-carlo-%ai-01.abstract.html`

[211] S. Thrun, Jens-Steffen Gutmann, D. Fox, W. Burgard und B.J. Kuipers. Integrating Topological and Metric Maps for Mobile Robot Navigation: A Statistical Approach, 1998.

[212] Sebastian Thrun. Learning Occupancy Grids with Forward Models. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Seiten 1676–1681, 2001.

[213] Sebastian Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20[5]:335–363, 2001.

[214] Sebastian Thrun. Robotic Mapping: A Survey. In G. Lakemeyer und B. Nebel (Hrsg.), *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[215] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Dellaert, Dieter Fox, Dirk Hähnel, Charles Rosenberg, Nocholas Roy, Jamieson Schilte und Dirk Schulz. MINERVA: A Second-Generation Museum Tour-Guide Robot. In *Proc. of the International Conference on Robotics and Automation 1999*, 1999.

[216] Sebastian Thrun, A. Brücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell und T. Schmidt. Map learning and high-speed navigation in RHINO. In R.P. Bonasso D. Kortenkamp (Hrsg.), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press, 1998, 1998.

[217] Sebastian Thrun, Wolfram Burgard und Dieter Fox. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning and Autonomous Robots*, 31[5]:1–25, 1998.

[218] Sebastian Thrun, Wolfram Burgard und Dieter Fox. A Probabilistic Aproach to Concurrent Mapping and Localization for Mobile Robots. In *Machine Learning and Autonomous Robots*, Band 31, Seiten 1–25, 1999.

[219] N. Tomatis, I. Nourbakhsh, Kai O. Arras und Roland Siegwart. A Hybrid Approach for Robust and Precise Mobile Robot Navigation with Compact Environment Modeling. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2001.

[220] N. Tomatis, I. Nourbakhsh und Roland Siegwart. Hybrid Simultaneous Localization and Map Building: Closing the Loop with Multi-Hypotheses Tracking. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.

[221] Frank Chee-Da Tsai. A Probabilistic Approach to Geometric Hashing using Line Features. Technischer Bericht 0640, Robotics Research Laboratory, Courant Institute of Mathematica Science, New York Univ., 1993.
`ftp://cs.nyu.edu/pub/tech-reports/`

[222] Frank Chee-Da Tsai. Using Line Invariants for Object Recognition by Geometric Hashing. Technischer Bericht 0625, Robotics Research Laboratory, Courant Institute of Mathematical Sciences, New York Univ., 1993.
`ftp://cs.nyu.edu/pub/tech-reports/`

[223] Vincent Tscherter. Rekonstruktion von Bahn und Karte aus Scandaten. Diplomarbeit, ETH Zürich, 1999.

[224] N. Tschichold-Gürman und S.J. Vestli. MOPS, a system for mail distribution in office type buildings. In *1st Euromicro Workshop on Advanced Mobile Robots (EUROBOT)*, 1996.
`http://www.ifr.mavt.ethz.ch/research/mops/index.html`

[225] Universität Utrecht. *CGAL-Computational Geometry Algorithms Library (Release 1.2)*, 1999.
`http://www.cs.uu.nl/CGAL`

[226] U.S. Air Force. *U.S. Air Force Fact Sheet Predator*, 2002.
`http://www.af.mil/news/factsheets/RQ_1_Predator_Unmanned_Aerial.html`

[227] Harrie van Dijck, Maarten Korsten und Ferdi van der Heijden. Geometric hashing: error analysis. Technischer bericht, Laboratory for Measurement and Instrumentation, Dep. of Electrical Eng., Univ. of Twente, 1997.
`http://utelmi01.el.utwente.nl/info/mi-projects/mb-vision/overview.htm%l`

[228] Robin Vaughan. Mars Pathfinder Navigation. via WWW, 1997.
`http://mars.jpl.nasa.gov/MPF/mpf/mpfnavpr.html`

[229] R. Vincent, B. Horling und V. Lesser. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator. In *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems.*, Band 1887. Wagner & Rana (eds.), Springer, Januar 2001.
`http://mas.cs.umass.edu/pub/paper_detail.php/200`

[230] H. Wittig W. Brenner, R. Zarnekow. *Intelligente Softwareagenten*. Springer-Verlag, 1998.

[231] R. Want, B.N. Schilit, N.I. Adams, R. Gold, K. Petersen, D. Goldberg, J.R. Ellis und M. Weiser. The ParcTab Ubiquitous Computing Experiment. Technischer Bericht CLS-95-1, Xerox Palo Alto Research Center, 1995.

[232] Roy Want, Trevor Pering, James Kardach und Graham Kirby. The Personal Server: The Center of Your Ubiquitous World. Intel Research and Mobile Products Group white paper, 2001.

[233] M. Weiser. The Computer of the 21th Centure. *Scientific American*, 265[3]:94–104, 1991.

[234] Gerhard Weiss, Christopher Wetzler und Ewald von Puttkamer. Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans. In *Proc. of Int. Conf. on Intelligent Robots and Systems*, Seiten 595–601, 1994.
`ftp://ag-vp-ftp.informatik.uni-kl.de/pub/Gerd/Weiss.Position_from_sca%ns.ps.Z`

[235] Greg Welch und Gary Bishop. An Introduction to the Kalman Filter. Technischer Bericht 95-041, University of North Carolina at Chapel Hill, 1995.
`http://www.cs.unc.edu/~welch/kalman/kalman_filter/kalman.html`

[236] Haim J. Wolfson. Model-Based Object Recognition by Geometric Hashing. In O. Faugeras (Hrsg.), *Proc. of 1st Europ. Conf. on Comput. Vision (ECCV 90)*, Nummer 427 in Lecture Notes in Computer Science, Seiten 526–536, 1990.

[237] Alexander Zelinsky. *Field and Service Robotics*. Springer Verlag, 2000.

[238] T.G. Zimmermann. Personal Area Networks: Near-field intrabody communcation. Technischer Bericht G321-5627, IBM Research, 1996.

Abbildungsverzeichnis

1 Beispiele für Roboter 2

2 Fehlerhülle bei Kartensegment $\overline{st}$ 23
3 Akzeptierte Segmente durch den Linienfilter 29
4 Kartendetails: Nischen, Vorsprünge 33
5 Kartendetails: Lücken 33
6 Abstandsintervall von Segmenten 39
7 Parametrisierung von Segmenten 42
8 Segment mit Fehlerabschätzung 43
9 Kompatible Modellfeatures zum Anfragefeature f 44
10 Berechnetes Gitter beim Pose-Clustering 46
11 Transformationsbestimmung für Segmentfeatures 56
12 Fehlerabschätzung der Transformationsbestimmung 57
13 Synthetische Karten 63
14 Lokalisationsgüte Alignment 64
15 Lokalisationsgüte Pose-Clustering 65
16 Lokalisationsgüte Geometrisches Hashing 65

17 Bewertungsfunktion eines Messpunktes 87
18 Kartierungstest 1 95
19 Kartierungstest 1 - Ergebnisse 96
20 Kartierungstest 1- Detailvergleich 96
21 Kartierungstest 2 97
22 Kartierungstest 2 - Ergebnisse 98
23 Kartierungstest 2- Detailvergleich 98
24 Kartierungstest 3 99
25 Kartierungstest 3 - Ergebnisse 100
26 Kartierungstest 3- Detailvergleich 101
27 Kartierungstest 4 102
28 Vereinfachte Minimax-Methde 107
29 Zusammenfassen zweier Segmente 111
30 Unterschiedlich generierte Karten mit ABE-Crust 114
31 Unterschiedlich generierte Karten mit DK-Crust 115
32 Generierte Instanzen mittels des Segment-Mergings 116
33 Generierte Instanzen mittels der Punktreduktion 119

34 Generierte Instanzen mittels des Segment-Mergings 120

35 Homogene Segmentierung einer Karte 127
36 Featuremengen der Teilkarten 128
37 Test 1-Erste Testumgebung 136
38 Test 1-Zweite Testumgebung 137
39 Test 1-Dritte Testumgebung 138
40 Test 2-Erste Testumgebung 139
41 Test 2-Zweite Testumgebung 140
42 Test 2-Dritte Testumgebung 141
43 Test 3-Erste Testumgebung 142
44 Test 3-Zweite Testumgebung 143
45 Test 3-Dritte Testumgebung 144

46 Roadmaps 157
47 Verzögerte Positionskorrektur 165
48 Regelung eines Roboters zu einem Zielpunkt 172

49 Beispiel eines Overlay-Baums 180
50 Fenster 183
51 Worst-Case in polygonalen Karten 184
52 Scanpolygon und freie Kanten 196
53 Sensorpunktmengen 197
54 Beispiel eines lokalen Overlay-Baums 202
55 Explorationstest 1 210
56 Explorationstest 2 212
57 Einsatzumgebungen des EFM-Verfahrens 213
58 Testlauf Hypothesenelimination 217

59 Aktive Umgebung 223
60 Aktives Objekt 224
61 Intelligentes Objekt 224
62 Intelligente Umgebung 225
63 Wissensbasiertes System 227
64 Informationsraum 228

65 Dienste und Komponenten 235
66 Dekomposition von Algorithmen 235
67 Prozessstruktur eines Roboter-Systems 242

68 Modellierung des intelligenten (Roboter-)Objekts 250
69 Modellierung des Informationsraumes des (Roboter-)Objekts 252
70 Konzept für eine generische Mensch-Maschine-Schnittstelle 252

71 SICK-LMS und zugehöriger Scan 274

72 Eingesetzte mobile Roboter 276

Algorithmen-Verzeichnis

2.1 Segmentextraktion aus Laser-Sensordaten 27
2.2 Split-Algorithmus 28
2.3 Partitionsdiskriminante 30
2.4 Fehlerprüfung 31
2.5 Hough-Transformation 32
2.6 Vergröbernde Feature-Extraktion 34
2.7 Greedy-Heuristik für MinSetCover 35
2.8 Alignment-Matching 37
2.9 Generierung einer Feature-Zuordnung 41
2.10 Pose Clustering 45
2.11 Preprocessing beim Geometrischen Hashing 49
2.12 Erkennungsschritt beim Geometrischen Hashing 50
2.13 Transformationsbestimmung 58
2.14 Verifikation durch Projektion 60
2.15 Feature-basierte Lokalisation 61

3.1 Berechnung des Bewegungsmodells 84
3.2 Berechnung des Sensormodells 85
3.3 Spiralsuche in einer Gitterkarte 86
3.4 Integration von Sensordaten in Gitterkarte 89
3.5 Inkrementelle probabilistische Kartierung 93
3.6 Reduktion der Punkte 105
3.7 Heuristik zur Kurven-Rekonstruktion 107
3.8 Reduktion der Segmente 108
3.9 Feature-basierte Kartierung durch Kurvenrekonstruktionn 109
3.10 Feature-basierte Kartierung durch Segment-Merging 112

4.1 Probabilistische Auswahl 125
4.2 Histogrammreduktion 126
4.3 Segmentierung der Karte 129
4.4 Schnelle iterative Lokalisation 131
4.5 Kombinierte feature-basierte Lokalisation 132

5.1 Probabilistische Roadmap-Berechnung 159
5.2 Pfadplanung auf Roadmaps 160
5.3 Pfadplanung mittels Zellzerlegungen 162

5.4 Potentialfeld-Algorithmus 163
5.5 Inkrementelle Sensor-basierte Positionsverfolgung 168
5.6 Modellbasierte Positionsverfolgung 170
5.7 Regelung entlang vorgegebenen Trajektorie 173
5.8 Dynamische Hindernisvermeidung 175
5.9 Navigation oder Zielsuche 176

6.1 Algorithmus von Schuierer 185
6.2 Monte-Carlo-Lokalisation 189
6.3 Aktive Markov-Lokalisation 190
6.4 Multiples Hypothesentracking 192
6.5 Explorationsstrategie nach Jensfeld 193
6.6 Sensorpunktauswahl 198
6.7 Explorationsschritt 200
6.8 Verifkation einer Hypothesenmenge 204
6.9 Globale Selbstlokalisation durch EFM 207