Autonome gründliche Exploration unbekannter Innenräume mit dem mobilen Roboter „Robbie“

Studienarbeit im Fach Computervisualistik

vorgelegt
von

Stephan Wirth

Geboren am 13.9.1981 in Solingen

Angefertigt am

Institut für Computervisualistik
Arbeitsgruppe Aktives Sehen
Universität Koblenz–Landau

Betreuer: Dietrich Paulus, Johannes Pellenz

Beginn der Arbeit: 01.01.2007

Abgabe der Arbeit: 24.04.2007

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Koblenz, den 24.04.2007

Inhaltsverzeichnis

Kapitel 1

Einleitung

Damit mobile Roboter autonom komplexe Aufgaben erfüllen können, müssen sie Wissen über den Aufbau ihre Umgebung besitzen [Thr98]. Dieses Wissen wird ihnen entweder in für sie nutzbarer Form direkt übergeben, oder sie müssen es sich selbst aneignen. Ersteres ist meist schwierig, da die Daten erst aufwändig erzeugt bzw. umgewandelt werden müssen, damit sie für den Roboter lesbar werden. Zudem kann es vorkommen, dass sich die Welt, in der sich der Roboter bewegt, zwischen der Erzeugung dieser Daten und dem Einsatz des Roboters verändert. Vergleicht man zum Beispiel den Grundriss eines Gebäudes, wie er vom Architekten erstellt wurde, mit dem tatsächlichen Aufbau, so sieht man schnell, dass diese meist nicht übereinstimmen. Die Stellung von Türen und Möbeln zum Beispiel können sich verändern.

In solchen sich verändernden und in völlig unbekannten Szenarien ist eine Explorations-Strategie erforderlich, welcher der Roboter folgt; eine Strategie zur Navigation in bereits bekanntem und zur effektiven Exploration von noch unbekanntem Gelände.

Bei Katastrophenszenarien, wie sie beim RoboCupRescue-Wettbewerb[1] simuliert werden, ist eine aktive Exploration essenziell. Nach einem Erdbeben oder einer Explosion sind vorher akquirierten Daten eines Gebäudes unbrauchbar. Roboter, die autonom bei der Suche nach Überlebenden helfen, müssen sich daher selbst zurecht finden.

[1] siehe `http://www.rescuesystem.org/robocuprescue/`

Der Roboter „Robbie“ der Arbeitsgruppe Aktives Sehen hat bereits im Rahmen des Projektpraktikums „Robbie-6“[2] erfolgreich am RoboCupRescue 2006 in Bremen teilgenommen, allerdings wurde er dort noch von einem Menschen ferngesteuert. Diese Arbeit hat zum Ziel, den Roboter alle ihm zugänglichen Bereiche einer unstrukturierten Umgebung autonom und möglichst vollständig erkunden zu lassen.

Das allgemeine Explorationsproblem lässt sich wie folgt beschreiben (vgl. [GBL02]):

Gegeben das bisher akquirierte Wissen über die Umwelt, wohin soll sich der Roboter als nächstes bewegen, um noch mehr Informationen über seine Umgebung zu erhalten?

Dieses Problem lässt sich in zwei Fragen aufteilen:

1. Die der Zielsuche: *Wo soll der Roboter hinfahren?* und
2. die der Pfadsuche: *Welchen Weg soll er dabei nehmen?*

Zunächst muss als Grundlage ein für den Roboter geeignetes Modell gefunden werden, in das die Welt als Karte abgebildet werden kann. Wichtig dabei ist, dass er sich gut orientieren und beim autonomen Fahren für sich zugängliche Bereiche identifizieren kann. Zwei Arten von Modellen haben sich bislang hierfür durchgesetzt: Topologische Ansätze und rasterbasierte Ansätze. Topologische Karten [Cho96, KB91] speichern Eigenschaften der Umgebung in Graphen. Knoten des Graphen sind zum Beispiel gut zu identifizierende Landmarken (Ecken, Linien etc.) oder wichtige Navigationspunkte (Durchgänge, Türen). Eine Verbindung zwischen zwei Knoten bedeutet, dass der Roboter auf direktem Weg von dem einen zum anderen gelangen kann.

Rasterbasierte Karten gehen auf die von Alberto Elfes eingeführten Occupancy Grids [Elf89] zurück. Ein Occupancy Grid ist eine meist zwei- oder dreidimensionale uniforme gitterförmige Aufteilung des Raumes in einzelne Zellen. Jede dieser Zellen enthält einen Wert, der angibt, mit welcher Wahrscheinlichkeit die Zelle durch ein Objekt belegt ist. Ein Vergleich der Vor- und Nachteile der beiden Ansätze und eine Kombination dieser findet sich in [Thr98]. Die daraus entnommene Tabelle 1.1 gibt hierüber einen Überblick.

[2] siehe `http://www.uni-koblenz.de/FB4/Institutes/ICV/AGPaulus/Researches/ActiveVision/Robbie-6`

Tabelle 1.1: Vor- und Nachteile von rasterbasierten und topologischen Ansätzen zur Kartengenerierung (aus [Thr98])

Rasterbasierte (metrische) Ansätze	**Topologische Ansätze**
+ einfach aufzubauen, darzustellen und zu pflegen + Wiedererkennen von Plätzen (Geometrie-basiert) ist nicht mehrdeutig und unabhängig vom Blickwinkel + vereinfacht die Berechnung des kürzesten Pfades	+ erlaubt effiziente Planung, geringer Speicheraufwand (Auflösung hängt von der Komplexität der Umgebung ab) + erfordert keine genaue Bestimmung der Position des Roboters + gebräuchliche Representation für symbolische Planer / Problemlöser, natürliche Sprache
- ineffiziente Planung, hoher Speicheraufwand (Auflösung hängt nicht von der Komplexität der Umgebung ab) - erfordert genaue Bestimmung der Position des Roboters - schlechtes Interface für die meisten symbolischen Problemlöser	- schwer zu Erstellen und zu pflegen in weiträumigen Umgebungen, wenn Sensorinformationen mehrdeutig sind - Wiedererkennen von Plätzen oft sehr schwierig, vom Blickwinkel abhängig - kann suboptimale Pfade ergeben

Die Erstellung von Karten durch Roboter ist bekannt als das SLAM-Problem (Simultaneous Localization and Mapping) [GK99, MCNT99]: Der Roboter muss seine Position in der bisher erstellten Karte finden (Lokalisation) und gleichzeitig diese mit Hilfe seiner Sensor-Informationen (z. B. die eines Laserscanners) erweitern (Mapping).

Im Projektpraktikum „Robbie-6“ wurde bereits ein einfaches, inkrementelles Verfahren zur Erstellung von Occupancy Grids implementiert. Diese Occupancy Grids bilden die Grundlage für die hier vorliegende Arbeit und werden in Kapitel 2.1 näher erläutert.

Brian Yamauchi stellt in [Yam97] einen Grenzen-basierten Ansatz zur Findung des nächsten Explorations-Zieles vor, welcher Occupancy Grids benutzt. Die Schlüsselidee dabei ist einfach: Um neue Informationen zu bekommen, bewege dich zu den Grenzen zwischen unbekanntem und bekanntem Gelände. Hierzu werden Bereiche, in denen Grenzen vor-

kommen, aus den Occupancy Grids extrahiert und gezielt angefahren. Hector González-Baños und Jean-Claude Latombe verfolgen in [GBL02] das Ziel, einen Punkt auf der Karte zu finden, von dem aus die Sensorinformationen des Roboters optimal zur Erweiterung der Karte ausgenutzt werden können (Next-Best-View-Problem). Hierzu werden Kandidaten von Punkten in der Nähe von Grenzen zu unbekannten Gebieten erzeugt und die Größe der Fläche berechnet, die vom jeweiligen Punkt aus über die Grenzen hinaus von den Sensoren erfasst werden könnte. Der Punkt, von dem aus die Größte Fläche sichtbar ist, wird als nächster Zielpunkt angefahren.

Die Suche nach dem kürzesten Pfad zwischen zwei Knoten eines Graphen mit gewichteten Kanten ist in der Informatik ein bekanntes Problem und kann zum Beispiel durch den Dijkstra-[Dij59] oder den A*-Algorithmus[HNR68] gelöst werden. Um aus einem Occupancy Grid einen Graphen zu erzeugen, wird jede einzelne Zelle als Knoten angesehen. Aneinander angrenzende Zellen werden mit einer Kante verbunden, die als Gewicht den Abstand der Zellen oder zusätzlich die Belegtheitswahrscheinlichkeit der Zellen trägt.

Zelinsky führt in [Zel88, Zel91] die Pfadtransformation (Path Transform) ein. Diese speichert in jeder Zelle die Pfadkosten zum Ziel, berechnet aus der Distanz zum Ziel plus der gewichtete Summen aller „Gefährlichkeitswerte“ der auf dem Pfad liegenden Zellen. Ein Gefährlichkeitswert berechnet sich aus der Nähe des nächsten Hindernisses. Je näher das nächste Hindernis ist, desto höher ist der Gefährlichkeitswert. Um nun einen sicheren Pfad zum Ziel zu finden, muss nur vom Startpunkt aus dem höchsten Gradienten der Pfadtransformation nachgegangen werden, bis das Ziel erreicht wurde.

Thruns Ansatz in [Thr98] verwendet zur Exploration direkt die Werte des Occupancy Grids um Pfadkosten zu berechnen. Die Kosten für das Betreten einer Zelle entsprechen der Belegtheitswahrscheinlichkeit dieser Zelle. Ausgehend von unbekannten Bereichen werden mit Hilfe des Verfahrens der Value Iteration für jede Zelle die Kosten des Pfades zur nächsten Grenze bestimmt. Zellen, die nahe an Zellen mit hoher Belegtheitswahrscheinlichkeit liegen, werden im Gegensatz zur Pfadtransformation hier nicht in der Pfadsuche ausgeschlossen bzw. als gefährlich eingestuft. Dadurch kann es dazu kommen, dass der berechnete Pfad zur nächsten Grenze für den Roboter nicht befahrbar ist.

In dieser Arbeit wird die Pfadtransformation Zelinskys zu einer Explorationstransformation (Exploration Transform) erweitert, indem nicht die Distanz zum Ziel, sondern die

Distanz zu den Grenzen, wie Yamauchi sie definiert, verwendet wird. So verläuft der absteigende Gradient der Explorationstransformation von jeder Zelle aus zu einer nahe liegenden Grenze. Zellen werden außerdem nicht nur als gefährlich eingestuft, wenn sie nah am nächsten Hindernis liegen, sondern auch, wenn sie weit vom nächsten Hindernis entfernt sind, damit die Reichweite der Sensoren des Roboters nicht überschritten wird. Die Lokalisation wird somit vereinfacht.

Der Aufbau der Arbeit ist wie folgt: Kapitel 2 beschreibt Occupancy Grids und die darauf basierenden untersuchten Verfahren zur Ziel- und Wegfindung. In Kapitel 3 folgt die Beschreibung des hier verfolgten Ansatzes. Experimente und Ergebnisse werden in Kapitel 4 aufgeführt und das Kapitel 5 bildet abschließend eine Zusammenfassung der Arbeit.

Kapitel 2

Stand der Wissenschaft

2.1 Occupancy Grids

Occupancy Grids (*engl.* Belegtheitsgitter) wurden von Elfes in [Elf89] als universelle Karte für mobile Roboter vorgeschlagen, mit deren Hilfe Pfadplanung, Navigation und das Umfahren von Hindernissen ermöglicht wird. Es handelt sich dabei um eine meist zwei- oder dreidimensionale gitterförmige Aufteilung des Raumes in einzelne Zellen. Die Größe einer Zelle bestimmt die Genauigkeit, mit der der Raum im Grid erfasst wird. Zu jeder dieser Zellen z_i wird die Wahrscheinlichkeit p gespeichert, mit der die Zelle belegt ist. Der Zustandsraum einer einzelnen Zelle beschränkt sich auf die zwei Zustände *belegt*, notiert als $s(z_i) = \mathrm{OCC}$ und *frei*, notiert als $s(z_i) = \mathrm{FREE}$. Es gilt ferner

$$p(z_i) = p(s(z_i) = \mathrm{OCC}) = 1 - p(s(z_i) = \mathrm{FREE})$$

Zu Anfang erhält jede Zelle die Belegtheitswahrscheinlichkeit p_{init} (meist gilt $p_{\mathrm{init}} = 0.5$). Diese Art der Abbildung der Welt auf ein mehrdimensionales Gitter hat den Vorteil, dass sie keinerlei Einschränkungen besitzt, was die Beschaffenheit der Welt angeht. Während topologische Karten meist nur Punkte, Linien, Rechtecke oder sonstige Polygone aufnehmen können, liegen dem Occupancy Grid keine speziellen Strukturen zu Grunde.

Der Nachteil von Occupancy Grids liegt in deren Speicheraufwand. Da die Einteilung der Zellen in festen Abständen vollzogen wird, kann die Menge der Zellen je nach Auflösung

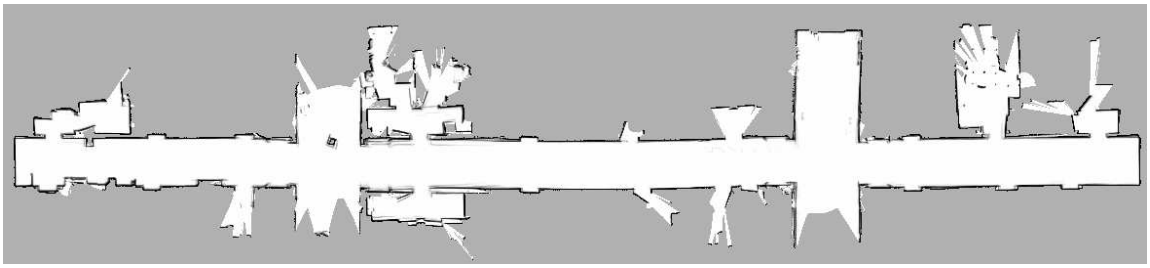


Bild 2.1: Occupancy Grid des 3. Stockwerks des B-Gebäudes der Universität in Koblenz. Je heller eine Zelle dargestellt ist, desto geringer ist die Wahrscheinlichkeit, dass an ihrer Position ein Hindernis steht. Die Seitenlänge einer Zelle entspricht hier 5 cm.

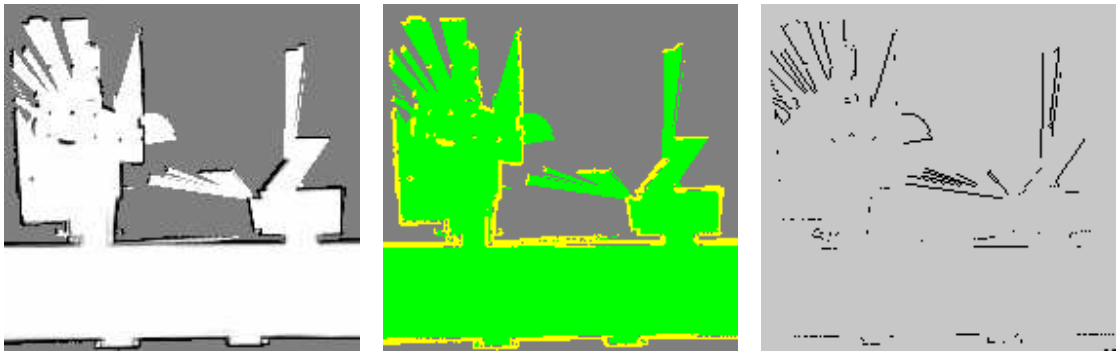
des Gitters schnell sehr groß werden. Der Speicheraufwand steigt nicht mit der Komplexität der Umgebung, sondern mit deren Ausmaß.

Auf Basis eines Occupancy Grids ist ein Pfad Z definiert als eine endliche Folge von n Zellen z_i, welche mit dem Startknoten Zelle $z_\mathrm{s} = z_0$ beginnt und mit dem Endknoten Zelle $z_\mathrm{g} = z_n$ endet, wobei für jede Zelle z_i außer z_g gilt: z_i und z_{i+1} sind im Occupancy Grid benachbart. Eine Menge von Pfaden sei im Folgenden notiert als χ.

2.2 Grenzen-basierte Exploration

Exploration im Sinne der Robotik ist der Prozess der Erkundung von unbekanntem Gelände ausgehend von bereits bekanntem. Ist ein Startpunkt in einer explorierten Umgebung gegeben, muss also ein Ziel und ein Weg dorthin berechnet werden, bei dessen Erreichen der Roboter durch Sensorinformationen sein Wissen über die Umwelt vergrößern kann.

Yamauchi schlägt in [Yam97] einen Grenzen-basierten Ansatz zur autonomen Exploration vor (Frontier-Based Exploration). Eine Grenze ist eine Zelle des Occupancy Grids, welche *frei* ist und eine *unbekannte* Zelle als Nachbarn besitzt. Regionen von zusammenhängenden Grenzzellen, die groß genug sind, dass der Roboter sie passieren könnte, werden als mögliche Ziele markiert. Der Roboter wählt als nächstes Ziel die ihm am nächsten liegende Region. Das Ziel wird angefahren und die Karte durch Messungen von Ultraschall-Sensoren und Laserscanner erweitert. Ist die Anfahrt eines Zieles zum Beispiel



(a) (b) (c)

Bild 2.2: Kartenausschnitt, die daraus bestimmten Occupancy Grid-Klassen: frei (grün), belegt (gelb) und unbekannt (grau) und extrahierte Grenzen

durch dynamisch auftretende Hindernisse nicht möglich, wird das Ziel als nicht erreichbar markiert. Danach wird aus der jetzt erweiterten Karte wieder die nächste Region von Grenzen bestimmt und der Roboter fährt diese an. So erkundet der Roboter nach und nach alle Grenzen, bis die erstellte Karte die Umgebung vollständig abbildet.

Das als Karte verwendete Occupancy Grid wird in folgender Weise benutzt, um Grenzen zwischen freien und unbekannten Bereichen zu bestimmen. Die Zellen des Occupancy Grid werden in die drei Klassen *frei*, *unbekannt* und *belegt* aufgeteilt (vgl. auch [Elf89]). Als Grundlage für die Aufteilung wird die Belegtheitswahrscheinlichkeit p_i einer Zelle z_i verwendet. Ist p_i größer als die initiale Belegtheitswahrscheinlichkeit p_{init}, wird die Zelle z_i als belegt markiert, ist sie kleiner, so wird die Zelle z_i als frei markiert. Gilt $p_{\text{init}} = p_i$, so ist dies ein Indiz dafür, dass der Roboter noch keine Informationen über diese Zelle hat und sie fällt daher in die Klasse der unbekannten Bereiche. Bild 2.2(b) verdeutlicht diese Klassifizierung.

Jede Zelle, die frei ist und eine unbekannte Zelle als Nachbarn hat, ist eine Grenze. Die im Folgenden vorgestellten Verfahren basieren ebenfalls auf so klassifizierten Karten.

2.3 Distanztransformation

Das Problem der Wegsuche von einem beliebigen Startpunkt zu einem festen Ziel bei bekannter Karte wurde von Jarvis und Byrne [JB86] mit der Distanztransformation gelöst. Die Distanztransformation eines Belegtheitsgitters speichert für jede freie Zelle die Anzahl der Zellen, die durchquert werden müssen, um von dort aus zum Ziel zu gelangen, bzw. die nötigen Bewegungskosten, die auf dem kürzesten Pfad zum Ziel entstehen würden. Die Bewegungskosten von einer Zelle zu einer anderen werden durch deren Abstand bestimmt. Es existieren verschiedene Methoden, diesen Abstand zu berechnen: Seien z_i und z_j zwei Zellen, zwischen denen sich kein Hindernis befindet. Die verschiedenen Abstandsmaße sind dann definiert als (vgl. [RP68]):

- Manhattan-Distanz (auch Taxi- oder City-Block-Distanz):

$$d_{\text{manhattan}}(z_i, z_j) = |z_i^x - z_j^x| + |z_i^y - z_j^y|$$

- Schachbrett-Distanz (Chessboard Distance):

$$d_{\text{chess}}(z_i, z_j) = \max(|z_i^x - z_j^x|, |z_i^y - z_j^y|)$$

- Euklid'scher Abstand:

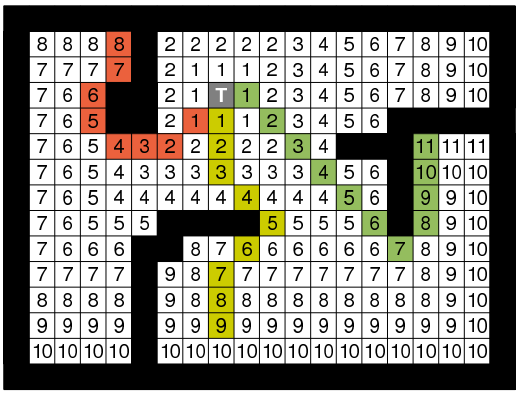
$$d_{\text{euclid}}(z_i, z_j) = \sqrt{(z_i^x - z_j^x)^2 + (z_i^y - z_j^y)^2}$$

.

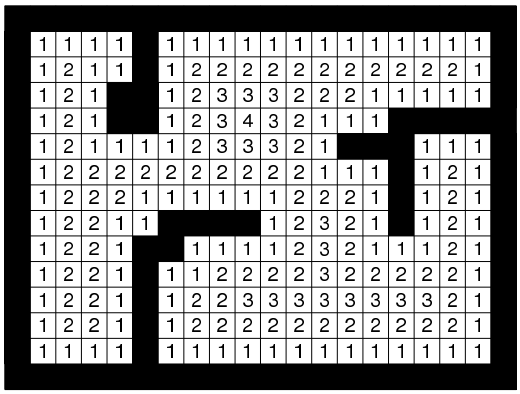
Bild 2.3(a) zeigt eine Distanztransformation mit der Schachbrett-Distanz.

Zur Berechnung der Distanztransformation werden zu Anfang alle ihre Zellen z_i mit dem Distanzwert $w(z_i) = \infty$ belegt, die Ziel-Zelle erhält den Wert 0. Die folgenden Schritte werden dann sooft wiederholt, bis die Distanztransformation sich nicht mehr ändert:

- Wähle für jede freie Zelle z_i aus ihren acht Nachbarzellen diejenige Zelle z_j aus, welche frei ist und den kleinsten Wert $w(z_j)$ besitzt.



(a) Von jeder Zelle aus kann dem steilsten absteigenden Gradienten gefolgt werden, um zum Ziel zu gelangen, wie es die drei eingezeichneten Wege verdeutlichen.



(b) Jede Zelle speichert die Distanz zum nächsten Hindernis.

Bild 2.3: Distanztransformation und Hindernistransformation.

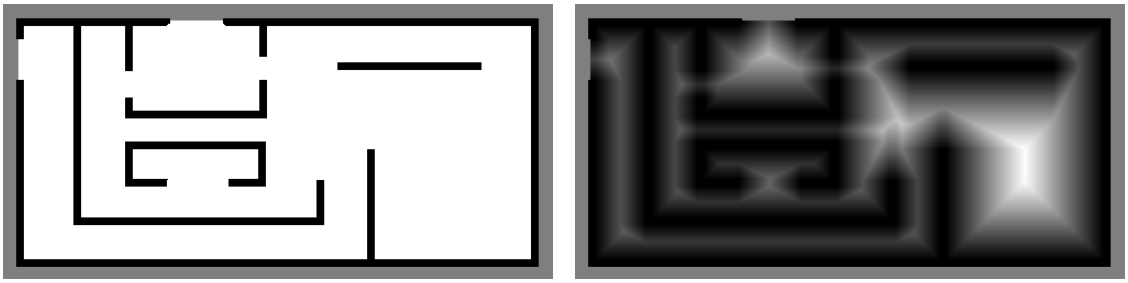
- Berechne den Abstand d zwischen diesen Zellen gemäß einer der oben genannten Distanzfunktionen.
- Wenn $w(z_j) + d < w(z_i)$, setze $w(z_i) = w(z_j) + d$

Nun kann von jeder beliebigen freien Zelle aus der kürzeste Pfad zum Ziel gefunden werden, indem vom Startpunkt aus dem steilsten absteigenden Gradienten gefolgt wird.

2.4 Pfadtransformation

Alexander Zelinskys Pfadtransformation [Zel88, Zel91] erweitert die Distanztransformation um die Komponente der Sicherheit. Da die Distanztransformation stets den kürzesten Pfad liefert, verläuft dieser oft sehr nah an Hindernissen vorbei und ist für Roboter daher nur bedingt geeignet, denn der Pfad muss nicht nur kurz, sondern auch gefahrlos abzufahren sein.

Für die Pfadtransformation wird zuerst die „Hindernistransformation“ (Obstacle Transform) Ω erstellt. Diese ist sehr ähnlich zur Distanztransformation, nur wird nicht von ei-



(a) (b)

Bild 2.4: Occupancy Grid und die dazu gehörige Hindernistransformation. In Bild (b) gilt: je heller die Zelle, desto weiter ist ihr Abstand vom nächsten Hindernis.

nem Ziel ausgehend die Distanz betrachtet, sondern von jeder belegten Zelle aus. Bild 2.3(b) zeigt die zu Bild 2.3(a) passende Hindernistransformation. Die so berechnete Transformation enthält in jeder Zelle als Wert die Distanz zum der Zelle am nächsten liegenden Hindernis. Bild 2.4 zeigt eine weitere Karte mit zugehöriger Hindernistransformation.

Die Pfadtransformation Φ einer Zelle z für das Erreichen des Ziels z_g ist definiert als

$$\Phi(z) = \min_{Z \in \chi} \left(l(Z) + \alpha \sum_{z_i \in Z} c_\mathrm{danger}(z_i) \right) \tag{2.1}$$

mit

- χ Menge aller möglichen Pfade von z nach z_g
- $l(Z)$ Länge des Pfades Z
- $c_\mathrm{danger}(z_i)$ Kostenfunktion für die „Gefährlichkeit“ der Zelle z_i
- α Gewichtungsfaktor ≥ 0

Die Länge $l(Z)$ eines Pfades Z kann auf verschiedene Weise berechnet werden. Meist wird sie inkrementell bestimmt:

$$l(Z) = l(z_0 \ldots z_n) = \sum_{i=0}^{n-1} d(z_i, z_{i+1})$$

Dabei bestimmt d die Distanz zwischen zwei Zellen. Zelinsky verwendet hier, wie auch zur Berechnung der Distanztransformation, die Schachbrett-Distanz.

Die Kostenfunktion c_{danger} wird mit Hilfe der Hindernistransformation Ω bestimmt. Sie ist ein Maß für die Nähe des nächsten Hindernisses. Zelinsky schlägt dafür eine kubische Funktion vor, die ab einer festen Distanz den Wert 0 annimmt und für kleiner werdende Distanzen ansteigt. Ein Beispiel für solch eine Funktion ist folgende:

$$c_{\text{danger}}(z_i) = \begin{cases} (X - \Omega(z_i))^3, & \text{wenn} \quad \Omega(z_i) \leq X \\ 0, & \text{sonst} \end{cases}$$

Die Konstante X gibt die Größe der Gefahrenzone an.

Das Gewicht α in Gleichung 2.1 bestimmt wie stark bei der Wegsuche Abstand zu Hindernissen gehalten wird.

Die Pfadtransformation speichert so in jeder Zelle die minimalen Kosten für den Weg zum Ziel. Folgt man nun wie bei der Distanztransformation dem Gradienten des größten Abstieges, gelangt man von jeder Zelle aus auf dem Pfad der geringsten Kosten zum Ziel. Da hier zusätzlich zur Länge des Pfades die Entfernung zum nächsten Hindernis jedes einzelnen Pfadpunktes optimiert wird, ist der entstehende Weg sicherer zu befahren, als der durch die Distanztransformation berechnete, wobei mit dem Gewicht α der „Trade-Off“ zwischen Sicherheit und schneller Erreichbarkeit gewählt wird.

Kapitel 3

Eigener Ansatz

Das Anfangs beschriebene Explorationsproblem lässt sich durch die Kombination der Arbeiten von Yamauchi und Zelinsky lösen. Die Pfadtransformation wird zu einer „Explorationstransformation“ erweitert, bei der nicht die Kosten für den Pfad zu einem Ziel in jeder Zelle gespeichert werden, sondern die Kosten für den Pfad, der am günstigsten zu einer nahen Grenze führt. Der Pfad muss nicht unbedingt der kürzeste sein und die Grenze nicht die nächste, da die Kosten insgesamt über die Länge *und* die Sicherheit des Pfades berechnet werden.

Von jeder einzelnen Zelle der Explorationstransformation aus gelangt man durch Folgen des am stärksten absteigenden Gradienten zur nächsten sicher erreichbaren Grenze.

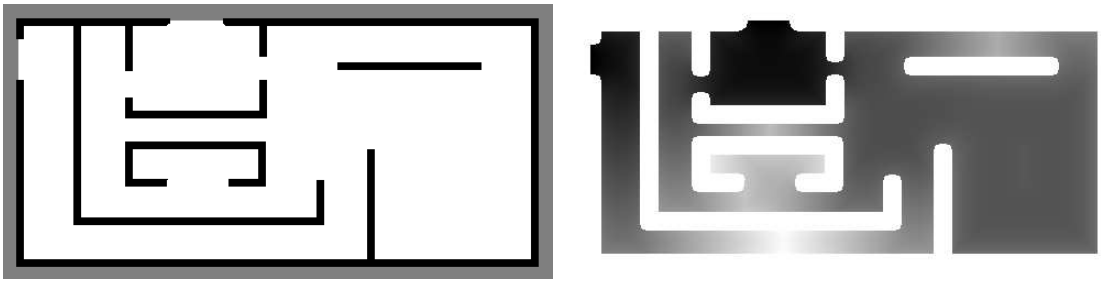
3.1 Explorationstransformation

Die Explorationstransformation Ψ ist definiert als

$$\Psi(z) = \min_{z_\mathrm{g} \in F} \left(\min_{Z \in \chi_z^{z_\mathrm{g}}} \left(l(Z) + \alpha \sum_{z_i \in Z} c_\mathrm{danger}(z_i) \right) \right)$$

mit

- F Menge aller Grenzzellen



(a) (b)

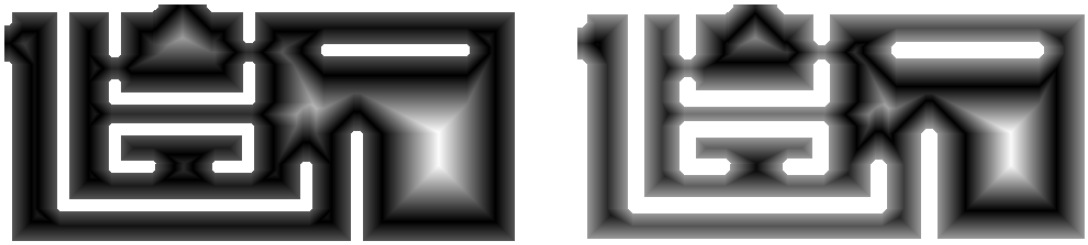
Bild 3.1: Occupancy Grid (a) und zugehörige Explorationstransformation (b). Je dunkler eine Zelle im rechten Bild dargestellt ist, desto niedriger ist der Wert der Explorationstransformation.

- $\chi_z^{z_\mathrm{g}}$ Menge aller möglichen Pfade von z nach z_g
- $l(Z)$ Länge des Pfades Z
- $c_\mathrm{danger}(z_i)$ Gefahrenkosten für das Betreten der Zelle z_i
- α Gewichtungsfaktor ≥ 0 (analog zur Pfadtransformation)

Ergänzend zur Pfadtransformation kommt bei der Explorationstransformation die Suche nach der nächsten Grenze hinzu. Besteht die Menge F aller Grenzzellen nur aus einem Element z, so ist die Explorationstransformation gleich der Pfadtransformation zum Ziel z. Betrachtet man die Karte aus Bild 3.1(a) und ihre Explorationstransformation in Bild 3.1(b) so sieht man, dass die Explorationstransformation in allen den Grenzen zu unbekanntem Gebiet nahen Bereichen kleine Werte annimmt.

Die Funktion der Gefahrenkosten c_danger wird erweitert, um nicht nur Bereiche nahe an Hindernissen zu meiden, sondern auch Bereiche, die sehr weit vom nächsten Hindernis entfernt sind. Diese Modifikation ist für die Kartenerstellung wichtig, da nur mit Hilfe der Sensorinformationen des Roboters seine Position bestimmt werden kann. Befindet sich der Roboter so weit von Hindernissen entfernt, dass seine Sensoren diese nicht mehr erreichen können, wird die Lokalisation schwieriger.

Sei d_min die Distanz zu Hindernissen, die der Roboter nicht unterschreiten darf und sei d_opt die optimale Distanz für die Navigation. Die Gefahrenkosten c_danger einer Zelle z mit



(a) (b)

Bild 3.2: Gefahrenkosten für unteschiedliche Parameter. (a): $d_{\min} = 3$, $d_{\text{opt}} = 20$, (b): $d_{\min} = 5$, $d_{\text{opt}} = 30$

der Distanz d zum nächsten Hindernis berechnen sich wie folgt:

$$c_{\text{danger}}(z) = \begin{cases} \infty, & \text{wenn } d < d_{\min} \\ |d_{\text{opt}} - d|, & \text{sonst.} \end{cases}$$

Der mit dieser Gefahrenkosten-Funktion berechnete Pfad hält, wo es möglich ist, immer die Distanz d_{opt} zum nächsten Hindernis. Die Bilder 3.2(a) und 3.2(b) zeigen die Gefahrenkostenfunktion für die Karte in Bild 2.4(a) mit unterschiedlichen Parametern.

Als Alternative zu der von Zelinsky verwendeten Schachbrett-Distanz, wird hier die präzisere euklid'sche Distanz verwendet. In Bild 3.3 wird der Unterschied deutlich. Distanzen über die Diagonalen von Zellen sind bei der euklid'schen Distanz kleiner als bei der Schachbrettdistanz. Durch diese Änderung erhält der Pfad weniger Kurven und diese sind zudem auch weicher.

Der Gewichtungsfaktor α der Explorationstransformation bestimmt, wie sehr ein sicherer Pfad einem kurzen Pfad vorgezogen wird. Die Abbildungen 3.4(a) und 3.4(b) zeigen unterschiedliche Pfade für verschiedene α.

3.2 Algorithmus der Explorationstransformation

Die Berechnung der Explorationstransformation erfolgt ähnlich zum Algorithmus von Dijkstra vom Ziel aus. Statt zu jeder freien Zelle denjenigen Pfad zu einer Grenze zu be-

(a) (b)

Bild 3.3: Hindernistransformation mit der Schachbrettdistanz (a) und mit der euklid'schen Distanz.

(a) (b)

Bild 3.4: Explorationstransformation mit unterschiedlicher Gewichtung der Kosten. In (a) wird ein kurzer, in (b) ein sicherer Weg bevorzugt.

stimmen, der die Kosten minimiert, wird von den Grenzen ausgehend eine Art „Welle“ - ähnlich zu einem Flood-Fill-Algorithmus - erzeugt, die die Kosten entsprechend ihres Verlaufes inkrementiert.

```
01  Queue V
02  ExplorationTransform E
03
04  for all cells z in E
05    set E(z) to infinity
06  endfor
07
```

```
08  for all cells z in Map
09    if z is frontier
10      set E(z) to 0
11      for all neighbours n of z
12        add n to end of V
13      endfor
14    endif
15  endfor
```

Der Algorithmus benutzt eine Schlange (Queue) zur Abarbeitung der einzelnen Zellen. Zu Anfang wird allen Zellen der Explorationstransformation der Wert ∞ zugewiesen (Zeilen 4-6). In den Zeilen 8-15 wird die „Saat“ für den Flood-Fill-Algorithmus erzeugt, indem Grenzzellen den Wert 0 erhalten und ihre Nachbarn zur weiteren Untersuchung in die Schlange eingereiht werden.

```
16  while V not empty
17    set s to first element of V
18    remove first element from V
19    if s is marked as free in Map
20      find neighbour t with lowest cost E(t) that is free in Map
21      set c to E(t) + distance(s, t) + alpha * obstacleCost(t)
22      if c < E(s)
23        set E(s) to c
24        for all neighbours n of z
25          add n to end of V
26        endfor
27      endif
28    endif
29  endwhile
```

Die Zeilen 17-30 arbeiten die Schlange in folgender Weise ab: Für jede Zelle wird geprüft, ob ein Pfad über einen ihrer Nachbarn existiert, der geringere Kosten hat, als in der aktuellen Zelle gespeichert sind. Wenn ja, werden die Kosten der Zelle aktualisiert und

ihre Nachbarn in die Schlange eingereiht, da sich deren Kosten jetzt eventuell auch noch verändern können.

Die Funktion `distance(s, t)` berechnet die Distanz zwischen zwei angrenzenden Zellen. Mit `obstacleCost(t)` werden die Gefahrenkosten einer Zelle `t` wie oben angegeben bestimmt.

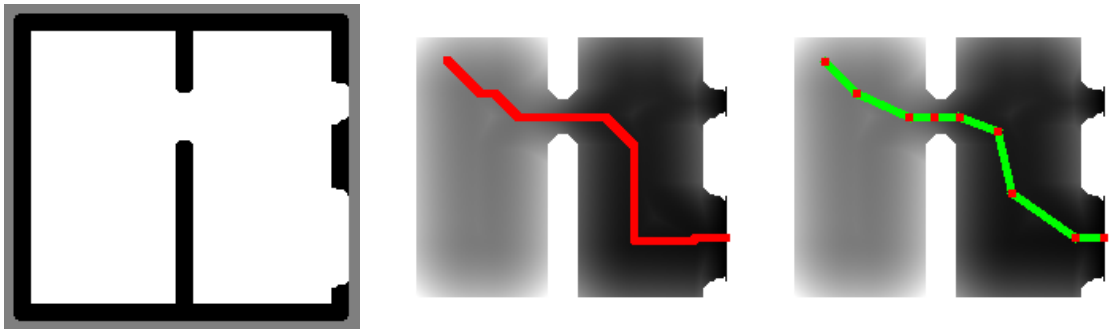
Nachdem der Algorithmus terminiert, enthält jede Zelle der Explorationstransformation die Kosten für den günstigsten Pfad zu einer nahen Grenze. Der Vorteil hiervon gegenüber der Bestimmung eines Pfades von einem festen Startpunkt aus ist, dass der Vorgang der Exploration unterbrochen und an jeder Stelle der bisher bekannten Gebiete wieder aufgenommen werden kann, ohne erneut Pfade berechnen zu müssen. Außerdem können mehrere Roboter gleichzeitig auf der selben Explorationstransformation arbeiten. Je nach Position der einzelnen Roboter werden sogar unterschiedliche Ziele für diese ausgewählt.

3.3 Pfadoptimierung

Ausgabe der Wegsuche ist eine Liste von aneinander angrenzenden Zellen im Occupancy Grid. Um unnötige Positionskorrekturen des Roboters zu vermeiden, werden aus dieser Liste Wegpunkte mit größeren Abständen extrahiert, die der Roboter abfahren soll.

In Regionen, bei denen es auf eine genaue Navigation ankommt - also in gefährlichen Bereichen - müssen mehr Wegpunkte vorliegen, als in offenen Bereichen, in denen nur die grobe Richtung eine Rolle spielt.

Zur Optimierung des Pfades werden daher so viele Wegpunkte aus der Liste aneinander angrenzender Zellen weggelassen, dass nachher bei jedem übrig gebliebenen Wegpunkt seine Distanz zum nächsten Wegpunkt höchstens so groß ist, wie seine Distanz zum nächsten Hindernis. Bild 3.5 zeigt einen Pfad aus 238 Punkten und seine vereinfachte Variante mit nur 9 Punkten.



(a) (b) (c)

Bild 3.5: Pfadoptimierung. (c) zeigt den optimierten Pfad zu (b). Man beachte, dass um die Verengung in der Mitte der Karte herum mehr Wegpunkte ausgewählt wurden, als in freien Bereichen.

3.4 Integration in die „Robbie“-Architektur

Der vorgestellte Algorithmus wurde in die bestehende Robotik-Softwarearchitektur der Arbeitsgruppe Aktives Sehen für den mobilen Roboter „Robbie“ (Bild 3.6) integriert.

Die „Robbie“-Software ist gegliedert in drei verschiedene Typen von Komponenten. Es gibt

- *Devices*, welche Gerätetreiber ansprechen bzw. implementieren und damit die Schnittstelle für das Ansprechen der Hardware bilden,
- *Worker*, welche Berechnungen auf den von *Devices* produzierten Daten ausführen und
- *Module*, welche *Devices* und *Worker* unterhalten, die Kommunikation zwischen solchen organisieren und den Zustand des Systems beschreiben.

Für diese Arbeit wurden ein neues Modul, das *NavigationModule*, und die beiden Worker *Explorer* und *SpeedControl* entworfen und implementiert. Das NavigationModule reagiert auf die beiden Nachrichten `StartNavigationM` und `StartExploringM`. Mit `StartNavigationM` wird dem Modul aufgetragen, den Roboter zu einer bestimmten

Bild 3.6: „Robbie“: Der Roboter der Arbeitsgruppe Aktives Sehen

Position zu navigieren, bei der Nachricht `StartExploringM` wählt das Modul mit Hilfe des Workers Explorer selbst anhand der Explorationstransformation ein Explorationsziel aus und navigiert den Roboter dort hin. Nach Erreichen des Zieles schickt das NavigationModule die Nachricht `TargetReachedM` bzw. `TargetUnreachableM` wenn das Ziel nicht erreichbar ist.

Folgende Stati nimmt das NavigationModule an:

- `IDLE`
  Das Modul ist abgeschaltet. Es wartet auf die Nachrichten `StartNavigationM` oder `StartExploringM`, alle übrigen Nachrichten, die das Modul erhält, werden verworfen.

- `AWAITING_MAP_FOR_NAVIGATION`
  Dieser Status wird angenommen, wenn das Modul im Status `IDLE` die Nachricht `StartNavigationM` erhält. Da als Grundlage für die Navigation eine Karte benötigt wird, wartet das Modul jetzt auf die Nachricht `MapDataM`, welche eine sol-

che Karte enthält. Kommt diese Nachricht an, wird mit Hilfe des Workers Explorer die Pfadtransformation und damit ein Pfad zum in der `StartNavigationM` übergebenen Ziel berechnet. Existiert ein Pfad zum Ziel, geht das Modul in den Status `AWAITING_POSITION_DATA` über, existiert keiner, so schickt das Modul die Nachricht `TargetUnreachableM` und kehrt in den Zustand `IDLE` zurück.

- `AWAITING_MAP_FOR_EXPLORATION`
  Erhält das Modul im Status `IDLE` die Nachricht `StartExploringM`, nimmt es diesen Status an. Die als Grundlage für die Explorationstransformation verwendete Karte befindet sich in der Nachricht `NavigationMapM`. Kommt diese Nachricht nun an, wird mit Hilfe der Explorationstransformation ein Pfad zu einer Grenze bestimmt und der Zustand `AWAITING_POSITION_DATA` angenommen. Existiert kein Pfad, sendet das Modul die Nachricht `NoTargetM` und wechselt zum initialen Zustand `IDLE`.

- `AWAITING_POSITION_DATA`
  In diesem Zustand wartet das Modul auf aktuelle Positionsnachrichten, welche vom SLAMModule verschickt werden. Nach Ankunft dieser Nachricht untersucht das Modul den abzufahrenden Pfad und löscht gegebenenfalls erreichte Wegpunkte. Ist kein Wegpunkt mehr übrig, beendet das Modul nach Schicken der Nachricht `TargetReachedM` den Ablauf, indem es in den Zustand `IDLE` zurückkehrt. Ansonsten wird der nächste Wegpunkt ausgewählt und Winkel und Distanz zu diesem berechnet. Übersteigt der Winkel eine gewisse Schwelle (hier $\pm 5°$), muss der Roboter sich neu auf den nächsten Wegpunkt ausrichten und eine entsprechende Nachricht `TurnPioneerM` wird verschickt. Ist der Winkel zum nächsten Wegpunkt klein genug, kann auf ihn zu gefahren werden. Mit Hilfe des Workers SpeedControl wird anhand aktueller Laser-Messungen berechnet, ob die anzustehende Bewegung ausgeführt werden könnte. Ist dem nicht so, so wird die Nachricht `TargetUnreachableM` verschickt und das Modul kehrt in den Zustand `IDLE` zurück. Ist die Bewegung möglich, schickt das Modul eine `MovePioneerM`-Nachricht.

- `WAITING_FOR_MOVE_FINISHED`
  Wurde im Zustand `AWAITING_POSITION_DATA` ein Bewegungsbefehl abgesen-

det, nimmt das Modul diesen Zustand an. Nach Erhalt einer der Nachrichten `MoveFinishedM` oder `TurnFinishedM` welchselt es wieder zum Zustand `AWAITING_POSITION_DATA`.

Die Unterscheidung zwischen Karten zur Navigation und Karten zur Exploration ermöglicht es, bei der Exploration zur Markierung abgesuchter Bereiche die Reichweite anderer Sensoren zu verwenden als jene, die zur Kartenerstellung benutzt werden. So können zum Beispiel in einer Karte zur Exploration nur diejenigen Bereiche als bekannt vermerkt werden, die von einem Wärmesensor erfasst werden, wie es für die Anwendung von „Robbie“ als Rettungsroboter gemacht wird. Wichtig ist nur, dass die Karte zur Exploration ebenfalls alle Hindernisse beinhaltet.

Die Parameter für die Explorationstransformation sind im Worker Explorer an die Hardware angepasst. Die minimale Distanz zu Hindernissen $d_{\min}$ entspricht dem Radius des Umkreises des Roboters von ca. 30 cm. Die optimale Distanz zum nächsten Hindernis ist abhängig vom verwendeten Sensor zur Lokalisation. „Robbie“ verfügt über einen Hokuyo URG-04LX Laserscanner mit dessen Hilfe die Lokalisation in Kombination mit der Kartenerstellung durchgeführt wird. Da der Laserscanner auf Grund seiner einfachen und Strom sparenden Bauweise nur eine Reichweite von 4 m besitzt, wurde für die optimale Distanz d_{opt} zum nächsten Hindernis ein Wert von 2 m verwendet.

Kapitel 4

Experimente und Ergebnisse

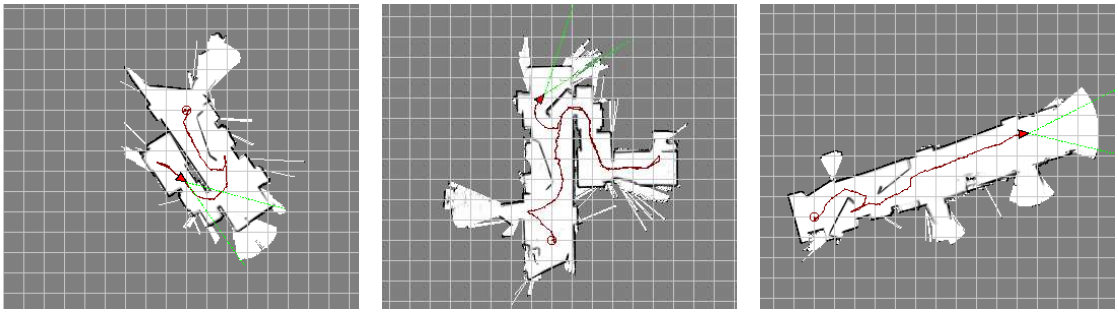
4.1 Versuchsaufbau und -durchführung

Verschiedene Szenarien wurden im Flur eines Bürogebäudes der Universität für die automatische Erkundung durch den Roboter vorbereitet. Die Umstellung verschiedener Gegenstände und Zwischenwände, sowie das Öffnen und Schließen von Türen ergaben unterschiedliche Umgebungen im selben Bereich des Gebäudes.

Der Roboter wurde in dem vorbereiteten Gelände durch das Senden einer `StartExploringM`-Nachricht an das NavigationModule gestartet. Sobald der Roboter am Ende des Explorationspfades ankam und das NavigationModule eine `TargetReachedM`-Nachricht verschickte, wurde wieder eine `StartExploringM`-Nachricht abgesendet. Dieser Kreislauf wurde sooft ausgeführt, bis das NavigationModule mit einer `NoTargetM`-Nachricht antwortete und damit die Exploration der Umgebung abgeschlossen war.

4.2 Ergebnisse

Drei der autonom erstellten Karten sind in Bild 4.1 zu sehen. Erwartungskonform folgt der Roboter im Experiment zu Bild 4.1(c) der großen und damit sicheren offenen Grenze auf der rechten Seite, statt durch die Türe im unteren Bereich zu fahren. Dabei hält er



(a) (b) (c)

Bild 4.1: Während der autonomen Exploration erstellte Karten. Die Rasterweite beträgt 1m, die Größe einer Zelle entspricht einer Fläche von 5×5 cm.

den dem System vorgegebenen optimalen Abstand von 80 cm. Bei den beiden anderen Experimenten ist ebenfalls zu erkennen, dass der Roboter an Weggabelungen den breiteren Weg nimmt.

Es lässt sich ein durchaus neugieriges, jedoch vorsichtiges Verhalten erkennen.

Probleme treten auf, wenn die Kartenerstellung fehl schlägt. Wird ein Hindernis nicht deutlich genug in die Karte eingetragen, kann es vorkommen, dass der Weg durch dieses hindurch gelegt wird. Durch die Kollisionsvermeidung bleibt der Roboter dann vor dem Hindernis stehen. Entwickelt sich die Karte im Folgenden weiter, wird das Hindernis darin erkennbar und eine erneute Anwendung der Explorationstransformation berechnet einen Weg um das Hindernis herum. Geschieht dies nicht, befindet sich das System in einem Zustand der gegenseitigen Blockade und der manuelle Eingriff wird erforderlich.

Kapitel 5

Zusammenfassung

Diese Arbeit stellte einen vollständigen Ansatz zur Erkundung unbekannter Innenräume mit dem mobilen autonomen Roboter „Robbie“ vor. Die als Basis für die Zielfindung und Wegeplanung verwendeten Occupancy Grids wurden erläutert, wie auch Yamauchis Idee der Grenzen-basierten Exploration. Die Distanztransformation und Zelinskys Pfadtransformation wurden als Lösungsansätze zur Pfadsuche erklärt. Eine darauf aufbauende Transformation - die Explorationstransformation - wurde entwickelt, welche das anfangs vorgestellte Explorationsproblem löst, indem Pfade zu unbekannten Bereichen gesucht werden, die gleichzeitig kurz und sicher zu befahren sind.

Die entwickelten Algorithmen wurden implementiert und in die Softwarearchitektur des Roboters „Robbie“ integriert. Tests in realen Umgebungen haben gezeigt, dass durch die vorgeschlagene Erkundungsstrategie Innenräume in den dem Roboter zugänglichen Bereichen vollständig kartografiert werden. Dabei werden leicht zugängliche Bereiche zunächst den schwerer zu erreichenden vorgezogen.

”Robbie” hat im April 2007 bei den RoboCup GermanOpen[1] als Rettungsroboter mit Hilfe der hier erarbeiteten Verfahren am Wettbewerb zur autonomen Erkundung eines simulierten Katastrophenszenarios mit Erfolg teilgenommen. In der Kategorie „Bester Autonomer Roboter“ gewann er den ersten Platz.

[1] siehe `http://www.robocup-german-open.de/`

Literaturverzeichnis

[Cho96] Howie Choset. Sensor based motion planning: The hierarchical generalized voronoi graph, 1996.

[Dij59] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271. Mathematisch Centrum, Amsterdam, The Netherlands, 1959.

[Elf89] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 6 1989.

[GBL02] Héctor H. Gonzáles-Baños and Jean-Claude Latombe. Navigation strategies for exploring indoor environments. *The International Journal of Robotics Research*, 2002.

[GK99] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proceedingsof the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, California, 1999.

[HNR68] Peter E. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum-cost paths. *IEEE Trans. on Systems Science and Cybernetics*, SSC-4(2):100–107, 7 1968.

[JB86] R. A. Jarvis and J. C. Byrne. Robot navigation: Touching, seeing an knowing. In *Proc. Australian Conf. on Artificial Intelligence*, Melbourne, Australia, 1986.

[KB91] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8(1-2):47–63, 1991.

[MCNT99] J. M. M. Montiel, Jose A. Castellanos, J. Neira, and J.D. Tardós. The spmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–952, 1999.

[RP68] A. Rosenfeld and J. L. Pfalz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[Thr98] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[Yam97] B. Yamauchi. A frontier-based approach for autonomous exploration. *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, page 146 ff., 1997.

[Zel88] Alexander Zelinsky. Robot navigation with learning. *Australian Computer Journal*, 20(2):85–93, 5 1988.

[Zel91] Alexander Zelinsky. *Environment Exploration and Path Planning Algorithms for a Mobile Robot using Sonar*. PhD thesis, Wollongong University, Australia, 1991.