Sensormodelle für probabilistische Verfahren zur Lokalisierung mobiler Roboter mit Abstandssensoren

Diplomarbeit

eingereicht von

Dirk Zitterell

Albert-Ludwigs-Universität Freiburg

Fakultät für Angewandte Wissenschaften

Arbeitsgruppe Autonome Intelligente Systeme

Prof. Dr. W. Burgard

April, 2004

Danksagung

Mein Dank richtet sich an all die Menschen, die mich während der Erstellung meiner Diplomarbeit freundlich unterstützt haben. Ganz besonders danken möchte ich Herrn Prof. Dr. Wolfram Burgard für die sehr gute Betreuung, seinen Rat und seine Hilfe. Des Weiteren danke ich den Mitarbeitern der Arbeitsgruppe Autonome Intelligente Systeme – Dirk Hähnel, Cyrill Stachniss, Maren Bennewitz, Rudolph Triebel, Daniel Sack, Michael Veeck und Patrick Pfaff –, die mir hilfreich zur Seite standen und stets für eine äußerst angenehme Arbeitsatmosphäre sorgten. Herzlicher Dank gebührt Vandana Gairola für das Korrekturlesen der Arbeit, ihre unermüdliche Geduld und ihren moralischen Beistand während der Höhen und Tiefen. Bei meiner Familie möchte ich mich für die vielfältige Unterstützung während meines ganzen Studiums bedanken, ohne welche diese Arbeit erst gar nicht möglich gewesen wäre.

«Sie schweben im Weltraum oder krabbeln am Meeresboden,
sie bauen, beobachten, ernten, graben, inspizieren, kleben, lackieren,
löschen, löten, melken, montieren, musizieren, pflücken, reinigen,
schiessen, schlachten, schleppen, schneiden, schweissen, sprengen.
Einige rollen Sushi. Andere spielen Fussball oder treten im Kino auf.»

Gero von Randow[1]

[1]Gero von Randow: *Roboter – Unsere nächsten Verwandten*; Rowohlt, 1977

Inhaltsverzeichnis

1. Einleitung

Der schon viele Jahrhunderte gehegte Wunsch der Menschen, mühsame Arbeiten von Maschinen erledigen zu lassen, ist vom heutigen Standpunkt aus gesehen ein kleines Stück Wirklichkeit geworden. Mit seinem Zitat hat dies Gero von Randow wunderbar auf den Punkt gebracht. Die Rede ist natürlich von mobilen Robotern, die vor allem durch die Erfindung des Transistors und durch die fortschreitende Entwicklung integrierter Schaltungen geschaffen werden konnten. Durch viele technische Innovationen und durch die rasche Evolution in der Computertechnik entwickelte sich nicht nur eine enorme Vielfalt dieser *Fronarbeiter*[1], sondern es gestaltete sich auch das Spektrum an Aufgaben, die mobilen Robotern übertragen werden, immer facettenreicher.

Entwickelte man vor einem Jahrzehnt noch mobile Roboter vorwiegend für den Einsatz als unbemanntes, autonomes Transportmittel oder als kybernetischer Briefbote, so sind in diesem Augenblick mobile Roboter gerade dabei, als Museumsführer [Burgard et. al. 1998, 2000; Thrun et. al. 2000] zu arbeiten, Fußball zu spielen [The Robocup Federation 2004], als Avatare auf Messen präsent zu sein [Burgard et. al. 2002], ältere Menschen im Haushalt zu unterstützen [Montemerlo et. al. 2002], Außenanlagen dreidimensional zu vermessen [Thrun et. al. 2003; Ferguson et. al. 2003] oder die Oberfläche fremder Planeten zu analysieren [NASA 2004]. In allen Bereichen besteht die Aufgabe eines Roboters darin, den Menschen bei dessen Tätigkeiten zu unterstützen. Gleichzeitig muss hierbei aber auch gewährleistet werden, dass der Roboter bei der Ausführung seiner Aufgaben weder Menschen noch sich selbst gefährdet. Sicherheitsvorkehrungen müssen bereits bei der Fortbewegung des mobilen Fahrzeugs oder der mobilen Laufmaschine getroffen werden. Denn vor allem bei Kollisionen besteht die Möglichkeit, dass Lebewesen oder Gegenständen erheblicher Schaden zugefügt wird.

Für den Menschen ist es kein Kunststück, nicht mit den Objekten seiner Umgebung zu kollidieren, obgleich er sich beinahe ausschließlich auf visuelle Wahrnehmungen stützt. Der mobile Roboter hingegen braucht eine genauere Vorstellung seiner Aufenthaltsposition innerhalb der ihn umgebenden Welt. Er benötigt diese Information in erster Linie für die zuverlässige Pfadplanung und zur sicheren Kollisionsvermeidung. Die Positionsbestimmung bzw. Lokalisierung gilt daher als

[1] Der tschechische Dichter Karel Capek (1890–1938) benutzte das Wort „robot“ in dem Stück *Opilek*, das 1917 veröffentlicht wurde. Dieses Wort lehnt sich an das alttschechische Wort „robota“ an, das soviel wie Fronarbeit bedeutet.

eine der wichtigsten Voraussetzungen für den zuverlässigen Betrieb mobiler Roboter.

Das Ziel der Lokalisierung mobiler Roboter besteht darin, mit Hilfe der Sensorik die aktuelle Roboterposition zu ermitteln und diese während der Bewegung des Roboters stets zu aktualisieren. Die für die Roboterlokalisierung in Frage kommende Sensorik umfasst Instrumente zum Messen des vom Roboter zurückgelegten Weges, Abstandssensoren oder auch Kameras. Die Gewinnung der gesuchten Roboterposition erfolgt in den meisten Fällen durch die Anwendung probabilistischer Verfahren. Diese Techniken versuchen, mit Hilfe der Sensordaten eine Aussage darüber zu machen, wo der Roboter sich befindet, indem sie aus den Rohdaten der Sensoren inhärente Fehler, Ungenauigkeiten etc. herausfiltern. Zur Auswertung der Sensordaten und zur Fehlermodellierung dienen sogenannte *Sensormodelle*. Die für Abstandssensoren eingesetzten Sensormodelle sind Thema dieser Arbeit.

1.1. Zielsetzung

In der mobilen Robotik erfreuen sich probabilistische Verfahren zur Positionsbestimmung großer Beliebtheit. Auch wir wollen in dieser Arbeit ein solches Verfahren einsetzen. Insbesondere wollen wir einen sogenannten *Partikelfilter* verwenden. Die Partikelfilter gehören zur Gruppe der Bayes-Filter, welche mit Hilfe von Sensordaten die Roboterposition durch eine Wahrscheinlichkeitsdichte über dem gesamten Zustandsraum darzustellen versuchen. Da eine solche Verteilungsdichte eine beliebige Gestalt annehmen kann, wurden Partikelfilter entwickelt, die durch eine Menge von Partikeln bzw. Stichproben (engl. Samples) diese Verteilungsdichte möglichst gut approximieren. Je dichter die Samples in einem Bereich beieinander liegen, desto größer ist auch die Wahrscheinlichkeit für die Verteilungsdichte in diesem Bereich.

Um die Verteilungsdichte zu berechnen und fortwährend entsprechend der Sensorinformation zu aktualisieren, benötigt ein Partikelfilter verschiedene Modelle, die dazu beitragen, aus den rohen, meist verrauschten Sensordaten entsprechende relevante Informationen zu extrahieren. Das *Sensormodell* bildet mit zwei weiteren Komponenten, die wir in den nächsten Abschnitten vorstellen werden, das Fundament des eingesetzten Partikelfilters. Das Sensormodell sorgt dafür, dass die rohen Messdaten der Abstandssensoren sinnvoll im Partikelfilter für die Bewertung der Samples verwendet werden können, indem es Fehler und äußere Einflüsse auf die Sensoren – vor allem bedingt durch die dynamische Umgebung – berücksichtigt.

Wir wollen in dieser Arbeit strahlenbasierte Sensormodelle untersuchen und unter verschiedenen praktischen Gesichtspunkten evaluieren. Denn ein großer Nachteil dieser Art von Sensormodellen liegt darin, dass sie – um die Bewertung der Samples

durchführen zu können – die an allen potentiellen Roboterpositionen zu erwartenden Abstände benötigen. Die Berechnung dieser Referenzlängen erfolgt durch sogenanntes *Ray-Casting* mit Hilfe einer Karte der Umgebung. Ray-Tracing wurde in der Computergrafik entwickelt, um eine dreidimensionale Szene in ein zweidimensionales Bild umzusetzen. Das Ray-Casting stellt einen Spezialfall dieser Technik dar. Beim Ray-Casting werden die Strahlen nämlich nur so lange verfolgt, bis sie zum ersten Mal auf ein Objekt treffen. Strahlreflexionen werden also nicht berücksichtigt. Nun gibt es grundsätzlich zwei Möglichkeiten. Entweder werden die Abstände während des Lokalisierungsprozesses berechnet, was sehr zeitaufwändig ist, oder durch eine Diskretisierung des Raums und einer Vorberechnung vor dem eigentlichen Lokalisierungsprozess gewonnen, was sehr viel Speicherplatz in Anspruch nimmt. In diesem Zusammenhang interessiert uns vor allem, welche Faktoren und welche Parameter sich wie auf welche Eigenschaften der Lokalisierung auswirken. Die Ziele bei der Entwicklung eines Sensormodells sind ein sparsamer Umgang mit Rechenressourcen, eine hohe Zuverlässigkeit sowie eine gewisse Resistenz gegenüber verrauschten Sensordaten und Einflüssen aus der dynamischen Umgebung wie beispielsweise Personen.

Letztendlich wollen wir ein Sensormodell entwickeln, das seinen Beitrag zu einem effizienten Lokalisierungssystem leistet. Dieses Modell soll gänzlich auf eine speicherintensive und teils unnötige Vorberechnung der Referenzlängen verzichten und sich möglichst gut an die vorhandene Umgebung anpassen. Da die Berechnung der zu erwartenden Distanzen nun in diesem Fall während des Lokalisierungsprozesses erfolgt, wollen wir versuchen, den Ray-Casting-Prozess entsprechend zu optimieren.

1.2. Aufbau dieser Arbeit

Wir werden nun auf die Gliederung dieser Arbeit eingehen. Nach dieser Einleitung werden wir in Kapitel 2 zunächst verwandte Arbeiten vorstellen, welche ähnliche Ziele bezüglich der Lokalisierung verfolgen. Das darauffolgende Kapitel beschäftigt sich genauer mit der Lokalisierung mobiler Roboter. Wir werden Lokalisierungsprobleme formulieren, erklären, welche Sensorinformationen uns zur Verfügung stehen, und uns die formalen und mathematischen Grundlagen der Monte-Carlo-Lokalisierung erarbeiten. Mit der gewonnenen Kenntnis von sämtlichen, beteiligten Komponenten der Lokalisierung mobiler Roboter, können wir uns in Kapitel 4 schließlich intensiv mit den Sensormodellen auseinandersetzen. Wir werden verschiedene Ray-Casting-Strategien kennenlernen und diese evaluieren sowie äußere Einflüsse, welche die Abstandswerte der Sensormessungen modifizieren, identifizieren. Wir werden schließlich auf ein sich dynamisch adaptierendes Sensormodell stoßen. In Kapitel 5 stellen wir zunächst einmal die Betriebssoftware vor, in deren Umgebung die Software für die Experimente – der *Localizer* – entwickelt wurde. Danach untersuchen

wir in Kapitel 6 experimentell die Eigenschaften und das Verhalten des entwickelten Sensormodells. Zuletzt werden wir in Kapitel 7 die gewonnenen Erkenntnisse zusammenfassen, Anregungen für Erweiterungen nennen sowie einen Ausblick auf zukünftige Arbeiten geben.

2. Verwandte Arbeiten

Die Lokalisierung ist ein grundlegendes Problem in der mobilen Robotik. Aus diesem Grund existiert auch eine Vielfalt an Arbeiten und Forschungsschwerpunkten auf diesem Gebiet. Von einem Lokalisierungsverfahren wird heute nicht nur eine geringe Komplexität, sondern vermehrt auch eine intelligente und effiziente Ressourcen-Nutzung gefordert. Die für uns relevanten Arbeiten auf dem Gebiet der Lokalisierung wollen wir in diesem Kapitel kurz vorstellen. Nach einem Überblick über verschiedene Lokalisierungsmethoden gehen wir in diesem Kapitel auf adaptierende Sampling-Strategien ein und im Anschluss daran lernen wir eine Alternative zu den auf Ray-Casting basierenden Sensormodellen kennen.

2.1. Lokalisierungsmethoden

Im Folgenden werden wir einige, mit der Monte-Carlo-Lokalisierung verwandte Methoden zur Lokalisierung mobiler Roboter mit Abstandssensoren vorstellen.

Kalman-Filter

Kalman-Filter verwenden unimodale Gaußverteilungen, um die Roboterposition zu schätzen. Da sie – selbst für mehrdimensionale Zustandsräume – sehr effizient implementiert werden können und über eine auf Gleitkommazahlen basierende Auflösung verfügen, wurden sie bereits in verschiedenen Arbeiten erfolgreich eingesetzt [Schiele und J.Crowley 1994; Gutmann und Schlegel 1996]. Aufgrund der Verwendung von unimodalen Gaußverteilungen müssen Kalman-Filter zu Beginn des Lokalisierungsprozesses die anfängliche Position des Roboters kennen. Darüberhinaus können sie keine beliebigen Verteilungsdichten approximieren. Aus diesem Grund besitzen sie nicht die Fähigkeit, das globale Lokalisierungsproblem zu lösen, sondern eignen sich in ihren ursprünglichen Form ausschließlich für die Positionsverfolgung.

Diskretisierung des Zustandsraumes

Eine andere Möglichkeit, die Position des Roboters zu ermitteln, bietet die Aufteilung des Zustandsraumes. Möglich sind in diesem Zusammenhang topologische Aufteilungen, wie es in der Arbeit von Nourbakhsh et. al. [1995] der Fall ist. Die

Topologie des Zustandsraums wird hierbei in Konstrukte zerlegt, die aus Zuständen für Wege, Kreuzungen, Abzweigungen, Räume etc. bestehen oder dem zugehörigen Voronoi-Diagramm entnommen wurden [Choset und Nagatani 2001]. Andererseits kann man den Zustandsraum aber auch wie in den Arbeiten von Burgard et. al. [1996, 1997] diskretisieren, wodurch man ein gut aufgelöstes Gitter erhält. Unter der Annahme, dass die Roboterposition aus einem Tripel besteht, welches die Roboterposition in der euklidischen Ebene und die Ausrichtung angibt, nimmt auch das entsprechende Gitter eine dreidimensionale Gestalt an. Wahrscheinlichkeitswerte in den Zellen repräsentieren nun den Aufenthaltsort des Roboters. Bewegt sich der Roboter, müssen diese Werte entsprechend aktualisiert werden.

Experimentelle Vergleiche verschiedener Lokalisierungsmethoden lassen sich in den Arbeiten von Gutmann et. al. [1998]; Gutmann und Fox [2002] und Kristensen und Jensfelt [2003] finden.

2.2. Adaptierende Sampling-Verfahren

Das Sampling ist neben dem Bewegungsmodell und dem Sensormodell die dritte wichtige Komponente, aus denen die Lokalisierung mit probabilistischen Verfahren besteht. Es dient bei Partikelfiltern dazu, die Verteilungsdichte, welche durch die Samplemenge repräsentiert wird, zu aktualisieren, indem es aus der aktuellen Samplemenge Samples für die neue Samplemenge *zieht*. Dieses Ziehen kann man sich als ein Ziehen mit Zurücklegen vorstellen, bei dem ein Sample mit einer Wahrscheinlichkeit proportional zu seiner Bewertung bzw. Gewichtung gezogen wird. In Abschnitt 3.3.2 beschäftigen wir uns eingehender mit den Grundlagen des Samplings.

Auch das Sampling muss – da immer mehr Module in einem Roboter-Betriebssystem gleichzeitig ausgeführt werden sollen – eine geringe Komplexität aufweisen und so wenig Prozessor- und Speicherlast wie möglich hervorrufen. Bei dieser Anforderung liegt die Schwierigkeit darin, trotz beschränkter Rechenzeit eine robuste und präzise Lokalisierung gewährleisten zu können. Die einzige Möglichkeit, die sich einer Sampling-Methode – neben einer geringen inhärenten Komplexität – hierfür bietet, ist das dynamische Anpassen der Sampleanzahl an die gegebenen Bedürfnisse.

KLD-Sampling

Das in der Arbeit von Fox [2001, 2003] vorgestellte KLD-Sampling, auf das wir in Abschnitt 3.3.4 näher eingehen werden, weil es u. a. auch für den experimentellen Teil dieser Arbeit eingesetzt wurde, ist ein solches adaptierendes Verfahren. Die Idee des KLD-Samplings besteht darin, in jedem Resampling-Schritt die Anzahl der

benötigten Samples zu schätzen, um den Fehler der bei Partikelfiltern angewandten Sample-basierten Approximation einer Verteilung gegenüber der tatsächlichen zu begrenzen. Dabei wird bei der tatsächlichen Wahrscheinlichkeitsdichte von einer diskreten, teilweise konstanten Verteilungsdichte ausgegangen. Die benötigte Sampleanzahl wird so gewählt, dass der Unterschied zwischen dem *Maximum-Likelihood*-Schätzer (engl. Maximum-Likelihood-Estimator) der Verteilung der Samplemenge und der wahren Verteilung eine frei wählbare Obergrenze ϵ nicht überschreitet. Das KLD-Verfahren erhielt seinen Namen von dem für die Unterschiedsmessung herangezogenen Maß: der Kullback-Leibler-Distanz, welche als Maß für die Distanz zwischen zwei Wahrscheinlichkeitsdichten eingesetzt wird.

Adaptierende Real-Time-Partikelfilter (ARTPF)

In der Arbeit von Kwok et. al. [2003a] werden Real-Time-Partikelfilter (RTPF) um die Funktionalität des KLD-Samplings erweitert. Ein Real-Time-Partikelfilter ist eine Weiterentwicklung eines gewöhnlichen Partikelfilters, um mit begrenzten Rechenressourcen umzugehen [Kwok et. al. 2003b]. Dabei werden immer k nacheinander eintreffende Observationen $z_{t_1}, \ldots, z_{t_k}$ zu einem Beobachtunsfenster (engl. Estimation Window) der Fenstergröße k zusammengefasst. Die ganze Samplemenge von N Samples wird daraufhin auf die k Beobachtungen aufgeteilt, so dass jeder Beobachtung z_{t_i} eine dieser k Teilmengen zugeordnet ist. Die gesamte Vermutung (Belief) bzw. die Verteilung der Samples eines jeden Beobachtungsfensters wird daraufhin aus den individuellen Teilvermutungen zusammengesetzt. Stünde nun genug Rechenzeit zur Verfügung, dann könnte man die Fenstergröße auf 1 reduzieren und für eine Observation alle N Samples generieren. Die Überzeugung des Fensters wäre damit optimal bezüglich der Samples. Weil aber nun die Rechenressourcen dazu nicht ausreichen, vollzieht man die Aufteilung und setzt die resultierende Überzeugung beim Schließen des Festers – d. h. nach dem Eintreffen der k-ten Observation des betreffenden Fensters – aus den Teilüberzeugungen wieder zusammen. Dazu müssen die einzelnen Teilüberzeugungen der Sampleteilmengen derart gewichtet werden, dass der resultierende Approximationsfehler für das Observationsfenster minimal wird. Der Approximationsfehler wird auch hier durch die Kullback-Leibler-Distanz gemessen und die Gewichte α_i der Samples werden durch Gradientenabstieg ermittelt. Der Vorteil dieses Verfahrens liegt nun vor allem darin, dass trotz limitierter Ressorcen keine Observationen ausgelassen werden müssen.

2.3. Endpunkt-Sensormodell

Die für Gitter traversierende Sensormodelle zur Bewertung der Samples notwendigen erwarteten Scans[1] werden mit Hilfe von Ray-Casting-Techniken ermittelt. Da diese Techniken allerdings sehr kostspielig sind, hat man mit dem auf Wahrscheinlichkeitskarten basierenden Endpunkt-Sensormodell ein Verfahren entwickelt, das gänzlich auf die Berechnung der erwarteten Strahllängen verzichtet [Konolige und Chou 1999]. Stattdessen werden die Samples durch eine Korrelation zwischen dem gemessenen Scan und der Umgebungskarte gewichtet. Obwohl sich dieses Sensormodell in keiner Weise an den physikalischen Grundlagen, auf welchen die Arbeitsweise der Laserscanner zur Gewinnung der Strahllängen basiert, orientiert, so birgt dieses Modell doch einige nennenswerte Vorteile in sich. Der bedeutendste davon dürfte wohl die niedrige Komplexität sein. Durch den Verzicht auf die Berechnung der erwarteten Referenzlängen, entfällt das teure Ray-Casting komplett. An dessen Stelle tritt eine Berechnung der Zellen in der globalen Umgebungskarte, in welche die Endpunkte der Strahlen des gemessenen Scans ausgehend von der aktuellen Sampleposition fallen. Je besser die Sampleposition mit der wahren Roboterposition übereinstimmt, desto wahrscheinlicher sind diese Endzellen auch belegt oder desto näher sind diese einer belegten Zelle. Für diese Bewertung ist eine Wahrscheinlichkeitskarte anzufertigen, die entweder direkt aus der vorhandenen Umgebungskarte, welche den Belegtheitsgrad der Zellen enthält, oder indirekt über eine Karte minimaler euklidischer Distanzen erzeugt wird. Wir werden uns in Abschnitt 4.6 eingehender mit dem Endpunkt-Sensormodell auseinandersetzen und die dazugehörigen Grundlagen formalisieren.

[1] Als Scan oder Scanning bezeichnet man ganz allgemein das Abtasten mit Hilfe eines Scanners – in unserem Fall also das Messen von Abständen. Wir werden in dieser Arbeit aber auch die durch diesen Vorgang zu einem bestimmten Zeitpunkt gewonnenen Daten (Abstandsmessungen) als Scan bezeichnen.

3. Lokalisierung mobiler Roboter

Die Lokalisierung spielt in der mobilen Robotik eine sehr wichtige und grundlegende Rolle. Ohne jegliche Kenntnis der eigenen Position hat ein autonomer Roboter keine oder nur eine sehr geringe Chance, effiziente und kollisionsfreie Pfade zu planen oder komplexe Aufgaben auszuführen. Verfügt der Roboter darüberhinaus über zusätzliche bewegliche Aktoren wie z. B. Greifarme, die selbst keine Sensoren zur Kollisionsvermeidung haben oder für deren Betrieb eine genaue Kenntnis der Roboterposition vorausgesetzt wird (siehe Abb. 3.1), dann steigen die Anforderungen an die Präzision und Zuverlässigkeit eines Lokalisierungsverfahrens stark an.

Wir wollen in diesem Kapitel zuallererst Lokalisierungsprobleme definieren und klären, welche Informationen in Form von Sensordaten uns überhaupt zur Lösung dieser Probleme zur Verfügung stehen. Danach gehen wir auf die mathematischen Grundlagen probabilistischer Verfahren ein und stellen ein effizientes Lokalisierungsverfahren – die Monte-Carlo-Lokalisierung – vor. Warum überwiegend probabilistische Verfahren in der Lokalisierung zum Einsatz kommen, liegt sicherlich an der Tatsache, dass diese Ansätze erstaunlich gut mit den in der Robotik allgegenwärtigen Unsicherheiten bzw. Messfehlern zurechtkommen.

3.1. Lokalisierungsprobleme

Bei der Lokalisierung unterscheidet man grundsätzlich drei Arten von Problemen: die *Positionsverfolgung* (engl. *Position Tracking*), die *globale Lokalisierung* sowie das *Problem des gekidnappten Roboters*.

- Im Falle der Positionsverfolgung ist dem Roboter seine anfängliche Position bekannt. Er muss hierbei seine Position entsprechend der Odometrie-[1] und Sensormessungen stets aktualisieren. Da aber sowohl die Odometrie als auch die übrige Sensorik in der Praxis keine genauen Messwerte liefern, stellt selbst dieses Problem schon eine Herausforderung in der mobilen Robotik dar. Die

[1] Als Odometrie bzw. Koppelnavigation bezeichnet man das Messen des vom Roboter zurückgelegten Weges. Die Messung erfolgt typischerweise durch Sensoren an den Rädern oder am Motor.

(a) Roboter „Zora“

(b) Roboter „Albert“

Abbildung 3.1.: Gezeigt werden die beiden Roboter Zora und Albert, die auf der B21r-Plattform der Firma iRobot basieren. (a) Zora ist mit einem mehrachsigen Arm mit einem zweiten Laserscanner sowie einer hochauflösenden Kamera ausgestattet. Dieses System dient der dreidimensionalen Kartografierung und dem dreidimensionalen Vermessen von Objekten. (b) Albert ist für den Einsatz als Museumsführer mit einem beweglichen Gesicht, einem Touch-Screen und einer Spracherkennungssoftware ausgestattet. Für beide Systeme ist eine genaue Lokalisierung der Roboterplattform unabdinglich.

Ungenauigkeit der Messdaten kann verschiedene Gründe haben. Diese reichen von ungenauen Abstandssensoren (Fehlmessungen, Reflexionen etc.) über Drift und Schlupf der Odometrie bis hin zu Einwirkungen durch dynamische Hindernisse in Form von Personen.

- Bei der globalen Lokalisierung weiß der Roboter zu Beginn des Lokalisierungsprozesses seine Position nicht. Er hat folglich die Aufgabe allein mit Hilfe eines Modells seiner Umgebung, d. h. einer Karte, und einer Folge von Sensor- und Odometriemessungen seine Position zu ermitteln.

- Das Problem des gekidnappten Roboters stellt wohl das schwierigste Lokalisierungsproblem dar. Hierbei liegt das Ziel in dem Erkennen einer falschen Position und im Wiederfinden der richtigen Position. Gerade beim Roboterfußball kommt es häufig vor, dass ein Roboter, der beispielsweise einen anderen Spieler foult, vom Spielfeld genommen werden muss und an einer anderen Stelle wieder aufgestellt wird [The Robocup Federation 2004]. Dieses Problem stellt eine Erweiterung der globalen Lokalisierung dar. Denn ein Lokalisierungssystem, das fähig ist, global zu lokalisieren, muss nur um eine Methode zum Erkennen von falschen Positionen erweitert werden, um das Problem des gekidnappten Roboters lösen zu können.

Auch wenn sich diese Probleme klar trennen lassen, so sind die Übergänge in der Praxis meist fließend: Ein Lokalisierungssystem geht von der Phase der globalen Lokalisierung kontinuierlich in die Phase der Positionsverfolgung über. Wir werden in Kapitel 6 hierzu mehr erfahren.

3.2. Sensorinformation

Sensoren spielen in der Robotik eine herausragende Rolle. Denn sie dienen – ähnlich wie die Sinnesorgane bei Mensch und Tier – der Wahrnehmung der Umgebung. Die Anwendungsbereiche von Sensoren erstrecken sich über die Kollisionsvermeidung, die Lokalisierung, das Mapping[2] sowie die Selbstkalibrierung [Roy und Thrun 1999]. Bei mobilen Robotern kommen häufig die folgenden Arten von Sensoren zum Einsatz: Tastsensoren (engl. Bumper), Odometriesensoren, Abstandssensoren (engl. Range Sensors) und Kameras. Während ein Bumper ausschließlich Kollisionen zu detektieren vermag, werden Abstandssensoren neben der Kollisionsvermeidung auch zur Lokalisierung oder sogar zur Erkennung und Verfolgung von Personen verwendet. Die Odometriesensoren messen die Bewegungen eines Roboters, welche in Abschnitt 3.3.3 behandelt werden. Typische Vertreter von Abstandssensoren sind Sonarsensoren (Ultraschallsensoren) und Laserscanner (siehe Abb. B.2 in Anhang B). Besonders letztere erfreuen sich in der mobilen Robotik großer Beliebtheit und sind dank ihrer Messgenauigkeit und Robustheit gut für die o. g. Anwendungen geeignet.

[2] Kartografierung der Umgebung

Um von diesen Merkmalen auch profitieren zu können, ist ein möglichst präzises, an die entsprechenden Gegebenheiten des Sensors angepasstes Sensormodell erforderlich. In der Regel sind aus diesem Grund Sensormodelle stark an der physikalischen Art und Weise der Messwertgewinnung des entsprechenden Sensors angelehnt. So werden für Ultraschallsensoren überwiegend auf Kegel basierende und für Laserscanner auf Strahlen basierende Modelle eingesetzt.

Beide Klassen von Sensoren liefern jedoch unglücklicherweise keine absolut exakten Messwerte. Während manche Abweichungen durch ein begrenztes Auflösungsvermögen der Sensoren erklärt werden können, trifft für viele dieser verfälschten Messwerte die Sensoren keine Schuld. Denn vor allem äußere Faktoren beeinflussen die Messungen erheblich. Auf diese äußeren Einflüsse werden wir in Abschnitt 4.3 zurückkommen, wenn wir näher auf die Sensoren und ihre Sensormodelle eingehen. Zur Lokalisierung soll uns jedenfalls Information in Form von Odometrie- und Abstandsdaten zur Verfügung stehen.

3.3. Grundlagen

Nun da wir die Probleme der Lokalisierung mobiler Roboter präziser formuliert haben und wissen, welche Sensorinformationen uns zur Verfügung stehen, wollen wir im Folgenden die mathematischen Grundlagen probabilistischer Lokalisierungsverfahren erarbeiten.

Russell und Norvig definieren einen mobilen Roboter als Agenten in einer realen Welt, wobei sich diese reale Welt als unzugänglich, nichtdeterministisch, dynamisch und kontinuierlich charakterisieren lässt [Russell und Norvig 1995]. Bezogen auf die Lokalisierung bedeutet unzugänglich, dass der Agent nicht in der Lage ist, alles Wissenswerte über seine Umgebung mit absoluter Sicherheit in Erfahrung zu bringen, weil seine Sensoren nur bestimmte physikalische Eigenschaften der Umgebung messen können und dabei auch noch ungenau sind. Des Weiteren kann der Agent auch nicht davon ausgehen, dass die von ihm ausgeführten Aktionen tatsächlich die erwarteten Wirkungen zeigen. Vielmehr muss der Agent z. B. wegen ungenauen Odometriesensoren oder Unebenheiten des Bodens in der Lage sein, mit Unsicherheiten zurechtzukommen. In diesem kontinuierlichen Zustandsraum soll der Agent nun mit Hilfe der ihm zur Verfügung stehenden Sensordaten seine Position bestimmen.

3.3.1. Bayes-Filter

Als Werkzeug zur Schätzung der Position des Roboters soll die in den Arbeiten von Dellaert et. al. [1999] und Fox et. al. [1999a] vorgestellte Monte-Carlo-Lokalisierung zum Einsatz kommen. Die Grundlage dieser Lokalisierungsmethode bilden

sogenannte Bayes-Filter. Bayes-Filter schätzen den Zustand s eines dynamischen Systems mit Hilfe von Sensormessungen. Im Falle der Lokalisierung setzt sich dieser Zustand aus der Position des Roboters in der euklidischen Ebene und dessen Ausrichtung zusammen; das dynamische System ist der Roboter in seiner realen Umgebung und die Sensormessungen bestehen aus Abstandsmessungen, Odometriemessungen oder auch Kamerabildern [Dellaert et. al. 1999; Wolf 2001; Wolf et. al. 2002].

Wir werden später noch genauer auf die Eigenschaften der Monte-Carlo-Lokalisierung eingehen. Zunächst wollen wir aber auf Grund der rekursiven Funktionsweise von Bayes-Filtern eine rekursive Berechnungsvorschrift zur Schätzung der Position herleiten. Dabei wird zunächst keine konkrete Position berechnet, sondern vielmehr eine Wahrscheinlichkeitsverteilung über dem Zustandsraum. Deshalb definieren wir die Vermutung bzw. den *Belief* unseres Agenten, sich zum Zeitpunkt t in Zustand s_t (engl. State) zu befinden, als Wahrscheinlichkeitsdichte:

$$Bel(s_t) := p(s_t|z_t, a_{t-1}, z_{t-1}, a_{t-2}, \ldots, z_0). \tag{3.1}$$

Dabei repräsentieren die Variablen $z_0, \ldots, z_t$ die Wahrnehmungen bzw. die Perzeptionen des Agenten in Form von Abstandsdaten und $a_0, \ldots, a_{t-1}$ die Aktionen des Agenten, d. h. die Bewegungen im Zustandsraum. Ferner nehmen wir o.B.d.A. an, dass diese beiden Arten von Sensordaten abwechselnd eintreffen. Mit Hilfe der Bayes'schen Regel können wir die rechte Seite der Gleichung (3.1) ersetzen und erhalten:

$$Bel(s_t) := \frac{p(z_t|s_t, a_{t-1}, \ldots, z_0)\, p(s_t|a_{t-1}, z_{t-1}, \ldots, z_0)}{p(z_t|a_{t-1}, z_{t-1}, \ldots, z_0)}. \tag{3.2}$$

Offensichtlich stellt der Nenner bezüglich der Variablen s_t eine Konstante dar, weshalb wir mit $\eta := p(z_t|a_{t-1}, z_{t-1}, \ldots, z_0)^{-1}$ vereinfacht schreiben:

$$Bel(s_t) := \eta\, p(z_t|s_t, a_{t-1}, \ldots, z_0)\, p(s_t|a_{t-1}, z_{t-1}, \ldots, z_0). \tag{3.3}$$

Bayes-Filter setzen nach Fox et. al. [2000] voraus, dass es sich bei den ankommenden Messdaten um einen Markov-Prozess handelt, d. h. dass neu ankommende Daten bedingt unabhängig von früheren Daten sind, sofern der aktuelle Zustand s_t bekannt ist. Markov-Prozesse haben daher die Eigenschaft, dass die Wahrscheinlichkeit eines Zustands nur vom unmittelbar vorhergehenden Zustand abhängt. Darüberhinaus sind bei einer zum Zeitpunkt t neu eintreffenden Perzeption z_t alle vorher eingetroffenen Daten bekannt. Formal bedeutet dies:

$$p(z_t|s_t, a_{t-1}, \ldots, z_0) = p(z_t|s_t). \tag{3.4}$$

Damit können wir Gleichung (3.3) vereinfachen zu:

$$Bel(s_t) := \eta \; p(z_t|s_t) \; p(s_t|a_{t-1}, z_{t-1}, \ldots, z_0). \tag{3.5}$$

Durch Integration über s_{t-1} zum Zeitpunkt $t-1$ erhalten wir:

$$Bel(s_t) := \eta \; p(z_t|s_t) \int p(s_t|s_{t-1}, a_{t-1}, \ldots, z_0) \; p(s_{t-1}|a_{t-1}, z_{t-1}, \ldots, z_0) \; ds_{t-1}. \tag{3.6}$$

Wiederum können wir die Markov-Annahme zur Vereinfachung anwenden:

$$Bel(s_t) := \eta \; p(z_t|s_t) \int p(s_t|s_{t-1}, a_{t-1}) \; p(s_{t-1}|a_{t-1}, z_{t-1}, \ldots, z_0) \; ds_{t-1}. \tag{3.7}$$

Rufen wir uns unsere ursprünglich Definition für den Belief noch einmal ins Gedächtnis (siehe Gleichung (3.1)) und setzen diese entsprechend ein, erhalten wir die Berechnungsvorschrift in der gewünschten rekursiven Form:

$$\boxed{Bel(s_t) := \eta \; p(z_t|s_t) \int p(s_t|s_{t-1}, a_{t-1}) \; Bel(s_{t-1}) \; ds_{t-1}} \tag{3.8}$$

Wegen der rekursiven Eigenschaft fehlen uns zur Implementierung dieser Vorschrift offensichtlich nur noch die Verteilungsdichte $Bel(s_0)$ im Anfangszustand sowie die beiden bedingten Verteilungsdichten $p(z_t|s_t)$ und $p(s_t|s_{t-1}, a_{t-1})$:

- Die Wahrscheinlichkeit $p(z_t|s_t)$ bezeichnet man als *Sensormodell*. Das Sensormodell hat die Aufgabe, die aktuelle Sensormessung z_t gegeben eine bestimmte Position s_t probabilistisch zu bewerten. Wir werden später feststellen, dass das Sensormodell entscheidend das Konvergenzverhalten, die Genauigkeit und die Robustheit eines Lokalisierungssystems charakterisiert.

- $p(s_t|s_{t-1}, a_{t-1})$ stellt gemäß Dellaert et. al. [1999] das sogenannte *Bewegungsmodell* dar. Es integriert die Aktion a_{t-1} des Roboters – in unserem Falle also dessen Bewegung –, die zu einem Übergang von der Verteilungsdichte im Zustand s_{t-1} in die Verteilungsdichte des Nachfolgezustands s_t führt.

- Natürlich muss auch die anfängliche Verteilungsdichte $Bel(s_0)$ bekannt sein. Im Falle der globalen Lokalisierung geht man hierbei von einer Gleichverteilung über dem Zustandsraum aus. Bei der Positionsverfolgung dagegen ist die Anfangsposition des Roboters bekannt und $Bel(s_0)$ wird in der Regel durch eine Gauß-Verteilung um diese Position beschrieben.

Die jeweils aktuelle Verteilung kann gemäß Gleichung (3.8) schrittweise mit dem Eintreffen neuer Odometrie- bzw. Abstandsdaten berechnet werden. Ein solcher Berechnungsschritt lässt sich in zwei Phasen zerlegen: Vorhersagephase und Aktualisierungsphase.

In der Vorhersagephase erfolgt die Schätzung des Nachfolgezustands $\widehat{Bel}(s_t)$ aus $Bel(s_{t-1})$ unter Verwendung des Bewegungsmodells $p(s_t|s_{t-1}, a_{t-1})$ und der Odometriedaten a_{t-1}:

$$\widehat{Bel}(s_t) := \int p(s_t|s_{t-1}, a_{t-1})\ Bel(s_{t-1})\ ds_{t-1}. \tag{3.9}$$

Aus dieser Vorhersage $\widehat{Bel}(s_t)$ für die Verteilung des Folgezustands wird in der Aktualisierungsphase unter Einbeziehung der Perzeption z_t und des Sensormodells $p(z_t|s_t)$ die aktuelle Wahrscheinlichkeitsverteilung ermittelt:

$$Bel(s_t) := \eta\ p(z_t|s_t)\ \widehat{Bel}(s_t). \tag{3.10}$$

Offenbar findet in dieser Phase eine Plausibilitätsüberprüfung und Korrektur der vorhergesagten Verteilung entsprechend der Abstandsdaten und des Sensormodells statt, was die Notwendigkeit eines guten Sensormodells unterstreicht. Der hier vorgestellte Filterungsschritt wird während des gesamten Lokalisierungsprozesses – sofern Messdaten vorliegen – iterativ wiederholt.

3.3.2. Monte-Carlo-Lokalisierung

Bei der Implementierung von Gleichung (3.8) ist man damit konfrontiert, eine beliebige Wahrscheinlichkeitsdichte möglichst effizient zu berechnen. Dies ist keine triviale Aufgabe. Eine Möglichkeit, eine solche Verteilungsdichte zu repräsentieren ist ein sogenannter Partikelfilter. Wie wir bereits gesehen haben, repräsentiert dieser die Verteilungsdichte durch eine Menge von Partikeln bzw. Samples [Fox et. al. 1999a]. Da mit dieser Methode quasi beliebige Verteilungsdichten repräsentiert werden können ist diese sogenannte *Monte-Carlo-Lokalisierung* im Gegensatz zu einem Kalman-Filter[3], der von einer Gauß'schen Verteilung ausgeht, in der Lage, das globale Lokalisierungsproblem zu lösen. Außerdem arbeitet sie im Vergleich zu gridbasierten Verfahren häufig wesentlich effizienter, da sie ihre Ressourcen im Zustandsraum auf Bereiche mit hoher Wahrscheinlichkeit konzentriert. Die Effizienz

[3]Kalman-Filter versuchen die Roboterposition durch Fusionierung der Sensordaten zu schätzen. Dabei versuchen sie stets den Positionsfehler zu minimieren. Sie sind zwar äußerst effizent, aber auch auf lokale Selbstlokalisierung eingeschränkt, da sie nicht mit mehreren Positionen umgehen können.

hängt allerdings von der Anzahl der verwendeten Samples ab. Deshalb ist eine möglichst robuste und genaue Lokalisierung mit möglichst wenigen Samples letztendlich das Ziel.

In der Ebene wird die Position des Roboters durch den Vektor $s = (x, y, \theta)^T$ beschrieben. Die Komponenten x und y sind dabei die Koordinaten im kartesischen Koordinatensystem und θ die Ausrichtung des Roboters. Ein Sample setzt sich aus dieser Position und einem zusätzlichen Faktor w, der das *Gewicht* (bzw. engl. Importance) des entsprechenden Samples angibt, zusammen: (s^i, w^i).

Die Verteilungsdichte, die den Belief aus Gleichung (3.8) beschreibt, wird bei der Monte-Carlo-Lokalisierung durch eine Menge von N Samples approximiert:

$$Bel(s_t) \approx S_t = \left\{(s^i, w^i)\right\}_{i=1\ldots N}. \tag{3.11}$$

Üblicherweise werden die Gewichte der N Samples nach jedem Schritt für das Resampling normiert, so dass sie sich stets zu 1 aufsummieren. Das Resampling dient bei Partikelfiltern nämlich dazu, durch Ziehen aus einer Menge gewichteter Samples eine neue Stichprobenmenge zu erhalten, wobei für ein Sample die Wahrscheinlichkeit, gezogen zu werden, proportional zu seinem Gewicht ist. Für das Resampling soll also gelten:

$$\sum_{i=1}^{N} w^i = 1. \tag{3.12}$$

Jedes Sample stellt dabei eine mögliche Roboterposition dar. Bei der globalen Lokalisierung werden nun diese N Samples gleichmäßig mit dem Gewicht $w^i = 1/N$ über den Raum aller möglichen Zustände verteilt; bei der Positionsverfolgung bilden diese Samples eine Gauß-Verteilung um die bekannte Anfangsposition.

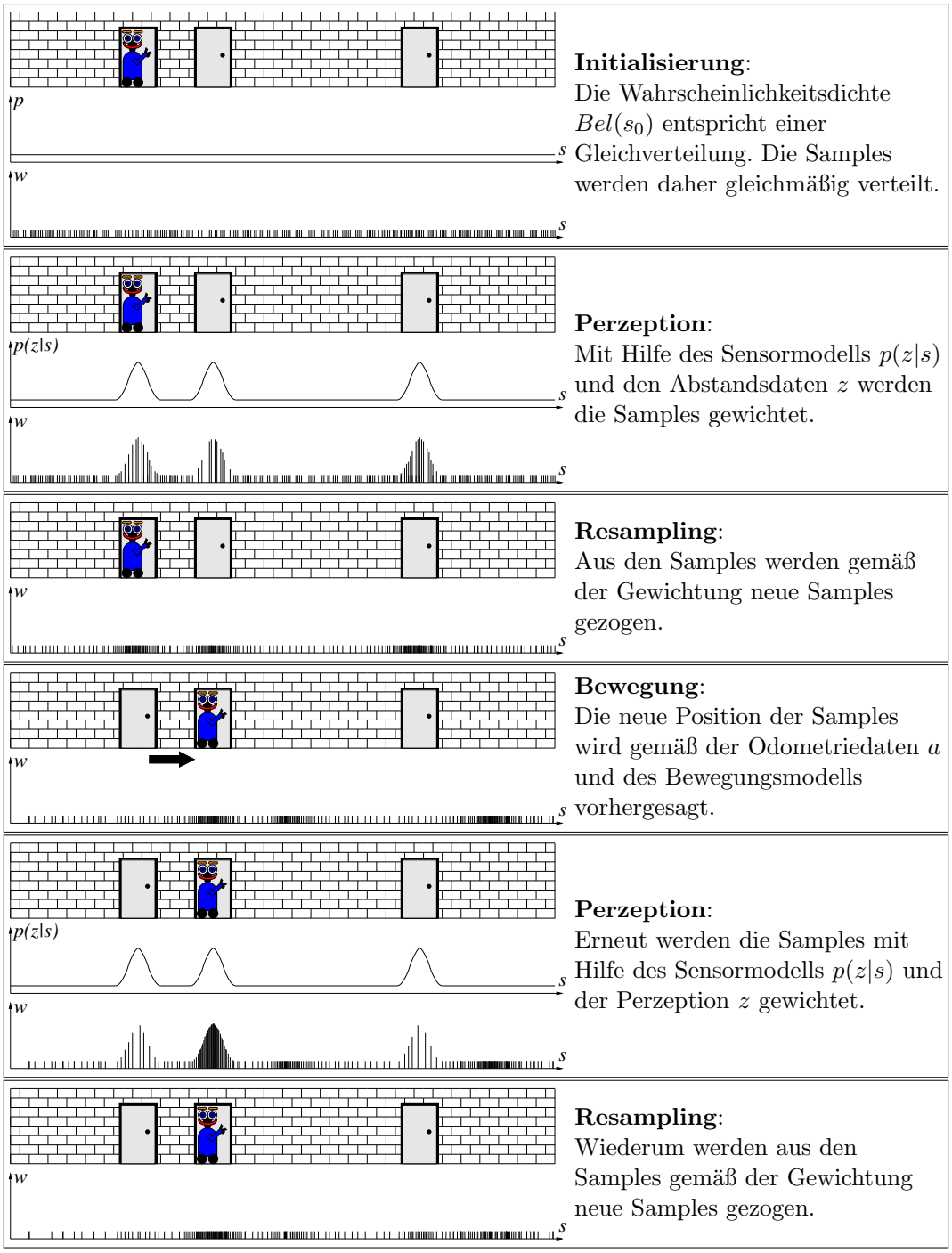
Mit dem Eintreffen neuer Odometrie- und Abstandsmessungen wird mit dem Filtern der Samplemenge begonnen. Algorithmus 1 zeigt einen Partikelfilter-Algorithmus für das Resampling mit fester Anzahl an Samples. Das Resampling (Zeilen 13–18) lehnt sich an das von Carpenter et. al. [1999] beschriebene Verfahren an. Es ermöglicht, den ganzen Filterschritt in linearer Zeit (Komplexität $O(N)$) durchzuführen. Auf die Vorhersagephase, die in Zeile 22 stattfindet, werden wir in Abschnitt 3.3.3 näher eingehen. Die Gewichtung der Samples für den nächsten Durchgang erfolgt in Zeile 26. Die Zeilen 33–35 sorgen dafür, dass die Gewichte wieder normiert werden. Die Normierung der Gewichte ist notwendig für das verwendete Resampling-Verfahren, weil das Ziehen der neuen Samples durch die Erzeugung von N im Intervall $[0; 1]$ gleichverteilten Zufallszahlen erreicht wird. Abbildung 3.2 zeigt anschaulich die prinzipielle Arbeitsweise eines MCL-Partikelfilters. Die ersten Schritte eines globalen Lokalisierungsprozesses sind in Abbildung 3.6 dargestellt.

Algorithmus 1 FiltereSamples

Eingabe: Samplemenge S_{t-1} mit N Samples, welche die Überzeugung $Bel(s_{t-1})$ repräsentiert; Bewegung a_{t-1}; Perzeption z_t

Ausgabe: Samplemenge S_t mit N Samples, welche die Überzeugung $Bel(s_t)$ zum Zeitpunkt t repräsentiert

1: *// Initialisierung*
2: $S_t := \emptyset$
3: $\eta := 0$
4: $i := 1$
5: $\delta :=$ Rand()$/N$ *// Rand() liefert zufälliges* γ *mit* $0 \leq \gamma < 1$
6: $q := w_{t-1}^1$
7:
8: *// Generieren der* N *Samples für* S_t
9: **for** $j := 1$ to N **do**
10:
11: *// Resampling: Ziehe ein Sample aus* S_{t-1} *mit Wahrscheinlichkeit*
12: *// proportional zu der Verteilung der Gewichte in* S_{t-1}
13: $u := \delta + j/N$
14: **while** $q < u$ **do**
15: $i := i + 1$
16: $q := q + w_{t-1}^i$
17: **end while**
18: $s_{t-1}^j := s_{t-1}^i$
19:
20: *// Vorhersage: Berechne* s_t^j *aus* s_{t-1}^j und der Bewegung a_{t-1}
21: *// unter Verwendung des Bewegungsmodells* $p(s_t|s_{t-1}, a_{t-1})$
22: $s_t^j :=$ BewegeSample(s_{t-1}^j, a_{t-1})
23:
24: *// Gewichtung: Berechne das Samplegewicht unter Verwendung*
25: *// des Sensormodells* $p(z_t|s_t^j)$ *und der Perzeption* z_t
26: $w_t^j := p(z_t|s_t^j)$
27:
28: $\eta := \eta + w_t^j$
29: $S_t := S_t \cup \{(s_t^j, w_t^j)\}$
30: **end for**
31:
32: *// Normieren der Gewichte*
33: **for** $j := 1$ to N **do**
34: $w_t^j := w_t^j/\eta$
35: **end for**



Initialisierung:
Die Wahrscheinlichkeitsdichte $Bel(s_0)$ entspricht einer Gleichverteilung. Die Samples werden daher gleichmäßig verteilt.

Perzeption:
Mit Hilfe des Sensormodells $p(z|s)$ und den Abstandsdaten z werden die Samples gewichtet.

Resampling:
Aus den Samples werden gemäß der Gewichtung neue Samples gezogen.

Bewegung:
Die neue Position der Samples wird gemäß der Odometriedaten a und des Bewegungsmodells vorhergesagt.

Perzeption:
Erneut werden die Samples mit Hilfe des Sensormodells $p(z|s)$ und der Perzeption z gewichtet.

Resampling:
Wiederum werden aus den Samples gemäß der Gewichtung neue Samples gezogen.

Abbildung 3.2.: Prinzip der Monte-Carlo-Lokalisierung nach Fox et. al. [1999b]. Von oben nach unten werden die ersten Schritte einer globalen Lokalisierung gezeigt. Nach ausreichend vielen Iterationen konvergieren die Samples zur wirklichen Position des Roboters.

3.3.3. Bewegungsmodelle

In den vorhergehenden Abschnitten haben wir festgestellt, dass die Charakteristik des Lokalisierungsprozesses bei der Monte-Carlo-Lokalisierung hauptsächlich von dem verwendeten Sensormodell und von dem verwendeten Bewegungsmodell geprägt wird. In diesem Abschnitt wollen wir uns näher mit dem Bewegungsmodell beschäftigen.

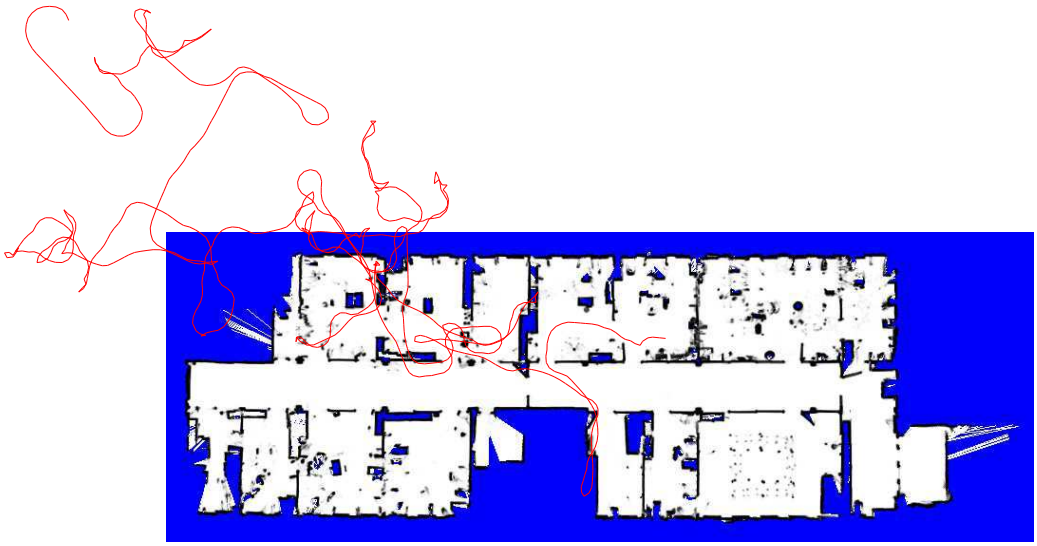
Als Odometrie bezeichnet man das Messen der vom Roboter zurückgelegten Wegstrecke. Diese Messung kann an den Radachsen oder direkt am Motor stattfinden. In der Regel messen an den Rädern angebrachte Rotationssensoren die Umdrehung des jeweiligen Rades, woraus sich der zurückgelegte Weg je Rad und somit auch der des Roboters ermitteln lässt. Allerdings sind Odometriemessungen fehlerhaft, weshalb auch ein geeignetes Bewegungsmodell für die Monte-Carlo-Lokalisierung notwendig ist, dessen Aufgabe darin besteht, die Fehler in den Daten zu modellieren. Denn je größer die Ungenauigkeit der Odometrie und je länger der zurückgelegte Weg des Roboters ist, desto größer ist auch der daraus resultierende Fehler, d. h. die Abweichung der gemessenen Position von der wirklichen Position des Roboters. Abbildung 3.3 zeigt neben dem tatsächlich zurückgelegten Pfad auch die von der Odometrie gemessene Trajektorie[4] des Roboters.

Für die Fehler in den Odometriedaten gibt es verschiedene Gründe: die inhärente Ungenauigkeit der Sensoren, unrunde Räder, das Rutschen der Räder beim Anfahren/Bremsen, Schlupf der Räder, der unebene Untergrund, eine unterschiedliche Gewichtsverteilung etc. All diese Fehler bewirken Abweichungen zwischen der gemessenen Bewegung und der tatsächlich ausgeführten Bewegung. Bei einer Rotationsbewegung führt dies zu einer Abweichung in der Ausrichtung; bei einer Translationsbewegung kommt es neben der Abweichung in der Distanz auch noch zu einer lateralen Abweichung sowie einer Abweichung in der Orientierung. Selbst Abweichungen, welche die Folge von Kollisionen sind, sind denkbar. Wir werden aber Kollisionen gänzlich ausschließen, weil wir eine funktionierende Kollisionsvermeidung voraussetzen und gleichzeitig Kontakt mit Personen, die den Roboter stoppen könnten, nicht in Betracht ziehen wollen.

Wie bereits in Abschnitt 3.3.1 beschrieben, bezeichnet $p(s_t|s_{t-1}, a_{t-1})$ formal das Bewegungsmodell, wobei s_{t-1} die Roboterposition unmittelbar vor dem Ausführen von Bewegung a_{t-1} und s_t die Position danach darstellt. Das Bewegungsmodell dient bei Partikelfiltern dazu, die Bewegungen des Roboters in die Samplemenge einfließen zu lassen. Doch auf Grund der o. g. Ungenauigkeiten entspricht die gemessene Bewegung nicht der tatsächlichen Bewegung des Roboters. Die Differenz wollen wir daher durch zwei Faktoren ausdrücken: systematische Fehler und zufällige Fehler. Die systematischen Fehler, welche die Drift bei der Bewegung des verwendeten Roboters beinhaltet, wollen wir durch experimentell ermittelte Parameter angeben.

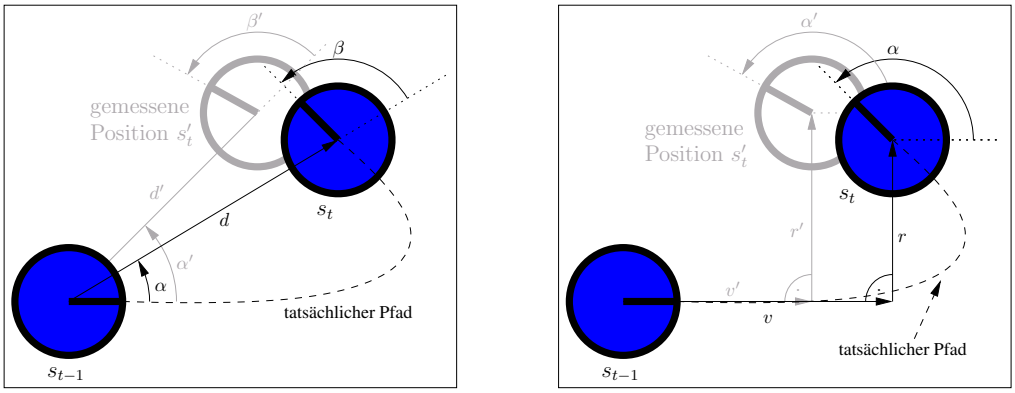
[4] In der mobilen Robotik versteht man unter der Trajektorie die Kurve, die der Roboter in seiner Umgebung zurücklegt.

(a) Die wahre Trajektorie des Roboters



(b) Die von der Odometrie gemessene Trajektorie

Abbildung 3.3.: (a) Gezeigt wird der vom Roboter zurückgelegte Weg im Gebäude 079. (b) Die von der Odometrie gemessene Trajektorie weist eine deutlich zu erkennende Drift auf. Die unkorrigierten Odometriedaten allein reichen zur Lokalisierung ganz offensichtlich nicht aus.



(a) (α, d, β)-Bewegungsmodell

(b) (v, r, α)-Bewegungsmodell

Abbildung 3.4.: (a) Das (α, d, β)-Bewegungsmodell versucht, durch eine Drehung um α, eine Translation der Länge d und eine abschließende Drehung um β die tatsächliche Bewegung zu beschreiben. (b) Beim (v, r, α)-Bewegungsmodell wird die tatsächliche Bewegung durch eine Translation der Länge v, eine Seitwärtsbewegung um r sowie eine Drehung um α repräsentiert.

Die zufälligen Fehler, die sich z. B. durch kleine Abweichungen von dem wahren Wert bemerkbar machen, nehmen wir als gaußverteilt an. In der Literatur findet man oft ein Bewegungsmodell, das die relative Bewegung zwischen zwei Positionen durch das Tripel $a_{t-1} = (\alpha, d, \beta)^T$ ausdrückt. Das bedeutet, dass eine Bewegung a durch eine Drehung um α, eine Translation um d und eine abschließende Drehung um β repräsentiert wird (siehe Abb. 3.4 (a)). Dieses Modell ist sehr stark an die Kinematik bestimmter Roboter angelehnt. Diese Roboter, zu denen beispielsweise die B21r-Roboter zählen (siehe Abb. 3.1), können sich mühelos auf der Stelle um die eigene Achse drehen und natürlich vorwärts bzw. rückwärts fahren. Wir wollen allerdings das in Abbildung 3.4 (b) skizzierte Modell verwenden. Es setzt sich aus einer Vorwärtsbewegung um v, einer Seitwärtsbewegung um r und aus einer abschließenden Drehung um α zusammen. Es orientiert sich auf Grund der Seitwärtsbewegung zwar nicht direkt an den Freiheitsgraden des Roboters, dafür ist es jedoch dank geometrischer Umformungen leicht zu implementieren und es erlaubt darüberhinaus in der experimentellen Phase eine einfache Anpassung der Parameter für die systematischen Fehler.

Aus der gemessenen Bewegung $a'_{t-1} = (v', r', \alpha')^T$ wollen wir unter Berücksichtigung der möglichen Abweichungen die tatsächliche Bewegung $a_{t-1} = (v, r, \alpha)^T$ rekonstruieren. Wir berechnen diese Bewegung daher wie folgt:

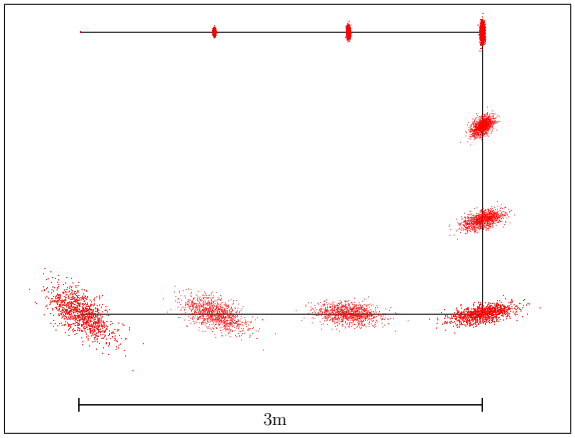


Abbildung 3.5.: Gezeigt werden die durch Samples approximierten Positionen des Roboters beim Ausführen mehrerer Fahrkommandos, welche die dargestellte U-förmige Trajektorie ergeben. Abstandsmessungen sind nicht in den Filterungsprozess miteingeflossen.

$$v = v' + |v'| \cdot \mathcal{N}(0, \sigma_v) + |r'| \cdot \mathcal{N}(0, \sigma_{rv}) \tag{3.13}$$
$$r = r' + |r'| \cdot \mathcal{N}(0, \sigma_r) + |v'| \cdot \mathcal{N}(0, \sigma_{vr}) \tag{3.14}$$
$$\alpha = \alpha' + |\alpha'| \cdot \mathcal{N}(0, \sigma_\alpha) + (|v'| + |r'|) \cdot \mathcal{N}(0, \sigma_{rv\alpha}) \tag{3.15}$$

Die Funktion $\mathcal{N}(0, \sigma)$ liefert eine Zufallszahl entsprechend einer Gauß-Verteilung mit Mittelwert 0 und Standardabweichung σ. Die Parameter σ_j stellen dabei systematische Fehler dar und werden experimentell ermittelt. Den Parametern σ_{rv}, σ_{vr} und $\sigma_{rv\alpha}$ kommt hierbei eine ganz besondere Bedeutung zu: Sie spezifizieren die Wechselwirkungen der einzelnen Komponenten untereinander. So hat beispielsweise eine gemessene Translation von $v' = 10\,m$ bei ungleichen Rädern sicherlich erheblichen Einfluss auf die Drift zur Seite sowie auf die Orientierung des Roboters. Die Gleichungen geben deutlich zu erkennen, dass wir linear mit der Bewegung anwachsende Rotations- und Translationsfehler annehmen wollen (siehe Abb. 3.5).

Innerhalb eines Partikelfilters wird die vermeintliche tatsächliche Bewegung $a_{t-1} = (v, r, \alpha)^T$ je Sample neu berechnet und jedes Sample i entsprechend dieser individuellen Bewegung a^i_{t-1} aktualisiert, d. h. dessen neue Position s^i_t vorhergesagt. Das Bewegungsmodell ermöglicht die Vorhersage der neuen Positionen der Samples gemäß der aktuellen Bewegung. Allerdings führt diese Vorhersagephase ohne eine anschließende Aktualisierungsphase nur zu einer fortschreitenden Zerstreuung der Samples, was Abbildung 3.5 verdeutlicht. Um den Filterungsprozess zu komplettieren, müssen demzufolge aus den potentiellen Positionen diejenigen extrahiert werden, welche am plausibelsten sind, d. h. am wahrscheinlichsten die aktuelle Abstandsmessung z_t

erwarten lassen. Dazu wird ein geeignetes Sensormodell $p(z_t|s_t)$ benötigt, welches wir in Kapitel 4 erläutern werden.

3.3.4. Sampling mit dynamischer Sampleanzahl

Ein großes Problem bei der Lokalisierung – vor allem zu Beginn einer globalen Lokalisierung – liegt in der Tatsache, dass man nie genau weiß, wie viele Samples überhaupt für eine zuverlässige Lokalisierung notwendig sind. Auf der einen Seite möchte man natürlich mit so wenigen Samples wie möglich auskommen, da für jedes Sample ein Scan, der wiederum aus einer Reihe von Strahlen besteht, ausgewertet werden muss und damit kostbare Rechenressourcen verbraucht werden. Auf der anderen Seite steht das Problem, bei zu wenigen Samples womöglich eine falsche Position zu ermitteln, weil die tatsächliche Roboterposition gar nicht durch die Menge der Samples abgedeckt wird. Das von Fox [2001, 2003] entwickelte KLD-Sampling löst dieses Problem, indem es durch dynamisches Anpassen der Sampleanzahl versucht, die gegenwärtige von den Samples aufgespannte Verteilungsdichte nach bestimmten, stochastischen Regeln zu approximieren. Insbesondere besteht die Idee darin, in jedem Schritt die Größe der Samplemenge so zu schätzen, dass mit einer Wahrscheinlichkeit von $1 - \delta$ der Fehler zwischen der tatsächlichen Verteilungsdichte des Beliefs und der durch die Samples approximierten kleiner ist als ein vorgegebener Wert ϵ.

Die Kullback-Leibler-Distanz (KL-Distanz) wird als Maß für die Distanz zwischen zwei Wahrscheinlichkeitsverteilungsdichten p_1 und p_2 herangezogen und ist für diskrete Zufallsvariablen wie folgt definiert:

$$K(p_1, p_2) = \sum_x p_1(x) \ \log \frac{p_1(x)}{p_2(x)}. \tag{3.16}$$

Die KL-Distanz ist immer nichtnegativ und – wie unschwer zu erkennen ist – nicht symmetrisch und deshalb auch kein exaktes Maß. Dennoch eignet sie sich gut für den Vergleich zweier Dichten und nimmt nur dann den Wert 0 an, wenn beide Verteilungsdichten identisch sind. Basierend auf diesem Maß für die Distanz zweier Verteilungen, möchte man nun eine geeignete Berechnungsvorschrift zur Ermittlung der benötigten Anzahl an Samples herleiten, um den Fehler der Approximation innerhalb der vorgegebenen Grenze zu halten.

Beim KLD-Sampling nimmt man nun an, dass n Samples aus einer diskreten Verteilung mit k verschiedenen Klassen (engl. Bins) gezogen werden. Der Zufallsvektor $(X_1, \ldots, X_k)$ soll für die Anzahl der Samples, die aus jeder einzelnen Klasse

Algorithmus 2 FILTERESAMPLESKLD

Eingabe: Samplemenge S_{t-1} mit N Samples, welche die Überzeugung $Bel(s_{t-1})$ repräsentiert; Bewegung a_{t-1}; Perzeption z_t; Grenzen ϵ und δ; Klassen-Größe Δ, minimale Anzahl von Samples $n_{\chi_{\min}}$

Ausgabe: Samplemenge S_t mit N' Samples, welche die Überzeugung $Bel(s_t)$ zum Zeitpunkt t repräsentiert

1: *// Initialisierung*
2: $S_t := \emptyset$; $n := 0$; $n_\chi := 0$; $k := 0$; $\alpha := 0$
3:
4: *// Generieren der Samples für S_t ...*
5: **repeat**
6:
7: *// Resampling: Ziehen eines Zustands*
8: Ziehe einen Index j aus der diskreten Verteilung,
9: die durch die Gewichte in S_{t-1} gegeben ist
10:
11: *// Sampling: Vorhersage des nächsten Zustands*
12: Berechne s_t^n aus s_{t-1}^j unter Verwendung
13: des Bewegungsmodells $p(s_t|s_{t-1}, a_{t-1})$ und der Bewegung a_{t-1}
14:
15: $w_t^n := p(z_t|s_t^n)$ *// Berechnung des Samplegewichts*
16: $\alpha := \alpha + w_t^n$
17: $S_t := S_t \cup \{(s_t^n, w_t^n)\}$
18:
19: *// Aktualisieren der erfassten Klassen*
20: *// und der gewünschten Sampleanzahl*
21: **if** (x_t^n fällt in eine neue Klasse b) **then**
22: $k := k + 1$
23: b :=besetzt
24: $n_\chi := \frac{k-1}{2\epsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}q_{1-\delta}\right)^3$
25: **if** ($n_\chi < n_{\chi_{\min}}$) **then**
26: $n_\chi := n_{\chi_{\min}}$
27: **end if**
28: **end if**
29: $n := n + 1$
30: **until** ($n \geq n_\chi$) *// ... bis die KL-Grenze erreicht ist*
31:
32: *// Normieren der Gewichte*
33: **for** $i := 1$ to n **do**
34: $w_t^i := w_t^i/\alpha$
35: **end for**

gezogen werden, stehen. Dieser Zufallsvektor ist multinomial verteilt (siehe Anhang A):

$$(X_1, \ldots, X_k) \sim \text{Mult}(n; p_1, \ldots, p_k) \tag{3.17}$$

mit den Parametern n und $p_1, \ldots, p_k$. Die Wahrscheinlichkeiten $p_1, \ldots, p_k$ geben die wahren Wahrscheinlichkeitswerte der Klassen an. Den Maximum-Likelihood-Schätzer von $p_1, \ldots, p_k$ bei der Verwendung von n Stichproben x_i erhält man nun mit $\widehat{p}_1, \ldots, \widehat{p}_k$ durch $\widehat{p}_j = n^{-1} p_j$. Dadurch hat man zwei Modelle, die man mit dem Likelihood-Quotienten vergleichen kann (siehe Anhang A):

$$LR = 2 \log \frac{\prod_{i=1}^{n} \widehat{p}(x_i)}{\prod_{i=1}^{n} p(x_i)}. \tag{3.18}$$

Durch die Aufteilung in Klassen fallen manche Samples in die gleiche Klasse. k sei im Folgenden die Anzahl der Klassen und X_j gebe die Anzahl der Samples an, die in die gleiche Klasse j fallen. Wir schreiben daher:

$$\begin{aligned}
LR &= 2 \log \frac{\prod_{j=1}^{k} \widehat{p}_j^{X_j}}{\prod_{j=1}^{k} p_j^{X_j}} && (3.19)\\
&= 2 \log \prod_{j=1}^{k} \widehat{p}_j^{X_j} - 2 \log \prod_{j=1}^{k} p_j^{X_j} && (3.20)\\
&= 2 \sum_{j=1}^{k} \log \widehat{p}_j^{X_j} - 2 \sum_{j=1}^{k} \log p_j^{X_j} && (3.21)\\
&= 2 \sum_{j=1}^{k} X_j \log \widehat{p}_j - 2 \sum_{j=1}^{k} X_j \log p_j && (3.22)\\
&= 2 \sum_{j=1}^{k} X_j \left(\log \widehat{p}_j - \log p_j \right) && (3.23)\\
&= 2 \sum_{j=1}^{k} X_j \log \left(\frac{\widehat{p}_j}{p_j} \right). && (3.24)
\end{aligned}$$

Wegen $X_j = n \cdot \widehat{p}_j$ kann man diese Gleichung auch wie folgt schreiben:

$$LR = 2n \sum_{j=1}^{k} \widehat{p}_j \log \left(\frac{\widehat{p}_j}{p_j} \right). \tag{3.25}$$

Die rechte Seite der Gleichung (3.25) entspricht nun der $2n$-fachen Kullback-Leibler-Distanz, falls man in Gleichung (3.16) für die beiden Verteilungsdichten einmal die tatsächliche Wahrscheinlichkeit $\underline{p} = (p_1, \ldots, p_k)$ der Klassen und einmal den Maximum-Likelihood-Schätzer $\underline{\widehat{p}} = (\widehat{p}_1, \ldots, \widehat{p}_k)$ einsetzt:

$$LR = 2n \cdot K(\underline{\widehat{p}}, \underline{p}). \tag{3.26}$$

In der Arbeit von Rice [1995] wird gezeigt, dass dieses Verhältnis gegen eine $\mathcal{X}^2$-Verteilung mit $k-1$ Freiheitsgraden konvergiert (siehe Gleichung A.12):

$$\lim_{n\to\infty} LR = \mathcal{X}^2_{k-1}. \tag{3.27}$$

Unter der Annahme, dass $\underline{p}$ die wahre Verteilung darstellt, wollen wir nun die Wahrscheinlichkeit, dass die KL-Distanz zwischen dieser wahren Verteilung und der durch die Samples approximierten Verteilung, d. h. dem Maximum-Likelihood-Schätzer $\underline{\widehat{p}}$, kleiner oder gleich ϵ ist, mit $P_{\underline{p}}(K(\underline{\widehat{p}}, \underline{p}) \leq \epsilon)$ benennen. Durch Erweiterung mit $2n$ auf beiden Seiten der eingeschlossenen Ungleichung und durch Substitution gemäß der Gleichungen (3.26) und (3.27) erhalten wir der Reihe nach:

$$\begin{aligned} P_{\underline{p}}(K(\underline{\widehat{p}}, \underline{p}) \leq \epsilon) &= P_{\underline{p}}(2nK(\underline{\widehat{p}}, \underline{p}) \leq 2n\epsilon) && (3.28)\\ &= P_{\underline{p}}(LR \leq 2n\epsilon) && (3.29)\\ &= P(\mathcal{X}^2_{k-1} \leq 2n\epsilon). && (3.30) \end{aligned}$$

Unser Ziel war eine Berechnungsvorschrift für die Mindestanzahl an Samples, die notwendig ist, damit der Fehler der von den Samples approximierten Verteilung gegenüber der wahren Verteilung mit einer Wahrscheinlichkeit von $1 - \delta$ kleiner oder gleich ϵ ist. Wir fordern also:

$$P_{\underline{p}}(K(\underline{\widehat{p}}, \underline{p}) \leq \epsilon) \doteq 1 - \delta. \tag{3.31}$$

Mit dieser Forderung gilt nach Gleichung (3.30) aber auch:

$$P(\mathcal{X}^2_{k-1} \leq 2n\epsilon) = 1 - \delta. \tag{3.32}$$

Diese Wahrscheinlichkeit ist gerade durch das (1-δ)-Quantil der $\mathcal{X}^2$-Verteilung gegeben:

$$P(\mathcal{X}^2_{k-1} \leq \mathcal{X}^2_{k-1,1-\delta}) = 1 - \delta. \tag{3.33}$$

Mit Hilfe der Gleichungen (3.32) und (3.33) wählen wir nun die Sampleanzahl n derart, dass gilt:

$$2n\epsilon = \mathcal{X}^2_{k-1,1-\delta} \tag{3.34}$$

und erhalten nach dem Auflösen nach n für unsere gesuchte Sampleanzahl:

$$n = \frac{1}{2\epsilon}\mathcal{X}^2_{k-1,1-\delta}. \tag{3.35}$$

Zur Berechnung des (1-δ)-Quantils der $\mathcal{X}^2$-Verteilung kann nun die Wilson-Hilferty-Transformation als gute Näherung herangezogen werden. Auf diese Weise erhält man letztendlich folgende Beziehung für die Abhängigkeit der gefragten Sampleanzahl n von der erwünschten Güte der Approximation:

$$n = \frac{1}{2\epsilon}\mathcal{X}^2_{k-1,1-\delta} \approx \frac{k-1}{2\epsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}q_{1-\delta}\right)^3, \tag{3.36}$$

$q_{1-\delta}$ steht dabei für das obere (1-δ)-Quantil der standardisierten Normalverteilung.

Das KLD-Sampling funktioniert in der Praxis prinzipiell ganz ausgezeichnet. Allerdings ist die Anzahl an Samples, die dieses Verfahren zu Beginn einer globalen Lokalisierung ermittelt, extrem groß und das Verfahren während dieser ersten Resampling-Schritte daher sehr zeitaufwändig. Dieses Verhalten kann jedoch leicht durch das Festlegen einer maximalen Anzahl von Samples, d. h. einer oberen Grenze, eingeschränkt werden. Algorithmus 2 zeigt das Grundgerüst eines Partikelfilters mit KLD-Sampling. Der Hauptunterschied zwischen dem KLD-Sampling-Algorithmus und dem Partikelfilter mit fester Sampleanzahl (vgl. Algorithmus 1) liegt darin, dass beim erstgenannten fortwährend eine Überprüfung dahingehend stattfindet, ob die gegenwärtige Sampleanzahl ausreicht, die gewünschte Approximationsqualität zu gewährleisten.

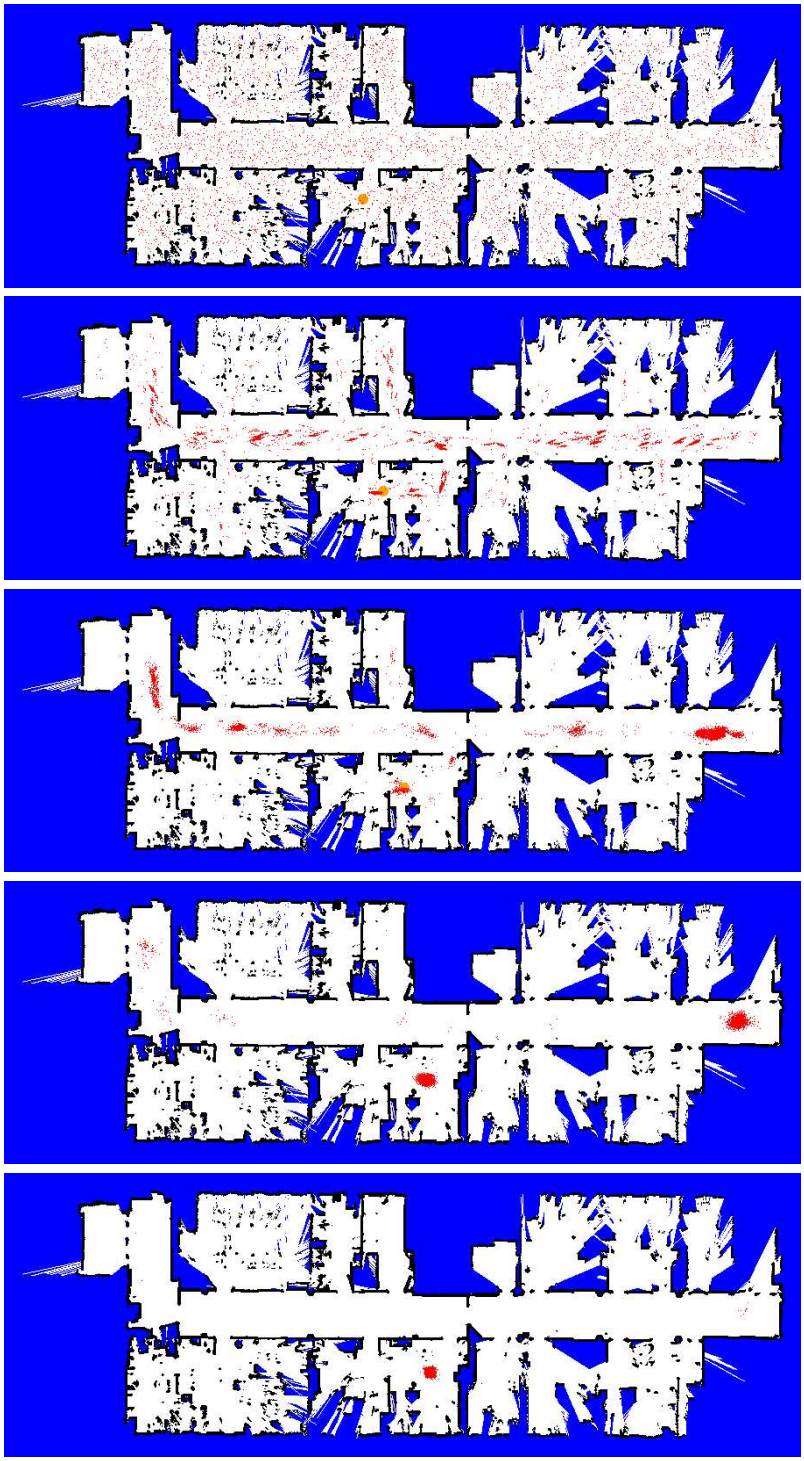


Abbildung 3.6.: Gezeigt werden die Samples in den ersten Schritten einer globalen Lokaliserung bei der Monte-Carlo-Lokalisierung mit einer festen Anzahl von 10000 Samples. Die gelbe Markierung repräsentiert die wahre Roboterposition; die Samples werden durch die roten Punkte gekennzeichnet.

4. Sensormodelle

Ein Sensormodell dient im Allgemeinen der Bewertung, wie wahrscheinlich an einer gewissen Position s_t eine gewisse Perzeption z_t ist. Welch wichtige Rolle das Sensormodell bei der Monte-Carlo-Lokalisierung spielt, haben wir bereits im vorhergehenden Kapitel erläutert. Die Bewertungsfunktion des Sensormodells ist nämlich das zentrale Steuerelement bei den Partikelfiltern. Sie regelt, wie schnell die Samples konvergieren, wie stark diese sich fokussieren und auch wie anfällig das ganze Lokalisierungssystem gegenüber Störungen ist. Auch die Genauigkeit der lokalisierten Position sowie die Fehlerrate bei der globalen Lokalisierung korrelieren stark mit dem eingesetzten Sensormodell.

In unserem Falle handelt es sich bei der Perzeption z_t um einen sogenannten *Scan*, d. h. um eine Reihe von M Abstandswerten, die von einem Abstandssensor mit gleichbleibender Winkelauflösung aufgenommen wurden. Abbildung 4.1 illustriert einen solchen Scan. Außerdem nehmen wir an, dass die einzelnen Strahlen unabhängig voneinander sind:

$$z_t = (z_t^1, \ldots, z_t^M). \tag{4.1}$$

Diese Unabhängigkeit ist in der Praxis meist nicht gegeben, weil benachbarte Einzelmessungen oft sehr nahe nebeneinander liegen und durch die gleichen Objekte und Faktoren beeinflusst werden. Wir werden deshalb nicht jeden möglichen Messwert einer Messreihe unseres Sensors betrachten, sondern immer ausreichend viele Werte überspringen.

4.1. Reduktion des Bewertungsproblems

Das Sensormodell hängt in erster Linie von dem verwendeten Sensor und der Umgebung ab. Da aber zur Lokalisierung eine Repräsentation der Umgebung benötigt wird, hängt das Sensormodell auch von dieser Repräsentation ab, die gewöhnlich in Form einer Karte vorliegt. Wann immer wir also den Term $p(z_t|s_t)$ verwenden, so meinen wir stattdessen $p(z_t|s_t, m)$, wobei der Parameter m für die die Umgebung repräsentierende Karte stehen soll. Der Einfachheit halber werden wir aber meistens die kürzere Schreibweise vorziehen. Die Karte m liegt in den meisten Fällen in

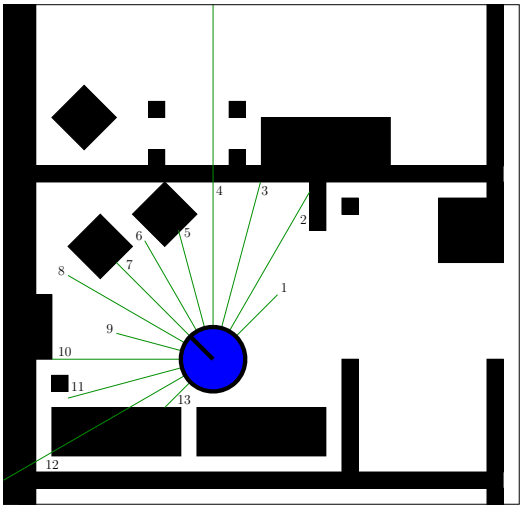


Abbildung 4.1.: Gezeigt wird der typische Scan eines Laserscanners mit einem Öffnungswinkel von 180°. Während die meisten Strahlen (2,3,5,7,10 und 13) recht genaue Längen vorweisen, findet man auch verkürzte (1,6,8,9 und 11) sowie Längen mit maximaler Reichweite (4 und 12) vor.

Form eines diskreten zweidimensionalen Gitters (engl. Occupancy Grid) vor, das in jeder Zelle einen Wahrscheinlichkeitswert für den Belegungsgrad dieser Zelle enthält [Moravec und Elfes 1985]:

$$m = \begin{pmatrix} m_{1,1} & \cdots & m_{1,L} \\ \vdots & & \vdots \\ m_{K,1} & \cdots & m_{K,L} \end{pmatrix}. \tag{4.2}$$

Die Werte $m_{k,l}$ liegen dabei im Intervall $[0;1]$, falls bei der Kartenerstellung der Belegungsgrad der betreffenden Zelle ermittelt werden konnte. Andernfalls besitzt die Zelle einen besonderen Wert (z. B. -1), wie beispielsweise bei abgeschatteten und eingeschlossenen Zellen. Die Karte besitzt darüberhinaus eine Auflösung, die angibt, welchen Raum jede Zelle in der realen Umgebung einnimmt. Abbildung 4.2 zeigt eine solche Karte mit entsprechenden abgeschatteten Bereichen.

Die Wahrscheinlichkeit des gemessenen Scans $z_t = (z_t^1, \ldots, z_t^M)$ an der Position s_t wollen wir auf einen Vergleich dieses Scans mit dem an der Position s_t zu erwartenden Scan $d = (d^1(s_t), \ldots, d^M(s_t))$ zurückführen:

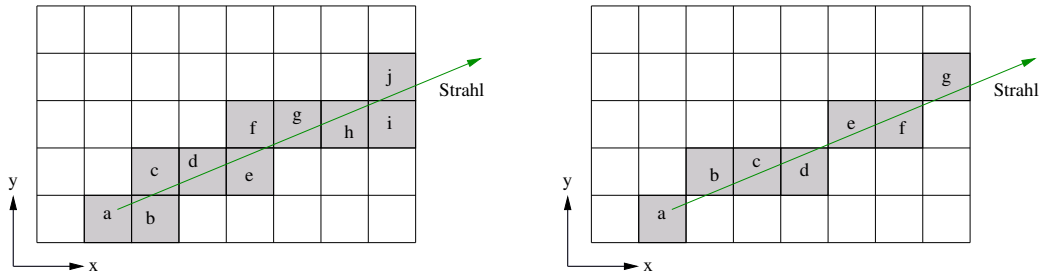
$$p(z_t|s_t) := \prod_{i=1}^{M} p(z_t^i|d^i(s_t)). \tag{4.3}$$

Abbildung 4.2.: Die Umgebungskarte der Arbeitsgruppe AIS am Institut für Informatik der Albert-Ludwigs-Universität Freiburg. Eine Zelle wird im Bild durch ein Pixel dargestellt. Die Grauwerte geben Aufschluss über den Belegungsgrad der jeweiligen Zelle. Schwarz entspricht einer maximalen Belegung (Wände, Möbel etc.), während weiß freie Zellen darstellt. Abgeschattete und eingeschlossene Bereiche, deren Belegungsgrad beim Mapping nicht ermittelt werden konnte, sind blau gekennzeichnet.

Auf diese Weise können wir das Bewertungsproblem des Sensormodells auf den Vergleich zweier Längen reduzieren. Doch da sowohl verschiedene äußere Einflüsse auf die gemessenen Distanzen als auch Ungenauigkeiten in den erwarteten Distanzen berücksichtigt werden müssen, gestaltet sich dieser Längenvergleich nicht ganz so trivial wie es zunächst scheint. Zuallererst bleibt die Frage zu klären, woher der zu erwartende Scan kommen soll. Denn offensichtlich muss an der Position eines jeden Samples der dort zu erwartende Scan ermittelt werden. Die Berechnung erfolgt durch Ray-Casting in einem durch die Karte m repräsentierten Gitter (engl. Grid).

4.2. Ray-Casting

Bei der Verwendung von Partikelfiltern erfordert die Bewertung der Samples unter der Verwendung von $p(z_t|s_t)$ ein Ray-Casting, um für jede Sampleposition den entsprechenden erwarteten Scan zu ermitteln. Beim Ray-Casting werden in der Karte ausgehend von der (x, y, θ)-Position eines Samples Strahlen ausgesendet – entsprechend der radialen Emission der Strahlen beim Laserscanner. Diese werden verfolgt, bis sie schließlich auf ein Hindernis treffen. Die dabei zurückgelegten Strecken werden dann mit den gemessenen Längen verglichen. Für das Ray-Casting benötigt man nur eine binäre Umgebungskarte, welche man direkt aus der Occupancy-Grid-Karte gewinnt, indem man einen Schwellwert festlegt, welcher die Zellen in freie und belegte klassifiziert. Unglücklicherweise erfordert Ray-Casting einen äußerst hohen Berechnungsaufwand. Aus diesem Grund gibt es hierfür verschiedene Methoden, die sich



(a) Inkrementelles Ray-Casting (b) Bresenham-Ray-Casting

Abbildung 4.3.: (a) Beim inkrementellen Ray-Casting durchquert der Strahl nacheinander die Zellen a-j. (b) Beim Bresenham-Ray-Casting werden nur die Zellen a-g durchquert. Die besuchten Zellen wurden jeweils grau eingefärbt.

vor allem in Geschwindigkeit und Genauigkeit unterscheiden.

4.2.1. Inkrementelles Verfahren

Das von Amanatides und Woo [1987] entwickelte, inkrementelle Ray-Casting-Verfahren hat den Vorteil, dass es mathematisch exakte Werte liefert und der Strahl an einem beliebigen Punkt in der Zelle beginnen kann. Für die Genauigkeit dieses Verfahrens zahlt man allerdings einen hohen Preis: Es werden nämlich sehr viele Zellen besucht, was sich durch einen hohen Bedarf an Ressourcen bemerkbar macht. Abbildung 4.3 (a) zeigt die Zellen, die von einem Strahl bei diesem Verfahren sukzessiv durchquert werden.

Der Algorithmus dieses Verfahrens (siehe Algorithmus 3) besteht aus zwei Phasen: einer Initialisierungsphase und einer Phase inkrementellen Traversierens. Der Strahl sei durch eine Gleichung $\vec{u} + t\vec{v}$ mit $t \geq 0$ gegeben. Die erste Zelle ist jene Zelle, in der der Strahl seinen Ursprung hat, d. h. in den $\vec{u}$ fällt. Die Variablen $cell_x$ und $cell_y$ werden mit den Koordinaten dieser Zelle initialisiert. Die Variablen $step_x$ und $step_y$ werden entweder mit -1 oder mit 1 initialisiert – je nachdem, ob beim Verfolgen des Strahls über Zellgrenzen hinweg die Zellkoordinaten inkrementiert oder dekrementiert werden müssen. Dies geht aus dem Richtungsvektor $\vec{v}$ hervor. Nun wird $t_x^{\max}$ auf den Wert t gesetzt, bei dem der Strahl die nächste vertikale Zellgrenze überschreitet; analog wird $t_y^{\max}$ auf den Wert t gesetzt, bei dem die nächste horizontale Zellgrenze überschritten wird. Das Minimum dieser beiden Werte gibt während des inkrementellen Traversierens stets an, wie weit auf dem Strahl gegangen werden kann, ohne die aktuelle Zelle zu verlassen. Nun werden noch die beiden Variablen $t_{\Delta x}$ und $t_{\Delta y}$ benötigt. Sie geben an, wie weit auf dem Strahl gegangen werden muss, damit in x-Richtung (bzw. y-Richtung) die Breite (bzw. Höhe) einer Zelle zurückgelegt wird.

Nach dieser Initialisierungsphase erfolgt dann iterativ das Durchqueren der Zellen, bis der Strahl schließlich auf eine belegte Zelle trifft oder das Gitter innerhalb der Karte verläßt. Algorithmus 3 zeigt diese zweite Phase.

4.2.2. Bresenham-Verfahren

Das Bresenham-Verfahren basiert auf dem von Bresenham [1965] entwickelten Algorithmus zur Rasterung von Strecken, welcher vorwiegend für das Zeichnen von Linien in der Computergrafik eingesetzt wird. Als Eingabe benötigt dieser Algorithmus den Anfangs- und den Endpunkt einer Linie. Die Ausgabe ist eine Liste von Zellen, die beim Einfärben entsprechend der Abbildung 4.3 (b) eine Linie vom Anfangs- zum Endpunkt bilden. Nun haben wir jedoch zunächst ein kleines Problem: Der Endpunkt unseres Strahls ist nicht bekannt; genau diesen wollen wir ja berechnen. Stattdessen kennen wir aber den Startpunkt sowie die Richtung des Strahls. Damit können wir glücklicherweise die Steigung m unserer Linie berechnen, so dass wir den Endpunkt gar nicht benötigen, um den Algorithmus anwenden zu können. Die Idee ist damit die folgende: Das Bresenham-Verfahren liefert uns nacheinander die Zellen vom Anfangspunkt zum (fiktiven) Endpunkt des Strahls und wenn wir auf eine belegte Zelle treffen, dann brechen wir den Algorithmus einfach ab.

Wir betrachten im Folgenden nur Strahlen, die sich im ersten positiven Oktanten unseres Gitters befinden, d. h. mit der x-Achse einen Winkel α im Bereich von $0 \leq \alpha \leq \frac{\pi}{4}$ bilden. Wir werden in dieser Arbeit Winkel immer im mathematischen Sinne messen und angeben, d. h. Winkel nehmen gegen den Uhrzeigersinn von der positiven x-Achse zur positiven y-Achse hin zu. Alle anderen Bereiche können später durch Vertauschen der x- und y-Koordinaten und Anpassen der Vorzeichen auf die selbe Art und Weise erschlossen werden. Nehmen wir einmal an, dass wir uns in der Zelle mit den Koordinaten (x, y) befinden, und dass unsere Methode in jedem Schritt um eine Zelle in x-Richtung weiterspringt, so kommen jeweils nur zwei Zellen für den nächsten Schritt in Frage: die nächste Zelle ist entweder die Zelle $(x + 1, y)$ oder die Zelle $(x+1, y+1)$. Die Fallentscheidung wird durch die Wahl des geringeren Fehlers bezüglich des Abstands zwischen Zellmitte und Linie getroffen. Dazu wollen wir uns die Skizze in Abbildung 4.4 zu Hilfe nehmen.

Da die Auflösung eines festen endlichen Rasters bei Weitem nicht ausreicht, um jeden (mathematischen) Punkt einer Linie repräsentieren zu können, kann auch das Verfahren zum Zeichnen einer Linie nur versuchen, diese Punkte möglichst gut zu approximieren. Jedenfalls macht die Methode beim Wechsel in eine benachbarte Zelle immer einen Fehler. Wir führen daher für die Abweichung in y-Richtung der aktuellen Zellmitte von der Linie den Fehler ϵ ein. Wir befinden uns in der Zelle mit Mittelpunkt bei $(x + 0.5, y + 0.5)$. Wir besuchen als nächstes die Zelle mit

Algorithmus 3 InkrementellesRayCasting

Eingabe: Strahlgleichung $\vec{u} + t\vec{v}$ mit normiertem Richtungsvektor $\vec{v}$

Ausgabe: Koordinaten $(cell_x, cell_y)$, bei welcher der Strahl auf eine belegte Zelle trifft oder aus dem Gitter fällt

1: *// Initialisierungsphase*
2: $cell_x := u_x$; $cell_y := u_y$
3: **if** $(v_x > 0)$ **then**
4: $\quad step_x := 1$
5: $\quad t_x^{\max} := (\lceil u_x \rceil - u_x)/v_x$
6: **else if** $(v_x < 0)$ **then**
7: $\quad step_x := -1$
8: $\quad t_x^{\max} := (\lfloor u_x \rfloor - u_x)/v_x$
9: **else**
10: $\quad step_x := 1$
11: $\quad t_x^{\max} := \infty$
12: **end if**
13: **if** $(v_y > 0)$ **then**
14: $\quad step_y := 1$
15: $\quad t_y^{\max} := (\lceil u_y \rceil - u_y)/v_y$
16: **else if** $(v_y < 0)$ **then**
17: $\quad step_y := -1$
18: $\quad t_y^{\max} := (\lfloor u_y \rfloor - u_y)/v_y$
19: **else**
20: $\quad step_y := 1$
21: $\quad t_y^{\max} := \infty$
22: **end if**
23: $t_{\Delta x} := |v_x|^{-1}$; $t_{\Delta y} := |v_y|^{-1}$
24:
25: *// Traversierungsphase*
26: **repeat**
27: $\quad$ **if** $(t_x^{\max} < t_y^{\max})$ **then**
28: $\quad\quad t_x^{\max} := t_x^{\max} + t_{\Delta x}$
29: $\quad\quad cell_x := cell_x + step_x$
30: $\quad$ **else**
31: $\quad\quad t_y^{\max} := t_y^{\max} + t_{\Delta y}$
32: $\quad\quad cell_y := cell_y + step_y$
33: $\quad$ **end if**
34: **until** (ZelleBelegt($cell_x$,$cell_y$) $\vee$ ZelleAusserhalbGrid($cell_x$,$cell_y$))

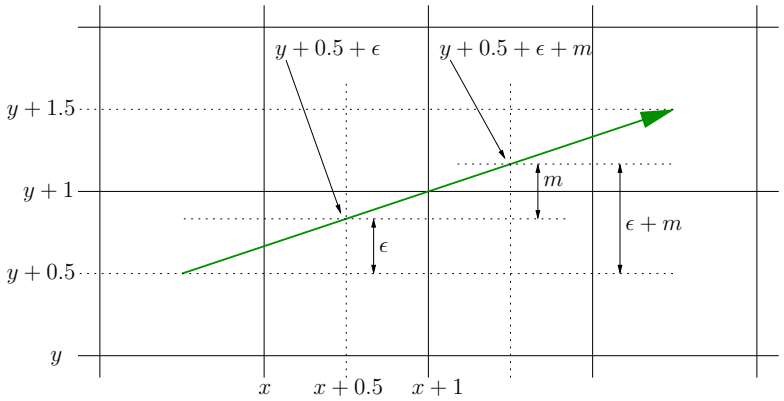


Abbildung 4.4.: Skizze zu Bresenhams Linien-Algorithmus.

Mittelpunkt bei $(x+1.5, y+0.5)$ – d. h. wir erhöhen die y-Koordinate nicht –, wenn folgende Gleichung erfüllt ist:

$$y + 0.5 + \epsilon + m < y + 1. \tag{4.4}$$

Andernfalls erhöhen wir auch die y-Koordinate. Damit wird der Fehler ϵ offensichtlich minimiert. Wir können nun auch den neuen Fehler ϵ_{new} aus dem alten berechnen. Je nachdem, welche der beiden möglichen Zellen als Nachfolger ausgewählt wird, ergeben sich zwei Möglichkeiten:

$$\epsilon_{new} = y + 0.5 + \epsilon + m - (y + 0.5) \tag{4.5}$$

$$\text{bzw.} \quad \epsilon_{new} = y + 0.5 + \epsilon + m - (y + 1.5). \tag{4.6}$$

Damit könnte man bereits einen Algorithmus angeben, der sukzessive die Zellen vom Anfangspunkt der Linie bis zum Endpunkt durchquert. Allerdings würden die zugehörigen Operationen auf Gleitkommazahlen basieren, was sich meist nachteilig auf die Performance auswirkt. Durch einfaches Umformen der Gleichungen (4.4)–(4.6) kann man jedoch einen Algorithmus hervorbringen, der nur auf Integer-Operationen beruht und folglich viel effizienter ist. Durch Subtraktion von $y+0.5$ und Multiplikation mit $2\Delta x$ auf beiden Seiten sowie anschließender Substitution durch $\epsilon' := \epsilon\Delta x$, erhält man aus Gleichung (4.4) der Reihe nach:

$$y + 0.5 + \epsilon + m < y + 1 \tag{4.7}$$
$$\Leftrightarrow \quad \epsilon + m < 0.5 \tag{4.8}$$
$$\Leftrightarrow \quad \epsilon + \frac{\Delta y}{\Delta x} < 0.5 \tag{4.9}$$
$$\Leftrightarrow \quad 2\epsilon\Delta x + 2\Delta y < \Delta x \tag{4.10}$$
$$\Leftrightarrow \quad 2(\epsilon' + \Delta y) < \Delta x. \tag{4.11}$$

Δx (bzw. Δy) ist dabei die Differenz der x-Koordinaten (bzw. y-Koordinaten) zwischen der Anfangs- und der Endzelle der Linie. Hierfür müssen wir jetzt natürlich tatsächlich einen Endpunkt außerhalb des Gitters berechnen, was aber mit Hilfe der Strahlgleichung kein Problem darstellt.

Analog zur Herleitung der Integer-Fallentscheidung in Gleichung (4.11), kann man auch die beiden Gleichungen (4.5) und (4.6) umformen, um die Berechnung des neuen Fehlers durch Integer-Operationen zu erreichen. Dazu betrachten wir im Folgenden die beiden möglichen Fälle (y-Koordinate ändert sich nicht oder wird erhöht) immer zusammen:

$$\epsilon_{new} = \epsilon + m \tag{4.12}$$
$$\text{bzw.} \quad \epsilon_{new} = \epsilon + m - 1 \tag{4.13}$$

Durch eine Multiplikation mit Δx und einer anschließenden Substitution mit $\epsilon' := \epsilon\Delta x$ erhält man letztendlich:

$$\epsilon'_{new} = \epsilon' + \Delta y \tag{4.14}$$
$$\text{bzw.} \quad \epsilon'_{new} = \epsilon' + \Delta y - \Delta x \tag{4.15}$$

Auf diese Weise gelangt man schließlich zu dem in Algorithmus 4 dargestellten Linien-Algorithmus, der auf Linien mit Steigungen m im Bereich $0 \leq m \leq 1$ begrenzt ist. Die Erweiterung auf alle möglichen Steigungen ist jedoch – wie bereits erwähnt – leicht möglich.

So schön diese beiden vorgestellten Verfahren in der Theorie erscheinen mögen, so haben sie doch einen gravierenden Nachteil: Sie sind in Anbetracht der großen Zahl an Strahlen, die untersucht werden müssen, sehr langsam – so langsam, dass es bei einer großen Zahl an Samples nicht möglich ist, das Ray-Casting *online*, also während des Lokalisierungsprozesses durchzuführen. Meistens wird dieses Problem durch eine Ray-Casting-Phase vor dem eigentlichen Lokalisierungsprozess umschifft. Dazu wird

Algorithmus 4 BresenhamRayCasting

Eingabe: Koordinaten (x_1, y_1) der Strahlursprungszelle und Koordinaten (x_2, y_2) der virtuellen Strahlendzelle

Ausgabe: Koordinaten (x, y), bei welcher der Strahl auf eine belegte Zelle trifft oder aus dem Gitter fällt

```
1:  // Initialisierungsphase
2:  ε' := 0
3:  y := y1
4:  Δx := x2 − x1
5:  Δy := y2 − y1
6:
7:  // Traversierungsphase
8:  for x := x1 to x2 do
9:    if (ZelleBelegt(x, y) ∨ ZelleAusserhalbGrid(x, y)) then
10:     Return()
11:   end if
12:   if (2(ε' + Δy) < Δx) then
13:     ε' := ε' + Δy
14:   else
15:     y := y + 1
16:     ε' := ε' + Δy − Δx
17:   end if
18: end for
```

von jeder freien Zelle aus (siehe Abb. 4.2) mit der gewünschten Winkelauflösung ein Ray-Casting durchgeführt und die dadurch gewonnenen Längen abgespeichert. Der Nachteil hierbei liegt darin, dass ein enormer Bedarf an Speicherplatz erforderlich ist, um all diese Längen zu speichern. Vor allem dann, wenn man bedenkt, dass ein sehr großer Teil dieser Zellen gar nicht oder nur einmal zu Beginn einer globalen Lokalisierung gebraucht wird, weil er in Bereiche fällt, in welche der Roboter nie gelangt, wünscht man sich ein Verfahren, das ein schnelles Ray-Casting während des Lokalisierungsprozesses möglich macht. Im nächsten Abschnitt werden wir daher ein Verfahren entwickeln, das durch eine Vorverarbeitung der Umgebungskarte diesem Wunsch gerecht wird.

4.2.3. Mask-Space-Leaping-Verfahren

Das Mask-Space-Leaping-Verfahren beruht auf der in der Arbeit von Yagel und Shi [1993] vorgestellten Space-Leaping-Methode. Es beschleunigt das Ray-Casting, indem es freie Bereiche von Zellen einfach überspringt. Die Distanz, um welche man mindestens von jeder freien Zelle der Karte aus risikolos – d. h. ohne versehentlich eine belegte Zelle zu übergehen – springen kann, entnimmt man einer zusätzlichen Information jeder Zelle. Dafür muss aus der Umgebungskarte lediglich eine Distanzkarte (engl. Euclidean Distance Map) extrahiert werden. Eine solche Karte enthält zu jeder freien Zelle die minimale euklidische Distanz zwischen dieser und der nächsten belegten Zelle. Abbildung 4.5 repräsentiert graphisch die Distanzkarte zur Umgebungskarte, die in Abbildung 4.2 vorgestellt wurde.

Diese Distanzkarte kann auf verschiedene Arten gewonnen werden. Der naive Ansatz besteht darin, für jede freie Zelle alle belegten Zellen der Reihe nach bis zum Ende durchzugehen, und immer dann, wenn eine kleinere euklidische Distanz als die bisher gefundene auftaucht, eine Aktualisierung der Distanzinformation der gegenwärtigen freien Zelle durchzuführen. Oder man benutzt eine Art von *Floodfill*-Verfahren, das ausgehend von den belegten Zellen unter Verwendung einer *Queue*-Datenstruktur schrittweise alle Zellen expandiert. Eine äußerst bemerkenswerte Technik ist auch das von Meijster et. al. [2000] vorgestellte Verfahren, das für das Anfertigen einer Distanzkarte nur linearen Aufwand benötigt (linear in der Anzahl der Grid-Zellen). Wir werden den entsprechenden Algorithmus in Abschnitt 4.2.4 kennenlernen.

Angenommen, wir befinden uns bei unserem Ray-Casting-Prozess in einer freien Zelle mit den Koordinaten (x, y) und entnehmen unserer Distanzkarte, dass die nächste belegte Zelle d Zelleinheiten entfernt ist. Dann können wir ausgehend von unserer Zelle doch gefahrlos um eben diese Distanz auf einem Strahl mit beliebiger Richtung springen, ohne eine belegte Zelle zu übersehen. Wir müssen nur immer die Zelle ausfindig machen, in welcher man beim Zurücklegen der Distanz d auf dem Strahl landet. Und genau hier bringen wir eine Art *Maske* ins Spiel, die uns das Identifizieren

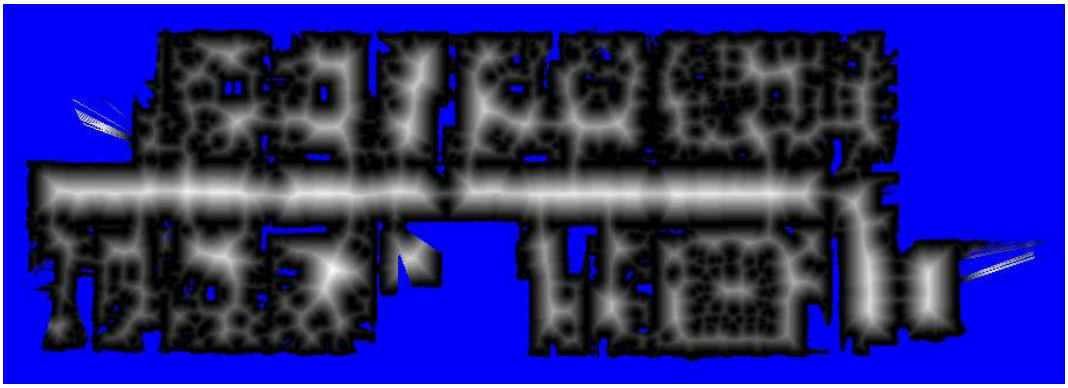
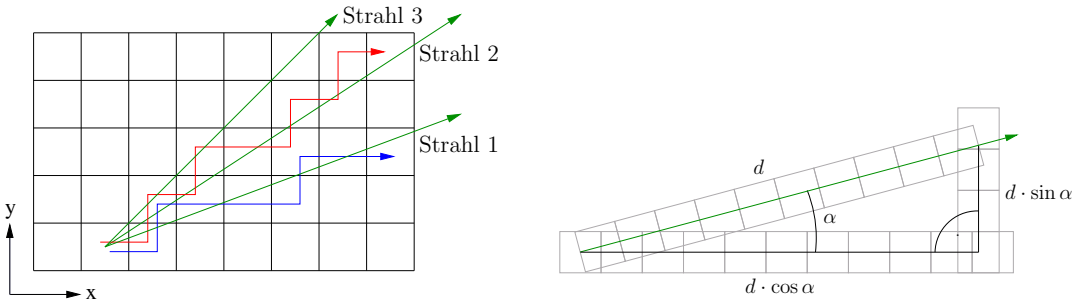


Abbildung 4.5.: Gezeigt wird ein Graubild der Karte minimaler euklidischer Distanzen. Je dunkler die Färbung, desto näher ist die entsprechende Zelle einer belegten Zelle, d. h. einem Hindernis. Deutlich zu erkennen ist auch die Ähnlichkeit der weißen Bereiche mit einem Voronoi-Diagramm. Der blaue Bereich steht wieder für Zellen mit unbekanntem Belegungsgrad.

der richtigen Zielzelle erleichtern soll. Zuerst diskretisieren wir die möglichen Orientierungen eines Strahls derart, dass die gewünschte Genauigkeit erreicht wird (z. B. 1°-Winkelauflösung). In einer Vorberechnung fertigen wir uns daraufhin eine Maske an, die für jede diskrete Strahlorientierung eine Sequenz der von dem entsprechenden Strahl traversierten Zellen in Form von relativen Zellindizes enthält. Abbildung 4.6 (a) zeigt exemplarisch für zwei Strahlen (Strahlen 1 und 2) bei einer bestimmten Winkelauflösung die jeweils traversierten Zellen, deren Koordinatendifferenzen zur Anfangszelle in die Maske aufgenommen werden. Dies kann z. B. mit einem der in den Abschnitten 4.2.1 und 4.2.2 behandelten Ray-Casting-Verfahren durchgeführt werden. Die Maske dient somit als sogenannte *Lookup*-Tabelle, und liefert für eine gegebene Strahlrichtung α und für einen gegebenen Index i immer zwei Werte, um die man die x- oder y-Koordinaten der Ursprungszelle inkrementieren oder dekrementieren muss, um in die i-te vom Strahl traversierte Zelle zu kommen. Verwendet man für eine solche Sequenz eine *Array*-Datenstruktur, stellt sich nun die Frage, wie man denn nun innerhalb dieses Arrays bei gegebener Distanz d an die richtige Stelle i im Array, an welcher die Koordinatenwerte der gewünschten Zelle stehen, herankommt. Die Beantwortung dieser Frage hängt natürlich von dem zu Grunde liegenden Traversierverfahren ab. Denn beim Bresenham-Verfahren sind „Übereck"-Sprünge (siehe Abb. 4.6 (a), Strahl 3) zu erwarten, während beim inkrementellen Verfahren stets nur in eine Richtung weitergegangen wird (siehe Abb. 4.3), d. h. diese Art von Sprüngen gar nicht erst auftaucht. Um bei dem erstgenannten Verfahren auf Nummer sicher zu gehen, bietet es sich an, dafür die Winkelfunktionen sin() und cos() einzusetzen, wie in Abbildung 4.6 (b) gezeigt wird:

$$i = \lfloor d \cdot \max(|\cos(\alpha)|, |\sin(\alpha)|) \rfloor. \tag{4.16}$$



(a) Traversieren der verschiedenen Strahlen

(b) Berechnung des Index i

Abbildung 4.6.: (a) Gezeigt werden die von verschiedenen Strahlen traversierten Zellen. Der blaue Linienzug zeigt die von Strahl 1 traversierten Zellen, der rote die von Strahl 2. Strahl 3 verdeutlicht sogenannte „Übereck"-Sprünge. (b) Die Skizze zeigt, wie man bei einer Bresenham-Maske aus der Distanz d, die man für jede Zelle der Distanzkarte entnehmen kann, den Index i im Array gewinnt, ohne das Risiko einzugehen, zu weit zu springen.

Damit gewährleistet man, dass auch im Falle von „Übereck"-Sprüngen des Strahls nicht mehr Zellen im Array übersprungen werden als erlaubt.

Beim inkrementellen Verfahren existieren keine Übereckspränge. Da in jedem Schritt nur um genau eine Zelle entweder in x- oder in y-Richtung gegangen wird, ergibt sich der gesuchte Index durch die Manhattan-Distanz:

$$i = \lfloor d \cdot |\cos(\alpha)| \rfloor + \lfloor d \cdot |\sin(\alpha)| \rfloor. \tag{4.17}$$

Um dies zu begründen, gehen wir von einer ausreichend großen Winkelauflösung der Maske aus. Nun besteht die Möglichkeit, dass ausgerechnet jene belegte Zelle, die den Eintrag d in der aktuellen Zelle der Distanzkarte verursacht hat, wirklich in Richtung des Strahls mit der Orientierung α liegt. Aus diesem Grund müssen wir bei unserem Space-Leaping-Verfahren stets gewährleisten, dass die durch den Index i angesprochene Zelle nicht weiter von der jeweils aktuellen Zelle entfernt ist als die genannte belegte Zelle.

Sei (x', y') die Zelle, in die wir über den Index i springen wollen, dann ist die Distanz d' zwischen dieser und der aktuellen Zelle gerade gleich dem Pythagoras der Koordinatendifferenzen:

$$d' = \sqrt{(\Delta x')^2 + (\Delta y')^2}. \tag{4.18}$$

Es muss also gewährleistet werden, dass $d' \leq d$ gilt. Die Distanz d lässt sich wie folgt in seine x- und y-Komponente zerlegen:

$$
\begin{aligned}
d &= \sqrt{(d \cdot |\cos(\alpha)|)^2 + (d \cdot |\sin(\alpha)|)^2} && (4.19)\\
&\geq \sqrt{\lfloor d \cdot |\cos(\alpha)| \rfloor^2 + \lfloor d \cdot |\sin(\alpha)| \rfloor^2} && (4.20)\\
&= \sqrt{(\Delta x')^2 + (\Delta y')^2} && (4.21)\\
&= d' && (4.22)
\end{aligned}
$$

Die vorletzte Gleichung gilt, da die ganzen Zahlen $\lfloor d \cdot |\cos(\alpha)| \rfloor$ und $\lfloor d \cdot |\sin(\alpha)| \rfloor$ den Komponenten $|\Delta x'|$ und $|\Delta y'|$ entsprechen, die wir für die Bestimmung des Index i in Gleichung (4.17) herangezogen haben.

Man kann sich nun aussuchen, ob man der Maske das inkrementelle Ray-Casting oder das Bresenham-Ray-Casting zu Grunde legen will. Wir werden für das Erstellen der Maske das inkrementelle Ray-Casting bevorzugen, da das Mask-Space-Leaping-Ray-Casting dadurch etwas genauer ist.

Wenn die neue Zelle nicht belegt ist, und sich nicht außerhalb der Karte befindet, kann man ausgehend von dieser wieder mit Hilfe der Distanzkarte und der Maske weiterverfahren. Es bleibt allerdings noch zu erwähnen, dass man, falls d unter eine bestimmte Schwelle fällt, mit einem schrittweisen Traversieren fortfahren muss, damit man nicht wegen $i = 0$ auf der Stelle springt. Das Mask-Space-Leaping-Ray-Casting reduziert während des Lokalisierungsprozesses deutlich die Anzahl der besuchten Zellen. In Abbildung 4.7 werden die drei vorgestellten Verfahren miteinander bezüglich der Anzahl traversierter Zellen beim Verfolgen der in Abbildung 3.3 (a) gezeigten Trajektorie verglichen.

4.2.4. Berechnung der Distanzkarte

Da die Distanzkarte sowohl für das Mask-Space-Leaping-Ray-Casting als auch für das Endpunkt-Sensormodell, welches in Abschnitt 4.6 diskutiert wird, eine grundlegende Rolle spielt, werden wir in diesem Abschnitt den für die effiziente Erzeugung solcher Karten verwendeten Algorithmus von Meijster et. al. [2000] vorstellen. Er basiert auf dem von Hesselink et. al. [1999] entwickelten Algorithmus für Distanztransformationen und bietet neben einer linearen Laufzeit auch noch weitere Vorteile, wie z. B. die Parallelisierbarkeit bei der Verwendung mehrerer Prozessoren oder die einfache Erweiterung auf mehrdimensionale Distanztransformationen, was vor allem für 3D-Anwendungen von großem Interesse sein dürfte.

Die Problembeschreibung für die euklidische Distanztransformation (EDT) lautet wie folgt: Gegeben sei ein zweidimensionales Gitter B mit Gitterpunkten (x, y), das durch eine zweidimensionale binäre Karte b der Dimension $m \times n$ repräsentiert

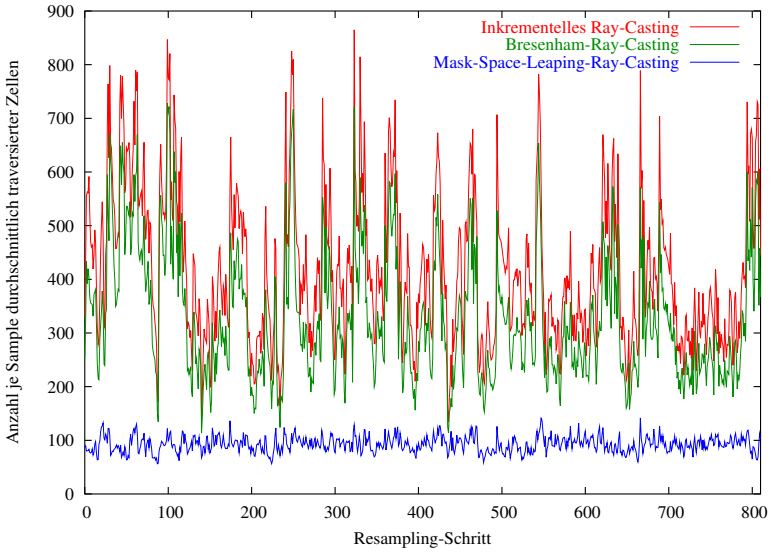


Abbildung 4.7.: Gezeigt werden die von jedem der drei vorgestellten Verfahren je Sample durchschnittlich traversierten Zellen. Das Überspringen von Zellen verhilft dem Mask-Space-Leaping-Ray-Casting zu einer guten Peformance.

wird; gesucht ist die zweidimensionale Karte $d[x, y] = \sqrt{\mathrm{EDT}(x, y)}$ exakter minimaler euklidischer Distanzen, die wie folgt definiert ist:

$$\begin{aligned} \mathrm{EDT}(x, y) := \min\{(x - i)^2 + (y - j)^2 \mid\ & 0 \leq i < m \\ & \wedge\ 0 \leq j < n \\ & \wedge\ b[i, j] \text{ belegt}\}. \end{aligned} \tag{4.23}$$

Der Algorithmus für den zweidimensionalen Fall besteht aus zwei Phasen, wobei sich jede dieser Phasen wiederum aus zwei sogenannten *Scans*, einem Vorwärtsscan und einem Rückwärtsscan, zusammensetzt. Die erste Phase dient dazu, das vorliegende binäre Bild – bzw. die vorliegende binäre Karte belegter und freier Zellen – spaltenweise abzusuchen, worauf in der zweiten Phase eine zeilenweise Abtastung erfolgt. Die Berechnung der Karte exakter minimaler euklidischer Distanzen wird zunächst auf die Berechnung einer Karte edt_{sqr}, welche die Quadrate der minimalen euklidischen Distanzen enthält, reduziert:

$$edt_{\mathrm{sqr}}[x, y] = \mathrm{EDT}(x, y).$$

Nach dem Terminieren des Algorithmus kann daraus leicht die gesuchte Distanzkarte d gewonnen werden, indem aus allen Einträgen die Wurzel gezogen wird.

Der Algorithmus beginnt in der ersten Phase zunächst damit, jede Spalte C_x mit festem x einzeln zu durchsuchen. Für jede Zelle (x, y) der Spalte C_x wird die Distanz $G(x, y)$ zur nächsten belegten Zelle in der gleichen Spalte C_x ermittelt. In der darauffolgenden zweiten Phase wird jede Zeile R_y separat gescannt und für jede Zelle (x, y) in R_y das Minimum von $(x - x')^2 + G(x', y)^2$ berechnet, wobei (x', y) sich über die Zeile R_y erstreckt. Gleichung (4.23) nimmt nun also folgende Gestalt an:

$$\mathrm{EDT}(x, y) := \min\left\{(x - i)^2 + G(i, y)^2 \mid 0 \leq i < m\right\}, \tag{4.24}$$

wobei

$$G(i, y) := \min\left\{|y - j| \mid 0 \leq j < n \ \wedge \ b[i, j] \text{ belegt}\right\}. \tag{4.25}$$

Die erste Phase des Algorithmus beschäftigt sich mit der Bestimmung der Funktion G. Um die Betragsfunktion zu umgehen, kann man G wie folgt in zwei Funktionen G_{top} und G_{bottom} zerlegen:

$$G_{\mathrm{top}}(i, y) = \min\left\{y - j \mid 0 \leq j \leq y \ \wedge \ b[i, j] \text{ belegt}\right\} \tag{4.26}$$

$$G_{\mathrm{bottom}}(i, y) = \min\left\{j - y \mid y \leq j < n \ \wedge \ b[i, j] \text{ belegt}\right\}. \tag{4.27}$$

Damit teilt man die Bestimmung von G in die Ausführung zweier Scans auf. Die berechneten Werte für G_{top} sollen in einem zweidimensionalen Array g gespeichert werden. Sicherlich gilt $G_{\mathrm{top}}(i, y) = 0$, falls Zelle $b[i, y]$ belegt ist, und $G_{\mathrm{top}}(i, y) = G_{\mathrm{top}}(i, y - 1) + 1$ sonst. Handelt es sich bei der aktuellen Zelle jedoch um die erste Zelle einer Spalte, d. h. im Fall von $y = 0$, dann steht $G_{\mathrm{top}}(i, y - 1)$ leider nicht zur Verfügung und $G_{\mathrm{top}}(i, y)$ nimmt den Wert ∞ an. Nach dem Scannen der Spalte von oben nach unten erfolgt der zweite Scan von unten nach oben, in welchem die Werte der Funktion G_{bottom} ermittelt werden. Gleichzeitig werden diese aber mit dem Ergebnis aus dem vorhergehenden Scan verglichen und auf diese Weise die Werte von $G(x, y)$ direkt berechnet. Diese erste Phase ist in Algorithmus 5 dargestellt. Enthält die zugrundeliegende Binärkarte mindestens eine belegte Zelle, so können die entsprechenden Zellen in g statt mit ∞ auch mit $m + n$ initialisiert werden. Die Komplexität der ersten Phase beträgt $O(m \cdot n)$.

Die zweite Phase des Algorithmus transformiert nun die Hilfskarte g in die gewünschte Distanzkarte edt_{sqr}. Dazu wird die Karte g zeilenweise gescannt. Obwohl der vorgestellte Algorithmus generalisiert für verschiedene Distanz- bzw. Metrikdefinitionen entwickelt wurde, werden wir uns hier auf die für unsere euklidische Distanzkarte relevante euklidische Distanzdefinition beschränken (siehe Gleichung

Algorithmus 5 ErstePhaseEDT

Eingabe: Binäre $m \times n$-Karte b
Ausgabe: $m \times n$-Karte g, welche die Werte der Funktion $G(x, y)$ enthält

```
1:  for x := 0 to m − 1 do
2:
3:    // Erster Scan
4:    if (b[x, 0] belegt) then
5:      g[x, 0] := 0
6:    else
7:      g[x, 0] := ∞
8:    end if
9:    for y := 1 to n − 1 do
10:     if (b[x, y] belegt) then
11:       g[x, y] := 0
12:     else
13:       g[x, y] := g[x, y − 1] + 1
14:     end if
15:   end for
16:
17:   // Zweiter Scan
18:   for y := n − 2 downto 0 do
19:     if (g[x, y + 1] < g[x, y]) then
20:       g[x, y] := g[x, y + 1] + 1
21:     end if
22:   end for
23:
24: end for
```

(4.24)). Das bedeutet, dass wir für ein festes y nun in der zweiten Phase die folgenden Werte berechnen müssen:

$$\mathrm{EDT}(x, y) := \min \{ f(x, i) \mid 0 \leq i < m \}, \tag{4.28}$$

wobei in unserem speziellen Fall gilt:

$$f(x, i) := (x - i)^2 + G(i, y)^2. \tag{4.29}$$

Zum besseren Verständnis dieses Algorithmus wird von Meijster et. al. [2000] eine geometrische Interpretation dieser Gleichung eingeführt. Für jedes i mit $0 \leq i < m$ sei mit F_i die folgende Funktion auf dem Intervall $[0; m-1] \subset \mathbb{R}$ definiert:

$$F_i \ : \ x \mapsto f(x, i). \tag{4.30}$$

Der Graph von F_i ist in unserem speziellen Fall euklidischer Distanzen eine Parabel mit Scheitelpunkt bei $(i, G(i, y))$. Die Funktionswerte der Funktion EDT ergeben sich nun durch die sogenannte *untere Hülle* der Menge $\{F_i \mid 0 \leq i < m\}$ ausgewertet an den ganzzahligen Koordinaten. Wie in Abbildung 4.8 (a) deutlich zu erkennen ist, setzt sich diese untere Hülle von links nach rechts aus einzelnen Kurvensegmenten $s[0], s[1], \ldots, s[q]$ zusammen, deren Projektion auf die x-Achse eine Partitionierung des Intervalls $[0; m)$ in verschiedene Regionen darstellt. In dem dritten Scan des Verfahrens wird nun von links nach rechts die Menge der Regionen inkrementell ermittelt, worauf in einem vierten Scan von rechts nach links daraus nur noch die gewünschten Distanzwerte berechnet werden müssen.

Dazu wird für das zeilenweise Scannen die obere Grenze m in Gleichung (4.24) durch eine verschiebbare Variable u ersetzt und die Funktion $F_{\mathrm{left}}(x, u)$ wie folgt definiert:

$$F_{\mathrm{left}}(x, u) := \min \{ f(x, i) \mid 0 \leq i < u \}. \tag{4.31}$$

Dadurch schränken wir die zweidimensionale Karte b auf die Fläche links von u ein. Es gilt jetzt offensichlich $\mathrm{EDT}(x, y) = F_{\mathrm{left}}(x, m)$ (vgl. Gleichung (4.28)). Wenn nun bei gegebener Schranke $u > 0$ bei einem Index $i_{\min}$ die Funktion $f(x, i)$ einen minimalen Wert annimmt, bezeichnen wir $i_{\min}$ als *Minimumindex*. Da es für eine Funktion theoretisch mehrere Minimumindizes geben kann, nehmen wir davon den kleinsten mit $0 \leq i_{\min} < u$, der die Beziehung $f(x, i_{\min}) \leq f(x, i)$ für alle i mit $0 \leq i < u$ erfüllt, und definieren ein weiterers Hilfsarray $I_{\min}$:

$$\begin{aligned} I_{\min}(x, u) := \min\{ i_{\min} \mid \ & 0 \leq i_{\min} < u \\ & \wedge\, f(x, i_{\min}) \leq f(x, i) \ \forall\, i, 0 \leq i < u \}. \end{aligned} \tag{4.32}$$

Algorithmus 6 ZweitePhaseEDT

Eingabe: $m \times n$-Karte g, welche die Werte der Funktion $G(x, y)$ enthält; Distanzfunktion $f(x, y)$; Separierungsfunktion Sep

Ausgabe: $m \times n$-Karte edt_{sqr}, welche die Quadrate minimaler Distanzen enthält

```
1:  for y := 0 to n − 1 do
2:
3:    // Initialisierung
4:    q := 0;  s[0] := 0;  r[0] := 0
5:
6:    // Dritter Scan
7:    for u := 1 to m − 1 do
8:      while (q ≥ 0 ∧ f(r[q], s[q]) > f(r[q], u)) do
9:        q := q − 1
10:     end while
11:     if (q < 0) then
12:       q := 0;  s[0] := u
13:     else
14:       w := Sep(s[q], u) + 1
15:       if (w < m) then
16:         q := q + 1;  s[q] := u;  r[q] := w
17:       end if
18:     end if
19:   end for
20:
21:   // Vierter Scan
22:   for u := m − 1 downto 0 do
23:     edt_sqr[u, y] := f(u, s[q])
24:     if (u = r[q]) then
25:       q := q − 1
26:     end if
27:   end for
28:
29: end for
```

Nun gilt wegen $F_{\text{left}}(x,u) = f(x, I_{\min}(x,u))$ auch:

$$\text{EDT}(x,y) = f(x, I_{\min}(x,m)). \tag{4.33}$$

Was bleibt, ist die Berechnung von $I_{\min}(x,m)$. Dazu definieren wir die Menge $S(u)$ als die Menge kleinster Minimalindizes eines horizontalen Scans von links nach rechts sowie mit $R(i_{\min}, u)$ jeweils eine Menge von Punkten, die den selben Minimalindex $i_{\min}$ haben:

$$S(u) := \{I_{\min}(x,u) \mid 0 \le x < m\} \tag{4.34}$$

$$R(i_{\min}, u) := \{x \mid 0 \le x < m \wedge I_{\min}(x,u) = i_{\min}\} \quad \text{für} \quad 0 \le i_{\min} < u. \tag{4.35}$$

Die Intervalle $S(u)$ stellen nichtleere Teilmengen von $[0;u)$ dar. Definiert man nun die oben erwähnten Regionen als die Mengen $R(i_{\min}, u)$, die nicht leer sind, dann erhält man letztendlich – d. h. wenn $u = m$ erreicht wird – mit den Regionen die gewünschte Partitionierung von $[0;m)$. Damit ist man dann auch am Ziel. Denn für ein $x \in R(i_{\min}, m)$ gilt nach Gleichung (4.35) $I_{\min}(x,m) = i_{\min}$ und damit für die gesuchte Distanz: $\text{EDT}(x,y) = f(x, i_{\min})$.

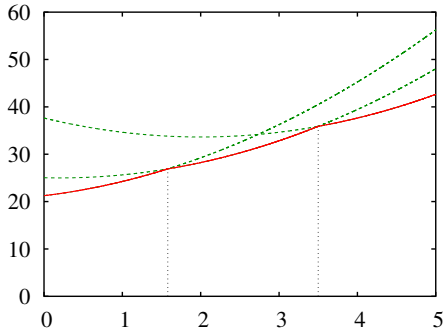
Zunächst werden wir aber klären, was passiert, wenn beim Scannen eine neue Kurve F_u hinzukommt. Dazu nehmen wir an, dass bereits $q+1$ Kurvensegmente vorliegen und die Menge der kleinsten Minimalindizes soweit wie die folgende aussieht: $S(u) = \{s[0], s[1], \ldots, s[q]\}$. Kommt nun die neue Kurve F_u hinzu, sind für die Lage des Graphen von F_u drei Fälle möglich (vgl. Abbildungen 4.8 (b)–(d)):

1. F_u liegt gänzlich oberhalb der soweit ermittelten unteren Hülle $[0; m-1]$. Folglich ist die Menge $R(u, u+1)$ leer und $S(u)$ ändert sich nicht: $S(u+1) = S(u)$.

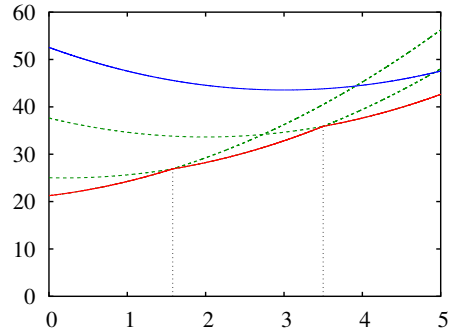
2. F_u liegt komplett unterhalb der bisherigen Hülle. Damit löst die Kurve F_u die bisherige untere Hülle ab ($S(u) = \{u\}$) und $[0,m)$ wird durch die einzige Region $R(u, u+1)$ partitioniert.

3. F_u und die untere Hülle $[0; m-1]$ schneiden sich. In diesem Fall können bereits bestehende Regionen verkleinert werden oder sogar ganz wegfallen. Auf jeden Fall gibt es eine neue Region $R(u, u+1)$.

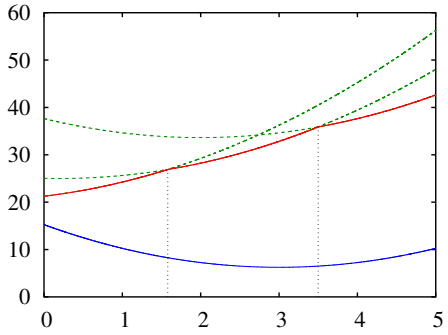
Doch wie kann man nun rechnergestützt diese Fallunterscheidung durchführen? Man beginnt damit, nacheinander von rechts nach links die Regionen $l = q, q-1, \ldots$ durchzugehen und an jedem Anfangspunkt $r[l]$ der aktuellen Region l die Funktionswerte von F_u und F_l miteinander zu vergleichen. Man sucht also nach dem ersten $l = l^*$, bei dem $F_u(r[l^*]) \ge F_{s[l^*]}(r[l^*])$ gilt. Dann nämlich ist F_u gerade nicht mehr der kleinste Minimalindex bei $r[l^*]$ und es muss einen Schnitt zwischen F_u und $F_{s[l^*]}$



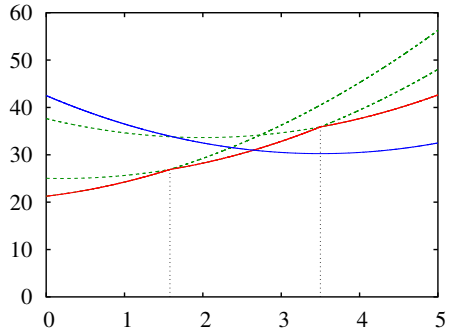
(a) Untere Hülle der EDT



(b) oberhalb



(c) unterhalb



(d) Schnitt

Abbildung 4.8.: (a) Gezeigt wird die sogenannte *untere Hülle* (roter Graph) der Kurven F_i für $0 \leq i < m$. Die Abbildungen (b)–(d) zeigen die für die Lage einer neuen Kurve F_u (blauer Graph) bezüglich der unteren Hülle möglichen drei Fälle. (b) Die neue Kurve F_u liegt oberhalb der unteren Hülle. (c) Die Kurve F_u liegt unterhalb und wird damit zur neuen unteren Hülle. (d) F_u schneidet die untere Hülle.

in der Region von l^* geben. Die x-Koordinate dieses Schnittpunktes sei x^*. Wenn nun $l^* = q$ und $x^* \geq m$ gilt, haben wir es mit dem ersten Fall zu tun, da der Schnittpunkt außerhalb der Fläche liegt. Wenn $l^* < 0$ gilt, handelt es sich um den zweiten Fall und ansonsten um den dritten.

Eine Separierungsfunktion $\mathrm{Sep}(i, u)$ soll zur Berechnung von x^* dienen. Die Funktion $\mathrm{Sep}(i, u)$ soll nämlich die erste ganze Zahl, die größer oder gleich der x-Koordinate des Schnittpunktes zwischen F_u und F_i mit $i < u$ ist, liefern:

$$x \leq \mathrm{Sep}(i, u) \Leftrightarrow F_i(x) \leq F_u(x). \tag{4.36}$$

Dann ergibt sich nämlich x^* wie folgt: $x^* = \mathrm{Sep}(s[l^*], u)$. In der Separierungsfunktion spiegelt sich die Distanzdefinition wieder. Für die euklidische Distanz ergibt sich Sep folgendermaßen (vgl. Definition von F_i in Gleichung (4.30) und Gleichung (4.29)):

$$\begin{aligned}
F_i(x) \leq F_u(x) &\Leftrightarrow (x-i)^2 + G(i,y)^2 \leq (x-u)^2 + G(u,y)^2 && (4.37)\\
&\Leftrightarrow x^2 - 2xi + i^2 + G(i,y)^2 \leq x^2 - 2xu + u^2 + G(u,y)^2 && (4.38)\\
&\Leftrightarrow 2(u-i)x \leq u^2 - i^2 + G(u,y)^2 - G(i,y)^2 && (4.39)\\
&\Leftrightarrow x \leq (u^2 - i^2 + G(u,y)^2 - G(i,y)^2)\,\mathrm{div}(2(u-i)). && (4.40)
\end{aligned}$$

Dabei steht in Gleichung 4.40 der Operator div für die ganzzahlige Division (*Integer*-Division). Die Separierungsfunktion Sep für die euklidische Distanz lautet jetzt wie folgt:

$$\mathrm{Sep}(i, u) := (u^2 - i^2 + G(u,y)^2 - G(i,y)^2)\,\mathrm{div}(2(u-i)). \tag{4.41}$$

Kommen wir nun wieder auf den Algorithmus zurück. Der dritte Scan sorgt für die Partitionierung des Intervalls $[0, m)$ in die einzelnen Regionen, welche im Anschluss daran von dem vierten Scan dazu benutzt werden, die Werte von EDT zu berechnen. Da es sich bei dem Algorithmus um einen generellen Algorithmus handelt, der gleichermaßen auch zur Berechnung sogenannter *Manhattan*- oder *Chessboard*-Distanzkarten eingesetzt werden kann, enthält Algorithmus 6, der die zweite Phase darstellt, keine unmittelbare Definition für die euklidische Distanz, sondern basiert stattdessen ganz allgemein auf der Distanzfunktion $f(x, y)$ und der entsprechenden Separierungsfunktion Sep, für die – je nachdem, welche Metrik verwendet wird – die entsprechenden Terme eingesetzt werden müssen. Mit einer Komplexität von $O(m \cdot n)$ für den gesamten Algorithmus und der ausschließlichen Verwendung von Integer-Operationen erhält man hiermit ein schnelles und effizientes Verfahren für das Berechnen von Distanzkarten.

Mit diesem Verfahren zur Erzeugung von Distanzkarten und dem in Abschnitt 4.2.3 vorgestellten Mask-Space-Leaping-Verfahren bieten sich neue Möglichkeiten für auf Ray-Casting basierende Lokalisierungsverfahren. Beispielsweise kann zur Laufzeit bei einer Änderung der zugrundeliegenden Umgebungskarte schnell eine neue Distanzkarte und eine neue Maske angefertigt werden, ohne eine zeitaufwändige und vor allem enormen Speicherplatz erfordernde Vorberechnung aller erwarteter Längen durchführen zu müssen. Auch das Umschalten zwischen Umgebungskarten verschiedener Auflösungen wird hiermit stark vereinfacht.

Nun, da wir ein effizientes Ray-Casting-Verfahren zur Verfügung haben, können wir wieder auf das Problem der Bewertung der Samples eingehen. Wir haben es in Abschnitt 4.1 auf den Vergleich zweier Strahllängen miteinander reduziert. Da die gemessenen Längen aber von verschiedenen äußeren Faktoren beeinflusst werden, beschäftigen wir uns im nächsten Abschnitt eingehender mit Sensormessungen und etwaiger Einflüsse, die vor allem durch die unzugängliche und dynamische Umgebung, in welcher der Roboter operiert, zustande kommen.

4.3. Äußere Einflüsse auf Sensormessungen

Unglücklicherweise liefern selbst die präzisesten Vertreter der Klasse der Abstandssensoren fehlerhafte Messergebnisse. Vergleicht man nämlich den typischen Scan eines Laserscanners mit der Geometrie der Umgebung (siehe Abb. 4.1), beobachtet man neben ziemlich präzisen Werten auch Längen, die verkürzt sind oder der maximalen Reichweite des Scanners entsprechen, sowie Werte, die völlig willkürlich erscheinen. Diese Fehler werden durch verschiedene Faktoren hervorgerufen. Reflexionen an bestimmten Oberflächen, die Empfindlichkeit und der Öffnungswinkel des Sensors sowie dynamische Hindernisse in Form von Personen sind nur einige davon. Aber auch die Umgebungskarte vermag die komplexe, reale Welt durch die Diskretisierung in Form von Zellen nur unzulänglich repräsentieren. Man denke hierbei nur an Glasflächen, Geländer etc. Je besser ein Sensormodell diese Fehlerquellen berücksichtigt, desto robuster und zuverlässiger wird die Lokalisierung und desto bessere Ergebnisse können erwartet werden. Es scheint jedoch praktisch unmöglich, ein absolut exaktes Sensormodell zu realisieren. Denn meist sind gar nicht alle Einflussfaktoren – wie beispielsweise die Oberflächenbeschaffenheit oder die Durchlässigkeit von Glas – bekannt. Dennoch werden wir im folgenden Abschnitt versuchen, verschiedene Ursachen für die Messfehler zu identifizieren.

4.3.1. Fehlerklassifizierung

Betrachten wir noch einmal Abbildung 4.1, so wird deutlich, dass die Messwerte durch verschiedene Einflüsse verfälscht werden. Manche Abstände sind zu kurz (engl. Short Readings) oder entsprechen sogar der maximalen Reichweite (engl. Max Range Readings) des Sensors. Die Gründe dafür lassen sich gemäß [Fox et. al. 1999b; Choset et. al. 2004] in die folgenden vier Ursachengruppen gliedern:

1. **Ungenauigkeit und Rauschen**
   Die Ungenauigkeit kommt durch das geringe Auflösungsvermögen des Sensors zustande. Vor allem aber limitiert die Auflösung der Umgebungskarte, in der das Ray-Casting durchgeführt wird, die Genauigkeit. Sowohl bei Laserscannern als auch bei Ultraschallsensoren kommt noch aufgrund der radialen bzw. kegelförmigen Signalemission das mit zunehmender Distanz abnehmende Auflösungsvermögen in Form von Rauschen hinzu.

2. **Phantommessungen**
   In diese Kategorie fallen Messwerte, die durch nicht in der Umgebungskarte verzeichnete Objekte oder durch sogenannte *Cross-Talks*[1] verursacht werden. Gerade in stark dynamischen Umgebungen gewinnt diese Fehlerquelle erheblich an Bedeutung. Sich bewegende Personen in der Reichweite des Roboters, die natürlich nicht in der Umgebungskarte verzeichnet sind, verkleinern die gemessenen Entfernungswerte derart, dass die Wahrscheinlichkeit, eine kleinere Entfernung zu messen als die gemäß der Umgebungskarte zu erwartende, mit kleineren Entfernungen zunimmt. Diese Phantommessungen treten sehr häufig bei Sonar-Sensoren auf, eher selten bei Laserscannern.

3. **Max-Range-Messungen**
   Diese Fehler treten meist dann auf, wenn Hindernisse nicht erkannt werden können. Das kann der Fall sein, wenn Ultraschallwellen zu flach auf Hindernisse auftreffen oder wenn Laserscannerstrahlen von Oberflächen absorbiert, vollständig in eine andere Richtung reflektiert (Spiegel) oder transmittiert (Glas) werden.

4. **Zufällige Fehler**
   Leider gibt es neben den o. g. Effekten in den Abstandsdaten auch völlig unerwartete Messergebnisse. Diese sollen durch die Gruppe der zufälligen Fehler repräsentiert werden.

Will man nun alle diese Einflüsse im Sensormodell berücksichtigen, muss man ein universelles Modell entwickeln, weil man nie sicher weiß, durch welche der o. g. Kategorien eine Messung beeinflusst wird. Dies verdeutlicht auch Abbildung 4.9. Die gra-

[1] Bei der Verwendung von mehreren auf derselben Frequenz arbeitenden Sensoren kann es bei einem Sensor zu einer irrtümlichen Verwendung eines Fremdsignals zur Berechnung des Abstands kommen, was als Cross-Talk bezeichnet wird und meist zu einem falschen Abstandswert führt.

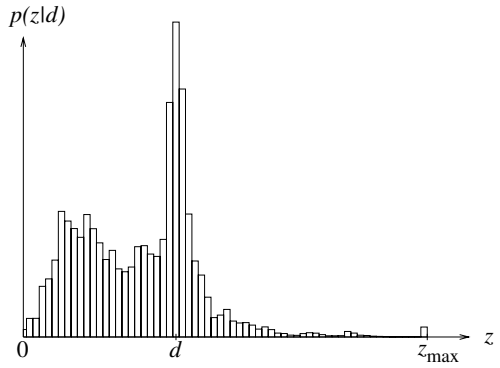


Abbildung 4.9.: Gezeigt wird das *A-Posteriori*-Wahrscheinlichkeits-Histogramm $p(z|d)$ für die zu erwartende Distanz von $d = 5\,\text{m}$ bei einer Klassenbreite von $0.2\,\text{m}$ und einem Max-Range von $12.8\,\text{m}$. Die zugrunde liegenden Daten wurden beim Tracking in der Büroumgebung des Gebäude 079 aufgezeichnet.

phische Darstellung zeigt die bedingte Wahrscheinlichkeit $p(z|d)$ in Form eines Histogramms. Die zu Grunde liegenden Distanzen entstammen einem ausgiebigen Positionsverfolgungsprozess in der in Abbildung 4.2 gezeigten Umgebung. Deutlich ist zu erkennen, wie die o. g. Einflussfaktoren das Histogramm prägen.

In den Arbeiten von Fox et. al. [1999b] und Choset et. al. [2004] werden die einzelnen Fehler daher durch (diskrete) Wahrscheinlichkeitsdichten approximiert (siehe Abb. 4.10) und zu einer gemischten, universellen Wahrscheinlichkeitsdichte zusammengefasst. Die vier Wahrscheinlichkeitsdichten werden dabei wie folgt definiert:

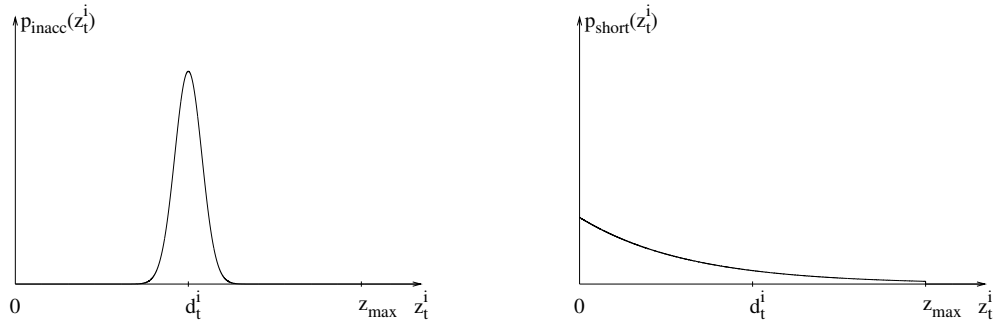
1. **Ungenauigkeit und Rauschen**
   Im Allgemeinen eignen sich Gauß- bzw. Normalverteilungen sehr gut, um kleine Abweichungen, Rauschen etc. zu beschreiben. Deshalb wird auch hier eine solche Verteilungsdichte verwendet:

$$p_{\text{inacc}}(z_t^i|d^i(s_t), m) = \begin{cases} \eta_{\text{inacc}}\,\mathcal{N}(d^i(s_t), \sigma_{\text{inacc}}; z_t^i) & \text{falls} \quad 0 \le z_t^i \le z_{\max} \\ 0 & \text{sonst} \end{cases} \tag{4.42}$$

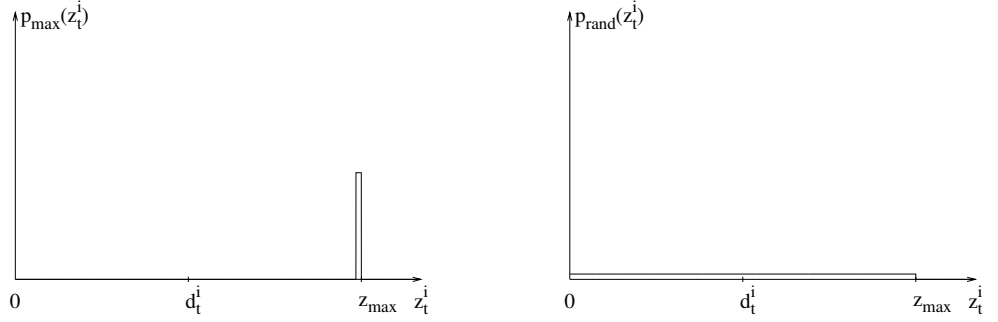
   $\mathcal{N}(d^i(s_t), \sigma_{\text{inacc}}; z_t^i)$ liefert den Funktionswert von z_t^i bei einer Standardnormalverteilung mit Mittelwert $d^i(s_t)$ und Standardabweichung σ_{inacc}. Der Faktor η_{inacc} dient der Normierung der Verteilung, da sie auf den endlichen Träger $0 \le z_t^i \le z_{\max}$ begrenzt wurde:

$$\eta_{\text{inacc}} = \left(\int_0^{z_{\max}} \mathcal{N}(d^i(s_t), \sigma_{\text{inacc}}; z_t^i)\, dz_t^i \right)^{-1} \tag{4.43}$$

2. **Phantommessungen**
   Dynamische Objekte und Cross-Talks verkürzen die gemessenen Abstände. In



(a) $p_{\text{inacc}}(z_t^i)$ (b) $p_{\text{short}}(z_t^i)$



(c) $p_{\text{max}}(z_t^i)$ (d) $p_{\text{rand}}(z_t^i)$

Abbildung 4.10.: Gezeigt werden die vier Wahrscheinlichkeitsdichten, aus denen sich die Bewertungsfunktion des Sensormodells zusammensetzt. (a) Eine Gauß-Verteilung repräsentiert das Rauschen und die Ungenauigkeit in den Längen. (b) Eine expoentielle Verteilung beschreibt Phantommessungen. (c) Max-Range-Messungen werden durch eine spezielle Gleichverteilung dargestellt. (d) Ebenfalls durch eine Gleichverteilung werden zufällige Messwerte repräsentiert.

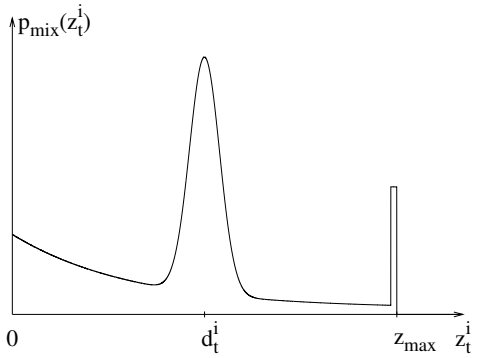


Abbildung 4.11.: Gezeigt wird die aus den vier verschiedenen Verteilungen zusammengesetzte Wahrscheinlichkeitsdichte $p_{\text{mix}}(z_t^i)$.

der Regel gilt hier: Je kürzer eine Distanz, desto wahrscheinlicher ist diese. Dieses Phänomen kann man durch eine Exponentialfunktion wie folgt ausdrücken:

$$p_{\text{short}}(z_t^i|d^i(s_t), m) = \begin{cases} \eta_{\text{short}}\ \lambda_{\text{short}}\ e^{-\lambda_{\text{short}}\ z_t^i} & \text{falls} \quad 0 \leq z_t^i \leq z_{\max} \\ 0 & \text{sonst} \end{cases} \tag{4.44}$$

Auch hier benötigt man einen Normierungsfaktor, da nur das Intervall $[0; z_{\max}]$ betrachtet wird. Der Normierungsfaktor beträgt:

$$\eta_{\text{short}} = \frac{1}{1 - e^{-\lambda_{\text{short}}\ z_{\max}}} \tag{4.45}$$

Denn dieser Term entspricht gerade dem Integral der Funktion p_{short} auf dem Intervall $[0; z_{\max}]$.

3. **Max-Range-Messungen**
   Max-Range-Messungen nehmen immer den Maximalwert des Abstandssensors an. Daher bietet sich die sogenannte Dirac-Verteilung, eine spezielle Gleichverteilung, für die Repräsentation solcher Messungen an:

$$p_{\max}(z_t^i|d^i(s_t), m) = \begin{cases} 1 & \text{falls} \quad z_t^i = z_{\max} \\ 0 & \text{sonst} \end{cases} \tag{4.46}$$

4. **Zufällige Fehler**
   Auch die zufälligen Messwerte werden, da sie beliebige Werte annehmen können, vorzugsweise durch eine Gleichverteilung approximiert:

$$p_{\text{rand}}(z_t^i|d^i(s_t), m) = \begin{cases} \frac{1}{z_{\max}} & \text{falls} \quad 0 \leq z_t^i \leq z_{\max} \\ 0 & \text{sonst} \end{cases} \tag{4.47}$$

Diese vier Verteilungsdichten werden nun mit Hilfe von vier Gewichtungsfaktoren zu einer einzigen Dichte zusammengesetzt:

$$p_{\text{mix}}(z_t^i|d^i(s_t),m) = \begin{pmatrix} \nu_{\text{inacc}} \\ \nu_{\text{short}} \\ \nu_{\text{max}} \\ \nu_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{inacc}}(z_t^i|d^i(s_t),m) \\ p_{\text{short}}(z_t^i|d^i(s_t),m) \\ p_{\text{max}}(z_t^i|d^i(s_t),m) \\ p_{\text{rand}}(z_t^i|d^i(s_t),m) \end{pmatrix} \tag{4.48}$$

wobei

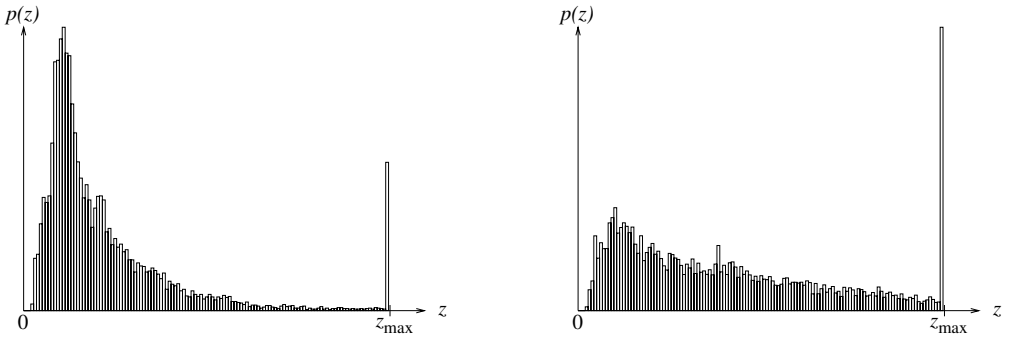
$$\nu_{\text{inacc}} + \nu_{\text{short}} + \nu_{\text{max}} + \nu_{\text{rand}} = 1 \tag{4.49}$$

Die resultierende Wahrscheinlichkeitsdichte ist in Abbildung 4.11 dargestellt. Sie approximiert das in Abbildung 4.9 dargestellte Histogramm erstaunlich gut.

Nun verfügen wir – zumindest formal – über alles, was ein robustes Lokalisierungssystem auszeichnet. Nur die Parameter $\sigma_{\text{inacc}}, \lambda_{\text{short}}, \nu_{\text{inacc}}, \nu_{\text{short}}, \nu_{\text{max}}$ und ν_{rand} müssen noch ermittelt werden. Diese können entweder basierend auf Experimenten manuell angepasst oder durch *Expectation-Maximization*-Algorithmen aus ausreichend großen Datensätzen gelernt werden. Dieses Lernen bedarf jedoch eines sehr großen Berechnungsaufwands (in der Regel mehrere Stunden auf einem modernen Rechner!). Aus diesem Grund werden wir in Abschnitt 4.4 versuchen, eine alternative Methode zur Ermittlung der Bewertungsfunktion zu entwickeln. Davor wollen wir allerdings kurz auf die Diskretisierung der Wahrscheinlichkeitsfunktion eingehen, da die Sensorinformation in Form diskreter Werte vorliegt und eine diskrete Bewertungsfunktion verschiedene Vorteile für die Implementierung mit sich bringt.

4.3.2. Histogrammbasierte Bewertung

Wir können annehmen, dass die verwendeten Abstandssensoren über ein begrenztes Auflösungsvermögen verfügen und dass die von den Sensoren stammende Abstandsinformation nur aus diskreten Werten besteht (vgl. Tabelle B.2 in Anhang B). Nehmen wir weiter an, dass die Auflösung im ganzen Messbereich $[0; z_{\text{max}}]$ konstant ist, dann gibt es auf diesem Intervall endlich viele diskrete Werte $z_0, \ldots, z_n$ mit $0 = z_0$ und $z_n = z_{\text{max}}$, die ein einzelner Messwert annehmen kann. Die Abstände zweier benachbarter diskreter Werte sind jeweils konstant und betragen $\Delta = z_{j+1} - z_j$, wobei $0 \leq j \leq n-1$ gilt. Diese Annahme wollen wir auch auf die zu erwartenden Längen d_j übertragen. Sie sollen dem gleichen diskreten Wertebereich entstammen, d. h. die selben diskreten Werte annehmen können wie ihre gemessenen Pendants. Wir können daher statt der stetigen Bewertungsfunktion $p(z_t^i|d^i(s_t))$ auch eine diskrete Bewertungsfunktion für den Vergleich verwenden. Allerdings ist sowohl die Reichweite als auch das Auflösungsvermögen heutiger Abstandssensoren so groß, dass



(a) Gebäude 079 (b) Gebäude 101

Abbildung 4.12.: Gezeigt werden die Verteilungsdichte-Histogramme für die *A-Priori*-Wahrscheinlichkeit eines Messwertes während Positionsverfolgungsprozessen in den Gebäuden 079 und 101 des Instituts für Informatik der Uni Freiburg. Die Abbildungen 4.2 und 6.4 zeigen die entsprechenden Umgebungskarten. Die maximale Reichweite liegt bei $z_{\max} = 12.8\,\mathrm{m}$ und die Klassenbreite des Histogramms beträgt $\Delta = 0.1\,\mathrm{m}$. Die geringen Wahrscheinlichkeiten $p(z)$ für die Werte bei $z = 0$ sind die Wirkung eines im Innern des Roboters befindlichen Laserscanners, einer robusten Kollisionsvermeidung sowie einer *leeren* Umgebung bezüglich dynamischer Objekte. Die vielen Fenster im Gebäude 101 spiegeln sich in einer großen relativen Häufigkeit bei $z_{\max}$ wieder.

die Bewertung durch eine diskrete Verteilungsfunktion sehr viel Speicherplatz in Anspruch nehmen würde. Eine Alternative hierzu stellt die histogrammbasierte Bewertung dar. Für die histogrammbasierte Bewertung fasst man in der Regel immer mehrere benachbarte diskrete Werte zu einer Klasse zusammen, deren Klassenbreite dann beispielsweise $\Delta' = 0.05\,\mathrm{m}$ beträgt. Man geht folglich einen Kompromiss zwischen Platzbedarf und Genauigkeit ein. Da jedoch bei der Lokalisierung nicht nur ein einzelner Messwert, sondern eine ganze Reihe von Messungen zur Bewertung eines Samples herangezogen werden, wirkt sich der eingegangene Kompromiss – wie wir in Abbildung 6.2 sehen werden – letztendlich nur ganz schwach auf die Genauigkeit der Lokalisierung aus. Die histogrammbasierte Bewertung bietet aber noch einen weiteren Vorteil: Die Wahrscheinlichkeiten können vorberechnet in einem Array abgelegt und bei Bedarf nachgeschaut werden.

4.4. Parameterbestimmung

Wir haben bereits erfahren, dass die Parameter für das in Abschnitt 4.3.1 vorgestellte Sensormodell zwar aus Datensätzen gelernt werden können, dass dafür aber auch ein sehr großer Aufwand in die Berechnung dieser Parameter gesteckt werden muss. Denn zur Berechnung der intrinsischen Parameter $\nu_{\mathrm{inacc}}, \nu_{\mathrm{short}}, \nu_{\max}, \nu_{\mathrm{rand}}, \sigma_{\mathrm{inacc}}$ und

λ_{short} wird der *Expectation-Maximation*-Algorithmus (EM-Algorithmus) benutzt, dessen Berechnungsaufwand bei diesen sechs Freiheitsgraden natürlich enorm ist. Vergleicht man darüberhinaus die Histogramme der bedingten Wahrscheinlichkeit $p(z|d)$ verschiedener Umgebungen miteinander, so stellt man zudem fest, dass sich die Charakteristik der Umgebung stark auf die Parameterwerte auswirkt. Große weite Räume (siehe Abb. 6.4) prägen das Aussehen der Verteilung $p(z|d)$ ganz anders als kleine, enge Umgebungen. Eine große Rolle spielt in diesem Zusammenhang natürlich auch der Effekt dynamischer Hindernisse: Umgebungen, in denen es von Menschen nur so wimmelt, spiegeln dies natürlich auch in der Wahrscheinlichkeitsdichte wieder. Ein einmal erstelltes universelles Sensormodell mag daher in der einen Umgebung wunderbar funktionieren, in einer anderen jedoch nur unbefriedigende Ergebnisse erzielen lassen. Damit für eine beliebige Umgebung rasch ein individuelles Sensormodell erzeugt werden kann, müssen wir daher den Aufwand für den Erzeugungsprozess erheblich reduzieren.

Um den Lernprozess der Parameter aus Sensordaten zu vereinfachen, sehen wir uns die formale Definition der Bewertungsfunktion in Gleichung (4.3) noch einmal genauer an:

$$p(z_t|s_t) := \prod_{i=1}^{M} p(z_t^i|d^i(s_t)) \tag{4.50}$$

Da wir uns ausschließlich für eine Bewertung in dem direkten Vergleich zweier diskreter Längen – nämlich z_t^i und $d^i(s_t)$ – interessieren und sich keine zeitliche oder ähnliche Information in dieser Bewertungsfunktion wiederfindet, werden wir der Übersichtlichkeit wegen im Folgenden eine etwas abgeänderte Schreibweise für die Messwerte bevorzugen. Wir werden also mit z_j den j-ten diskreten Wert aus dem Wertebereich aller möglichen gemessenen Werte und mit d_k den k-ten diskreten Wert aus dem Wertebereich aller erwarteten – d. h. der mittels Ray-Casting gewonnenen – Messwerte bezeichnen. Allerdings wollen wir nicht vergessen, dass diese beiden Wertebereiche identisch sind und somit für alle j mit $0 \leq j \leq n$ gilt: $z_j = d_j$. Von nun an steht also der Term $p(z_j|d_k)$ für unser gesuchtes Sensormodell. Mit Hilfe der Bayes'schen Regel können wir diesen wie folgt umformen:

$$p(z_j|d_k) = \frac{p(d_k|z_j)\ p(z_j)}{p(d_k)} \tag{4.51}$$

Da sowohl die diskreten Werte z_j als auch die diskreten Werte d_k eine Zerlegung unseres Wahrscheinlichkeitsraums darstellen, erhalten wir nach dem Satz der totalen

Wahrscheinlichkeit (vgl. Gleichung (A.2) in Anhang A) die Wahrscheinlichkeit $p(d_k)$ wie folgt:

$$p(d_k) = \sum_{l=0}^{n} p(d_k|z_l)\; p(z_l) \tag{4.52}$$

und damit wird Gleichung (4.51) zu:

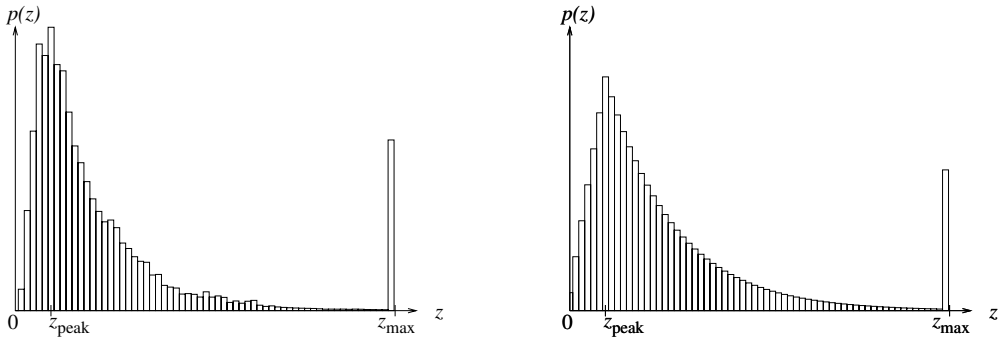
$$p(z_j|d_k) = \frac{p(d_k|z_j)\; p(z_j)}{\sum_{l=0}^{n} p(d_k|z_l)\; p(z_l)} \tag{4.53}$$

Dank dieser Umformung müssen wir jetzt nur noch die A-Priori-Wahrscheinlichkeit $p(z)$ sowie die bedingte Wahrscheinlichkeit $p(d|z)$ ermitteln.

Die A-Priori-Wahrscheinlichkeit $p(z_t^i)$ kann relativ einfach ermittelt werden. Man benötigt dazu ausschließlich die vom Roboter in der entsprechenden Umgebung gemessenen Abstände. Da diese für das Mapping, d. h. für das Erstellen der Umgebungskarte, ohnehin schon benötigt werden, entsteht in der Praxis kein zusätzlicher Aufwand, um diese Werte in Erfahrung zu bringen. Abbildung 4.12 zeigt die $p(z_t)$-Histogramme in Form relativer Häufigkeiten von gemessenen Abständen in zwei sehr unterschiedlichen Umgebungen. Man kann den Histogrammen entnehmen, dass die Beschaffenheit der Umgebung auch im Sensormodell Berücksichtigung finden muss. Gerade große Räume, deren Dimension die Reichweite des verwendeten Sensors übersteigt, oder Räume, die sehr viele Fenster aufweisen, müssen entsprechend behandelt werden. Dank der leicht zu ermittelnden *A-Priori*-Wahrscheinlichkeit $p(z)$ kann jedoch rasch ein geeignetes individuelles Sensormodell erstellt werden.

Um die ermittelte diskrete Wahrscheinlichkeitsdichte $p(z)$ verwenden zu können, muss sie entweder *geglättet* oder die entsprechenden charakteristischen Parameter bestimmt werden. Wir werden die letztere Variante bevorzugen und daher versuchen, die Dichte durch eine diskrete, zusammengesetzte Verteilungsdichte zu approximieren. Dazu sehen wir uns die Histogramme der Abbildung 4.12 genauer an. Die Dichte verfügt neben einem (lokalen) Maximum bei $z_{\max}$ über ein zusätzliches Maximum bei einer relativ kleinen Länge z_{peak}. Mit der Repräsentation des Bereichs zwischen $z = 0$ und z_{peak} durch eine lineare Funktion und der Repräsentation des Bereichs zwischen z_{peak} und $z_{\max}$ durch eine Exponentialfunktion erhält man eine gute Approximation der gelernten Verteilungsdichte. Unsere zusammengesetzte Verteilung definieren wir daher wie folgt:

$$p(z) := \begin{cases} \frac{p(z_{\text{peak}})}{z_{\text{peak}}} z & \text{falls} \quad 0 \le z \le z_{\text{peak}} \\ p(z_{\text{peak}})\; e^{\lambda(z_{\text{peak}} - z)} & \text{falls} \quad z_{\text{peak}} < z < z_{\max} \\ z_{\max} & \text{sonst} \end{cases} \tag{4.54}$$



(a) Messung (b) Approximation

Abbildung 4.13.: Gezeigt werden die *A-Priori*-Verteilungsdichte der gemessenen Längen sowie die Approximation derselben durch unsere approximierende Funktion. Die etwas kleineren Funktionswerte bei den Maxima der approximierenden Verteilung ergeben sich als Folge der Normierung.

Die erste Komponente existiert nur deshalb, weil der Sensor sich im Innern des Roboters befindet – was übrigens bei allen, zur Erstellung dieser Arbeit verwendeten Robotern der Fall ist – und weil der Roboter zur Vermeidung von Kollisionen stets einen gewissen Sicherheitsabstand zu Objekten einzuhalten pflegt. Wäre der Abstandssensor außen an der Karosserie des Roboters angebracht und befände der Roboter sich in einer äußerst dynamischen Umgebung, wie es beispielsweise auf Messen oder Ausstellungen meist der Fall ist, dann würde die erste Komponente unserer Definition verschwinden, was die Anzahl unbekannter Paramter verringert und damit die Parameterbestimmung deutlich vereinfacht. Wir werden allerdings diese Komponente miteinbeziehen und versuchen, aus dem gelernten Histogramm die genannten Parameter zu extrahieren. Glücklicherweise ergibt sich die Funktion in Gleichung (4.54) direkt aus den die Verteilungsdichte charakterisierenden Werten z_{peak}, $p(z_{\text{peak}})$ sowie $p(z_{\text{max}})$ mit Ausnahme des Parameters λ. Lediglich diesen Faktor gilt es noch durch ein geeignetes Schätzverfahren zu ermitteln. Wir setzen zu diesem Zweck das $\mathcal{X}^2$-Minimumprinzip ein, welches mit Hilfe der $\mathcal{X}^2$-Funktion (siehe Anhang A), die als Maß für die Abweichung der Häufigkeitsverteilung h_j der gemessenen Daten von der durch die in Gleichung (4.54) gegebenen, approximierenden Häufigkeitsverteilung eingesetzt werden kann, diesen Parameterwert λ schätzt. In Abbildung 4.13 sind die Histogramme der beiden Häufigkeitsverteilungen gegenübergestellt. Offensichtlich repräsentiert das Histogramm unserer Verteilungsfunktion die Häufigkeitsverteilung der gemessenen Längen ausreichend gut.

Der Term $p(d_k|z_j)$ steht für die Wahrscheinlichkeit einer erwarteten Länge d_k unter der Voraussetzung, dass die gemessene Länge z_j beträgt. Diese Wahrscheinlichkeitsdichte kann leider nicht bereits im Vorfeld berechnet werden, da diese bedingte

Wahrscheinlichkeit erst während der Lokalisierung bekannt wird, und muss deshalb entweder aus einem früheren Lokalisierungprozess übernommen oder im gegenwärtigen Prozess aus den bisher angefallenen Daten extrahiert werden. Bei Gitter traversierenden Lokalisierungsverfahren wird während der Aktualisierungsphase zur Gewichtung der Samples für jedes einzelne Sample der zugehörige erwartete Scan durch Ray-Casting berechnet. Jeder gemessenen Strahllänge z_j können somit die N entsprechenden ermittelten erwarteten Distanzen d_k der Samples gegenübergestellt werden. Abbildung 4.14 zeigt die Entwicklung der Dichte $p(d|z)$ während der ersten Schritte einer globalen Lokalisierung. Eine Gauß-Verteilung stellt ganz offensichtlich eine gute Approximation dieser Dichte dar. Bei der Gestalt dieser Verteilung werden wir daher von einer Gauß-Verteilung ausgehen. Ganz deutlich erkennt man, dass mit dem durch das Filtern verursachten Wegfallen unwahrscheinlicher Samples auch die Abweichung kleiner wird, bis sich schließlich eine von der verwendeten Klassenbreite des Histogramms abhängige Sättigung einstellt.

4.5. Dynamische Adaption der Wahrscheinlichkeitsfunktion

Wenn bei der globalen Lokalisierung weniger Samples eingesetzt werden sollen, dann muss offensichtlich jedes einzelne Sample nicht nur einen einzigen Punkt, sondern einen kleinen Bereich von Positionen repräsentieren. Dies ist jedoch nur dann möglich, wenn auch die Bewertungsfunktion, welche die Strahllängen des erwarteten Scans mit denen des gemessenen Scans vergleicht, nicht *zu streng* in der Bewertung ist. Die Verteilungsdichte $p(d|z)$ ist die Komponente in der Bewertungsfunktion, die regelt, welchen Spielraum eine erwartete Länge bezüglich einer gemessenen Länge hat. Eine Veränderung dieser Komponente – in Form einer Vergrößerung der Standardabweichung $\sigma_{p(d|z)}$ – wird automatisch auch auf den Spielraum der Position $(x, y, \theta)^T$ des Samples abgebildet. Die angestrebte Kulanz bei der Bewertung kann somit durch diesen Parameter geregelt werden. Wie wir im nächsten Kapitel sehen werden, kann man grundsätzlich zwei Anforderungen an die Standardabweichung $\sigma_{p(d|z)}$ stellen. Bei der globalen Lokalisierung sollte mit keinem zu kleinen Wert gestartet werden, damit die Erfolgsquote verbessert werden kann. Denn bei einem zu kleinen Wert und keiner besonders großen Anzahl an Samples kann es schnell passieren, dass kein Sample die wahre Roboterposition trifft und Samples, welche falsche Positionen beschreiben, besser bewertet werden, so dass bei zu schnellem Konvergieren eine Falschlokalisierung die Folge ist. Die zweite Anforderung betrifft die Positionsverfolgung. In diesem Fall ist es gerade umgekehrt: Der Wert für $\sigma_{p(d|z)}$ sollte möglichst klein sein, damit die Samplemenge möglichst gut fokussiert wird und die resultierende Position möglichst genau der wahren Roboterposition entspricht.

Während des Lokalisierungsprozesses berechnet man für jeden verwendeten Strahl

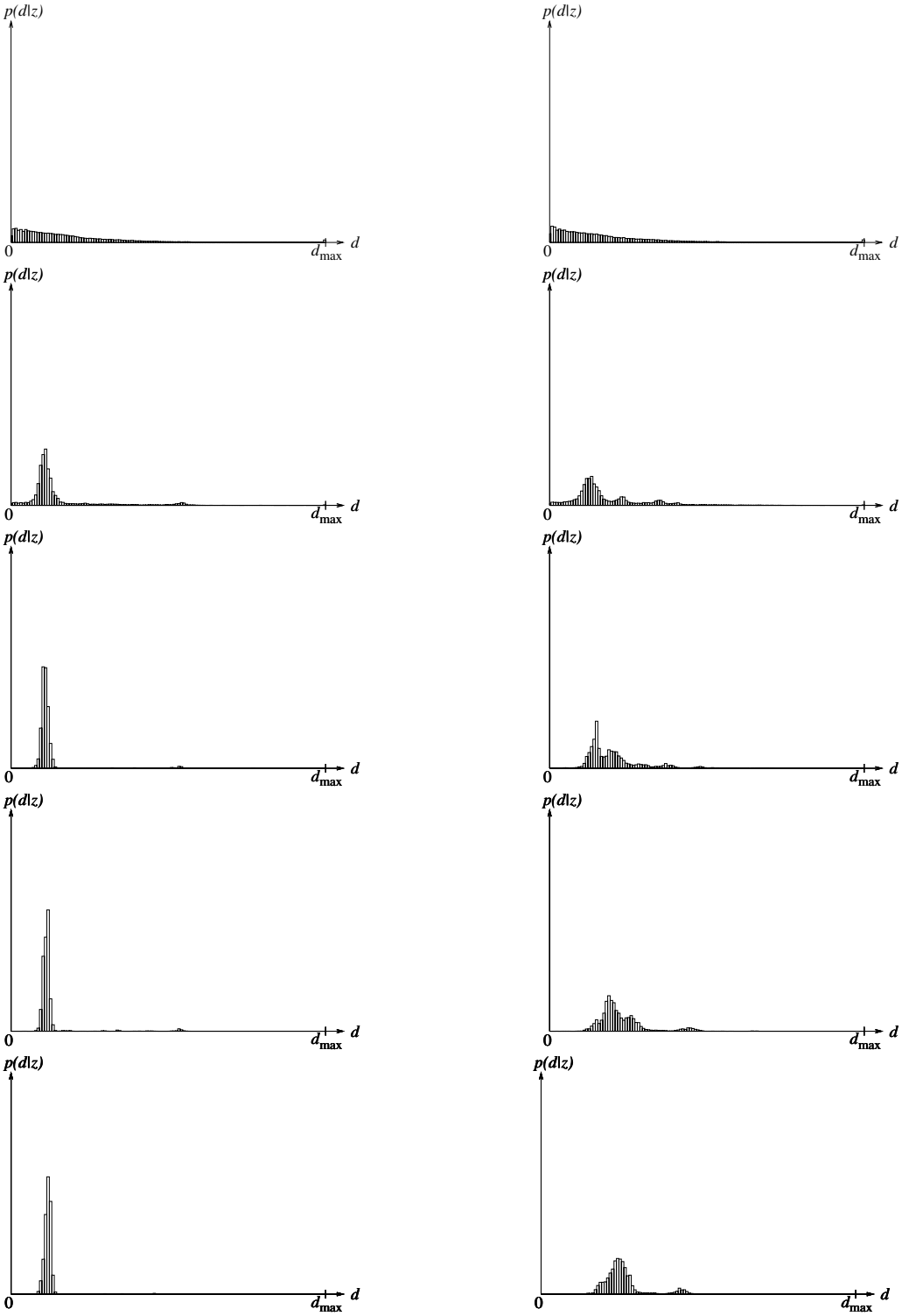


Abbildung 4.14.: Die Histogramme zeigen von oben nach unten die Entwicklung der Verteilungsdichte $p(d|z)$ für zwei ausgewählte Richtungen des Scanners während des Resamplings in den ersten Schritten einer globalen Lokalisierung. Man beachte, dass sich die Werte für die Längen z auch von Zeile zu Zeile verändern können.

z^k die n korrespondierenden Längen d^k der Samplemenge. Daraus kann – entsprechend der Zahl der verwendeten Strahlen – eine Reihe von Standardabweichungen $\sigma_{p(d|z)}$ gewonnen werden, von denen man den Median, der bekanntlich unempfindlich gegenüber Ausreißern ist, oder – um eine noch strengere Bewertung zu erhalten – das Minimum direkt für die $p(d_k|z_j)$-Komponente der Bewertungsfunktion verwenden und somit für die Bewertung der Samples im nächsten Resampling-Schritt zur Aktualisierung des Bewertungshistogramms einsetzen kann.

4.6. Endpunkt-Sensormodell

Auf das in Abschnitt 2.3 bereits vorgestellte Endpunkt-Sensormodell wollen wir an dieser Stelle noch einmal zurückkommen, um dessen mathematische Beziehungen zu formalisieren. Die Idee des Endpunkt-Sensormodells ist die folgende: Ausgehend von jeder Sampleposition $(x^i, y^i, \theta^i)^T$ – d. h. von jeder einzelnen potentiellen Roboterposition – werden die Zellen berechnet, in welche die Endpunkte der Strahlen z^k des gemessenen Scans fallen würden, wenn sich der Roboter tatsächlich an dieser Position befände. Dieser Vorgang stellt somit eine Projektion des Scans in das globale Koordinatensystem der Umgebungskarte dar. Gehen wir davon aus, dass unser Sensor immer einen Öffnungswinkel von 180° besitzt und die einzelnen Messungen gegen den Uhrzeigersinn gemacht werden und bezeichnen wir die Winkelauflösung mit γ, dann erhalten wir die Endpunktkoordinaten $(x^{i,k}_{\text{end}}, y^{i,k}_{\text{end}})^T$ des k-ten Strahls bezüglich des i-ten Samples durch folgende Beziehung:

$$\begin{pmatrix} x^{i,k}_{\text{end}} \\ y^{i,k}_{\text{end}} \end{pmatrix} = \begin{pmatrix} x^i \\ y^i \end{pmatrix} + \begin{pmatrix} z^k \cos(\theta^i - \frac{\pi}{2} + k\gamma) \\ z^k \sin(\theta^i - \frac{\pi}{2} + k\gamma) \end{pmatrix} \tag{4.55}$$

Die Ermittlung der Strahlendzellen bedarf nur weniger geometrischer Berechnungen und die Komplexität ist proportional zur Anzahl der verwendeten Strahlen des Scans, also konstant, falls die Anzahl der verwendeten Strahlen nicht geändert wird, weshalb dieses Verfahren auch sehr schnell ist.

Die Belegungsgrade der berechneten Zellen geben nun Aufschluss darüber, wie wahrscheinlich der gemessene Scan an dieser Sampleposition ist. Wie in Abschnitt 2.3 bereits erwähnt wurde, benötigt man hierfür eine Wahrscheinlichkeitskarte. Eine solche Wahrscheinlichkeitskarte verfügt in jeder Zelle über einen Wert, der aussagt, wie wahrscheinlich es ist, dass ein Strahl in dieser Zelle endet. Offensichtlich ist dieser Wahrscheinlichkeitswert bei einer belegten Zelle maximal und bei einer unbelegten minimal. Die einfachste Art, eine solche Wahrscheinlichkeitskarte zu erhalten, ist die direkte Übernahme aus der Belegungskarte, welche wir in Abschnitt 4.1 definiert haben. Diese Vorgehensweise stellt aber ein kleines Problem dar: Liegt eine Zelle nur knapp neben einer belegten Zelle, sollte ihr Wahrscheinlichkeitswert – auch wenn sie nur zu einem ganz geringen Grad belegt ist – nicht minimal sein. Denn die

Auswirkungen auf die Zuverlässigkeit bei der globalen Lokalisierung wären äußerst ungünstig. Ein Grundsatz bei der Lokalisierung ist nämlich der folgende: Je spitzer (engl. peaked) die Verteilung ist, desto eher fallen auch relativ gute Samples – im Sinne von: nah an der tatsächlichen Roboterposition – dem Filterungsschritt zum Opfer. Eine Abhilfe stellt hier eine sogenannte Faltung der ursprünglichen Karte dar. Damit werden die Übergänge zwischen belegten und unbelegten Bereichen fließender gestaltet, um letztendlich eine glattere Verteilung zu erhalten.

Eine andere Möglichkeit bieten die Distanzkarten. Hierzu wird die ursprüngliche Umgebungskarte zuerst in eine binäre Karte umgewandelt, so dass eine Zelle nur entweder belegt oder frei sein kann. Daraus wird z. B. mit Hilfe des in Abschnitt 4.2.4 vorgestellten Transformationsalgorithmus die entsprechende Karte minimaler euklidischer Distanzen berechnet (siehe Abbildung 4.5). Mit Hilfe der Distanzkarte, die wir bisher immer mit d_{edt} bezeichnet haben, und der folgenden Funktion kann nun die für das Endpunkt-Sensormodell erforderliche Wahrscheinlichkeitskarte, die wir mit w_{prob} bezeichnen wollen, generiert werden:

$$w_{\text{prob}} = e^{-\frac{(d[x,y]\cdot r)^2}{2\sigma^2_{\text{fuzzi}}}} \tag{4.56}$$

Dabei steht r für die Auflösung der Karte in Meter je Zelleneinheit und σ_{fuzzi} gibt den Grad der Faltung (engl. Fuzziness) an. Damit ist es beispielsweise möglich, für die globale Lokalisierung eine andere Wahrscheinlichkeitskarte zu benutzen als für die Positionsverfolgung. Im Anschluss an die Berechnung schließt sich in der Regel noch eine Normierung der Wahrscheinlichkeitskarte dahingehend an, dass die Zelle mit der größten Wahrscheinlichkeit gerade einen Wahrscheinlichkeitswert von 1 aufweist.

Mit diesem und den vorhergehenden Kapiteln haben wir uns alle notwendigen, theoretischen Grundlagen strahlenbasierter Sensormodelle erarbeitet, um ein effektives Lokalisierungssystem implementieren zu können. Das nächste Kapitel repräsentiert den praktischen Teil dieser Arbeit. In ihm werden wir daher die Softwareumgebung, in welcher das Lokalisierungssystem, eingesetzt wurde, und im Anschluss daran die durchgeführten Experimente vorstellen.

5. Implementierung

Als Roboter-Betriebssoftware kommt das Carnegie-Mellon-Navigation-Toolkit (CARMEN) zum Einsatz. Diese Open-Source-Software wurde an der Carnegie-Mellon-Universität in Pittsburgh entwickelt mit dem Ziel, die Programmierung mobiler Roboter in Wissenschaft, Forschung und Entwicklung zu standardisieren [Montemerlo et. al. 2003]. Für die Durchführung der Experimente dieser Arbeit wurde ein Lokalisierungs-Modul namens *Localizer* implementiert. Dieses kann das in der CARMEN-Software enthaltene Lokalisierungsmodul vollständig ersetzen.

5.1. CARMEN

CARMEN basiert auf einer modularen, dreischichtigen Architektur. In der untersten Schicht befinden sich die maschinennahen Module, welche abstrakte Schnittstellen zur Roboterplattform, zur Sensorik und zur Aktorik bilden. Da möglichst viele verschiedene Robotertypen unterstützt werden sollen und sich Plattformen in Technik und Ansteuerung aber erheblich voneinander unterscheiden, existiert für jeden Robotertyp ein eigenes Modul. Dieses stellt für die darüberliegende Schicht abstrakte Methoden für primitive Fahrbefehle zur Verfügung. Darüberhinaus existieren in der untersten Schicht auch Module für verschiedene Abstandssensoren sowie für Bumper (zur Detektion von Kollisionen). Die zweite Schicht enthält Module für die Pfadplanung, die Lokalisierung und das Verfolgen dynamischer Objekte. In der dritten Schicht können nun die Methoden der darunterliegenden Schicht verwendet werden, um eigene Anwendungen mit komplexeren Aufgabenbereichen zu entwickeln. Weiter enthält das CARMEN-Toolkit diverse nützliche Werkzeuge wie graphische Benutzerschnittstellen (engl. GUIs), Editoren oder Daten-Rekorder.

Die drei wichtigsten Ziele der CARMEN-Entwickler sind eine einfache Benutzbarkeit, eine leichte Erweiterbarkeit sowie eine große Zuverlässigkeit bezüglich etwaiger Ausfälle. Die folgenden sieben Ansätze im Entwurf der Software tragen zur Verwirklichung dieser Ziele bei:

Modularität

Die CARMEN-Software besteht aus einem Bündel von Modulen, die miteinander über eine Kommunikationsarchitektur namens *Inter-Process-Communication* (IPC), welche von Simmons [2004] an der Carnegie Mellon Universität entwickelt wurde, kommunizieren. Diese Kommunikation erfordert natürlich zusätzliche Rechenressourcen verglichen mit monolithischen Systemen, was jedoch durch die folgenden Vorteile schnell wieder aufgewogen werden kann:

- **Verteilung auf externe Ressourcen**: Die Kommunikation der Module kann bei IPC über ein Netzwerk erfolgen. Auf diese Weise müssen nicht alle Module auf ein und demselben Rechner ausgeführt werden. Gerade dann, wenn – wie in unserem Fall – der Roboter selbst nur über einen Einplatinencomputer mit begrenzter Rechenleistung verfügt, ist es sinnvoll, rechenaufwändige Prozesse wie die Kartografierung auf externe Rechner zu verlagern.

- **Flexibilität**: Da man für jede Roboterplattform individuelle Anpassungen vornehmen muss, werden diese für jeden Roboter getrennt in dem entsprechenden Modul implementiert. An die Stelle eines einzigen komplexen, aufgeblähten Moduls treten so verschiedene, an die jeweilige Plattform optimal angepasste Module. Durch die Kommunikationsarchitektur wird das einfache Zuschalten des gewünschten Moduls unterstützt.

- **Erweiterbarkeit**: Durch die Verwendung genau definierter Schnittstellen, kann das System auf einfache Weise um neue Module, die diese Schnittstellen zur Datengewinnung nutzen oder über diese Schnittstellen bereits implementierte Funktionen verwenden können, ergänzt werden.

Einfachheit der Hauptmodule

Die für die Steuerung eines Roboters notwendigen elementaren Funktionen wie die Steuerung des Antriebs, die Lokalisierung sowie die Pfadplanung werden bei CARMEN in einzelne Module gepackt, um für komplexere Aufgaben der darüberliegenden Schicht entsprechende Funktionalität zur Verfügung zu stellen. Durch diese Trennung sinkt die Komplexität der einzelnen Module im Vergleich zu einem universellen, monolithischen Kern erheblich, was sich gerade in der Prototypentwicklung positiv auf die Fehlersuche auswirkt. Denn grundsätzlich gilt: Je größer und komplexer ein Programm, desto schwieriger gestaltet sich die Fehlersuche – besonders dann, wenn mehrere, voneinander unabhängig arbeitende Personen an der Entwicklung beteiligt sind.

Trennung von Steuerung und Benutzeroberfläche

Eine bemerkenswerte Richtlinie stellt die strikte Trennung von Steueralgorithmen und graphischen Benutzeroberflächen (GUIs) dar. Kein Hauptmodul benötigt für die Ausführung ein GUI. Stattdessen stellen die GUIs eigene Module dar, die über die Kommunikationsarchitektur mit den Hauptmodulen Daten austauschen oder Zugriff auf deren Funktionen dem Benutzer zur Verfügung stellen. Auf diese Weise kann zu jedem beliebigen Zweck eine individuelle Oberfläche entworfen werden. Für Präsentationen bietet sich beispielsweise ein zentrales GUI an, das den gegenwärtigen Zustand der Hauptmodule graphisch in einem Fenster zusammenfasst. Da die CARMEN-Software auch die Simulation sowie das Aufzeichnen von Daten unterstützt, ist es auch denkbar, die graphische Benutzeroberfläche bei umfangreichen Experimenten nur hin und wieder zur Kontrolle zu aktivieren, um teure Rechenzeit zu sparen.

Abstraktion in der Kommunikation

Die eigentliche Kommunikations-Infrastruktur der CARMEN-Software-Module wurde aus verschiedenen Gründen von der eigentlichen Funktionalität getrennt. Der Hauptgrund hierfür liegt darin, dass dies einen möglichen Umstieg auf ein neues Kommunikationsprotokoll erheblich vereinfacht. Der für die Kommunikation verantwortliche Teil eines Moduls befindet sich in einer eigenen Quellcode-Datei und die Schnittstelle zur eigentlichen Funktionalität des Moduls wurde von diesem Teil derart abstrahiert, dass die Kommunikationskomponente ohne großen Aufwand durch eine andere ersetzt werden kann.

Abstraktion in den Hardware-Schnittstellen

CARMEN unterstützt eine Vielzahl von Roboterplattformen, die sich nicht nur in Gestalt und Aussehen unterscheiden, sondern vielmehr in der Steuerung. Während die eine Plattform Fahrbefehle in Form von Translations- und Rotationsgeschwindigkeit erwartet, verlangt eine andere die Umdrehungszahl je Rad. Um eine einheitliche Grundlage für die Ansteuerung dieser Roboter zu schaffen, wandeln die Plattform-Module intern die einheitlichen abstrakten Steuerbefehle in die entsprechende verständliche Form der Plattform um. Gleichzeitig konvertieren diese Module aber auch die von der Odometrie der Plattform gemessenen Bewegungen in eine einheitliche Form um und stellen diese für andere Module (einer höheren Schicht) abstrahiert von der eingesetzten Plattform zur Verfügung. Die Funktionen für diese Transformationen sind damit in den Plattform-Modulen eingeschlossen und bleiben anderen Modulen verborgen, was als *Information Hiding* bezeichnet wird und bei der Entwicklung moderner Software immer größere Verbreitung findet.

Verwendung standardisierter Einheiten und Größen

Um eine internationale Standardisierung zu erreichen entschieden sich die CARMEN-Entwickler dafür, bei allen verwendeten Einheiten das internationale Einheitensystem (SI) zu verwenden [Taylor 1995]. Dieses System ist auch unter dem Namen MKS-System bekannt, da dessen Haupteinheiten der Meter, das Kilogramm und die Sekunde sind. Das globale Koordinatensystem lehnt sich ebenfalls an den Standard in der Mathematik an. Die x-Achse verläuft von links nach rechts, die y-Achse von unten nach oben. Die Orientierung in diesem Koordinatensystem wird durch den Winkel zwischen der positiven x-Achse und der Richtung gegen den Uhrzeigersinn in Radianten[1] gemessen. Die Roboterorientierung bzw. -ausrichtung θ liegt im Bereich $-\pi \leq \theta \leq \pi$.

Verwendung eines zentralen Parametermodells

Zuletzt soll das in CARMEN eingesetzte, zentrale Parameter-Modul nicht ungenannt bleiben. Dieses Modul hat die Aufgabe alle anderen Module mit den notwendigen Parameterwerten zu versorgen. Dies bedeutet, dass es nur noch eine Parameterdatei mit Einstellungen gibt, die von dem Parameter-Modul, dem *Parameter-Server*, eingelesen wird. Dies bietet enorme Vorteile gegenüber dezentralisierten Ansätzen. Denn auf diese Weise existiert zu jedem Zeitpunkt nur ein eindeutiges Parametermodell, so dass die Wahrscheinlichkeit, dass verschiedene Module über verschiedene Werte für ein und denselben Parameter verfügen, minimiert wird. Des Weiteren können zur Laufzeit Parameter über den Parameter-Server geändert werden, was dieser daraufhin anderen Modulen mitteilt, die darauf entsprechend reagieren können. Der Neustart oder die Neukompilierung eines Moduls wird dadurch in den meisten Fällen hinfällig. Zu den Parametern zählen auch die für den Roboterbetrieb notwendigen Umgebungskarten. Um die Datenmenge bei der Übertragung einer solchen Karte möglichst klein zu halten, wurde in CARMEN das verbreitete Komprimierungswerkzeug ZLIB verwendet [Gailly und Adler 2004].

[1] Einheit des Winkels im Bogenmaß

6. Experimente

In diesem Kapitel wollen wir die Ergebnisse der im Rahmen dieser Arbeit durchgeführten Experimente zusammenfassen. Bei dem in den Experimenten benutzten Roboter handelt es sich um einen Pioneer 2DX der Firma ActivMedia (siehe Abbildung B.1 in Anhang B). Die wichtigsten technischen Daten dieser mobilen Plattform sind in Tabelle B.1 in Anhang B zusammengestellt. Dieser Roboter wurde mit einem Stayton Board der Firma Intel als Rechner und mit Linux als Betriebssystem ausgestattet. Um eine Wireless-LAN-Karte erweitert kann man auf diese Weise externe Rechner mit dem Roboter verbinden.

6.1. Positionsverfolgung

In der Praxis ist ein Lokalisierungssystem die meiste Zeit mit der Positionsverfolgung beauftragt. Die wichtigsten Anforderungen an das System sind dabei eine möglichst geringe Abweichung von der wirklichen Roboterposition und eine möglichst schwache Anfälligkeit gegenüber Sensorrauschen und dynamischen Hindernissen. Die Bewertungsfunktion $p(z|s,m)$ muss diesen Ansprüchen gerecht werden, indem sie möglichst nur die besten Samples gut bewertet, ohne dadurch an Robustheit zu verlieren. Gleichzeitig sollen aber auch die begrenzten Rechenressourcen geschont werden, da es innerhalb einer Roboter-Betriebssoftware noch weitere Module und andere komplexe Aufgaben zu erledigen gibt. In diesem Abschnitt wollen wir untersuchen, wie sich die Charakteristik der Bewertungsfunktion auf die Genauigkeit bei der Positionsverfolgung auswirkt. Wir interessieren uns daher in dem folgenden Experiment für die Abhängigkeit des Fehlers zwischen der lokalisierten Position und der wahren Roboterposition von dem der Bewertungsfunktion zugrunde liegenden Parameter $\sigma_{p(d|z)}$. Abbildung 6.1 zeigt diesen Fehler für verschiedene Standardabweichungen $\sigma_{p(d|z)}$. Für kleinere Werte sinkt auch der resultierende Fehler. Allerdings lässt sich bei einer histogrammbasierten Bewertung die Standardabweichung nicht beliebig verkleinern. Unter einem bestimmten Wert, wird die sogenannte *Gauß-Glocke* der Gauß-Verteilung nur noch impulsförmig durch eine einzige Klasse repräsentiert. Nur die Längen, die in diese Klasse fallen, erhalten dann noch eine gute Bewertung. Die Genauigkeit bei der Positionsverfolgung wird daher natürlich auch von der gewählten Klassenbreite des Histogramms festgelegt. Je kleiner die Klassenbreite, desto besser lassen sich die Strahllängen auflösen; desto mehr Klassen existieren aber auch – bei gleichbleibender maximaler Reichweite – und umso mehr Aufwand muss auch

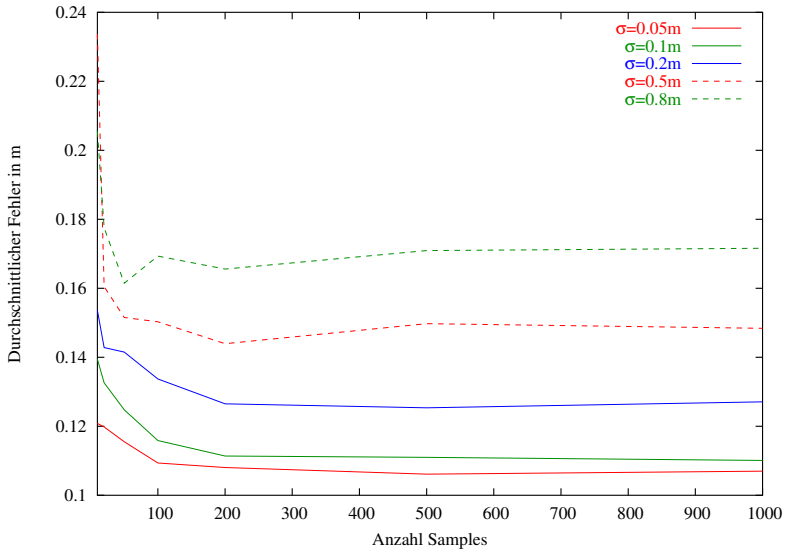
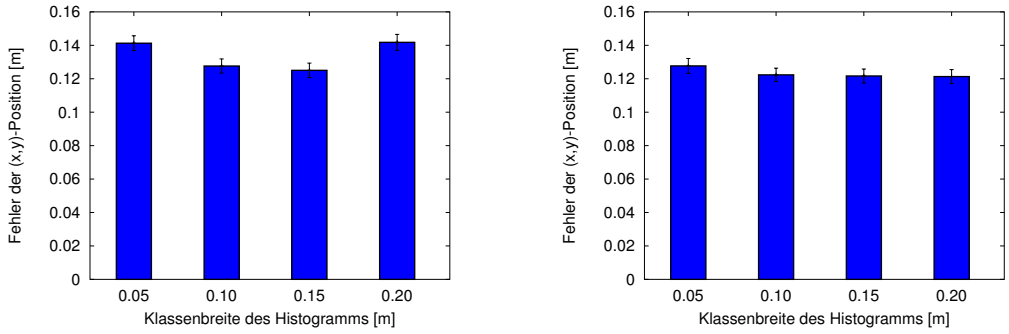


Abbildung 6.1.: Gezeigt wird der durchschnittliche Fehler der vom Lokalisierungsverfahren ermittelten Roboterposition bezüglich der wahren Position des Roboters für die Bewertungsfunktion mit verschiedenen Werten für $\sigma_{p(d|z)}$ beim Tracking auf der gesamten Trajektorie im Gebäude 101. Je größer die Standardabweichung $\sigma_{p(d|z)}$ ist, desto größer wird auch der resultierende Fehler. Für die Positionsverfolgung empfielt es sich daher, eine möglichst kleine Standardabweichung zur Bewertung der Längen heranzuziehen.



(a) 100 Samples

(b) 300 Samples

Abbildung 6.2.: Gezeigt wird der Einfluss der Histogramm-Klassenbreite auf die Genauigkeit bei der Positionsverfolgung. Die Diagramme zeigen den durchschnittlichen Positionsfehler in der euklidischen Ebene für verschiedene Klassenbreiten und für Mengen mit 100 und 300 Samples.

in die Erzeugung der Histogramme und in die Bewertung der Samples gesteckt werden. Jedoch kann man Abbildung 6.2 entnehmen, dass sich die Klassenbreite nicht gravierend auf die Genauigkeit beim Tracking auswirkt.

6.2. Globale Lokalisierung

Das Ziel bei der globalen Lokalisierung ist es, sowohl mit einer hohen Erfolgsquote zu lokalisieren als auch so schnell wie möglich die über den gesamten freien Bereich der Umgebungskarte gleichmäßig verteilten Samples konvergieren zu lassen, um mit der Positionsverfolgung fortfahren zu können. Die Erfolgsquote hängt vor allem von der Größe der verwendeten Samplemenge ab. Je mehr Samples verwendet werden, desto wahrscheinlicher hat man bei der globalen Lokalisierung Erfolg. Der Grund hierfür liegt darin, dass mit zunehmender Sampleanzahl der zu erwartende Abstand des zur tatsächlichen Roboterposition nächsten Samples kleiner und demzufolge die wahre Roboterposition umso wahrscheinlicher *erfasst* wird. Die Entscheidung, wie viele Samples wirklich für eine erfolgreiche globale Lokalisierung notwendig sind, stellt jedoch in der Robotik ein bisher ungelöstes Problem dar.

Neben der Größe der Samplemenge beeinflusst aber auch das verwendete Sensormodell die Erfolgsquote bei der globalen Lokalisierung erheblich. Abbildung 6.3 zeigt die Fehlerrate bei der globalen Lokalisierung in Abhängigkeit von der Standardabweichung $\sigma_{p(d|z)}$ der $p(d|z)$-Komponente der Bewertungsfunktion. Bei einem bestimmten, relativ großen Wert für die Abweichung (im Bereich $\sigma_{p(d|z)} = 0.5$ bis

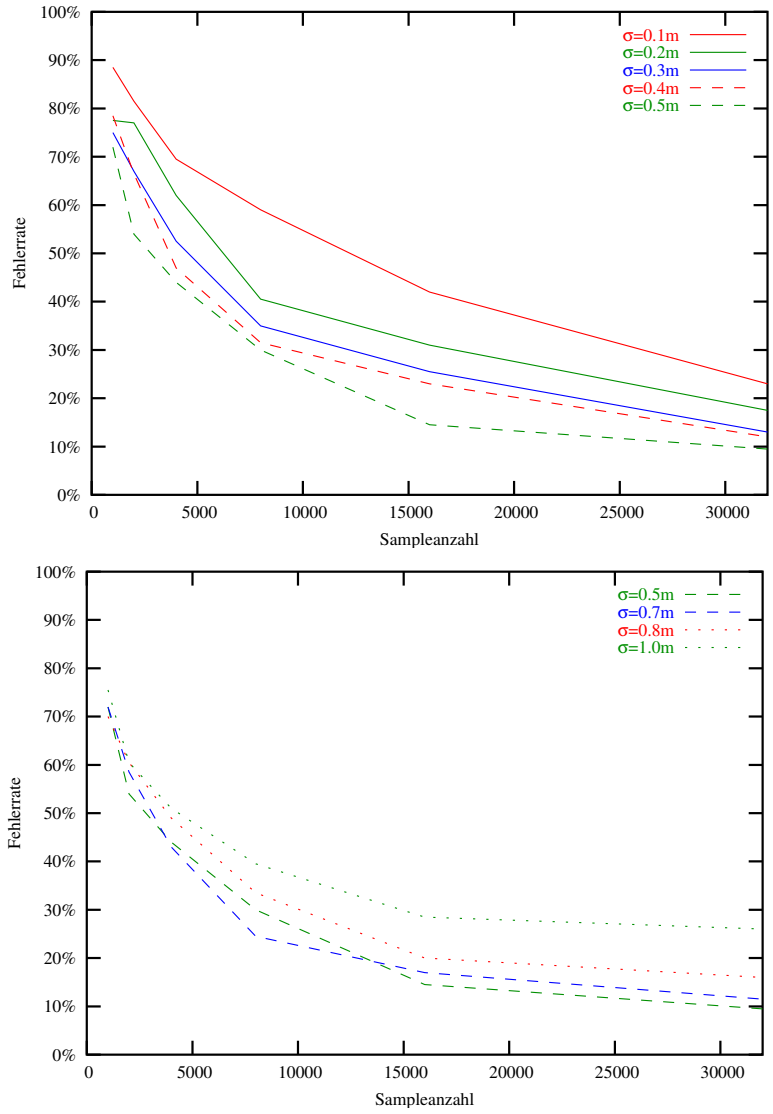


Abbildung 6.3.: Gezeigt wird die Fehlerrate für verschiedene Werte von $\sigma_{p(d|z)}$ und verschiedene Größen der Samplemenge bei der globalen Lokalisierung. Das obere Diagramm zeigt, dass sich kleine, für die Positionsverfolgung sinnvolle Standardabweichungen nicht besonders gut für die globale Lokalisierung eignen, da gute Samples zu früh dem Filterungsprozess zum Opfer fallen. Auf der anderen Seite sollte das Sensormodell aber auch nicht zu kulant sein, da mit zunehmender Standardabweichung die Erfolsquote wieder abnimmt (unteres Diagramm).

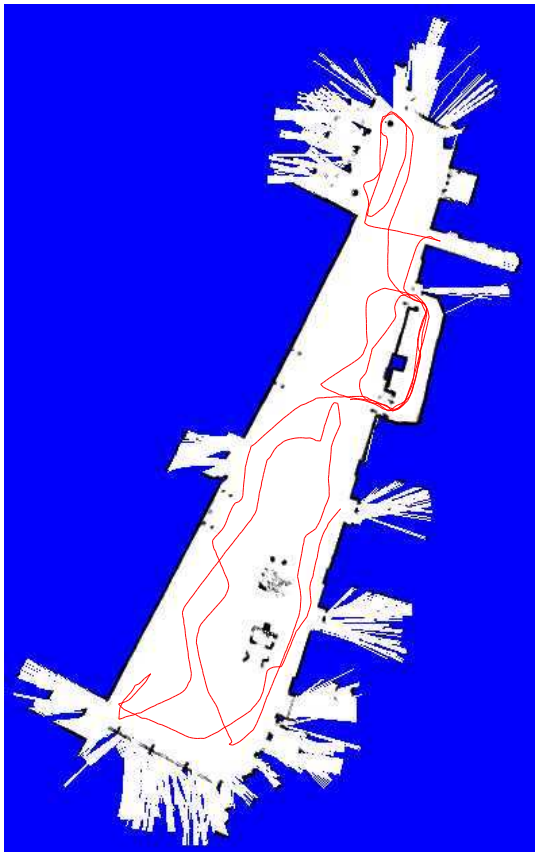


Abbildung 6.4.: Die Abbildung zeigt den vom Roboter im Gebäude 101 an der Fakultät für Angewandte Wissenschaften der Uni Freiburg gefahrenen Pfad.

$\sigma_{p(d|z)} = 0.7$) wird die Fehlerrate minimiert, d. h. die Erfolgsquote maximiert. Vergrößert man $\sigma_{p(d|z)}$ weiter, nimmt die Fehlerrate wieder zu. Diesen Sachverhalt zeigt das untere Diagramm in Abbildung 6.3. Wird also statt der bei der Positionsverfolgung eingesetzten strengen Bewertungsfunktion eine kulantere verwendet, dann kann mit einer kleineren Sampleanzahl dieselbe Erfolgsquote erzielt werden, während weniger Rechenressourcen aufgebracht werden müssen.

Alle bisher vorgestellten Experimente wurden mit einer festen Anzahl an Samples durchgeführt. Lokalisierungssysteme mit fester Sampleanzahl haben jedoch den Nachteil, dass man die notwendige Anzahl an Samples vorgeben muss. Weder bei der globalen Lokalisierung noch beim Tracking darf diese Zahl zu klein sein. Das System würde dann nämlich Gefahr laufen, eine falsche Position zu ermitteln bzw. die wahre Position zu verlieren. Wie viele Samples allerdings tatsächlich notwendig sind, lässt sich nicht entscheiden. Bei der Positionsverfolgung innerhalb realer Roboteranwendungen sollte die Sampleanzahl nicht zu hoch gewählt werden, damit für andere Prozesse und Aufgaben mehr Rechenzeit zur Verfügung steht. Bei Systemen mit fester Sampleanzahl bietet es sich daher an, die Zahl der Samples nach Abschluss

der Phase der globalen Lokalisierung herabzusetzen. Allerdings ist der Übergang zwischen globaler Lokalisierung und Positionsverfolgung fließend. Der Zeitpunkt für den Wechsel kann beispielsweise durch den Grad der Konvergenz festgelegt werden. Eine Möglichkeit für ein Konvergenzkriterium stellt folgende dar: Falls es kein Sample mehr gibt, das von der lokalisierten Position, welche meist aus dem (gewichteten) arithmetischen Mittel der Samplepositionen gewonnen wird, weiter entfernt ist als eine bestimmte Schwelle (z. B. 1 m), dann betrachtet man die globale Lokalisierung als abgeschlossen und fährt mit der Positionsverfolgung fort.

6.3. KLD-Sampling

Beim KLD-Sampling benötigt man eine solche Fallentscheidung nicht. Denn das KLD-Sampling passt die Sampleanzahl dynamisch an die vorliegenden Verhältnisse an. Dies hat zur Folge, dass bei einer unzureichenden Anzahl an Samples die Anzahl während des Resamplings erhöht wird, während beim Tracking die Anzahl sehr gering ausfällt. Abbildung 6.5 zeigt die ersten Schritte einer globalen Lokalisierung mit KLD-Sampling. Man kann sehr gut sehen, wie die Anzahl an Samples zunächst zunimmt, weil die anfänglich gewählte Anzahl nicht ausreicht, die gewünschte Approximationsgüte der Verteilungsdichte zu gewährleisten. In der Praxis werden beim KLD-Sampling eine Ober- und eine Untergrenze für die Anzahl an Samples vorgegeben. Erfahrungsgemäß wird bei vorgegebenem maximalem Fehler ϵ und vorgegebener Wahrscheinlichkeit δ zu Beginn der globalen Lokalisierung die Obergrenze fast immer erreicht. Im Folgenden werden wir deshalb versuchen, unser dynamisches Sensormodell in Kombination mit dem KLD-Sampling einzusetzen, um auch mit einer kleineren oberen Grenze die gleiche Erfolgsquote erzielen zu können.

Betrachtet man globale Lokalisierung und Tracking getrennt, kann man jeweils einen optimalen Wert für die Standardabweichung $\sigma_{p(d|z)}$ experimentell ermitteln. Für die Umgebung im Gebäude 079 des Instituts für Informatik der Uni Freiburg stellen sich etwa die Werte $\sigma_{p(d|z)} = 0.50\,\mathrm{m}$ für die globale Lokalisierung sowie $\sigma_{p(d|z)} = 0.05\,\mathrm{m}$ für das Tracking als optimal heraus. Der durchschnittliche Positionsfehler für diese beiden Standardabweichungen ist in Abbildung 6.6 zu sehen, die Fehlerrate bei der globalen Lokalisierung in Abbildung 6.9. Für die Ermittlung des optimalen Werts für die globale Lokalisierung wurden sehr zeitaufwändige Experimente durchgeführt, da hinreichend viele globale Lokalisierungsversuche mit hinreichend vielen verschiedenen Startpositionen durchgeführt werden müssen.

Sind die Standardabweichungen für die beiden Lokalisierungsprobleme einmal ermittelt, dann kann man ein Sensormodell verwenden, das von der Standardabweichung der globalen Lokalisierung zur Standardabweichung der Positionsverfolgung wechselt, sobald das Konvergenzkriterium erfüllt wird. In unseren Versuchen haben wir dieses wie folgt festgelegt: Wenn es kein Sample mehr gibt, das mehr als 1 m von

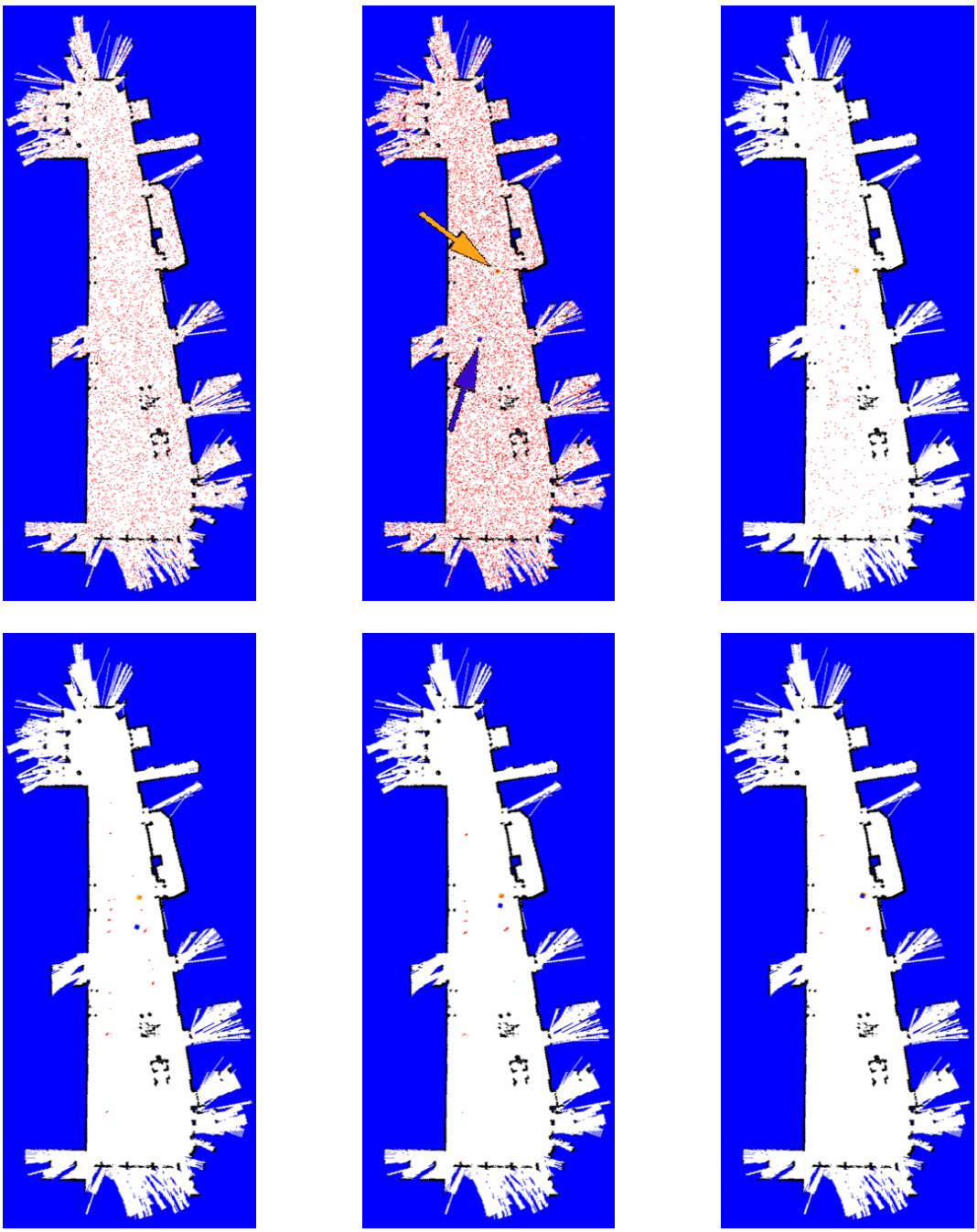
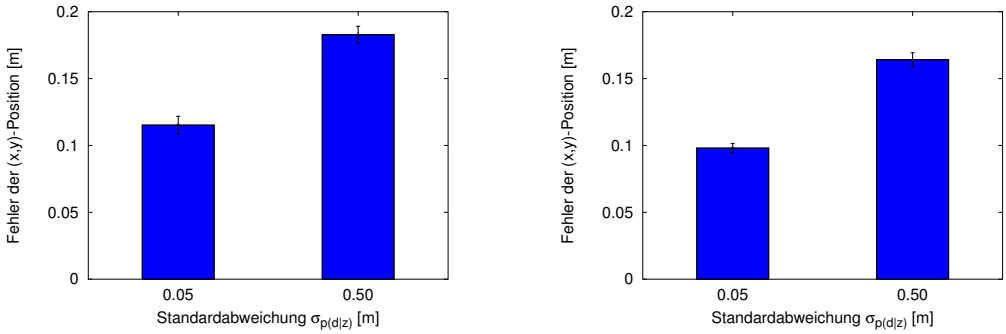


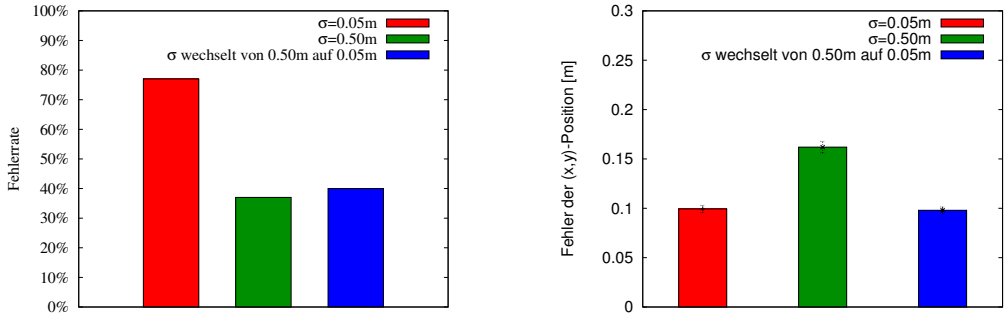
Abbildung 6.5.: Gezeigt werden die ersten sechs Resampling-Schritte einer globalen Lokalisierung im Gebäude 101 mit KLD-Sampling. Die anfängliche Samplezahl wurde auf 5000 gesetzt. Diese Samplezahl wird zunächst erhöht, da die gewünschte Approximationsgüte nicht gewährleistet werden kann (oberste Zeile, mittleres Bild). Die gelbe Markierung zeigt die tatsächliche Position des Roboters, die blaue Markierung die lokalisierte Position. Die Samples werden durch die roten Punkte dargestellt.



(a) min. 100 Samples

(b) min. 300 Samples

Abbildung 6.6.: Gezeigt wird der durchschnittliche Fehler der lokalisierten Position beim Tracking mit KLD-Sampling für die ermittelten, optimalen Standardabweichungen bei der globalen Lokalisierung und beim Tracking. Die Obergrenze für die Anzahl Samples wurde auf 10000 Samples festgelegt, die Untergrenze auf 100 bzw. 300 Samples gesetzt. Man erkennt deutlich, dass sich die für eine globale Lokalisierung optimale Standardabweichung von $\sigma_{p(d|z)} = 0.50\,\mathrm{m}$ nicht besonders gut für die Positionsverfolgung eignet.



(a) Globale Lokalisierung

(b) Tracking

Abbildung 6.7.: Gezeigt wird die Fehlerrate bei der (a) globalen Lokalisierung und der durchschnittliche Fehler der lokalisierten Position beim (b) Tracking mit KLD-Sampling bei der Verwendung zweier Sensormodelle mit fester Standardabweichung und eines Modells mit wechselnder. Das wechselnde Sensormodell startet mit einer Standardabweichung von $\sigma_{p(d|z)} = 0.50\,\mathrm{m}$ und wechselt, sobald das vorgestellte Konvergenzkriterium erfüllt wird, auf eine Standardabweichung von $\sigma_{p(d|z)} = 0.05\,\mathrm{m}$. Beim KLD-Sampling wurde die Untergrenze für die Anzahl an Samples auf 300 Samples und die Obergrenze auf 20000 gesetzt.

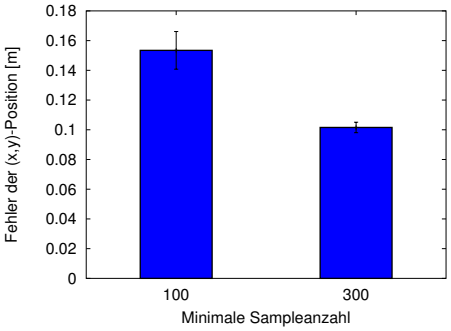


Abbildung 6.8.: Gezeigt wird der durchschnittliche Tracking-Fehler beim KLD-Sampling mit dem dynamisch adaptierenden Sensormodell während der globalen Lokalisierung und der Histogramm-Klassenbreite von 0.05 m als Standarabweichung für das Tracking. Die Obergrenze für die Anzahl Samples wurde auf 10000 Samples festgelegt, die Untergrenze auf 100 bzw. 300 Samples gesetzt.

der lokalisierten Position entfernt ist, dann gilt die globale Lokalisierung als abgeschlossen. Das gewichtete, arithmetische Mittel der Samplepositionen wird dabei für die lokalisierte Position verwendet. Das resultierende Sensormodell vereint auf diese Weise die Vorteile eines für die globale Lokalisierung optimalen Sensormodells und die eines für das Tracking optimalen. Die Ergebnisse von Versuchen mit diesem Sensormodell zeigt Abbildung 6.7.

Hat man in der Praxis die optimalen Werte nicht zur Verfügung, kann man als Alternative auch ein dynamisch adaptierendes Sensormodell verwenden. Dazu wählt man die anfängliche Standardabweichung $\sigma_{p(d|z)}$ relativ groß (z. B. mindestens 0.50 m) und aktualisiert diese in jeder Aktualisierungsphase des Partikelfilters – d. h. in jener Phase des Filterungsprozesses, in welcher die gemessenen Distanzen mit den zu erwartenden Distanzen verglichen werden. Bei der Verwendung von M Laserscanner-Strahlen erhält man auf diese Weise für jede gemessene Distanz eine Standardabweichung $\sigma^i_{p(d|z)}$ $(1 \leq i \leq M)$. Die Idee ist nun folgende: Nimmt man nun in jedem Resampling-Schritt das Minimum dieser Werte zur Berechnung eines neuen Bewertungs-Histogramms, solange das Konvergenzkriterium noch nicht erfüllt wird, und wählt man, sobald das Kriterium erfüllt wird, die Histogramm-Klassenbreite als Wert für die Standardabweichung der Bewertungsfunktion, dann erhält man einen sehr guten Kompromiss, der gleichzeitig eine hohe Erfolgsquote bei der globalen Lokalisierung sowie eine hohe Genauigkeit bei der Positionsverfolgung gewährleistet. Abbildung 6.8 zeigt, dass in unseren Versuchen die Genauigkeit dieses Verfahrens ziemlich genau mit der Genauigkeit der optimalen Standardabweichung (siehe Abbildung 6.6 (b)) übereinstimmt, falls mindestens 300 Samples verwendet werden. Auch die Erfolgsquote bei der globalen Lokalisierung liegt nahe bei der Erfolgsquote der dafür optimalen Standardabweichung, was Abbildung 6.10 verdeutlicht. Bei der Verwendung einer Obergrenze von mehr als 10000 Samples lässt sich in

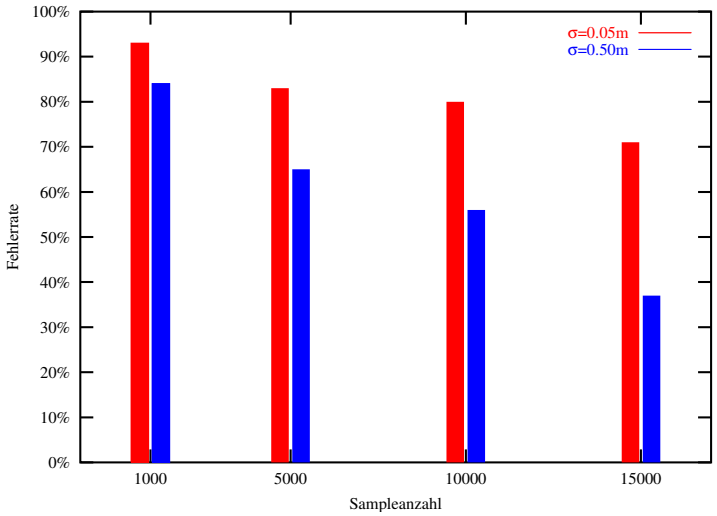


Abbildung 6.9.: Die Abbildung zeigt die Fehlerrate bei der globalen Lokalisierung mit KLD-Sampling, bei dem die Obergrenze für die Sampleanzahl auf die angegebene Zahl begrenzt wurde. Die für das Tracking optimale Standardabweichung von $\sigma_{p(d|z)} = 0.05\,\mathrm{m}$ liefert eine wesentlich höhere Fehlerrate als die Standardabweichung von $\sigma_{p(d|z)} = 0.50\,\mathrm{m}$.

der Abbildung keine Verbesserung erkennen. Dieser Effekt entsteht dadurch, dass mit zunehmender Anzahl an Samples bei relativ großer Standardabweichung das Konvergenzkriterium immer später erfüllt wird. Erfolgt der Test auf Erfolg bei der globalen Lokalisierung zu früh – was hier der Fall ist –, fällt dieser Test trotz eines möglichen späteren Erfolgs oft negativ aus.

Wie die Experimente in diesem Kapitel gezeigt haben, eignet sich das von uns vorgestellte Sensormodell hervorragend zur Lokalisierung mobiler Roboter in realen Umgebungen. Sowohl bei der Positionsverfolgung als auch bei der globalen Lokalisierung liegen die ermittelten Positionen sehr nahe an der wahren Position.

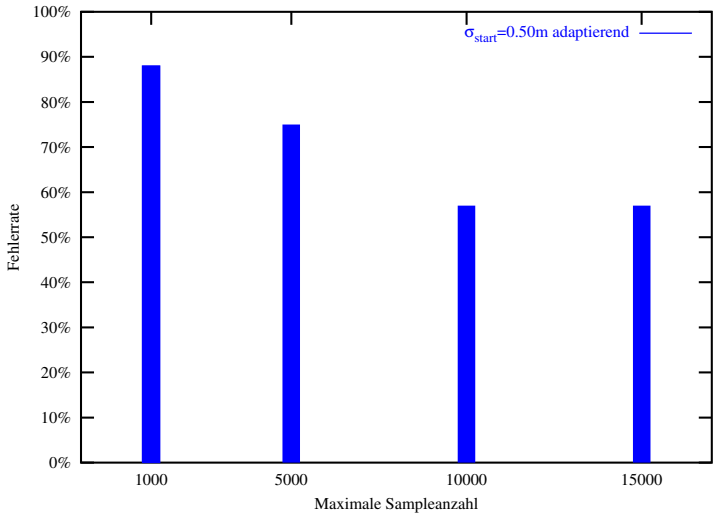


Abbildung 6.10.: Die Abbildung zeigt die Fehlerrate bei der globalen Lokalisierung mit KLD-Sampling und dem adaptierenden Sensormodell während der globalen Lokalisierungsphase. Gestartet wird dazu mit einer Standardabweichung von $\sigma_{p(d|z)} = 0.50\,\mathrm{m}$. Diese wird dynamisch adaptiert, bis das Konvergenzkriterium erfüllt ist. Danach dient die Histogramm-Klassenbreite von $\sigma_{p(d|z)} = 0.05\,\mathrm{m}$ als Standardabweichung für das Tracking.

7. Zusammenfassung

In dieser Arbeit wurde ein dynamisch adaptierendes Sensormodell für die Lokalisierung mobiler Roboter mit Abstandssensoren vorgestellt. Dabei haben wir uns zunächst auf die Theorie und die Arbeitsweise der Monte-Carlo-Lokalisierung konzentriert, um die Ansprüche seitens des Partikelfilters an das Sensormodell herauszustellen: Zuverlässigkeit bzw. eine hohe Erfolgsquote bei der globalen Lokalisierung sowie Präzision, Robustheit und eine hohe Aktualisierungsrate bei der Positionsverfolgung. Gleichzeitig haben wir aber auch die für Abstandssensoren in einer realen, unzugänglichen und dynamischen Umgebung typischen Fehler, Ungenauigkeiten und Einflüsse analysiert, um deutlich zu machen, welche Faktoren im Sensormodell Berücksichtigung finden müssen. Vor allem unter dem Aspekt einer geringen Zeit- und Speicher-Komplexität stellen diese Ansprüche eine hohe Herausforderung an die Entwicklung eines solchen Modells dar.

Um die Komplexität des Lokalisierungssystems zu reduzieren, haben wir innerhalb des Sensormodells verschiedene Ansätze zur Performance-Steigerung verfolgt. Denn mit dem Verzicht auf eine speicherintensive und zu einem gewissen Teil unnötige Vorberechung aller zu erwartenden Distanzen, werden sehr teure Ray-Casting-Operationen in die Phase des Lokalisierunsprozesses transferiert. Um die Zeitkomplexität dennoch möglichst niedrig zu halten, haben wir verschiedene aus der Computergrafik stammende Techniken zur Beschleunigung der Ray-Casting-Prozedur miteinander verglichen und schließlich mit dem Mask-Space-Leaping-Ray-Casting die Möglichkeit geschaffen, auch zur Laufzeit eine effiziente Berechnung der zu erwartenden Längen durchzuführen.

Wir haben gelernt, dass – obgleich man die Roboterlokalisierung meist auf die Probleme globale Lokalisierung, Positionsverfolgung und Roboter-Kidnapping aufteilt – die Lokalisierung doch fließend von der Phase der globalen Lokalisierung in die Tracking-Phase übergeht und auch dass das Verlieren der tatsächlichen Position nicht plötzlich erfolgt, sondern vielmehr allmählich vonstatten geht. Bedenkt man auch, dass sich verschiedene Umgebungen, in welcher der Roboter operiert, erheblich in ihrer Beschaffenheit voneinander unterscheiden können, dann kann man daraus und aus den durchgeführten Experimenten folgern, dass kein universelles, statisches Sensormodell dazu in der Lage ist, in jeder Umgebung und in jeder Phase des Lokalisierungsprozesses optimale Ergebnisse zu erzielen. Daher haben wir die zur Bewertung der Samples eingesetzte Wahrscheinlichkeitsfunktion des Sensormodells in zwei Komponenten zerlegt: in eine von der Umgebung charakterisierte, statische Komponente, die man mit geringem Aufwand aus den für das Mapping erforderlichen Daten

lernen kann, und in eine adaptierende Komponente, die sich dem Fortschritt der Lokalisierung anpasst. Auf diese Weise konnte gleichzeitig die Erfolgswahrscheinlichkeit der globalen Lokalisierung bei gleicher oder geringerer Sampleanzahl erhöht und die Genauigkeit bei der Positionsverfolgung verbessert werden.

Das auf diese Weise entstandene Lokalisierungssystem konnte seine Praxistauglichkeit bereits unter Beweis stellen. Gerade dann, wenn beispielsweise im Bereich der Service-Robotik autonome Roboter in Museen, auf Ausstellungen oder auf Messen eingesetzt werden und daher die Fähigkeit besitzen müssen, sich schnell und speziell an eine neue Umgebung anzupassen, bietet ein solches Lokalisierungssystem erhebliche Vorteile.

7.1. Ausblick

Sicherlich kann der Ray-Casting-Prozess zur Laufzeit noch verbessert werden. Vor allem durch intelligente Caching-Strategien oder dynamische Programmierung sollte es möglich sein, die innerhalb eines Resampling-Schritts beim Ray-Casting zur Gewinnung der Strahllängen durchgeführten Mehrfachberechnungen, welche sich u. a. durch die Zell- und Winkel-Diskretisierungen ergeben, zu eliminieren. Dies dürfte gerade während des Trackings, d. h. wenn die Samples sich in fokussiertem Zustand befinden, viele Ray-Casting-Operationen überflüssig machen.

Denkt man den Ansatz der dynamischen Adaption des Sensormodells noch weiter, stellt man unmittelbar fest, dass durch den Grad der Dynamik in ein und derselben Umgebung auch die darin stattfindenden Abstandsmessungen betroffen sind. Steigt beispielsweise in einem Museum die Besucherzahl stark an, nimmt auch der Anteil an Short-Readings bei den Messungen zu. Außerdem ist es naheliegend, dass sich ein autonomer, kybernetischer Museumsführer auf seiner Tour durch verschiedenartige Räume – man denke da an Spiegel, Glas, Gitter, Geländer etc. – bewegt, auf welche sich das verwendete Sensormodell einstellen könnte. In diesem Zusammenhang ist auch die Integration des Parameterlernens in den Lokalisierungsprozess vorstellbar. Der Roboter würde dann hinsichtlich des Sensormodells wahrhaftig seine Umgebung *kennenlernen*, d. h. sein Sensormodell anhand der anfallenden Daten kalibrieren.

Im Laufe der letzten Jahre ging der Trend beim Mapping jedoch mehr und mehr in Richtung Merkmalsextraktion. Die Merkmalsextraktion stellt eine Alternative zu den Umgebungskarten dar, die in Zellen den Belegungsgrad angeben und hier in dieser Arbeit eingesetzt wurden. Ziel der Merkmalsextraktion ist es, aus den Rohdaten der Abstandssensoren Punktemodelle zu erzeugen und aus diesen wiederum Merkmale wie Linien, Flächen oder geometrische Körper zu extrahieren – je nachdem, ob 2D- oder 3D-Mapping erfolgt. Die durch die Merkmalsextraktion gewonnenen Karten (engl. Feature-Based Maps) haben vor allem den Vorteil, dass sie ohne eine

Diskretisierung in Zellen auskommen und im Idealfall nur relevante Merkmale enthalten und deshalb extrem kompakt sind. Angesichts des doch sehr niedrigen Informationsgehaltes von Umgebungskarten mit Belegungsgrad, könnte man sich daher für die Lokalisierung Merkmalskarten wünschen, welche die Umgebung ausreichend detailliert wiedergeben und als zusätzliches Feature z. B. die Oberflächenbeschaffenheit oder die Durchlässigkeit von Objekten dem Lokalisierungssystem zur Verfügung stellen würden.

Des Weiteren kann man sich darauf einstellen, dass in den nächsten Jahren in immer stärkerem Maß 3D-Verfahren gefragt sein werden. Heute schon gelangt man nämlich mit 2D-Karten an die Grenzen der Repräsentation von Umgebungen. Ein mobiler Roboter mit einer räumlichen Ausdehnung erhält mit einer 2D-Karte, welche seine Umgebung auf Höhe der Abstandssensoren repräsentiert, nur eine ungenügende Vorstellung von der umgebenden Welt. Dies lässt in Kombination mit zu wenigen, für die Kollisionsvermeidung zuständigen Sensoren viele „tote Winkel“ enstehen und hat in der Vergangenheit nicht selten zu Zusammenstößen während der Navigation geführt. Ein 3D-Laserscanner, welcher Treppenstufen, Absätze sowie „beinlose“ Objekte detektieren vermag, könnte hier Abhilfe schaffen. Schön zu wissen, dass sich die in dieser Arbeit vorgestellten Verfahren leicht zur 3D-Lokalisierung erweitern lassen.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und sämtliche Stellen, die wörtlich oder sinngemäß aus veröffentlichter oder unveröffentlichter Literatur entnommen wurden, als solche kenntlich gemacht habe. Außerdem erkläre ich, dass die Arbeit nicht – auch nicht auszugsweise – bereits für eine andere Prüfung angefertigt wurde.

(Dirk Zitterell)
Freiburg, den 20. April 2004

A. Grundlagen der Stochastik

In diesem Abschnitt wollen wir die Grundlagen der Wahrscheinlichkeitsrechnung, die in dieser Arbeit Verwendung finden, kurz zusammenfassen (siehe auch [Krengel 1998; Stöcker 1999; Bosch 1999; Henze 2000]).

Definition der bedingten Wahrscheinlichkeit

Sei (Ω, P) ein Wahrscheinlichkeitsraum und $B \in \Omega$ ein Ereignis mit $P(B) > 0$. Dann definieren wir die bedingte Wahrscheinlichkeit von $A \in \Omega$ durch:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{A.1}$$

und bezeichnen $P(A|B)$ als Wahrscheinlichkeit von A unter der Bedingung (Voraussetzung) von B.

Satz von der totalen Wahrscheinlichkeit

Sei (Ω, P) ein Wahrscheinlichkeitsraum und seien $A_1, A_2, \ldots, A_n$ disjunkte Ereignisse mit $P(A_i) > 0, i = 1, 2, \ldots, n$ und $\sum_{i=1}^{n} A_i = \Omega$, was als Zerlegung von Ω bezeichnet wird. Dann gilt für jedes Ereignis B:

$$P(B) = \sum_{i=1}^{n} P(B|A_i)\ P(A_i) \tag{A.2}$$

Satz von Bayes (Bayes'sche Regel)

Sei (Ω, P) wiederum ein Wahrscheinlichkeitsraum und $A_1, A_2, \ldots, A_n$ eine Zerlegung desselben. Falls $P(B) > 0$, so gilt für jedes $j = 1, 2, \ldots, n$:

$$P(A_j|B) = \frac{P(B|A_j)\ P(A_j)}{P(B)} \tag{A.3}$$

und nach der Formel der totalen Wahrscheinlichkeit (A.2):

$$P(A_j|B) = \frac{P(B|A_j)\ P(A_j)}{\sum_{i=1}^{n} P(B|A_i)\ P(A_i)} \tag{A.4}$$

Definition der Likelihood-Funktion

Sei $(X_1, \ldots, X_k)$ eine zufällige Stichprobe aus einer stetigen Verteilung, die durch die Dichtefunktion $f(x|\Theta)$ und durch den Parametervektor Θ beschrieben wird, dann ist die Likelihood-Funktion wie folgt definiert:

$$\mathcal{L}(\Theta) := \prod_{j=1}^{k} f(X_j; \Theta) \tag{A.5}$$

Definition der Multinomialverteilung

Der Zufallsvektor $(X_1, \ldots, X_k)$ besitzt eine Multinomialverteilung mit Parametern n und $p_1, \ldots, p_k$ $(k \geq 2, n \geq 1, p_1 \geq 0, \ldots, p_k \geq 0, p_1 + \cdots + p_k = 1)$, falls für $i_1, \ldots, i_k \in \mathbb{N}_0$ mit $i_1 + \cdots + i_k = n$ die Identität

$$P(X_1 = i_1, \ldots, X_k = i_k) = \frac{n!}{i_1! \cdot i_2! \cdot \ldots \cdot i_k!} \cdot p_1^{i_1} \cdot p_2^{i_2} \cdot \ldots \cdot p_k^{i_k} \tag{A.6}$$

gilt; andernfalls setzen wir $P(X_1 = i_1, \ldots, X_k = i_k) := 0$. Für einen multinomialverteilten Zufallsvektor schreiben wir kurz

$$(X_1, \ldots, X_k) \sim \text{Mult}(n; p_1, \ldots, p_k) \tag{A.7}$$

Das Likelihood-Verhältnis

Der Likelihood-Verhältnis-Test (engl. Likelihood-Ratio-Test) dient zum Vergleich von Modellen auf Grundlage des Maximum-Likelihood-Schätzverfahrens. Insbesondere wird ein Ausgangsmodell mit einem Modell, in welchem den Parametern Restriktionen auferlegt wurden, verglichen. Der Test soll nun entscheiden, ob das Ausgansmodell signifikant besser ist, als das restringierte Modell. Sei also $\mathcal{L}_u(\Theta)$ die Likelihood-Funktion des unrestringierten und $\mathcal{L}_r(\Theta)$ die Likelihood-Funktion des restringierten Modells, dann wird das Likelihood-Verhältnis wie folgt definiert:

$$LR := 2 \log \frac{\mathcal{L}_u(\Theta)}{\mathcal{L}_r(\Theta)} \tag{A.8}$$

Die $\mathcal{X}^2$-Minimum-Methode

Die $\mathcal{X}^2$-Funktion dient der Bestimmung des bestmöglichen Parameterwertes a, falls die Art der Verteilung vorgegeben ist. Ausgehend von einer Tabelle relativer Häufigkeiten $h_j = h(K_j)$ mit vorgegebener Klassifizierung $K_j, j = 1, \ldots, k$, ist die $\mathcal{X}^2$-Funktion wie folgt definiert:

$$\mathcal{X}^2 := \left\{ \sum_{j=1}^{k} \frac{h_j - f(X_j; a)}{f(X_j; a)} \right\} \tag{A.9}$$

wobei $f(x; a)$ eine ideale Verteilung zu gegebenem Parameterwert a und X_j die Intervallmitte der j-ten Klasse darstellt.

Beim $\mathcal{X}^2$-Minimumprinzip definiert das Minimum der $\mathcal{X}^2$-Funktion bezüglich a – analog zum Maximum-Likelihood-Schätzer – den Paramterschätzwert $\hat{a}$ für a und damit die Schätzfunktion $f(x; \hat{a})$, falls man das Modell $f(x; a)$ vorgibt.

Die $\mathcal{X}^2$-Verteilung (Helmert-Pearson)

Unter der Stichprobenfunktion versteht man die Abbildung der Messwerte $x_1, \ldots, x_n$ einer Stichprobe auf einen Wert $W_n(x_1, \ldots, x_n)$, mit dessen Hilfe eine Eigenschaft der Grundgesamtheit abgeschätzt oder überprüft werden soll.
Die Verteilung einer Stichprobenfunktion $f(W_n)$ ergibt sich durch das mehrfache Wiederholen des Auswahlvorgangs, wie z. B. durch das mehrmalige Ziehen von n Losen.
Unter der $\mathcal{X}^2$-Verteilung versteht man die Verteilung $f_{\mathcal{X}}(Y_n; n)$, die sich für die Messgröße $Y_n(x_1, \ldots, x_n)$ mit

$$W_n = Y_n(x_1, \ldots, x_n) := \mathcal{X}^2 = \sum_{i=1}^{n} x_i^2 \tag{A.10}$$

ergibt, wenn die einzelnen Messwerte x_i $(i = 1, \ldots, n)$ jeweils standardnormalverteilt sind:

$$f(x_i) = f_{\mathrm{sn}}(x_i). \tag{A.11}$$

Die Wahrscheinlichkeitsdichte der $\mathcal{X}^2$-Verteilung berechnet sich wie folgt:

$$f_{\mathcal{X}}(Y_n; n) = \frac{1}{2^{n/2}\Gamma(n/2)} Y_n^{n/2-1} e^{-Y_n/2}. \tag{A.12}$$

B. Technische Daten

Der Roboter

Abbildung B.1.: Gezeigt wird der mit einem SICK Laserscanner ausgestattete Roboter Pioneer 2DX von Active Media Robotics.

Geschwindigkeit	
Translation	max. 1600 mm/Sek.
Rotation	max. 300°/Sek.
Batterien	3×12 VDC
Größe (L×B×H)	44 cm×33 cm×22 cm
Gewicht	9 kg
Nutzlast	20 kg

Tabelle B.1.: Technische Daten des Roboters Pioneer 2DX von Active Media Robotics.

Der Laserscanner

Abbildung B.2.: Gezeigt wird der Laserscanner LMS 291 der Firma SICK.

Reichweite	max. 80 m
Scanwinkel	max. 180°
Messauflösung	10 mm
Winkelauflösung	0, 25°/ 0, 5°/ 1° (einstellbar)
Versorgungspannung	24 VDC ± 15%
Leistungsaufname	ca. 20 W
Größe (B×H×T)	155 mm×185 mm×156 mm
Gewicht	4,5 kg

Tabelle B.2.: Technische Daten des Laserscanners SICK LMS 291

Abbildungsverzeichnis

Literaturverzeichnis

[**Amanatides und Woo 1987**] Amanatides, J.; Woo, A.: A Fast Voxel Traversal Algorithm for Ray Tracing. In: *Proc. Eurographics '87*, 1987, S. 1–10

[**Bosch 1999**] Bosch, Karl: *Elementare Einführung in die Wahrscheinlichkeitsrechung*. Vieweg, 1999

[**Bresenham 1965**] Bresenham, E. J.: Algorithm for Computer Control of a Digital Plotter. In: *IBM System Journal* 4 (1965), Nr. 1

[**Burgard et. al. 1998**] Burgard, W.; Cremers, A. B.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; Thrun, S.: The Interactive Museum Tour-Guide Robot. In: *Proc. of the National Conference on Artificial Intelligence*, 1998

[**Burgard et. al. 2000**] Burgard, W.; Cremers, A.B.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; Thrun, S.: Experiences with an Interactive Museum Tour-Guide Robot. In: *Artificial Intelligence* 114 (2000), Nr. 1-2

[**Burgard et. al. 1997**] Burgard, W.; Fox, D.; Hennig, D: Fast Grid-Based Position Tracking for Mobile Robots. In: *Proc. of the 21th German Conference on Artificial Intelligence, Germany*, 1997

[**Burgard et. al. 1996**] Burgard, W.; Fox, D.; Hennig, D.; Schmidt, T.: Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids. In: *Proc. of the National Conference on Artificial Intelligence*, 1996

[**Burgard et. al. 2002**] Burgard, W.; Trahanias, P.; Hähnel, D.; Moors, M.; Schulz, D.; Baltzakis, H.; A., Argyros: TOURBOT and WebFAIR: Web-Operated Mobile Robots for Tele-Presence in Populated Exhibitions. In: *Proc. of the IROS'02 Workshop on Robots in Exhibitions*, 2002

[**Carpenter et. al. 1999**] Carpenter, J.; Clifford, P.; Fearnhead, P.: An Improved Particle Filter for Non-linear Problems. In: *IEE Proceedings on Radar, Sonar and Navigation* Bd. 146, 1999, S. 2–7

[**Choset et. al. 2004**] Choset, Howie; Lynch, Kevin; Hutchinson, Seth; Kantor, George; Burgarg, Wolfram; Kavraki, Lydia; Thrun, Sebastian: *Principles of Robot Motion: Theory, Algorithms, and Implementation*. N.N., 2004. – noch nicht erschienen

[**Choset und Nagatani 2001**] Choset, Howie; Nagatani, K.: Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. In: *IEEE Transactions on Robotics and Automation* 17 (2001), April, Nr. 2, S. 125–137

[**Dellaert et. al. 1999**] Dellaert, F.; Burgard, W.; Fox, D.; Thrun, S.: Using the Condensation Algorithm for Robust, Vision-based Mobile Robot Localization. In: *Proc. of the IEEE ComputerSocietyConference on Computer Vision and Pattern Recognition (CVPR'99)*, 1999

[**Ferguson et. al. 2003**] Ferguson, D.; Morris, A.; Hähnel, D.; Baker, C.; Omohundro, Z.; Reverte, C.; Thayer, S.; Whittaker, W.; Burgard, W.; Thrun, S.: An Autonomous Robotic System for Mapping Abandoned Mines. In: Thrun, S. (Hrsg.); Saul, L. (Hrsg.); Schölkopf, B. (Hrsg.): *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, MIT Press, 2003

[**Fox 2001**] Fox, D.: KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization / Department of Computer Science & Engineering, University of Washington, Seattle. 2001. – Forschungsbericht

[**Fox 2003**] Fox, D.: Adapting the Sample Size in Particle Filters Through KLD-Sampling. In: *International Journal of Robotics Research (IJRR)* (2003)

[**Fox et. al. 1999a**] Fox, D.; Burgard, W.; Dellaert, F.; Thrun, S.: Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In: *Proc. of the National Conference on Artificial Intelligence*, 1999

[**Fox et. al. 1999b**] Fox, D.; Burgard, W.; Thrun, S.: Markov Localization for Mobile Robots in Dynamic Environments. In: *Journal of Artificial Intelligence Research* 11 (1999)

[**Fox et. al. 2000**] Fox, D.; Thrun, S.; Dellaert, F.; Burgard, W.: Particle Filters for Mobile Robot Localization. In: Doucet, A. (Hrsg.); Freitas, N. de (Hrsg.); Gordon, N. (Hrsg.): *Sequential Monte Carlo Methods in Practice*. New York : Springer Verlag, 2000

[**Gailly und Adler 2004**] Gailly, Jean-loup; Adler, Mark: *ZLIB – A Massively Spiffy Yet Delicately Unobtrusive Compression Library*. 2004. – URL `http://www.gzip.org/zlib/`. – Stand 24. März 2004

[**Gutmann et. al. 1998**] Gutmann, J.-S.; Burgard, W.; Fox, D.; Konolige, K.: An Experimental Comparison of Localization Methods. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998

[**Gutmann und Fox 2002**] Gutmann, J.-S.; Fox, D.: An Experimental Comparison of Localization Methods Continued. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002

[**Gutmann und Schlegel 1996**] Gutmann, J.-S.; Schlegel, C.: AMOS: Comparison of scan matching approaches for self localization in indoor environments. In: *Proc. of the 1st Euromicro Workshop on Advanced Mobile Robots*, IEEE Computer Society Press, 1996

[**Henze 2000**] Henze, Norbert: *Stochastik für Einsteiger*. Vieweg, 2000

[**Hesselink et. al. 1999**] Hesselink, W. H.; Meijster, A.; Roerdink, J. B. T. M.: An exact Euclidean distance transform in linear time / Institute for Mathematics and Computing Science, University of Groningen, the Netherlands. April 1999 (IWI 99-9-04). – Forschungsbericht

[**Konolige und Chou 1999**] Konolige, Kurt; Chou, Ken: Markov Localization using Correlation. In: Dean, Thomas (Hrsg.): *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, Morgan Kaufmann, 1999, S. 1154–1159. – ISBN 1-55860-613-0

[**Krengel 1998**] Krengel, Ulrich: *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg, 1998

[**Kristensen und Jensfelt 2003**] Kristensen, Steen; Jensfelt, Patric: An Experimental Comparison of Localisation Methods, the MHL Sessions. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'03)*, 2003, S. 992–997

[**Kwok et. al. 2003a**] Kwok, C.; Fox, D.; Meila, M.: Adaptive Real-Time Particle Filters for Robot Localization. In: *Proc. of the IEEE International Conference on Robotics & Automation*, 2003

[**Kwok et. al. 2003b**] Kwok, Cody; Fox, Dieter; Meila, Marina: Real-Time Particle Filters. In: S. Becker, S. T. (Hrsg.); Obermayer, K. (Hrsg.): *Advances in Neural Information Processing Systems 15*. Cambridge, MA : MIT Press, 2003, S. 1057–1064

[**Meijster et. al. 2000**] Meijster, A.; Roerdink, J.B.T.M.; Hesselink, W.H.: A General Algorithm for Computing Distance Transforms in Linear Time. In: Goutsias, J. (Hrsg.); Vincent, L. (Hrsg.); Bloomberg, D.S. (Hrsg.): *Mathematical Morphology and its Applications to Image and Signal Processing*. Kluwer, 2000, S. 331–340

[**Montemerlo et. al. 2002**] Montemerlo, M.; Pineau, J.; Roy, N.; Thrun, S.; Verma, V.: Experiences with a Mobile Robotic Guide for the Elderly. In: *Proceedings of the AAAI National Conference on Artificial Intelligence*. Edmonton, Canada : AAAI, 2002

[**Montemerlo et. al. 2003**] Montemerlo, Michael; Roy, Nicholas; Thrun, Sebastian: Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)* Bd. 3. Las Vegas, NV, October 2003, S. 2436–2441

[**Moravec und Elfes 1985**] Moravec, Hans P.; Elfes, A. E.: High Resolution Maps from Wide Angle Sonar. In: *Proc. IEEE Int. Conf. Robotics and Automation*, 1985, S. 116–121

[**NASA 2004**] NASA: *Mars Exploration Rover Mission*. 2004. – URL `http://marsrovers.jpl.nasa.gov/home/index.html`. – Stand 27. März 2004

[**Nourbakhsh et. al. 1995**] Nourbakhsh, Illah; Powers, Rob; Birchfield, Stan: Dervish: An Office-Navigating Robot. In: *AI Magazine* 16 (1995), Nr. 2

[**Rice 1995**] Rice, John A.: *Mathematical Statistics and Data Analysis*. second edition. Duxbury Press, Belmont, California, 1995

[**Roy und Thrun 1999**] Roy, Nicholas; Thrun, Sebastian: Online Self-Calibration For Mobile Robots. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999

[**Russell und Norvig 1995**] Russell, Stuart J.; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ : Prentice Hall, 1995

[**Schiele und J.Crowley 1994**] Schiele, B.; J.Crowley: A Comparison of Position Estimation Techniques Using Occupancy Grids. In: *IEEE International Conference on robotics and Automation*, 1994

[**Simmons 2004**] Simmons, Reid: *Inter Process Communication (IPC)*. 2004. – URL `http://www-2.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html`. – Stand 24. März 2004

[**Stöcker 1999**] Stöcker, Horst (Hrsg.): *Taschenbuch mathematischer Formeln und moderner Verfahren*. Harri Deutsch, 1999

[**Taylor 1995**] Taylor, Barry N.: *Guide for the Use of the International System of Units (SI)*. NIST Special Publication 811. 1995

[**The Robocup Federation 2004**] The Robocup Federation: *Robocup*. 2004. – URL `http://www.robocup.org/`. – Stand 24. März 2004

[**Thrun et. al. 2000**] Thrun, S.; Beetz, M.; Bennewitz, W.; Cremers, A.B.; Dellaert, F.; Fox, D.; Haehnel, C.; Roy, N.; Schulte, J.; Schulz, D.: Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva, 2000

[**Thrun et. al. 2003**] Thrun, Sebastian; Diel, Mark; Hähnel, Dirk: Scan Alignment and 3-D Surface Modeling with a Helicopter Platform. In: *Proc. of the 4th International Conference on Field Robotics*, 2003

[**Wolf 2001**] Wolf, J.: *Bildbasierte Lokalisierung für mobile Roboter*, Albert-Ludwigs-Universität Freiburg, Diplomarbeit, 2001

[**Wolf et. al. 2002**] Wolf, J.; Burgard, W.; Burkhardt, H.: Robust Vision-based Localization for Mobile Robots using an Image Retrieval System Based on Invariant Features. In: *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002

[**Yagel und Shi 1993**] Yagel, R.; Shi, Z.: Accelerating Volume Animation by Space-Leaping. In: *Proceedings of Visualization '93*, 1993, S. 62–69