Zielgerichtete Navigation und Kollisionsvermeidung eines mobilen Roboters

Vorgelegt von
Harald Stark
Kiel, im Juni 1998

Diplomarbeit
am Lehrstuhl Kognitive Systeme
des Informatikinstituts II
der Christian-Albrechts-Universität zu Kiel
Prof. Dr. G. Sommer

Betreuer: Dr. Jörg Bruske

Zusammenfassung

Ziel dieser Arbeit ist es, ein möglichst hardwareunabhängiges Softwarepaket zur Steuerung eines mobilen Roboters in einer Büroumgebung bereitzustellen. Die Navigationsaufgabe besteht darin, von der momentanen Roboterposition zu einem anderen (auswählbaren) Raum zu fahren und dort eine bestimmte Position einzunehmen. Die einzig unterstützende Maßnahme ist dabei ein maßstabsgetreuer Grundriß der Büroetage. Es sind also explizit keine baulichen Maßnahmen oder Vorgaben notwendig, um die Navigation zu erleichtern. Insbesondere sind also keine Magnetspuren, Induktionsschleifen, Infrarotemitter oder externe (nicht auf dem Roboter installierte) Kameras notwendig, ebenso keine architektonischen Beschränkungen wie rechtwinklig angeordnete Gänge. Es ist nicht notwendig, sich schnell bewegende Hindernisse, wie z.B. Personen, in der vorgegebenen Karte zu verzeichnen. Der Roboter erkennt und vermeidet selbständig solche Hindernisse. Die einzelnen Komponenten der Software sind unabhängige Prozesse. Dies fördert die Wartbarkeit und Erweiterbarkeit des Pakets, bzw. hat sich bei der Entwicklung positiv bei der Fehlervermeidung und -suche ausgewirkt. Zusätzlich wird die Beteiligung von mehreren Entwicklern am Projekt unterstützt. Aufgrund der arbeitsteiligen Vorgehensweise werden die Wahrnehmungs- und Lokalisationsanteile der Software in der Diplomarbeit von Timm Haase "Driftkompensation auf Basis von Besetztheitsgittern" beschrieben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Ziel der Arbeit ist es, einen mobilen autonomen Roboter in einer vorab kartierten Büroumgebung zielgerichtet von einem Raum zum anderen zu navigieren. Der Roboter ist autonom, d.h. die Umgebung wird nicht zur Erleichterung der Navigationsaufgabe verändert und alle zur Navigation notwendigen Komponenten sind in den Roboter integriert. Einzige Ausnahme bildet eine Funkverbindung zum fest installierten Rechnernetz des Instituts, die den Zweck hat, die verfügbare Rechenleistung zu erhöhen und die Planungsvorgänge zur Laufzeit des Systems zu dokumentieren. Prinzipiell ist die Verbindung aber entbehrlich.

Die Vorgaben für diese Arbeit sind die Verwendung eines Grundrisses der Büroetage und die Verwendung einer expliziten Selbstlokalisation auf dieser Karte, wobei die Selbstlokalisation in der Diplomarbeit von Timm Haase "Driftkompensation auf Basis von Besetztheitsgittern" behandelt wird.

1.1 Stand der Forschung

1.1.1 Das RHINO-Projekt

Anstoß zu dieser Arbeit war das RHINO-Projekt [1,2,11] von Sebastian Thrun, Dieter Fox und Wolfram Burgard. Rhino ist u.a. in der Lage, auf vorgegebenen Karten Zielpunkte anzusteuern. Die globale Planung findet ausgehend von der vorhandenen Karte mit einem Potentialfeldansatz statt, das über *value iteration*, ein Verfahren aus der dynamischen Programmierung, erzeugt wird. Dieser Potentialfeldplaner [11] setzt einen Zielpunkt in der Nähe des Roboters, der mit einem reaktiven Hindernisvermeider angesteuert wird. Die Wahrnehmung erfolgte in der Frühphase des Projektes mit Ultraschallsensoren. Zum Zeitpunkt der Erstellung dieser Arbeit wird aber schon seit längerer Zeit mit Laserentfernungsmessern gearbeitet. Mit diesen Sensoren kann, wegen der hohen Winkelauflösung von weniger als einem Grad zusammen mit einem maximalen Entfernungsfehler von 5 cm, die Selbstlokalisation und die Hindernisvermeidung um Größenordnungen zuverlässiger durchgeführt werden als mit Ultraschallsensoren.

Die hohe Winkelauflösung ermöglicht es zusätzlich, einzelne Messungen bei der Selbstlokalisation zu unterdrücken. Beispielsweise in einer dichten Menschenmenge ist RHINO noch immer in der Lage, sich zu orientieren, da genügend Meßstrahlen bis zu den Wänden vordringen und diejenigen, die auf dynamische Hindernisse treffen, erkannt und ausgefiltert werden können. Bei Ultraschallsensoren ist diese Vorgehensweise abwegig.

Zur lokalen Wegplanung werden, abhängig vom aktuellen Zustand des Roboters[1] in die direkt aus den Sensordaten erstellten bzw. ergänzten Karten, Kreisbögen gelegt, die den möglichen Pfaden entsprechen, die aus den aktuell möglichen Änderungen der Translations- und Rotationsgeschwindigkeit resultieren. Es wird diejenige Zustandsänderung zur Steuerung der Hardware gewählt, deren Pfad bezüglich der Zielverfolgung und Kollisionsvermeidung am günstigsten ist. Die Sensormessungen werden dabei als sog. Linefields gespeichert. Jede Hindernisdetektion ist als eine Linie/Stecke abgespeichert, die nicht im zu fahrenden Kreisbogen liegen darf.

Die von den Sensoren gewonnenen Besetztheitskarten können dazu benutzt werden, eine globale Karte aufzubauen. Bessere Resultate bei der Navigation erzielt man aber mit vorgegebenen Karten. Insbesondere schwierig zu detektierende Hindernisse können mit vorgefertigten Karten erheblich besser vermieden werden.

Die RHINO-Software hat sich leider als nicht portierbar bzw. von bisher unbeteiligten erweiterbar erwiesen. Deshalb wurde von einer Verwendung der vorhandenen Software als Grundlage der Arbeit Abstand genommen.

1.1.2 Mobot-IV

Der Mobot-IV [3] verfügt über Laserentfernungsmesser mit einem Sichtfeld von $360°$. Die globale Karte wird erlernt. Dabei werden lokale Sektorkarten, die aus einen Sensoreindruck der Umgebung in Sichtweite bestehen, zu einer globalen Karte zusammengesetzt, wobei gleichzeitig eine Topologische Repräsentation erzeugt wird. Eindeutig identifizierbare Sektorkarten, beispielsweise von Abzweigungen, werden als Landmarken zur Rekalibrierung der Selbstlokalisation verwendet. Dies ist wegen der hochauflösenden Laserentfernungsmesser möglich, die eine zuverlässige Detektion von Öffnungen und Wänden ermöglichen. Sektorkarten, die aus Messungen von Ultraschallsensoren erzeugt werden, sind nicht zuverlässig genug, um Landmarken wiederzuerkennen. Deshalb wurden lokale Sektorkarten in der vorliegenden Arbeit nicht verwendet.

[1] Position, Translations- und Rotationsgeschwindigkeit

1.1.3 HelpMate

Der HelpMate [15] ist ein Botenroboter, der in einem Krankenhaus autonom Transporte durchführt. Die Navigation wird mit dem VisBug-Algorithmus [21] durchgeführt. Die Selbstlokalisation erfolgt durch Odometrie in Verbindung mit einfach erkennbaren Referenzpositionen (z.B. Gangkreuzungen) und einer visuellen Auswertung der Position von Deckenleuchten.

1.1.4 Theoretische Ansätze

In [4] wird ein Ansatz vorgestellt, nur aufgrund einer qualitativen Beschreibung der Umgebung zu navigieren. Die Umgebung wird aufgrund von topologischen Gesichtspunkten in Gebiete unterteilt. Die Navigation basiert auf der Erkennung dieser Gebiete. Einfach detektierbare topologische Informationen werden anstelle von konkreten Koordinatenangaben benutzt. Dieser Ansatz setzt allerdings eine sehr strukturierte Umgebung voraus. Ohne unterscheidbare topologische Konfigurationen ist die Navigation nicht durchführbar. Aufgrund der ungenügenden Abbildungsgenauigkeit der Ultraschallsensoren und des damit einhergehenden Fehlens von unterscheidbaren Konfigurationen[2] ist der Ansatz nicht verwendbar. Ähnlich ist der Ansatz in [8] und [9]. Hier wird ausgehend von Erkenntnissen über neurologische Befunde ein neuronales Modell der Orientierung von Ratten in der Nähe von Landmarken vorgestellt. Da Landmarken mit Ultraschallsensoren nicht zuverlässig erkannt werden können, ist auch dieser Ansatz nicht geeignet. In [7] wird der Ansatz vorgestellt, anstelle von expliziten zielorientierten Potentialfeldern/Vektorfeldern direkt ein Vektorfeld zu erzeugen, das Sensormessungen auf Zielorientierungen an Verzweigungspunkten abbildet. In Umgebungen, die keine Navigationsentscheidungen verlangen, beispielsweise in Gängen, wird nur eine einfache Hindernisvermeidungsstrategie verfolgt. Die Abbildung der Sensoreindrücke an Verzeigungspunkten wird über ein Kohonennetz geleistet. Dieser Ansatz ist insofern ungeeignet, da er nur erlaubt, eine einzige Position nach einen Lernvorgang immer wieder anzusteuern. Die Aufgabe, sofort beliebige Punkte anzusteuern, ist damit nicht handhabbar. Als zusätzliche Komponente, die bei immer wieder anzusteuernden (Zwischen-)Zielen die Navigation durch Lernen im Betrieb unterstützt, ist der Ansatz vielversprechend, allerdings sprengt die Berücksichtigung dieses Ansatzes den Zeitrahmen der Arbeit. Hinzu kommt der Punkt der ungenügenden Umgebungswahrnehmung, die eine Planung nur aufgrund von Sensoreindrücken erschwert.

[2] Beispielsweise Türöffnungen in Gängen sind kaum detektierbar

1.1.5 Potentialfeldansätze

Potentialfelder sind eine Möglichkeit, durch Abstoßung von Hindernissen und Attraktion in Zielrichtung Robotersysteme zu steuern. Eine der frühesten Arbeiten, in der Potentialfelder zur Unterstützung einer Robotersteuerung vorgestellt wurden, ist die von O. Khatib [12]. Viele spätere Arbeiten behandeln dieses Thema, beispielsweise [13], [14], [16] und [17]. Hauptschwierigkeit bei der Verwendung von Potentialfeldern ist das Vermeiden von lokalen Minima und stabilen Sattelpunkten in der Potentialfeldfunktion. Bei Beschränkung auf Spezialfälle, beispielsweise auf kreisförmige Hindernisse ([14],[17]), wurden aber praktikable Verfahren zur Konstruktion von Potentialfunktionen ohne lokale Minima gefunden.

1.1.6 Graphbasierte Wegplanung

Die andere große Klasse von Steuerungsalgoithmen basiert auf einer Graphrepräsentation entweder der Umgebung oder des Konfigurationsraumes des Systems [20], [19], [18]. Zur Abbildung der Realität auf die graphbasierte Modellvorstellung werden Quadtrees oder auch polygonorientierte Geometrische Ansätze benutzt.

Graphbasierte Verfahren haben (häufig) den Vorteil, bei gegebener Umgebungsrepräsentation um Größenordnungen schneller zu sein als andere Wegplanungsverfahren. Nachteil ist die aufwendige Umwandlung von Sensorergebnissen in die graphbasierte Repräsentation, die in der Literatur häufig als schon gegeben vorausgesetzt wird.

1.2 Eigene Zielvorstellungen

Da zur Erstellung dieser Arbeit ein limitierter Zeitrahmen gegeben war, verzichten wir auf das Erlernen einer globalen Umgebungskarte und geben statt dessen einen Grundriß der Testumgebung vor, wie er in Abbildung 1.1 zu sehen ist.

Die Navigationsaufgabe, von der Startposition zur in der Karte vorgegebenen Zielposition zu fahren, wird in Abstraktionsebenen unterteilt. Die Abstraktionsebenen werden in eine Hierarchie von Modulen abgebildet. Der höchsten Ebene wird die Navigationsaufgabe gestellt. Beim Abstieg in der Hierarchie wird die Aufgabe sukzessive zerlegt und modifiziert bis schließlich auf der niedrigsten Ebene konkrete Fahrbefehle erzeugt werden. Es wird wegen der Modifikation also keine ausschließlich vertikale Kommandostruktur verwendet, sondern übergeordnete Module müssen auf die Modifikationen, die untergeordnete Module vornehmen, reagieren. Die Modifikationen stellen kein Fehlverhalten dar, sondern modellieren den horizontalen Aspekt der Entscheidungsfindung.

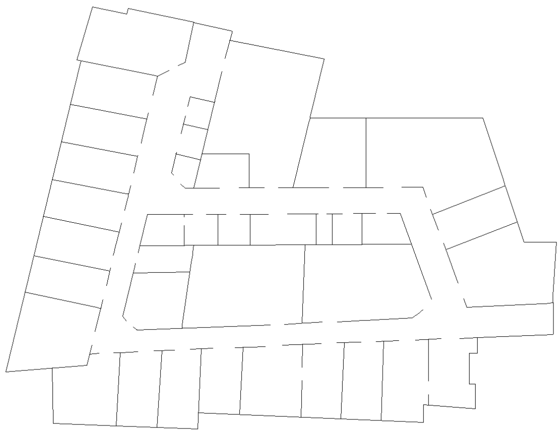

Abbildung 1.1: Die dritte Etage des Informatikinstituts II der CAU

Die Gliederung in eine Modulhierarchie hat zusätzlich den Zweck, die Fehleranfälligkeit bei der Entwicklung zu reduzieren und die Wartbarkeit der Software zu erhöhen. Ebenso wird die Abhängigkeit der Software von der verwendeten Hardware gering gehalten, weil die direkte Hardwareansteuerung auf wenige untergeordnete Module beschränkt bleibt. Damit ist auch der Portierungsaufwand beim Wechsel auf andere Robotersysteme gering.

Da der Wahrnehmungszweig Karten auf der Basis von Besetztheitsgittern erzeugt[3], wird der auf diesen Gittern unmittelbar anwendbare Potentialfeldansatz bevorzugt.

1.3 Aufbau der Arbeit

Die Arbeit ist in zwei Teile und einen Anhang gegliedert. Im ersten Teil - von Kapitel zwei bis sieben - stehen eher technische Aspekte der Arbeitsumgebung im Vordergrund. Im Kapitel zwei wird die Abstraktion der Hardware motiviert und beschrieben. In Kapitel drei wird der RWI-Roboter vorgestellt, auf dem die reale Implementation stattfand. In Kapitel vier wird kurz die Büroumgebung betrachtet und dann in Kapitel fünf der Simulator. In Kapitel sechs wird die Struktur der Navigationssoftware mit der Aufgabendefinition der Module beschrieben und in Kapitel sieben auf die Kommunikation zwischen den Modulen eingegangen.

Im zweiten Teil geht es um die Problemlösungsstrategien, die in den Navigationsmodulen verwendet werden. In Kapitel 8 wird der Navigator behandelt, der auf

[3]siehe die Diplomarbeit "Driftkompensation auf der Basis von Besetztheitsgittern" von Timm Haase

der globalen Ebene navigiert. In Kapitel 9 wird der Pilot betrachtet, der eine lokale Wegplanung auf der Basis von Besetztheitskarten vornimmt und im Kapitel 10 das Fahrermodul zur reaktiven Hindernisvermeidung bei gleichzeitiger Zielverfolgung. In Kapitel 11 ist ein Ansatz beschrieben, wie man mit einfachen Mitteln das auf dem Roboter vorhandene Kamerasystem dazu zu verwenden kann, die Besetztheitskarten zu verbessern.

Im Anhang werden im Detail Algorithmen erklärt, die in mehreren Modulen verwendet werden bzw. Grundlageninformationen gegeben, deren Behandlung bei der Modulbeschreibung zuviel Raum in Anspruch nehmen würde.

Der nur an den verwendeten Navigationsalgorithmen interessierte Leser sollte zumindest die Kapitel drei und sechs aus dem ersten Teil lesen, da die Struktur der Modulhierarchie und die Eigenschaften des realen Roboters notwendig zum Verständnis des zweiten Teils sind.

Teil I

Kapitel 2

Das abstrakte Robotermodell

In diesem Abschnitt wird die Modellvorstellung eines stark abstrahierten Roboters beschrieben, der als abstraktes Hardwareinterface für die darauf aufbauende Navigationssoftware dient. Die Abbildung des real verwendeten Roboters auf die Modellvorstellung wird durch das Hardwareinterface geleistet. Die Funktionalität dieses Moduls entspricht der eines Hardwaretreibers bei einem Rechnerbetriebssystem. Die Zielstellung bei der Konstruktion der Modellvorstellung ist dementsprechend die Trennung der konkreten Hardwareplattform von der Software, ohne die ansteuerbare Funktionalität zu sehr einzuschränken. Der praktische Nutzen dieser Vorgehensweise ist nicht nur die Vermeidung von unnötigem Aufwand bei einer Portierung der Software auf einen anderen Roboter, sondern auch die bessere Testbarkeit der Software. Durch die klare Trennung von Hardwareansteuerung und Entscheidungsbildung sind Fehler[1] leichter einzugrenzen und getrennt testbar.

2.1 Das Fahrwerk

Jeder mobile Roboter hat definitionsgemäß die Möglichkeit, sich fortzubewegen. Weit verbreitet sind Konstruktionen auf zwei Antriebsrädern bzw. Raupenketten, die in der Lage sind, auf der Stelle zu rotieren. Seltener sind Vehikel, die, wie beim PKW, schwenkbare Führungsräder besitzen und deshalb nur in Bewegung die Richtung wechseln können. In diesem Abschnitt wird zunächst auf die Spezifikation der Fahrwerksfähigkeiten eingegangen, anschließend der Raumbedarf im Zusammenhang mit der Kollisionsvermeidung betrachtet und abschließend die verwendeten abstrakten Fahranweisungen behandelt.

[1] Damit sind auch Fehler in der Hardwarespezifikation gemeint

2.1.1 Spezifikation der Fahrwerksfähigkeiten

Die unterschiedlichen Arten der Fortbewegung müssen zu einem allgemeinen Fortbewegungsmechanismus abstrahiert werden, damit Hardwareabhängigkeiten auf Programmteile beschränkt bleiben können, die unmittelbar für die Ansteuerung der mechanischen Komponenten notwendig sind. Es muß vermieden werden, daß in einer übergeordneten Softwarekomponente verdeckte Abhängigkeiten zur verwendeten Hardware entstehen können.

Eine Möglichkeit, dies zu erreichen, ist, jedem Fahrwerk eine Menge von Aktionen zuzuordnen, die mit der jeweiligen Hardware ausführbar sind. Beispielsweise könnte einem Roboter auf Raupenketten u.a. die Aktion 'auf der Stelle um 10 Grad drehen' zugeordnet werden. Mit einer größeren Anzahl solcher Aktionen ließen sich die Freiheitsgrade, die eine bestimmte Hardware bietet, näherungsweise beschreiben. Das Problem, dem Fahrwerk Anweisungen zu geben, wird trivial. Es ist nur notwendig, den Index der Aktion anzugeben, die ausgeführt werden soll.

Diese Vorgehensweise hat aber eine Reihe von Nachteilen. Zum Ersten schränkt die unvermeidliche Auswahl einer begrenzten Anzahl von Aktionen die Möglichkeiten einer übergeordneten Softwareschicht ein, die im folgenden 'Fahrer' genannt wird. Ein Fahrer könnte mit einem Aktionssatz brauchbare Ergebnisse liefern, während er mit einem anderen evtl. völlig versagt. Es entstehen versteckte Abhängigkeiten zwischen den unterschiedlichen Softwareschichten. Es ist das Ziel gerade diese zu vermieden. Zum Zweiten würde i.a. die Notwendigkeit bestehen, dem jeweiligen Fahrer mitzuteilen, was für Auswirkungen die einzelnen Aktionen haben. Diese Spezifikation aller Aktionen ist für die hier betrachteten Fahrwerke jedoch mindestens genauso aufwendig, wie die allgemeine Beschreibung der Fahrwerksfähigkeiten. Hinzu kommt das Problem des Fahrers, aus einer u.U. völlig ungeordneten Liste von Aktionen die gerade geeigneteste herauszufinden.

Benötigt wird also eine Beschreibung der Fahrwerksfähigkeiten, die möglichst knapp und systematisch ist, dem Fahrer keine vermeidbaren Abhängigkeiten an die jeweils verwendete Hardware aufzwingt und außerdem möglichst viele Alternativen offenläßt, diese Fähigkeiten zu nutzen.

Spezifikation der Translationsfähigkeit

Jedes in der Einleitung des Abschnitts genannten Vehikel ist in der Lage, sich vorwärts zu bewegen. Einige können auch rückwärts fahren. Sie tun dies mit einer bestimmten Maximalgeschwindigkeit, sowie die das Vehikel mit einer bestimmten Beschleunigung erreicht bzw. von der es mit einer bestimmten Bremsbeschleunigung wieder zum Stehen kommt. In der Spezifikation müssen also die Maximalgeschwindigkeit, die Beschleunigungswerte für Beschleunigen und Bremsen enthalten sein. Zusätzlich wird eine Minimalgeschwindigkeit angegeben, da manche

Fahrzeuge nicht in der Lage sind, beliebig langsam zu fahren.

2.1.2 Zusammenhang von Translation, Rotation und Kurvenradius

Ebenfalls ist jedes der betrachteten Fahrzeuge in der Lage, seine Fahrtrichtung zu ändern. Manche können auf der Stelle drehen, fast alle können es in Bewegung, wobei das Maß der möglichen Richtungsänderung i.a. von der Geschwindigkeit des Vehikels abhängt. Der Zusammenhang zwischen der Translationsgeschwindigkeit v, der Winkelgeschwindigkeit ω und dem Kurvenradius r ist gegeben durch:

$$r = \frac{v}{\omega}$$

Eine Richtungsänderung läßt sich also vollständig durch zwei der drei Größen v, ω und r, zusammen mit der Dauer des Vorgangs bzw. der abschließenden Orientierung, beschreiben.

Bei Fahrzeugen mit schwenkbaren Führungsrädern liegt ein Spezialfall des obigen Zusammenhangs vor. Bei konstantem Einschlag ist die Winkelgeschwindigkeit proportional zur Translationsgeschwindigkeit. Da der Kurvenradius vom Quotienten aus Translationsgeschwindigkeit und Rotationsgeschwindigkeit abhängt, werden unter diesen Voraussetzungen unabhängig von der Translationsgeschwindigkeit Kreisbögen mit demselben Kurvenradius abgefahren. Es genügt hier also, einen minimalen Kurvenradius anzugeben, um die Fähigkeiten zur Richtungsänderung zu spezifizieren.

Allgemein muß man sich für zwei der drei Größen v, ω und r entscheiden.

Ein subjektives Argument für die Verwendung von Winkelgeschwindigkeiten zusammen mit Translationsgeschwindigkeiten ist, daß Richtungsänderungen fast immer durch einen Winkel beschrieben werden, um den die Fahrtrichtung verändert werden soll. Beispielsweise bei der Hindernisvermeidung ist das Ziel eine Orientierung einzunehmen, auf der keine Kollisionen mit Hindernissen drohen. Die neue Orientierung ist das primäre Ziel, wobei die Geschwindigkeit gegeben und der Kurvenradius ein daraus resultierender Nebeneffekt ist.

Eine andere mögliche Sichtweise ist, vor den Roboter Kreisbögen mit unterschiedlichen Kurvenradien zu legen. Es wird dabei betrachtet, welcher Kreisbogen auf der Karte der Umgebung ohne Kollision mit einem Hindernis am ehesten in Richtung Ziel führt. Hier resultiert aus der Auswahl des Kreisbogens und der momentanen Geschwindigkeit die Winkelgeschwindigkeit des Fahrzeugs. Dieser Ansatz wurde beim RHINO-Projekt [1,11] verfolgt.

Im Programm wird das Vermögen, die Richtung zu ändern, durch fünf Konstanten spezifiziert. Die Konstanten geben die maximale Winkelgeschwindigkeit

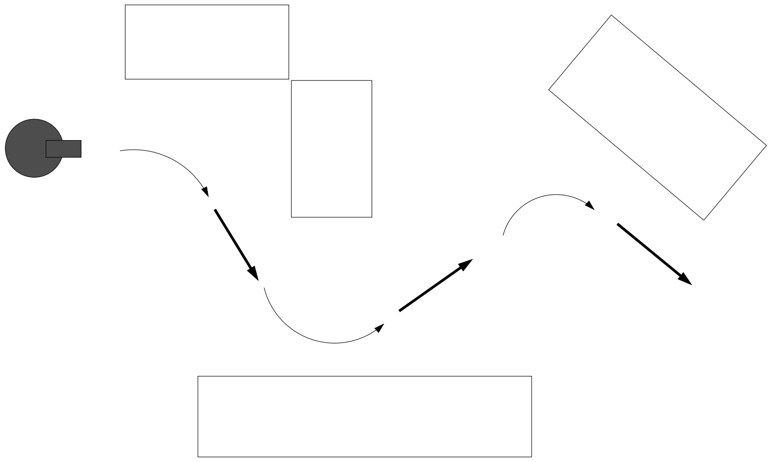

Abbildung 2.1: Kurvenradius als Nebeneffekt

im Stillstand, bei einer Geschwindigkeit von 100 mm/sec, 200 mm/sec, 400 mm/sec und bei 800 mm/sec an.

Auch Fahrzeuge mit schwenkbaren Führungsrädern lassen sich auf diese Weise charakterisieren, da die Translationsgeschwindigkeit mit der resultierenden Winkelgeschwindigkeit in einem linearen Zusammenhang steht. D.h. zwischen den angegebenen Stützstellen entstehen durch lineare Interpolation immer korrekte Werte.

2.1.3 Spezifikation des Platzverbrauches des Roboters

Wenn der Roboter sich fortbewegt, sollten Kollisionen vermieden werden. Man benötigt also einen Mechanismus, mit dem man das Auftreten von Kollisionen vorhersagen kann. Dieser Vorhersagemechanismus sollte dabei in praktikabler Weise alle mögliche Kollisionszustände verbieten, aber dennoch nicht unnötig einschränken.

Gewählt wurde eine Repräsentation, bei der der Raumbedarf des Roboters zunächst auf die Bodenfläche projiziert und anschließend diese Fläche noch einmal durch mehrere Parallelprojektionen auf skalare Werte abgebildet wird.

Der Vorteil dieser Spezifikationsmethode liegt in ihrer Kompaktheit. Mit nur sehr wenigen skalaren Werten wird durch den Schnitt der Projektionen ein Gebiet definiert, das die Grundfläche des Roboters überdeckt. Dabei ist es unerheblich, daß einige Flächen überdeckt werden, die evtl. nicht zum 'Grundriß' des Roboters gehören. Für die Vorhersage, ob eine Kollision auftritt, ist nur die konvexe Hülle des Grundrisses entscheidend. Die konvexe Hülle kann aber durch die Überdeckungsfläche der Projektionen approximiert werden. Sollte ein übergeordnetes Modul z.B. die Darstellung als Bitmap erfordern, so läßt sich diese leicht aus den Projektionswerten konstruieren. Die Rekonstruktion ist dabei in jeder beliebigen Orientierung,

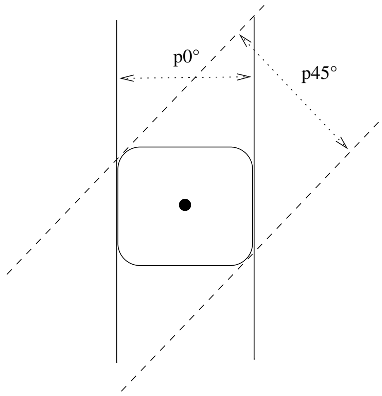

Abbildung 2.2: Projektion des Grundrisses auf Spezifikationskenngrößen

ohne den Qualitätsverlust wie bei der Rotation einer Bitmap, möglich.

Für einen Piloten oder Navigator, der die Aufgabe hat, den Roboter z.B. durch Lücken oder enge Gänge zu manövrieren, sind die obigen Kenngrößen direkt nutzbar. Sie geben unmittelbar an, wie breit eine Lücke sein muß, damit der Roboter in einer bestimmten Orientierung in ihr Platz findet und ob genug Raum vorhanden ist, um beispielsweise ohne Zurücksetzen in der Lücke zu drehen. Dies kann problematisch sein, wenn das Roboter erheblich länger als breit ist und damit schon bei geringer Änderung der relativen Orientierung zum Hindernis kollidiert oder in einem Gang verkantet.

2.1.4 Datenstruktur zur Spezifikation des Fahrwerks:

```
struct Base_specs
{
  // How much space is needed for operation ?
  int free_turn_radius;  // in mm relative to the center of rotation
                         // Radius that has to be obstacle free for
                         // a complete 360 deg turn

  int clear_path_0deg;  // [mm]
  int clear_path_10deg;  // [mm]
  int clear_path_20deg;  // [mm]
  int clear_path_30deg;  // [mm]

  int front_bumper_dist; // im mm relative to the center of rotation
  int rear_bumper_dist;

  // What are the abilities to change direction ?
  // Turn rates in [deg/sec]
  int max_turn_rate_when_stopped;  // For a car this would be zero   [deg/sec]
  int max_turn_rate_100;  //  Turn rate with 100 mm/sec   [deg/sec]
  int max_turn_rate_200;  //  Turn rate with 200 mm/sec   [deg/sec]
  int max_turn_rate_400;  //  Turn rate with 400 mm/sec   [deg/sec]
  int max_turn_rate_800;  //  Turn rate with 800 mm/sec   [deg/sec]

  // What are the abilities to change speed ?
  int max_speed;        // [mm/sec]
  int min_speed;
  int acceleration; // [mm/sec^2]
  int deacceleration; // [mm/sec^2]
}
```

2.1.5 Das Komandointerface des Fahrwerks

Der virtuelle Roboter kennt nur einen einzigen Befehl CURVE. Ein Curve-Befehl teilt dem Roboter mit, daß er eine mit einer bestimmten Translationsgeschwindigkeit v und einer der Translation überlagerten Rotationsgeschwindigkeit ω maximal eine Strecke d fahren soll, wobei die Rotation bei einer bestimmten absoluten Orientierung Θ gestoppt wird.

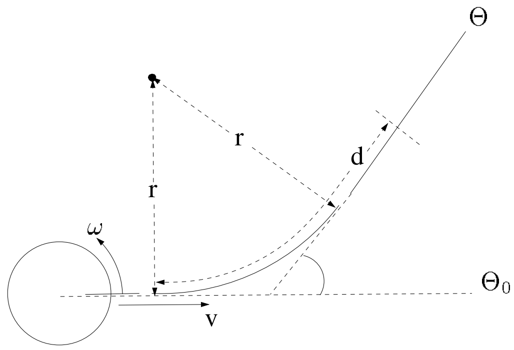

Abbildung 2.3: Der Curve-Befehl

Die Beschränkung auf eine begrenzte Strecke ist eine Sicherheitsmaßnahme, die bei Ausfall des Fahrers den Roboter stoppt, wenn keine Befehle mehr gegeben werden. Im normalen Betrieb sollte schon **vor** der vollständigen Ausführung eines Befehls ein Folgebefehl eintreffen, der die Ausführung des alten Befehls sofort abbricht und dann unmittelbar selbst abgearbeitet wird bis der nächste Befehl eintrifft.

Dieser eine Befehlstyp reicht aus, um die Bewegungsmöglichkeiten des Roboters voll auszuschöpfen. Wenn der Roboter beispielsweise auf der Stelle rotieren soll, so wird die Translationsgeschwindigkeit auf Null gesetzt und eine Rotationsgeschwindigkeit mit Zielwinkel angegeben. Der Fahrer muß sich allerdings anhand der Fahrwerksspezifikation vergewissern, daß eine Rotation im Stillstand möglich ist. Geradeausfahren ist ebenso einfach. Die Translationsgeschwindigkeit wird auf den gewünschten Wert gesetzt, ebenso die Entfernung. Die Rotationsgeschwindigkeit wird auf Null gesetzt und der Roboter fährt die gewünschte Strecke geradeaus. Zwischen diesen beiden Extremen kann jeder beliebige Kreisbogen abgefahren werden. Lediglich Bögen mit kontinuierlich wechselndem Kurvenradius sind schwer zu realisieren, da hierzu ständig neue Curve-Befehle gegeben werden müssen. Da sich bei der Implementation die Latenzzeit, nach der ein Fahrbefehl vom Fahrer zur Hardware übermittelt wird, als Problem erwiesen hat, ist es möglich, einen modifizierten Befehl zu geben, bei dem die Wahl der Rotationsgeschwindigkeit demjenigen Modul überlassen wird, das die Hardware direkt ansteuert. Es wird also nur die Translationsgeschwindigkeit, die maximale Distanz und die Zielorientierung übermittelt. Diese Vorgehensweise hat sich bei großen Latenzzeiten in der Kommunikation als erheblich robuster erwiesen als die explizite Angabe von Winkelgeschwindigkeiten. Die sinnvolle Winkelgeschwindigkeit zur schnellen und genauen Ansteuerung der Zielorientierung hängt empfindlich von dem Zustand[2] des Roboters bei Erhalt des Befehls ab und ist zusätzlich abhängig von spezifischen Eigenschaften[3] der Hardware, die nicht in der Spezifikation enthalten sind. Beispielsweise das Fuzzy-Modul, das im zweiten Teil der Arbeit beschrieben wird, interpretiert die Fahrbefehle auf diese Weise.

2.2 Die Sensoren

Der Roboter benötigt Sensoren, um seine Umgebung wahrzunehmen. Standardmäßig werden im Programmpaket nur Ultraschallsensoren berücksichtigt. Unter einem Ultraschallsensor wird in dieser Arbeit ein Gerät verstanden, das einen gerichteten Ultraschallimpuls aussendet und aus der gemessenen Zeit bis zur Detektion des Echos die Entfernung zum Hindernis ermittelt.

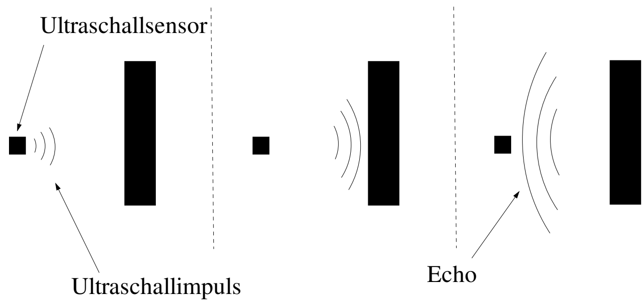

Abbildung 2.4: Ultraschallentfernungsmessung

Das abstrakte Hardwareinterface sendet beim Aufbau der Verbindung zum Fahrer eine grobe Spezifikation der verwendeten Ultraschallsensoren. Berücksichtigt werden die Positionen, die Orientierungen, der Öffnungswinkel und die maximale Reichweite der Sensoren. Die konkrete Datenstruktur ist wie folgt definiert:

```
struct Sonar_specs
{
  int no;  // Number of sonar sensors
  int max_range; // [mm]
  int cone_angle; // [deg]
   // Positions are relative to the center of rotation
```

[2]Restwinkel und momentane Winkelgeschwindigkeit

[3]Bei bestimmten Komandofolgen kann die Rotationsmechanik des Roboter beispielsweise in Oszillation geraten

```
  int pos_x[ NO_OF_SONARS ];  // [mm]
  int pos_y[ NO_OF_SONARS ];  // [mm]
   // Positive angle relative to the front vector of the robot
  int bearing[ NO_OF_SONARS ];  // [deg]
};
```

In der Realität ist es nicht möglich, alle Ultraschallsensoren gleichzeitig die Entfernung zum jeweils nächsten Hindernis messen zu lassen. Das Echo eines Sensors könnte von einem anderen fälschlich als das eigene Echo aufgefaßt werden, was die Messungen wertlos macht. Deshalb ermitteln benachbarte Sensoren zeitlich versetzt die Entfernung. Im Programm werden die Messungen trotzdem so interpretiert, als ob sie gleichzeitig stattgefunden hätten. Für einen Roboter in einer Büroumgebung ist es sinnvoll, sich nur mit Geschwindigkeiten deutlich unter einem Meter pro Sekunde fortbewegen. Bei diesen Geschwindigkeiten ist die Vereinfachung ohne Qualitätsverlust bei der Kartenbildung möglich, da jeder Sensor mehrmals pro Sekunde feuert und deshalb die Wegstrecke zwischen zwei Messungen unter der Auflösungsschwelle der daraus erzeugten Karten liegt. Die gesammelten Daten 'einer' Messung werden zu einem Sonar-Report gebündelt und vom Hardwareinterface an den Fahrer gesendet. Ein Sonar-Report zur Übermittlung der aktuellen Ultraschallentfernungswerte ist folgendermaßen definiert:

```
struct Sonar_data
{
  int no;     // serial no of the report
  int time;   // ms since start
  int distance[ NO_OF_SONARS ];  // [mm]
};
```

Zusätzlich zu den Ultraschallentfernungsmessern sind beim abstrakten Robotermodell zunächst keine weiteren Sensoren vorgesehen. Da die Entfernungswerte in einem dem Fahrer übergeordneten Modul zu einer Besetztheitskarte integriert werden, ist es aber dort möglich, die Ergebnisse von anderen Sensoren z.B. die Tiefenkarte eines Laser-Radars in den Wahrnehmungsprozeß einzubinden.

Kapitel 3

Der verwendete RWI-Roboter

In diesem Kapitel wird die Hardware des RWI-Roboters beschrieben, auf dem die Navigationssoftware getestet wurde.

3.1 Das Fahrwerk und Karosserie

Der RWI-Roboter ist zylindrisch aufgebaut mit einem Durchmesser von ca. 60 cm und einer Höhe (ohne Aufbauten) von ca. 120 cm.

Abbildung 3.1: Der RWI-Roboter

Das Fahrwerk erlaubt es dem Roboter, voneinander unabhängige Tranlations- und eine Rotationsbewegungen auszuführen. Der Roboter ist also in der Lage, auf der Stelle zu rotieren, beliebige Kurven zu fahren und selbstverständlich auch geraden Linien zu folgen. Bei der Geradeausfahrt erreicht der Roboter dabei maximal eine Geschwindigkeit von einem Meter pro Sekunde. Die Bremsleistung ist so groß, daß der Roboter aus praktisch jedem Betriebszustand heraus unmittelbar zum Stillstand kommen kann. Eine Betrachtung der Bremsbeschleunigung erübrigt sich also.

3.2 Die Ultraschallsensoren

Der RWI-Roboter hat 24 Ultraschallsensoren, die regelmäßig in $15°$ Abständen rund um den Roboter in einer Höhe von ca. 80 cm angebracht sind. Der Detektionskegel hat - laut Hersteller - ebenfalls einen Öffnungswinkel von $15°$, und innerhalb dieses Kegels kann ein Objekt in einer Entfernung bis zu 5 Metern erkannt werden.

Das Diagramm 3.2 zeigt die vom Hersteller angegebene Richtcharakteristik der Ultraschallsensoren.

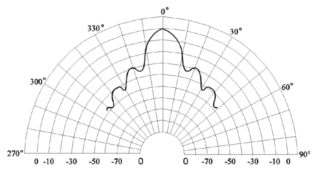

Abbildung 3.2: Richtcharakteristik eines Ultraschallsensors

Die Ultraschallsensoren des RWI-Roboters haben eine Besonderheit. Es ist weitestgehend ausgeschlossen, daß sie eine zu kurze Entfernung zu einem Hindernis anzeigen. Bei herkömmlichen Ultraschallsensoren sind Echos von benachbarten Sensoren häufig der Grund für Fehlmessungen. Im Bild 3.3 ist eine mögliche Fehlersituation dargestellt.

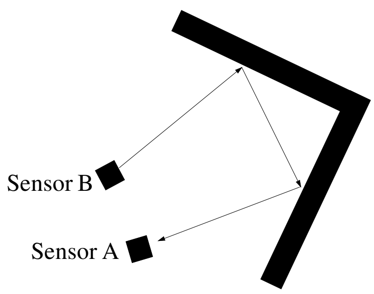

Abbildung 3.3: Falsches Echo

Das Echo des Ultraschallimpulses vom Sensor B, der früher feuert als der Sensor A, wird durch eine Folge von Reflexionen auf den Sensor A geleitet. Da der Sensor A erst dann feuert, wenn der Impuls von B schon unterwegs ist, kommt der B-Impuls früher beim Sensor A an als der eigene A-Impuls. In Bild 3.4 ist die von Sensor A gemessene Intensität der beiden Impulse dargestellt. Primitive Ultraschallsensoren sehen grundsätzlich nur den allerersten Impuls, der eine bestimmte Intensitätsschwelle überschreitet, als Echo an.

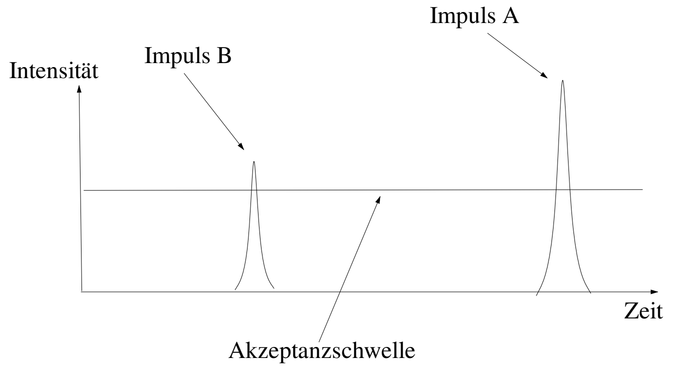

Abbildung 3.4: Intensität der von Sensor A gemessenen Impulse mit konstanter Akzeptanzschwelle

Die Echoauswertung beim RWI-Roboter (schematisch in Bild 3.5 dargestellt) nutzt den Effekt der Dämpfung des Ultraschallimpulses während seiner Laufzeit. Ein Echo aus nächster Nähe ist viel stärker als ein entferntes Echo. Zur Unterdrückung von zu kurzen Messungen wird die Akzeptanzschwelle am Anfang der Messung sehr hoch angesetzt und fällt langsam ab.

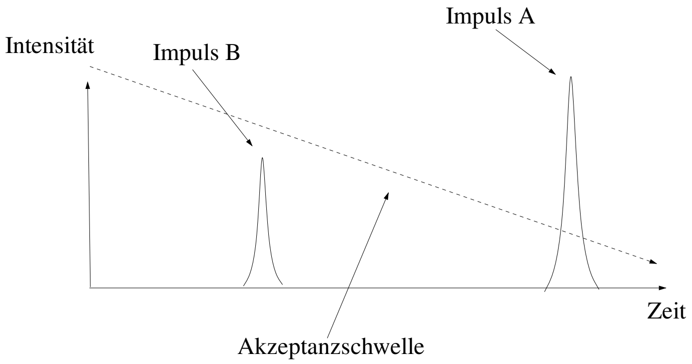

Abbildung 3.5: Intensität der von Sensor A gemessenen Impulse mit abfallender Akzeptanzschwelle

Echos, die durch Mehrfachreflexion entstehen, sind in der Regel zu stark abgedämpft, als daß sie anstelle des korrekten Echos die Entfernungsmessung auslösen könnten.

Der Schwachpunkt des Verfahrens ist in Bild 3.6 dargestellt. Es gibt Hindernisse, die den Ultraschallimpuls bei der Reflexion größtenteils verschlucken. Ein solches Hindernis kann nicht mehr detektiert werden, obwohl er ein meßbares Echo zurückwirft!

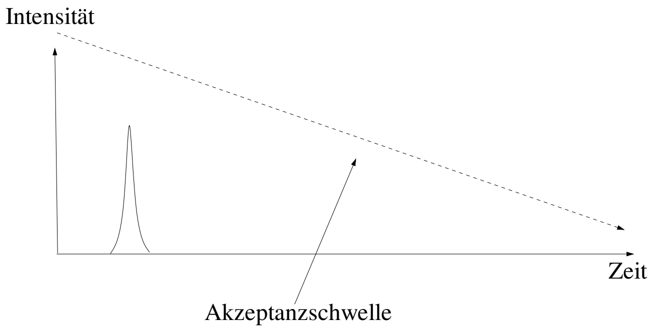

Abbildung 3.6: Intensität der von Sensor A gemessenen Impulse mit abfallender Akzeptanzschwelle

Eine Anpassung der Schwellwertkurve beinhaltet allerdings die Gefahr, wieder fälschlich Echos anderer Sensoren zu detektieren. Diese Schwäche kann also i.a. nicht behoben werden.

Eine weitere grundsätzlich nicht behebbare Quelle von Fehlmessungen ist der Effekt der Totalreflexion des Ultraschallimpulses ohne daß der reflektierte Impuls wieder auf den Sensor trifft. Dieser Effekt tritt besonders häufig bei glatten Wänden oder Türen auf. Das Bild 3.7 zeigt eine typische Ultraschallkarte von einer Wand, an der an einigen Stellen Totalreflexion auftritt.

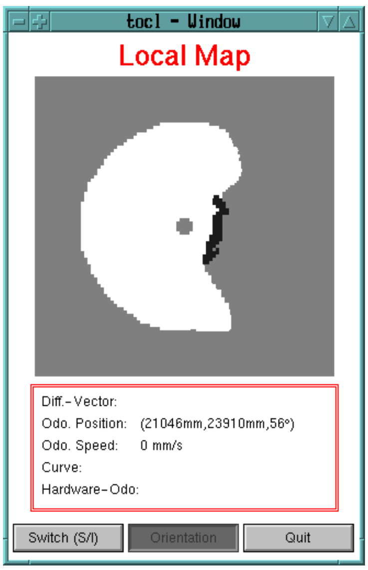

Abbildung 3.7: Typische Ultraschallkarte einer Wand

Die Bereiche mit steilem Einfallswinkel werden noch detektiert, während diejenigen mit flachem Einfallswinkel nicht mehr wahrgenommen werden.

Noch schlimmer ist die Situation bei glatten Türen. Hier muß der Ultraschallimpuls fast exakt senkrecht auftreffen, um detektiert zu werden. Eine Abweichung von wenigen Grad verhindert jegliche Detektion.

3.3 Nicht spezifizierte Eigenschaften des Roboters

Der Roboter hat eine Reihe von Infrarotsensoren und Bumpern, die nicht in der Spezifikation des Roboters berücksichtigt werden. Die Infrarotsensoren haben allerdings nur eine Reichweite von wenigen Zentimetern und können deshalb nicht wesentlich zur Kartenbildung beitragen. Ebenso wurden der Roboterarm und die Kameras nicht berücksichtigt.

Kapitel 4

Die Umwelt des Roboters

Um den Roboter sinnvoll steuern bzw. regeln zu können, ist es notwendig, eine Modellvorstellung der Umgebung zu haben, in der der Roboter agieren soll. Wie schon in der Einleitung erwähnt, soll der Roboter in einer Büroumgebung navigieren. Im folgenden wird kurz die Büroumgebung beschrieben, in der die Navigationssoftware getestet wurde. Anschließend wird erörtert, wie diese Umgebung im Programm repräsentiert werden kann.

4.1 Beschreibung der Testumgebung

Die Testumgebung ist der östliche Flügel der dritten Etage des Informatikinstituts II der CAU. Wie im Grundriß (Abb. 4.1) zu erkennen ist, sind entlang einem ringförmigen Gang viele kleine Büros und einige größere Laboratorien und Seminarräume verteilt.

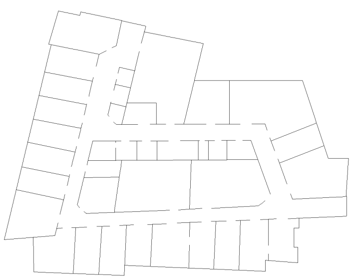

Abbildung 4.1: Die dritte Etage des Informatikinstituts II der CAU

Die Gänge haben eine Breite von ca. 1.70 m und sind (meistens) frei von Hindernissen. Die Türöffnungen zu den Räumen sind etwa 1.00 m breit. Die Türpfosten und besonders die Türen sind extrem glatt, weshalb sie für Ultraschallsensoren fast unsichtbar sind.

Die Gänge sind nicht rechtwinklig angeordnet! Allerdings dominieren wenige Grundorientierungen.

Abbildung 4.2: Eine Türöffnung aus der Sicht des Roboters

Die Wände sind mit Rauhfasertapete bedeckt. Sie können wegen der aufgerauhten Oberfläche gut mit Ultraschallsensoren detektiert werden.

4.2 Darstellung im Programm

Da der Roboter nur ebenen hindernisfreien Boden zur Fortbewegung nutzen kann, genügt eine zweidimensionale Repräsentation der Büroetage. In diesem Grundriß muß zunächst zwischen befahrbaren und unbefahrbaren Gebieten unterschieden werden. Diese Informationen sind die Grundlage der Navigation. Zusätzlich ist es sinnvoll, wahrnehmbare Eigenschaften der Umgebung in der Karte zu repräsentieren. Dies dient der Selbstlokalisation des Roboters. Da in dieser Arbeit primär mit Ultraschallsensoren gearbeitet wird, gibt es nur geringfügige Diskrepanzen zwischen den unbefahrbaren Gebieten und denjenigen Gebieten, die von den Ultraschallsensoren (prinzipiell) als 'Besetzt' registriert werden. Es genügt also eine binäre Unterscheidung zwischen befahrbaren und unbefahrbaren Gebieten.

Für derartige Karten gibt es zwei kanonische Darstellungsformen. Die erste Möglichkeit ist, den Grundriß vektoriell zu repräsentieren. Da die Büroetage hauptsächlich durch gerade Wände strukturiert ist, liegt es nahe, die Karte als (leeres) Gebiet zu repräsentieren, in dem alle Hindernisse durch Polygonzüge dargestellt werden. Die zweite einfachere Möglichkeit besteht darin, die Büroetage als Bitmap zu speichern, in dem gesetzte Bits einem Hindernis entsprechen. Die Bitmapdarstellung hat den Nachteil, daß sie bei steigender Kantenlänge des kartierten Gebiets einen quadratisch ansteigenden Speicherplatzverbrauch aufweist. Der Speicherplatzbedarf der vektoriellen Darstellung steigt nur linear mit der Anzahl der vorhandenen Hindernisstrecken an. Bei gleichmäßiger Hindernisdichte steigt diese Anzahl aber ebenfalls quadratisch mit der Gebietskantenlänge an. Die Kollisionserkennung wird bei naiver Implementation durch eine Liste von Strecken sogar erheblich aufwendiger. Durch die Verwendung von intelligenteren Datenstrukturen, wie z.B. zweidimensionale Suchbäume, kann der Aufwand zwar reduziert werden, allerdings wird nie die Geschwindigkeit und Einfachheit der Bitmapdarstellung erreicht.

Da die für die Anwendung vorgesehenen Gebiete relativ klein sind[1], wurde die für die Programmierung einfachere Bitmapdarstellung gewählt.

Die im Programm konkret verwendeten Karten bestehen aus zweidimensionalen char-Arrays mit ca. 400 x 300 Elementen. Eine Karte der Umgebung verbraucht also ca. 120 Kilobytes Speicherplatz, was bei heute üblichen Hauptspeichergrößen über 32 MB kein Problem darstellt. Eine echte Binärdarstellung wäre noch kompakter. Die verwendete Repräsentation ermöglicht allerdings Erweiterungen, um beispielsweise Zusatzinformationen für eine verbesserte Selbstlokalisation zu speichern.

[1]Die Testumgebung mißt ca. 30m x 40 m

Kapitel 5

Die Simulation des Roboters

In diesem Abschnitt wird beschrieben, warum und wie der Roboter simuliert wird. Die Grenzen der implementierten Simulationsumgebung werden dargestellt und die vorhandenen Einschränkungen, in Bezug auf das Ziel, einen praktikablen Simulator mit der vorhandenen Rechnerausstattung zu implementieren, gerechtfertigt.

5.1 Ziele der Simulation

Die Bereitstellung der Simulationsumgebung dient folgenden Zielen:

- Es wird Schaden an teurer Ausrüstung (Roboter/Labor) verhindert
- Eine Testumgebung wird mehreren Entwicklern gleichzeitig zugänglich gemacht
- Der Vergleich zwischen Simulation und Realität zeigt, ob das reale System schon vollständig verstanden wurde
- Es sind Testläufe in Umgebungen möglich, in die der reale Roboter aus organisatorischen oder Kostengründen nicht gebracht werden kann
- Die Softwareentwicklung wird unabhängig von fehleranfälliger Hardware

Völlig ungetestete Software zur Steuerung von teurer Hardware zu benutzen, ist fahrlässig und allein schon eine mehr als ausreichende Rechtfertigung für eine Simulationsumgebung.

Hinzu kommt die Schwierigkeit, daß es fast unmöglich ist, das gewünschte 'Verhalten' eines Navigations- bzw. Steuerungsalgorithmus zu spezifizieren. Damit ist auch der Weg der Softwareverifikation per Beweis nicht gangbar. Da die teure Ressource Roboter nur von jeweils einem Entwickler benutzt werden kann, wird die Entwicklung durch die von einem Simulator erweiterten Testmöglichkeiten beschleunigt und/oder preisgünstiger. Der Vergleich zwischen Realität und Simulation kann Hinweise darauf geben, was zu Recht vernachlässigt werden kann und was überraschenderweise einen erheblichen Einfluß auf den Erfolg der Steuerung hat. Um zu überprüfen, ob ein Verfahren auch in fremden Umgebungen funktioniert, ist eine Simulation wesentlich praktikabler als der aufwendige Transport des Roboters in ein anderes Gebäude. In der Regel ist ein solcher Test mangels geeigneter Gebäude ohnehin nicht durchführbar. Die reproduzierbaren Umgebungsbedingungen innerhalb der Simulation erleichtern die Entwicklung erheblich. Im realen Test auftretende Unzulänglichkeiten der Hardware, wie beispielsweise eine nachlassende Zuverlässigkeit der Ultraschallsensoren bei niedriger Batteriespannung, können so unabhängig von reinen Softwarefehlern behandelt werden.

5.2 Die Simulation im Detail

Die Simulation hat das Ziel, das Fahrverhalten des Roboters in der realen Testumgebung möglichst genau nachbilden. Als Eingaben bekommt die Simulation dazu eine Bitmapkarte der zu befahrenden Umgebung, den initialen Standort des Roboters mit Orientierung und die Spezifikationen zum Fahrwerk und den Ultraschallsensoren, wie sie in den Abschnitten 2.2 und 3.2 beschrieben wurden. Aus diesen Informationen und den Fahrbefehlen der Steuerungssoftware generiert der Simulator einen Positionsupdate des Roboters und gibt dazu passende simulierte Entfernungsmessungen aus. Diese Informationen werden dann graphisch genauso auf dem Bildschirm dargestellt wie bei einem realen Testlauf.

In der Modulhierarchie der Navigationssoftware ist der Simulator Teil des Hardwareinterfaces. Alle dem Hardwareinterface übergeordneten Module sind also nicht in der Lage, einen Simulationslauf von einem realen Testlauf zu unterscheiden.

5.2.1 Zustandsgrößen des Roboters

Zur Simulation des Roboters muß der Zustand, in dem der Roboters sich befindet, spezifiziert werden. Der Zustand des Roboters wird beschrieben durch die Position (x, y) und die Orientierung Θ im Koordinatensystem der Umgebungskarte. Weiterhin ist in der Zustandsbeschreibung die Translationsgeschwindigkeit v in Richtung Θ und die Rotationsgeschwindigkeit ω enthalten.

5.2.2 Simulation der Roboterbewegung

Die Belegung der obigen Zustandsbeschreibung muß bei fortschreitender Simulationszeit verändert werden, wenn der simulierte Roboter sich fortbewegt. Das Hard-

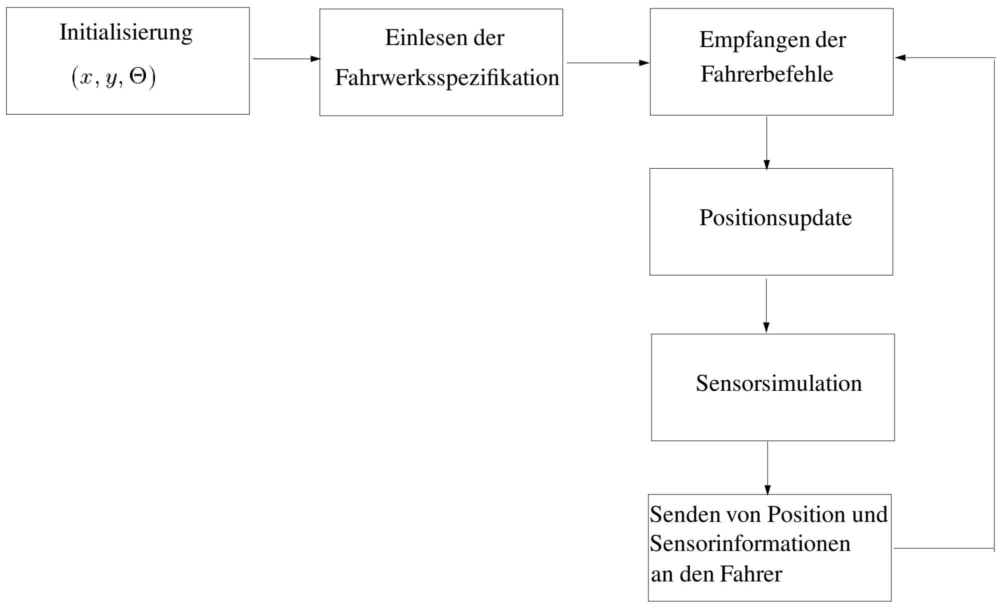

Abbildung 5.1: Ablaufschema der Simulation

wareinterface erhält von der Navigationssoftware als Eingaben Curve-Befehle. Das Hardwareinterface generiert aus dem Curve-Befehl eine Translationsgeschwindigkeit und eine Rotationsgeschwindigkeit, die bei einem realen Versuch den Roboter und bei einem Simulationslauf den simulierten Roboter steuern.

Physik der Roboterbewegung

Die reale Roboterbewegung resultiert aus der zeitlich variierten Kraft, mit der die Motoren des Roboters diesen vorwärts bewegen oder ihn in Rotation versetzen. In einem physikalischen Modell der Bewegung können wegen der konstanten Masse des Roboters statt dieser Kräfte direkt Beschleunigungen betrachtet werden. Zur Berechnung der Bewegung des Roboters mit diesen Beschleunigungen sind die folgenden physikalischen Zusammenhänge von Interesse.

$$v_{t_2} = \int_{t_1}^{t_2} a_t dt + v_{t_1}$$

$$\omega_{t_2} = \int_{t_1}^{t_2} \aleph_t dt + \omega_{t_1}$$

$$\Theta_{t_2} = \int_{t_1}^{t_2} \omega_t dt + \Theta_{t_1}$$

$$x_{t_2} = \int_{t_1}^{t_2} v_t \cdot cos(\Theta_t) dt + x_{t_1}$$

$$y_{t_2} = \int_{t_1}^{t_2} v_t \cdot sin(\Theta_t)dt + y_{t_1}$$

Dabei ist v_t die Translationsgeschwindigkeit zum Zeitpunkt t, a_t die Translationsbeschleunigung zum Zeitpunkt t, Θ_t die Orientierung zum Zeitpunkt t, ω_t die Winkelgeschwindigkeit zum Zeitpunkt t, $\aleph_t$ die Winkelbeschleunigung und (x_t, y_t) die Position des Roboters zum Zeitpunkt t.

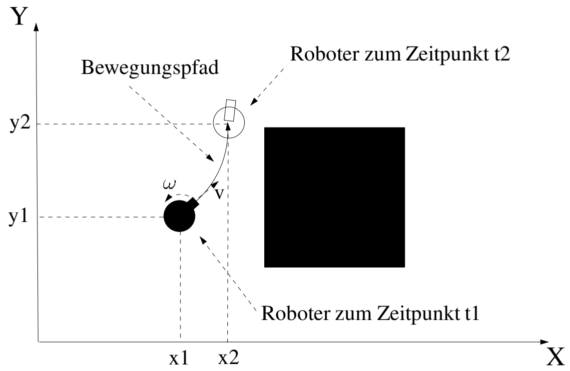

Abbildung 5.2: Positionsupdate des simulierten Roboters

Zeitliche Diskretisierung der Positionsberechnung

Im implementierten Simulator wird eine diskrete und geringfügig vereinfachte Version des obigen Modells verwendet. Der Zeitraum von $t1$ bis $t2$ wird in diskrete Zeitschritte unterteilt und zwischen den Zeitschritten jeweils eine konstante Belegung der Parameter angenommen. Weil der reale Roboter sehr große Winkelbeschleunigungen erreicht, wird diese vernachlässigt und direkt die Rotationsgeschwindigkeit verwendet.

$$v_{t_i} = \sum_{j=t_0}^{t_{i-1}} a_{t_j} \cdot (t_{j+1} - t_j) + v_{t_0}$$

$$\Theta_{t_i} = \sum_{j=t_0}^{t_{i-1}} \omega_{t_j} \cdot (t_{j+1} - t_j) + \Theta_{t_0}$$

$$x_{t_n} = \sum_{i=t_0}^{t_{n-1}} v_{t_i} \cdot cos(\Theta_{t_i}) \cdot (t_{i+1} - t_i) + x_{t_0}$$

$$y_{t_n} = \sum_{i=t_0}^{t_{n-1}} v_{t_i} \cdot sin(\Theta_{t_i}) \cdot (t_{i+1} - t_i) + y_{t_0}$$

Bei einer Grenzbetrachtung mit unendlich vielen Zeitschritten erhält man wieder die kontinuierliche Darstellung.

5.2.3 Konstante oder variable Zeitschritte ?

Bisher wurde die Frage offengelassen, welche Abstände die in der Positionsberechnung betrachteten diskreten Zeitpunkte voneinander haben. Zur Auswahl stehen grundsätzlich zwei Alternativen.

Konstante Zeitschritte

Die erste Möglichkeit ist, die Zeitspanne zwischen zwei betrachteten diskreten Zeitpunkten konstant zu halten. Dieses Vorgehen vereinfacht die Berechnung des jeweiligen Folgezustands des Roboters, da die Zeitspanne nicht mehr explizit in der Berechnung berücksichtigt werden muß. Die Geschwindigkeits und Beschleunigungsparameter für Translation und Rotation werden durch einen Wechsel des Bezugssystems so umskaliert, daß die Zeitspanne $\Delta t_i = t_{i+1} - t_i$ mit 1.0 gleichgesetzt werden kann. Die Berechnung vereinfacht sich zu:

$$v'_{t_n} = \sum_{i=t_0}^{t_{n-1}} a'_{t_i} + v'_{t_0}$$

$$\Theta_{t_n} = \sum_{i=t_0}^{t_{n-1}} \alpha'_{t_i} + \Theta_{t_0}$$

$$x_{t_n} = \sum_{i=t_0}^{t_{n-1}} v'_{t_i} \cdot cos(\Theta_{t_i}) + x_{t_0}$$

$$y_{t_n} = \sum_{i=t_0}^{t_{n-1}} v'_{t_i} \cdot sin(\Theta_{t_i}) + y_{t_0}$$

wobei

$$a'_{t_i} = \frac{a_{t_i}}{\Delta t_i}, \alpha'_{t_i} = \frac{\alpha_{t_i}}{\Delta t_i}, v'_{t_i} = \frac{v_{t_i}}{\Delta t_i}$$

Variable Zeitschritte

Die zweite Möglichkeit ist beliebige[1] Zeitspannen $\Delta t_i = t_{i+1} - t_i$ zwischen den betrachteten diskreten Zeitpunkten zuzulassen. Die Berechnung ist im Vergleich zum Vorgehen mit konstanten Zeitschritten grundsätzlich aufwendiger, da in jeder der Bewegungsgleichungen der Faktor Δt_i auftritt. Der Zusatzaufwand ist aber im konkreten Fall der Berechnung des neuen Roboterzustands so gering, daß er im Vergleich zum Gesamtaufwand zur Ausführung des Simulators nicht mehr wahrnehmbar ist.

[1]Die Zeitspannen sollten trotzdem kurz gewählt werden

Der große Vorteil von variablen Zeitschritten ist, daß die Zeit, die das Navigationsprogramm benötigt, neue Befehle zu generieren, und auch die Latenz bei der Kommunikation zwischen den Modulen realistisch in die Simulation eingeht. Beispielsweise bei Steuerungsalgorithmen, die in Abhängigkeit von der Sensorinformation unterschiedlich lange benötigen, um neue Fahrbefehle zu erzeugen, kann schon in der Simulation festgestellt werden, ob das Zeitverhalten des Programms die Navigation und Hindernisvermeidung negativ beeinflußt. Ebenso kann der Effekt, daß das Programm sich je nach Rechengeschwindigkeit der Computerhardware unterschiedlich verhält, schon in der Simulation realistisch beurteilt werden.

5.2.4 Der implementierte Zustandsupdate

Implementiert wurde der Zustandsupdate mit variablen Zeitschritten, da dieser bei nur unwesentlich höherem Berechnungsaufwand erheblich realitätsnäher arbeitet, als die Variante mit konstanten Zeitschritten.

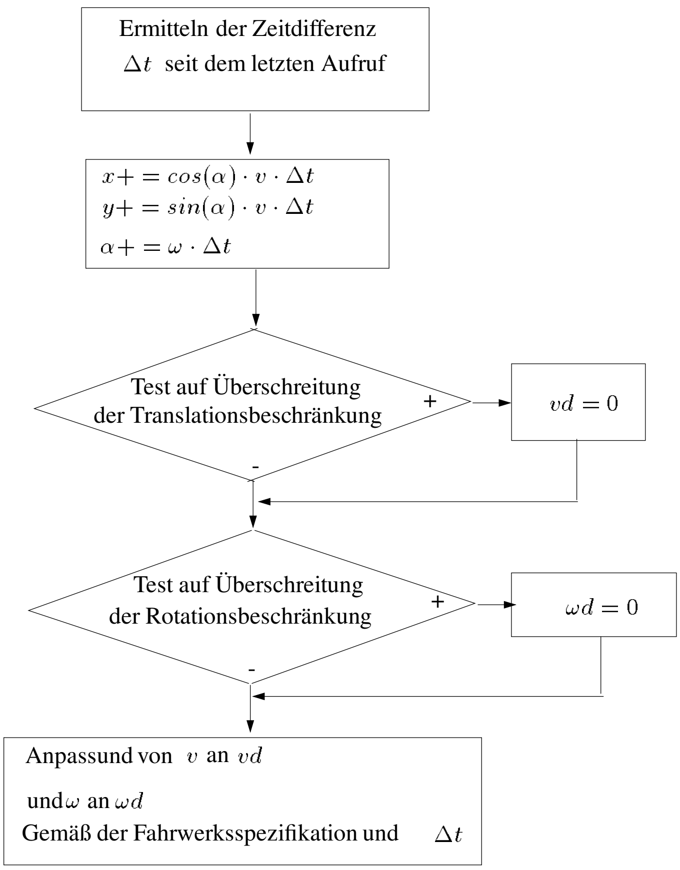

Abbildung 5.3: Flußdiagramm des Positionsupdates

Der Zustandsupdate, wie er in der Grafik 5.3 schematisch dargestellt ist, ist wie folgt implementiert. Zunächst wird die Position um die seit dem letzten Aufruf zurückgelegte Strecke inkrementiert. Danach wird analog dazu die neue Orientierung des simulierten Roboters mit dem Produkt von Rotationsgeschwindigkeit und verstrichener Zeit aktualisiert. Sowohl bei der Translation wie bei der Rotation wird überprüft, ob die durch den gerade gültigen Curve-Befehl vorgegebenen Beschränkungen für die Translationsdistanz und Rotationsdistanz überschritten wurden. Falls ja, so wird die entsprechende Aktion abgebrochen, d.h. in der Zustandsbeschreibung die angestrebte Rotations- bzw. Translationsgeschwindigkeit auf 0 gesetzt. Im letzten Schritt werden die Rotations- und Translationsgeschwindigkeit gemäß der Beschleunigungswerte aus der Roboterspezifikation an die angestrebten angeglichen.

In der Realität entspricht die vom Roboter angezeigte odometrische Position niemals perfekt der tatsächlichen. Durch Schlupf der Antriebsräder und mechanische Ungenauigkeiten bzw. ungenau ausgeführte Fahrbefehle entsteht eine Diskrepanz zwischen der errechneten und der tatsächlichen Position. Diese Diskrepanz muß selbstverständlich ebenfalls simuliert werden, da ansonsten der Roboter, in Abwesenheit von nicht in der globalen Karte verzeichneten Hindernissen, völlig blind zum Ziel finden könnte, was keinen realistischen Test für das Selbstlokalisationsmodul darstellt. Deshalb wird auf die intern vom Simulator errechnete Position und Orientierung ein einer Brownschen Bewegung folgender Driftvektor aufaddiert. Da die physikalischen Vorgänge, die die reale Drift verursachen, sehr komplex sind, ist eine akkuratere Modellierung wenig ergiebig in Bezug auf das Ziel einer realistischeren Simulation.

5.2.5 Simulation der Ultraschallsensoren

Die Ultraschallsensoren sind in der Simulation die einzigen Informationsquellen des Roboters, mit denen er seine Umgebung wahrnehmen kann. In der Realität sendet ein Ultraschallsensor einen Ultraschallimpuls aus und mißt aus der Laufzeit des Impulses bis zur Detektion des Echos die Entfernung bis zum Hindernis. Das Echo des Impulses kann nur registriert werden, wenn das Hindernis in einem Kegel von ca. $15° - 30°$ vor dem Roboter steht, das Material des Hindernisses den Ultraschallimpuls nicht zu stark dämpft und die Oberfläche des Hindernisses so orientiert ist, daß das Echo wieder zurück zum Ultraschallsensor gelangt.

Im folgenden wird beschrieben, wie die Simulation der Ultraschallsensoren abläuft. Dabei wird die Bitmaprepräsentation des Grundrisses der Büroetage als Umgebung verwendet, in der die simulierten Sensoren ihre Messungen vornehmen.

Ultraschallsensoren als perfekte Entfernungsmesser

In erster Näherung kann man Ultraschallsensoren als Entfernungsmesser ansehen, die die Entfernung zum nächstgelegenen Hindernis innerhalb des Sensorkegels angeben.

Auf der Grundlage dieser Modellvorstellung genügt es, diejenigen Kartenpixel zu betrachten, die sich, relativ zum simulierten Roboter, innerhalb des zum Sensor gehörigen Kegels befinden.

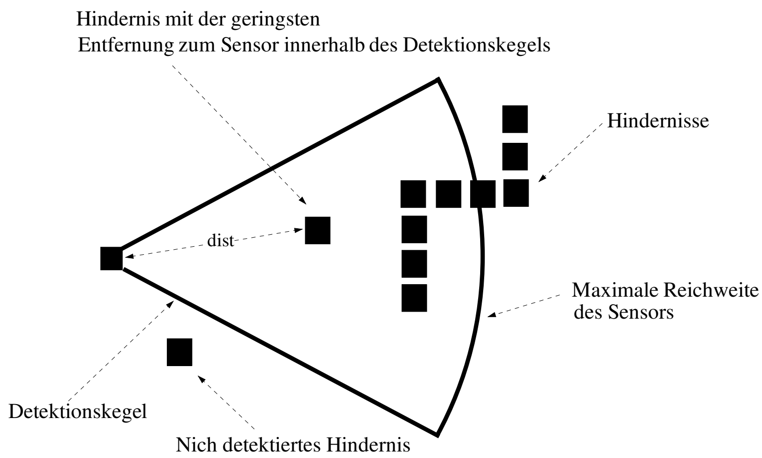

Abbildung 5.4: Ultraschallsensor als perfekter Entfernungsmesser

Die Entfernung, die der simulierte Sensor ausgibt, ist dann mit der kürzesten Entfernung zwischen Kegelspitze und einem gesetzten Pixel innerhalb des Kegels identisch.

In Bild 5.5 ist dargestellt wie dieses Pixel bestimmt wird. Um diejenigen Pixel in der Karte zu finden, die sich innerhalb des Kegels befinden, wird zunächst eine Bounding-Box ermittelt, die den Kegel, der an der Maximalreichweite der Ultraschallsensoren abgeschnitten wird, umschließt. Dann wird ein normierter Zentrumsvektor $\vec{z}$ errechnet, der genau in die Orientierung des Ultraschallsensors zeigt.

Die Projektion des normierten Ortsvektors $\vec{p}$ eines Pixel P innerhalb der Bounding-Box auf den Zentrumsvektor ist genau dann kleiner als der Cosinus des halben Kegelöffnungswinkels, wenn das Pixel innerhalb des Kegels ist. Also :

$$P \text{ ist im Kegel } \iff \vec{p} \bullet \vec{z} \leq cos(\gamma/2)$$

Die gesetzten Pixel in der Bounding-Box werden diesem Test unterzogen, und dasjenige mit der geringsten Entfernung zur Kegelspitze bestimmt die vom simulierten Ultraschallsensor ausgegebene Entfernung.

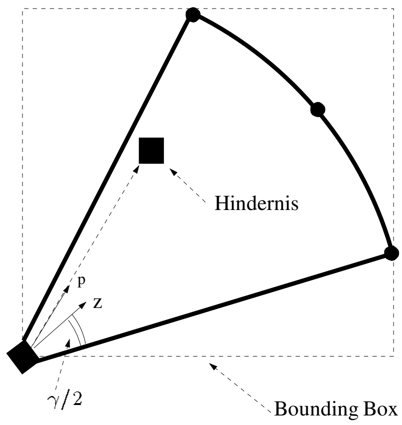

Abbildung 5.5: Ermittlung des kürzesten Abstands

Betrachtung der Reflexionseigenschaften des Hindernisses

Wie schon im Abschnitt 2.2 bei der Beschreibung der real vorhandenen Ultraschallsensoren erwähnt, werden Ultraschallimpulse besonders an glatten Flächen vom Sensor wegreflektiert. Dieser Effekt hinterläßt in den Ultraschallkarten sichtbare Störungen.

Ein möglicher Ansatz, diese Störungen in der Simulation zu brücksichtigen, ist, zusätzlich zur Bitmapkarte eine Orientierungskarte zu generieren, die angibt, in welche Richtung die zum Pixel gehörige Umgebung hauptsächlich orientiert ist und wie stark diese Orientierung ausgeprägt ist. Wenn das am wenigsten entfernte Pixel im Kegel so orientiert ist, daß der Einfallswinkel flacher ist als eine vorher festgelegte Schwelle und die Ausprägung dieser Orientierung stark genug ist, so gibt der simulierte Sensor aus, daß kein Echo empfangen wurde. Aus Zeitgründen wurde dieser Ansatz allerdings nicht weiterverfolgt.

5.3 Grenzen der Simulation

Jede Simulation kann nur einen Teilaspekt der Wirklichkeit wiedergeben. In diesem Abschnitt werden einige Effekte beschrieben, die zwar beim realen Betrieb des Roboters auftreten, aber nicht in der Simulation berücksichtigt werden.

Zunächst werden die Ultraschallsensoren nur unvollkommen simuliert. Abgesehen von der groben Repräsentation der Umgebung[2] wird der Weg des Ultraschallimpulses nicht in einem physikalisch akkuraten Modell verfolgt, sondern nur von der Umgebung direkt auf das Sensorergebnis geschlossen. Dabei bleiben Effekte, wie z.B. Dämpfung bei der Reflexion, Mehrfachreflexionen und Beugungseffekte un-

[2]Binärkarte mit 10 cm Auflösung

berücksichtigt. Die Energie, mit der ein Hindernis den Ultraschallimpuls aufgrund seiner Reflexionsfläche zurückwirft, wird ebenfalls nicht betrachtet, was zur Folge hat, daß ein einzelnes gesetztes Pixel in der Karte genauso gut detektiert wird, wie eine Wand. Ebenfalls nicht betrachtet werden bewegliche Hindernisse, deren Behandlung für den verwendeten Kartenbildungsalgorithmus nicht unproblematisch sind. Das zugrundeliegende mathematische Modell setzt nämlich eine statische Umgebung voraus. Auch die Simulation der Roboterbewegung ist nicht völlig perfekt. Beispielsweise der Schlupf wird nicht in Abhängigkeit von den Fahrbefehlen generiert.

Ein Grund für diese Ungenauigkeiten ist die Forderung der Echtzeitfähigkeit des Systems. Da das Verhalten der Steuerungskomponenten auch in der Simulation in Echtzeit geprüft werden muß, ist ein Simulator gefordert, der eine geringe Last für den Rechner darstellt und damit den Ablauf der Steuerungssoftware nur unwesentlich stört. Ein weiterer Grund ist, daß schon der vorhandene Simulator in praktikabler Weise einen Test der Steuerungssoftware zuläßt. Der erhebliche Mehraufwand, um die obigen Effekte in die Simulation aufzunehmen, rechtfertigt nicht die eher marginalen Gewinne bei Realitätsnähe der Simulation. Bei einer noch genaueren Simulation würde das Problem auftauchen, die notwendigen Parameter, die zur Simulation notwendig sind, zu erfassen. Die dazu nötige Messungen am Roboter wären höchstwahrscheinlich aufwendiger als die Testläufen der Steuerungssoftware mit der realen Roboterhardware, die man durch die Simulation ersetzen will.

Kapitel 6

Modulstruktur der Software

In diesem Kapitel wird die Modulstruktur der auf dem virtuellen Roboter aufbauenden Navigations- und Steuerungssoftware vorgestellt und deren Aufgabenbereiche definiert. Jedes Modul ist ein eigenständig ablaufender Prozeß, der über Netzwerkverbindungen mit anderen Modulen Nachrichten austauscht. Dies bedeutet, daß die Module fast beliebig in einem Rechnernetz verteilt werden können. Einschränkungen gibt es z.B. bei Prozessen, die direkt die Hardware ansteuern. Speziell also der Prozeß (oder die Prozesse), der (die) den virtuellen Roboter repräsentieren und deshalb auf Hardwareschnittstellen zugreifen müssen, sind an vorgegebene Rechner gebunden. Eine weitere Einschränkung ist, daß in langsamen Netzwerken die Latenzzeiten so stark ansteigen können, daß die Software unbenutzbar wird. Eine Kommunikation über mehrere Router hinweg ist also nicht zu empfehlen.

Der zugrundeliegende Gedanke bei der Aufgabenverteilung ist die zunehmende Abstraktion von eingehenden Sensorinformationen und die zunehmende Konkretisierung von Fahrbefehlen, die letztendlich direkt die Hardware steuern. Jedes Modul kann dabei entweder dem planenden oder einem wahrnehmenden Teil der Software zugeordnet werden, wobei die verschiedenen Teile hauptsächlich auf gleichen Ebenen von Konkretisierung bzw. Abstraktion miteinander kommunizieren.

Ein Modul, das sehr abstrakt gehaltene Fahrbefehle generiert, erhält die dazu notwendigen Informationen über die Umwelt fast ausschließlich von einem Wahrnehmungsmodul, das ebenfalls nur sehr abstrakte Informationen liefert.

6.1 Die Planungsmodule

Die Planungsmodule haben die Aufgabe, aus einer abstrakten Aufgabenstellung konkrete Fahrbefehle zur Hardwareansteuerung zu generieren. Es folgt zunächst eine grobe Aufgabenbeschreibungen für die einzelnen Planungsmodule. Implementationsdetails, wie die verwendeten Algorithmen und die exakte Beschreibung der

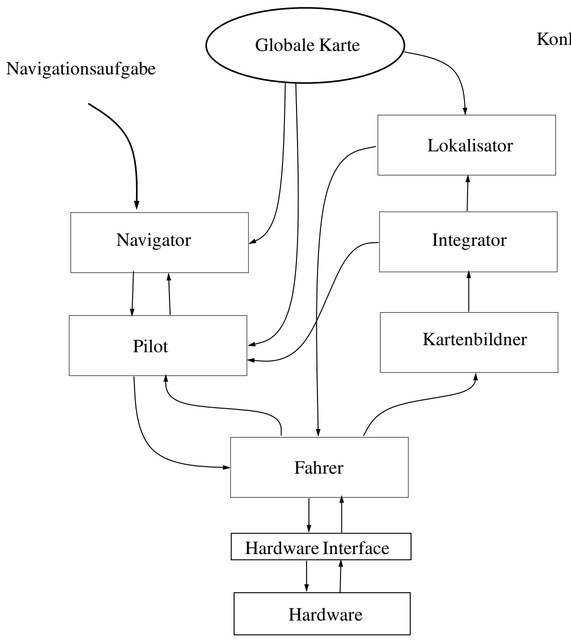

Abbildung 6.1: Modulübersicht

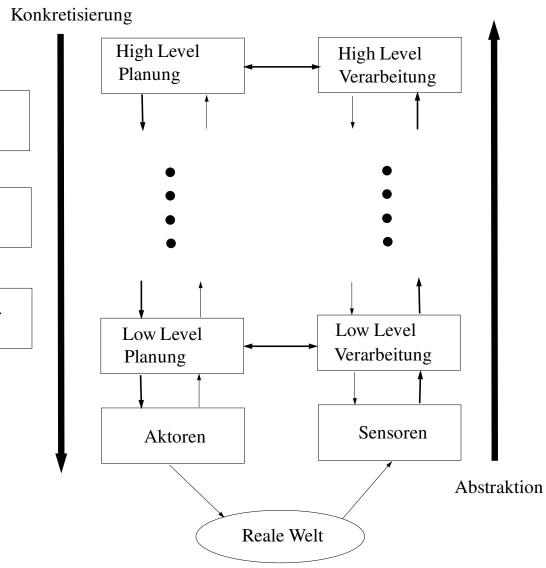

Abbildung 6.2: Prinzipien der Aufgabenteilung

Kommunikation der Module untereinander, folgen im Abschnitt 7.1.1.

6.1.1 Der Fahrer

Der Fahrer ist die Schnittstelle der Navigationssoftware zum (virtuellen) Roboter. Der Fahrer sendet konkrete Fahrbefehle[1] an den Roboter. Vom Roboter erhält er die Odometriedaten und die Entfernungsmessungen der Ultraschallsensoren. Diese Sensordaten werden an den Kartenbildner weitergeleitet und zusätzlich innerhalb des Fahrers zur reaktiven Hindernisvermeidung verwendet. Die reaktive Hindernisvermeidung erhält vom übergeordneten Piloten zwei Wegpunkte, die unter Vermeidung von Kollisionen angesteuert werden sollen. Da die Fahrbefehle des virtuellen Roboters immer nur für wenige Sekunden gültig sind, erwartet der Fahrer, daß vor dem Ablauf einer solchen Frist ein aktualisierter Fahrbefehl eintrifft. Ansonsten wird ein Nothalt ausgelöst.

[1] Curve-Befehle

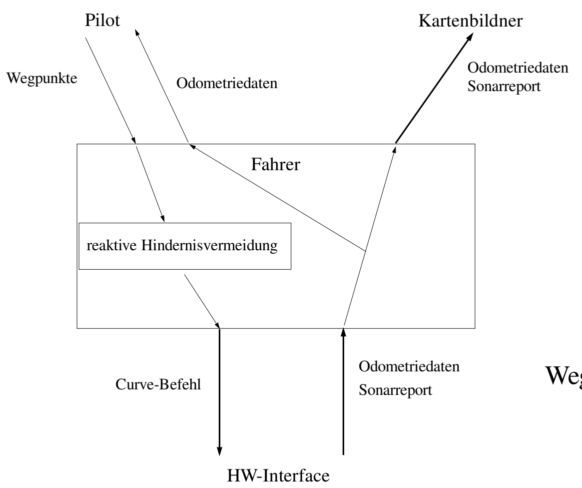

Abbildung 6.3: Datenflußschema des Fahrers

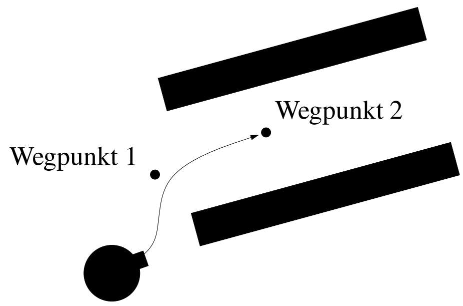

Abbildung 6.4: Aufgabe des Fahrers

6.1.2 Der Pilot

Der Pilot hat die Aufgabe, die lokale Wegplanung in einem ca. 2 Meter großen Radius um den Roboter durchzuführen. Die Grundlage für diese Planung ist ein abstrakterer Fahrbefehl des übergeordneten Navigators. Der Navigator gibt einen ca. 2 Meter oder weniger vom Roboter entfernten Punkt als Ziel an. Der Pilot hat nun die Aufgabe, dem Fahrer einen Fahrbefehl zu übermitteln, der den Roboter auf einem 'optimalen' Pfad zum Ziel führt. Was bei dieser Problemstellung ein optimaler Pfad ist, ist nicht eindeutig festgelegt. Mögliche Zielsetzungen wären: geringer Energieverbrauch, hohe Geschwindigkeit, geringer Zeitbedarf oder geringes Risiko einer Kollision. Dabei ist zu beachten, daß die Planung in Echtzeit stattfindet, also ein perfekter Pfad im obigen Sinne mit zu langer Planungsdauer inakzeptabel und ein etwas vom theoretischen Optimum abweichender Pfad mit ausreichend kurzer Planungszeit das beste erreichbare Ergebnis sein kann. Der Zeitraum, der jeweils zur Planung zur Verfügung steht, wird durch die Frist, nach der der aktuelle Fahrbefehl ungültig wird, begrenzt[2]. Der Fahrer verhindert zwar bei einer Zeitüberschreitung eine Kollision[3], aber ständiges Stoppen und Anfahren, bedingt durch ein zu kompliziertes Wegplanungsverfahren, ist in Bezug auf jedes vernünftige Optimalitätskriterium inakzeptabel.

[2] Siehe Def. des virtuellen Roboters

[3] Im Rahmen seiner Möglichkeiten. Wenn das Hindernis nicht rechtzeitig detektiert wird, so kann auch der Fahrer eine Kollision nicht verhindern

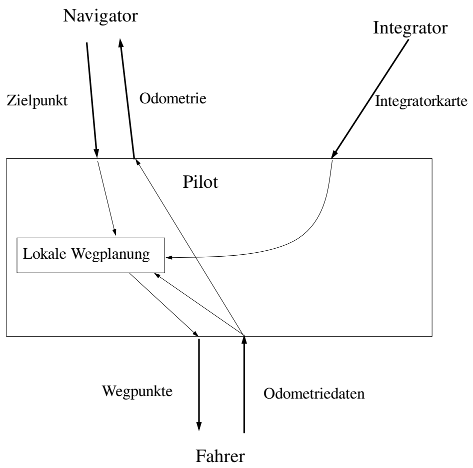

Abbildung 6.5: Datenflußschema des Piloten

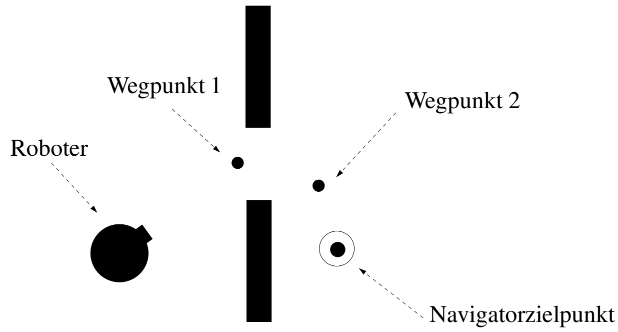

Abbildung 6.6: Aufgabe des Piloten

6.1.3 Der Navigator

Der Navigator nimmt von der Benutzeroberfläche abstrakte Fahrbefehle entgegen und konkretisiert diese zu Befehlsfolgen, die der Pilot weiter verarbeiten kann. Ein abstrakter Befehl könnte z.B. lauten: 'Fahre zum Raum 327' oder 'Fahre zu den Koordinaten (173,22) auf dem Grundriß'. Dazu steht ihm der Grundriß des Gebäudes und die korrigierte Odometrische Position zur Verfügung. Er legt einen Pfad vom aktuellen Standort zum Ziel fest und zerlegt den Pfad in Teilstücke, die dem Piloten als Aufgabe übertragen werden können. Der Navigator übernimmt diese Teilaufgaben nicht selbst, weil davon ausgegangen wird, daß der Grundriß nicht in jedem Detail mit der tatsächlichen Situation übereinstimmt und er deshalb gar nicht in der Lage ist, eine detaillierte Planung durchzuführen. Damit er überhaupt erfolgreich agieren kann, darf der Grundriß also nur Ungenauigkeiten enthalten, die der Pilot ausgleichen kann.

6.2 Die Wahrnehmungsmodule

Die Wahrnehmungsmodule werden in dieser Arbeit nur der Vollständigkeit halber erwähnt. Eine detaillierte Betrachtung ist in der Diplomarbeit "Driftkompensation auf der Basis von Besetztheitsgittern" von Timm Haase zu finden.

6.2.1 Der Kartenbildner

Der Kartenbildner hat die Aufgabe aus einem einzigen Satz von Ultraschallentfernungsdaten eine Kurzzeitkarte zu erzeugen. Diese Kurzzeitkarte wird dann an den

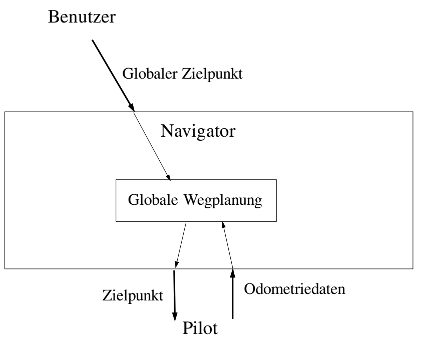

Abbildung 6.7: Datenflußschema des Navigators

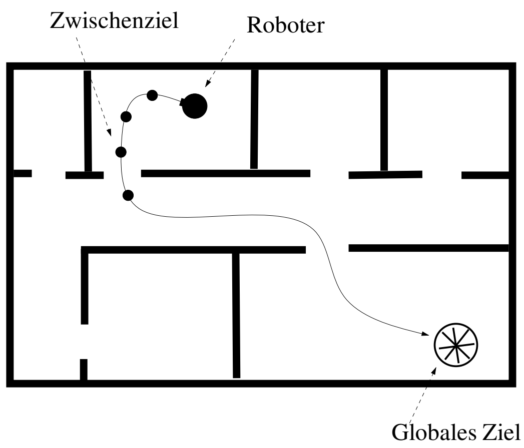

Abbildung 6.8: Aufgabe des Navigators

Integrator weitergegeben.

6.2.2 Der Integrator

Der Integrator empfängt Kurzzeitkarten vom MAPMAKER und integriert diese gemäß [6] zu Besetztheitskarten. Diese werden dann an den Lokalisator und den Piloten gesandt.

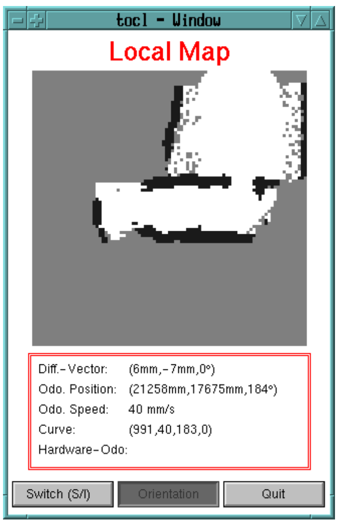

Abbildung 6.9: Eine Integratorkarte

6.2.3 Der Lokalisator

Der Lokalisator hat die Aufgabe, die Position des Roboters durch Vergleich der globalen Karte mit der Integratorkarte zu bestimmen und dadurch die Drift der Odometriedaten auszugleichen oder sogar die Odometriedaten vollständig zu ersetzen.

Kapitel 7

Kommunikation der Module untereinander

Im vorangegangenen Kapitel wurde behandelt, wie und warum die Steuerungssoftware des Roboters in Module unterteilt ist und welche Aufgaben die einzelnen Module übernehmen. Damit diese Module ein funktionierendes System ergeben können, ist Kommunikation notwendig. In diesem Kapitel wird beschrieben, wie diese Kommunikation abläuft. Zunächst wird inhaltlich beschrieben, welche Informationen die Module austauschen müssen. Daraus und aus einer Betrachtung, wie der Informationsaustausch zeitlich synchronisiert werden könnte, wird ein Anforderungskatalog für die Kommunikationsverbindungen erstellt und schließlich beschrieben, wie die geforderten Verbindungen in einem UNIX-Rechnernetz realisiert wurden.

7.1 Informationsströme zwischen den Modulen

In diesem Abschnitt wird der Informationsaustausch zwischen den einzelnen Modulen beschrieben. Kommunikationsanteile, die zur Synchronisation bzw. zur Verbesserung des Laufzeitverhaltens der Module benötigt werden, werden hier noch nicht behandelt. Die Beschreibung dient als Grundlage für den in nachfolgenden Abschnitt behandelten Anforderungskatalog an die Kommunikationsverbindungen.

7.1.1 Inhaltliche Beschreibung der Kommunikation

Die Graphik zeigt die grobe Verbindungsstruktur der Module. Die inhaltliche Beschreibung der Informationsströme erfolgt in den folgenden Unterabschnitten.

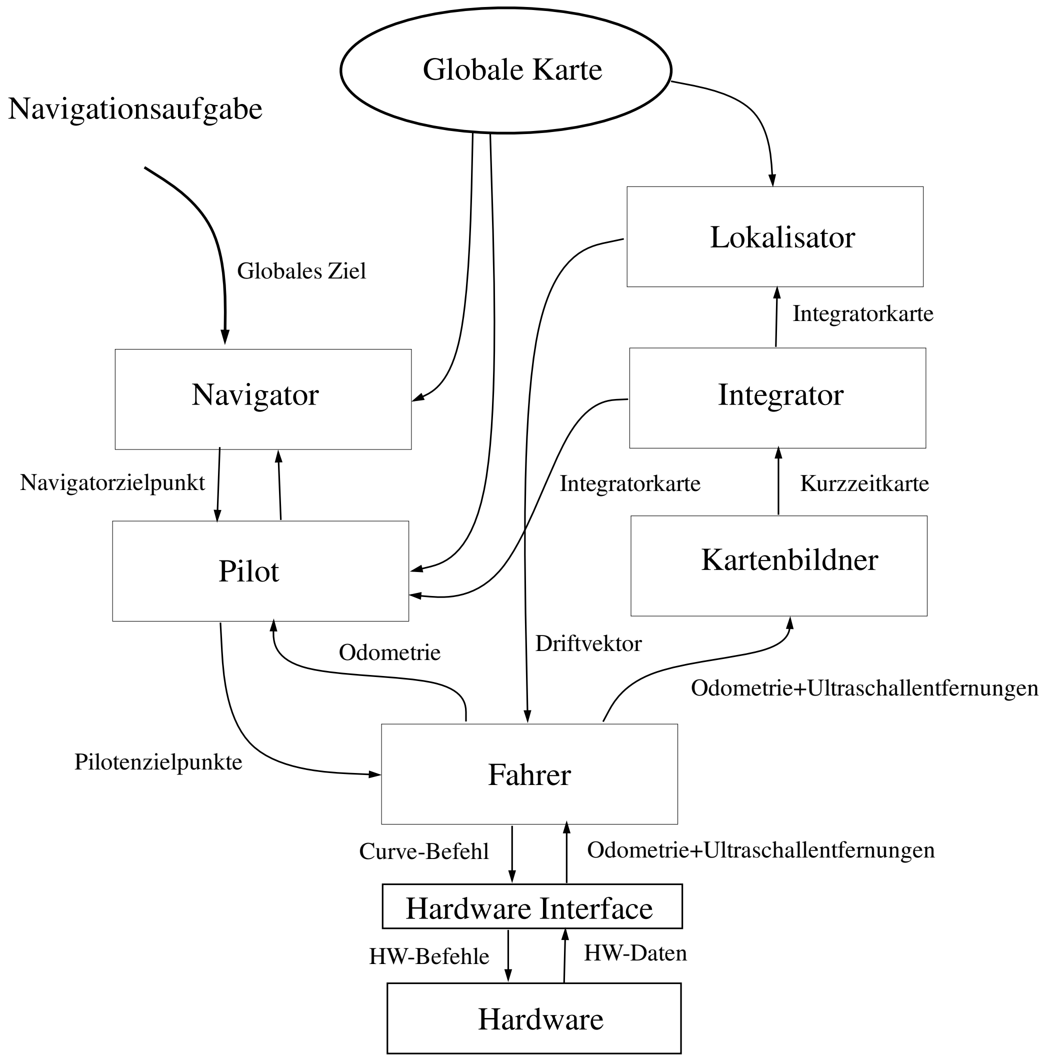

Abbildung 7.1: Modulübersicht

Kommunikation zwischen Hardwareinterface und Fahrer

Der Fahrer ist das hardwarenächste Modul, das selbst noch hardwareunabhängig ist. Der Fahrer empfängt vom Hardwareinterface die aktuellen Odometriedaten und sendet hardwareunabhängige Fahrbefehle an das Hardwareinterface, das die hardwareabhängige Steuerung übernimmt.
Die Odometriedaten setzen sich folgendermaßen zusammen:

- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Orientierung des Roboters in globalen Koordinaten

- Translationsgeschwindigkeit des Roboters
- Rotationsgeschwindigkeit des Roboters
- Ausführungsgrad des letzten gegebenen Fahrbefehls

Im Laufe der Entwicklung haben sich zwei verschiedene Varianten von hardwareunabhängigen Fahrbefehlen herauskristallisiert. Die erste Variante gibt die Translations- und Rotationsgeschwindigkeit des Roboters zusammen mit einer Maximaldistanz für die Translation an. Die zweite Variante gibt eine absolute Zielorientierung mit Translationsgeschwindigkeit und Maximaltranslationsdistanz an. Die Gründe für diese Variantenbildung werden im Kapitel 10 behandelt. Um das Umschalten zwischen den einzelnen Varianten nicht unnötig zu erschweren, werden grundsätzlich alle diese Informationen übertragen, wobei bei einer konkreten Konfiguration nicht verwendete Komponenten vom Empfänger ignoriert werden.
Ein Fahrbefehl ist also aus folgenden Komponenten zusammengesetzt:

- Gewünschte Rotationsgeschwindigkeit der Roboters
- Gewünschte Translationsgeschwindigkeit der Roboters
- Erlaubte Maximaltranslationsdistanz
- Gewünschte Orientierung des Roboters

Kommunikation zwischen Fahrer und dem lokalen Kartenbildner

Der lokale Kartenbildner benötigt zur Bildung eines lokalen Sensoreindrucks vom Fahrer die aktuellen Entfernungsangaben der Ultraschallsensoren. Zusätzlich stellt der Fahrer die zu den Sensordaten zugehörigen Odometriedaten zur Verfügung, damit die vom lokalen Kartenbildner erzeugten Karten in anderen Modulen zur Navigation verwendet werden können.
Im Falle des RWI-Roboters besteht ein Sensordatenpaket, das vom Fahrer zum lokalen Kartenbildner gesendet wird, aus folgenden Komponenten:

- 24 Ultraschallentfernungswerte
- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Orientierung des Roboters in globalen Koordinaten

Bei anderen Robotern ist evtl. die Anzahl der Ultraschallentfernungswerte unterschiedlich.

Kommunikation zwischen lokalem Kartenbildner und Integrator

Der lokale Kartenbildner versorgt den Integrator mit Karten, die entsprechend dem Sensormodell aus einer einzigen Messung aller Ultraschallsensoren erstellt werden. Diese Kurzzeitkarten werden zusammen mit den zu den Sensordaten gehörigen Odometriedaten an den Integrator gesandt. Eine lokale Karte ist ein char-Array mit 41 mal 41 Elementen, was einem Gebiet von ca. 4 mal 4 Meter entspricht. Die zur Karte gehörigen Odometriedaten sind bei der Kartenbildung immer die unkorrigierten Daten, die das Fahrwerk liefert. Das ist notwendig, um die Konsistenz der Karten zu erhalten. Näheres dazu kann man in der Diplomarbeit von Timm Haase "Driftkompensation auf der Basis von Besetztheitsgittern" nachlesen.
Eine Kurzzeitkarte besteht aus folgenden Komponenten:

- Anzahl der Kartenpixel in X-Richtung
- Anzahl der Kartenpixel in Y-Richtung
- Auflösung der Karte im mm pro Pixel
- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Die Kurzzeitkarte als char-Array

Kommunikation zwischen Integrator und Lokalisator

Der Lokalisator erhält vom Integrator die Integratorkarte.
D.h. übertragen wird:

- Anzahl der Kartenpixel in X-Richtung
- Anzahl der Kartenpixel in Y-Richtung
- Auflösung der Karte im mm pro Pixel
- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Die Integratorkarte als char-Array

Kommunikation zwischen Fahrer und Lokalisator

Der Lokalisator sendet an den Fahrer die vermutete Drift der Hardwareodometrie, die der Fahrer dazu benutzt, den Navigationsmodulen driftunabhängig die korrekte Position in globalen Koordinaten zur Verfügung zu stellen.

Kommunikation zwischen Fahrer und Pilot

Der Fahrer sendet an den Piloten aktuelle Odometrieinformationen und empfängt vom Piloten Zielpunktpaare, die er in konkrete Fahrbefehle umsetzt.
Eine Nachricht des Fahrers an den Piloten besteht aus:

- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Orientierung des Roboters in globalen Koordinaten

Der Pilot sendet Befehle in folgender Form an den Fahrer:

- X-Position des ersten Zielpunktes in globalen Koordinaten
- Y-Position des ersten Zielpunktes in globalen Koordinaten
- X-Position des zweiten Zielpunktes in globalen Koordinaten
- Y-Position des zweiten Zielpunktes in globalen Koordinaten

Kommunikation zwischen Pilot und Navigator

Dem Navigator ist als einzigem Modul das globale Navigationsziel bekannt. Gemäß der Aufgabenbeschreibung des Navigators zerlegt er diese Navigationsaufgabe in Kurzstrecken-Navigationsaufgaben. Eine Kurzstrecken-Navigationsaufgabe wird dem Piloten durch einen Zielpunkt gegeben, der sich noch innerhalb der momentanen Integratorkarte befinden muß. Der Navigator sendet also einen Zielpunkt an den Piloten.
Der Navigator sendet also folgende Nachricht an den Piloten:

- X-Position des Zielpunktes in globalen Koordinaten
- Y-Position des Zielpunktes in globalen Koordinaten

In einer anderen Variante sendet der Navigator vier Kurzstreckenziele an den Piloten. Diese Ziele markieren das Anfangsstück des vom Navigator geplanten Pfades. Die Punkte liegen in regelmäßigen Abständen auf dem Pfad. Unter diesen Zielen wählt der Pilot dann das für die lokale Wegplanung geeignetste aus. Dieses Vorgehen ist notwendig, wenn ein nicht in der globalen Karte verzeichnetes Hindernis genau die Position des Navigatorziels blockiert (siehe 9.4).

Kommunikation zwischen Fahrer und Navigator

Der Fahrer sendet die Odometriedaten an den Navigator.
Eine Nachricht des Fahrers an den Navigator besteht aus:

- X-Position des Roboters in globalen Koordinaten
- Y-Position des Roboters in globalen Koordinaten
- Orientierung des Roboters in globalen Koordinaten

7.1.2 Kommunikation zur Synchronisation der Module

Die obige Beschreibung unterschlägt das Zeitverhalten der Module. Beispielsweise erzeugt der Kartenbildner aus wenigen Entfernungsdaten eine sehr viel speicherintensivere Karte. Es ist noch nicht klar, wie der Fahrer Sensordaten behandelt, die theoretisch an den Kartenbildner gesendet werden könnten, aber von diesem nicht oder noch nicht bearbeitet werden können, weil der Kartenbildungsprozeß auf der Basis von älteren Sensordaten noch nicht abgeschlossen ist. Ein ähnliches Problem tritt z.B. bei der Kommunikation von Fahrer und Pilot auf. Wenn der Pilot ein Zielpunktpaar an den Fahrer übergeben hat, ist nicht klar, wann es günstig oder sogar notwendig ist den nächsten Fahrbefehl zu senden.

Es muß also festgelegt werden, wann welche Information gesendet werden soll. Im folgenden werden zwei Möglichkeiten behandelt, dieses Problem zu lösen.

Nicht synchronisierte Kommunikation

Wenn ein Modul keine Rücksicht darauf nimmt, ob der Kommunikationspartner die Nachrichten, die es versendet, auch wirklich verarbeiten kann, so ist die Kommunikation nicht synchronisiert. Wenn neue Daten versandfertig sind, so werden sie übertragen. Es liegt in der Verantwortung des Empfängers, die Daten so schnell zu verarbeiten wie sie gerade eintreffen. Eine Alternative ist, Nachrichten, die aus Kapazitätzgründen nicht verarbeitet werden können, zu verwerfen. Beispielsweise bei dem Kommunikationspaar Fahrer - Kartenbildner würde das bedeuten, daß der Fahrer neue Sensordaten an den Kartenbildner abschickt, sobald er dazu in der Lage ist. Der Kartenbildner müßte nun entweder seine Karten schneller erzeugen als der Fahrer neue Daten abschickt, oder er verwirft veraltete Informationen und behandelt z.B. nur die zuletzt angekommene Nachricht.

In der Praxis hat sich herausgestellt, daß nicht synchronisierte Kommunikation sehr problematisch ist. Der scheinbare Gewinn auf zusätzliche Kommunikation zu Synchronisationszwecken zu verzichten, wird dadurch ins Gegenteil verkehrt, daß das Erzeugen, Übertragen und Verwerfen von nicht benötigten Daten im erheblichen Maße Rechenzeit kostet[1].

Ein Grund dafür ist, daß die hardwarenäheren Module eine kürzere Zykluszeit als die Module auf höheren Abstraktionsebenen haben. Da der Fahrer z.B. überra-

[1] Beim Wechsel zur synchronisierten Kommunikation konnte die Rechnerlast mehr als halbiert werden

schenden Hindernissen ausweichen muß, muß die Hauptschleife des Fahrermoduls mehrmals pro Sekunde ausgeführt werden, wohingegen der Navigator nur nach jeweils einigen Sekunden aktiv werden muß. Bei den übergeordneten 'langsamen' Modulen staut sich der kontinuierliche Datenstrom, der vom Empfänger entsorgt und vom Absender völlig unnötig erzeugt wird.

In einem Computersystem, in dem sich die Module wenige oder evtl. sogar nur einen Prozessor teilen müssen, überwiegt schnell der Anteil der verschwendeten Rechenkapazität. Die Scheinlösung, die Geschwindigkeit der Module per 'nice' Anweisung[2] zu nivellieren, ist nicht praktikabel, da, wie schon erwähnt, die hardwarenahen Module naturgemäß 'schneller' ablaufen müssen als die übrigen. Ansonsten würde z.B. die extrem wichtige Kollisionsvermeidung künstlich reaktionsträger gemacht. Dies stellt keinesfalls eine akzeptable Lösung des Synchronisationsproblems dar.

Die Idee, die schnelleren Module um einen festen Faktor n häufiger ablaufen zu lassen und nur bei jedem n'ten Durchlauf Nachrichten zu versenden, ist zwar im Prinzip durchführbar, allerdings nur mit Echtzeitbetriebssystemen, die ein derartiges festes Teilungsverhältnis im Ablauf verschiedener Prozesse unterstützen. In einer UNIX-Umgebung kann per 'nice' nur die Priorität eines Prozesses aber nicht exakt die Häufigkeit der Ausführung eines bestimmten Codestücks festgelegt werden. Bei auf mehrere Rechner verteilten Modulen würde in jedem Fall Kommunikation zur Sicherstellung des festen Geschwindigkeitsverhältnisses erforderlich sein.

Eine unkoordinierte Kommunikation ist nur dann akzeptabel, wenn der empfangende Prozeß garantieren kann, zum Zeitpunkt der Sendung empfangsbereit zu sein. Selbst wenn Nachrichten, die an einen nicht empfangsbereiten Prozeß gesendet werden, vom System gepuffert werden, muß der Empfänger zumindest im zeitlichen Mittel mehr Daten verarbeiten können, als vom Sender geliefert wird. Ansonsten entsteht ein Nachrichtenstau, was eine geringere Verarbeitungsleistung des Gesamtsystems zur Folge hat.

Synchronisierte Kommunikation

Eine synchronisierte Kommunikation stellt sicher, daß Nachrichten, die abgeschickt werden auch empfangen werden können. Dafür ist allerdings zusätzlicher Kommunikationsaufwand notwendig. Der empfangende Prozeß muß dem sendenden mitteilen, daß er bereit ist, die nächste Nachricht zu bearbeiten, oder ein Puffer speichert die Nachricht so lange, bis der Empfänger die Nachricht akzeptiert. Da im vorliegenden Fall mit puffernden Verbindungen gearbeitet wird, die im wesentlichen wie ein FIFO-Stack arbeiten, geht die Nachricht in keinem Fall verloren. Sobald

[2] Ein Unix Systemkommando

der sendende Prozeß in der Lage ist wieder zu kommunizieren, empfängt er die Anfrage und sendet daraufhin die angeforderte Nachricht, auf die der Empfänger blockierend wartet.

Um die negativen Folgen einer unkoordinierten Kommunikation zu vermeiden, sind in der Implementation fast alle Verbindungen so konstruiert, daß der Empfänger die Datenübertragung anstößt.

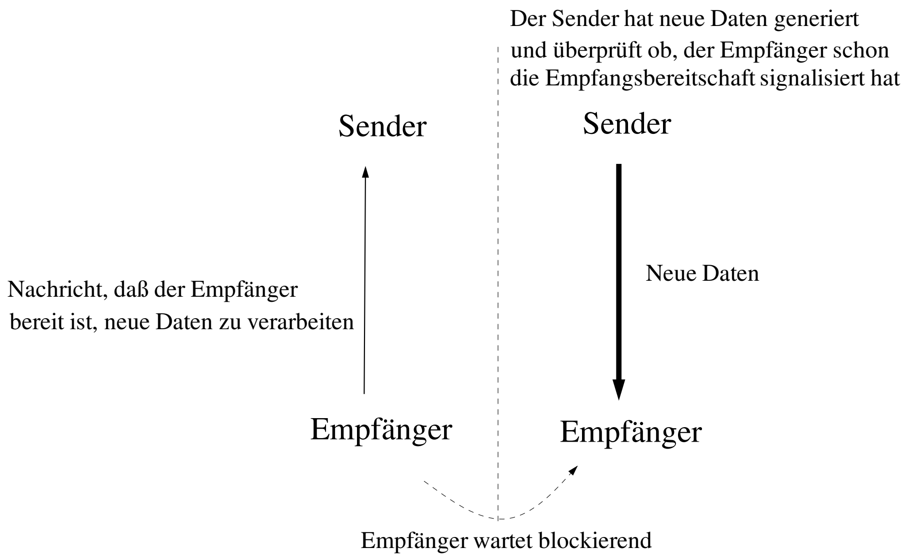

Abbildung 7.2: Datenübertragung auf Anforderung

7.2 Implementation des Kommunikationssystems

In diesem Abschnitt wird zunächst die abstrahierte Softwareschnittstelle zum Kommunikationssystem vorgestellt. Nachfolgend werden die Probleme bei der Initialisierung der einzelnen Module und der damit verbundenen initialen Herstellung der Verbindungen zwischen den Modulen behandelt und schließlich die konkrete Implementation der Kommunikationsverbindungen beschrieben.

7.2.1 Die abstrakte Kommunikationsklasse

Um das Programm weitgehend unabhängig von der konkreten Implementation der Kommunikationsverbindungen schreiben zu können und durch die Beschränkung auf die absolut notwendige Abstraktion der Schnittstelle die Programmierung einfacher und damit sicherer zu gestalten, wurde eine C++ Klasse 'messageport' mit folgenden Methoden eingeführt.

```
void offer_messageport(char *name);
void connect_to_messageport(char *name, char *own_name);
void close_messageport(void);
void send_msg(char *str,int len,char *debug_info);
char *get_msg(int block, int &len, char *debug_info);
```

Ein 'messageport' dient als Endpunkt einer Kommunikationsverbindung. Bei aufgebauter Verbindung kann mit der Methode 'send_message' ein Speicherbereich an den Kommunikationspartner versandt werden und mit 'get_message' empfangen werden.

Pollen oder Blockieren ?

Beim Nachrichtenempfangen existieren zwei Varianten. In der ersten Variante wartet der Empfänger blockierend, bis eine Nachricht ankommt. In der zweiten Variante wird überprüft, ob eine Nachricht zum Empfang bereitliegt. Falls nicht, so wird ein entsprechender Fehlercode erzeugt.

Falls eine Nachricht gesendet werden soll, die durch die Kommunikationsverbindung nicht gepuffert werden kann, so blockiert der sendende Prozeß so lange, bis wieder genug Speicherplatz zur Pufferung vorhanden ist. Das reale Pufferverhalten der Kommunikationsverbindung ist leider implementationsspezifisch. Einige Implementationen puffern fast beliebig viele Nachrichten, wohingegen andere nur eine einzige Nachricht puffern können, was schon bei der zweiten gesendeten aber noch nicht empfangenen Nachricht zum Blockieren des Sendeprozesses führt.

Byte-Order

Die Nachrichten sind char-strings. Wenn der Benutzer kompliziertere Datentypen als chars übertragen will/muß, so liegt es in seiner Verantwortung, die Byte-Order der übertragenen Daten richtig zu interpretieren. Beispielsweise laufen bei den Versuchen mit dem RWI-Roboter einige Prozesse auf einer SUN-Workstation mit Little-Endian- und die restlichen Prozesse auf Intel-Prozessoren mit Big-Endian-Format.

Verbindungsaufbau aus der Sicht des Benutzers

Der Verbindungsaufbau ist für den Benutzer genauso einfach wie das Versenden und Empfangen von Nachrichten. Im Bild 7.3 ist abgebildet, wie der erste Prozeß eines Kommunikationspaares mit der Funktion 'offer_messageport' und einer Zeichenkette als Identifikator die Bereitschaft signalisiert, eine Verbindung aufzubauen. Er blockiert nun solange bis sich ein Kommunikationspartner findet, der das Kommunikationsangebot annimmt. Der zweite Prozeß kann nun mit 'connect_to_messageport' unter Angabe desselben Identifikators die Verbindung zum ersten Prozeß herstellen.

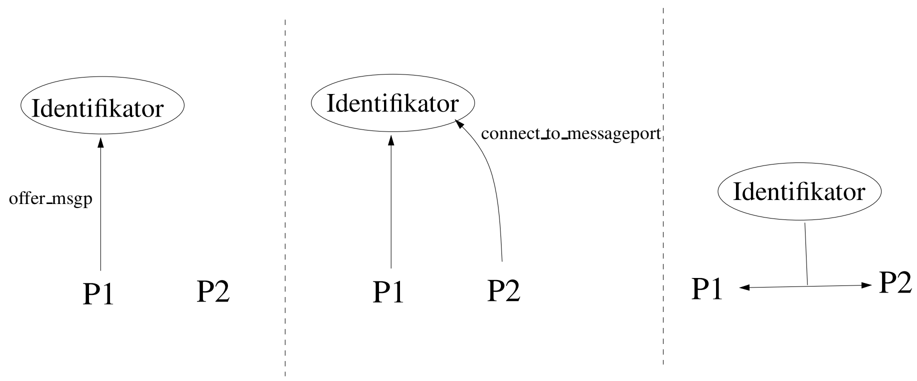

Abbildung 7.3: Verbindungsaufbau aus der Sicht des Benutzers

Sollte der erste Prozeß noch nicht bis zur Anweisung ’offer_messageport’ gekommen sein, blockiert der zweite Prozeß so lange, bis dies der Fall ist. Nach erfolgreichem Connect steht eine bidirektionale Verbindung zur verfügung, auf die mit ’send_message’ und ’get_message’ zugegriffen werden kann.

7.2.2 Start der Module und Aufbau der Verbindungen

Die einzelnen Module werden von einem separaten Startprozeß aus gestartet. Ein Modul baut dann zunächst eine Verbindung[3] zum Startprozeß auf. Danach wird die Initialisierungsfunktion des Moduls ausgeführt. In der Initialisierungsfunktion wird u.a. für jede Kommunikationsverbindung des Moduls entweder einmal ’offer_messageport’ oder ein ’connect_to_messageport’ aufgerufen.

Bei einem ’offer_messagport’ wird über die Verbindung zum Startprozeß der Identifikationsstring und die implementationsspezifischen Verbindungsparameter übertragen. Im Startprozeß werden diese Informationen dann in eine Liste eingetragen.

Bei einem ’connect_to_messageport’ wird ebenfalls der Identifikationsstring zum Startprozeß übertragen. Der Startprozeß prüft dann, ob der Identifikator schon in der Liste eingetragen ist. Wenn dies der Fall ist, so sendet der Startprozeß die in der Liste eingetragenen Verbindungsparameter zurück. Das Modul hat nun alle Informationen, um eine direkte Verbindung aufzubauen. Wenn noch kein passender Listeneintrag vorhanden ist, so antwortet der Startprozeß, daß das Modul die Anfrage zu einem späteren Zeitpunkt wiederholen soll.

[3] Die Verbindung basiert auf Internet-Sockets

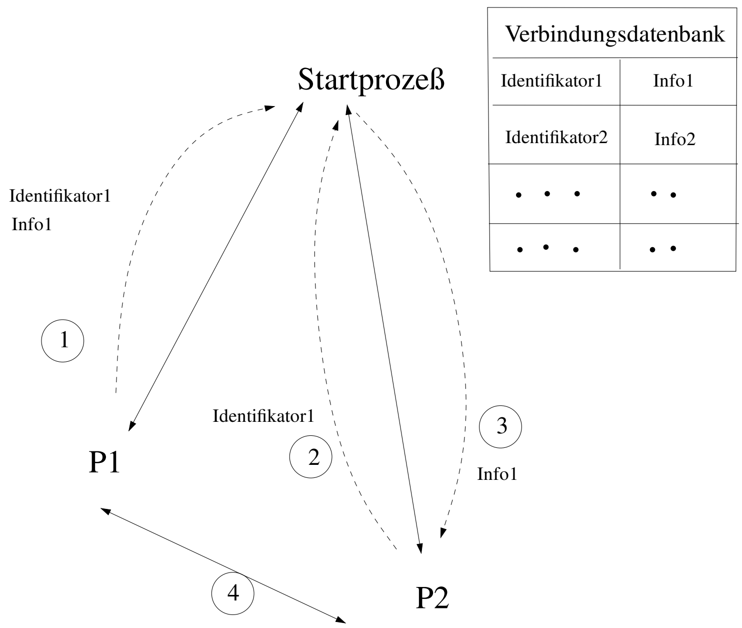

Abbildung 7.4: Startprozeß mit Datenbank

7.2.3 Die softwaretechnische Seite der Kommunikation

Die Messageport-Klasse ist nur eine Hülle für die konkrete Implementation der Verbindungen. In der Steuerungssoftware wurden drei verschiedene Varianten implementiert.

Implementation mit PVM

PVM ist ein Softwarepaket, mit dem man weitgehend betriebssystemunabhängig verteilte Programme implementieren kann. Es stehen Funktionen sowohl zur Kommunikation als auch zur Prozessererzeugung und -überwachung zur Verfügung.

Der große Vorteil von PVM ist die Unabhängigkeit vom Betriebssystem. Nur mit Standard-C und PVM-Funktionen ist man in der Lage, ein verteiltes Programm zu schreiben. Die Portierung auf andere Plattformen ist sehr einfach und Spezialkenntnisse über das konkret verwendete Kommunikationssystem sind nicht erforderlich. Zudem verwendet PVM, falls eine Wahlmöglichkeit besteht, die effizienteste zur Verfügung stehende Methode zum Informationstransport. Die PVM-Kommunikationsfunktionen sind sehr einfach zu verstehen, so daß an dieser Stelle nicht auf die konkrete Implementation eingegangen wird. Die Implementation der Messageport-Methoden erfordert nur wenige Zeilen Programmcode.

Zur Kommunikation via PVM muß auf jedem beteiligten Rechner ein PVM-Server-Programm laufen. Jegliche Kommunikation wird über diesen Server abgewickelt, was die Kommunikation, im Vergleich zu direkten Verbindungen, verlangsamt.

Trotz des Nachteils der (theoretisch) geringeren Geschwindigkeit wäre das PVM-System eine attraktive Implementationsbasis für die Messageport-Klasse gewesen. Leider machte eine nicht nachvollziehbare Weigerung des PVM-Systems, eine Verbindung zwischen zwei bestimmten Rechnern aufzubauen, den Einsatz unmöglich. Ob ein Fehler in der Konfiguration der beteiligten Rechner oder eine Inkompatibilität der beteiligten Betriebssysteme den Verbindungsaufbau verhindert haben, war nicht feststellbar. Ebenfalls möglich ist ein Fehler im PVM-System. Wegen der Komplexität und des enormen Umfangs des PVM-Quellcodes konnte diese Möglichkeit nicht befriedigend überprüft werden.

Direktimplementation mit UNIX-TCP/IP-Sockets

Das TCP/IP-Protokoll ist als Internetstandard für fast jedes Betriebsystem implementiert. Bei der Beschränkung auf UNIX-Betriebssysteme kann sogar von einer weitgehenden Quellcodekompatibilität ausgegangen werden. Das Verhalten einer TCP/IP-Verbindung ist in bezug auf die hier vorliegende Aufgabenstellung leider nicht optimal. Nachrichten, die nur wenige Bytes lang sind benötigen genausoviel Zeit zum Transport wie Nachrichten, die einige tausend Bytes lang sind !

In einem Versuch werden Datenpakete in der Größe von einem Byte bis 50 KB von einem Prozeß zu einem anderen und wieder zurück übertragen. Für jedes Paket wird die Zeit gemessen, die mit einer TCP/TP-Verbindung für diese Rundreise des Datenpakets erforderlich ist.

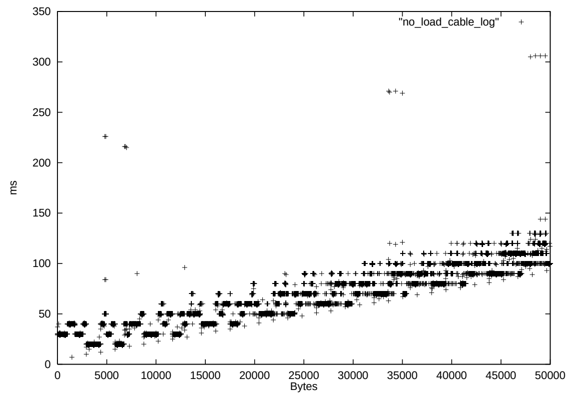

Abbildung 7.5: Paketgröße vs. Übertragungszeit bei TCP/IP-Verbindung

Grafik 7.5 zeigt eine Messung während eines Testlaufs der Navigationssoftware. Man erkennt, daß für Paketgrößen unter 5 KB fast konstant 40 ms für die Übertragung eines Pakets benötigt werden und der Zeitbedarf erst ab einer Paketgröße von 5 KB linear ansteigt. Bei Verwendung des TCP/IP-Protokolls werden also sehr kurze Nachrichten wesentlich ineffizienter übertragen als längere. Die gewünschte Eigenschaft, sehr kurze Nachrichten in entsprechend kurzer Zeit zu übertragen bzw. sogar die größte Effizienz bei sehr kurzen Nachrichten zu erreichen, ist gerade nicht erfüllt! Die Ineffizienz bei der Übertragung sehr kleiner Nachrichten ist allerdings nicht allein dem Übertragungsprotokoll anzulasten. Da bei vielen aktiven Prozessen die Zeit vom physikalischen Nachrichtenempfang bis zur Aktivierung des Empfängerprozesses durch den Scheduler einige Millisekunden vergehen können, muß von der gemessenen Latenz ein Sockelbetrag abgezogen werden, um die Latenz des Protokolls zu erhalten.

Verwendung von shared-memory Segmenten zur lokalen Kommunikation

Um zumindest die Kommunikation innerhalb eines Rechners so effizient wie nur möglich zu implementieren, wurde die Messageport-Klasse um die Fähigkeit erweitert, zu erkennen, ob eine Kommunikation lokal innerhalb eines Rechners oder über ein Netzwerk zu einem anderen Rechner stattfindet. Lokal wird dann anstelle von TCP/IP-Sockets ein shared-memory Segment zur Kommunikation genutzt.

Das shared-memory Segment wird bei 'offer_messageport' bzw. 'connect_to_messageport' jeweils in den Adressraum des jeweiligen Prozesses eingebunden und durch Semaporen in den Methoden 'get_msg' bzw. 'put_msg' vor dem jeweils anderen Prozeß geschützt.

In der Grafik 7.6 erkennt man die Übertragungsleistung einer shared-memory Verbindung. Der Zeitaufwand für die Übertragung ist fast konstant.

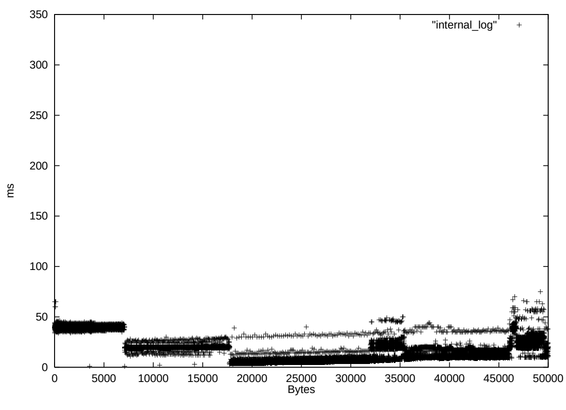

Abbildung 7.6: Paketgröße vs. Übertragungszeit bei SHM-Verbindung

7.2.4 Vor- und Nachteile der Implementationen

Die Implementation mit PVM war vom Entwicklungsaufwand her gesehen wesentlich einfacher als die TCP/IP Variante mit lokaler shared-memory Kommunikation. Ebenso kann davon ausgegangen werden, daß die Datenübertragung über das Netzwerk mit PVM effizienter implementiert ist als mit TCP/IP Sockets. In der Dokumentation zu PVM wird angedeutet, daß direkt Datagramme erzeugt werden, wobei die Garantie der Datenzustellung über ein PVM spezifisches Protokoll gewährleistet wird. Da PVM allerdings mit der entscheidenden Gerätekonfiguration nicht funktionierte und eine schnelle Korrektur des Quellcodes zur Fehlerbehebung bzw. als Workaround für inkompatible Systeme völlig illusorisch ist, wurde der PVM-Ansatz nicht weiter verfolgt.

Das TCP/IP-Protokoll ist zwar wegen seiner Ineffizienz bei kurzen Nachrichten nicht optimal für die gegebene Aufgabenstellung, es hat sich aber im praktischen Betrieb als ausreichend erwiesen. Durch die Verwendung von shared-memory Segmenten für die lokale Kommunikation wird zudem für den überwiegenden Teil der Verbindungen die Latenz auf das praktisch mögliche Minimum reduziert.

Hervorzuheben ist, das für den Benutzer die verschiedenen Implementationen völlig transparent sind. Der Programmcode zur Kommunikation innerhalb der Navigationssoftware ist für jede Version vollkommen identisch.

Teil II

Kapitel 8

Der Navigator

Der Navigator ist das einzige Modul, welchem von der Benutzeroberfläche der globale Zielpunkt mitgeteilt wird. Die Aufgabe des Navigators ist, das durch den globalen Zielpunkt gestellte Navigationsproblem in Teilprobleme zu zerlegen, die der Pilot bewältigen kann. Als Grundlage für diesen Vorgang dient ein Grundriß der Büroetage. Der Navigator erhält vom Fahrer die aktuelle Position des Roboters. Der Navigator plant auf dem Grundriß ausgehend von der aktuellen Position das Anfangsstück eines Pfades zum Zielpunkt. Auf diesem Anfangsstück werden in gleichmäßigen Abständen vier Zwischenziele ausgewählt und deren Position an den Piloten weitergegeben. Der Pilot hat dann die Aufgabe, falls möglich das entfernteste Zwischenziel anzufahren, ansonsten das nächstnähere usw. .

Die Grundrißkarte ist binär. Dem Navigator ist also bekannt, welche Pixel der Karte befahrbar sind. Zusätzlich hat er nur die Position des Roboters innerhalb der Karte zur Verfügung. Informationen über Orientierung, Geschwindigkeit etc. sind nicht vorhanden.

8.1 Planung auf dem Grundriß

Wie schon erwähnt, hat der Navigator die Aufgabe, eine Reihe von Zwischenzielen zu erzeugen, die dann an den Piloten übergeben werden. Die Zwischenziele sollen dabei so gewählt sein, daß der Roboter möglichst schnell und ohne Kollisionen zum globalen Zielpunkt gelangt. Es wird also ein Kostenmaß benötigt, mit dem die Qualität eines Zwischenzieles bewertet werden kann.

In einem ersten Vorbereitungsschritt wird hierzu eine euklidische Distanzkarte gebildet, die für jeden Punkt der Karte angibt, wie weit entfernt das nächste Hindernis ist. Die Bildung einer solchen Karte wird im Anhang A.1 im Detail behandelt. In nächsten Schritt wird dann jedem Entfernungswert ein Kostenwert zugeordnet. Diese Kosten sollen das Kollisionsrisiko wiedergeben, das eingegangen wird, wenn

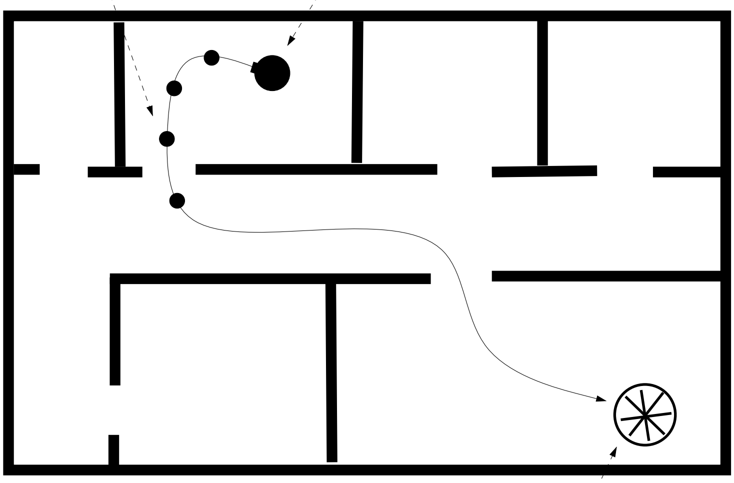

Abbildung 8.1: Zwischenziele als Pilotenaufgabe

der Roboter das zu den Kosten gehörige Feld betritt. Die Definition solcher Entfernungskosten wird in Anhang A.2 behandelt.

Die Karte wird mit diesen Entfernungskosten für jedes Feld als gewichteter und gerichteter Graph aufgefaßt. Betrachtet werden die nicht besetzten Felder[1] der Karte. Jedes Feld hat maximal acht Kanten, die in die Nachbarfelder[2] führen und maximal acht Kanten, die aus den Nachbarfeldern in das Feld hineinführen. Kanten, die in ein besetztes Feld hinein oder daraus herausführen würden, werden entfernt.

Jede der übrigen Kanten wird nun mit den Entfernungskosten ihres Zielfeldes gewichtet, wobei die Gewichtung der diagonalen Kanten zusätzlich um den Faktor $\sqrt{2}$ vergrößert wird. Das Problem, einen optimalen Pfad zum Zielpunkt zu finden, ist gleichbedeutend mit dem Problem, in dem gerichteten und gewichteten Graphen den kostengünstigsten Pfad zum Zielknoten zu finden.

8.1.1 Der Flood-Fill-Algorithmus

Für die obig formalisierte Problemstellung gibt es eine Vielzahl von Standardalgorithmen, wie z.B. den Dijkstra-Algorithmus. Hier wird ein dem Dijkstra-Algorithmus ähnlicher Flood-Fill-Algorithmus verwendet, der es ermöglicht nicht iterativ[3] für

[1] Nur nicht besetzte Felder sind erreichbar und können deshalb mit Kosten belegt werden
[2] bezüglich einer Achternachbarschaft
[3] Der Aufwand ist auch absolut betrachtet wesentlich geringer als z.B. bei *value iteration*

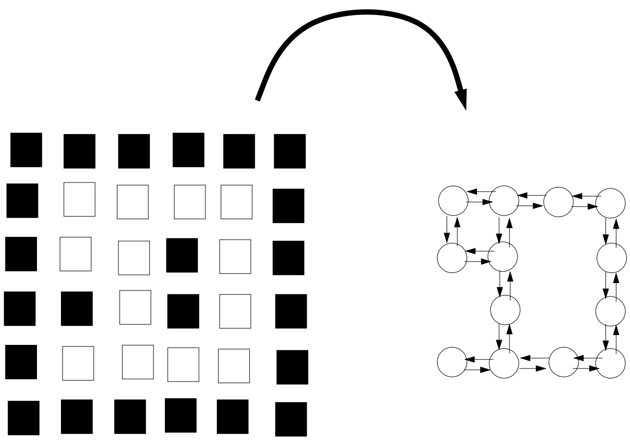

Abbildung 8.2: übergang von der Binärkarte zum Graphen

alle Punkte der Karte den optimalen Pfad zum Ziel zu finden.

Bei der Initialisierung wird jeder Knoten mit einem zusätzlichen Wert versehen. Dieser Wert wird im folgenden G-Wert des Knoten genannt. Alle G-Werte sind anfangs mit der größten darstellbaren Zahl belegt. Nur der Zielknoten hat den G-Wert Null. Weiterhin existiert ein FIFO-Speicher in dem Referenzen auf Knoten gespeichert werden. Anfangs enthält der FIFO-Speicher nur die Referenz auf den Zielknoten.

- Im ersten Schritt des Algorithmus wird dem FIFO-Speicher eine Referenz auf einen Knoten K entnommen. Wenn der Speicher leer sein sollte, terminiert der Algorithmus.

- Im zweiten Schritt werden die G-Werte der Nachbarknoten von K mit der Summe aus dem G-Wert von K und der Gewichtung der Kante, die in den Nachbarknoten führt, aktualisiert, falls diese Summe kleiner ist als der schon vorhandene G-Wert.

- Im dritten Schritt werden Referenzen auf die im zweiten Schritt aktualisierten Knoten in den FIFO-Speicher gegeben. Anschließend wird beim ersten Schritt wieder angefangen.

Der Algorithmus terminiert, da bei jeder Aktualisierung eines G-Wertes der Wert nur echt kleiner werden kann. Die G-Werte sind aber nach unten beschränkt, da die kleinste Summe aus Kantengewichtung und G-Werten größer als Null sein muß. Wenn keine G-Werte mehr aktualisiert werden, so leert sich der FIFO-Speicher und der Algorithmus terminiert.

Nach der Termination bilden die G-Werte der Karte ein Potentialfeld, dessen Minimum gerade im Zielpunkt ist. Um optimale Pfade zum Zielpunkt zu finden, genügt es, der Greedy-Strategie zu folgen (siehe Bild 8.3). D.h., man erzeugt von einem beliebigen Startpunkt aus einen optimalen Pfad zum Ziel, durch Verlängerung des schon vorhandenen Pfadanfanges um denjenigen Nachbarpunkt des Pfadendes mit der niedrigsten Summe aus G-Wert und Kosten für den Schritt zu diesem Nachbarpunkt.

3.8	2.8	2.4	2	2.4	2.8
3.4	2.4	1.4	1	1.4	2.4
3	2	1	0	1	2
3.4	2.4	1.4	1	1.4	2.4
3.8	2.8	2.4	2	2.4	2.8

Abbildung 8.3: Abstieg in der mit G-Werten belegten Karte

Die Zwischenziele für den Piloten können nach Aufbau des Potentialfeldes mit sehr geringem Rechenaufwand gewonnen werden. Dabei ist es belanglos, ob der Pilot dem vom Navigator geplanten Pfad exakt folgt oder davon abweicht. Durch das Potentialfeld sind implizit optimale Pfade ausgehend von allen Positionen in der Karte definiert.

In der Bilderserie 8.4, 8.5, 8.6 wird ein Beispielablauf des Algorithmus skizziert. Die 'Flutung' der Karte startet im Zielpunkt im Druckerraum links unten in der Karte. Mit der Zeit wird die gesamte Karte mit G-Werten belegt, bis schließlich mit Bild 8.6 das endgültige Potentialfeld vorhanden ist.

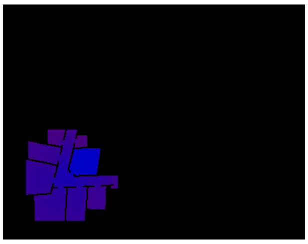

Abbildung 8.4:

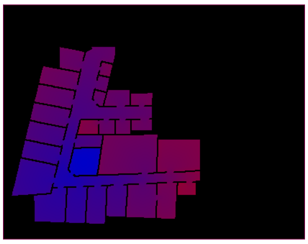

Abbildung 8.5:

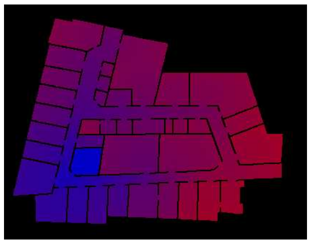

Abbildung 8.6:

8.2 Anforderungen an die Odometriekorrektur

Da der Navigator keine Sensorinformationen über seine Umgebung erhält, ist er darauf angewiesen, daß die odometrische Position des Roboters passend zur Karte korrigiert wird. Falls die vom Fahrwerk gelieferte Position verschoben ist und diese nicht korrigiert wird, sind die vom Navigator gesetzten Zwischenziele ebenfalls um denselben Betrag verschoben. Im Bild 8.7 kann man erkennen, welche Auswirkungen dies auf die Navigation haben kann.

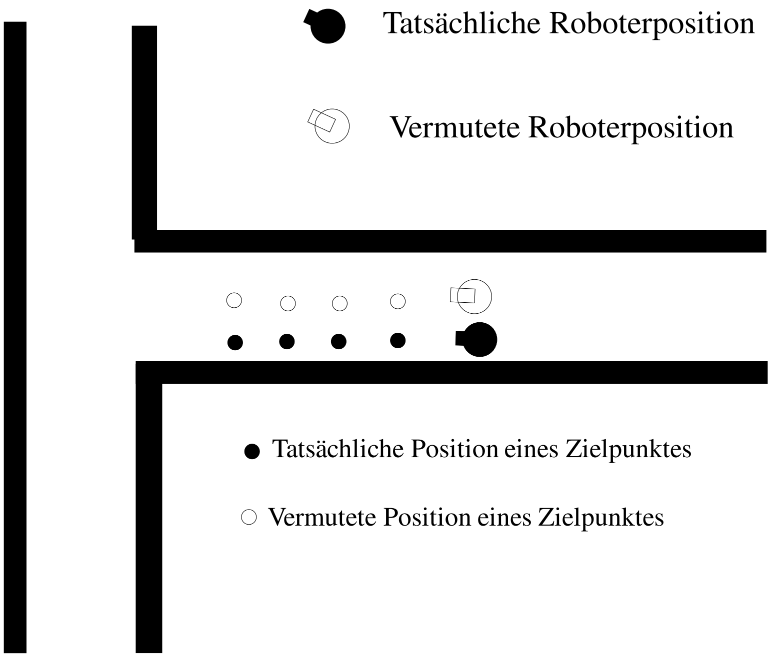

Abbildung 8.7: Navigation mit verschobener Lokalisation

Zwischenziele, die in der globalen Karte mitten in einem Gang liegen, können in der Realität dicht bei Wänden oder sogar hinter Wänden liegen! Erkennbare Abweichungen, wie beispielsweise eine Abweichung quer zu einem Gang, sollten deshalb auf jeden Fall korrigiert werden, da sonst unsinnige Zwischenziele erzeugt werden, die Kollisionen provozieren.

Eine weitere Fehlerquelle kann eine geringfügig falsche Skalierung der globalen Karte sein. Selbst wenn das Fahrwerk die zurückgelegten Strecken korrekt mißt, kommt es allmählich zu einer Differenz zwischen der tatsächlichen und der angenommenen Position. Gerade an Abzweigungen, muß an Abzweigungen in ansonsten merkmalslosen Gängen die Odometriekorrektur schlagartig eine große Abweichung kompensieren.

Ohne die Fähigkeit zur sofortigen Korrektur solcher Abweichungen steuert der Navigator zielstrebig nicht vorhandene Gangöffnungen an.

Die Lösung zu diesen Problemen findet man in der Diplomarbeit von Timm Haase "Driftkompensation auf Basis von Besetztheitsgittern".

Kapitel 9

Der Pilot

Der Planungs- und Kommandozyklus des Piloten beginnt mit dem Empfang einer Integratorkarte. Auf dieser Karte hat der Pilot die Aufgabe, einen vom Navigator festgesetzten Zielpunkt innerhalb der Integratorkarte anzusteuern. Agieren kann der Pilot nur indirekt durch den Fahrer. Um den Zielpunkt zu erreichen, gibt der Pilot dem Fahrer Zielpunktpaare so vor, daß der Roboter den Navigatorzielpunkt möglichst schnell und ohne Kollisionen erreicht.

Der Pilot hat also im Gegensatz zum Navigator eine Karte der tatsächlichen Umgebung zur Verfügung. Der Navigator plant nur global auf der Grundlage der vermuteten Position und der, im Vergleich zur realen Lage, vereinfachten globalen Karte. Der Pilot hat die Aufgabe, die Planung des Navigators an die tatsächlich vorhandenen Hindernisse anzupassen bzw. geringfügige Fehler zu korrigieren.

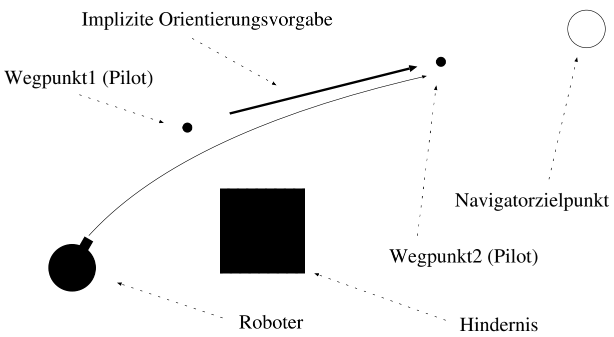

Abbildung 9.1: Der Pilot gibt dem Fahrer zwei Wegpunkte vor, um den Navigatorzielpunkt zu erreichen

Die Zielpunktpaare geben dem Fahrer nicht nur vor, wohin er fahren soll, sondern auch eingeschränkt in welcher Orientierung. Ausführliche Erläuterungen zu den Zielpunktpaaren sind im folgenden Fahrerkapitel zu finden.

Das zügige Erreichen des Zielpunktes und die unbedingte Kollisionsvermeidung sind konkurrierende Zielvorgaben. Da Pfade, die ein zügiges Erreichen des Ziels sicherstellen, in Bezug auf Kollisionen häufig risikoreicher sind als längere/langsamere Pfade, ist es i.a. nicht möglich Lösungen zu finden, die bezüglich aller Teilziele optimal sind.

Es ist notwendig, ein quantitatives Kostenmaß für die Güte eines Pfades festzulegen, um einen sinnvollen Kompromiß zu ermöglichen. Ein solches Kostenmaß sollte in Bezug auf einen betrachteten Pfad die Zeit bis zum Erreichen des Ziels mit dem Kollisionsrisiko abwägen. Da auf den zur Verfügung stehenden Rechenanlagen die Berücksichtigung der Geschwindigkeit bei der Pfadplanung leider zu nicht akzeptablen Planungszeiten führt und das Kollisionsrisiko, wegen ungenauer Sensoren ohnehin nicht exakt zu bestimmen ist, wird nur eine stark vereinfachte Situation betrachtet. Das Kollisionsrisiko wird implizit durch Betrachtung der Entfernung zum nächsten Hindernis berücksichtigt, und die Zeit bis zum Erreichen des Zielpunktes durch die Pfadlänge ersetzt. Bei einigen Ansätzen werden zusätzliche Kosten für Richtungsänderungen des Vehikels berücksichtigt.

Im folgenden wird eine Reihe von alternativen Implementationen des Piloten vorgestellt. Dabei wird der ursprüngliche Optimalitätsbegriff in dem Bestreben, die knappe Zeitbeschränkung einhalten zu können, teilweise erheblich aufgeweicht. Da die Karte, in der die Planung stattfindet, eine Auflösung von 100 mm hat und der Roboter mit Geschwindigkeiten von 0 bis ca. 600 mm/sec fährt, wird als maximal akzeptable Planungszeit 1 Sekunde festgesetzt. In dieser Zeit darf der Roboter etwa 1-4 Pixel auf der Karte zurücklegen. Wenn der Roboter eine Aktion ausgeführt hat, muß der Planungsprozeß für die nächste Aktion bereits abgeschlossen sein.

9.1 Graphverfahren

Die Graphverfahren fassen den zu generierenden Pfad als Pfad in einem gewichteten Graphen auf, wobei die Knoten die Positionen und Orientierungen sind, die der Roboter auf der Karte einnehmen kann. Die Kanten des Graphens sind die Aktionen, die von einem Knoten aus ausführbar sind, ohne mit einem Hindernis zu kollidieren. Genauer entspricht einer Kante des Graphen ein 'Curve' Befehl des virtuellen Roboters. Um die Anzahl der Knoten nicht ausufern zu lassen, werden die Positionen auf eine Auflösung von 10 cm und die Orientierungen auf eine Auflösung von $10°$ diskretisiert.

Ein Pfad auf der Karte entspricht also einem Pfad im Aktionsgraphen. Das Optimierungsproblem in der Karte ist nun die Suche nach einem kostengünstigsten Pfad in einem mit Kosten belegten gerichteten Graphen. Die Art und Weise, wie diese Kosten definiert werden, ist unterschiedlich, aber alle diese Verfahren haben gemeinsam, daß zusätzlich zur lokalen Karte das Vorhandensein einer Entfernungs-

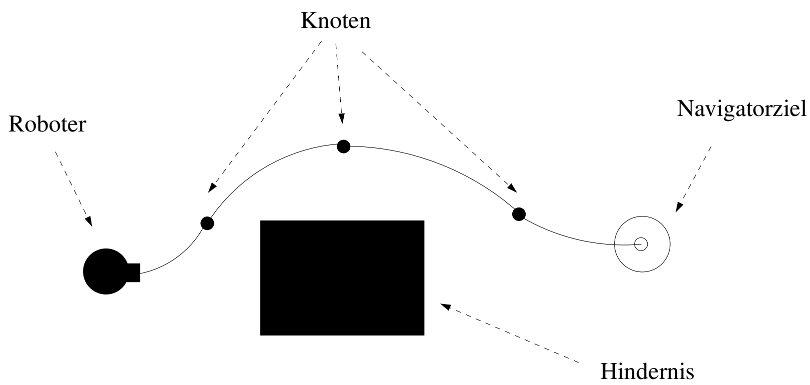

Abbildung 9.2: Verkettete Curve-Befehle erzeugen einen Pfad zum Zielpunkt

karte vorausgesetzt wird, die für jedes unbesetzte Pixel der lokalen Karte angibt, in welcher euklidischen Entfernung sich das nächstgelegene Hindernis befindet. Die zusätzliche Entfernungskarte ist eine wesentliche Grundlage für alle Kostenfunktionen, mit denen der Graph belegt wird. Sie wird im folgenden die zur lokalen Karte gehörige Entfernungskarte genannt. Der Algorithmus zur Erzeugung der Entfernungskarte wird im Anhang A.1 beschrieben.

Möglichkeiten zur Belegung der Kanten mit Kosten sind z.B.:

- Entfernungskosten[1]
- Geschwindigkeitskosten
- Wegkosten
- Zeitkosten

Die Entfernungskosten repräsentieren das Risiko des Roboters, eher mit einem Hindernis zu kollidieren, wenn die Entfernung zum nächsten Hindernis gering ist. Die Funktion, die Hindernisentfernungen in Kosten bzw. Kollisionsrisiko umrechnet, hat dabei ihr Maximum bei der Entfernung 0 (sichere Kollision) und fällt bis zu einer bestimmten Schwellentfernung ab, ab der eine Kollision nur noch eine theoretische Möglichkeit darstellt.

Da Entfernungskostenfunktionen auch an anderer Stelle der Arbeit benötigt werden, werden sie im Detail gesondert im Anhang A.2 behandelt.

Die Geschwindigkeitskosten modellieren das große Kollisionsrisiko bei hohen Geschwindigkeiten. Kollisionen mit niedrigen Geschwindigkeiten sind weniger zerstörerisch. Zudem werden die Ultraschallsensoren bei hohen Geschwindigkeiten

[1]Entfernung zum nächsten Hindernis

unzuverlässig, und auch der Latenzeffekt bei der Kommunikation zwischen den Modulen stört die Hindernisvermeidung wirkungsvoller.

Über die Wegkosten wird im Planungsprozeß darauf hingewirkt, daß möglichst kurze Pfade geplant werden. Eng damit hängen die Zeitkosten zusammen. Wenn schnelles Ankommen wichtiger ist als kurze Wege, so sollte man die Wegkosten durch Zeitkosten ersetzen, die die Zeit bis zum Erreichen des Ziels mit Kosten belegen.

Die konkrete Definition dieser Kostenfunktionen ist ein sehr heuristischer Vorgang. Der Hauptgrund dafür ist die Schwierigkeit, das Kollisionsrisiko quantitativ zu bestimmen. Ungenaue Karten und die Ungewißheit, ob die Fahrbefehle, beispielsweise durch Schlupf der Antriebsräder, exakt so wie geplant ausgeführt werden, entziehen scheinbar exakten Berechnungen jeden realen Bezug. Hinzu kommt, daß u.a. das akzeptierten Kollisionsrisiko von der persönlichen Meinung des Entwicklers abhängt. Absolute Maßstäbe für die Qualität einer Kostenfunktion existieren nicht.

9.1.1 A^* Suche

In diesem Abschnitt wird untersucht, ob der A^*-Algorithmus geeignet ist, das obig beschriebene Optimierungsproblem zufriedenstellend zu lösen, d.h. in der Praxis unter Einhaltung der Zeitbeschränkungen einen kollisionfreien und möglichst kurzen Pfad zum Zielpunkt zu planen.

Der A^* Algorithmus

Sei $G(V, E)$ ein gewichteter Graph mit Zielknoten z und Startknoten s. Sei $c : E \to \mathbb{R}_+$ diejenige Funktion, die die Kanten gewichtet. Sei weiter $h : E \to \mathbb{R}_+$ eine sog. **optimistische Schätzfunktion**, die für jeden Knoten des Graphen eine Schätzung der minimalen Kosten bis zum Zielknoten abgibt, die kleiner oder gleich den tatsächlichen Kosten ist. In diesem Zusammenhang heißt eine Schätzfunktion h_1 **besser informiert** als h_2, wenn $h_1 \geq h_2$ gilt.

Der A^*-Algorithmus ([22]) hat das Ziel, den kostengünstigsten Pfad vom Startknoten s zum Zielknoten z zu finden. Dabei wird folgendermaßen vorgegangen.

1. Der Startknoten wird in eine noch leere Prioritätsqueue mit dem Prioritätswert $p(s) := h(s)$ eingefügt.

2. Das Element e mit dem niedrigsten Prioritätswert $p_{min} = p(e)$ wird aus der Queue entnommen und alle n Nachbarknoten $k_i, i = 1, \ldots, n$ von e betrachtet. Falls ein Knoten k_i noch nicht in der Queue enthalten ist, so wird er mit der Priorität $p(k_i) := p(e) + c([e, k_i]) + h(k_i)$ in die Queue eingefügt. Falls der Knoten schon in der Queue enthalten ist und es gilt $p(k_i) \leq$

$p(e) + c([e, k_i]) + h(k_i)$, so wird die Queue nicht verändert, ansonsten wird die Priorität von k_i auf $p(e) + c([e, k_i]) + h(k_i)$ geändert.

3. Falls ein Knoten im letzten Schritt eingefügt oder in seiner Priorität verändert wurde, merke man sich für diesen Knoten seinen Vorgänger e.

4. Falls eines der k_i schon der Zielknoten war, stoppe die Ausführung. Andernfalls mache bei Punkt 2 weiter.

Es wurde bewiesen, daß der erste Pfad, der von diesem Algorithmus zum Zielknoten gefunden wird, auch schon der kostengünstigste ist. Das heißt i.a. aber nicht, daß der A^*-Algorithmus den kostengünstigsten Pfad auch mit dem geringstmöglichen Aufwand findet !

Anwendung des A^*-Algorithmus zur Pfadplanung

Mit der in der Einleitung dieses Abschnitts vorgenommenen Umformulierung des Wegplanungsproblems ist der A^*-Algorithmus direkt anwendbar. Als Kanten, die von einem Knoten wegführen, werden in einer Tabelle aufgelistete Curve-Befehle verwendet, wobei für jeden Befehl direkt die Befehlskosten angegeben werden, d.h. wieviel die Ausführung des Befehls kostet. In diesen Befehlskosten werden die von der Umgebung unabhängigen Anteile der Geschwindigkeits- , Weg- und Zeitkosten zusammengefaßt. Um die Kosten einer Kante im Graphen endgültig zu bewerten, werden zu den Befehlskosten noch die Entfernungskosten des Zielpunktes des Curve-Befehls hinzuaddiert.

Als Grundlage für die Schätzfunktion dient der euklidische Abstand vom aktuellen Knoten zum Zielpunkt. Durch passende Definition von Befehls- und Entfernungskosten wird garantiert, daß pro Entfernungseinheit die tatsächlichen Kosten größer sind als die geschätzten. D.h. durch die geeignete Definition der Kosten wird erzwungen, daß die Schätzfunktion optimistisch ist.

Experimentelle Ergebnisse

Das Ergebnis einer (isoliert) simulierten Pfadplanung ist in Bild 9.3 dargestellt.

Die grünen Punkte markieren die von A^*-Algorithmus besuchten Positionen, der graue Block am rechten Bildrand ist das Hindernis und die gelben Kreuze markieren den vom Algorithmus gefundenen Pfad. Als Befehl gibt der Pilot die Parameter des Curve-Befehls, der ausgehend vom Startknoten[2] auf dem optimalen Pfad liegt.

Der Pfad ist nach menschlicher Betrachtungsweise durchaus brauchbar, um kollisionsfrei zum Zielpunkt zu fahren. Was man nicht in der Grafik erkennt ist, daß ein

[2] Der Startknoten entspricht der aktuellen Position des Roboters

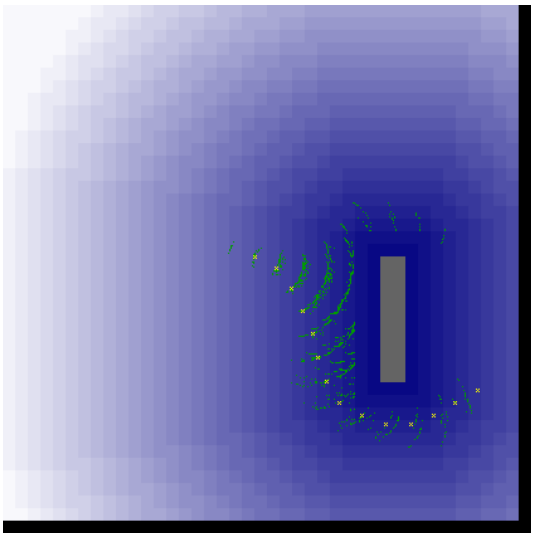

Abbildung 9.3: Pfadplanung mit dem A*-Alg

Rechner mit 90 MHz Taktfrequenz 1.5 Sekunden zu Planung dieses Pfades benötigte! Die Situation in der realen Anwendung ist noch viel schlimmer, da dann die Rechenleistung eines Computers auf bis zu zehn Module aufgeteilt wird. Wenn mehr als eine Sekunde verstreicht bis die Planung vollständig ist, ist der Roboter normalerweise schon so weit von der der Planung zugrundegelegten Position entfernt, daß die ausgegebenen Fahrbefehle kein sinnvolles Verhalten mehr erzeugen.

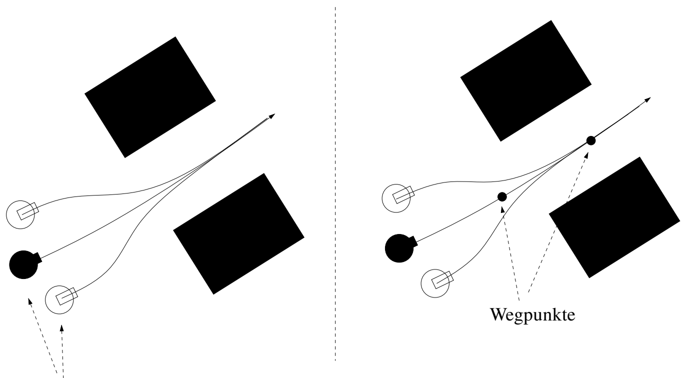

Abbildung 9.4: Pfadplanung auf der Grundlage von Entfernungskosten und auf der Grundlage von Wegpunkten

Als etwas robustere Alternative zur direkten Weitergabe eines Curve-Befehls an den Fahrer wurde erprobt, den durch den A^*-Algorithmus gefundenen optimalen Pfad durch zwei Wegpunkte annähernd zu beschreiben. In Bild 9.4 sind auf der linken Seite die Pfade eingezeichnet, die ausgehend von drei leicht unterschiedlichen Positionen, auf der Grundlage der Verkettung von Curve-Befehlen gebildet wurden. Auf der rechten Seite des Bildes sind die entsprechenden Pfade dargestellt, die entstehen, wenn man zwei Punkte auf einem Pfad[3] des linken Bildes als Wegpunkte benutzt. Es entstehen wieder Pfade, die denen auf der linken Seite stark ähneln.

Man kann also die explizite Handlungsanweisung an den Fahrer - den Curve-Befehl - durch die implizite Anweisung, einer Folge von Wegpunkten zu folgen, ersetzen. Dem Fahrer wird mehr Verantwortung übertragen, da er nicht mehr nur passiv durch Curve-Befehle gesteuert[4] wird, sondern aktiv den Wegpunkten folgt.

In Bild 9.4 erkennt man auch, daß Pfade, die an fast identischen Positionen beginnen, bei der Pfadplanung mit Curve-Befehlen die Tendenz haben, zu einem einzigen Pfad zu konvergieren.

Diese Pfadkonvergenz ist hauptsächlich auf den großen Einfluß der Wandentfernung auf die Pfadplanung zurückzuführen. In der Regel sind Pfade, die 'optimal' sind in Bezug auf die Position größerer Hindernisse, auch schon diejenigen Pfade, auf denen man risikoarm schnell fahren kann, ohne mit diesen Hindernissen zu kollidieren. In Umgebungen, in denen der befahrbare Raum von allen Seiten eng mit Wänden und Hindernissen umgeben ist, läuft jede vernünftige Kostenfunktion auf die Strategie hinaus:"Möglichst allen Hindernissen fernbleiben und so schnell auf das Ziel zufahren wie es gerade noch vertretbar ist".

Genau diese Aufgabe wird dem Fahrer mit den Wegpunkten indirekt übertragen. Der Fahrer regelt den Roboter beim Wegpunktverfolgen auf der Grundlage der aktuellen Position und verfolgt so aktiv einen Pfad, der Hindernissen fernbleibt. Bei kleinen Abweichungen beim Abfahren des vorgegebenen Pfades können die Wegpunkte beibehalten werden, da die Pfade mit geringfügig unterschiedlichen Startpunkten auch bei völliger Neuplanung konvergieren. Die Wegpunkte stabilisieren also das Abfahren des vorgegebenen Pfades ohne die Optimalität der Steuerung zu beeinträchtigen.

Durch die Einführung von Wegpunkten als Befehle für den Fahrer ist der Nachteil der langen Planungszeit des A^*-Algorithmus etwas entschärft, da nicht bei jeder geringfügigen Abweichung vom anfänglich geplanten Pfad die Planung wiederholt werden muß. Ein einmal durch den A^*-Algorithmus geplanter Pfad wird durch die wiederholte Vorgabe von Wegpunkten auf diesem Pfad fehlertolerant abgefahren. Dennoch bleiben wegen der langen Planungszeiten, besonders in Situationen, bei denen häufig die Richtung gewechselt wird, Kollisionen nicht aus.

[3] Ein Pfad der durch den A -Algorithmus gefunden wurde

[4] Mit Ausnahme der Auslösung des Nothalts

9.1.2 Tiefensuche

Der Schwachpunkt des A* Algorithmus in Zusammenhang mit verketteten Curve-Befehlen ist vor allem die zu aufwendige Suche im Aktionsgraphen.

Ein andere Methode ist die Tiefensuche im Aktionsgraphen. Da der Suchaufwand gering gehalten werden muß, ist es nicht möglich, eine volle Tiefensuche durchzuführen. Es ist notwendig, die Anzahl der Nachfolger eines Knotens zu begrenzen und, wie bei der A* Suche, durch eine Schätzfunktion zu ermitteln, für welche Zweige eine weitere Suche keine Aussicht auf Erfolg hat.

Als Schätzfunktion wurde das Ergebnis der Anwendung des Potentialfeldansatzes aus dem Navigatormodul auf die Integratorkarte verwendet. Es stellt sich allerdings heraus, daß auf diese Weise gewonnene Pfade praktisch identisch mit denjenigen Pfaden sind, die durch direkten Abstieg in der Kostenkarte entstehen.

9.1.3 Vor und Nachteile der graphbasierten Verfahren

Der größte Vorteil der graphbasierten Verfahren ist, daß mit sehr gut verstandenen und dokumentierten Algorithmen gearbeitet wird. Tiefen- oder Breitensuche in Graphen sind Standardverfahren. Leider zeigt sich in der Praxis, daß diese Verfahren sehr anfällig gegen geringfügige Störungen sind, sei es in den der Entscheidung zugrundeliegenden Odometriewerten oder in noch stärkerem Maße in der ungenauen Ausführung der erzeugten Befehle. Wesentliches Element der graphbasierten Wegplanung ist die exakte Verkettung von Curve-Befehlen. Unbedingte Voraussetzung für das Funktionieren dieser Ansätze ist daher absolute Präzision, die ein realer Roboter nicht liefern kann. Die Einführung von Wegpunkten als Fahrerbefehle stabilisiert zwar das Verhalten des Roboters, kann aber den langen Planungszeiten nichts ändern. Hinzu kommt der Aspekt, daß der Aufwand, einen Pfad detailliert mit Geschwindigkeiten, Orientierungen und wohlmöglich den Auswirkungen der Massenträgheit zu planen, mit dem Ergebnis von zwei Wegpunkten etwas übertrieben wirkt. Die Erfahrung, daß die Schätzfunktion bei der Tiefensuche allein schon Pfade erzeugt, die denen bei der Graphsuche stark ähneln, führt zum Gradientabstiegsverfahren.

9.2 Das Gradientabstiegsverfahren

Beim Gradientabstiegsverfahren wird auf der Karte eine Funktion definiert, die ihr Minimum im Zielpunkt hat. Die Funktion ist so konstruiert, daß es von jeder Position der Karte aus genügt, dem Weg des steilsten Abstiegs zu folgen, um kollisionsfrei zu diesem Zielpunkt zu gelangen. Die Handlungsanweisung für den untergeordneten Fahrer besteht aus zwei Wegpunkten auf dem Abstiegspfad. Der unmittelbare

Vorteil der Methode besteht darin, daß es bei unveränderter Karte, aber veränderter Position des Roboters, nicht notwendig ist, die Abstiegsfunktion neu zu erzeugen. Weil der Gradientabstieg selbst nur sehr wenig Rechenleistung erfordert, kann der Pilot während der Wartezeit bis zum Empfang der nächsten Karte den Fahrer mehrmals aktualisierte Wegpunktpaare vorgeben.

Es wird prinzipiell derselbe Algorithmus verwendet wie beim Flood-Fill im Navigator. Zunächst wird aus der Besetztheitskarte eine Entfernungskarte[5] erzeugt, in der für jedes Kartenfeld die Entfernung zum nächstgelegenen Hindernis gespeichert ist. Diese Karte wird dann mit der Entfernungskostenfunktion[6] in eine Kostenkarte umgewandelt, in der in jedem Kartenfeld die Kosten für den Aufenthalt auf dieser Position verzeichnet ist. Diese Kostenkarte dient dann als Grundlage für den im Abschnitt 8.1.1 beschriebenen Algorithmus, der eine Kostenkarte erzeugt, in der in jedem Kartenfeld jeweils die minimalen Kosten für die Bewegung zum Navigatorzielpunkt, ausgehend von aktuellen Feld, eingetragen sind.

Wenn eine solche Karte gegeben ist, so reduziert sich die Aufgabe, den bezüglich der definierten Kosten optimalen Pfad zu finden, darauf, sich vom jeweiligen Standort aus jeweils zu dem Nachbarfeld zu bewegen, das die geringsten Feld-Kosten aufweist.

3.8	2.8	2.4	2	2.4	2.8
3.4	2.4	1.4	1	1.4	2.4
3	2	1	0	1	2
3.4	2.4	1.4	1	1.4	2.4
3.8	2.8	2.4	2	2.4	2.8

Abbildung 9.5: Abstieg in der Kostenkarte

Dies garantiert, daß dem optimalen Pfad gefolgt wird, da die Summe aus den so ausgewählten unmittelbaren Kosten zum Feldwechsel zusammen mit den im Zielfeld verzeichneten Kosten bis zum Ziel gerade die minimal erreichbaren Kosten vom aktuellen Feld bis zum Ziel sind. Die Auswahl desjenigen Nachbarfeldes mit der geringsten Summe ist also gleichbedeutend mit der Auswahl der global geringsten Pfadkosten.

5 siehe Anhang A.1

6 siehe Anhang A.2

9.3 Verschieben des Navigatorzielpunktes

Wegen der in Abschnitt 8.2 beschriebenen Probleme, insbesondere der in Bild 8.7 dargestellten, wird der Navigatorzielpunkt vom Piloten modifiziert. Das Ziel der Modifikation ist es, den Zielpunkt in der Integratorkarte so zu verschieben, daß er in der tatsächlichen Gangmitte und nicht nur in der angenommenen liegt. Hierzu wird in der unmittelbaren Umgebung der momentanen Zielpunktposition nach Kartenfeldern gesucht, die eine größere minimale Entfernung zu Hindernissen haben. Genauer wird der Zielpunkt mehrmals zu dem Nachbarfeld verschoben, das den größten Entfernungswert aufweist.

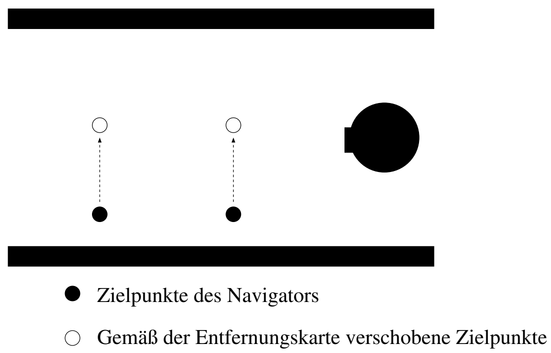

Abbildung 9.6: Anpassen von Zielpunkten

9.4 Varianten bei der Verwendung der Integratorkarten

Eine Fehlerquelle, die bisher ausgeklammert wurde, sind die fast immer unvollständigen oder sogar fehlerhaften Integratorkarten. Durch die geringe Winkelauflösung der Ultraschallsensoren bedingt, werden Objekte, die gerade erst in Sichtweite der Sensoren geraten sind, nur grob abgebildet. Türöffnungen werden - wenn überhaupt - nur aus allernächster Nähe erkannt, oder werden genau wie enge Gänge durch Scheinhindernisse verbaut. Erst wenn der Roboter schon an einem Hindernis vorbeigefahren ist, kann man erwarten, daß es einigermaßen korrekt in die Karte eingezeichnet wird. Das andere Extrem kommt ebenfalls vor. Glatte Wände und besonders geschlossene Türen werden nur mit Glück erkannt, da ein steiler Einfallswinkel der Ultraschallimpulse nicht immer gewährleistet ist.

In diesem Abschnitt wird beschrieben, wie diese leider unvermeidbaren Effekte die Wegplanung beeinflussen, und es werden Ansätze vorgestellt, um ihre negativen Auswirkungen zu unterdrücken.

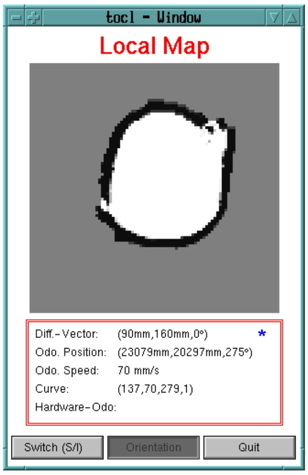

Abbildung 9.7:

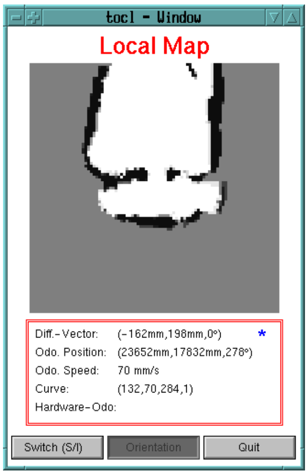

Abbildung 9.8:

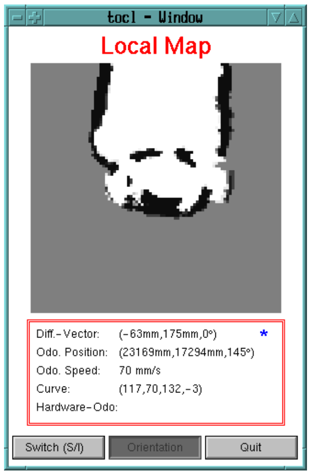

Abbildung 9.9:

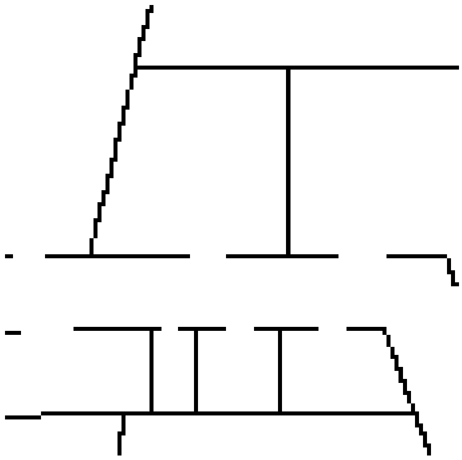

Abbildung 9.10: Tatsächliche Umgebung

9.4.1 Alleinige Verwendung der Integratorkarte

Ursprüngliches Ziel war es, beim Piloten ausschließlich die Integratorkarten als Grundlage für die Planung heranzuziehen. Die Idee hinter diesem Ziel ist, im Verantwortungsbereich des Piloten nur auf der Grundlage der realen Umgebung zu planen und damit von den evtl. vorhandenen Ungenauigkeiten der vorgegebenen globalen Karte unabhängig zu werden.

Scheinhindernisse

Häufig tritt der Effekt auf, daß Felder in der Integratorkarte als 'besetzt' gekennzeichnet werden, die in der Realität befahrbar sind. Besonders störend tritt dieser Effekt bei Türen auf. Dies ist in der Abbildung 9.7 zu erkennen. Wenn der Navigatorzielpunkt hinter die tatsächlich offene Tür gesetzt wird, so gibt es normalerweise keine alternative Route zu diesem Ziel. Wegen der geringen Winkelauflösung der Ultraschallsensoren hat der Kartenbildner nur dann überhaupt eine Chance, die Türöffnung zu entdecken, wenn der Roboter in die unmittelbare Nähe der Öffnung fährt, wie es in Bild 9.8 zu sehen ist. Da der Roboter in der Situation des Bildes 9.7 ziellos umherfährt, wird als Abhilfe dem Piloten nicht nur ein Ziel sondern gleich mehrere Ziele zur Auswahl gegeben. Wie in Abbildung 9.11 dargestellt, liegen alle auf Zielpunkte dem Pfad, den der Navigator nur in Kenntnis der globalen Karte geplant hat. Der Pilot fängt mit dem, entlang dem Pfad, entferntesten Zielpunkt an und wendet den lokalen Pfadplanungsalgorithmus versuchsweise an. Wenn kein Pfad zum Ziel gefunden wird, so wird der nächstnähere Zielpunkt ausprobiert.

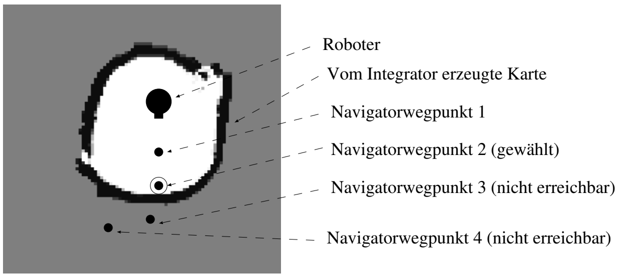

Abbildung 9.11: Auswahl eines Navigatorzielpunktes

Der vorletzte Zielpunkt liegt in einer so geringen Entfernung vom Roboter, daß er garantiert noch vor einer fälschlicherweise als geschlossen angezeigten Tür liegt. An diesem Punkt angekommen hat der Kartenbildner dann die Chance, die Türöffnung zu detektieren und der Roboter kann gesteuert vom Piloten den Raum verlassen bzw. betreten.

Unsichtbare Hindernisse

Schon an anderer Stelle wurde der Effekt der Totalreflexion des Ultraschallimpulses ohne meßbares Echo beschrieben. Die Folge dieses Effektes ist z.B. als Scheinöffnung in einer sonst korrekt eingezeichneten Wand sichtbar[7].

Nur auf Grundlage der Integratorkarte ist es leider prinzipiell unmöglich, unsichtbaren Hindernissen auszuweichen. Sie sind wie der Name schon aussagt 'nicht

[7] siehe Bild 3.7 in Abschnitt 3.2

wahrnehmbar'. Um solche Hindernisse zu vermeiden, sind entweder zusätzliche Sensoren oder die Verwendung von Hintergrundwissen erforderlich.

9.4.2 Kombination von globaler und Integratorkarte

Da das Problem der unsichtbaren Hindernisse mit der Integratorkarte allein nicht zu lösen ist, wird hier der Ansatz verfolgt, die Integratorkarte durch Informationen aus der globalen Karte zu ergänzen. Die Idee ist, daß bei hinreichend genauer Lokalisation die Integratorkarte und der entsprechende Ausschnitt der globalen Karte zur Deckung kommen müßten.

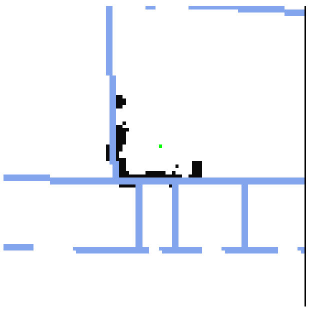

Abbildung 9.12:

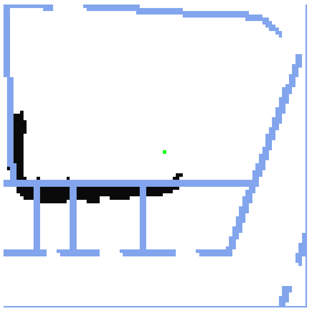

Abbildung 9.13:

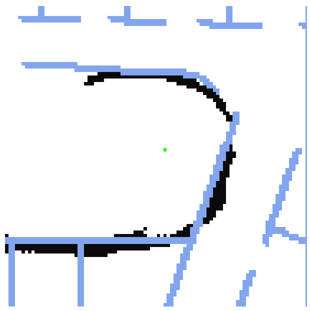

Abbildung 9.14:

Nicht wahrgenommene Wände können so ergänzt werden. Da außer unverrückbaren Wänden nichts in der globalen Karte eingetragen ist, insbesondere keine transportablen Hindernisse, sollten durch die Kombination keine neuen Scheinhindernisse entstehen. Es werden ab jetzt Karten betrachtet, die ausgehend von einer Integratorkarte um die besetzten Pixel des entsprechenden Ausschnitts der globalen Karte ergänzt werden. In der Bildserie 9.12, 9.13 und 9.14 erkennt man, wie die Ultraschallsensoren nur einen kleinen Teil der Wände detektieren und die eingeblendete globale Karte ergänzen.

In der Praxis sind leider Schwankungen in der Genauigkeit der Lokalisation nicht zu vermeiden, was u.a. auch eine Folge der ungenauen Integratorkarten ist. Die besonders kritische Stelle ist wieder die Türöffnung. Wenn die Lokalisation weit genug von der tatsächlichen Position abweicht, so kann es passieren, daß die Türöffnung so weit verschlossen wird, daß der Pilot sich weigert, einen Pfad durch die Tür zu planen, obwohl die Türöffnung in voller Größe in der Integratorkarte zu erkennen ist. Um das Dilemma aufzulösen, wird ein kreisförmiger Bereich um den Roboter von der Einblendung der besetzten Pixel der globalen Karte ausgenommen.

Auf geringe Entfernung ist der Totalreflexionseffekt erheblich seltener als die Situation der verschobenen globalen Karte, so daß es vernünftig erscheint, bei kurzen Distanzen der Integratorkarte zu vertrauen.

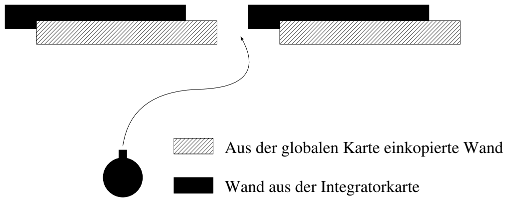

Abbildung 9.15: Problem bei ungenauer Lokalisation

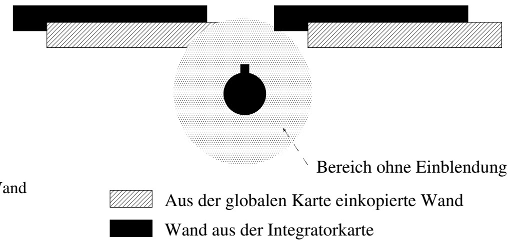

Abbildung 9.16: Lösung

9.5 Vergleich der Verfahren

Die graphbasierten Verfahren sind für die Anwendung auf der vorhandenen Hardware zu rechenintensiv. Hinzu kommt, daß durch die eingeschränkte Wahrnehmung der Umgebung schlicht an der Realität vorbeigeplant wird. Die geplanten Pfade, in denen detailliert Kosten für Geschwindigkeiten und Zeitverbrauch berücksichtigt werden, unterscheiden sich kaum von denen, die durch den Gradientabstieg erzeugt werden, in die nur die Entfernung zum jeweils nächsten Hindernis einfließt. Der erhebliche Mehraufwand bei der Graphplanung läßt sich deshalb bei Verwendung von ungenauen Ultraschallkarten nicht rechtfertigen.

Der Versuch, schon auf der Ebene des Piloten konkrete Curve-Befehle zu planen, hat sich als Irrweg herausgestellt. Die Planung von konkreten Hardwarebefehlen muß dem vielfach reaktionsfähigeren Fahrermodul überlassen bleiben. Letztlich hat sich die Gradientabstiegsmethode mit der Vorgabe von lokalen Zielpunkten für den Fahrer in Verbindung mit der im vorigen Abschnitt beschriebenen Modifikation der Integratorkarten, als praktikabelste Lösung herausgestellt.

Kapitel 10

Der Fahrer

In diesem Kapitel werden einige Verfahren zur reaktiven Hindernisvermeidung bei gleichzeitiger Berücksichtigung der Anweisungen des Piloten behandelt.

10.1 Einleitung

Der Fahrer ist das hardwarenächste Modul der Navigationssoftware[1]. Er hat die Aufgabe, Kollisionen zu verhindern und gleichzeitig den vom Piloten vorgegebenen Zielpunkten zu folgen, soweit das möglich ist. Während der Pilot nur alle ein bis zwei Sekunden ein neues Zielpunktpaar erzeugen muß bzw. kann, darf der Entscheidungszyklus des Fahrers maximal eine zehntel Sekunde betragen, um zuverlässig auch in einer dynamischen Umgebung Kollisionen zu vermeiden. Auf dieser Zeitskala ist die direkte Berücksichtigung von aufwendigen Repräsentationen der Umwelt, etwa der Integratorkarte, nicht sinnvoll und aus Kapazitätsgründen auch nicht durchführbar. Die Hindernisvermeidung basiert deshalb auf einem reaktiven Ansatz, der unmittelbar aus den Rohdaten der Ultraschallsensoren Fahrbefehle erzeugt, wobei die Berücksichtigung von längerfristigen Zielen durch die gleichzeitige Verfolgung der Pilotenzielpunkte gewährleistet wird.

Ein schwieriges Problem ist die **quantitative** Spezifikation des Fahrer-Verhaltens. Qualitative Kriterien sind dagegen sehr einfach zu formulieren. Der Roboter soll niemals mit Hindernissen kollidieren und sich trotzdem mit hoher Geschwindigkeit in Richtung Ziel fortbewegen. Der mathematisch korrekte Weg, um zu einem ”guten” Hindernisvermeider im Sinne der obigen Bedingungen zu kommen, ist sicherlich, die qualitative Zielvorstellung ”schnell zum Ziel gelangen ohne zu kollidieren” als Kostenfunktion zu formulieren und dann das damit gegebene Optimierungsproblem zu lösen. Im konkreten Fall könnte man für alle ausführbaren Aktionen das Kollisionsrisiko bestimmen und unter denjenigen Aktionen, deren Kollisionsrisiko

[1] siehe Abschnitt 6.1.1

unter einer akzeptablen Schwelle bleibt, diejenige wählen, die die schnellste Fortbewegung zum Ziel verspricht.

In der Praxis ist die deduktive Konstruktion des optimalen Verhaltens leider fast ausgeschlossen. Das Haupthindernis für die Herleitung eines optimalen Fahrers ist die Schwierigkeit, das Kollisionsrisiko auf der Grundlage der dem Fahrer zur Verfügung stehenden Daten realistisch herzuleiten. Die Aufgabe ist mit einem expliziten Modell, in einer nur unvollkommen wahrgenommenen, dynamischen Umgebung, kaum erfaßbar. Auf rein statistischem Weg zum Ziel zu kommen, ist ebenfalls aussichtslos, da reale Versuche mit der Hardware sehr aufwendig sind und für eine aussagekräftige Statistik erheblich mehr Versuchen notwendig sind, als mit vertretbarem Aufwand durchführbar sind. Das Problem kann durch Simulation des Robotersystems zwar entschärft werden, allerdings kann auf reale Versuche nicht völlig verzichtet werden. Ansonsten ist derselbe unvertretbar große Aufwand bei der Kalibrierung des Simulators notwendig.

Hinzu kommt die Echtzeitanforderung. Selbst wenn man das Vorhandensein eines wahrscheinlichkeitstheoretischen Modells voraussetzt, so wäre es mit Sicherheit so komplex, daß die Forderung an die Echtzeitfähigkeit des Algorithmus nicht mehr erfüllbar ist. Ein weiteres Argument gegen das deduktive Vorgehen bei dieser Problemstellung ist die Fülle an Parametern, die nicht oder nur unter sehr großem Aufwand direkt meßbar sind. Die Grundlage für einen "optimalen" Fahrer wären ungenaue Daten und grob vereinfachte Zusammenhänge, die im Ergebnis eine Optimierung an der Realität vorbei bewirken.

Herleitungen aus expliziten Modellierungen des Problems sind also nicht geeignet, einen Fahreralgorithmus zu konstruieren. Eine andere Möglichkeit ist, bereits vorhandene Problemlösungskompetenz in den Fahrer einzubauen. Im konkreten Einzelfall kann ein menschlicher Beobachter sehr gut beurteilen, ob ein Verhalten des Fahrers der jeweiligen Situation angemessen ist oder nicht. Es werden zwei Ansätze betrachtet, die dieses nicht explizit vorhandene Wissen vom Menschen auf das Robotersystem übertragen.

Im ersten Ansatz wird das vorhandene Wissen durch *Fuzzy Logik* ausgedrückt. Im zweiten Ansatz werden für Teilmengen der Sensoren jeweils Experten bereitgestellt, die aus ihrer lokalen Sicht heraus Anforderungen an die Steuerungsgrößen[2] stellen. Alle Anforderungen zusammen werden dann zu den endgültigen Steuerungsgrößen zusammengesetzt.

Beide Ansätze benötigen als Eingabe nur die Zielorientierung und die aktuellen Ultraschallentfernungswerte und geben eine Rotations- und eine Translationsgeschwindigkeit aus. Da der Fahrer streng genommen nur mit Curve-Befehlen mit dem Hardware-Interface kommunizieren darf, verbleibt bei diesen Ansätzen nur die Umwandlung der Zielpunktpaare in Zielorientierungen im Fahrermodul. Die Hin-

[2] Rotations- und Translationsgeschwindigkeit

dernisvermeidung wird ins Hardwareinterface verlagert, wo direkt die Rotations- und Translationsgeschwindigkeit beeinflußt werden kann. Das hat u.a. den Vorteil einer geringeren Latenzzeit bei der Reaktion auf überraschende Hindernisse. In der Endversion des Programms wurde schließlich auch dar Fahrerprozeß in den Hardwareinterfaceprozeß integriert[3].

10.2 Berechnung der gewünschten Orientierung

Die folgenden reaktiven Hindernisvermeider benötigen als Eingabe neben den aktuellen Entfernungswerten die Zielorientierung des Roboters. Als Anweisung erhält der Fahrer aber ein Wegpunktpaar.

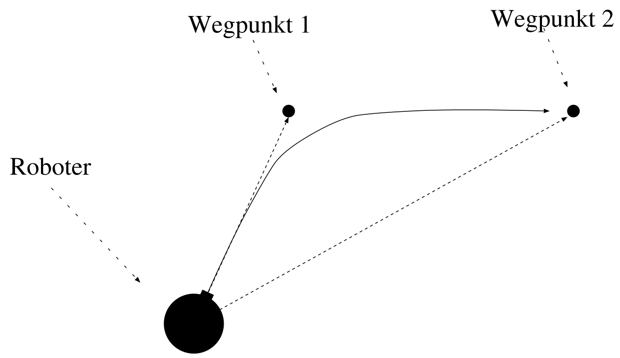

Abbildung 10.1: Umwandlung der Wegpunktpaares in eine Zielorientierung

In Bild 10.1 ist die Problemstellung dargestellt. Die naheliegenste Lösung ist das direkte Anfahren des ersten Wegpunktes und anschließend das direkte Anfahren des zweiten Wegpunktes. Diese naheliegende Vorgehensweise ist im realen Betrieb mit erheblichen Schwierigkeiten verbunden.

Der Roboter ist nicht in der Lage, mathematisch exakt den Kurs auf einen Zielpunkt zu halten. Durch ungleichen Schlupf der Antriebsräder, Schwankungen bei der Selbstlokalisation und verzögerte Ausführung von Fahrbefehlen sind Abweichungen von wenigen Grad unvermeidlich. Wenn der Zielpunkt weit entfernt ist, haben diese kleinen Störungen keine großen negativen Auswirkungen. Je geringer aber die Entfernung zum Zielpunkt ist, desto gravierender wirken sich kleine Störungen aus. Im Bild 10.2 ist eine Situation dargestellt, in der der Roboter immer eine geringfügige Abweichung in der Fahrtrichtung nach rechts hat.

In unmittelbarer Nähe des ersten Wegpunktes hat die geringfügig vom optimalen Pfad abweichende Position, die hervorgerufen wurde durch die geringfügige

[3]Die Module bleiben aber logisch getrennt. D.h. nur die Kommunikationsverbindung wurde durch gemeinsame Variablen ersetzt, um die Reaktionsfähigkeit des Fahrers zu erhöhen

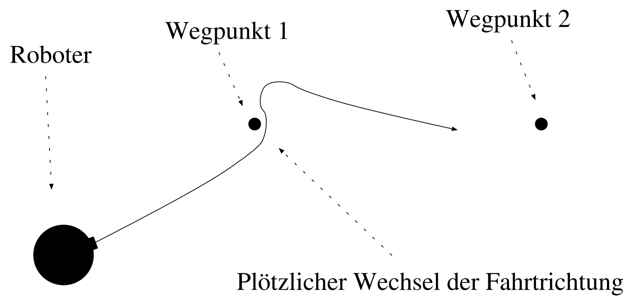

Abbildung 10.2: Weg eines Roboters mit leichter Rechtsabweichung von der korrekten Orientierung

Abweichung von der optimalen Orientierung, zur Folge, daß die Orientierung zum Wegpunkt sich bei weiterer Annäherung an den Wegpunkt dramatisch ändert. Bis der Roboter den Wegpunkt hinter sich gelassen hat, hat er den Kurs um fast $90°$ ändern müssen. Erst dann wird der nächste Wegpunkt berücksichtigt und der Roboter muß erneut den Kurs ändern.

Eine einfache Lösung des Problems ist, schon vor Erreichen des Wegpunktes die Zielverfolgung auf den nächsten Wegpunkt umschalten zu lassen. Wie man in Bild 10.3 erkennt, gibt es aber auch bei diesem Vorgehen eine plötzliche, wenn auch bei weitem nicht so extreme, Richtungsänderung.

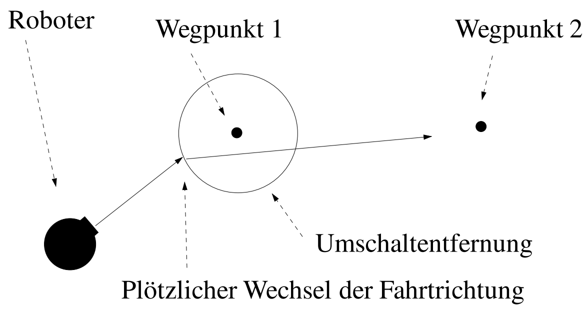

Abbildung 10.3: Weg eines Roboters bei vorzeitigem Wechsel des Zielpunktes

Eine elegantere Lösung ist in Bild 10.4 skizziert. Der Einfluß des zweiten Wegpunktes steigt bei der Annäherung des Roboters an den ersten Wegpunkt kontinuierlich an, während gleichzeitig der Einfluß des ersten Wegpunktes abnimmt. Wenn der Roboter schließlich beim ersten Wegpunkt ankommt, orientiert sich der Roboter ausschließlich am zweiten Wegpunkt.

Vom Prinzip her wird die aktuelle Zielorientierung Θ als gewichtete Summe

$$\Theta = \eta \cdot \Theta_1 + (1 - \eta) \cdot \Theta_2 \tag{10.1}$$

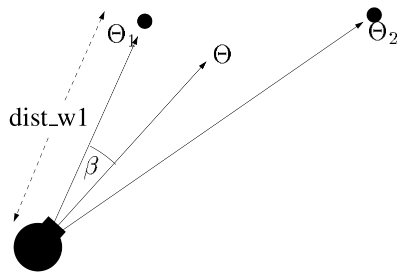

Abbildung 10.4: Kontinuierlicher Wechsel des Zielpunktes

bestimmt, wobei der Gewichtungsparameter η von 1 auf 0 variiert, während der Roboter sich dem ersten Wegpunkt nähert.

In der Implementation des Fahrers wird ab einer Mindestentfernung d_min vom ersten Wegpunkt die Gewichtung der beiden Zielorientierungen vorgenommen. Der Gewichtungsparameter wird dabei durch

$$\eta = \frac{dist\_w1}{d\_min} \tag{10.2}$$

bestimmt. Bei der Berechnung der Gewichtung muß zusätzlich der Übergang von $0°$ auf $360°$ berücksichtigt[4] werden. Um die Berechnung zu vereinfachen wird die Gleichung (10.1) umgeformt zu:

$$\Theta = \Theta_1 + \beta \tag{10.3}$$

wobei

$$\beta = (1 - \eta) \cdot (\Theta_2 - \Theta_1) \tag{10.4}$$

Der Übertrag ist hier einfacher zu behandeln, da nur die Winkeldifferenz von zwei Orientierungen berechnet werden muß.

Bei diesem Verfahren ist darauf zu achten, daß noch vor dem Erreichen des ersten Wegpunktes das nächste Wegpunktpaar vorgegeben werden muß.

10.3 Reaktive Hindernisvermeidung mit Fuzzy-Regelung

In diesem Abschnitt wird die Konstruktion eines Fuzzy-Reglers zur reaktiven Kollisionsvermeidung und dessen Verhalten in der Praxis behandelt.

[4]Die Gleichung (10.1) ist also nicht dazu geeignet, die Zielorientierung konkret zu berechnen, sondern gibt nur die zugrundeliegende Idee wieder.

10.3.1 Ein- und Ausgabegrößen des Fuzzy Reglers

Als Ausgaben liefert der Fuzzy-Fahrer in Abständen von einer Zehntel Sekunde die gewünschte Translations- und Rotationsgeschwindigkeit des Fahrwerks. Diese Stellgrößen werden aus fünf Entfernungswerten gebildet. In der Abbildung 10.5 erkennt man fünf Winkelbereiche am Roboter, die mit f, r1, r2, l1 und l2 bezeichnet sind. Im folgenden werden diese Bereiche Segmente genannt. Jedes dieser Segmente deckt einen Bereich von ca. $60°$ ab. Die Ultraschallsensoren, deren Detektionskegel innerhalb eines Winkelbereichs liegen, werden dem entsprechenden Segment zugeordnet. Beim RWI-Roboter sind das jeweils vier Ultraschallsensoren pro Segment.

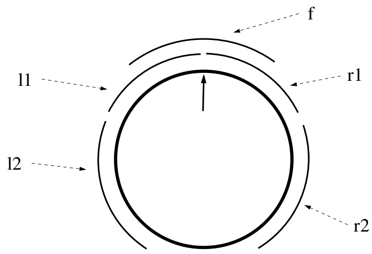

Abbildung 10.5: Zuordnung der Ultraschallsensoren zu Segmenten

Die zu einem Winkelsegment gehörige Eingabegröße wird durch Minimumsbildung der Ausgaben der vier zum Segment gehörigen Ultraschallentfernungsmesser gebildet. Da die Sensoren (siehe Abschnitt 3.2) nie eine zu geringe Entfernung anzeigen können, aber ein Hindernis durchaus 'unsichtbar' für die Sensoren sein kann, werden durch dieses Vorgehen sehr viele Fehlmessungen unterdrückt und man hat trotzdem zu jedem Zeitpunkt eine definierte Eingabegröße. Aus den fünf Entfernungswerten der fünf Segmente wird ebenfalls durch Minimumsbildung ein weiterer Eingabewert erzeugt, der die kürzeste überhaupt gemessene Entfernung zu einem Hindernis anzeigt.

Die Eingabewerte werden in keiner Weise zeitlich verarbeitet. Eine Ausgabe des Fuzzy-Fahrers entsteht aus einer einmaligen Messung aller Ultraschallsensoren. Zusätzlich wird dem Fahrer durch die Angabe einer Orientierung, relativ zur aktuellen Ausrichtung des Roboters, die Richtung zum Ziel gewiesen. Die Aufgabe des Fuzzy-Fahrers ist es nun, aus den Entfernungsangaben und der Zielrichtung eine Rotations- und Translationsgeschwindigkeit zu berechnen, die bewirkt, daß Hindernisse vermieden und möglichst schnell in Richtung Ziel gefahren wird.

10.3.2 Linguistische Variablen des Fahrers

Die Eingabe und Stellgrößen werden im Fuzzy-Fahrer als linguistische Variable modelliert.

- Die Entfernungsangaben zu den Segmenten r1, r2, f, l1, l2 und die Minimaldistanz werden durch jeweils eine Variable repräsentiert, die in sechs Fuzzy-Terme unterteilt sind, die nichtlinear den Bereich von 0 bis 2000 mm abdecken.

- Die Zielorientierung wird durch eine Variable repräsentiert, die in 17 Terme unterteilt ist.

- Die Translationsgeschwindigkeit wird durch eine Variable mit sieben Termen dargestellt

- Die Rotationsgeschwindigkeit wird durch eine Variable mit 11 Termen dargestellt.

Jeder Term besitzt eine Zugehörigkeitsfunktion, die beim zentralen Wert des Terms genau 1.0 beträgt und linear zu dem nächsten Nachbar abfällt. Ein Beispiel dafür ist in Bild 10.6 zu erkennen.

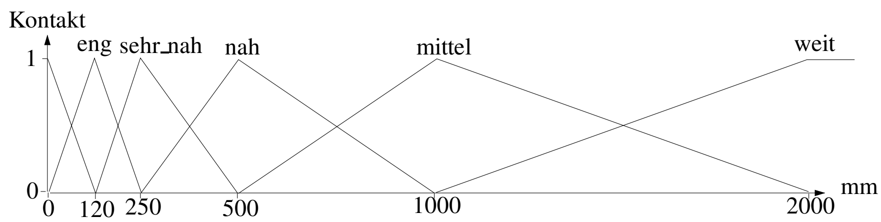

Abbildung 10.6: Linguistische Variable Entfernung

10.3.3 Regelmenge des Fahrers

Die konkrete Regelmenge wird an dieser Stelle nicht aufgelistet, da dies für das Verständnis des Algorithmus keinen Gewinn bringt. Stattdessen werden die zugrundeliegenden Ideen, die zur Regelmenge führten, beschrieben. Bei Interesse am konkreten Regelsatz kann diese im File 'HARDWARE/fc.cc' unter Berücksichtigung der Implementation der Fuzzy-Variablen in 'HARDWARE/fuzzy_var.cc' gefunden werden.

Die Regelmenge ist in die drei folgenden Gruppen unterteilt.

Kollisionsvermeidung

Die erste Regelgruppe verhindert Kollisionen durch Regelung der Rotationsgeschwindigkeit. Dazu werden hauptsächlich die Segmente r1 und l1 verwendet, die an der Front des Roboters $120°$ abdecken. Die Grundidee ist sehr einfach. Falls das rechte Segment r1 eine geringere Entfernung anzeigt, als das linke Segment l1, so wird nach links ausgewichen, um dem Hindernis auf der rechten Seite auszuweichen. Da dieses Ausweichverhalten noch mit anderen Verhaltensweisen kombiniert werden muß, wird bei zunehmender Entfernung vom Hindernis die Gewichtung der Ausgaben herabgesetzt. Ein Hindernis auf der rechten Seite löst zwar unabhängig von der Entfernung immer eine Ausweichreaktion aus, diese kann aber bei großen Entfernungen durch die geringe Gewichtung leicht übersteuert werden. Die Gewichtung gibt also die Dringlichkeit der Ausweichreaktion an. Zusätzlich wirken die Segmen-

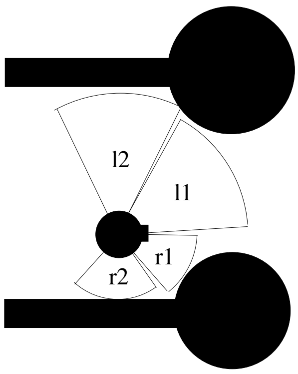

Abbildung 10.7: Kollisionsvermeidung unter Berücksichtigung der Segmente r1 und l1

te r2 und l2 an den Seiten des Roboters als Wandvermeider. Wenn der Roboter fast parallel zu einer Wand fährt, so werden diese von den Segmenten r1 und l1 nicht detektiert. Bei geringen seitlichen Entfernungen wird der Roboter geringfügig von der Wand weggedreht, um ein seitliches hineindriften in die Wand zu verhindern. Allerdings sind die Gewichtungen für die Wandvermeidungsregeln so gering, daß sie jederzeit von einer Hindernissvermeidungsreaktion überdeckt werden können.

Die Vorbedingungen der Fuzzy-Regeln zur Kollisionsvermeidung decken jede mögliche Kombination der linguistischen Belegungen der Entfernungsvariablen der Segmente r1 und l1 (bzw. r2 und l2) ab. In ihrer Gesamtheit entspricht die Regelmenge einer vollständig ausgefüllten Tabelle, die jeder möglichen linguistischen Eingabebelegung eine Ausweichreaktion zuordnet.

Zielverfolgung

Die Zweite Regelgruppe behandelt die Zielverfolgung. Bei Abweichung vom Zielwinkel wird die Rotationsgeschwindigkeit so beeinflußt, daß der Roboter auf den Zielwinkel zudreht. Die Gewichtungen sind dabei so verteilt, daß bei geringen Abweichungen sich eher die Kollisionsvermeidung durchsetzt und bei extremen Zielabweichungen oder sehr entfernten Hindernissen eher die Zielverfolgung.

Geschwindigkeitsregelung

Die Dritte Regelgruppe steuert die Translationsgeschwindigkeit. Ohne Geschwindigkeitsregelung könnte es trotz der schon vorhandenen Ausweichreaktion noch zu Kollisionen kommen. Die Regelung der Translationsgeschwindigkeit ist notwendig zur Kollisionsvermeidung, da der realisierbaren Rotationsgeschwindigkeit physikalische Grenzen gesetzt sind und außerdem die Ultraschallsensoren bei hohen Rotationsgeschwindigkeiten unzuverlässig arbeiten. Die Geschwindigkeitsregelung zerfällt in mehrere Verhaltensweisen, die sich gegenseitig überlagern. Zum Ersten wird die Frontdistanz, die vom Segment f gemessen wird, direkt mit hoher Priorität in ein Bremssignal umgesetzt. Unterhalb einer gewissen Frontdistanz soll keine Translation mehr stattfinden. Als zweites wird die schon erzeugte Rotationsgeschwindigkeit dazu verwendet, eine Ausweichsituation zu erkennen. Falls die Rotationsgeschwindigkeit hoch ist, wird ein mögliches Ausweichmanöver durch eine geringe Geschwindigkeit oder sogar einen vollständigen Stop unterstützt. Ähnlich wirkt die Umwandlung von extremen Zielabweichungen in ein Stopsignal. Falls der Pilot plötzlich die lokalen Zielpunkte so verschiebt, daß eine extreme Zielabweichung beim Fuzzy-Fahrer angezeigt wird, so ist es wahrscheinlich, daß die Richtungsänderung auf engstem Raum stattfinden muß, was vernünftigerweise mit geringer Translationsgeschwindigkeit durchgeführt wird.

10.3.4 Vor- und Nachteile des Fuzzyansatzes

Der Hauptvorteil der Anwendung eines Fuzzy-Reglers ist die Möglichkeit, sehr schnell aus einer umgangssprachlich formulierten Handlungsanweisung ein funktionierendes System zu bauen. In nur zwei Tagen war es möglich, aus der Idee ein Programm zu entwickeln, das ansatzweise das vom Entwickler angestrebte Verhalten zeigte. Dabei wurde der überwiegende Teil der Entwicklungszeit für die Implementierung der Fuzzy-Klasse benötigt. Ein Prototyp der endgültigen Regelmenge war schon nach wenigen Stunden lauffähig und zeigte sinnvolles Verhalten.

Der Hauptnachteil ist, daß dieser Prototyp nur mit sehr viel Mühe und im Endstadium nur noch durch blindes 'Trial and Error' spürbar zu verbessern war. Eine einmal durch diesen Prozeß optimierte Regelmenge ist kaum erweiterbar, ohne die schon vorhandene Funktionalität wieder zu zerstören. Ein weiterer Kritikpunkt ist die ausufernde Regelflut. Verschiedene Regelgruppen, die dieselbe Ausgabevariable beeinflussen, können wegen der eingeschränkten Beeinflussungmöglichkeiten[5] bei der Defuzzifikation entweder nicht das Verhalten zeigen, das vom Anwender gewünscht wird, oder durch die Notwendigkeit, jede mögliche Kombination von Zuständen der beteiligten Eingabevariablen zu betrachten, tritt eine kombinatorische Explosion der Regelanzahl ein.

Bei der reaktiven Hindernisvermeidung hat sich der Fuzzyansatz als mächtig genug erwiesen, um praktikable Ergebnisse zu erzielen. Nur bei Hindernissen, die für die Ultraschallsensoren faktisch unsichtbar sind, kam es bei den Experimenten überhaupt zu Kollisionen. Allerdings ist eine wesentliche Leistungssteigerung mit einem erweiterten Regelsatz nicht zu erwarten.

10.4 Kombinierte Experten

Bei diesem Ansatz wird die Steuerungsaufgabe in mehrere einfach zu lösende lokale Teilprobleme zerlegt, wobei die lokalen Ergebnisse in einem zweiten Schritt zur globalen Lösung kombiniert werden. In Bild 10.8 ist die grundsätzliche Vorgehensweise dargestellt.

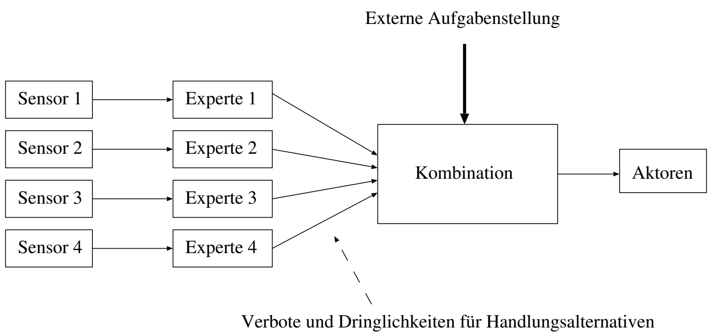

Abbildung 10.8: Kombination von lokalen Expertenmeinungen zur globalen Lösung

Die Ausgaben von jedem Sensor wird getrennt von einem Experten begutachtet. Dieser Experte entscheidet aus seiner lokalen Sicht heraus über Verbote, die schädliche Handlungen verhindern, und gibt Dringlichkeiten für Handlungsalternativen an, die positive Auswirkungen haben.

Im konkreten Fall der reaktiven Hindernisvermeidung sind die Sensoren, wie beim Fuzzyansatz, zu Segmenten zusammengefaßte Gruppen von Ultraschallentfernungsmessern.

[5]Regeln mit der gleichen Ausgabevariable werden grundsätzlich als ODER-Verknüpft angesehen

Für jedes Segment gibt es einen 'Experten', der aus dem zum Segment gehörigen aktuellen Entfernungswert lokal die Entscheidung trifft, welche Stellgrößen für Rotation und Translation zu vermeiden sind. Zunächst wird die Rotation betrachtet.

10.4.1 Bestimmung des Rotationsparameters

Die Experten der Segmente l1 und l2 ermitteln eine obere Schranke und die von r1 und r2 eine untere Schranke für den Rotationsparameter. Die beiden linken Segmente l1 und l2 verhindern mit den oberen Schranken, daß bei Detektion eines Hindernisses (auf der linken Seite) in Richtung dieses Hindernisses gedreht wird. Die rechten Segmente erzeugen entsprechend untere Schranken, und nur der Restbereich, der durch keine Schranke verboten wird, enthält zulässige Belegungen für den Rotationsparameter.

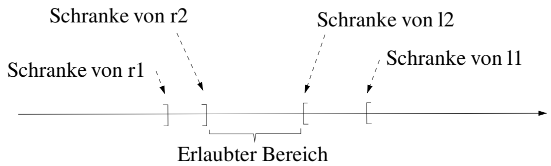

Abbildung 10.9: Verbotsschranken für den Rotationsparameter

Zusätzlich zu den Verbotsschranken erzeugt jeder Segmentexperte ein Signal, das die Dringlichkeit einer Ausweichreaktion aus der Sicht dieses Experten angibt. Wenn z.B. das Segment l1 eine sehr geringe Entfernung anzeigt, so signalisiert der zugehörige Experte eine sehr hohe Dringlichkeit, dem detektierten Hindernis (nach rechts) auszuweichen. Bei großen Entfernungen tendiert die Ausweichdringlichkeit gegen Null.

Wenn der erlaubte Bereich für den Rotationsparameter das Intervall $[l, u]$ ist und die Segmentexperten für l1, l2, r1, r2 jeweils die Dringlichkeiten l1d, l2d, r1d, r2d ausgeben, so wird der Rotationsparameter folgendermaßen festgelegt:

$$\alpha = \frac{l+u}{2} + \frac{u-l}{2} \cdot \frac{(r1d + r2d)^2 - (l1d + l2d)^2}{(r1d + r2d)^2 + (l1d + l2d)^2}$$

Je nach Dringlichkeit einer Ausweichreaktion nach links oder rechts wird also der zulässige Spielraum im Intervall $[l, u]$ nach oben oder unten ausgenutzt.

Die so gewonnene Rotationsstellgröße berücksichtigt allerdings nur die Hindernisvermeidung. Um zusätzlich auch den Zielwinkel zu berücksichtigen, wird α mit einer Stellgröße zur Zielverfolgung η kombiniert. Damit erhält man dann die neue Stellgröße α'.

$$\alpha' = (1 - \gamma) * \eta + \gamma * \alpha$$

$$\gamma = \mu * (r1d + r2d + l1d + l2d)$$

Der Parameter μ gibt dabei an wie stark die Hindernisvermeidung bzw. die Zielverfolgung gewichtet wird. Der Parameter μ muß so klein gewählt werden, daß γ immer kleiner als eins ist.

10.4.2 Bestimmung des Translationsparameters

Zur Steuerung der Translationsgeschwindigkeit erzeugt jeder Segmentexperte aus seiner lokalen Sicht eine obere Schranke für die maximal zulässige Geschwindigkeit. Als tatsächliche Translationsgeschwindigkeit wird dann das Minimum aller oberen Schranken benutzt.

10.4.3 Bestimmung der Schranken und Dringlichkeiten

Es ist noch die Frage offen, wie die Bildung der Schranken und Dringlichkeiten aus den Entfernungswerten der Segmente konkret abläuft.

Da jeder Experte nur genau einen Eingabeparameter[6] zu beachten hat, ist es problemlos möglich, diesen direkt auf die zugehörige Schranke und den Dringlichkeitswert abzubilden. Dies geschieht mittels linearer Interpolationsfunktionen, die auf der Basis von wenigen Stützpunkten die Abbildungen definieren.

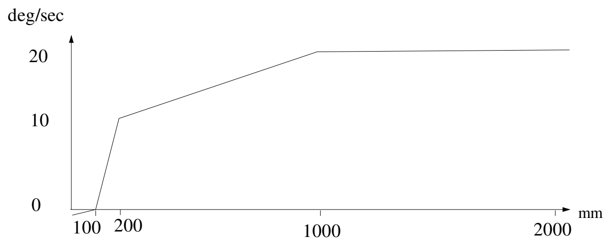

Abbildung 10.10: Obere Schranke für die Rotation bzgl. l2

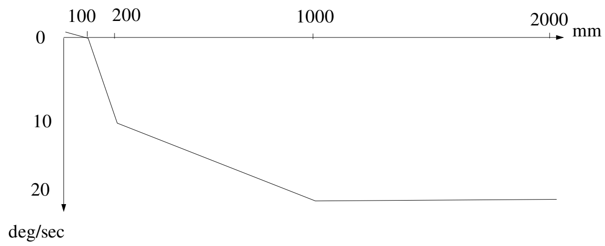

Abbildung 10.11: Untere Schranke für die Rotation bzgl. r2

[6] den Entfernungswert des Segments

10.4.4 Bewertung des Verfahrens

Das Verfahren ist einfach genug, um problemlos die knappe Zeitbeschränkung zu erfüllen. Das große Problem, eine hochdimensionale Eingabe mit geringem Aufwand auf die Steuerungsparameter Rotation und Translation abzubilden, wird durch die Zerlegung in viele eindimensionale Probleme mit anschließender Kombination der Teilergebnisse gelöst. Diese Teilprobleme sind unmittelbar mit der Erfahrung eines Operateurs lösbar, und der Zusammenhang zwischen den Kenngrößen des Verfahrens[7] und dem Verhalten des Roboters ist direkt nachvollziehbar, da auch Menschen dazu neigen, Einzelaspekte eines Problems getrennt zu bewerten und anschließend eine Gewichtung dieser Aspekte vorzunehmen. Es kann also, wie beim Fuzzy Ansatz, schon vorhandene Problemlösungskompetenz auf den Hindernisvermeider übertragen werden, ohne den Makel, daß die Modifikationen im Entwurfsprozeß zunehmend undurchschaubarer werden. Durch die Verwendung von harten Schranken werden unerwünschte Reaktionen explizit ausgeschlossen. Der Algorithmus hat nur die Möglichkeit, innerhalb von risikoarmen Schranken zu agieren.

Die Definition der Interpolationsfunktionen kann prinzipiell auch durch ein Lernverfahren unter der Maßgabe einer Kostenfunktion erfolgen. Dabei bleibt der Vorteil der direkten Verständlichkeit weiterhin erhalten.

Als Lernverfahren bietet sich ein genetischer Algorithmus an, bei dem mutierte Interpolationsfunktionen rekombiniert werden. Aus einer Population von Hindernisvermeidern selektiert man diejenigen, die in Testläufen bei geringer Neigung zu Kollisionen die Ziele am schnellsten anfahren. Die Kostenfunktion, die einem Individuum seine Fittness zuordnet, definiert bei diesem Selektionsvorgang, was 'schnell' bzw. 'kollisionsarm' bedeutet.

[7] Die Stützstellen der Interpolationsfunktionen

Kapitel 11

Das Kameramodul

Um die Kartenbildung durch die Ultraschallsensoren zu unterstützen, wurde ein Modul implementiert, das den Kamerakopf des RWI-Roboters zusammen mit einem Laser dazu verwendet, Entfernungen zu messen. Die Kameras haben die Aufgabe, immer dann, wenn der Navigator Zielpunkte dort plaziert, wo die Integratorkarten Hindernisse anzeigen, zu klären, ob die Hindernisse tatsächlich existieren oder nur Artefakte aufgrund der geringen Winkelauflösung der Ultraschallsensoren sind. Eine typische Anwendungsmöglichkeit ist das Vermessen von Türöffnungen, die von Ultraschallsensoren nur unvollkommen wahrgenommen werden. Es soll hauptsächlich die Frage beantwortet werden, ob eine Tür geschlossen oder offen ist.

Herkömmliche Verfahren zur Tiefenextraktion aus Stereobildern sind fast immer auf das Vorhandensein von vertikalen Kanten im Bild angewiesen. Diese Kanten werden dann auf unterschiedliche Weise im Stereobildpaar einander zugeordnet. Daraus wird die Disparität bestimmt, aus der schließlich die Entfernung resultiert.

Im Falle der Problemstellung, ob die Tür offen oder geschlossen ist, sind die Ergebnisse dieser Verfahren völlig unbrauchbar. Sowohl bei offener wie geschlossener Tür sind die detektierbaren vertikalen Kanten die des Türrahmens. In beiden Fällen wird als Entfernung die Entfernung des Rahmens zurückgegeben. Die konturlose weiße Türfläche bzw. die konturlose weiße Wand, die bei geöffneter Tür zu sehen ist, können mit diesen Verfahren nicht unterschieden werden. Ein zusätzliches Manko ist der enorme Rechenaufwand, der zur Tiefenextraktion getrieben werden muß. Eine Echtzeitverarbeitung auf den im Roboter installierten Rechnern ist kaum denkbar.

11.1 Tiefenextraktion mit Kamera und Laser

Da die bisher genannten Ansätze zeitaufwendig und eher als unzuverlässig einzustufen sind, wurde ein simplerer Weg gewählt, um Tiefeninformation zu gewinnen. Zusätzlich zu den Kameras wurde auf dem Kamerakopf ein roter Laser installiert. Er hat die Aufgabe, im Blickfeld der beiden Kameras zumindest für ein Punktepaar eine eindeutige Dispariätsangabe zu ermöglichen, da der rote Lichtpunkt in beiden Bildern eindeutig detektiert wird.

11.1.1 Detektion eines roten Lichtpunktes in heller Umgebung

Eine vermeintlich zuverlässige Methode, den Laserpunkt einfach als Maximum des Rotanteils des Bildes zu detektieren, ist bei einer 'normalen' Büroumgebung mit ungleichmäßiger Beleuchtung und vielen farbigen Gegenständen zum Scheitern verurteilt.

Ein besonders augenfälliges Beispiel dafür ist die Tatsache, daß für das menschliche Auge ein roter Lichtfleck selbst auf einem hell beleuchteten Stück Papier klar erkennbar ist, während im Rotauszug des Bildes der Laserfleck praktisch nicht mehr wahrnehmbar ist.

Der Hauptgrund dafür ist, daß der Laser aus Sicherheitsgründen so leistungsschwach sein muß, da eine direkt von der Sonne angestrahlte weiße Fläche - auch nach der Filterung - luminanter sein kann als der Laserpunkt, der evtl. an einer ganz anderen Stelle im Bild zu sehen ist.

Da auf dem Roboter Farbkameras installiert sind, wird in der Implementation Gebrauch von allen zur Verfügung stehenden Farbkomponenten gemacht, um dieses Problem zu umgehen.

Der rote Punkt wird hauptsächlich durch das Intensitätsverhältnis der Farbanteile im RGB-Farbraum identifiziert. In den Bildern 11.1 und 11.2 ist jeweils eine Kameraaufnahme mit Laserpunkt dargestellt. In den Ursprungsbilder werden pixelweise die Abstände zur vermuteten Laserfarbe bestimmt. Diese Abstände werden als Grauwerte in den unteren Bildern 11.3 und 11.4 dargestellt. Je schwärzer ein Pixel dargestellt ist, desto ähnlicher der Farbton des korrespondierenden Pixels im Ursprungsbild zur Laserfarbe.

Nach der Filterung der Kamerabilder hat leider nicht, wie erwartet, immer ein Pixel des Laserflecks die höchste Intensität, sondern es tritt häufig auf, daß einzelne Pixel oder auch kurze horizontale Strecken eine intensivere Filterantwort zurückgeben, die für den menschlichen Betrachter nicht auffällig rot leuchten. Dies ließ sich auch durch einen Wechsel zu dem wesentlich aufwendigeren C.I.E. (u,v,Y)-Farbraum [23] nicht beheben, da die Störpixel teilweise identische RGB-Werte wie die Pixel im Laserfleck aufweisen. Die Störungen lassen sich also prinzipiell nicht allein durch Farbmerkmale vom Signal trennen.

Abbildung 11.1:

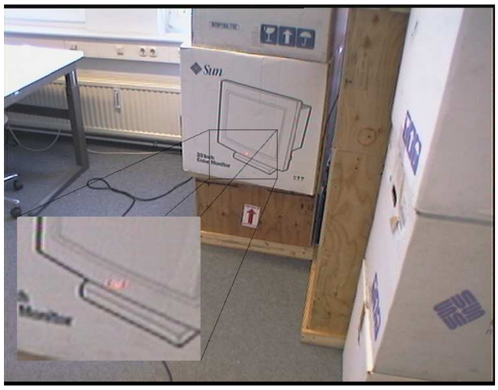

Abbildung 11.2:

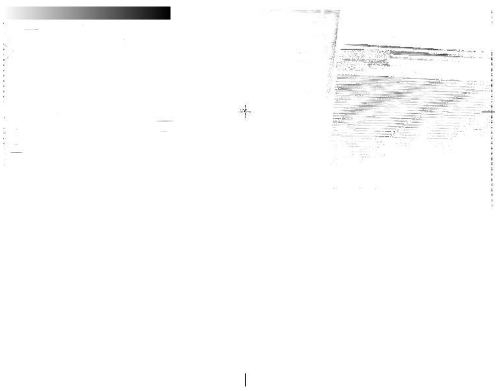

Abbildung 11.3:

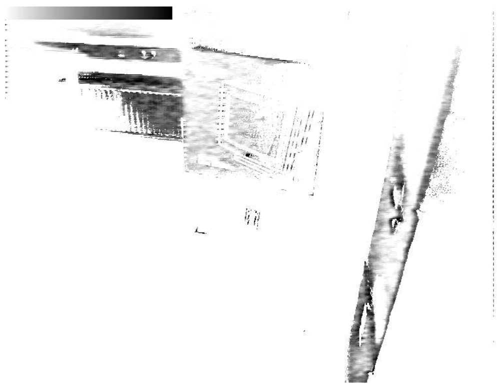

Abbildung 11.4:

Bild 11.5 ist eine Vergrößerung von Bild 11.1 im Bereich oberhalb des Schlüssels (der schwarze Fleck). An der Kante zur Oberseite des geschlossenen Teils des Schranks sind in der Vergrößerung mehrere purpurfarbene horizontale Linien zu erkennen. Dies sind die störenden Artefakte, die bei der Bildaufnahme entstehen. In der Realität ist der gesamte Schrank rein weiß.

Diese Störungen fallen beim beiläufigen Betrachten der Bilder nicht auf, da durch das menschliche Auge Farbinformation von vielen Pixeln zu einem Farbeindruck integriert werden. Diesen Effekt machen sich die Hersteller von Videokameras zunutze und codieren im Y/C-Format viermal soviel Luminanz- wie Chrominanzinformation. Die daraus resultierende Verschlechterung der Farbtreue einzelner Pixel[1] wird durch das geringe Farbauflösungsvermögen des menschlichen

[1] Bei der analogen Übertragung des Videosignals zum Framegrabber gibt es streng genommen keine Pixel, da jede Zeile durch jeweils ein Synchronisationsimpuls gefolgt von einem kontinuierli-

Abbildung 11.5: Vergrößerung von Bild 11.1

Auges wieder kompensiert, so daß keine Verfälschungen bemerkt werden.

Um einzelne rote Störpixel zu unterdrücken, wird die Filterantwort mit einer Maske der vermuteten Intensitätsverteilung des Laserpunktes nach der Filterung gefaltet (siehe Bilder 11.6 und 11.7).

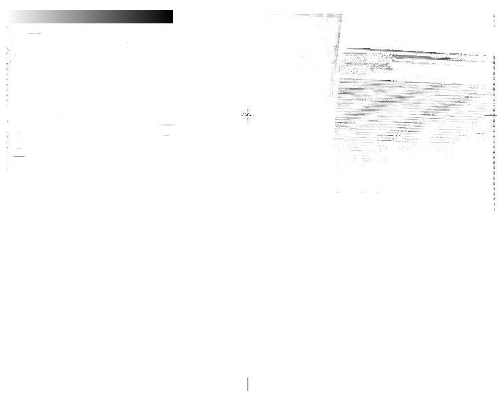

Abbildung 11.6: Filterung Pixelweise

Abbildung 11.7: Zusätzlich mit Intensitätsmaske

chen Signal codiert wird. Störungen beeinflussen so mehrere horizontal aufeinanderfolgende Pixel im digitalisierten Bild.

Der Laserpunkt wird allerdings immer noch nicht zuverlässig genug detektiert. Deshalb werden zwei Bilder aufgenommen. Das erste Bild mit eingeschaltetem Laser, das zweite Bild ohne Laser. Auf die Differenz des ersten und zweiten Bildes wird der bisherige Algorithmus angewendet.

Mit diesem Vorgehen wird insbesondere eine Farbverfälschung durch den Untergrund[2] des Laserflecks vermieden. Nachteilig ist, daß der Roboter zur Aufnahme der Differenzbilder gestoppt werden muß. Der große Vorteil ist eine fast hundertprozentig sichere Detektion des Punktes.

Ein weiterer Trick, um die Detektion robuster zu gestalten, ist nur auf der Epipolargraden des Laserstrahls im Bild nach dem Laserpunkt zu suchen. Störungen, die nicht in der Nähe der Epipolargeraden sind, haben so keinen Einfluß auf die Detektion.

11.1.2 Entfernungsbestimmung

Wenn die Position des Laserpunktes im Bild bekannt ist, so bleibt immer noch das Problem, aus dieser Information einen Entfernungswert zu berechnen. In diesem Abschnitt werden mögliche Ansätze zur Lösung dieses Problems beschrieben und der implementierte Ansatz mit diesen verglichen.

Entfernungsbestimmung durch direkte Triangulation

Das Standardverfahren zur Entfernungsbestimmung ist die Triangulation. Aus den bekannten Positionen von Kamera und Laser, kann der Schnittpunkt von Laser- und Sichtstrahl der Kamera bestimmt werden.

Die praktische Durchführung des Verfahrens ist leider deutlich komplizierter als die Theorie. Schon die exakte (relative) Positionsbestimmung von Laser und Kamera ist aufwendig und bei einem für direkte Messungen nicht geeigneten mechanischen Aufbau fehlerträchtig. Spätestens die Bestimmung der internen Kamperaparameter ist eine Aufgabe, zu deren Bewältigung viel Aufwand erforderlich ist.

Aus diesen Gründen wurde eine einfachere Methode zur Entfernungbestimmung gewählt, bei der die Kalibrierungsprozedur nur die Messung von wenigen einfach zu bestimmenden Größen erfordert.

Der implementierte Ansatz

Das Ziel ist es, so direkt wie nur möglich aus den Laserpunktkoordinaten im Bild die Entfernung zum Hindernis zu berechnen, damit durch eventuelle Rechenungenauigkeiten oder gar numerische Instabilität keine vermeidbaren Fehler entstehen. Das

[2] Die Projektionsfläche

der Entfernungsbestimmung zugrundeliegende Modell des Kamera/Laser-Systems sollte so beschaffen sein, daß Parameter, die nur umständlich oder sogar überhaupt nicht meßbar sind, aus einfach meßbaren Parametern rechnerisch hergeleitet werden können. D.h., die unvermeidliche Kalibrierungsprozedur sollte unkompliziert und die zu bestimmenden Parameter mit geringem Aufwand und hoher Genauigkeit meßbar sein. Annahmen oder Schätzungen sind zu vermeiden. Der im folgenden beschriebene Algorithmus erfüllt all diese Kriterien und ist zusätzlich sehr einfach zu verstehen und damit zuverlässig und schnell implementierbar.

Wenn man von Laserpunktkoordinaten[3] ausgeht, die von einer Lochkamera stammen, so stellt sich die Geometrie des Problems so wie in der Zeichnung 11.8 dar.

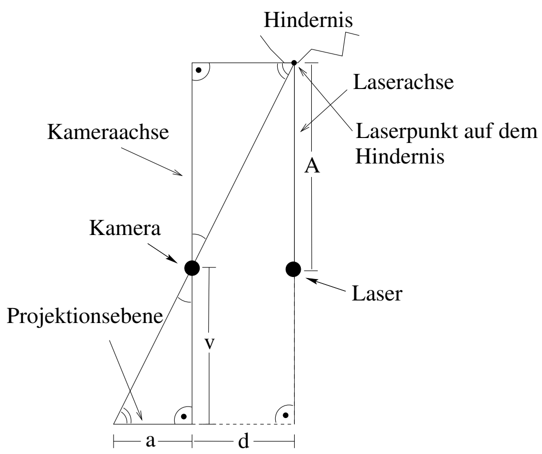

Abbildung 11.8: Geometrie der Kamerainstallation mit dem Laser

Kamera- und Laserachse sind parallel um die Strecke d versetzt. Das Hindernis befindet sich in einer Entfernung von A vor dem Laser und erzeugt auf der Projektionsfläche der Lochkamera einen Lichtpunkt, der a Einheiten von der Kameraachse entfernt ist.

Im Kamerabild messen kann man nur die Größe a. Da die beiden Dreiecke in der Zeichnung kongruent zueinander sind, gilt:

$$\frac{v}{A} = \frac{a}{d}$$

D.h.:

$$A = \frac{vd}{a}$$

[3] Auf der Bildebene

Das Problem stellen die Parameter d und vor allem v dar. Die seitliche Versetzung des Lasers von der Kamera könnte man mit Mühe einigermaßen genau bestimmen. Der Parameter v aber ist ein interner Kameraparameter, der **nicht** direkt meßbar ist.

In der Praxis ist dies aber nicht störend, da zur Entfernungsberechnung nur das Produkt va benötigt wird, welches theoretisch schon aus einer einzigen Kalibrierungsmessung, bei der die Entfernung zum Hindernis A bekannt ist, bestimmt werden kann.

Wenn a und die zugehörige Entfernung A vorliegen, berechnet man trivialerweise :

$$vd = Aa$$

Ein praktisches Problem, das die Situation geringfügig kompliziert, ist, daß es bei einer realen Kamera nicht trivial ist, dasjenige Pixel im Bild zu bestimmen, welches der optischen Achse der Kamera entspricht. Der CCD-Sensor kann verschoben, rotiert oder sogar verkantet zur Kameraoptik eingebaut sein, weshalb der Bildmittelpunkt fast nie mit der optischen Achse zusammenfällt.

Anstelle des Schnittpunktes der Projektionsebene mit der Kameraachse wird als Referenzpunkt im Bild dasjenige Pixel gewählt, welches den Laserpunkt abbildet, wenn der Roboter unmittelbar vor einem Hindernis steht und nicht weiter auf es zufahren kann (Der Punkt ganz links unten im Bild 11.9).

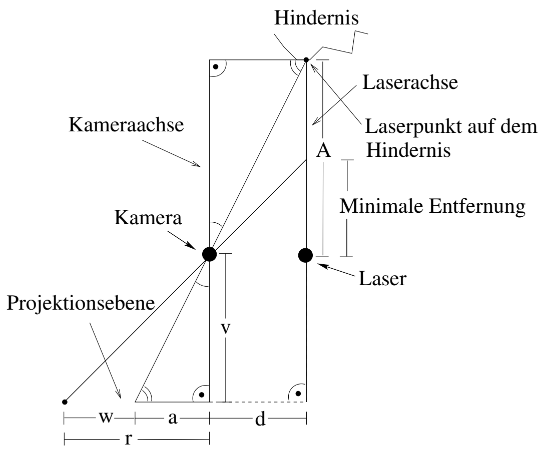

Abbildung 11.9: Geometrie der Kamerainstallation mit dem Laser II

Für jeden detektierten Laserpunkt wird ab jetzt die Entfernung w[4] zu diesem Referenzpunkt angegeben. Um die Entfernung der optischen Achse von diesem Referenzpunkt r und damit den gesuchten Wert $a = r - w$ zu bestimmen, sind zwei

[4]siehe Bild 11.9 links unten

Kalibrierungsmessungen mit unterschiedlichen Entfernungen zum Hindernis notwendig. Aus w_1, A_1 und w_2, A_2 berechnet man:

$$r = \frac{w_2 A_2 - w_1 A_1}{A_2 - A_1}$$

weil

$$(r - w_1)A_1 = dv = (r - w_2)A_2$$

Wenn man Meßfehler bei der Kalibrierung unterdrücken will, so bestimmt man eine große Anzahl von w_i und A_i und löst das überbestimmte Gleichungssystem:

$$\begin{pmatrix} A_1 w_1 \\ A_2 w_2 \\ \vdots \\ A_n w_n \end{pmatrix} = \begin{pmatrix} A_1 & -1 \\ A_2 & -1 \\ \vdots & \vdots \\ A_n & -1 \end{pmatrix} \begin{pmatrix} r \\ dv \end{pmatrix}$$

Dieses Gleichungssystem löst man mit den üblichen numerischen Methoden und erhält so r und dv genauer.

Wenn die Parameter r und dv bereitstehen, ist die Entfernungsbestimmung sehr einfach. Vom detektierten Laserpunkt wird die Entfernung w vom Referenzpunkt bestimmt und A durch

$$A = \frac{dv}{r - w}$$

berechnet.

Die vorhandenen Kameras haben nicht die Abbildungseigenschaften einer Lochkamera. Normalerweise verzerren die Kameraobjektive das Bild. Da ein explizites Modell dieser Verzerrung sehr aufwendig ist, wird das Bild so interpretiert, als ob es abhängig vom Ort mit verschiedenen Lochkameras aufgenommen wird. Da der Laserpunkt nur auf der zum Laserstrahl gehörigen Epipolargeraden im Bild zu finden ist, wird diese in mehrere Streckenabschnitte unterteilt, für die einzeln kalibriert wird.

Experimente haben ergeben, daß die Entfernungswerte auf 3 m nur um ca. 10 cm von der tatsächlichen Entfernung abweichen. Dies ist für die Roboteranwendung vollständig ausreichend, da die der Robotersteuerung zugrundeliegenden Karten ebenfalls nur eine Auflösung von 10 cm besitzen. Auf kürzere Entfernung sind die Messungen sogar erheblich genauer.

Kapitel 12

Verhalten des Gesamtsystems

In diesem Kapitel wird das Verhalten des Gesamtsystems in der Praxis beschrieben. Von besonderem Interesse ist dabei das Zusammenwirken der verschiedenen Module.

12.1 Zusammenspiel zwischen Pilot und Fahrer

Weder Pilot noch Fahrer sind allein für die Bewegung des Roboters verantwortlich. Der Fahrer weicht zwar Hindernissen aus, ist aber generell an die Vorgaben des Piloten gebunden. Im Idealfall leitet der Pilot den Fahrer in Regionen, in denen Kollisionen mit Hindernissen unwahrscheinlich sind. Der Fahrer modifiziert in solchen Situationen die Vorgaben des Piloten nur geringfügig. In Bild 12.1 ist eine typische Ausweichsituation mit viel Raum zum Ausweichen dargestellt. Unter der Annahme, daß alle Hindernisse korrekt in der Integratorkarte eingezeichnet werden, wird schon auf der Pilotenebene großräumig ausgewichen. Der Fahrer folgt im Wesentlichen den Wegpunkten.

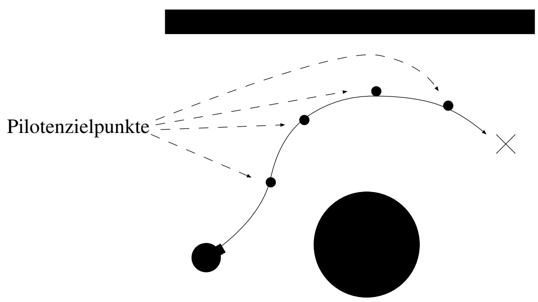

Abbildung 12.1: Bei ausreichendem Raum leitet der Pilot den Fahrer in freie Regionen

Wenn es aber, wie bei Türöffnungen, keine Alternativen zu einem Pfad mit großem Kollisionsrisiko gibt, so ist es wichtig, das Kräfteverhältnis zwischen Pilot und Fahrer korrekt einzustellen. In Abbildung 12.2 ist die Situation dargestellt, daß der Pilot aufgrund von unvollkommener Abbildung der offenen Tür einen Zielpunkt zu nah an einen Türpfosten legt.

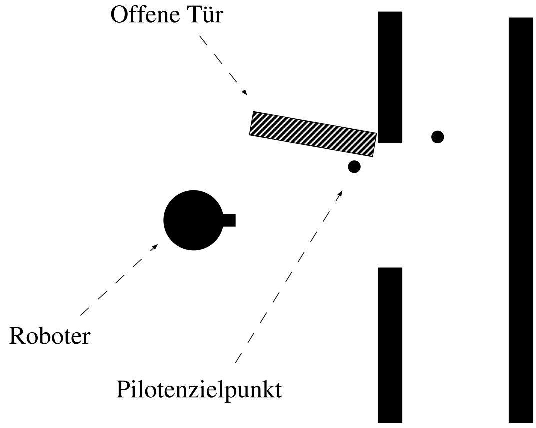

Abbildung 12.2: Der Pilot setzt aufgrund von mangelhafter Detektion der offenen Tür einen Wegpunkt zu nah an den Türpfosten

In dieser Situation muß der Fahrer den Türpfosten vermeiden, obwohl der Pilotenwegpunkt den Roboter in die Kollisionssituation leitet. Das richtige Gleichgewicht zu finden ist deshalb schwierig, weil kompromißlose Hindernisvermeidung ebenfalls zu einem unerwünschten Verhalten führt. In Grafik 12.3 ist der Pfad des RWI-Roboters dargestellt, der von einer frühen Version des Fuzzy-Fahrers gesteuert wird, wobei die Zielverfolgungskomponente zu schwach gewichtet ist.

Der Pilot erkennt aufgrund eines Navigatorzielpunktes, der im Gang positioniert ist, und der Integratorkarte des Raumes, in der der Roboter sich befindet, daß das Risiko, durch die enge Türöffnung zu fahren, nicht zu umgehen ist. Er kann allerdings aufgrund der zu schwachen Gewichtung der Zielverfolgungskomponente den Fahrer nicht dazu zwingen, durch die Türöffnung zu fahren.

Im Idealfall ist die Hindernisvermeidung stark genug, um von sich aus keine Kollisionsrisiken einzugehen und vor allem das Auftreten von Kollisionen zuverlässig zu vermeiden. Gleichzeitig ist der Einfluß des Piloten auf die Roboterbewegung groß genug, um vertretbare Risiken einzugehen, wenn diese auf einer höheren Entscheidungsebene (Pilot/Navigator) als notwendig erkannt wurden.

Dieses Gleichgewicht konnte beim Fuzzy-Fahrer nur mit großen Anstrengungen annähernd korrekt hergestellt werden. Haupthindernis ist die undurchschaubare Composition von vielen Regeln zu einer Ausgabe.

Der Ansatz mit den kombinierten Experten hat dieses Problem beseitigt. Durch

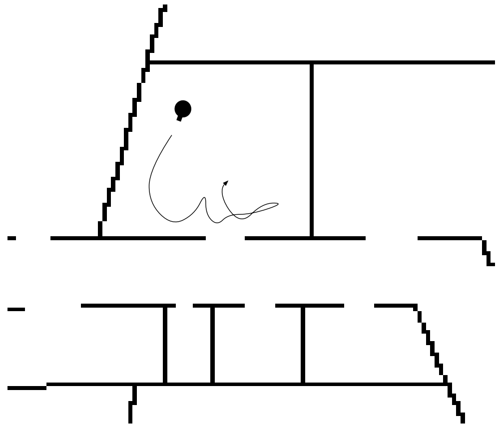

Abbildung 12.3: Pfad des Roboters mit zu starker Hindernisvermeidung

die Angabe von expliziten Verbotsschranken und die unmittelbare Zuordnung von Sensoreingaben auf Dringlichkeiten ist das Verhalten des Fahrers wesentlich genauer einstellbar als beim Fuzzy-Fahrer. In der Bilderserie von 12.4 bis 12.9 ist das erfolgreiche Durchfahren einer Türöffnung unter Verwendung des Ansatzes der kombinierten Experten dargestellt.

Abbildung 12.4:

Abbildung 12.5:

Abbildung 12.6:

Abbildung 12.7:

Abbildung 12.8:

Abbildung 12.9:

12.2 Zusammenspiel zwischen Navigator, Pilot und Lokalisation

Auf der Ebene des Navigators ist die Umgebungswahrnehmung vollständig zur aktuellen Position auf der vorgegebenen Karte abstrahiert. D.h. der Navigator ist vollständig darauf angewiesen, daß die Lokalisation die richtige Position angibt. Eine andere Vorgehensweise wäre auch sinnlos, da eine nachträgliche Verbesserung der Selbstlokalisation auch schon im Lokalisationsmodul erfolgen könnte. Andererseits macht eine globale Navigation, die direkt auf dem Wiedererkennen der Umgebung basiert, die Lokalisation überflüssig.

Beim Piloten ist eine gewisse Fehlertoleranz vorgesehen, da die reale Umgebung nicht nur aus Wänden besteht. Möbelstücke und langsame bewegliche Hindernisse müssen ebenfalls vermieden werden.

In einer Situation in der die Lokalisation fehlerhaft arbeitet und deshalb der Navigator einen Zielpunkt (in der Realität) zu nah an eine Wand setzt, kann der Pilot im beschränkten Maße korrigieren (siehe Abschnitt 9.4).

In den folgenden Abbildungen sind die protokollierten Positionen des Roboters während verschiedener Testfahrten dargestellt. Die weißen Punkte sind dabei die vom Fahrwerk gelieferten unverarbeiteten Odometriedaten und die schwarzen Punkte sind aus, von der Lokalisation korrigierten, Odometriedaten entstanden.

Die tatsächliche Position kann von der korrigierten Odometrieposition noch leicht abweichen. Beispielsweise im Bild 12.10 fährt der Roboter laut Protokoll ein kurzes Stück durch die Wand. In der Realität werden solche geringfügigen Abweichungen vom Piloten und dem Fahrer durch die Gewichtung von Zielverfolgung und Hindernisvermeidung kompensiert.

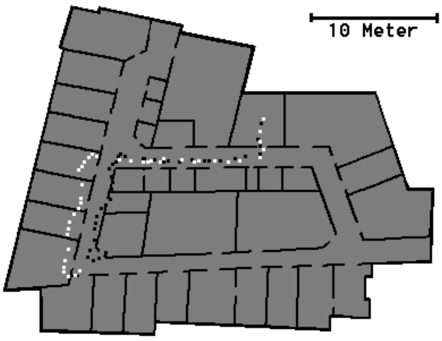

Abbildung 12.10:

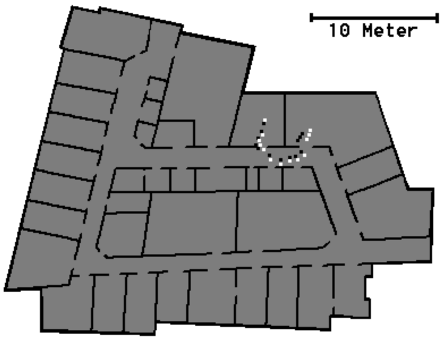

Abbildung 12.11:

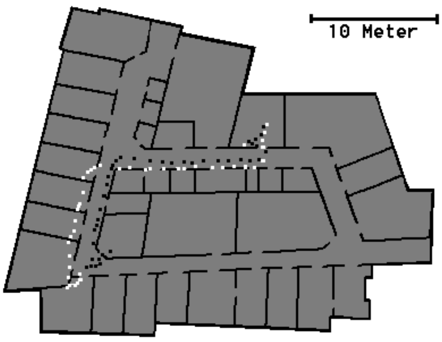

Abbildung 12.12:

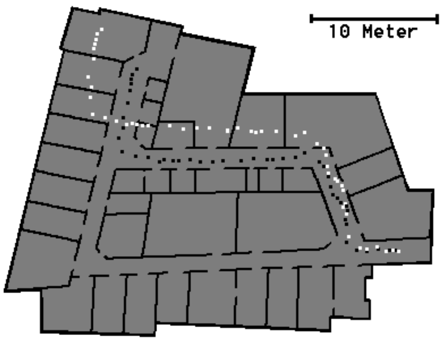

Abbildung 12.13:

Kapitel 13

Schlußwort und Ausblick

Das implementierte Programmpaket ist in der Lage, den Roboter per Mausklick an jeden beliebigen Ort auf der Karte zu schicken.

Die schwach hierarchische Gliederung der Software in Module, die auf verschiedenen Abstraktionsebenen bzw. Zeitskalen arbeiten, hat sich grundsätzlich bewährt.

Die Potentialfeldansätzte beim Navigator und Piloten ermöglichen eine effiziente und risikoarme Steuerung des Roboters, wobei der Pilot innerhalb seines Verantwortungsbereichs fehlertolerant agiert. Es treten dabei keine Probleme mit lokalen Minima oder stabilen Sattelpunkten auf wie z.B. bei [12] auf. Zusätzlich ist die Umgebung keinerlei Beschränkungen unterworfen, wie z.B. bei [14] oder [17] auf kreisförmige Hindernisse. Zu jeder beliebigen Karte entsteht immer ein Potentialfeld mit genau einem globalen Minimum. Im Vergleich zur *value iteration* aus [11] hat der verwendete Ansatz den Nachteil, erst einsetzbar zu sein, wenn die Karte vollständig 'geflutet' ist. Da dies aber nur wenige Sekunden dauert[1], ist das keine wesentliche Einschränkung. Beim flood-fill werden nur sehr wenige Kartenbereiche häufiger als ein einziges mal mit neuen Werten überschrieben. Bei *value iteration* muß fortwährend für *jedes* Kartenpixel die Updatevorschrift angewandt werden. Dieses Verfahren ist also prinzipbedingt um Größenordnungen langsamer. Dieser Geschwindigkeitsunterschied relativiert sich zwar bei veränderlichen Karten, auf denen *value iteration* bei lokalen Veränderungen schon nach wenigen Iterationen ein annähernd korrektes Potentialfeld erzeugt, aber wenige Iterationen aller Kartenpixel sind immer noch mehr als im Durchschnitt ein bis zwei Iterationen beim flood-fill.

Bei der reaktiven Hindernisvermeidung erlaubt der Ansatz der kombinierten Experten dem Fahrer, zuverlässig Situationen zu vermeiden, die der Entwickler vorher als gefährlich erkannt hat. Die Steuerung innerhalb des 'erlaubten' Parameterberei-

[1] für eine Karte mit den Ausmaßen 400x300 Pixel

ches kann prinzipiell auch mit anderen Verfahren als der einfachen quadratischen Gewichtung von Dringlichkeiten vorgenommen werden. Dieser Ansatz bietet noch erhebliches Entwicklungspotential.

Der Versuch die Module fast beliebig im Rechnernetz zu verteilen, hat sich als zweischneidig erwiesen. Einerseits begünstigt die modulare Aufteilung im Netzwerk die Fehlersuche, die Aufteilung der Programmierung auf mehrere Entwickler, die Verwendung von mehreren Rechnern zur Leistungssteigerung, den Wechsel zwischen verschiedenen Varianten der Software und eine geordnete Strukturierung der Aufgabe. Andererseits handelt man sich Probleme mit großen Latenzzeiten bei der Kommunikation ein. Ebenso ist die zeitliche Koordinierung der einzelnen Module schwierig. Wenn man an der modularen Aufteilung festhalten will, so wäre ein Mehrprozessorsystem mit schnellen Kommunikationswegen zwischen den Prozessoren optimal, wobei die Kommunikationswege gerade sehr kurze Nachrichten effizient übertragen sollten. Alternativ wäre der Einsatz eines Systems denkbar, das ein festes Teilungsverhältnis in der Verteilung der Rechenleistung auf verschiedene Prozesse zuläßt.

Eine größere Rechenleistung würde es ermöglichen, die vorhandenen Kameras besser zu nutzen bzw. die Verwendung von komplexeren Steuerungs- und Navigationsalgorithmen begünstigen. Eine Weiterentwicklung, weg vom Ansteuern von Koordinaten, hin zu einem verhaltensbasierten Vorgehen, etwa einem Gang bis zur dritten Tür auf der rechten Seite zu folgen und dann diesen Raum zu betreten, erscheint allein mit den Ultraschallsensoren nicht realisierbar. Im Beispiel würde die sichere Detektion selbst offener Türen scheitern. Eine Verbesserung der Umgebungswahrnehmung durch ein Laserradar oder eine ernsthafte Verwendung der Kameras würde die Navigation erheblich verbessern und die Anwendung robusterer Navigationsverfahren, die auf dem Wiedererkennen und der Orientierung an Merkmalen der Umgebung basieren, ermöglichen.

Anhang A

A.1 Bildung einer euklidischen Distanzkarte

In diesem Kapitel wird die in vielen Modulen verwendete Bildung einer euklidischen Distanzkarte aus einer binären Besetztheitskarte beschrieben. In jedem Feld dieser Distanzkarte ist die euklidische Entfernung zum nächstgelegenen besetzten Kartenpixel der Ausgangskarte verzeichnet.

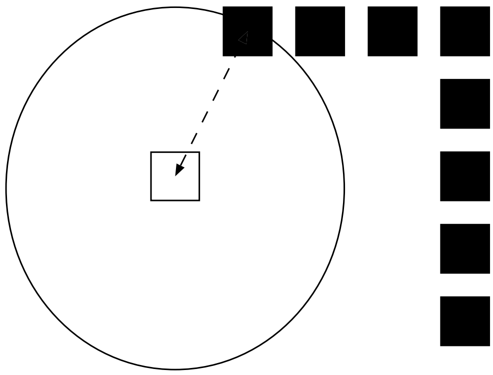

Abbildung A.1: In jedes Feld ist die minimale euklidische Distanz zum nächsten Hindernis verzeichnet

A.1.1 Voraussetzungen

Die Ausgangskarte ist in einem zweidimensionalen Array `map` gespeichert, deren Elemente nur mit den Werten 0 und 1 besetzt sind. Eine 1 bedeutet, daß das Feld besetzt ist und eine 0 bedeutet, daß das Feld leer ist. Das Ausgangskartenfeld hat die Ausmaße `map_x_size` mal `map_y_size`. Dies sind ebenfalls die Ausmaße der zu bildenden Distanzkarte `dmap`, sowie von zwei Hilfsarrays `dmap_dx` und `dmap_dy`, die alle aus reellen Elementen bestehen.

Das Distanzarray muß mit Werten, die größer als jede möglicherweise auftretende Entfernung sind, vorbelegt werden. Man kann dazu beispielsweise den Wert `map_x_size+map_y_size` verwenden. Die Hilfskarten `dmap_dx` und `dmap_dx` sind mit dem Wert 0 in jedem Feld vorinitialisiert.

Als Hilfsfunktion wird die Funktion $norm(x, y) := \sqrt{x^2 + y^2}$, die den euklidischen Abstand eines Vektors (x,y) vom Koordinatenursprung berechnet, benötigt.

A.1.2 Der Distanzalgorithmus

Der Algorithmus zerfällt in fünf Teilschritte. Im folgenden wird die algorithmische Vorgehensweise und das Teilergebnis jedes dieser Schritte beschreiben.

Schritt I

Im ersten Schritt wird die Ausgangskarte zeilenweise von links nach rechts (aufsteigende Indizes) durchgegangen und bei Auftreten eines besetzten Feldes ein Zähler dx auf 0 gesetzt. Solange kein weiteres besetztes Feld in der Ausgangskarte gefunden wird, so wird bei jedem Vorangehen zum nächsten Zeilenelement der Zähler um den Wert 1.0 erhöht und der Wert des Zählers in das entsprechende Feld der Hilfskarte `dmap_dx` eingetragen.

Ein entsprechendes Codefragment sieht folgendermaßen aus:

```
// left to right
for(y=0;y<map_y_size;y++)
{
  flag = 0;
  dx=0;
  for(x=0;x<map_x_size;x++)
  {
    if(map[y][x]==1) // Hindernis bei (x,y)
    {
      dx=0;
      flag=1;
    }
    else
    {
      dx++;
      if(flag!=0)
      {
        dmap_dx[y][x]=dx;
      }
    }
  } // for x
} // for y
```

Das Ergebnis des ersten Schrittes ist eine in `dmap_dx` gespeicherte Karte, in der in jedem Kartenfeld die Entfernung zum nächstgelegenen linken besetzten Feld in der Ausgangskarte verzeichnet ist.

Schritt II

Im zweiten Schritt wird die Ausgangskarte zeilenweise von rechts nach links (absteigende Indizes) durchgegangen. Bei dem Auftreten eines besetzten Feldes wird

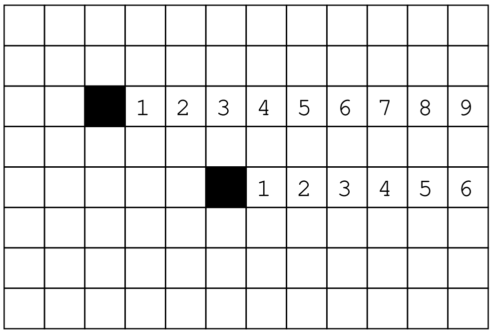

Abbildung A.2: Das Feld dmap_dx nach Schritt I

der Zähler dx wieder auf 0 gesetzt. Anders als im ersten Schritt wird in den leeren Folgefeldern nur jeweils dann der pro Feld um 1 inkrementierte Zähler in dmap_dx eingetragen, wenn der dort schon stehende Wert größer ist als der Inhalt von dx.

```
// right to left
for(y=0;y<map_y_size;y++)
{
  dx=2*map_x_size;
  for(x=map_x_size-1;x>=0;x--)
  {
    if(map[y][x]==1) // Hindernis bei (x,y)
    {
      dx=0;
    }
    else
    {
      dx++;
      if( dmap_dx[y][x] > dx )
      {
        dmap_dx[y][x]=dx;
      }
    }

  } // for x
} // for y
```

Das Ergebnis dieses Teilschritts ist eine in dmap_dx gespeicherte Karte, in der in jedem Kartenfeld die Entfernung zum horizontal (innerhalb der Zeile) nächstgelegenen Hindernis verzeichnet ist.

Schritt III

Im dritten Schritt wird die Ausgangskarte spaltenweise von oben nach unten (aufsteigende Indizes) durchgegangen. Jetzt werden aber nicht etwa nur die vertikalen Entfernungen betrachtet, sondern gleichzeitig vertikale und horizontale Abstände unter Berücksichtigung der gewünschten Norm. Analog zum Vorgehen in den vorangegangenen Fällen gibt der Zähler dy an, in welcher vertikalen Entfernung zur aktuell betrachteten Position in der Karte sich ein Hindernis befindet. Jetzt wird aber zusätzlich überprüft, ob die Entfernung (bezüglich der gegebenen Norm) geringer ist als die Entfernung, die schon in den Hilfsfeldern dmap_dx und dmap_dx

2	1		1	2	3	4	5	6	7	8	9
5	4	3	2	1		1	2	3	4	5	6

Abbildung A.3: Das Feld `dmap_dx` nach Schritt II

verzeichnet ist. Wenn dies der Fall ist, wird der Entfernungsvektor (`dx`,`dy`) in die Hilfsfelder eingetragen. Andernfalls wird der schon vorhandene Eintrag aus den Hilfsfeldern in die Zähler übernommen.

```
// down
for(x=0;x<map_x_size;x++)
{
  dx=dmap_dx[0][x];
  dy=dmap_dy[0][x];

  for(y=0;y<map_y_size;y++)
  {
    if(map[y][x]==1)
    {
      dy=0;
      dx=0;
    }
    else
    {
      dy++;
      if( norm(dx,dy) < norm(dmap_dx[y][x],dmap_dy[y][x]) )
      {
        dmap_dx[y][x] = dx;
        dmap_dy[y][x] = dy;
      }
      else
      {
        dx = dmap_dx[y][x];
        dy = dmap_dy[y][x];
      }
    }
  } // for y
} // for x
```

Durch dieses Vorgehen stellt man sicher, daß in den Hilfsfeldern ein Entfernungsvektor (ohne Vorzeichen) gespeichert wird, der relativ zum aktuellen Feld ein nächstgelegenes Hindernis angibt, das sich oberhalb oder auf gleicher Höhe wie das aktuelle Feld befindet.

Schritt IV

Im Schritt IV werden mit einer zu Schritt III völlig analogen Vorgehensweise zusätzlich die Hindernisfelder berücksichtigt, die unterhalb der jeweiligen Kartenposition befinden.

2/1

2/0	1/0		1/0	2/0	3/0	4/0	5/0	6/0
2/1	1/1	0/1	1/1	2/1	3/1	4/1	5/1	6/1
2/2	1/2	0/2	2/0	1/0		1/0	2/0	3/0
2/3	1/3	0/3	2/1	1/1	0/1	1/1	2/1	3/1
2/4	1/4	3/2	2/2	1/2	0/2	1/2	2/2	3/2
1/5	2/5	0/5	2/3	1/3	0/3	1/3	2/3	3/3

Abbildung A.4: Das Feld `dmap_dx` / `dmap_dy` nach Schritt III

```
// up
for(x=0;x<map_x_size;x++)
{
  dx=dmap_dx[map_y_size][x];
  dy=dmap_dy[map_y_size][x];
  for(y=map_y_size-1;y>=0;y--)
  {
    if(map[y][x]==1)
    {
      dy=0;
      dx=0;
    }
    else
    {
      dy++;
      if( norm(dx,dy) < norm(dmap_dx[y][x],dmap_dy[y][x]) )
      {
        dmap_dx[y][x] = dx;
        dmap_dy[y][x] = dy;
      }
      else
      {
        dx = dmap_dx[y][x];
        dy = dmap_dy[y][x];
      }
    }

  } // for y
} // for x
```

Als Ergebnis erhält man für jeden Punkt der Karte einen in `dmap_dx` und `dmap_dx` gespeicherten Vektor (ohne Vorzeichen), der auf ein bezüglich der Norm nächstgelegenes Hindernis zeigt.

2/1

2/2	1/2	0/2	1/2	2/2	0/4	1/4	2/4	3/4
2/1	1/1	0/1	1/1	2/1	0/3	1/3	2/3	3/3
2/0	1/0		1/0	2/0	0/2	1/2	2/2	3/2
2/1	1/1	0/1	1/1	1/1	0/1	1/1	2/1	3/1
2/2	1/2	0/2	2/0	1/0		1/0	2/0	3/0
2/3	1/3	0/3	2/1	1/1	0/1	1/1	2/1	3/1
2/4	1/4	3/2	2/2	1/2	0/2	1/2	2/2	3/2
1/5	2/5	0/5	2/3	1/3	0/3	1/3	2/3	3/3

Abbildung A.5: Das Feld `dmap_dx` / `dmap_dy` nach Schritt IV

Schritt V

Im letzten Schritt werden aus den Entfernungsvektoren euklidische Distanzen gebildet, die dann endgültig in die Distanzkarte `dmap` eingetragen werden können.

```
for(y=0;y<map_y_size;y++)  // dist
{
  for(x=0;x<map_x_size;x++)
  {
    dmap[y][x] = (int)( norm(dmap_dx[y][x],dmap_dy[y][x]) );
  }
}
```

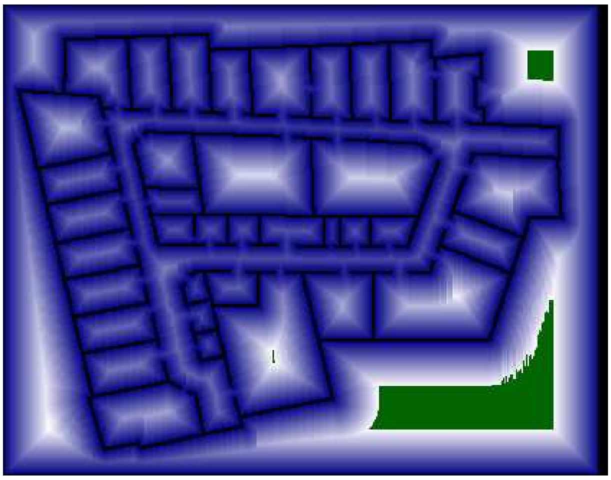

Abbildung A.6: Distanzkarte des Grundrisses

A.2 Die Definition von Entfernungskosten

Bei Navigation auf der Grundlage einer Entfernungskarte, werden zwei Hauptziele verfolgt.

- Der Pfad zum Ziel soll kurz sein
- Der Pfad soll Hindernissen fern bleiben

Diese beiden Ziele können nicht gleichzeitig optimal erfüllt werden. Ein optimal kurzer Pfad wird i.a. häufig Hindernisse berühren bzw. ein Pfad, der Hindernissen fern bleibt, ist i.a. nicht der kürzeste Pfad zum Ziel. Es ist ein quantitatives Kostenmaß erforderlich, das einen - bezüglich dieser Kosten - optimalen Kompromiß ermöglicht. Formal ist damit das Problem der Pfadplanung gelöst. In der Praxis bleibt das Problem, aus dem vom Entwickler gewünschten Fahrverhalten eine

Kostenfunktion zu konstruieren. In diesem Abschnitt werden Anhaltspunkte gegeben, welche Eigenschaften eine Kostenfunktion haben sollte, die zur Navigation auf der Grundlage einer Entfernungskarte benötigt wird. Die Kosten eines Pfades sind nämlich keineswegs natürlich gegeben. Auf die Frage, welche Annäherung an Hindernisse durch eine Pfadverkürzung aufgewogen wird, gibt es keine allgemeingültige Antwort.

A.2.1 Bewertung eines Pfades mittels der Entfernungskosten

Grundlage der Bewertung von Pfaden ist die Bewertung von Hindernisentfernungen. Je näher ein Hindernis, desto höher ist das Risiko mit diesem Hindernis zu kollidieren. Die Kosten, die einer Distanz zu einem Hindernis zugeordnet werden, werden hier Entfernungskosten genannt. Die Entfernungskosten haben ihr Maximum bei der Entfernung Null. D.h. wenn der Roboter ein Hindernis berührt, ist die Kollision bereits erfolgt. Bei zunehmender Entfernung nehmen die Kosten monoton ab und geben damit das geringer werdende Kollisionsrisiko wieder.

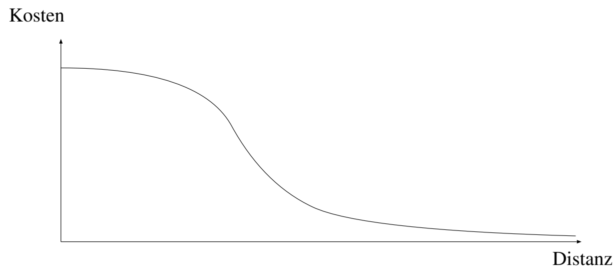

Abbildung A.7: Monoton fallende Entfernungskosten

Die Entfernungskosten bewerten nur die Kosten, in einer bestimmten Entfernung zu einem Hindernis zu stehen. Benötigt wird aber ein Kostenmaß für die Bewertung eines Pfades.

Die Pfadkosten gewinnt man durch Summation der Entfernungskosten entlang des Pfades. Die Summe ist damit das akkumulierte Kollisionsrisiko des Pfades. Es werden also Pfade mit geringem Kollisionsrisiko bevorzugt. Das Ziel, einen möglichst kurzen Pfad zu wählen, wird nur implizit berücksichtigt. Kurze Pfade haben in der Regel ein geringeres akkumuliertes Risiko.

A.2.2 Minimaler Sicherheitsabstand

Eine Bedingung, die ein Entwickler immer gewährleistet sehen will, ist die unbedingte Einhaltung eines minimalen Sicherheitsabstandes. Dies wird dadurch erreicht, daß Entfernungen unterhalb des Sicherheitsabstandes mit den größten darstellbaren Kosten belegt werden, d.h. jeder Pfad ohne Unterschreitung des Sicherheitsabstands ist günstiger als eine Verletzung des Sicherheitsabstands.

A.2.3 Erwünschter Ausweichradius

Neben dem Sicherheitsabstand der unbedingt eingehalten werden soll, möchte ein Entwickler den normalen Ausweichradius um ein ansonsten freistehendes Hindernis beeinflussen.

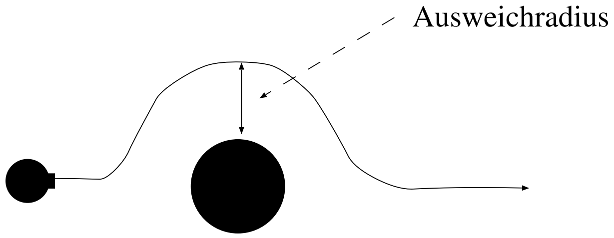

Abbildung A.8: Erwünschter Ausweichradius

Die Entfernungskosten sollten so definiert sein, daß bei einem freistehenden Hindernis der Ausweichradius nicht übermäßig groß wird. Wenn beispielsweise die Steigung der Entfernungskostenfunktion ab einer bestimmten Entfernung echt geringer als -1 ist, d.h die Kosten schneller fallen als die Entfernung zunimmt, so hätte der kostengünstigste Pfad einen unendlich großen Ausweichradius. Unter der Voraussetzung, daß die Entfernungskosten echt positiv sein müssen, könnte die Kostenfunktion zwar nicht unendlich lange mit einer solchen Steigung abfallen, d.h. der Radius des Kostengünstigsten Pfades wäre beschränkt, aber könnte durch Umdefinition der Kostenfunktion beliebig groß werden.

Es ist also vernünftig, den Ausweichradius über die Definition der Kostenfunktion sowohl nach oben, wie nach unten zu beschränken.

Es nun wird eine geringfügig vereinfachte Situation betrachtet. Wie im Bild A.9 dargestellt, wird angenommen, daß der Pfad bis zu einer gewissen Entfernung r an das Hindernis heranreicht und dann mit Radius r in einen Halbkreis um das Hindernis führt.

Das Ziel ist es, über die Definition der Entfernungskosten für einen gewünschten Radius r_{opt} die minimalen Pfadkosten zu garantieren. Für alle Radien r_1, r_2 mit $r_1 < r_{opt} < r_2$ muß also gelten:

$$\pi r_1 K(r_1) > \pi r_{opt} K(r_{opt}) < \pi r_2 K(r_2) \qquad (A1.1)$$

Dabei ist K die Entfernungskostenfunktion.

Die Ungleichung A1.1 kann nur dann erfüllt sein, wenn für die Kostenfunktion gilt:

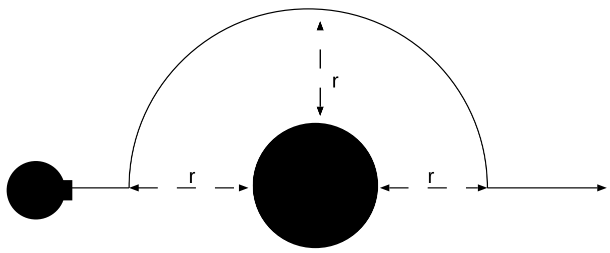

Abbildung A.9:

$$\forall d \in \mathbb{R}_{>0} \setminus \{r_{opt}\} : K(d) > \frac{r_{opt}}{d} \cdot K(r_{opt}) \qquad (A1.2)$$

In Bild A.10 ist eine Kostenfunktion mit der charakteristischen Delle bei der erwünschten Ausweichdistanz abgebildet. Die gepunktete Kurve deutet dabei die Schranke an, die links und rechts von der Stelle r_{opt} überschritten werden muß, damit die Kosten für den erwünschten Pfad minimal sind.

Wenn in Ungleichung A1.2 anstelle des $>$ ein $=$ stehen würde, so wären die akkumulierten Kosten für einen Halbkreispfad unabhängig vom Radius immer identisch.

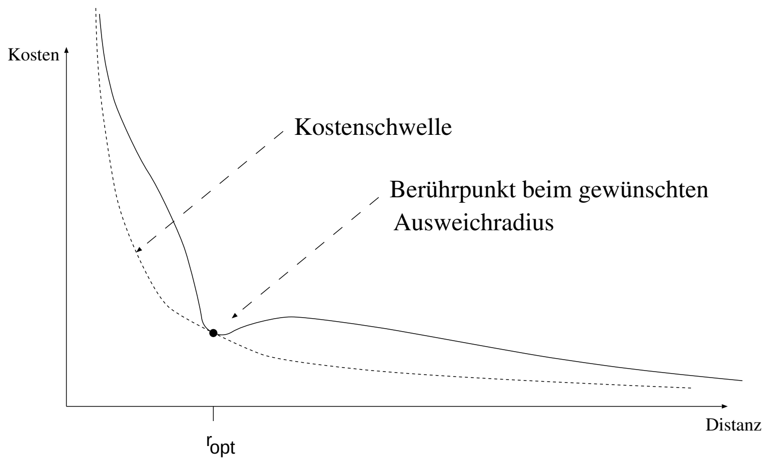

Abbildung A.10: Entfernungskosten mit der charakteristischen Delle beim erwünschten Ausweichradius

Literaturverzeichnis

[1] Dieter Fox, Wolfram Burgard, Sebastian Thrun *The Dynamic Window Approach to Collision Avoidance*, IEEE Robotics & Automation Magazine, 4(1), March 1996

[2] W. Burgard, D. Fox, D. Henning, T. Schmidt *Position Tracking with Position Probability Grids*, Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots, IEEE Computer Society Press, 1996

[3] Thomas Edlinger, Gerhard Weiß *Exploration, Navigation and Self-Localization in an Autonomous Mobile Robot*, ftp://ftp.uni-kl.de/reports_uni-kl/computer_science/mobile_robots/1995/papers/Edlinger.Exp_Nav_Loc.ps.gz

[4] Jiming Liu *Spatial Reasoning About Robot Compliant Movements and Optimal Paths in Qualitatively Modeled Enviroments*, The International Journal of Robotic Research, Vol. 15, April 1996, p. 181-210

[5] Long-Ji Lin *Reinforcement Learning for Robots Using Neural Networks*, CMU-CS-93-103

[6] H.P. Moravec *Sensor fusion in certainty grids for mobile robots*, AI Magazine, S.61-74, Sommer 1988

[7] Jun Tani, Naohiro Fukumura *Learning Goal-Directed Sensory-Based Navigation of a Mobile Robot*, Neural Networks, Vol 7, pp 553-563, 1994

[8] A. David Redish, David S. Touretzky *Modeling Interactions of the Rat's Place and Head Direction Systems*, Advances in Neural Information Processing Systems (8), MIT Press

[9] Hank S. Wan, David S. Touretzky, A.David Redish *Computing Goal Locations from Place Codes*, Proceedings of the 16th Anual Conference of the Cognitive Science Society, August 13-16, Atlanta, GA

[10] Constantin von Altrock *Fuzzy Logic*, Band 1, R. Oldenbourg Verlag

[11] Sebastian Thrun, Arno Bücken, Wolfram Burgard, Dieter Fox, Thorsten Fröhlinghaus, Daniel Henning, Thomas Hofmann Michael Krell, Timo Schmidt *Map Learning and High-Speed Navigation in RHINO*, To appear in: AI-based Mobile Robots: Case studies of successful robot systems, MIT Press, D. Kortenkamp, R.P. Bonasso, and R.R. Murphy(eds), invited paper

[12] O. Khatib *Real time obstacle avoidance for manipulators and mobile robots*, International Journal of Robotics Research, 5(1):90-98, Spring 1986

[13] Charles W. Warren *Global Path Planning Using Artificial potential Fields*, Proceedings of the 1989 IEEE International Conference on Robotics and Automation, p. 316-321

[14] Reza Shahidi, Mark Shayman *Mobile Robot Navigation Using Potential Functions*, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, p. 2047-2053

[15] T. Skewis, J. Evans, V. Lumelsky, B. Krishnamurthy, B. Barrows *Motion Planing for a Hospital Transport Robot*, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, p. 58-63

[16] Jen-Hui Chuang, Narendra Ahuja *Path Planning Using The Newtonian Potential*, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, p. 558-559

[17] Elon Rimon *A Navigation Function for Simple Rigid Body*, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, p. 546-551

[18] Yun-Hui, Suguru Arimoto *Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots*,Proceedings of the 1991 IEEE International Conference on Robotics and Automation,p. 312-317

[19] Brad Paden, Alistair Mees, Mike Fisher *Path Planing Using a Jacobian-Based Freespace Generation Algorithm*, Proceedings of the 1989 IEEE International Conference on Robotics and Automation, p. 1732-1737

[20] Hiroshi Nonorio, Tomohide Naniwa, Surguru Arimoto *A Feasible Motion-Planning Algorithm for a Mobile Robot on a Quadtree Representation*, Proceedings of the 1989 IEEE International Conference on Robotics and Automation, p. 327-332

[21] V. Lumelsky, A. Stepanov *Path planing strategies for a point mobile automaton moving admidst unknown obstacles of arbitrary shape*, Algorithmica 3(4), 1987

[22] N.J. Nilson. *Principles of Artificial Intelligence*,Springer Publisher, Berlin, New York, 1982

[23] Anil K. Jain *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, N.J. 07632

Anhang B

B.1 Eidesstattliche Erklärung

Ich Harald Stark geboren am 18.02.1970 erkläre hiermit, daß ich diese Arbeit ohne fremde Hife und nur unter Angabe der verwendeten Literatur erstellt habe.