Ein Beitrag zur Entwicklung von Strategien zur koordinierten Steuerung autonomer mobiler Multirobotersysteme

Von der Fakultät Informatik und Automatisierung
der Technischen Universität Ilmenau
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
(Dr.-Ing.)

von
Dipl.-Ing.
Uwe Altenburg
geb. am 27.Mai 1970 in Sömmerda

Gutachter: Univ.-Prof. Dr.-Ing. habil. J. Wernstedt, TU-Ilmenau,
Univ.-Prof. Dipl.-Ing. Dr. med. (habil.) H. Witte, TU-Ilmenau,
Prof. Dr.-Ing. habil. S. Bergmann, FH Frankfurt am Main

Tag der Einreichung: 11.12.2003

Tag der wissenschaftlichen Aussprache: 05.07.2004

Verfahrensnr.: IA 124

“Intelligence is in the eye of the observer.”
Rodney A. Brooks

Vorwort

Die vorliegende Arbeit entstand während meiner Zeit als externer Promotionsstudent an der Fakultät Informatik und Automatisierung, Fachbereich Systemanalyse der Technischen Universität Ilmenau.

Mein besonderer Dank gilt dem Fachbereichsleiter, Herrn Univ.-Prof. Dr.-Ing. habil. J. Wernstedt, der diese Arbeit ermöglicht und mit großem Interesse gefördert hat. Ebenso möchte ich mich bei den Mitarbeitern des Fachbereichs, insbesondere bei Herrn Dr.-Ing. Th. Rauschenbach sowie bei Herrn Dipl.-Ing. Th. Glotzbach und Herrn Dipl.-Ing. T. Pfützenreuter, für die entgegengebrachte Unterstützung sowie die zahlreichen Diskussionen bedanken.

Ferner möchte ich mich bei meinen Kollegen der Firma Jetter AG, stellvertretend bei Herrn Joachim Kittelberger, für die freundliche Unterstützung und das entgegengebrachte Interesse bedanken. In diesem Sinne danke ich auch dem Ingenieurbüro Wächter, namentlich Herrn Dr. Krieger, für die Motivation und Zusammenarbeit.

Frau Anita Semmler danke ich für ihre Geduld beim Korrekturlesen.

Schließlich möchte ich mich ganz herzlich bei meiner Familie bedanken, insbesondere bei meiner Ehefrau und meinen beiden Töchtern, die während der Zeit dieser Arbeit so manchen Abend auf mich verzichten mussten. Nicht zuletzt danke ich auch meinen Eltern, die mir stets mit Rat und Tat behilflich waren sowie meinem Bruder, ohne dessen Roboter diese Arbeit nicht entstanden wäre.

Sömmerda, im Sommer 2003 Uwe Altenburg

Kurzfassung

Die koordinierte Bewegungssteuerung von Multirobotersystemen ist eine vielschichtige und komplexe Aufgabe, für die in der Literatur unterschiedliche Lösungsansätze beschrieben sind. Oft steht dabei die Kollisionsvermeidung oder Auflösung von Konfliktsituationen an Kreuzungspunkten im Vordergrund. In der vorliegenden Arbeit wird insbesondere eine Strategie vorgestellt, mit der es möglich ist, eine Gruppe von Robotern so zu steuern, dass diese sowohl eine Formation weitestgehend beibehalten als auch mögliche Kollisionen mit Hindernissen vermeiden. Die Strategie umfasst im Wesentlichen die Modellbildung sowie die Missions- und die Verhaltensplanung. Grundlage bildet dabei eine hybride Steuerungsarchitektur, die sich aus einer globalen kartenbasierten Planung und einer lokalen reaktiven Ausführung zusammensetzt. Die Planung erfolgt offline und liefert im Ergebnis eine optimale Route für eine vorgegebene Formation (*Missionsplanung*). Die Route wird dahingehend erweitert, dass einzelnen Robotern günstige Steuerverhalten vorgeschlagen werden (*Verhaltensplanung*). Die Aktivierung dieser Steuerverhalten erfolgt dann lokal an zuvor festgelegten Synchronisationspunkten entlang der geplanten Route unter Ausnutzung einer hierarchischen Struktur innerhalb der Gruppe.

Ausgehend von einer geeigneten Modellierung eines solchen Multirobotersystems wird eine konkrete Implementierung dieser Planungsstrategie in allen Schritten vorgestellt. Beginnend mit einer Bildanalyse zur Erstellung einer Karte wird darauf aufbauend ein adaptives Routenplanungsverfahren für die gesamte Formation vorgestellt. Die geplante Route wird im Nachgang analysiert, um der jeweiligen Hindernissituation entsprechend günstige Verhalten für die einzelnen Roboter zu bestimmen. Anhand der Grenzen dieser konkreten Implementierung wird eine alternative Variante abgeleitet, die die Missions- und die Verhaltenplanung integriert.

Schließlich wird eine Experimentalplattform mit Kleinstrobotern beschrieben, die im Verlauf der Arbeit entstanden ist und für Versuche mit Multirobotersystemen geeignet erscheint. Es wird die hardwarseitige Umsetzung dokumentiert und vor allem die dazu geschaffene Steuersoftware erläutert, zu der auch eine Simulationsumgebung gehört. In diesem Zusammenhang wird detailliert auf eine Lösung der globalen kameragestützten Positionsbestimmung für kleine Roboter eingegangen. Dabei wird ein Verfahren vorgeschlagen, das im Gegensatz zur üblichen Farberkennung auf der Basis von Binärbildern arbeitet und Farbinformationen lediglich zur Unterscheidung der Roboter sowie zur Erkennung der Hindernisse nutzt. Die Anwendung von Kamerabildern als Ergänzung der lokalen Sensorik eines Roboters wird schließlich kurz angeschnitten.

INHALTSVERZEICHNIS

1 Einleitung

1.1 Multirobotersysteme

Im Juli 1997 erreichte Sojourner, ein Forschungs-Roboter der NASA, die Marsoberfläche. Trotz anfänglicher technischer Schwierigkeiten konnte Sojourner aufschlussreiche Daten, aber vor allem auch Bilder zur Erde übertragen. Allein die Perspektive dieser Aufnahmen zog den Betrachter in den Bann einer scheinbar unerreichbaren Welt.
Anhand dieses populären Beispiels lassen sich mehrere Gründe ableiten, die für den Einsatz von Multirobotersystemen sprechen. In erster Linie ist natürlich die größere Redundanz hinsichtlich eines Totalausfalls zu nennen. Es ergeben sich aber weitere Vorteile, die nachfolgend an unterschiedlichen Anwendungsfällen aufgezeigt werden sollen.
Im Falle von Inspektions- und Wartungsaufgaben ist zum Beispiel die Möglichkeit unterschiedlicher Kameraperspektiven bzw. Beleuchtungswinkel mitunter entscheidend. Nicht immer sind Schwachstellen eindeutig lokalisierbar, schon gar nicht, wenn Beleuchtung und Kamera aus dem selben Winkel auf ein Objekt gerichtet sind. Man denke hier an Materialfehler wie Verformungen oder Risse, die erst durch eine Schattenwirkung deutlich werden. In [Asama, Fukuta, Arai u. Endo, 1996 S.50 ff.] sind detailliert Anordnungen beschrieben, bei denen durch geeignete Platzierung mehrerer Kameraroboter entsprechende Untersuchungen im Umfeld von Industrieanlagen realisierbar sind. Schließlich sind auch bei der Durchführung von Handhabe- oder Manipulationsaufgaben unterschiedliche Blickwinkel von Vorteil, um sowohl einen Überblick zu erhalten, als auch den konkreten Vorgang beobachten zu können.
Im Bereich der Serviceroboter spielt unter anderem die effektive Reinigung großer Flächen eine Rolle. Immer wieder werden solche Probleme in Wettbewerben ausgeschrieben, die teils spielerisch, vorwiegend aber vor akademischem Hintergrund ausgetragen werden. Aktuell werden dazu einzelne Roboter eingesetzt, die Gegenstände einsammeln oder zumindest auf einen Haufen schieben, wobei aber gerade hier die koordinierte Steuerung einer Gruppe von Robotern offensichtliche Vorteile bietet.
Im Bereich der Unterwassererkundung, sei es zu Forschungszwecken oder beispielsweise bei der Seekabelverlegung, der Erdölgewinnung oder der Begleitung von U-Booten, ist der Einsatz mehrere Roboter schon wegen der begrenzten Reichweite der Sensoren und nicht zuletzt wegen der höheren Ausfall-wahrscheinlichkeit interessant.
Nicht unerwähnt darf hier der militärische Aspekt bleiben. Die Kampfmittelräumung sieht sich heutzutage mehr denn je mit dem Problem der Landmienen konfrontiert; die binnen kürzester Zeit ausgebracht werden können, deren Beseitung aber weitaus aufwendiger ist. Wäre der Einsatz von Robotern hier nicht geradezu optimal? Aber wer möchte sich darauf verlassen, dass „zufällig“ umherfahrende – wenn auch mehrere – Roboter wirklich jeden Fleck abgesucht haben?

Die hier quasi willkürlich aufgeführten Aufgabenstellungen bzw. Einsatzfälle können mit einer begrenzten Anzahl identischer oder auch spezialisierter Roboter (*heterogene Multirobotersysteme*) abgedeckt werden. Letztere bieten die größte Flexibilität und Ausfallsicherheit bei vergleichsweise geringer Komplexität und geringen Kosten des Einzelroboters.
Ein Punkt, der bisher jedoch noch nicht betrachtet wurde, ist die Koordination der Roboter untereinander. Die Reduzierung der Komplexität der Einzelkomponenten führt zu einer Erhöhung der Anforderungen an die Koordination. Die daraus resultierenden Probleme sollen im folgenden Abschnitt beleuchtet werden.

1.2 Problematik der Koordination

Das gleichzeitige Agieren mehrerer Roboter im selben Arbeitsbereich wirft zunächst das klassische Problem der Ressourcenverteilung auf. Der freie Arbeitsbereich kann dabei als eine globale Ressource betrachtet werden, deren Verteilung auf mehrere Roboter einer Koordination bedarf. Die Art der Koordination – lokal oder global – ist mit den Zugriffsmechanismen in Rechnernetzen vergleichbar. Das Transportmedium ist dort die global verfügbare Ressource. Lokal ausgerichtete Zugriffsverfahren erlauben kurze, aber nicht determininistische Reaktionszeiten – sie konkurrieren quasi um die zur Verfügung stehende Ressource. Global gesteuerte Zugriffsverfahren hingegen ermöglichen die exakte Bestimmung einer Zykluszeit, haben aber bei vielen Teilnehmern große Reaktionszeiten zur Folge.
In der Robotik werden entsprechend verhaltens- und modellbasierte Steuerungen unterschieden. Zusätzlich zur eindimensionalen zeitlichen Zuteilung einer Ressource, muss hier auch die räumliche Verteilung berücksichtigt werden. Verhaltensbasierte Ansätze lösen nur räumlich begrenzte Konflikte auf, während mit Weltmodellen die Beschreibung aller relevanten Zusammenhänge angestrebt wird. In der Folge liefern verhaltensbasierte Ansätze schnelle Lösungen, die aber im Kontext der Aufgabe nicht optimal sind. Umgekehrt liefern Weltmodelle optimale Lösungen, die aber keine oder nur wenig Dynamik berücksichtigen.

Im Sinne des Einsatzes mehrerer Roboter zur Lösung einer gemeinsamen Aufgabe kommt der Koordination eine weitere entscheidende Rolle zu. Das gemeinsame Handeln der auch als *Agenten*[1] bezeichneten autonomen Systeme muss aufeinander abgestimmt erfolgen. Die Koordination umfasst deshalb auch: die Bildung von Teilaufgaben bzw. Teilzielen, die Übertragung an die entsprechenden Roboter und einen Informationsaustausch zur Synchronisierung der Roboter. Diese Funktionen müssen systemübergreifend realisiert werden, da einzelne Agenten nicht über die nötige Menge der Informationen verfügen.
An dieser Stelle dürfen Experimente mit rein verhaltensorientierten Ansätzen nicht unerwähnt bleiben, bei denen eine kleine Anzahl festprogrammierter Roboter zufällig verteilte Objekte scheinbar koordiniert an wenigen Positionen ansammeln [Martinoli, Mondada, 1998]. Diese Demonstrationen implizieren den Schluss, dass es nicht zwingend einer übergeordneten Koordination, zumindest aber keiner Kommunikation bedarf, um verteilt komplexe Aufgaben zu lösen. Tatsächlich muss man jedoch fragen, ob die realisierten Verhalten nicht das Ergebnis einer vorangegangenen Planung (oder

[1] Agenten sind autonome Einheiten (Objekte im Sinne eines Modells), die mit ihrer Umwelt interagieren und dabei zielgerichtet aktiv werden. (vgl. [Odell, 2002])

eines andauernden Lernprozesses) sind, und ob sich umgekehrt diese Verhalten reproduzierbar aus einer entsprechenden Aufgabenstellung ableiten lassen? Außerdem muss man feststellen, dass die einmal bewegten Objekte zu Trägern von Informationen werden. Die gewonnenen Informationen über Anzahl und momentane Position der Objekte werden somit indirekt an andere Roboter weitergegeben. Die Koordination erfolgt also schon systemweit. Es zeigt sich aber, dass es keiner physischen Instanz bedarf, die die Aufgabe der Koordination explizit übernimmt. Diese als *Emergence*[2] [Arkin, 1998] bezeichnete Erscheinung in Multiagentensystemen ist Gegenstand zahlreicher Untersuchungen und gewinnt mit steigender Zahl der Agenten zunehmend an Bedeutung [Johnson, 2001].

Offensichtlich führt die alleinige Anwendung von weltmodell- oder verhaltensmodell-basierten Ansätzen zu unvollständigen Lösungen. Vielmehr ist die Kombination der Vorteile beider Ansätze für ein System gewinnbringend.

1.3 Zielsetzung und Aufbau der Arbeit

Im Verlauf der vorliegenden Arbeit soll eine Strategie erläutert werden, mit deren Hilfe die Steuerung kleiner Multirobotersysteme in der Ebene (R^2) realisiert werden kann. Ziel ist die zentralisierte Bahnplanung für eine begrenzte Anzahl mobiler Roboter in unstrukturierter Umgebung. Dabei sollen die Roboter eine definierbare Formation (relative Lage zueinander) weitestgehend beibehalten, ohne jedoch untereinander oder mit Hindernissen zu kollidieren. Nach Vorgabe von Start- und Zielkonfiguration sowie der Formation soll das Verfahren die Steuerung der einzelnen Fahrzeuge übernehmen. Entsprechend den Gegebenheiten kann die Formation zeitweilig abgeändert oder ganz aufgelöst werden. Dabei soll die Autonomie der Einzelsysteme ausgenutzt werden, um die Zahl der Steuerkommandos zu reduzieren. Mehr noch sollen die Roboter als ein System gesteuert werden. Wie bereits in Abschnitt 1.2 angedeutet, wird eine umfassende Lösung der sich ergebenden Probleme schwerlich möglich sein. Deshalb werden an geeigneter Stelle auch die Grenzen des Verfahrens in Bezug auf konkrete Einsatzfälle dargestellt.

Ausgehend von einer kurzen Betrachtung der derzeit verwendeten Verfahren zur Bahnplanung für mobile Roboter in statischer als auch dynamischer Umgebung wird in Kapitel 2 insbesondere auch die verhaltensbasierte Robotik betrachtet. Nach einer Zusammenfassung der Möglichkeiten und Grenzen wird anschließend im Kapitel 3 eine mehrstufige Strategie zur Bewegungsplanung in Multirobotersystemen vorgestellt. Die praktische Umsetzung erfolgt auf einer Experimentalplattform, die im Kapitel 4 näher erläutet werden soll. Die dazu geschaffene Softwareumgebung wird anschließend im Kapitel 5 beschrieben. Hier werden aber vorrangig die Achitektur beziehungsweise deren Schnittstellen betrachtet. Im Kapitel 6 werden schließlich die Ergebnisse anhand von exemplarischen Szenarien diskutiert, wobei auch auf einzelne Detailprobleme eingegangen wird. Die Arbeit schließt mit einer Einordnung und Zusammenfassung sowie einem Ausblick auf weitere Überlegungen.

[2] „Emergence is what happens when an interconnected system of relatively simple elements self-organizes to form more intelligent, more adaptive higher-level behavior.“ [Johnson, 2001]

2 Bewegungssteuerung mobiler Roboter

2.1 Klassische routenbasierte Ansätze

Bevor eine Darstellung der derzeit verwendeten Konzepte erfolgt, sollen in diesem Abschnitt einige Grundelemente aufgezeigt werden. Beim Studium der Veröffentlichungen zum Thema Bewegungsplanung/Routenplanung findet man ähnliche Herangehensweisen. In [Arkin, Bekey, 1997] wird deshalb der Versuch unternommen alle möglichen Arten der Bewegungsplanung, insbesondere für kooperierende Roboter, abzuleiten, zu klassifizieren und wo bereits bekannt, auf ausgewählte Lösungen zu verweisen. Damit ist quasi eine gemeinsame Basis beschrieben, anhand derer neue Lösungen erarbeitet werden können bzw. die eine Einordnung bestehender Ansätze erlaubt. Diese eher abstrakte Annäherung an das Problem soll hier durch konkrete Teillösungen ergänzt werden. Die Grundelemente sind dabei für Einzelroboter bzw. für Multirobotersysteme weitestgehend gleich, weshalb in den folgenden Abschnitten nicht explizit unterschieden wird. An den betreffenden Stellen werden Unterschiede oder unterschiedliche Verwendung herausgestellt.

Die allgemeine Routenplanung stellt sich als mehrstufiger Prozess mit folgenden Schritten dar:

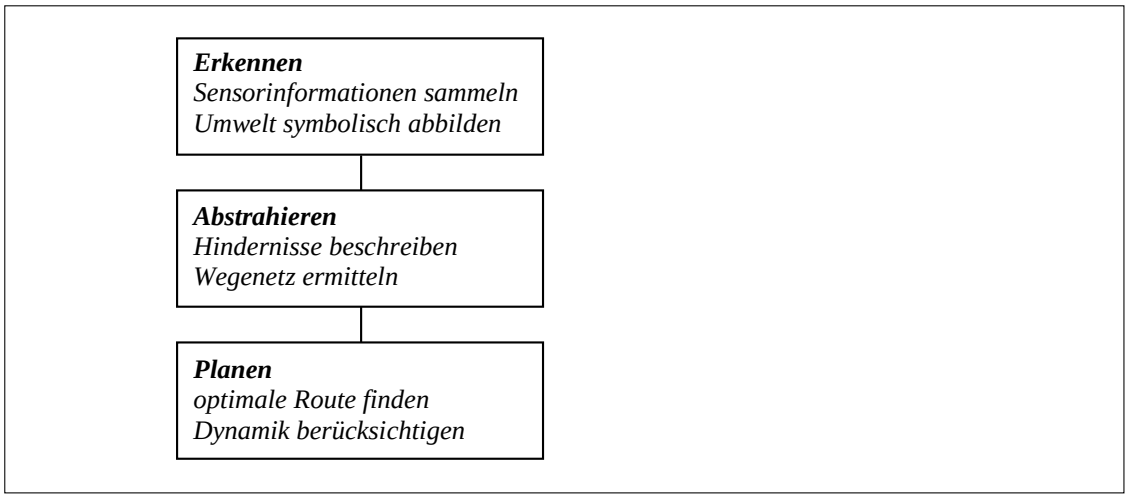


Abbildung 2.1 Vertikaler Planungsprozess

Dabei soll die erste Stufe *Erkennen* in den nächsten Abschnitten nicht weiter betrachtet werden. Das Spektrum der Verfahren reicht von Tastsensor über Ultraschall- bis hin zur Bilderkennung und liefert eine mehr oder minder vollständige Abbildung der Umwelt. Alle nachfolgend beschriebenen Verfahren und Algorithmen nutzen eine *symbolische* Darstellung der Umwelt, unabhängig davon, wie sie gewonnen wurde.

2.1.1 Standardproblem

Das Standardproblem der Bewegungsplanung für mobile Roboter ist die Suche nach einem möglichst optimalen Weg von einer Start- zu einer Zielkonfiguration innerhalb eines Arbeitsraumes. Dabei ist eine Kollision mit statischen und dynamischen Hindernissen zu vermeiden.

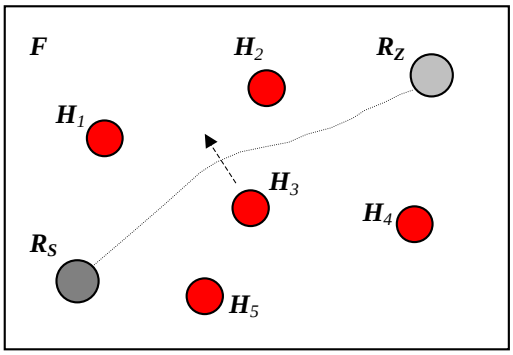


Abbildung 2.2 Standardproblem der Bewegungsplanung für mobile Roboter

Warum ist das ein Problem? Es handelt sich um einen Planungsprozess, dessen Komplexität mit der Anzahl der Freiheitsgrade steigt. Ein Roboter besitzt in der Ebene 3 Freiheitsgrade, die zu einem Zeitpunkt *t* als Konfiguration *K(t)* (2.1) des Roboters bezeichnet werden. Es gilt somit:

$$K(t) = \begin{bmatrix} x(t) \\ y(t) \\ \varphi(t) \end{bmatrix}. \tag{2.1}$$

Bei der Planung einer Trajektorie, das heißt der zeitabhängigen Darstellung der Konfigurationen, müssen die Freiheitsgrade so bestimmt werden, dass zusätzliche Randbedingungen stets eingehalten werden. Im einfachsten Fall eines rotationssymmetrischen holonomen Roboters gilt daher zusätzlich, dass kein Punkt *P* auf dem Durchmesser *D* des Roboters gleichzeitig Teil der Menge der Hindernispunkte sein darf. Hier zeigt sich, dass insbesondere die Randbedingungen aufwendiger zu testen sind, als die Konfigurationen selbst. Im Allgemeinen werden daher Roboter und Hindernisse durch geometrische Formen, wie Kreise oder Polynome, abstrahiert, da hierfür Verfahren existieren, um Schnittpunkte und Abstände berechnen zu können. Für die Planung selbst ist eine Dekomposition des Problems sinnvoll, um die eigentliche Wegsuche nicht mit Tests auf Kollisionen zu belasten.

2.1.2 Darstellung im Konfigurationsraum

Da insbesondere die Routenplanung in unstrukturierter Umgebung eine sehr aufwendige Modellierung des Arbeitsraumes zur Folge haben kann, wird in erster Linie eine Reduzierung der Komplexität angestrebt. Die analytische Berechnung der Abstände zwischen allen Punkten der Hülle eines Roboters zu allen möglichen Hüllen

der Hindernisse während der Bahnplanung ist praktisch nicht durchführbar. Das Mittel der Wahl ist deshalb die *Diskretisierung* des Arbeitsraumes. Alle Hindernisse H_i werden vereinfacht durch Polygone beschrieben. Der oder die Roboter werden lediglich durch Referenzpunkte abstrahiert. Damit diese Vereinfachung möglich wird, ist die Transformation des Arbeitsraumes in einen *Konfigurationsraum C* erforderlich. Der Konfigurationsraum ordnet jeder möglichen Konfiguration $K = [x,y,\phi]^T$ des Roboters einen Punkt zu. Jeder Punkt, der innerhalb von C bezüglich H_i nicht kollisionsfrei ist, wird als verboten (d.h. nicht einnehmbar) markiert. Die verbleibende Menge der Punkte im Konfigurationsraum wird *Freiraum F* genannt.
Zu beachten bleibt, dass der Konfigurationsraum die selbe Dimension wie die Anzahl der Freiheitgrade des Roboters aufweist. Weiterhin besteht das Problem mögliche Kollisionen des Roboters mit den Hindernissen zu erkennen – es wurde nur aus der Routenplanung in die Transformation in den Konfigurationsraum vorverlegt.

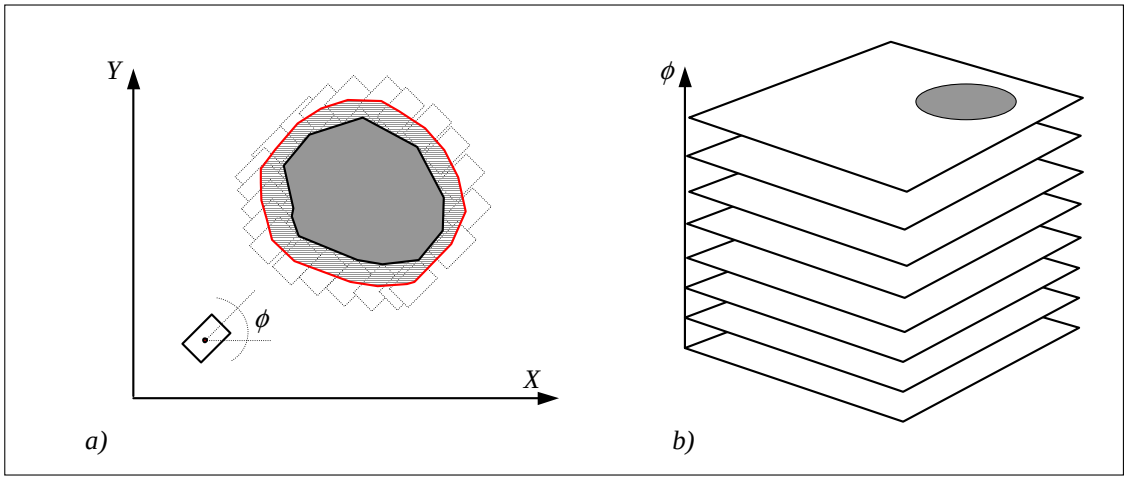


Abbildung 2.3 Transformation des Arbeitsraumes in den Konfigurationsraum
a) Konfigurationsebene (ϕ=constant)
b) Konfigurationsraum

2.1.3 Abstraktion eines Wegenetzes

Praktisch gibt es zwei Arten den entstandenen Freiraum *F* zu abstrahieren und so die Basis für eine anschließende Routenplanung zu schaffen. Die Art und Weise, wie man zur einen oder anderen Darstellung gelangt, ist sehr verschieden und selbst Gegenstand der Forschungen.

1.) Die einfache *Rasterung* des Arbeitsraumes in Segmente, die entweder vollständig frei oder vollständig belegt sind, ermöglicht die Anwendung von Suchalgorithmen auf der Basis von künstlichen Potentialfeldern (*Lee-Algorithmus*) oder von Suchbäumen (*A*-Algorithmus*) zur Routenplanung.

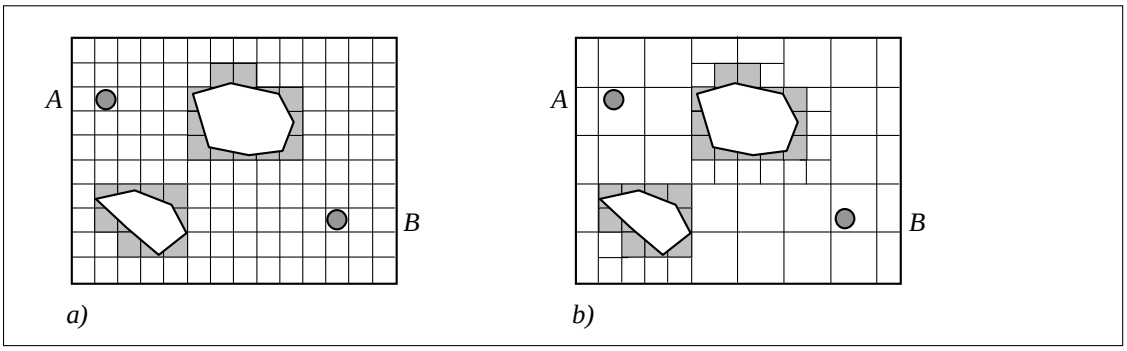


Abbildung 2.4 Segmentierung des Freiraums
a) äquidistant
b) dynamisch

Die Vorteile dieser Darstellung liegen in der einfachen Gewinnung durch äquidistante Rasterung entlang der Koordinatenachsen. Auf diese Weise werden aber sehr viele Raster erzeugt, die später nicht zur Lösung des Wegeproblems beitragen. Varianten dieser Darstellung bedienen sich deshalb einer *dynamischen Rasterung*. Dabei wird zu Beginn ein sehr grobes Raster verwendet. Lediglich um Hindernisse herum wird ein feineres Raster gelegt. Während der Suche nach einer Route von *A* nach *B* wird das Raster verfeinert, sofern noch kein Weg existiert.

2.) Die topologische Darstellung des Freiraums *F* in Form von *Graphen* ermöglicht zudem die Berücksichtigung von Wertigkeiten entlang der Kanten. Auch hier sind verschiedene Verfahren beschrieben, einen optimalen Weg zu ermitteln (*Dijkstra's Algorithmus*). Die Gewinnung eines geeigneten Graphen ist unterschiedlich aufwändig. Zunächst muss eine repräsentative Menge kollisionsfreier Knoten innerhalb des Freiraumes ermittelt werden. Dazu können beispielsweise die Eckpunkte der Hindernis-Polygone verwendet werden, um einen *Visibility Graph* zu erstellen. Insbesondere bei komplexen Hindernisformen bieten sich aber auch *Voronoi Diagramme* an, die durch einen iterativen Prozess ermittelt werden. Die besondere Eigenschaft eines solchen Diagramms ist die, dass Kanten immer den größtmöglichen Abstand zu Hindernissen haben. Im Falle einer Passage führt der Weg folglich entlang der Mittellinie.

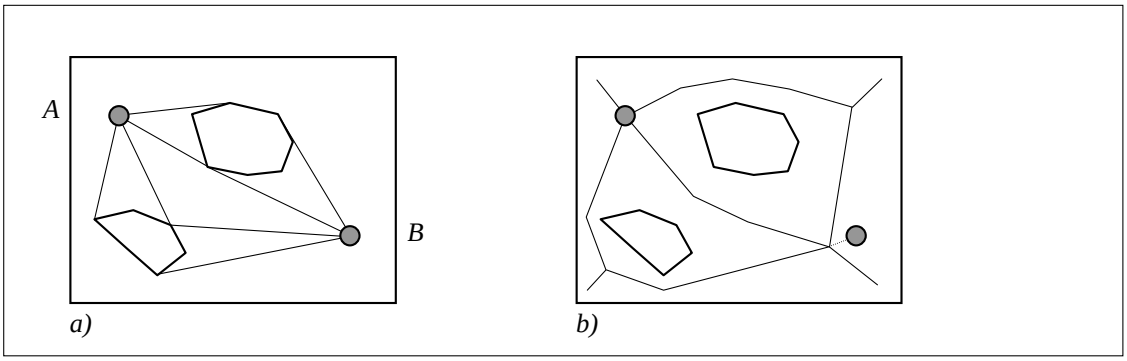


Abbildung 2.5 Wegenetze im Freiraum in Form von Graphen
a) Visibility Graph
b) Voronoi Diagramm

Die so entstandenen Knoten werden kollisionsfrei miteinander verbunden. Sofern sich die Start- und Zielkonfiguration des Roboters nicht mit dem Graphen decken, muss eine entsprechende Auf- und Abfahrt getrennt berechnet werden.
Der Vorteil dieser Darstellung liegt in der Nutzung der Graphentheorie. Damit stehen insbesondere auch Methoden für die Umformung bzw. die Kombination mehrerer Graphen (Multirobotersysteme) zu einem Supergraphen [Laumond, 1998 S.255 ff.] zur Verfügung. Die Aufgabe der Routenplanung ist dann, die Knoten und Kanten zu ermitteln, die den gewünschten Weg ergeben.

Bei näherer Betrachtung ist die Variante in 1.) ein Spezialfall der Graphendarstellung. Alle Rasterelemente können ebenso als Knoten angesehen werden, von denen jeder (außer Randknoten) jeweils genau 8 Nachbarknoten (in R^3 schon 26) besitzt, zu denen jeweils eine Kante führt. Bei gleicher Länge aller Kanten ergibt sich die Weglänge zwischen zwei Positionen aus der Anzahl der übersprungenen Kanten.

2.1.4 Bestimmung einer optimalen Route

Nachdem ein Wegenetz in Form eines Graphen oder zumindest ein segmentierter Freiraum gebildet wurde, erfolgt die eigentliche Routenplanung. Im Sinne einer optimalen Route ist im Allgemeinen die kürzeste oder schnellste Verbindung gesucht, aber optimal kann auch die geringste statistische Kollisionswahrscheinlichkeit in Bezug auf dynamische Hindernisse [Kruse, 1998] bedeuten. Daher ist die Route gesucht, für die eine gewählte *Kostenfunktion E* mit:

$$E = f(e_1,e_2,...,e_n) \tag{2.2}$$

extrem wird. Zur Berechnung der Kostenfunktion werden den Teilstrecken einer Route Attribute e_i zugewiesen. Die Attribute kennzeichnen die Kosten für diese Teilstrecke in Bezug auf ein Optimierungskriterium. Eine Teilstrecke kann auch mehrere Attribute besitzen, die je nach Wahl des Kriteriums unterschiedlich gewichtet werden. Die Zuweisung der Attribute erfolgt sinnvollerweise im Verlauf der Wegenetzbildung, kann aber auch als Teil der eigentlichen Routenplanung angesehen werden. Graphenbasierte Verfahren sind im Hinblick auf Optimierung flexibler als Verfahren auf der Basis künstlicher Potentialfelder. Der Grund dafür liegt in der expliziten Zuordnung der Attribute als Eigenschaft der Kanten. Potentialfelder hingegen besitzen lediglich eine implizite Darstellung in Form von Gradienten. Ebenso ist die Zuordnung von Attributen zu Potentialfeldern kritisch, da lokale Extremstellen entstehen können, zumindest aber ist der Aufwand dafür höher.

2.1.5 Einbezug von Dynamik

Sobald in die Wegsuche dynamische Objekte einbezogen werden müssen, ist es erforderlich die Zeit als zusätzliche Koordinate einzuführen. Damit wird aus dem Konfigurationsraum C die *K-Raum-Zeit* C_T . Die dynamischen Hindernisse befinden sich darin zu allen Zeitpunkten t_i an anderen Positionen.

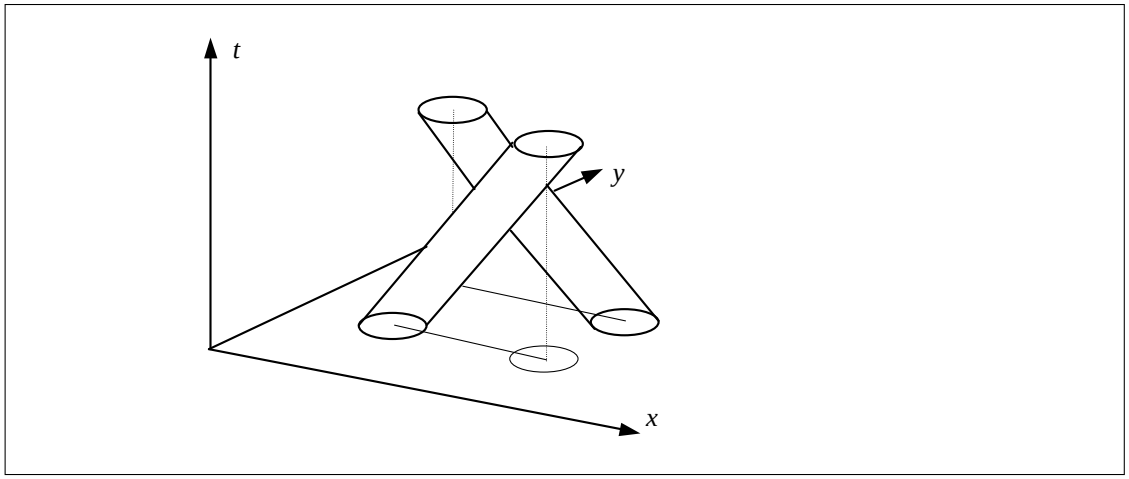


Abbildung 2.6 Darstellung der Raum-Zeit für dynamische Hindernisse in R_2

Durch Anwendung der bereits für statische Hindernisse beschriebenen Planungsverfahren kann auch hier eine Trajektorie gefunden werden. Die Trajektorie stellt die Erweiterung des Pfades in Bezug zur Zeit dar. Dadurch wird es möglich, ein Geschwindigkeitsprofil für den Roboter zu ermitteln. Ein Beispiel für eine solche Bewegungsplanung ist das Überqueren einer Straße. Dabei ist der Weg im einfachsten Fall eine Gerade zwischen der Start- und der Zielposition. Um eine Kollision mit den vorbeifahrenden Autos zu vermeiden, ist es jedoch erforderlich, ein geeignetes Geschwindigkeitsprofil zu bestimmen. Hieran lassen sich auch die beiden verwendeten Ansätze ableiten:

1.) Weg und Geschwindigkeitsprofil werden in separaten Schritten bestimmt

Der Weg ergibt sich aus der statischen Routenplanung ohne Einbezug der dynamischen Hindernisse. Die Verfahren wurden in 2.1.2 erwähnt. Das Geschwindigkeitsprofil wird anschließend in Bezug zu den dynamischen Hindernissen ermittelt.

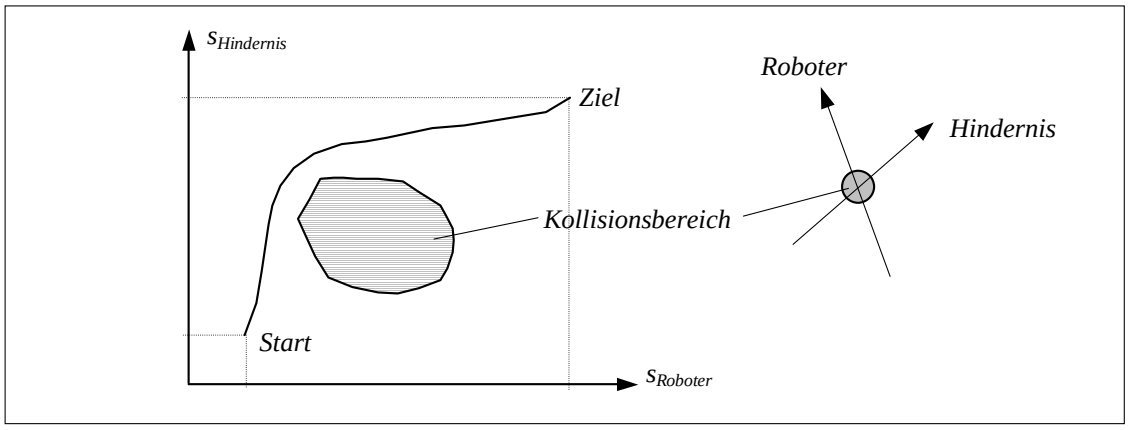


Abbildung 2.7 Bestimmung eines kollisionsfreien Geschwindigkeitsprofils

In Abbildung 2.7 wird das dafür verwendete Prinzip verdeutlicht. Die zurückgelegte Wegstrecke des Roboters wird über der Wegstrecke des Hindernisses aufgetragen. Es ergibt sich ein Kollisionsbereich, der durch eine geeignete Wahl der Streckenverhältnisse umgangen werden muss. Die Geschwindigkeit kann anschließend

durch Verhältnisbildung zur Geschwindigkeit des Hindernisses zu jedem Zeitpunkt berechnet werden.
Die Suche nach dem optimalen Geschwindigkeitsprofil erfolgt nach dem gleichen Verfahren wie bei der Suche nach der Route. Es können auch mehrere Hindernisse berücksichtigt werden. Dabei kann die Suche in einem Planungslauf unter Zugrundelegung eines mehrdimensionalen Raums erfolgen oder wie in [IASTED, 2001] beschrieben als sequenzielles Verfahren in einem zweidimensionalen Koordinatensystem.

2.) Weg und Geschwindigkeitsprofil werden gleichzeitig bestimmt.

Alternativ kann die Trajektorie in einem geometrischen Verfahren ermittelt werden.

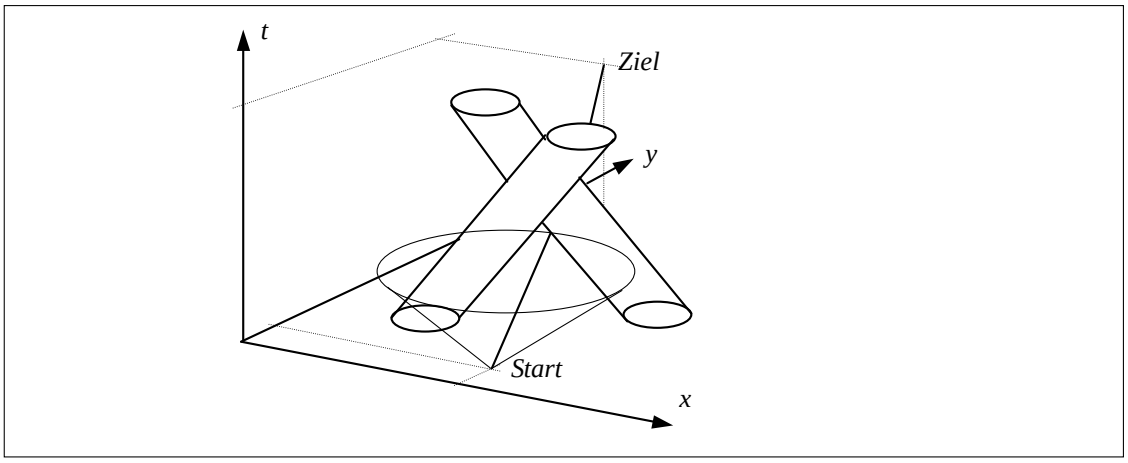


Abbildung 2.8 Berücksichtigung dynamischer Objekte in der Raum-Zeit

Dazu werden die Mantelflächen der Hindernisse in der Raum-Zeit berechnet. Bei kreisförmiger Grundläche der Objekte entstehen Kreiszylinder, die ensprechend ihrer Geschwindigkeit parallel verschoben sind. Die Trajektorie wird in einem successiven Verfahren so bestimmt, dass keine Schnittpunkte mit den Mantelflächen auftreten.

Zusammenfassend lässt sich feststellen, dass die getrennte Ermittlung von Weg und Geschwindigkeitsprofil einfacher ist. Damit sind aber nur Lösungen möglich, bei denen keine gegenseitige Beeinflussung der Ergebnisse besteht. So sind z.B. direkte Kollisionskurse zwischen Hindernis und Roboter nicht durch eine Variierung der Geschwindigkeit allein auflösbar *(deadlock)*. Im Gegensatz dazu ist die Modellierung der K-Raum-Zeit C_T für dynamische Hindernisse aufwendiger. Die Routenplanung kann dann aber als vollständig bezeichnet werden, d.h. sofern eine Trajektorie existiert, ist sie auch bestimmbar. Im Hinblick auf Multirobotersysteme ergeben sich weitere Probleme dadurch, dass nicht eine Trajektorie, sondern mehrere einander beeinflussende Trajektorien zu bestimmen sind.

2.2 Verhaltensbasierte reaktive Ansätze

Während im vorangegangenen Abschnitt immer von einer symbolischen persistenten Darstellung – einem Weltmodell – ausgegangen wurde, werden nachfolgend Ansätze aufgezeigt, die keine solche Darstellung benutzen. Damit entfällt jede Form der Planung einer Aktivität zu Gunsten einer unmittelbaren dynamischen Reaktion. Diese im Zusammenhang mit der Steuerung von mobilen Robotern auch als *bottom up* bezeichnete Herangehensweise geht auf [Brooks, 1991] zurück und bildet die Grundlage der verhaltensbasierten Robotik.
Die Tatsache, dass dieses Fachgebiet durch eine Vielzahl biologisch motivierter Lösungsansätze geprägt ist, spiegelt sich auch in einer entsprechenden Begriffswelt wider. In den nachfolgenden Abschnitten werden übliche Fachbegriffe deshalb zusätzlich in Klammern angegeben.

Unter einem Verhalten wird die Kopplung von Sensorinformationen (*stimuli*) mit entsprechenden Aktorreaktionen (*responses*) verstanden. Dabei werden *reaktive* und *adaptive* Verhalten unterschieden.
Die rein reaktiven Verhalten werden auch als Reflexe bezeichnet und stellen einen deterministischen Zusammenhang zwischen Sensorinformation und Reaktion dar.
Adaptive Verhalten besitzen dagegen eine Wissensbasis (*representational knowledge*), anhand derer zunächst eine Interpretation der Sensordaten erfolgt und erst anschließend die Verhaltensreaktion bestimmt wird.

2.2.1 Beschreibungsformen

Die abstrakte Darstellung eines Verhaltens wird als Situations-Reaktions-Diagramm (*stimulus-response-diagram*) oder kurz SRD bezeichnet. Damit wird einer Situation, mit ihren zugehörigen Sensorinformationen $\underline{S}$, eine bestimmte Reaktion $\underline{R}$ zugeordnet.

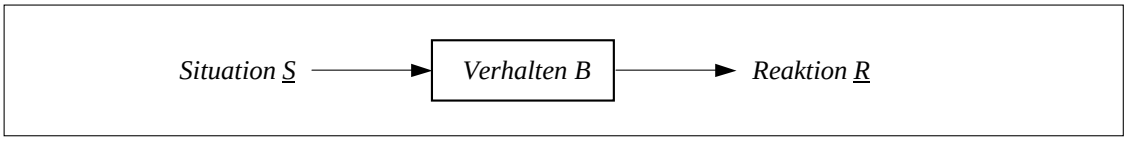


Abbildung 2.9 Situations-Reaktions-Diagramm (SRD)

Für eine reale Umsetzung auf einem Roboter bedarf es jedoch einer spezialisierten Formulierung. Für eine Programmierung eignet sich insbesondere die funktionale Notation:

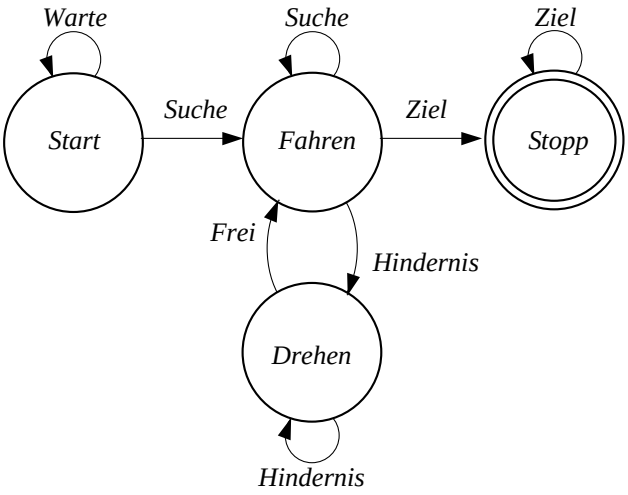
$$\underline{R} = f_B(\underline{S}) \,. \tag{2.3}$$

Dabei ist f_B eine Verhaltensfunktion, die für einen Sensorvektor $\underline{S}$ einen Reaktionsvektor $\underline{R}$ ermittelt. Unter der Annahme, dass sowohl $\underline{S}$ als auch $\underline{R}$ implizit sind, ist das Verhalten B durch einen Aufruf von $f_B()$ vollständig beschrieben.
Zur Lösung einer Aufgabe benötigt ein Roboter jedoch meist mehrere Verhalten. Beispielsweise benötigt die Suche eines Roboters nach einem markierten Zielpunkt bei gleichzeitiger Vermeidung von Kollisionen mit möglichen Hindernissen mindestens zwei unterschiedliche Verhalten – Fahren und Drehen. Die Abfolge dieser beiden

Verhalten ist sequenziell und kann mit Hilfe eines endlichen Automaten *A* (*finite state acceptor*) beschrieben werden, der in diesem Zusammenhang auch als *Sequenzer* bezeichnet wird. Allgemein wird der endliche Automat durch einen 5-Tupel der Form:

$$A = (F_B, f_S, S, \delta, E_B) \tag{2.4}$$

beschrieben. Dabei bildet $F_B = \{f_0,f_1,...,f_n\}$ die Menge der Verhaltensfunktionen, $f_S \in F_B$ ist die Funktion für das Startverhalten. Die Menge $S = \{s_0,s_1,...,s_n\}$ enthält die Folge der Sensorinformationen. Die Übergangsfunktion $\delta = f(F_B,S)$ beschreibt die Übergänge von einem Verhalten zu einem anderen, wobei jeweils nur das aktuelle Verhalten f_i und die aktuelle Sensorinformation s_i entscheidend sind. Bei imperativer Programmierung ist außerdem zu berücksichtigen, dass ein kontinuierliches Verhalten nur durch den zyklischen Aufruf der Verhaltensfunktionen erreicht wird. Daher müssen stets Übergänge der Verhalten zu sich selbst vorgesehen werden, sofern keine anderen Übergänge aktiv sind. Der Automat endet dann, wenn eines der in $E_B \subseteq F_B$ definierten Endverhalten erreicht wird.



a)

δ	F_B	S
Start	*Start*	*Warte*
Fahren	*Start*	*Suche*
Fahren	*Fahren*	*Suche*
Fahren	*Drehen*	*Frei*
Drehen	*Fahren*	*Hindernis*
Drehen	*Drehen*	*Hindernis*
Stop	*Fahren*	*Ziel*
Stop	*Stop*	*Ziel*

b)

Abbildung 2.10 Serialisierung mehrerer Verhalten
a) Darstellung als endlicher Automat
b) Übergangsfunktion δ in Form einer Tabelle

Der Begriff Verhalten ist nicht auf ein bestimmtes Abstraktionsniveau begrenzt. Es ist daher auch zulässig, die so durch einen Automaten erzeugte Verhaltensfolge wieder in einem Situations-Reaktions-Diagramm zu abstrahieren. Es existiert jedoch eine weitere Form, komplexe Verhalten abzuleiten bzw. zu beschreiben. Dabei werden mehrere,

meist elementare Verhalten (*primitives*) kombiniert. Die individuellen Reaktionen werden dazu in einem *Koordinator* zusammengefasst, um eine Gesamtreaktion $\underline{R}_G$ zu ermitteln.

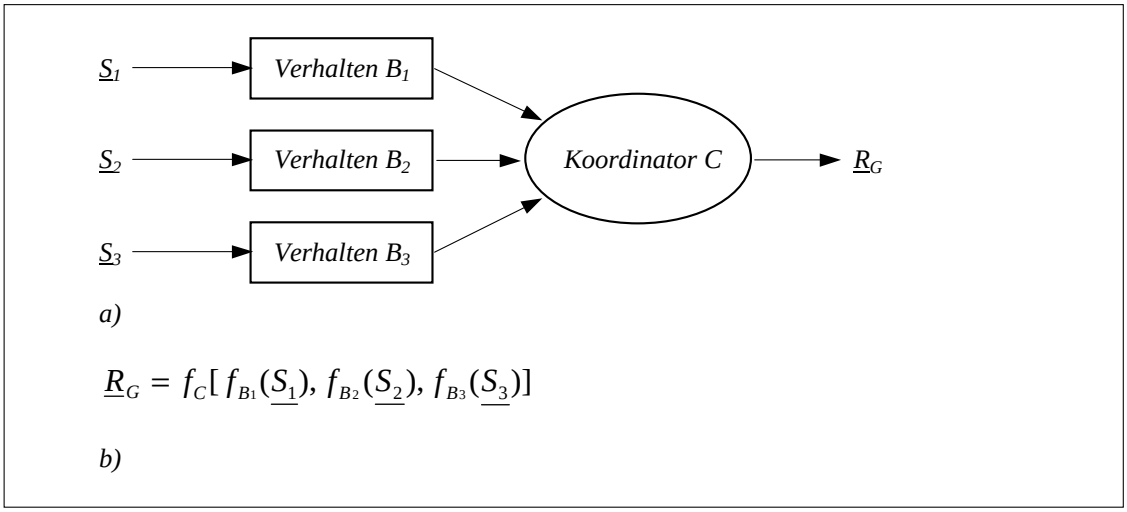


Abbildung 2.11 Kombination mehrerer Verhalten
a) Darstellung als SR-Diagramm
b) Funktionale Darstellung

An dieser Stelle wird der große Vorteil der funktionalen Darstellung in Bezug auf eine Implementierung deutlich. Die allgemeine Form der Koordinationsfunktion f_C ist der einer Verhaltensfunktion sehr ähnlich. Es gilt:

$$\underline{R}_G = f_C(\underline{R}_B)\,. \tag{2.5}$$

Dabei ist $\underline{R}_B = [\underline{R}_0,\underline{R}_1,...,\underline{R}_n]^T$ eine Matrix, die alle zu koordinierenden Verhaltensreaktionen beinhaltet. Die Gesamtreaktion $\underline{R}_G$ kann ihrerseits wieder als ein Eingangsvektor eines Koordinators angesehen werden, um auf diesem Weg komplexe Abhängigkeiten zu bilden.

Grundsätzlich werden *kooperierende* und *konkurrierende* Varianten eines Koordinators unterschieden. An dieser Stelle sollen exemplarisch zwei Vertreter vorgestellt werden, die in der Robotik häufig zum Einsatz kommen und auch in der vorliegenden Arbeit noch verwendet werden. In [Arkin, 1998] werden weitere detailliert beschrieben.

2.2.2 Hierarchische Beschreibung

Mit dem Begriff *Subsumption* beschreibt Rodney Brooks eine mehrschichtige Struktur priorisierter asynchroner Verhalten. Inzwischen hat sich daraus eine eigene hierarchische Steuerarchitektur für mobile Roboter entwickelt.
Das Grundprinzip besteht in der aufgabenspezifischen Gruppierung einzelner Verhalten in verschieden priorisierte Schichten (*layer*). Die höheren Schichten haben dabei Zugriff auf die darunter liegenden und können so die darin befindlichen Verhalten beeinflussen. Eine Besonderheit besteht darin, dass Verhaltensreaktionen höherer Priorität solche mit niederer Priorität unterdrücken oder ersetzen können.

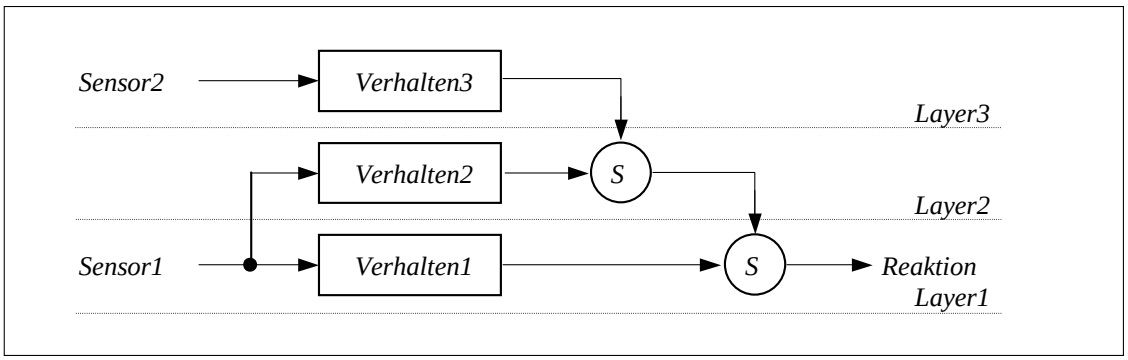


Abbildung 2.12 Hierarchisches SR-Diagramm (*subsumption*)

Auf diesem Weg kann eine bestehende Struktur schichtweise um neue Funktionalität erweitert werden. Die Koordination der Verhaltensreaktionen erfolgt durch einen priorisierten Arbitrierungsprozess. Die verschiedenen Schichten konkurrieren quasi um den Einfluss auf die Gesamtreaktion.
Die Verhalten werden als asynchron angesehen und besitzen keine gemeinsame Verbindung. Alle Sensorinformationen sowie die Reaktionen unterlagerter Schichten stehen jedoch in allen höheren Schichten zur Verfügung. Eine direkte Kommunikation zwischen den Schichten ist in Grenzen zulässig, vielmehr wird aber eine indirekte Kommunikation über die Umwelt angestrebt.

2.2.3 Strukturelle Beschreibung

Ein Versuch, die Informationsverarbeitung im Gehirn zu erklären, ist die *Schematheorie*. Sie beruht auf einer funktionalen Dekomposition komplexer Zusammenhänge in Teilprobleme. Dieser Ansatz wurde von Ronald Arkin [Arkin, 1998] aufgegriffen und zu einer biologisch motivierten Methode der verhaltensbasierten Robotik entwickelt. Das Grundprinzip besteht in einer einheitlichen Darstellung der Verhaltensreaktionen als Vektor mit Betrag und Richtungsinformation (*schema*). Auf Grund dieser Vereinheitlichung ist eine Koordination durch Superposition möglich.

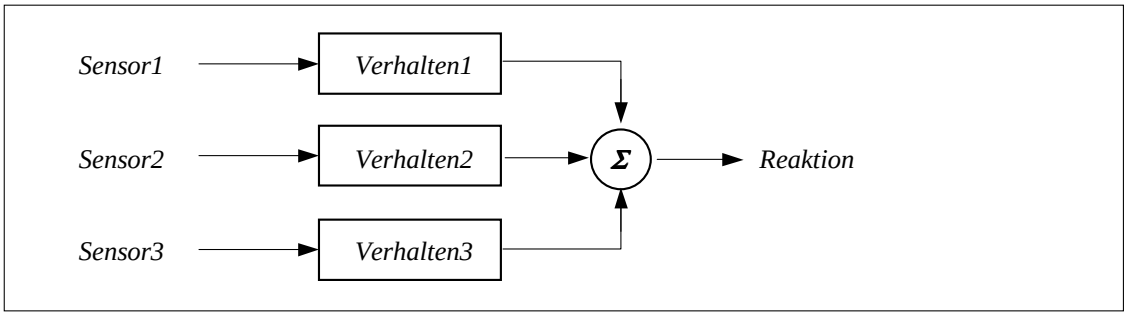


Abbildung 2.13 Strukturelles SR-Diagramm (*schema*)

Die einzelnen Verhaltensreaktionen $\underline{R_i}$ werden zunächst gewichtet und anschließend zur Gesamtreaktion $\underline{R_G}$ summiert. Da das Ergebnis wiederum ein Vektor ist, können Verhalten baumartig strukturiert werden. Somit kann formuliert werden:

$$\underline{R}_G = \sum (g_i * \underline{R}_i) . \tag{2.6}$$

Die Implementierung der Verhalten schließt auch hier, wie schon bei der hierarchischen Beschreibung, eine Sensorinterpretation mit ein. Dazu stehen alle Sensorinformationen global zur Verfügung. Im Gegensatz zur Subsumption besitzen die Verhalten aber keine Priorität. Eine gegenseitige Beeinflussung oder gar Unterdrückung sowie der Zugriff auf einzelne Verhaltensreaktionen ist ebenfalls nicht möglich. Im Sinne einer Anpassung oder Optimierung besitzen die Verhalten jedoch Parameter, deren Änderung zur Laufzeit erfolgen kann. Schemata selbst werden als dynamische Instanzen angesehen.

2.3 Steuerverhalten

Ausgangspunkt für Steuerverhalten war der Versuch der Beschreibung sowie der Simulation einer Gruppe von Individuen, wie z.B. Fisch- oder Vogelschwärmen. Reynolds entwickelte dazu das *Boids-Modell*[3] [Reynolds, 1987] und konnte damit eine realistische Simulation zeigen, die mit nur drei grundlegenden Verhalten für jedes Individuum auskam:

1. Abstoßung (*separation*) – dient der Vermeidung von Kollisionen zwischen benachbarten Individuen sowie mit Hindernissen
2. Ausrichtung (*alignment*) – dient der Ausrichtung der einzelnen Individuen in eine bestimmte Bewegungsrichtung und Angleichung der Geschwindigkeit
3. Anziehung (*cohesion*) – dient dem Zusammenhalt der Gruppe durch Anziehung zum Mittelpunkt der Nachbarn.

Durch Weiterentwicklung des Modells entstand eine ganze Reihe elementarer Verhalten, die sich nicht mehr nur auf die Simulation von Schwärmen bezogen. Der Begriff Steuerverhalten (*steering behaviors*) wurde erst später geprägt [Reynolds, 1999] und bezeichnet im Wesentlichen eine Reihe von Motor-Schemata.

Besonders interessant ist die Tatsache, dass Steuerverhalten nicht primär mit dem Ziel entstanden, sie in der Robotik einzusetzen. Vielmehr stand eine realistische, natürlich wirkende Simulation biologischer Verhalten innerhalb von Schwärmen oder Herden im Vordergrund. Im Ergebnis können jedoch Steuerverhalten als Motor-Schemata angesehen werden, die sich speziell mit der Interaktion und Bewegung innerhalb von Gruppen autonomer Systeme beschäftigen. Teilweise ähnliche Verhalten finden sich in [Arkin, 1998], wobei diese eher problemorientiert sind (Fahre zum Ziel / Vermeide Kontakt mit Hindenissen). Steuerverhalten sind hingegen eher gruppenorientiert (Bleibe innerhalb der Gruppe / Vermeide Kontakt mit Nachbarn). In diesem Sinne sollen nachfolgend zunächst das Verhaltensmodell und darauf basierend einige elementare Steuerverhalten sowie einige kombinierte Steuerverhalten beschrieben werden.

[3] *boids* = Abkürzung des Kunstworts *birdoids*, welches aus *birds* und *droids* gebildet wurde.

2.3.1 Bewegungsmodell

Das bei Reynolds zugrundegelegte Bewegungsmodell basiert auf einer Anzahl gleichartiger Objekte (Individuen, Fahrzeuge, Roboter, etc.), die neben allgemeinen Merkmalen, wie z.B. der Geometrie und ihren Dimensionen, immer folgende grundsätzliche Merkmale aufweisen:

- Position $\underline{s} = [s_x,s_y,s_z]^T$ und
- Geschwindigkeit $\underline{v} = [v_x,v_y,v_z]^T$.

Die Beschreibung des Bewegungszustandes $\underline{p}$ der Objekte beruht auf einer Masse m, die sich mit einer Geschwindigkeit $\underline{v}$ durch ein globales Koordinatensystem bewegt.

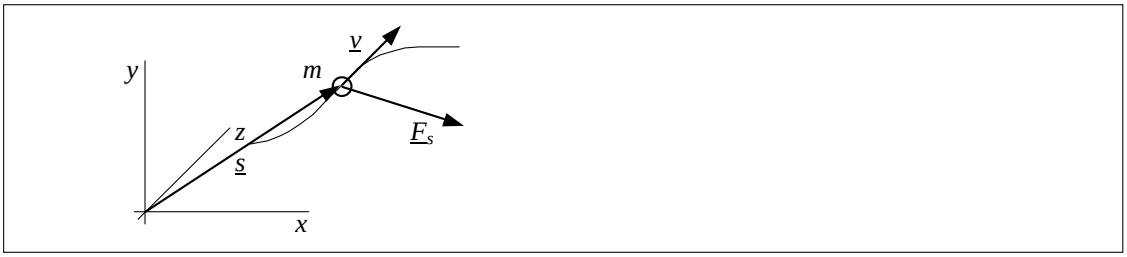


Abbildung 2.14 Modell einer Punktmasse

Dabei wird jedes Objekt als eine Punktmasse betrachtet, die nur aufgrund einwirkender Kräfte ihren Bewegungszustand $\underline{p} = m\underline{v}$ ändert:

$$\sum \underline{F} = m\frac{d\underline{v}}{dt}. \tag{2.7}$$

Die Massen der Objekte werden nicht weiter betrachtet, d.h. es wird von einer Einheitsmasse $m = 1$ ausgegangen. Die Summe der zeitweise einwirkenden Kräfte $\underline{F_s}$ führt in Folge zu einer Geschwindigkeitsänderung der Objekte:

$$\underline{v} = \underline{v}_0 + \int \underline{F}_s(t)dt. \tag{2.8}$$

Die resultierende Geschwindigkeit $\underline{v}$ der Objekte führt dann zu einer Änderung ihrer Position $\underline{s}$:

$$\underline{s} = \underline{s}_0 + \int \underline{v}(t)dt. \tag{2.9}$$

Die einwirkende Kraft $\underline{F_s}$ ensteht durch die Steuerverhalten selbst, indem die gewichteten Verhaltensreaktionen $\underline{F_i} = g_i * \underline{R_i}$, vektoriell addiert werden und so zu jedem Zeitpunkt t einen resultierenden Steuervektor $\underline{F_s}(t) = \Sigma \underline{F_i}(t)$ ergeben.

Für die Simulation der Bewegung eines Objekts ist dessen nächste Sollposition gesucht – bekannt sind die aktuelle Position und Geschwindigkeit sowie der momentane Steuervektor. Zur Lösung bietet sich eine numerische Integration mit konstantem Zeitschritt t_s (Euler-Integration) an. Obwohl die Art der Integration bei Reynolds nicht

angegeben ist, besteht neben der einfachen Implementierung der Euler-Integration auch die Möglichkeit der kontinuierlichen Darstellung des Bewegungszustandes.

Zusammenfassend lässt sich feststellen, dass das vorgestellte Bewegungsmodell zunächst sehr einfach ist. Eine sinnvolle Bewegung hängt in besonderem Maße von den Steuerverhalten und den sich daraus ergebenden Steuervektoren ab. Eine erste Implementierung zeigte, dass das Modell auf Grund fehlender Dämpfung sehr empfindlich reagiert. Zusätzliche Randbedingungen müssen formuliert werden, um eine übermäßige und damit unrealistische Geschwindigkeit oder Beschleunigung der Objekte bzw. sprunghafte Richtungswechsel in der Bewegung zu vermeiden.
Da es sich jedoch um ein verteiltes Verhaltensmodell (*distributed behavior model*) handelt, bei dem jedes Objekt nur seine eigene Bewegung steuert, lässt es sich idealerweise auch auf reale Objekte (Roboter) übertragen. Eine solche Umsetzung bedarf jedoch der Lösung weiterer Teilprobleme, wie beispielsweise der Berücksichtigung kinematischer Einschränkungen. Die Bewegung der Objekte wird hier als omnidirektional angenommen, was nur bei wenigen realen Robotern der Fall ist. Dennoch ist die Zweiteilung einer realen Robotersteuerung in ein unterlagertes Bewegungsmodell mit übergeordneter Vorgabe eines Steuervektors gerade für mobile verhaltensbasierte Roboter sinnvoll. Einige ausgewählte Steuerverhalten, die für die Umsetzung auf Robotern geeignet sind, sollen nachfolgend beschrieben werden.

2.3.2 Elementare Steuerverhalten

Unter elementaren Steuerverhalten sind solche zu verstehen, bei denen nur ein Verhalten den Steuervektor bildet. Dabei sind die Verhalten Anziehung (*Seek*) bzw. Abstoßung (*Separation*) die Grundlage jeder motivierten Bewegung. Es muss immer genau einen Zielpunkt *Z* geben, auf den das Objekt zusteuert oder eine Reaktion, die zur Abstoßung führt, andernfalls wird sich der Bewegungszustand nicht ändern. Die Bestimmung der Steuervektoren ist bis auf die Richtung für Anziehung und Abstoßung gleich.

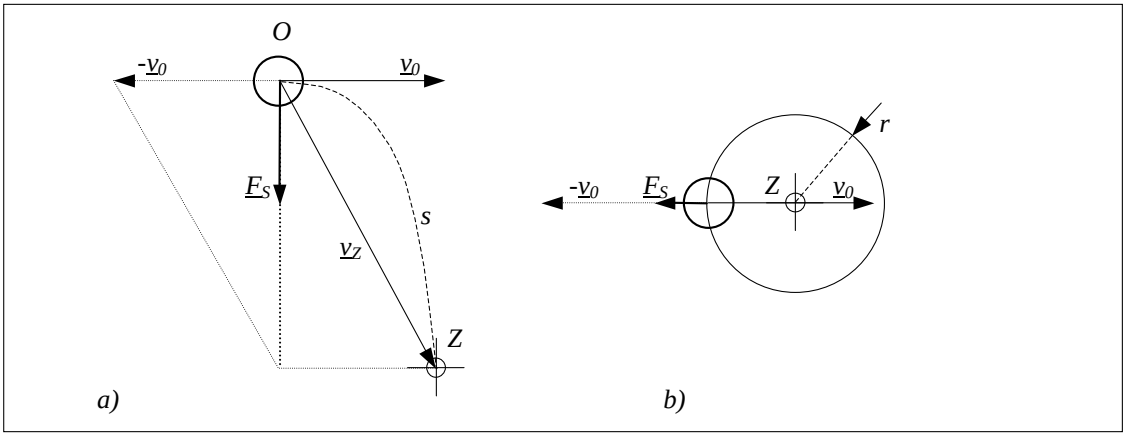


Abbildung 2.15 Ziel-Steuerverhalten
a) Anziehung (*Seek*)
b) Annäherung (*Arrive*)

Der Steuervektor $\underline{F_S}$ wird durch Subtraktion der aktuellen Geschwindigkeit $\underline{v_0}$ von der Zielgeschwindigkeit $\underline{v_Z} = \underline{OZ}$ berechnet:

$$\underline{F}_s = \underline{v}_z - \underline{v}_0 \,. \tag{2.10}$$

Der Steuervektor $\underline{F_S}$ ist ein Maß für die Geschwindigkeitsänderung pro Zeiteinheit. Da je nach Abstand des Objekts zum Ziel die Zielgeschwindigkeit $\underline{v_Z}$ und damit der Steuervektor $\underline{F_S}$ betragsmäßig sehr gross werden können, muss eine Begrenzung des Steuervektors einsetzen. Durch diese Maßnahme wird der eigentliche Bahnverlauf bestimmt. Wird $\underline{F_S}$ sehr groß zugelassen, ändert sich die Bewegung dynamisch, während sich bei kleinen $\underline{F_S}$ nur noch träge Richtungsänderungen ergeben. Es hat sich als praktisch erwiesen, eine Begrenzung des Verhältnisses zwischen Betrag des Steuervektors $|\underline{F_S}|$ und Betrag der momentanen Geschwindigkeit $|\underline{v_0}|$ einzuführen, damit bei kleinen Geschwindigkeiten keine abrupten Richtungswechsel auftreten, aber bei höheren Geschwindigkeiten der Einfluss des Steuervektors genügend groß ist. In unmittelbarer Nähe des Zielpunktes kann es bei reiner Anziehung zu einem Überschwingen oder gar einer Pendelbewegung des Objekts um den Zielpunkt kommen. Der Einfluss des Steuervektors ist in dieser Situation möglicherweise gerade zu gering, sodass sich das Objekt über den Zielpunkt hinaus bewegt, wodurch sich die Richtungsverhältnisse umkehren. Aus diesem Grund gibt es ein spezialisiertes Verhalten Annäherung (*arrive*), bei dem die Geschwindigkeit des Objekts *O* innerhalb eines Radius *r* proportional zur Entfernung zum Zielpunkt *Z* reduziert wird, sofern sie oberhalb eines entsprechenden Schwellwertes liegt. Das entspricht in etwa einer Art Bremsvorgang (Dämpfung) des Objekts.

Im Gegensatz zur Anziehung kann eine Abstoßung auch von mehreren Punkten ausgehen. Es wird daher in Flucht (*flee*) und Abstoßung (*separation*) unterschieden, je nachdem, ob die Reaktion immer nur von einem oder auch von mehreren Punkten ausgeht. Im Allgemeinen resultieren beide Verhalten aus der Nähe zu anderen Objekten innerhalb ihres Umkreises.

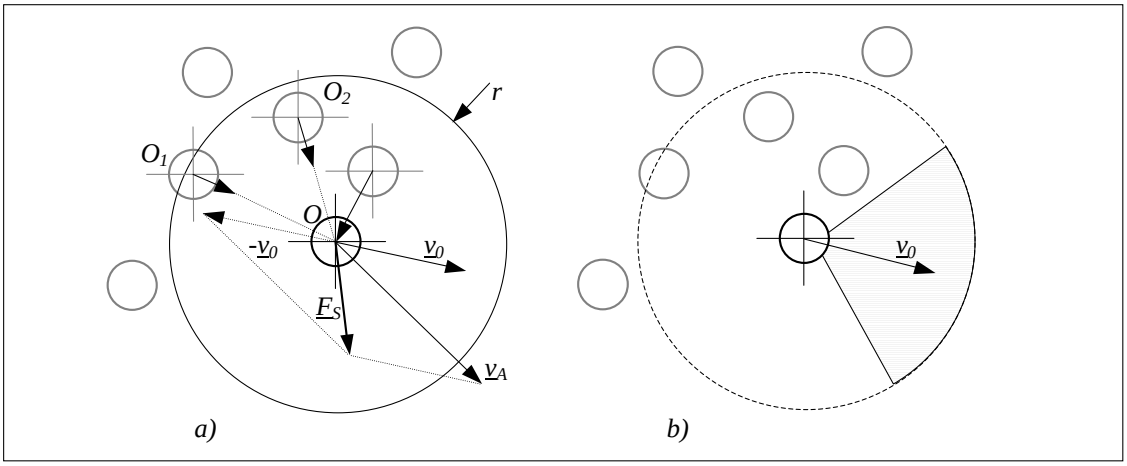


Abbildung 2.16 Steuerverhalten Abstoßung
a) Allseitige Abstoßung
b) Gerichtete Abstoßung

Zunächst wird die aus den Objekten innerhalb eines Umkreises *r* resultierende Abstoßungsgeschwindigkeit $\underline{v_A}$ ermittelt. Dazu wird ausgehend von jedem Abstoßungspunkt innerhalb des Radius *r* ein Vektor $\underline{v_n}$ zum Objekt *O* bestimmt. Der Betrag der Vektoren verhält sich dabei umgekehrt proportional zur Entfernung. Die Summe der Vektoren ergibt dann die Abstoßungsgeschwindigkeit:

$$\underline{v}_A = \sum \underline{v}_n \ . \tag{2.11}$$

Der Steuervektor $\underline{F_S}$ ergibt sich, wie schon bei der Anziehung, durch Subtraktion der momentanen Geschwindigkeit $\underline{v_0}$ von der ermittelten Abstoßungsgeschwindigkeit $\underline{v_A}$. Eine in [Feilkas, Schnellhammer, 2002] beschriebene Abwandlung dieses Verhaltens betrachtet nur einen Sektor des Umkreises, der sich aus einem Öffnungswinkel um die momentane Bewegungsrichtung $\underline{v_0}$ ergibt. Dadurch wird vermieden, dass Objekte „angeschoben" werden. Eine Testimplementierung zeigte, dass auch eine Neigung zu Schwingungen besteht, wenn stets alle Objekte im Umkreis berücksichtigt werden.

Abstoßung von Hindernissen lässt sich in der Form schlecht beschreiben, da ausgedehnte Hindernisse nicht auf einen Punkt reduziert werden können. In [Reynolds, 1988] sowie in [Feilkas, Schnellhammer, 2002] werden auch hierfür geeignete Steuerverhalten beschrieben. Obwohl beide Ansätze stets auf lokalen Informationen basieren, werden umfassende Kenntnisse der Umgebung vorausgesetzt. Es erfolgt nicht eine Simulation von speziellen Sensorinformationen, sondern vielmehr eine Operation auf bereits abstrahierten Umweltinformationen.

Neben Anziehung, Annäherung und Abstoßung werden in [Reynolds, 1999] noch weitere elementare Steuerverhalten erwähnt. Dabei handelt es sich im Wesentlichen um simulationsspezifische Verhalten wie Wandern (*wander*) oder Verfolgen (*pursiut*) bzw. Entkommen (*evade*), auf die hier jedoch nicht näher eingegangen werden soll. Es handelt sich dabei um Abwandlungen der bereits vorgestellten Steuerverhalten Anziehung und Abstoßung. In [Arkin, 1998] werden Problemssituationen beschrieben, bei denen vergleichbare Verhalten (dort Motor-Schemata) unter anderem zur Suche von Auswegen verwendet werden. Dabei wird z.B. einer Bewegung ein Rauschanteil überlagert, was hier dem Verhalten Wandern gleichkommt.

Die elementaren Steuerverhalten allein führen meist noch nicht zu einer brauchbaren Bewegung. Sie sind mehr als ein Pool zu verstehen, um daraus angepasste Verhalten zu kombinieren. Deshalb sollen im Folgenden eine Reihe solcher kombinierten Steuerverhalten erläutert werden.

2.3.3 Kombinierte Steuerverhalten

Die Verbindung von zwei oder mehr Verhalten zu einem kombinierten Steuerverhalten erfolgt, wie bereits in Abschnitt 2.2.1 beschrieben, in einem Koordinator *C*. Dabei werden die Verhaltensreaktionen zunächst gewichtet und vektoriell zu einer Gesamtreaktion aufsummiert. In [Reynolds, 1999] werden zwei verschiedene Koordinatoren vorgeschlagen. Die erste Variante (*simple mind*) ist identisch mit den von Arkin verwendeten Motor-Schemata (vgl. Gl. 2.6). Jedes Verhalten hat einen zuvor definierten Einfluss g_i auf die Gesamtreaktion. Die Gesamtreaktion wird, wie bereits bei den elementaren Steuerverhalten auch, anschließend auf einen Maximalwert begrenzt. Damit wird jedes Verhalten prozentual in der Gesamtreaktion berücksichtigt – zumindest hinsichtlich seiner Richtung.

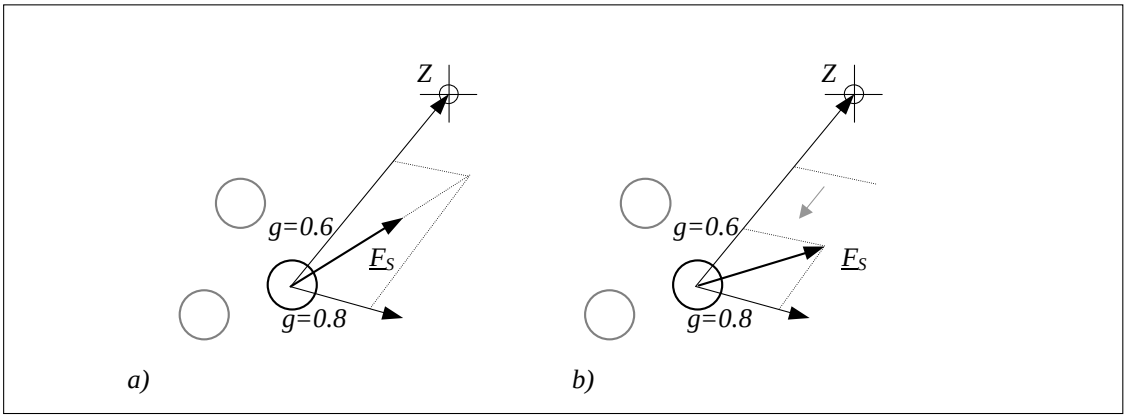


Abbildung 2.17 Koordination von Steuerverhalten
a) Gewichtete Superposition (*simple mind*)
b) Priorisierte Superposition (*priority mind*)

Im Gegensatz dazu, wird eine priorisierte Superposition (*priority mind*) vorgeschlagen, bei der die gewichteten Verhaltensreaktionen in der Reihenfolge der größten Einflussfaktoren g_i aufsummiert werden. Wird dabei der maximale Betrag der Gesamtreaktion $|\underline{F_S}|_{max}$ erreicht, wird das letzte Verhalten begrenzt bzw. noch nicht berücksichtige Verhalten werden ganz unterdrückt. Durch diese Art der Superposition erhalten höher priorisierte Verhalten dynamisch mehr Einfluss auf die Gesamtreaktion. Am Beispiel einer Ausweichbewegung bedeutet das, dass die Abstoßung mit einem größeren Gewicht berücksichtigt wird, wenn eine konkrete Situation vorliegt (Abbildung 2.17). Solange die Abstoßung nur einen kleinen Teil der Gesamtreaktion ausmacht, wird sie nur prozentual berücksichtigt.

Kombinierte Steuerverhalten ergeben sich oft durch eine Überlagerung von anziehenden und abstoßenden Komponenten. Bestes Beispiel ist hier sicher die Objektverfolgung (*leader following*). Der Zielpunkt eines folgenden Objekts ist die Position des verfolgten Objekts. Damit bei unterschiedlicher Geschwindigkeit stets ein Abstand zwischen beiden Objekten bleibt, wird der Anziehung durch das verfolgte Objekt eine Abstoßung zwischen beiden entgegengesetzt, die sich umgekehrt proportional zum Abstand verhält. In [Reynolds, 1999] wird dieses Verhalten auf *n* Objekte verallgemeinert. Alle Objekte haben einen gemeinsamen Zielpunkt (*leader*) stoßen sich aber untereinander und vom Zielobjekt stets ab. Dadurch entsteht der Eindruck, dass ein Objekt von allen anderen bedrängt wird. Eine Abwandlung dieses Verhaltens, bei dem jedes Objekt von nur einem weiteren Objekt verfolgt wird, hat eine Bewegung wie eine Schleppkette zur Folge. Weitere Abwandlungen sind darüberhinaus denkbar. Wenn beispielsweise jedes Objekt immer eine zurückliegende Position eines anderen als Zielpunkt annimmt, entsteht eine Schlangenbewegung.

Andere kombinierte Steuerverhalten basieren auf Bahnverfolgung und Abstoßung, womit sich Verkehrs- oder Rennsituationen beschreiben lassen. In [Arkin, Bekey, 1997] wird zur Simulation der Bewegung virtueller Roboter Gruppenverhalten zugrunde gelegt. Die dort beschriebenen Algorithmen sind den Steuerverhalten recht ähnlich.

Zusammenfassend lassen sich Steuerverhalten also durch zwei wesentliche Merkmale charakterisieren:

- einfaches Bewegungsmodell
- dezentrales Verhaltensmodell.

Im Ergebnis lassen sich damit komplexe dynamische Bewegungsvorgänge in Gruppen auf grundlegende Vektoroperationen zurückführen. Die Bestimmung entsprechender Vektoren obliegt dabei den elementaren Steuerverhalten. Einige Ansätze dazu sind beschrieben worden. Steuerverhalten eignen sich jedoch nicht nur zur Beschreibung und Simulation existierender biologischer Systeme (Gruppen, Schwärme, Herden), sondern sind vielmehr auch auf den Entwurf und die Implementierung von Multi-Roboter-Applikationen anwendbar.

Ohne weitere Randbedingungen oder Regulation führen Steuerverhalten zwar zu sinnvollen motivierten Bewegungen, aber die Reproduzierbarkeit ist eher gering. Die Bewegungsbahn eines einzelnen Objekts ist von den Anfangsbedingungen aller abhängig und kann gerade durch Wechselwirkungen mit anderen Objekten sehr stark beeinflusst werden.

2.4 Hybride Systeme als Symbiose

Nachdem in den vorangegangenen Abschnitten sowohl klassische routenbasierte als auch verhaltensbasierte reaktive Ansätze zur Bewegungssteuerung vorgestellt wurden, soll hier zunächst ein Vergleich der beiden Methoden angestellt werden. Offensichtlich sind beide Ansätze nicht für jede Robotikanwendung gleichermaßen gut geeignet bzw. weisen unter bestimmten Umständen sogar erhebliche Nachteile auf. Immer dann, wenn die Umwelt der Roboter quasi statisch ist und mögliche Interaktionen mit Hindernissen oder anderen Robotern beschreibbar sind, wird man auf ein Weltmodell orientieren. Ein Beispiel sind hier Fertigungszellen, die speziell ausgelegt werden und in denen idealisierte Bedingungen herrschen. Sobald jedoch wechselnde oder unstetige Umweltbedingungen auftreten können, ist ein verhaltensbasierter Ansatz robuster. Viele Problemstellungen in der Robotik sind aber der Art, dass es sowohl statische als auch dynamische Randbedingungen gibt. Bestes Beispiel sind hier Transportaufgaben, bei denen optimale Routen geplant werden, aber stets unerwartet Hindernisse auftreten können. Aus diesem Grund wurden hybride Systeme geschaffen, die sowohl Planung als auch Reaktion miteinander verbinden.

2.4.1 Vergleich von modell- und verhaltensbasierten Ansätzen

Die nachfolgend aufgeführten Schwachpunkte sollen einen Vergleich der bisher diskutierten Ansätze ermöglichen:

Rein reaktive Ansätze...

...können keine optimalen Trajektorien ermitteln. Verschiedene Projekte in [Kortenkamp, Bonasso u. Murphy, 1998], [Arkin, 1998], bei denen verhaltensbasierte mobile Roboter praktisch erprobt wurden, zeigen, dass sie allein durch Reaktion auf

ihre unmittelbare Umgebung selbst komplexe Hindernissituationen erfolgreich durchfahren können. Dabei entstehen aber mitunter sehr viele unnötige Bewegungen.

...sind nicht in der Lage das optimale Verhalten zu bestimmen. In [Schmidt, 1995] werden Hindernissituationen erprobt, die mit unterschiedlichen Regelmengen der Form: IF <*Situation*> THEN <*Reaktion*> passiert werden sollen. Dabei zeigt sich, dass die Zahl der Verhaltensregeln mit steigendem Schwierigkeitgrad erhöht werden muss, um immer noch eine Route zu finden bzw. Deadlocks zu vermeiden. Umgekehrt ist es jedoch ebenso ungünstig, immer die umfassendste Regelmenge zu verwenden. Für eine optimale Regelmenge ist a priori Wissen über die aktuelle Situation erforderlich.

...können sich nicht selbst überwachen. Situationen in denen sich unbeabsichtigt zyklisch die selben Verhalten wiederholen (z.B. *deadlocks* oder *loops*) können nicht erkannt werden. In [Arkin, 1998] wird ein Verhalten (*avoid past*) beschrieben, welches insbesondere diese Problematik lösen kann. Dazu ist aber die Speicherung der bisherigen Wegstrecke nötig, was über rein reaktive Lösungen hinausgeht. Allgemein kann aber auch dieses zusätzliche Verhalten nicht erkennen, wie effektiv die letzten Schritte waren.

Modellbasierte Ansätze...

...sind von der Vollständigkeit ihrer Wissensbasis abhängig. In einer modellgestützten Routenplanung können nur die Randbedingungen berücksichtigt werden, die zuvor symbolisch hinterlegt wurden. Das Ergebnis einer Planung, die auf unvollständigen Informationen beruht, ist unbestimmt. Das ist insbesondere auch dann der Fall, wenn Unstetigkeiten auftreten.

...sind in ihrer Komplexität von der Zahl der Freiheitsgrade abhängig. Ein Roboter besitzt in der Ebene drei Freiheitsgrade (x,y,ϕ). Eine Routenplanung würde daher im Konfigurationsraum stattfinden. Für zwei Roboter hätte der Konfigurationsraum schon 6 Dimensionen!

...sind nicht echtzeitfähig. Im Wesentlichen sind es zwei Zeiten, die die Dynamik einer modellbasierten Planung bestimmen; die Zeit für die Erstellung bzw. Aktualisierung des Modells und die Zeit für die Planung selbst. Zusätzlich müssen die in der Planung angenommenen Bedingungen für die Zeit der Ausführung stabil bleiben.

Die Vorteile einer reaktiven Steuerung liegen in der Dynamik und Robustheit gegenüber Veränderungen der Umgebung – die Vorteile der modellbasierten Steuerung liegen in der räumlichen Abstraktion und der damit verbundenen Planbarkeit von optimalen Reaktionen. Eine Verbindung im Sinne einer Symbiose schließt dabei beide Informationsrichtungen ein. Globale Planungsergebnisse beeinflussen das lokale Verhalten ebenso, wie lokale Sensorinformationen in die Planung einfließen. Zusätzlich besteht auch die Möglichkeit einer Kontrolle, um aktive Verhalten zu korrigieren oder gar abzubrechen.

2.4.2 Hybride Architekturen

Die Struktur einer hybriden Architektur teilt sich in mindesten drei Schichten auf. Die oberste Schicht ist die globale (*deliberative*) Planung. Das Planungsergebnis ist eine vollständige Beschreibung des Lösungsweges unter Zugrundelegung des Modells und einer Strategie. Im Fall einer Bewegungsplanung ist das Ergebnis die gesamte Route. Dabei können Vorgaben (z.B. Optimierungskriterium) die Strategie dahingehend beeinflussen, einen kurzen oder einen sicheren Weg zu wählen. Eine Zwischenschicht ist erforderlich, um aus der übergeordneten Planung Teilaufgaben zu extrahieren und an die Ausführungsschicht zu übergeben. Die Ausführungsschicht selbst besteht im Wesentlichen aus einer Reihe von Verhalten.

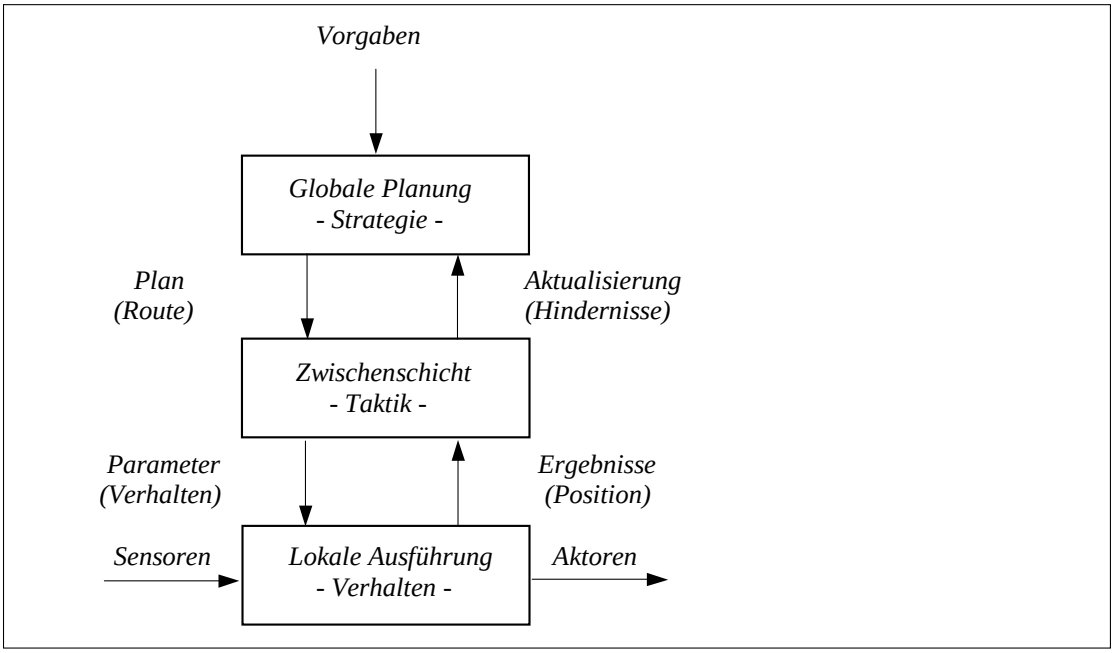


Abbildung 2.18 Hybride Steuerungsarchitektur

Hybride Architekturen lassen sich nach [Arkin, 1998] im Wesentlichen in vier Gruppen einteilen. Dabei wird unterschieden, welchen Einfluss die globale Planung auf die lokalen Reaktionen hat.

Die Planung beeinflusst die Ausführung (adaption).
Die Planung erfolgt kontinuierlich. Die Ergebnisse werden dazu verwendet, um Verhaltensparameter zu ändern. In diesem Fall ist eine explizite Zwischenschicht nicht erforderlich oder kann als Teil der Planung angesehen werden, da das Planungsergebnis schon die Verhaltensparameter sind.

Die Planung wird als Konfiguration angesehen (selection).
Die Planungsergebnisse werden dazu verwendet, passende Verhalten zu bestimmen und deren Parameter vorzugeben. Die Ausführung erfolgt danach weitestgehend selbstständig.

Die Planung wird als Lösungsvariante angesehen (advising).
Als Ergebnis der Planung entsteht eine Lösungsvariante oder auch Alternative, die während der Ausführung verwendet werden kann.

Die Planung erfolgt nur nach Bedarf (postponing).
Die Ausführungsschicht initiiert die Planung.

2.4.3 Schnittstelle zwischen Planung und Ausführung

Die Schnittstelle zwischen Planung und Ausführung wurde bisher abstakt als Zwischenschicht bezeichnet. Ihre Funktion besteht in der Ableitung von Teilaufgaben und deren Weitergabe an die Ausführungsschicht. Gerade diese Schnittstelle bedarf jedoch einer weitergehenden Betrachtung. Sowohl für die Planungs- als auch für die Ausführungsschicht existieren in der Literatur recht übereinstimmende Beschreibungsformen. Speziell für die Bewegungssteuerung sind einige Ansätze in den vorangegangenen Abschnitten vorgestellt worden. Offensichtlich ist aber die Verbindung durch eine Zwischenschicht im Sinne einer hybriden Architektur stark implementierungsabhängig. Einige frei gewählte Beispiele sollen das belegen:

Planung und Ausführung erfolgen auf einem Roboter:
Das ist sicher der Standardfall für einen mobilen Roboter, der seine Sensorinformationen entsprechend einem vorgegebenen Modell der Umwelt abstrahiert und darauf basierend seine nächsten Aktionen plant. Die Aufgabe der Zwischenschicht ist hier eine Art Serialisierung der Aktionen und ggf. eine Parametrierung von Basisverhalten (*adaption*).

Planung erfolgt stationär und Ausführung auf einem Roboter:
Unter stationär soll hier eine vom mobilen Roboter abgesetzte zeitdiskrete (*offline*) Planung verstanden werden. Eine solche Konstellation ergibt sich dann, wenn in die Planung zusätzliche Informationen über die Umgebung des Roboters einfließen, die nicht lokal verarbeitet werden können oder durch die zusätzliche Reaktionszeiten entstehen. Dabei ergibt sich die Notwendigkeit einer Vorausplanung (*selection*/*advising*). Die Zwischenschicht serialisiert das Planungsergebnis und kontrolliert die Ausführung.

Planung erfolgt stationär mit anschließender Ausführung auf mehreren Robotern:
Für den Fall, dass eine Planung mehrere Roboter umfasst, müssen Teilaufgaben festgelegt und an die entsprechenden Roboter übertragen werden (*selection*). Die Zwischenschicht wird sich in einen übergeordneten und einen roboterspezifischen lokalen Teil aufspalten. Während die Bestimmung der Teilziele und eine Koordination übergeordnet erfolgen, muss die Serialisierung und Kontrolle lokal erfolgen.

Es zeigt sich daher, dass die Schnittstelle zwischen Planungs- und Ausführungsschicht recht unterschiedlich ausgelegt werden kann.

3 Bewegungsplanung in Multirobotersystemen

3.1 Planungsstrategie

In Kapitel 2 wurden grundlegende Ansätze zur Bewegungssteuerung beschrieben. Dabei wurde nicht explizit auf Multirobotersysteme orientiert. In diesem Kapitel soll nun ein neu entwickelter Ansatz vorgestellt werden, der insbesondere für die Steuerung mehrerer mobiler Roboter geeignet ist. Die bereits angesprochene Problematik der Koordination wird durch eine hybride Architektur dahingehend aufgespalten, dass Abhängigkeiten der Roboter untereinander sowie von ihrer Umwelt in einem globalen Planungsmodul berücksichtigt werden, während die Roboter selbst verhaltensbasiert interagieren. Die Verbindung von der globalen Planung zur lokalen Ausführung erfolgt dabei durch einen Prozess der Auswahl und Parametrierung situationsabhängiger Steuerverhalten.
Nach einer Einordnung der betrachteten Systeme und einer Formulierung der Problemstellung wird schließlich eine entsprechende Planungsstrategie entwickelt und beschrieben.

3.1.1 Einordnung der betrachteten Systeme

Multirobotersysteme sind vielfach Gegenstand der Forschungen, obgleich sie im praktischen Einsatz noch vergleichsweise selten anzutreffen sind. Am bekanntesten sind hier sicher Transportroboter, die sowohl in industrieller Umgebung als auch bereits in Gesundheits- oder Verwaltungseinrichtungen eingesetzt werden. Diese Systeme bilden aber nur einen Teilbereich der mobilen Multirobotersysteme. Nachfolgend soll deshalb eine Einteilung nach der Art ihres Zusammenwirkens oder auch ihrer internen Kopplung erfolgen.

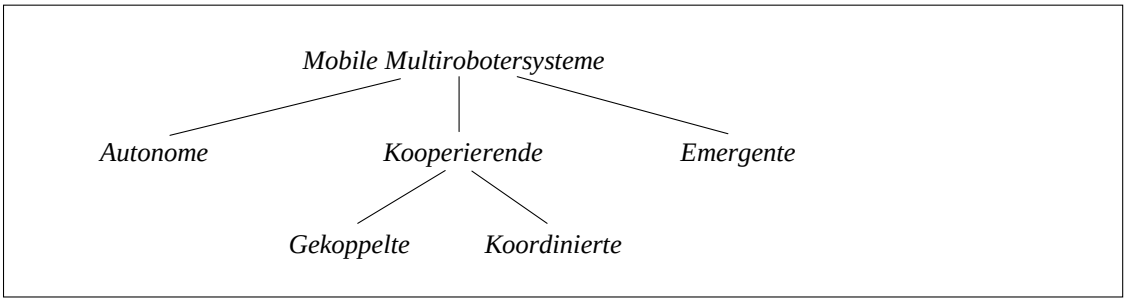


Abbildung 3.1 Einteilung mobiler Multirobotersysteme

Dabei gehören die eingangs erwähnten Transportsysteme im Allgemeinen zu den autonomen Multirobotersystemen. Autonom deshalb, weil die einzelnen Roboter weitestgehend unabhängig voneinander eigenständig operieren. Probleme treten bei diesen Systemen durch die Überlagerung oder Kreuzung von Bahnen auf. In [Freund, Rokossa, 1992], [Li, 1996] werden unterschiedliche Ansätze beschrieben, um die dabei

auftretenden Kollisions- und Deadlock-Situationen aufzulösen. Der Übergang zu kooperierenden Multirobotersystemen ist aber auch bei den Transportsystemen fließend. Insbesondere wenn am Transport nacheinander oder zeitgleich mehrere Roboter beteiligt sind, muss man von kooperierenden Multirobotersystemen sprechen. In [Inoue, Ota, Hirano, Kurabayashi u. Arai, 1998] wird ein Experimentalsystem kooperierender Transportroboter vorgestellt, welches Objekte ggf. von einem zum anderen Roboter weiterreicht, um im Gegenzug dafür optimale Routen für die einzelnen Roboter planen zu können. Ein vergleichbarer Effekt wird beim bekannten Roboter-Fußball angestrebt. Um den Ball schließlich ins gegnerische Tor zu bekommen, muss dieser möglichst effektiv weitergegeben werden. Diese Beispiele kooperativen Handelns werden als koordiniert bezeichnet. Die Roboter haben dabei eine gemeinsame Vorstellung vom Ablauf oder auch eine gemeinsame Absicht (*intention*). Ebenfalls intensiv untersucht und in der Literatur veröffentlicht sind kooperierende Manipulationen. In [Desai, 1998] wird die Kinematik zweier Roboter beschrieben, die über einen Körper gekoppelt sind. Diese Anordnung beschreibt den Fall, eines gemeinsamen Transports (*box pushing*). Im Gegensatz zur koordinierten Kooperation überwiegt hier jedoch der reaktive Anteil. Schließlich darf hier die emergente Kooperation (*emergence*) nicht unberücksichtigt bleiben, bei der die Gesamtreaktion eines Multirobotersystems mehr als die Summe der Einzelreaktionen darstellt. Diese Form der Kooperation entsteht offensichtlich, wenn genügend Roboter sich gegenseitig beeinflussen. Der Entwurf eines solchen Systems erfolgt jedoch überwiegend empirisch und ist bisher mehr eine Kunst als eine Wissenschaft. Es besteht jedoch starkes Interesse an der Erforschung der Ursachen und Grundlagen, da Emergence eine gebietsübergreifende Erscheinung komplexer Systeme ist.

Nach dieser Einteilung befasst sich die nachfolgende Planungsstrategie mit koordiniert kooperierenden Robotern. Hinzu kommt, dass die Zahl der Roboter klein ist, also Systeme mit 2..10 Robotern. Es ist vorstellbar, dass Erkundungs-, Überwachungs-, Sicherungs- oder Inspektionsaufgaben damit ausführbar sind.

3.1.2 Problemstellung

Verfahren zur Bestimmung einer optimalen Route für einen Roboter wurden bereits beschrieben. Prinzipiell können diese Verfahren auch auf eine Gruppe von Robotern übertragen werden. Dabei kann die Gruppe unterschiedlich betrachtet werden:

- als ein Roboter mit einer entsprechenden Geometrie und Kinematik oder
- als eine Menge einzelner Roboter.

Für die Planung erhöht sich im Wesentlichen nur die Zahl der Freiheitsgrade. Wird die Gruppe aber zu einem Roboter abstrahiert, entstehen möglicherweise kinematische Einschränkungen, die ein einzelner Roboter nicht hat. Außerdem werden auch Lösungswege ausgeschlossen, die auf Grund der resultierenden Geometrie nicht mehr passierbar sind. Auf der anderen Seite agieren einzelne Roboter nicht als Gruppe. Während der Planung könnten unterschiedliche Routen pro Roboter entstehen, sodass die Gruppe aufgelöst wird. In unstrukturierter Umgebung könnten Roboter daher abgespalten oder gar verloren gehen.

Prinzipiell soll das Ziel darin bestehen, eine Gruppe von Robotern als Einheit zu steuern. Damit wird es einem Operator möglich, komplexe Bewegungen mit wenigen Vorgaben zu erreichen.

Für die koordinierte Bewegung eines solchen Multirobotersystems sollen daher folgende Anforderungen erfüllt werden:

- Planungsvorgaben, wie Start- und Zielkonfiguration, für die gesamte Gruppe
- Planung einer gemeinsamen Route für alle Roboter unter Berücksichtigung einer relativen Lage zueinander (Formation)
- Bewegung der Roboter als Gruppe in einer Formation
- Ausweichen um Hindernisse, ohne dabei die Gruppe zu verlassen (Formation jedoch änderbar).

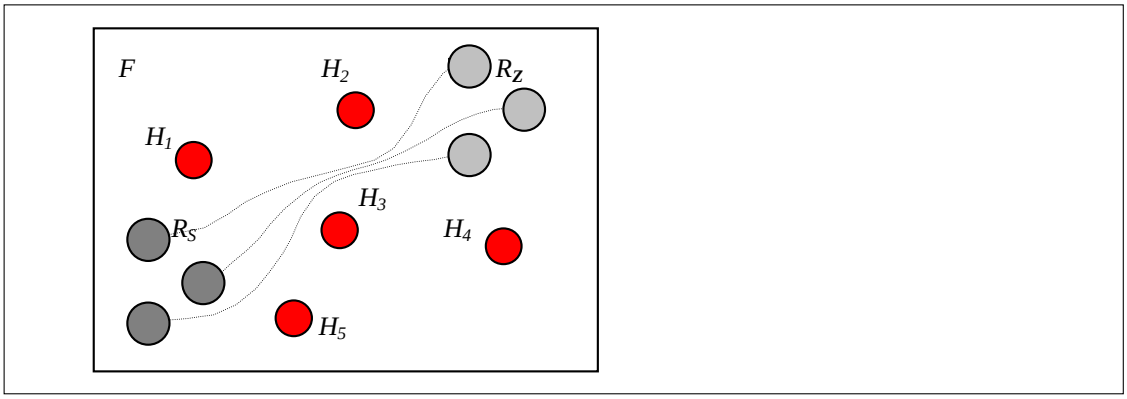


Abbildung 3.2 Problemstellung für Multirobotersystem

3.1.3 Ein Ansatz für eine hybride Planungsstrategie

Ausgehend von der beschriebenen Problemstellung und den Erkenntnissen aus Kapitel 2, soll hier nun eine hybride Planungsstrategie entwickelt werden.

Die Strategie umfasst im Wesentlichen die folgenden Schritte:

- Modellbildung
- Missionsplanung
- Verhaltensplanung
- Lokale Ausführung.

Modellbildung:
Basis für alle nachfolgenden Planungsschritte ist eine Darstellung aller a priori bekannten bzw. gewonnenen Umweltinformationen in einem entsprechenden Modell. Weiterhin ist eine geeignete Beschreibung der einzelnen Roboter als auch ihrer Formationen nötig. Diese Abstraktion geschieht im Verlauf einer Modellbildung. Im Ergebnis liegen dann drei unterschiedliche Beschreibungsformen vor, die nachfolgend als *Welt-*, *Roboter-* und *Formationsmodell* bezeichnet werden. Das *Weltmodell* beinhaltet alle Informationen über Lage und Größe bekannter Hindernisse bzw. eine Abstraktion des Freiraums in Form einer Karte. Diese Daten können über zusätzliche globale Sensorik oder aber im Verlauf einer Mission durch die Roboter selbst aufgenommen werden. Auch Kombinationen beider Varianten sind denkbar. Im

Robotermodell erfolgt die Beschreibung eines Roboters durch seine Abmessungen und seine Kinematik. Schließlich werden die Informationen zusammengefasst, die zu jeder einzelnen Formation benötigt werden. Dazu zählen in erster Linie die Verhalten und Abhängigkeiten der Roboter, um die Formation einzunehmen bzw. beizubehalten, aber auch die Abmessungen der Formation selbst, bis hin zu kinematischen Einschränkungen, die sich durch die Formation ergeben. Diese Beschreibungen bilden die Knoten eines ungerichteten *Formationsgraphen*. Die Kanten dieses Graphen beschreiben dann die möglichen Übergänge zwischen den einzelnen Formationen. Für eine optimale Auswahl benötigter Formationswechsel, werden den Kanten Kosten zugeordnet, die in Bezug auf ein Optimierungskriterium entstehen. Solche Kriterien sind beispielsweise Energie-, Zeit-, Platzbedarf oder Risiko eines Formationswechsels.

Missionsplanung:
Während der Missionsplanung wird eine Route für die gesamte Gruppe bestimmt. Dabei werden zusätzliche Randbedingungen berücksichtigt, die sich aus der Geometrie der gewählten Formation ergeben. Eine mögliche Vorgehensweise besteht darin, die Route so zu bestimmen, dass die Formation nicht geändert werden muss, um Hindernissen auszuweichen, daher einen ausreichend großen Abstand sichert. Diese Variante führt aber zu längeren Routen und unter Umständen können keine Lösungen gefunden werden, wenn z.B. eine Passage einen Wechsel der Formation erfordern würde. Deshalb muss die Missionsplanung zumindest mit einem *adaptiven* Verfahren arbeiten, um dafür eine Lösung zu finden.

Verhaltensplanung:
Die Missionsplanung beantwortet im Wesentlichen die Frage, was zu tun ist. Die sich anschließende Verhaltensplanung muss nun bestimmen, wie es zu tun ist. Dazu werden die Routensegmente *sequenziell* analysiert. Jedes Segment besitzt eine Eigenschaft, anhand derer die Verhaltensplanung das optimale Verhalten auswählen kann. Für die Wahl der optimalen Formation besitzt jedes Routensegment ein Maß für die nutzbare Breite. Die Verhaltensplanung bestimmt darüber und anhand des Formationsgraphen die Formationswechsel, die mit den geringsten Kosten verbunden sind. Es ist vorstellbar, dass es mehrere geeignete Formationen für ein Routensegment gibt. Da nicht alle Formationswechsel direkt möglich sind, kann eine schlechte Wahl der Formation zu ungünstigen Bedingungen in einem späteren Routensegment führen. Um dieses Problem zu lösen, muss die Analyse der Routensegmente nicht sequenziell, sondern *rekursiv* erfolgen. Damit ist es möglich, die Folgekosten eines Formationswechsels über die gesamte Route zu berücksichtigen.

Lokale Ausführung:
Da aus der Verhaltensplanung eine Liste angepasster Verhalten resultiert, besteht die Aufgabe der lokalen Ausführung auf den Robotern im Wesentlichen darin, die entsprechenden Verhalten zum richtigen Zeitpunkt zu aktivieren. Um Synchronisationsprobleme der Roboter untereinander zu vermeiden, werden im Formationsmodell Abhängigkeiten der Form 1:N definiert.

Damit ergibt sich folgendes Schema für die Planungsstrategie:

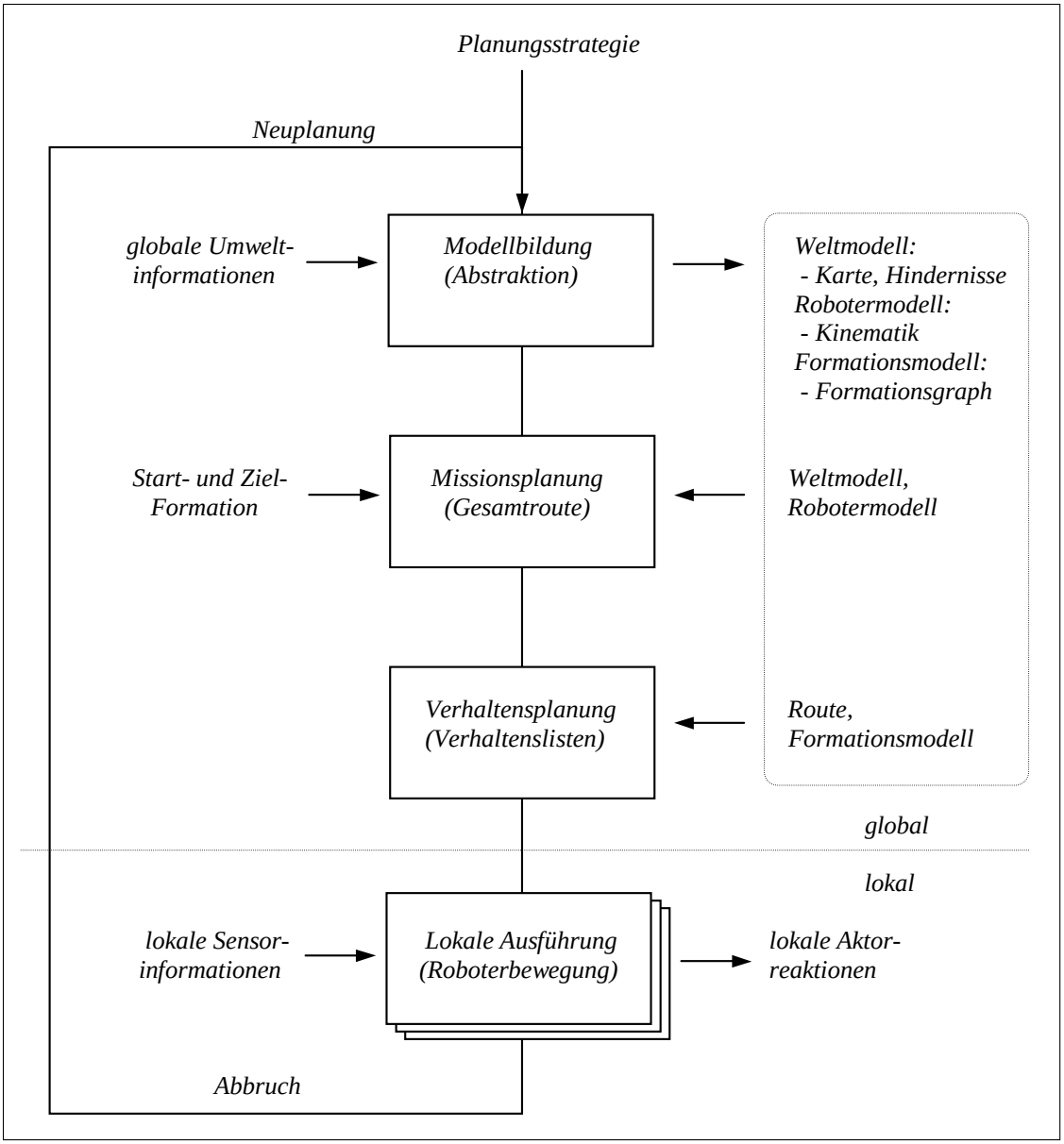


Abbildung 3.3 Planungsstrategie für Multirobotersystem

Ein wesentlicher Punkt ist die Möglichkeit, die Ausführung abzubrechen und eine Neuplanung durchzuführen, wenn bestimmte Abbruchkriterien erfüllt werden. Die Kontrolle dieser Kriterien kann dabei sowohl lokal als auch global erfolgen. Lokale Abbruchbedingungen könnten Fehlermeldungen der Roboter selbst sein oder unerwartete Hindernisse, die in der vorangegangenen Planung nicht berücksichtigt wurden. Aus globaler Sicht ist die Überwachung des Ablaufs sicher die entscheidende Kontrollmöglichkeit.

In den folgenden Abschnitten soll die Planungsstrategie anhand einer konketen Realisierung verdeutlicht werden. Dabei wurde insbesondere Wert auf eine vollständige Implementierung aller Schichten gelegt. Die dabei gemachten Vereinfachungen werden später analysiert.

3.2 Modellierung eines Multirobotersystems

Für den im vorangegangenen Abschnitt entwickelten Ansatz einer hybriden Planungsstrategie werden drei unterschiedliche Modelle benötigt; ein Robotermodell, ein Formationsmodell und ein Weltmodell. Der Begriff Weltmodell ist in der Literatur umstritten, weil eine vollständige Modellierung der Umwelt nur schwerlich möglich ist – der Begriff ist zu allgemein. Im Zusammenhang mit Bewegungsplanung wird deshalb vom Konfigurationsraum gesprochen. Für die vorgestellte Planungsstategie ist er aber insofern berechtigt, weil darin alle außerhalb des Robotersystems liegenden Einflussgrößen zusammengefasst werden. Je nach konkreter Umsetzung der Planungsstrategie kann das Weltmodell daher mehr oder weniger umfassend sein. In der vorliegenden Implementierung ist die Umgebung der Roboter (Experimentierfeld) stark idealisiert und in ihrer räumlichen Ausdehnung genau abgegrenzt und somit vollständig beschreibbar.

3.2.1 Robotermodell

Ein häufig genutztes Antriebsprinzip für Roboter ist der Differentialantrieb.

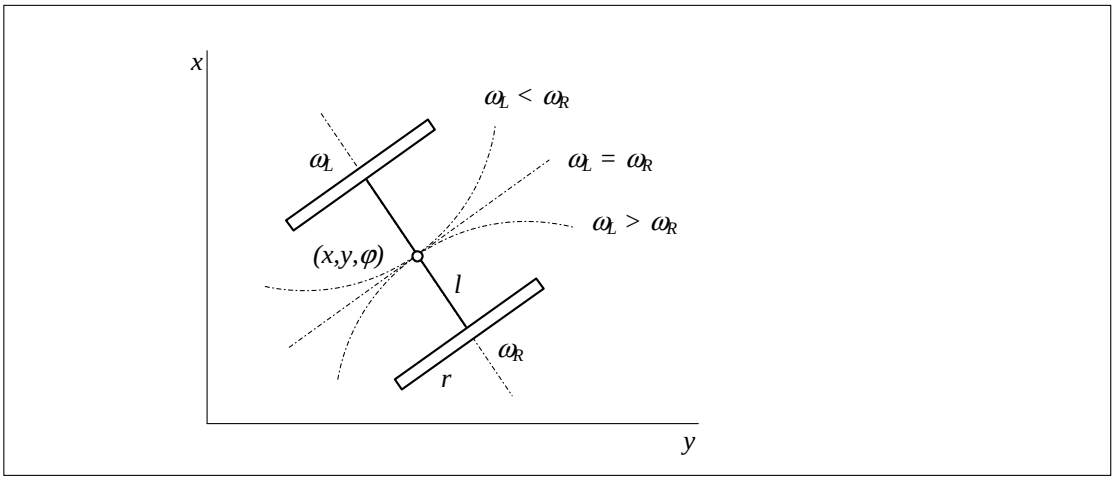


Abbildung 3.4 Prinzip des Differentialantriebs (*differential drive)*

Dabei werden zwei getrennt angetriebene Räder mit dem Radius *r* auf einer Achse im Abstand *l* angeordnet. Solange beide Räder die gleiche Winkelgeschwindigkeit $\omega_L = \omega_R$ haben, bewegt sich der Roboter auf einer Geraden. Durch Variation der Winkelgeschwindigkeiten entsteht eine Kreisbahn, deren Radius mit zunehmender Differenz immer kleiner wird. Für den Fall $\omega_L = -\omega_R$ dreht sich der Roboter schließlich um den Mittelpunkt der Achse. Die Kinematik dieses Antriebs kann allgemein durch folgende Gleichung beschrieben werden:

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta\varphi \end{bmatrix} = \begin{bmatrix} \frac{r}{2}(\omega_L + \omega_R)\cos\varphi \\ \frac{r}{2}(\omega_L + \omega_R)\sin\varphi \\ \frac{r}{l}(\omega_R - \omega_L) \end{bmatrix}. \qquad (3.1)$$

Besondere Vorteile bietet diese Antriebsart in Verbindung mit einer kreisförmigen Grundfläche des Roboters. Wenn Achs- und Kreismittelpunkt zusammenfallen, kann man von einem quasi holonomen Roboter ausgehen. Der Drehwinkel θ des Roboters muss dann bei der Routenplanung nicht mehr berücksichtigt werden. Die Dimension des Konfigurationsraumes verringert sich dadurch. Die kreisförmige Grundfläche kann außerdem zu einem Punkt reduziert werden, wenn dafür die Hindernisse entsprechend erweitert werden. Diese Vorgehensweise ist sinnvoll, da so Tests auf Kollisionen des Roboters mit Hindernissen im Konfigurationsraum auf einen Punktvergleich reduziert werden können. Es ergibt sich für einen Roboter also die folgende Darstellung im Koordinatensystem ***W*** des Weltmodells.

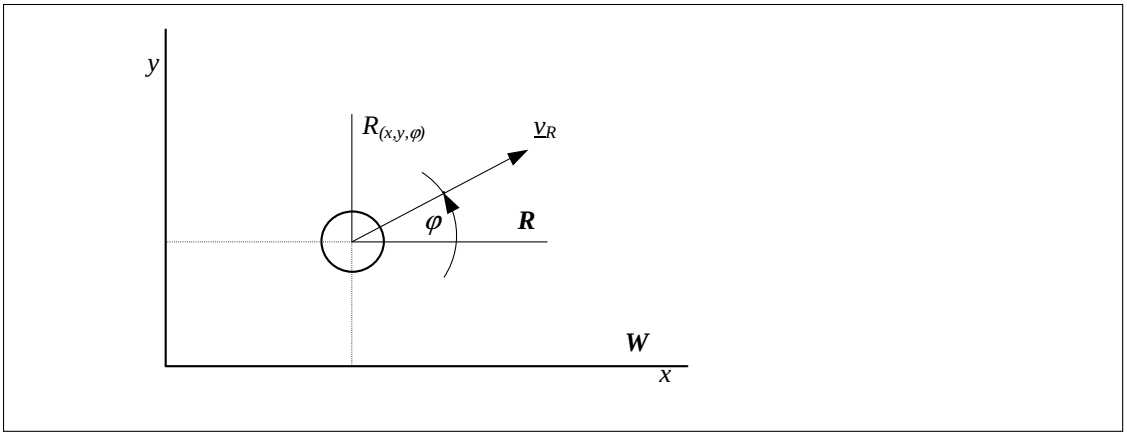


Abbildung 3.5 Roboter-Koordinatensystem im Weltmodell

Der Roboter besitzt ein eigenes Koordinatensystem ***R***, dessen Ursprung mit dem Drehpunkt zusammenfällt. Dadurch wird es einfacher, Steuervektoren zu berechnen, weil lediglich Offsets zu den eigentlichen Roboterkoordinaten betrachtet werden müssen. Die Koordinatenachsen beider Systeme bleiben jedoch stets parallel, um so die Bezüge der Roboter untereinander zu vereinfachen (vgl. Formationsmodell). Jeder Roboter kann direkt in jedes Koordinatensystem übertragen werden.

3.2.2 Formationsmodell

Eine Formation beruht im Wesentlichen darauf, dass jeder Roboter eine ihm zugewiesene Position innerhalb der Gruppe einnimmt bzw. einhält. Daraus ergeben sich zwei unterschiedliche Forderungen an die Beschreibung einer Formation:

- Positionsbeschreibung
- Verhalten zum Einnehmen/-halten

Positionsbeschreibung:
Prinzipiell wäre es möglich, ausgehend von einer gemeinsamen Route durch Bildung von entsprechenden Offsets die Positionen der einzelnen Roboter zu bestimmen. In Konsequenz gäbe es für *N* Roboter aber *N* unterschiedliche Routen. Abgesehen von der größeren Datenmenge, wäre dadurch insbesondere der Zusammenhalt der Gruppe fraglich. Im Fall von Störungen durch Hindernisse würden die Roboter unterschiedliche Ausweichrouten wählen und sich so möglicherweise verlieren.
Um diesen Nachteil zu vermeiden, wird hier eine relative Positionsbeschreibung verwendet. Dazu werden zunächst zwei unterschiedliche Arten von Robotern gebildet –

führende und *folgende* Roboter. Es gibt genau einen führenden Roboter R_1, der eine absolute Weginformation in Form einer segmentierten Route besitzt. Alle anderen Roboter folgen diesem direkt oder indirekt in geeigneter Weise. Dabei kann jeder folgende Roboter wiederum führend für andere sein. Es entstehen hierarchische Abhängigkeiten.

Die Angabe der Position eines folgenden Roboters geschieht unter Berücksichtigung der Bewegungsrichtung des führenden Roboters durch einen Führungsvektor $\underline{v}_{LF}$ mit:

$$\underline{v}_{LF} = \begin{bmatrix} d_{LF} \\ \varphi_{LF} \end{bmatrix}. \tag{3.2}$$

Damit ergibt sich die Position des folgenden Roboters R_F aus der des führenden Roboters R_L mit Hilfe der Gleichung (3.3) zu:

$$\begin{bmatrix} x_F \\ y_F \end{bmatrix} = \begin{bmatrix} x_L \\ y_L \end{bmatrix} + d_{LF} \begin{bmatrix} \cos(\varphi_L + \varphi_{LF}) \\ \sin(\varphi_L + \varphi_{LF}) \end{bmatrix}. \tag{3.3}$$

Danach benötigt jeder Roboter nur die Position seines Führungsroboters sowie den Führungsvektor, um seine Position innerhalb der Formation bestimmen zu können. Der Führungsvektor ist eine Konstante, während die Position des Führungsroboters zyklisch aktualisiert werden muss. Jeder Führungsroboter benötigt deshalb eine Liste der ihm folgenden Roboter.

Der Roboter R_1 bildet bei der Bestimmung seiner Position insofern eine Ausnahme, da er keinem Roboter, sondern der vorgegebenen Route folgt. Für Roboter R_1 entfällt daher bei Verwendung einer segmentierten Route die Berechnung der Position. Wie alle führenden Roboter muss aber auch R_1 seine Position zyklisch an die folgenden Roboter übertragen.

Verhaltensbeschreibung:
Nachdem jeder Roboter in der Lage ist, seine Position innerhalb der Formation zu bestimmen, soll nun geklärt werden, wie er diese erreichen kann. Die lokale Ausführung basiert auf Steuerverhalten, wie sie in Abschnitt 2.3 beschrieben wurden. Für die Formationsfahrt eignet sich das kombinierte Verhalten Zielverfolgung. Der Zielpunkt ist dabei die Position innerhalb der Formation. Solange eine Abweichung zwischen aktueller Roboterposition und angestrebter Zielposition besteht, wird ein Steuervektor generiert, der eine Bewegung des Roboters zur Folge hat (vgl. Abschnitt 2.3.1). Dabei wird deutlich, dass alle Roboter das gleiche grundlegende Verhalten – Zielverfolgung – besitzen, lediglich die Bestimmung der Zielpunkte unterschiedet sich bei Roboter R_1.

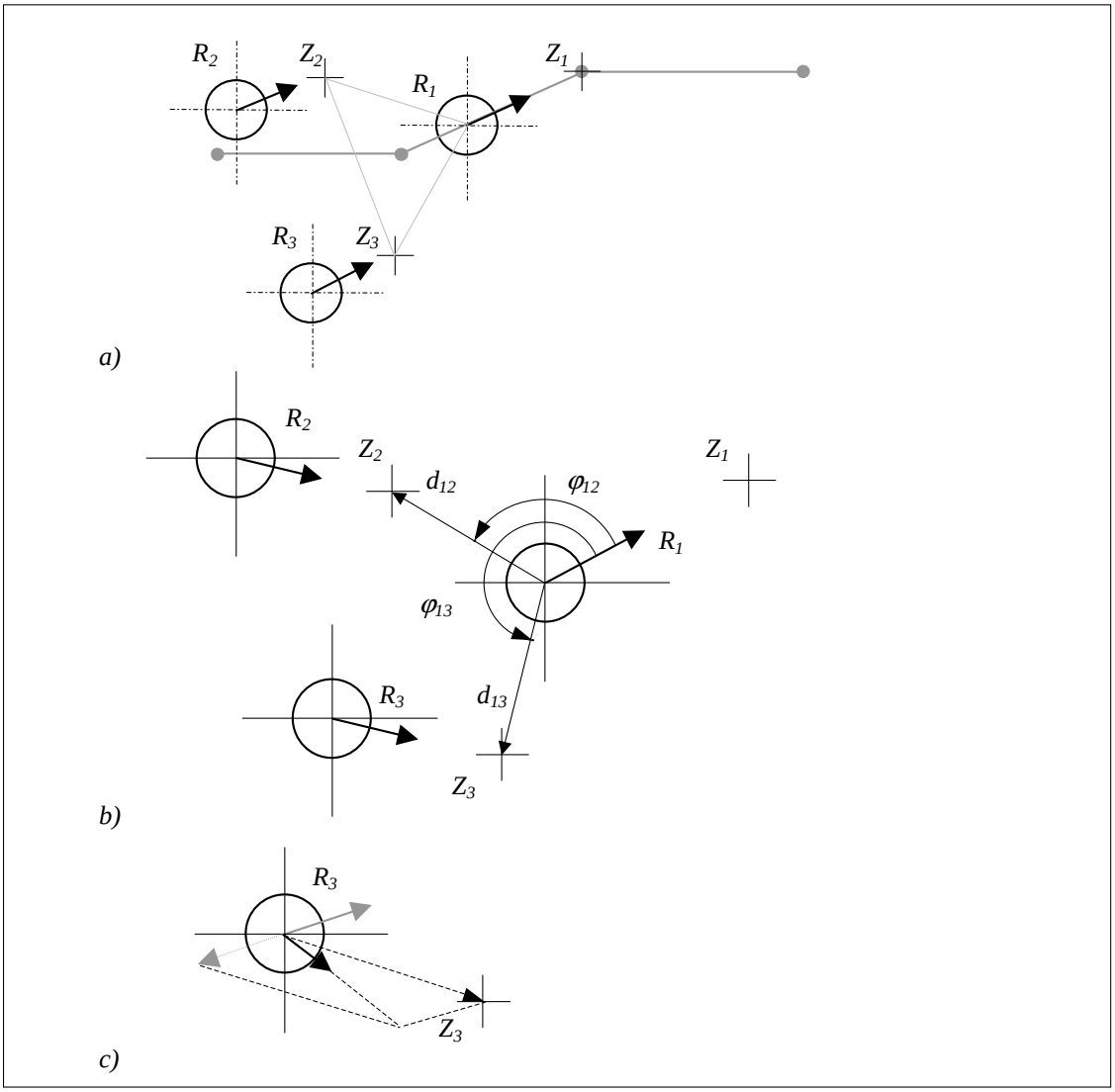


Abbildung 3.6 Steuerverhalten für Formation
a) Prinzip der Zielpunkte
b) Bestimmung der Zielpunkte
c) Bestimmung der Steuervektoren

Zusätzlich zur Formationsfahrt wurde für Passagen, die in Formation nicht befahrbar sind, das Verhalten Objektverfolgung implementiert. Dabei werden als Zielpunkte die Positionen der führenden Roboter angenommen. Roboter R_2 folgt R_1 und R_3 folgt R_2, sodass sich eine Reihe ergibt.

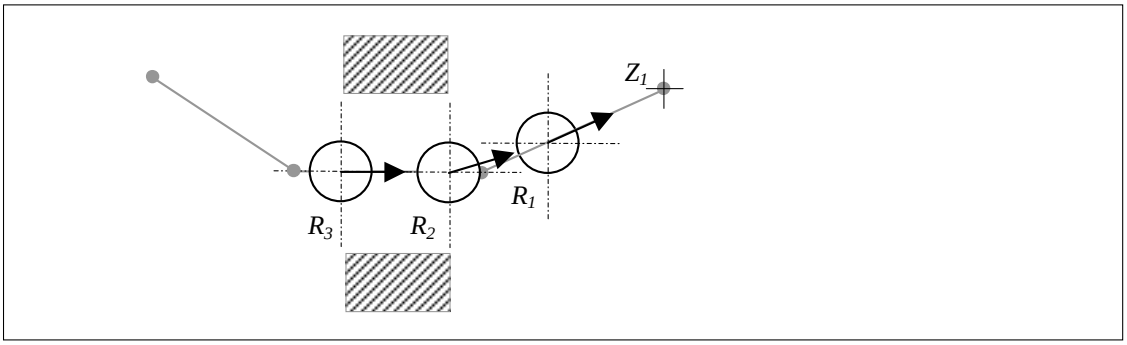


Abbildung 3.7 Steuerverhalten für Reihe

Bei der Objektverfolgung wird der Betrag des Steuervektors proportional zum Abstand begrenzt. Damit hält der folgende Roboter immer einen Mindestabstand zum führenden Roboter.

3.2.3 Formationswechsel

Es zeigt sich, dass mit einem kombinierten Steuerverhalten sowie der Vorgabe von entsprechenden Führungsvektoren für die Roboter Formationsfahrten realisiert werden können. Für Passagen muss das Steuerverhalten ggf. geändert werden. Es stellt sich also prinzipiell die Frage, wie ein Formationswechsel beschrieben werden kann.

Jede Formation wird durch eine Anzahl von Parametern pro Roboter beschrieben:

- Führungsroboter
- Führungsvektor
- Steuerverhalten.

Um eine Formation zu aktivieren, müssen jedem Roboter nur diese Parameter übertragen werden. Praktisch kann das jedoch zu Problemen führen, wenn die Roboter dann gegensätzliche Ziele verfolgen. Wenn ein Formationswechsel in Abbildung 3.8 von Formation *A* zu Formation *B* direkt erfolgt, so kann es zu Kollisionen kommen. Sofern die Roboter das Basisverhalten Abstoßung ausführen, werden Kollisionen evtl. vermieden, aber es entstehen dann Wechselwirkungen, die bis hin zu periodischen Pendelbewegungen der Roboter um ihre Zielposition führen können.
Aus diesen Gründen wird hier ein anderes Verfahren vorgeschlagen. Für Übergänge zwischen Formationen werden, wenn nötig *Übergangsformationen* eingefügt. Der in Abbildung 3.8a dargestellte Übergang von Formation *A* nach *B* erfolgt sinnvollerweise über die Formation *A'*.

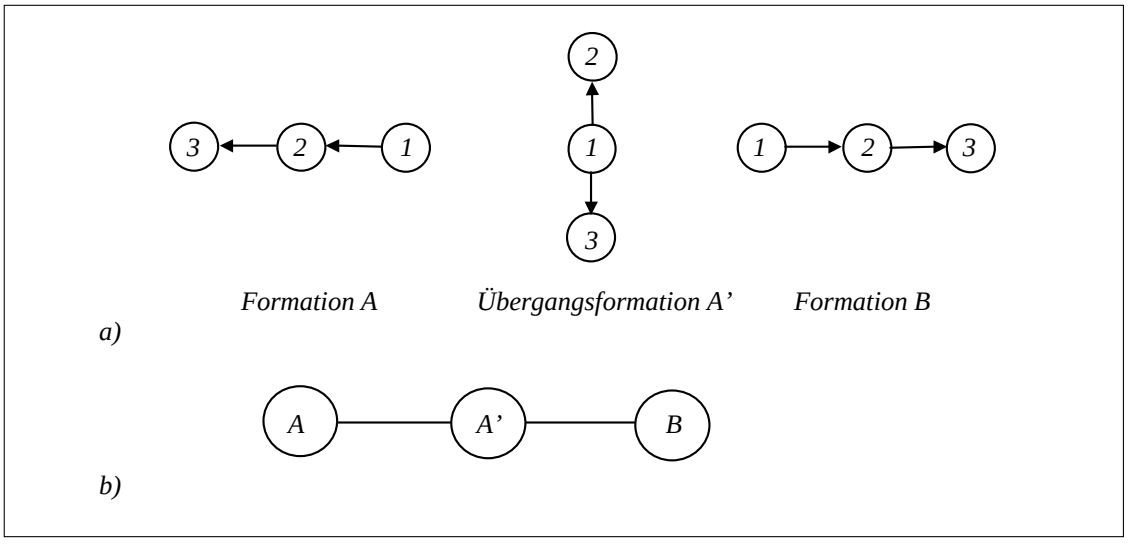


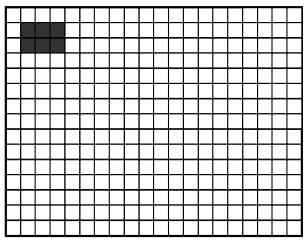
Abbildung 3.8 Formationswechsel
a) Formationswechsel mit Übergangsformation
b) Formationsgraph

Dabei ist der Formationswechsel in beiden Richtungen möglich: A-A'-B ist ebenso zulässig wie B-A'-A. Es bietet sich daher an, alle Formationen in einem ungerichteten

Graphen darzustellen. Für den Formationswechsel ist der Graph in Abbildung 3.8b dargestellt. Übergangsformationen sind generell gleichberechtigt und können ebenso für eine Formationsbewegung genutzt werden. Daraus ergibt sich die Bildungsvorschrift für einen Formationsgraph: *Alle sinnvollen Formationen werden als Knoten in den Graph aufgenommen; diejenigen, die ineinander überführt werden können, werden durch Kanten miteinander verbunden*.

3.2.4 Weltmodell und Hindernisdarstellung

Die Grundlage des Weltmodells bildet eine Karte, die den gesamten Arbeitsbereich der Roboter umfasst und außerdem eine Beschreibung aller darin bekannten statischen Hindernisse enthält. Die Wahl einer geeigneten Darstellung hängt im Wesentlichen von der räumlichen Ausdehnung des Arbeitsbereichs, der Art und Anzahl der Hindernisse sowie der Größe der Roboter, daher der erforderlichen Auflösung, ab. Für kleine Arbeitsbereiche (z.B. Experimentalsysteme) ist die äquidistante Rasterung eine praktische Herangehensweise. In [Azarm, 1997] wird eine entsprechende Darstellung sowohl für die globale Bahnplanung als auch für lokale Korrekturplanungen in Grundrissen von Gebäuden verwendet. Durch diese Zweiteilung, verbunden mit einer Reduzierung der Auflösung während der globalen Bahnplanung, bleibt die Datenmenge und damit auch die Reaktionszeit akzeptabel. Generell ist jedoch für ausgedehntere Arbeitsbereiche eine effektivere Darstellung angebracht, die entweder speziell die Hindernisse oder die Freiräume beschreibt. Im Hinblick auf die in der vorliegenden Arbeit verwendete Experimentalplattform sowie die Gewinnung der Karte durch ein Bilderkennungssystem wurde ebenfalls auf eine äquidistante Rasterung orientiert.



a)

$$\underline{G} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 1 & \dots \\ 0 & 1 & 1 & 1 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

b)

Abbildung 3.9 Karte des Weltmodells
a) Rasterkarte (*occupancy grid*)
b) Darstellung als Matrix

Die äquidistante Rasterung kann als eine *n* x *m* -Matrix dargestellt werden, wobei n der Zahl der Segmente in x-Richtung und m der Zahl der Segmente in y-Richtung entspricht. Jedes Segment wird durch einen Wert innerhalb der Matrix repräsentiert. Dabei werden freie Segmente mit 0 in die Matrix übernommen, während belegte Segmente durch einen Wert verschieden von 0 dargestellt werden. Der wesentliche Vorteil dieser Kartendarstellung liegt in der unmittelbaren Überführung in eine Datenstruktur. Vektoren können rechentechnisch sehr gut und effektiv in Algorithmen implementiert werden. Darüber hinaus existieren bereits Algorithmen zur Routenplanung, die auf einer derartigen Matrix basieren.

3.3 Missionsplanung

Unter Missionsplanung werden die Schritte zusammengefasst, die nötig sind, um eine geeignete Route für das Multirobotersystem zu bestimmen. Auf der Basis einer gegebenen Karte können aber, in Abhängigkeit von der verwendeten Kostenfunktion, höchst unterschiedliche Routen bestimmt werden.

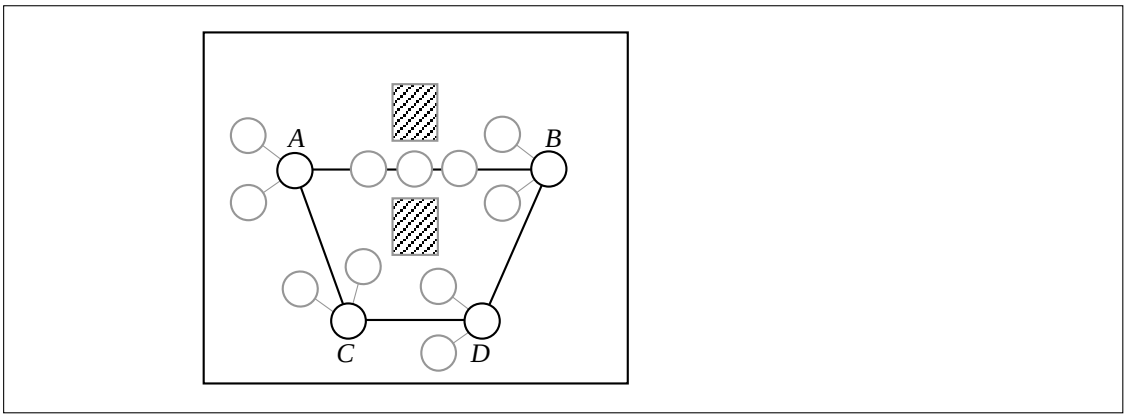


Abbildung 3.10 Varianten einer Missionsplanung

Aus Abbildung 3.10 wird deutlich, dass die Route A-B offensichtlich kürzer ist als die Route A-C-D-B. Dabei kann aber insbesondere die Engstelle dazu führen, dass die Formation der drei Roboter geändert werden muss. Es ist also erforderlich, Informationen über die Formation mit in die Routenplanung einzubeziehen.

3.3.1 Aufgaben und Struktur

In einer ersten Implementierung der Missionsplanung wurde für den Routenplanungsalgorithmus folgende Strategie gewählt:

1. Finde den kürzesten Weg, der in der gegebenen Formation befahrbar ist.
2. Akzeptiere alternativ Wege, die eine Formationsänderung erfordern.
3. Wähle davon den, der den breitesten Korridor aufweist.

Als Maß für die Befahrbarkeit einer Route in einer bestimmten Formation wird dabei die erforderliche Mindestbreite des Korridors angenommen. Nach der obigen Strategie stellt sich die Implementierung dann als ein iterativer Prozess dar. Im ersten Schritt wird für die Ausgangsformation zunächst die benötigte Breite des Korridors bestimmt. Für den so festgelegten Korridor wird im zweiten Schritt versucht, die kürzeste Route zu bestimmen. Sofern die Routenplanung nicht zu einem Ergebnis führt, wird schließlich die Breite des Korridors solange verringert, bis entweder eine Route gefunden wird oder der Korridor die Breite für einen Roboter unterschreitet.

Damit ergibt sich für die Missionsplanung folgende Struktur.

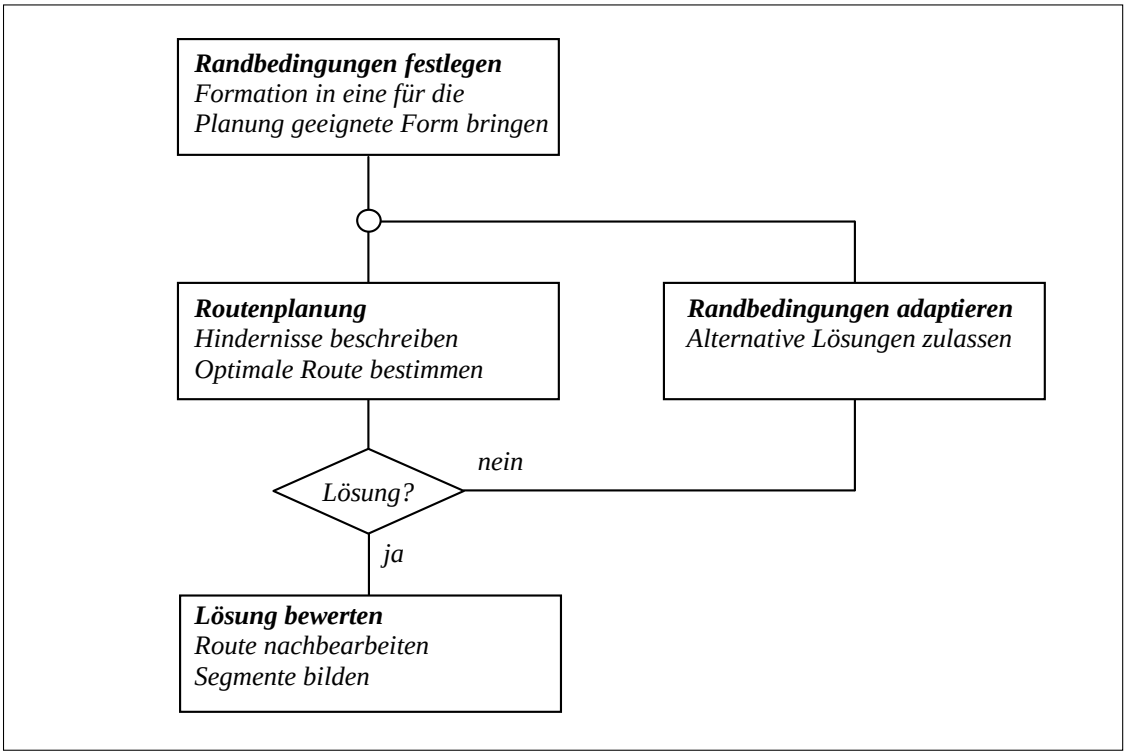


Abbildung 3.11 Struktur der Missionsplanung

3.3.2 Formation als Randbedingung

Als Randbedingung für den eigentlichen Planungsvorgang wird zunächst die Korridorbreite *B* bestimmt, die sich bei einer gegebenen Formation aus deren *Referenzkreis* ableiten lässt. Als Referenzkreis wird der Kreis um einen Roboter angesehen, dessen Durchmesser ein Maß für die gesamte Breite der Formation in Bewegungsrichtung ist. Bei Abstraktion der Formation durch ihren Referenzkreis vereinfacht sich die Routenplanung, da nur ein Punkt (Referenzpunkt) betrachtet werden muss. Die Ausrichtung der Formation sowie die der einzelnen Roboter kann unberücksichtigt bleiben. Darüber hinaus ist eine Adaption der Randbedingung, wie in Abbildung 3.11 bezeichnet, einfach durch Variation des Durchmessers möglich.

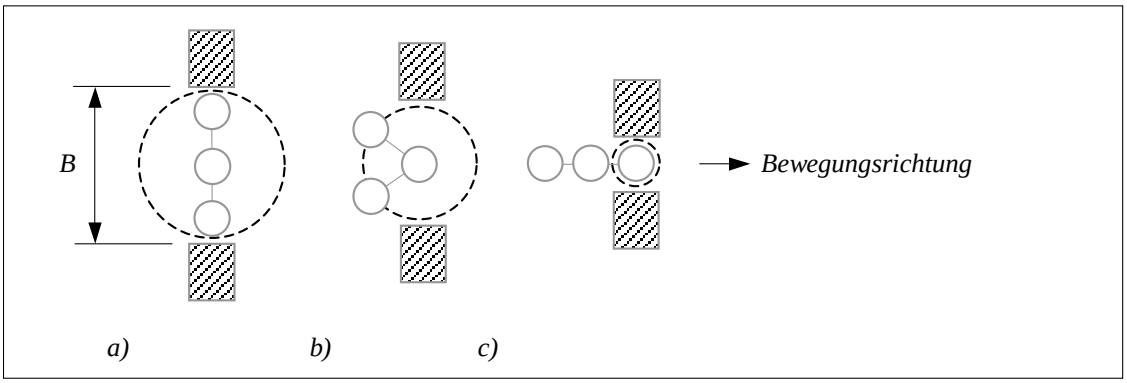


Abbildung 3.12 Referenzkreise verschiedener Formationen
a) Linie b) Dreieck c) Reihe

Abbildung 3.12 zeigt einige Formationen mit ihren Referenzkreisen. Die Roboter müssen nicht innerhalb des Referenzkreises bleiben. Es muss aber gewährleistet sein, dass die Formation einen Korridor mit entsprechender Breite passieren kann. Dabei können u.U. auch lokale Basisverhalten, wie Kollisionsvermeidung, ausgenutzt werden. Eine Formation wird eher als dynamischer Verbund angesehen, als eine starre Einheit.

Die Formation Reihe stellt einen Sonderfall dar. Diese Formation ist die Lösung für alle Passagen, die in keiner anderen Formation mehr gefahren werden können. Der erforderliche Korridor ist gerade noch so breit wie der Durchmesser eines Roboters. Prinzipiell kann natürlich jede Engstelle, die für die Ausgangsformation nicht befahrbar ist, in Reihe absolviert werden. Das ist aber aus energetischer Sicht sowie hinsichtlich der benötigten Zeit für die Formationswechsel ungünstig.

3.3.3 Bestimmung des befahrbaren Freiraums

Die Routenplanung selbst erfolgt nur für einen einzelnen Referenzpunkt innerhalb der Formation. Damit Kollisionen mit Hindernissen aber vermieden werden, müssen die Dimensionen der Formation auf die Hindernisse übertragen werden. Bei Reduzierung der Formation auf ihren Referenzkreis mit dem Radius *R*, müssen alle Hindernisse allseitig um *R* erweitert werden. Daraus resultiert schließlich der Freiraum *F*. Nur innerhalb von *F* ist der Referenzpunkt und damit die Roboter in der Formation kollisionsfrei bezüglich der bekannten statischen Hindernisse.

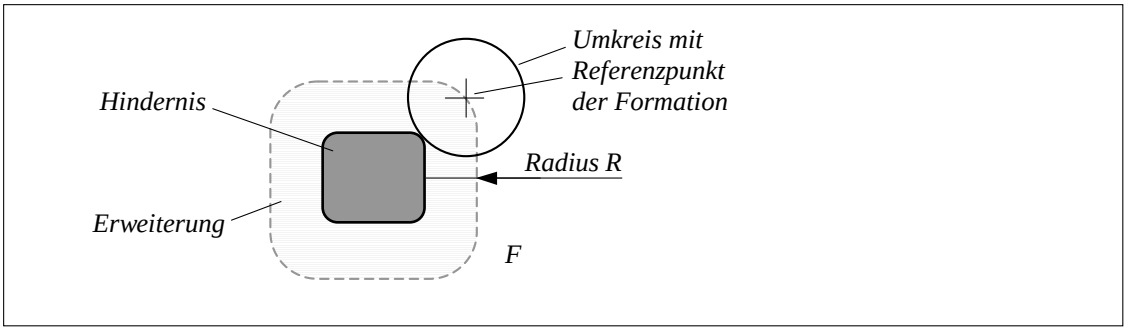
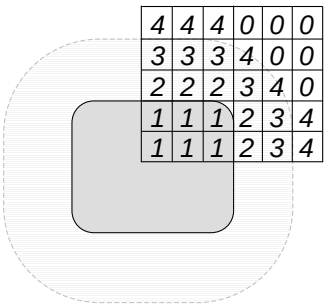


Abbildung 3.13 Prinzip der Hinderniserweiterung

Die Erweiterung der Hindernisse erfolgt sinnvollerweise als Preprozess zur eigentlichen Routenplanung. Dabei wird die zuvor erstellte Karte (*occupancy grid*) durch einen iterativen Algorithmus bearbeitet. In Analogie zu einer Art Anlagerung werden stets die Elemente des verbliebenen Freiraums, die unmittelbar an ein Hindernis grenzen, selbst als Hindernis markiert. Der Vorgang wird solange wiederholt, bis die Erweiterung den Radius *R* abdeckt. Für einen Vergleich, ob ein Element des Freiraums an ein Hindernis angrenzt, werden nur vier der acht möglichen Nachbarelemente $N(i,j)=\{(i+1,j+1),(i+1,j-1),(i-1,j+1),(i-1,j-1)\}$ verwendet. Damit wird zunächst weniger Zeit für einen Vergleich benötigt, außerdem folgt aber auch die Erweiterung besser der Kontur der Hindernisse.

```
n = 2
do
   for ∀G [ i , j ] ∈ F do
      if  G [ N( i , j ) ] = n - 1 then
          G [ i , j ] = n
   - incrementiere n
while n < R
```

a)



b)

Abbildung 3.14 Erweiterung der Hindernisse
a) Algorithmus in Pseudocode
b) Darstellung im Occupancy Grid $\underline{G}$

Im Ergebnis dieses Preprozesses kann eine Routenplanung durchgeführt werden, die anstelle der Formation nur noch einen Referenzpunkt betrachtet. Wenn eine Route für diesen Punkt gefunden wird, so ist es gleichzeitig eine Route, die auch für die Formation befahrbar ist.

3.3.4 Potentialfeld durch simulierte Wellenausbreitung

Die Routenplanung basiert auf einem künstlichen Potentialfeld, in dem ein Gradient zwischen Ausgangs- und Zielpunkt existiert. Indem der Gradient nun in Richtung eines globalen Extremwertes verfolgt wird, entsteht eine kollisionsfreie Route durch das Potentialfeld. Problematisch ist die Erstellung des Potentialfeldes insofern, da es keine lokalen Extremstellen aufweisen darf. Es existieren deshalb unterschiedliche physikalische Analogien, um geeignete Potentialfelder zu generieren.
In [Azarm, 1997] wird die Erstellung eines Potentialfeldes beschrieben, welches sich bei simulierter Diffusion ergibt. Dabei wird im Zielpunkt die Konzentration eines virtuellen Lockstoffs konstant gehalten, während mit zunehmender Entfernung die Konzentration im Freiraum abnimmt. Hindernisse besitzen die Konzentration null, weshalb sich durch Suche des steilsten Konzentrationsanstieges eine kollisionsfreie Route finden lässt.
In ähnlicher Weise wird in [Lee, 1961] die simulierte Ausbreitung einer Wellenfront für die Erstellung eines Potentialfeldes verwendet. Dabei wird das Ziel als Ausgangspunkt einer Welle angesehen. Jeder Punkt der Wellenfront ist wieder Ausgangspunkt einer neuen Welle (*Hygensches Prinzip*). Jede weitere Ausbreitung der Welle ist mit einer Erhöhung der Kosten verbunden. Die Simulation wird solange

wiederholt, bis der gesamte Freiraum mit Kostenwerten versehen ist. Schließlich wird hier, ähnlich wie beim Lockstoff-Algorithmus, der Weg durch Suche des steilsten Abstiegs ermittelt. Dieses auch als Lee-Algorithmus bekannte Verfahren wurde ursprünglich für den automatisierten Entwurf von Leiterplatten verwendet, ist aber inzwischen zu einem allgemeinen Verfahren der Routenplanung geworden und wird insbesondere dann eingesetzt, wenn schon eine Rasterkarte existiert.

Das nachfolgend vorgestellte Verfahren ist eine Abwandlung des Lee-Algorithmus mit dem Ziel glattere Wegverläufe zu erhalten. Dabei soll schon die Erzeugung des Potentialfeldes günstigere Voraussetzungen für die nachfolgende Routensuche schaffen. Grundlage des Algorithmus sind neben der Kostenfunktion insbesondere die Nachbarschaftsbeziehungen der Elemente *N(i,j)* im Verlauf der Wellenausbreitung. Da jede Zelle genau acht Nachbarzellen hat, stellt sich die Frage, wie eine annähernd kreisförmige Wellenfront generiert werden kann. Bei allseitiger Ausbreitung einer Zelle der Wellenfront ergibt sich Abbildung 3.14 a). Die Wellenfronten der einzelnen Simulationsschritte ergeben jedoch Quadrate, da insbesondere die Diagonalen auf Grund von Überlagerungen bevorzugt werden. Wird im Gegensatz dazu, nur die Ausbreitung in vier der acht Nachbarzellen simuliert, ergibt sich Abbildung 3.14 b). Eine gleichmäßige Ausbreitung in Richtung aller acht Nachbarzellen kann offensichtlich nur statistisch durch entsprechenden Wechsel der Nachbarschafts-beziehungen erreicht werden.

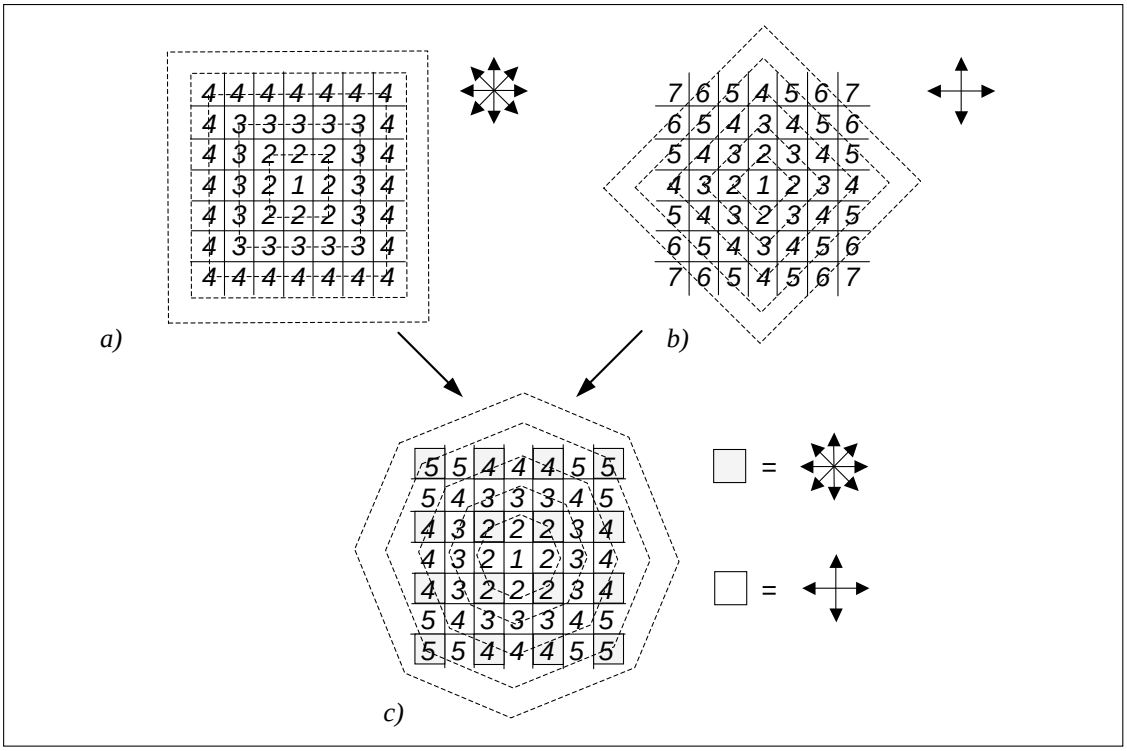


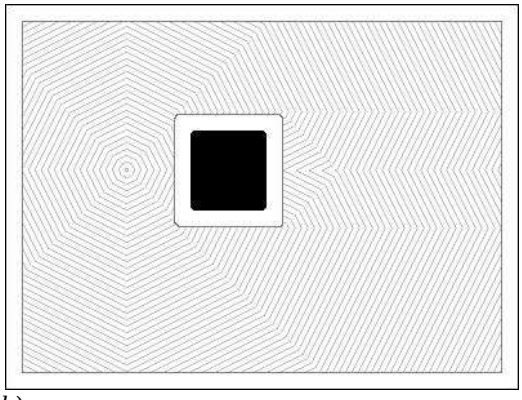
Abbildung 3.15 Simulierte Wellenausbreitung
a) Allseitige Ausbreitung ohne Vorzugsrichtungen
b) Ausbreitung nur in Vorzugsrichtungen
c) Überlagerung beider Varianten

Deshalb wird in Abbildung 3.14 c) ortsabhängig die Nachbarschaftsbeziehung und damit die Ausbreitungsrichtung gewechselt. Statistisch betrachtet, breitet sich so nur

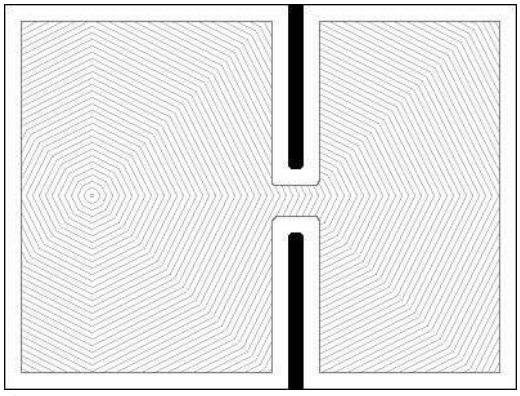
noch jede vierte Zelle in alle acht Richtungen gleichermaßen aus. Dadurch wird die Überhöhung der Diagonalen kompensiert.
Der besondere Vorteil dieser Methode besteht darin, dass der Mehraufwand bei der Berechnung des Potentialfeldes keinen nennenswerten Einfluss auf die gesamte Rechenzeit hat. Nachfolgend ist der Algorithmus mit einigen Beispielen dargestellt.

```
N_O (i, j) = {(i+1, j), (i-1, j), (i, j+1), (i, j-1)}  (d.h. die orthogonalen Nachbarzellen)
N_D (i, j) = {(i+1, j+1), (i+1, j-1), (i-1, j+1), (i-1, j-1)}  (d.h. die diagonalen Nachbarzellen)

NewList = ( i_Z , j_Z )  (d.h. trage nur Zielposition in NewList ein)
n = 0 (d.h. Anzahl der Wellenfronten – für Kostenfunktion)

do
    - kopiere NewList in OldList und lösche NewList
    - incrementiere n
    do
        - nimm nächste Position P = (i_P,j_P) aus OldList
        if G [ i_P, j_P ] ∈ F  (d.h. P ist kein Hindernis) then
            - N_P = N_O (i_P, j_P) (d.h. nur die 4 orthogonalen Nachbarzellen von P)
            if  (i_P* j_P) ist ungerade (d.h. Zellen, mit allseitiger Ausbreitung) then
                - N_P = N_P ∩ N_D (i_P, j_P) (d.h. alle 8 Nachbarzellen von P)
            for ∀G [ N_P(i_P, j_P) ] ∈ F  (d.h. Nachbarzelle ist kein Hindernis) do
                if  G [ N_P(i_P, j_P) ] = 0 (d.h. Nachbarzelle ohne Kostenwert)  then
                    G [N_P(i_P, j_P)] = K(n) (d.h. Kostenwert für n-te Wellenfront)
                    - trage N_P(i_P, j_P) in NewList ein
        - lösche P = (i_P,j_P) aus OldList
    while OldList ≠ leer
while NewList ≠ leer
```

a)



b)



c)

Abbildung 3.16 Künstliche Potentialfelder
a) Algorithmus in Pseudocode
b) Wellenausbreitung um Hindernis
b) Wellenausbreitung durch Engstelle

Die Beispiele zeigen die vollständige Berechnung des Potentialfeldes über den verfügbaren Freiraum. Wenn jedoch schon eine Startposition für die Routenplanung angenommen wird, ist es sinnvoll, den Algorithmus vorzeitig abzubrechen, sobald ein

Kostenwert an die entsprechende Zelle zugewiesen wird. Das Potentialfeld ist dann bereits soweit ausgedehnt, dass die eigentliche Routensuche gestartet werden kann.
Unter dieser Bedingung lässt sich der dargestellte Algorithmus hinsichtlich der Geschwindigkeit weiter optimieren, indem zunächst nur die Nachbarzellen mit Kostenwerten besetzt werden, die in Richtung der Startposition liegen. Erst, wenn die gerichtete Ausbreitung durch Hindernisse eingeschränkt wird, werden auch die Nachbarzellen bestimmt, die zuvor unberücksichtigt geblieben waren. Dieser Ansatz entspricht dem des A*-Algorithmus und kann durch Einbezug von Rekursion implementiert werden. Nachteilig ist jedoch, dass das Potentialfeld dann nicht immer den optimalen Weg beinhaltet, da die erste gefundene Lösung bevorzugt wird. Der möglicherweise größere Speicherbedarf ist auf leistungsfähigen Plattformen akzeptabel. Wenn aber der verfügbare Speicherplatz begrenzt ist – z.B. bei Umsetzung auf einem Kleinstroboter zur lokalen Navigation – ist ein Algorithmus mit dynamischen Datentypen oder Rekursion schwierig und unter Umständen gar nicht implementierbar. Der dargestellte Algorithmus lässt sich jedoch dahingehend umformen, dass keine Listen benötigt werden. Dabei wird ähnlich vorgegangen wie schon bei der Erweiterung der Hindernisse (Abbildung 3.13). Die Wellenfront *n* ergibt sich durch direkte Anlagerung an die Wellenfront *n-1*. Es wird lediglich der Speicherplatz zur Aufnahme der Matrix $\underline{G}$ benötigt. Die vorgestellte Erweiterung zum Lee-Algorithmus[4] bleibt erhalten, da die Wahl der Nachbarschaftsbeziehungen nur von der aktuellen Zellenposition abhängt, nicht jedoch von der verwendeten Implementierung.
Die Vorteile des vorgestellten Algorithmus liegen also in seiner einfachen Implementierung sowie der möglichen Umsetzung auf Plattformen mit geringen Ressourcen. Nachteilig ist die Abhängigkeit der Rechenzeit von der Größe der Matrix $\underline{G}$. Wobei hier, im Gegensatz zu graphenbasierten Algorithmen, die Rechenzeit mit steigender Zahl der Hindernisse abnimmt.

3.3.5 Routenplanung mittels interpolierter Gradientensuche

Nachdem im verfügbaren Freiraum ein Potentialfeld erstellt wurde, beginnt die eigentliche Routensuche. Dabei wird ausgehend von der Startposition stets der Weg des steilsten Abstiegs verfolgt. Zwei unterschiedliche Varianten werden dazu in [Azam, 1997] beschrieben. Die Ein-Schritt-Wegsuche ermittelt immer die Nachbarzelle mit dem größtmöglichen Abstieg, während die interpolierte Gradientensuche zunächst eine Ausgleichsfläche über alle acht Nachbarzellen bildet und schließlich deren Gradient für die Richtungsbestimmung verwendet. Der wesentliche Nachteil beider Varianten ist die ausschließlich lokale Betrachtungsweise des Potentialfeldes. Die daraus resultierenden Wege besitzen mitunter viele Richtungsänderungen und müssen in einem anschließenden Prozess nachbearbeitet werden.

Das nachfolgend beschriebene Verfahren ist prinzipiell auch eine interpolierte Gradientensuche. Um jedoch gleich glattere Wegverläufe zu erzielen und um eine Nachbearbeitung so einfach wie möglich zu halten, wird hier ein Suchkreis zugrunde gelegt. Dabei wird in einem Radius *r* um eine aktuelle Position *P* das Minimum des Potentialfeldes gesucht und daraus der Richtungsvektor für den nächsten Schritt ermittelt. Unter der Annahme, dass das Potentialfeld keine lokalen Extremstellen

[4] Erweiterung gilt auch in Verbindung mit dem A*-Algorithmus.

aufweist, kann der Radius des Suchkreises zunächst groß gewählt werden. Prinzipiell ist ein großer Suchkreis von Vorteil, um glatte Kurvenverläufe zu erzielen.

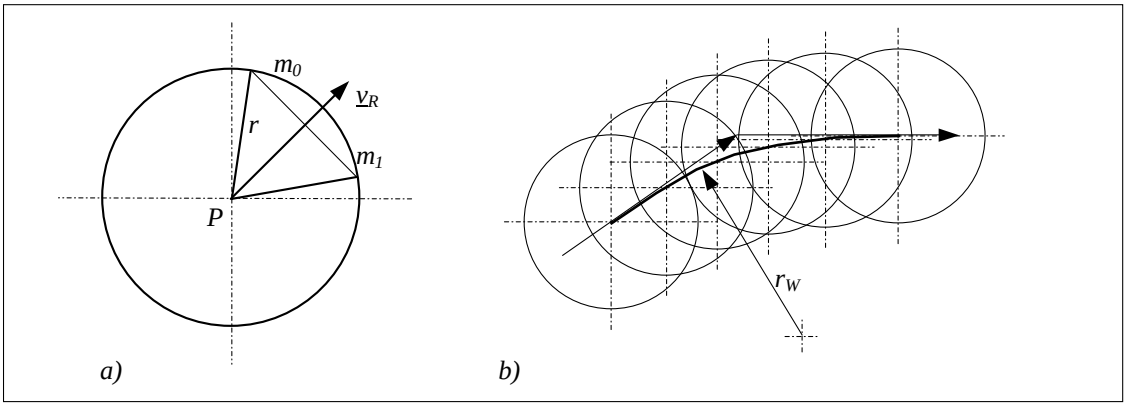


Abbildung 3.17 Suchkreisinterpolation
a) Bestimmung des Gradienten
b) Prinzip der Gradientensuche

Die Problematik der zuvor bestimmten Potentialfelder liegt in der nicht ideal kreisförmigen Ausbreitung der Wellenfronten. Dadurch haben die Gradienten stets Vorzugsrichtungen. Durch direktes Verfolgen dieser Gradienten erhält man dann Routen mit diskontinuierlichen Richtungswechseln. Durch die Interpolation der Gradientenübergänge am Radius des Suchkreises werden diese Übergänge jedoch geglättet – der Routenverlauf wird kontinuierlich. Je größer der Radius r des Suchkreises im Verhältnis zur ausgeführten Schrittweite s in Richtung des interpolierten Gradienten $\underline{v}_R$ ist, desto größer wird der Radius r_W des resultierenden Wegs. Die Bestimmung des Gradienten erfolgt in zwei Schritten. Zunächst wird das Minimum des Potentialfeldes auf dem Umfang des Suchkreises ermittelt. Durch die Diskretisierung sowohl der einzelnen Wellenfronten als auch des Suchkreises innerhalb der Matrix $\underline{G}$, ist es häufig so, dass es nicht ein Minimum, sondern vielmehr ein minimales Kreissegment zwischen $m_0 = m_1$ gibt. Die Richtung des Gradientenvektors ergibt sich dann aus der Winkelhalbierenden des Kreissegments.
Bei der Wahl des optimalen Durchmessers für den Suchkreis sind einige Einschränkungen zu beachten.

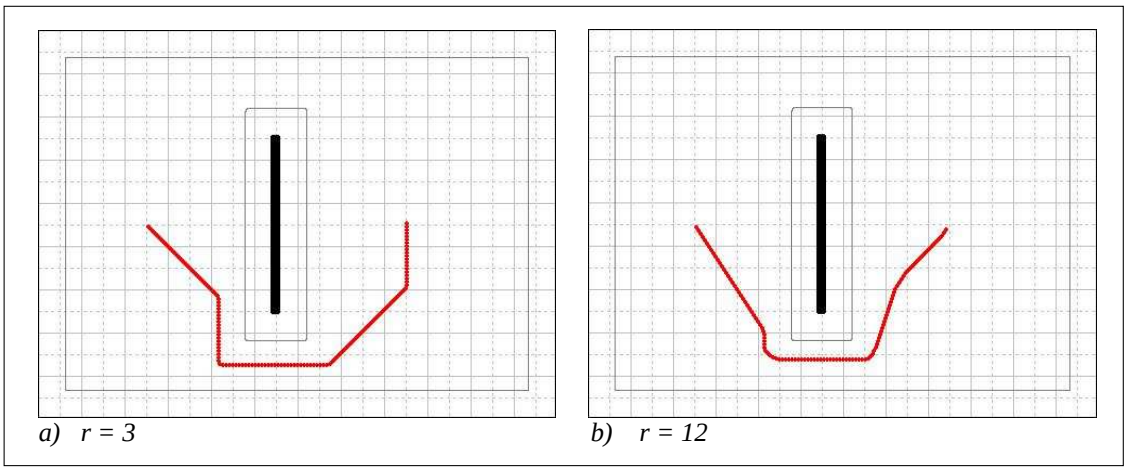


Abbildung 3.18 Auswirkung unterschiedlicher Suchkreisradien
a) Suchkreisradius zu klein (quasi Ein-Schritt-Suche)
b) Suchkreisradius günstig (1/2 Roboterradius)

Wird der Suchkreis zu klein gewählt, so verschwindet der Vorteil gegenüber der Ein-Schritt-Suche. Bei zu großer Wahl des Durchmessers können im Potentialfeld Hindernisse übersehen oder auf Grund der Vorzugsrichtungen wechselnde Gradienten verfolgt werden. Als günstig haben sich Suchkreisdurchmesser im Bereich des Durchmessers eines Roboters und darunter erwiesen. Hier spielt allerdings auch das Verhältnis der Robotergröße zur Rastergröße eine Rolle.

Insgesamt hat sich die Kombination aus Potentialfeld auf der Basis simulierter Wellenausbreitung und Routenplanung mittels Suchkreisinterpolation als leicht implementierbar und außerdem als sehr robust erwiesen. Denkbar wäre eine Optimierung hinsichtlich eines dynamischen Suchkreisdurchmessers, um der idealen Route noch näher zu kommen.

3.3.6 Segmentierung der ermittelten Route

Um die gefundene Route an den führenden Roboter der Formation übertragen zu können, muss zunächst noch eine Datenreduktion erfolgen. Die Route besteht, je nach Auflösung des Rasters, aus einer mehr oder minder großen Zahl von Wegpunkten. Viele dieser Wegpunkte bringen jedoch keine neuen Informationen bezüglich der Richtung mit sich. Deshalb ist es sinnvoll, nur die relevanten Punkt einer Route beizubehalten und alle anderen zu entfernen. Ziel ist somit die Erstellung einer segmentierten Route, wobei ein Segment die Zusammenfassung gleichgelagerter Wegpunkte darstellt.

Für die Segmentierung der Route werden deshalb zwei Arten von Wegpunkten unterschieden:

- Punkte, die ein Segment beginnen und
- Punkte, die zu diesem Segment gehören.

Der erste Punkt der Route beginnt immer ein neues Segment, während der zweite stets zu diesem Segment gehört. Die Wegpunkte werden dann sequenziell durchlaufen und dabei in die obigen beiden Gruppen unterteilt. Das Kriterium, ob ein Punkt ein neues Segment beginnt oder noch dem aktuellen Segment zugeordnet wird, ist dabei der Richtungswechsel: $\Delta\varphi = |\varphi_n - \varphi_{n-1}|$ mit $\tan\varphi = \Delta y/\Delta x$.

Da die Route auch Kurven unterschiedlicher Radien enthält, ist zusätzlich eine Schranke ε erforderlich, ab der eine Winkeländerung $\Delta\varphi$ als Richtungswechsel in ein neues Segment interpretiert wird. Zwei Varianten bieten sich an, solche Richtungswechsel zu detektieren:

- eine oder mehrere Winkeländerungen $\Delta\varphi$ überschreiten in Summe die Schranke ε oder
- die Schranke ε wird schrittweise verringert, bis eine Winkeländerung $\Delta\varphi$ sie überschreitet, danach nimmt sie wieder einen Ausgangswert ε_0 an.

Die zweite Variante bietet im Hinblick auf die möglichst exakte und sichere Überführung der geplanten Route in eine Liste von Segmenten einen Vorteil – kleine Winkeländerungen werden mit zunehmender Segmentlänge eher als Segmentbeginn erkannt. Dadurch werden einerseits Radien im Anschluss an Geraden besser

approximiert und andererseits auch kleine Winkeländerungen zwischen zwei Geraden nicht unterdrückt. Zusätzlich werden zu lange Segmente dadurch vermieden, dass die Schranke ε sehr klein werden kann und somit ein Segmentbeginn nach n Wegpunkten garantiert ist.

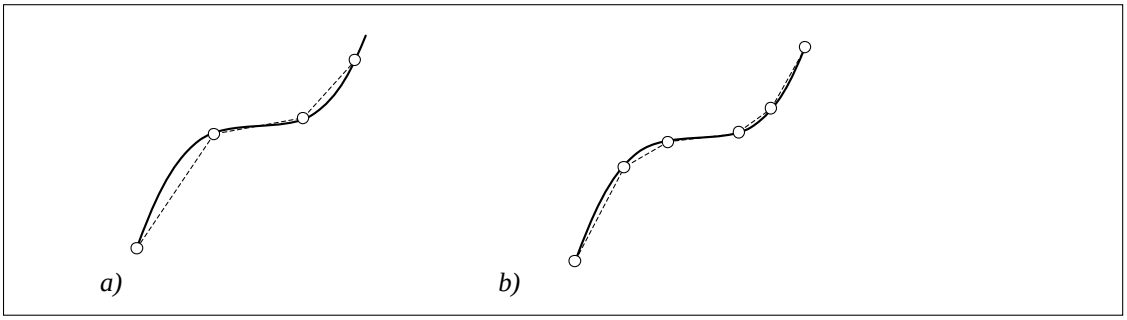


Abbildung 3.19 Vergleich unterschiedlicher Segmentierungsverfahren
a) konstante Schranke
b) fallende Schranke

Der Zusammenhang, nach dem die Schranke ε fällt, ist im einfachsten Fall linear. Im Hinblick auf eine effiziente Implementierung, hat sich jedoch folgende Beschreibung als vorteilhaft erwiesen. Der Vergleich der Schranke erfolgt nicht mit der Winkeländerung $\Delta\varphi$, sondern mit der Anstiegsänderung Δm (Differenzenquozient), damit gilt: $\varepsilon \leq \tan \Delta\varphi$. Wählt man für die Änderung der Schranke ε eine Exponentialfunktion der Form:

$$\varepsilon(t) = \varepsilon_0 c^t \tag{3.4}$$

mit $c < 1$, so kann der Verlauf der Tangens-Funktion zunächst hinreichend linearisiert werden. Für kleinere Winkel überwiegt dann der asymptotische Verlauf der Exponentialfunktion, daher ε wird nicht Null. Die Zeit t wird dem Schritt n gleich gesetzt und kann dann in die Iteration integriert werden. Damit ergibt sich der nachfolgende vergleichsweise einfache Algorithmus, der ohne Winkelfunktionen auskommt, was für kleinere Plattformen mitunter von Vorteil ist.

```
- trage ersten Punkt P0 in Segment-Liste ein
- setze n = 0
- setze ε = ε0   (d.h. t = n = 0)
do
    - incrementiere n
    - nimm Punkt Pn aus Route
    - berechne Anstieg mn zwischen Pn und Pn-1
    if | mn - mn-1 | ≥ ε then
        - trage Pn in Segment-Liste ein
        - setze ε = ε0
    else
        - setze ε = ε * c   (d.h. c < 1, z.B. 0.9, damit ε fällt)
while n < Länge der Route - 1
```

Abbildung 3.20 Segmentierungsalgorithmus in Pseudocode

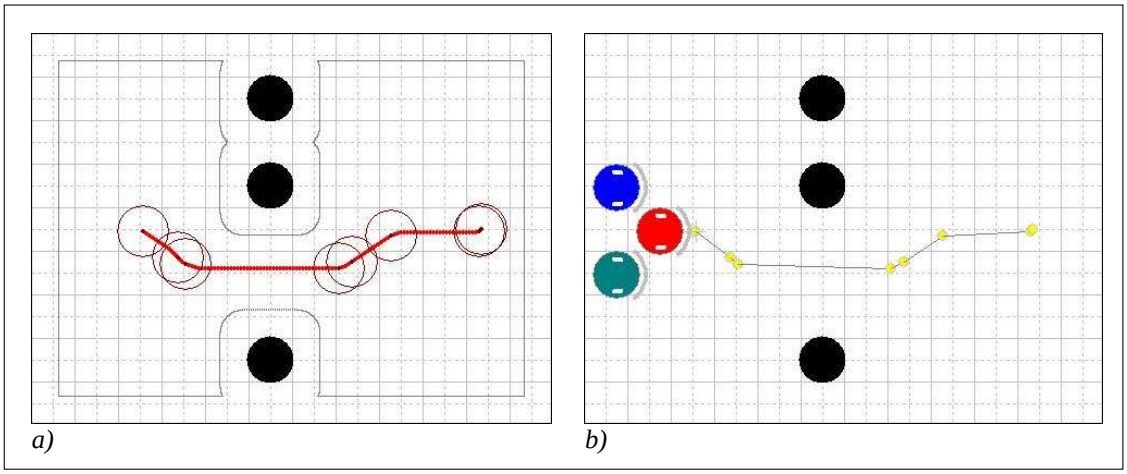


Abbildung 3.21 Segmente einer geplanten Route
a) Detektierte Segmentpunkte
b) Resultierende Liste des führenden Roboters

Damit ist die Missionsplanung abgeschlossen. Es existiert eine Beschreibung für die Route, die für die Formation befahrbar ist, in Form einer Segmentliste. Diese Liste kann an den führenden Roboter übertragen werden.

3.4 Verhaltensplanung

Nur ein Roboter der Formation kennt die zu fahrende Route. Alle anderen Roboter folgen diesem direkt oder indirekt. Damit nun jeder Roboter seine Position einnehmen bzw. halten kann, benötigt er eine Beschreibung des dafür geeigneten Verhaltens. Ausgehend vom vorgestellten Formationsmodell (vgl. Abschnitt 3.2.2) sowie einer üblichen funktionalen Beschreibung von Verhalten (vgl. Abschnitt 2.2.1) stellt sich ein Verhalten hier als eine Struktur aus beschreibenden Informationen (Eigenschaften, Parametern) sowie einem dazugehörenden Zusammenhang zwischen Eingangs- und Ausgangsgrößen (Methoden) dar. Ein Verhalten kann demnach auch als ein *Objekt* im Sinne einer objektorientierten Betrachtungsweise angesehen werden. Nach [Odell, 2002] wäre die Bezeichnung *Agent* sogar treffender, da ein Objekt primär als passiv betrachtet wird, während ein Agent selbsttätig und zielgerichtet aktiv wird. Das ist insofern gegeben, da ein Verhalten ja durch die unmittelbare bzw. kontinuierliche Reaktion auf Eingangsgrößen gekennzeichnet ist. Diese Unterscheidung soll nachfolgend jedoch nicht weiter verfolgt werden, da durch entsprechende Implementierung in Form von aktiven Objekten mit eigenem Ausführungskontext (Task/Thread) der Übergang fließend wird. Stattdessen sollen die Verhalten im Rahmen der hier vorgestellten Planung als *Verhaltensobjekte* bezeichnet werden. Dabei stellen Verhaltensobjekte Instanzen dar, die entsprechend den äußeren Gegebenheiten und auf Basis zuvor hinterlegter Definitionen dynamisch erstellt bzw. auch wieder entfernt werden.

Die Nutzung einer objektorientierten Beschreibung bietet dabei ein hohes Maß an Flexibilität. So lassen sich übliche Beschreibungsformen für Verhalten (vgl. Abschnitt 2.2.1) durch Elemente des objektorientierten Paradigmas nachbilden:

- Sensor- und Aktorinformationen in Form von Nachrichten und Botschaften

- Zustandsautomaten (FSA/FSM) durch Kapselung von Instanzvariablen mit entsprechenden Methoden
- Subsumption in Form von Vererbung, Redefinition und dynamischem Binden
- Schemata durch Typisierung und Überladung.

Verhaltensobjekte können so recht unterschiedlich implementiert werden. Die Aufgabe der Verhaltensplanung ist es, unabhängig davon, die Definitionen der benötigten Verhaltensobjekte sowie Kriterien für deren Instanziierung zu generieren.

Vor der eigentlichen Verhaltensplanung muss jedoch noch der Aufbau bzw. die Funktionalität der lokalen Ausführung definiert werden. Offensichtlich benötigen die Verhaltensobjekte eine einheitliche Schnittstelle, über die sie mit Sensorinformationen versorgt werden und über die sie ihre Verhaltensreaktion an die Roboterplattform zurückgeben.

3.4.1 Lokale Steuerarchitektur

Die Steuerarchitektur eines Roboters teilt sich in einen Sensorpfad und einen Aktorpfad auf. Dabei bildet die Plattform mit den daran angebrachten Sensoren und Aktoren die Schnittstelle zwischen Umwelt und Roboter. Der Roboter wirkt über seine Aktoren auf die Umwelt ein und beobachtet die Reaktionen mittels seiner Sensoren. Deshalb bilden ein Pilot im Aktorpfad und ein Beobachter im Sensorpfad die untere Ebene der Steuerarchitektur. Der Beobachter stellt alle erforderlichen Informationen kontinuierlich zur Verfügung (*blackboard*). Dazu gehören in erster Linie die Informationen der eigenen Sensoren – z.B. Lage und Position anhand der Radencoder (vgl. Gl. 3.1) oder Abstand zu Hindernissen durch Auswertung der IR-Sensoren. Für die Koordination der Bewegungen mit anderen Robotern, müssen aber auch deren Positionen bekannt sein. Diese können nur über eine externe Kommunikation an den Beobachter übermittelt werden, da die eigene Sensorik sie im Allgemeinen nicht direkt ermitteln kann. Ein globaler Beobachter registriert die Positionen der einzelnen Roboter und übermittelt sie über einen Telemetriekanal.
Der Pilot ist mit einer Reihe von Basisverhalten ausgestattet, die beispielsweise eine Ausweichbewegung zur Folge haben, wenn über die IR-Sensoren eine bevorstehende Kollision mit einem Hindernis erkannt wird. Wie dabei das Verhalten „Kollisionsvermeidung“ definiert ist, ist dem Piloten a priori vorgegeben. Die benötigten Informationen, beispielsweise die der Abstandssensoren, erhält der Pilot direkt vom Beobachter. Es besteht deshalb eine direkte Verbindung zwischen Beobachter und Pilot. Ausweichen ist jedoch nicht immer das gewünschte Verhalten bei Ännäherung an ein Hindernis. Wenn es situationsabhängig besser ist, vor dem Hindernis anzuhalten, muss das Verhalten „Kollisionsvermeidung“ entsprechend substituiert werden. Die Basisverhalten stellen demnach nur ein Notprogramm dar und müssen für eine sinnvolle und vor allem zielgerichtete Bewegung entsprechend erweitert oder ersetzt werden.
Dieser Ansatz entspricht sowohl einer verhaltensbasierten hierarchischen Architektur (*subsumption*) als auch einer objektorientierten Sicht.

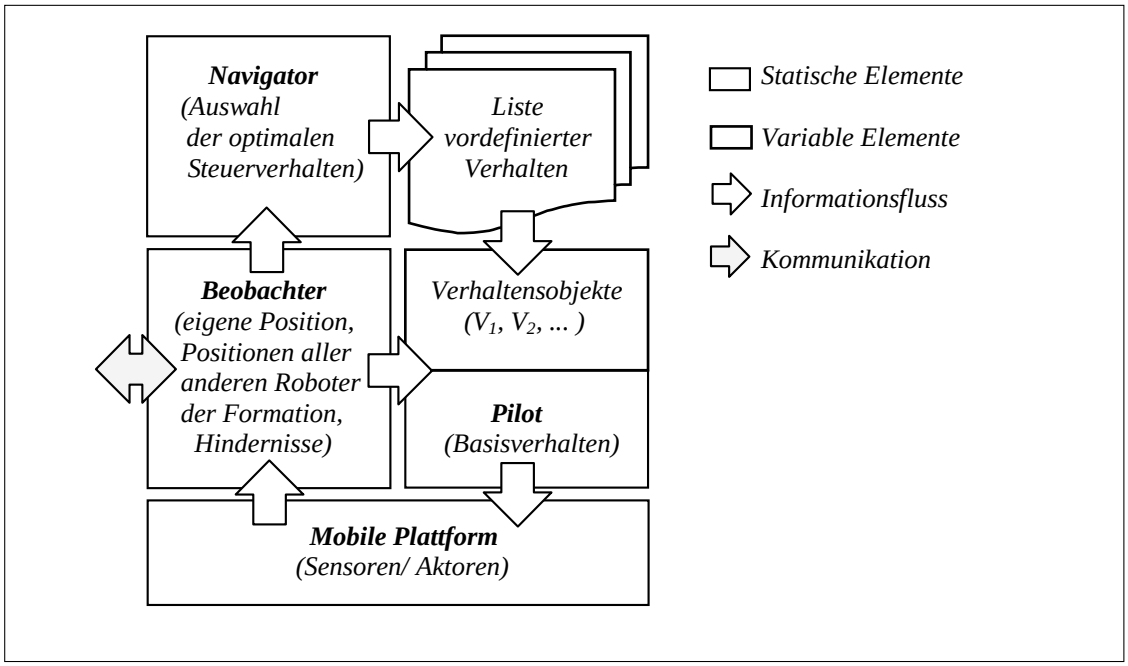


Abbildung 3.22 Steuerarchitektur eines Roboters

Die obere Ebene bildet der Navigator, dessen Aufgabe darin besteht, auf der Basis der kontinuierlichen Informationen des Beobachters den Piloten mit dem jeweils optimalen Verhalten zu versehen. Dazu existiert eine Liste vorab definierter Verhalten, die einzeln oder auch in Kombination instanziiert und an den Piloten übergeben werden können. Für den Piloten sind diese Verhaltensobjekte das Bindeglied zwischen den Sensorinformationen des Beobachters einerseits und einer Verhaltensreaktion, die der Pilot in Steuersignale für die Hardware (*Motoren*) umwandeln kann, andererseits.

Die Verhaltensobjekte implementieren also immer genau zwei Schnittstellen (*interfaces*) – einerseits zum Beobachter *IBeobachter* und andererseits zum Piloten *IPilot*. Hinzu kommen Konfigurationsparameter, die das implementierte Verhalten beeinflussen und schließlich besteht das Verhaltensobjekt noch aus einer geeigneten Beschreibungsform, die in einer Binärform, einem Skipt oder gar als Quellcode vorliegen kann.

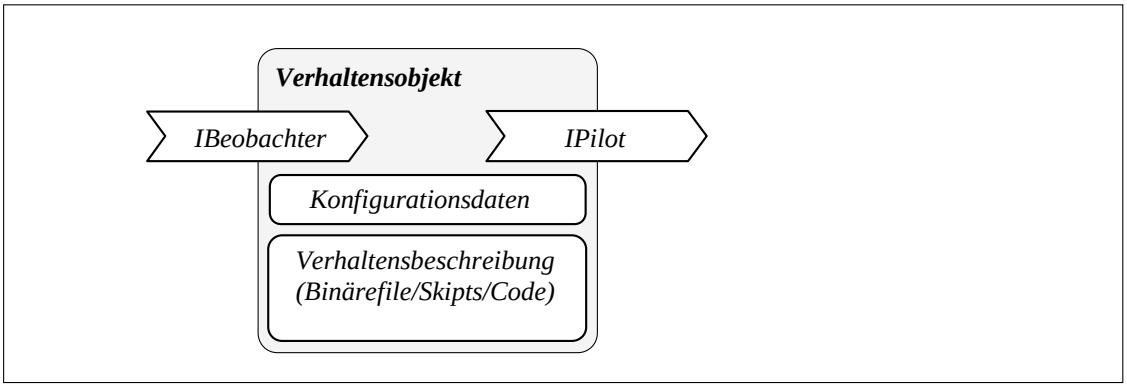


Abbildung 3.23 Schnittstellen und Eigenschaften eines Verhaltensobjekts

Diese Darstellung (Abbildung 3.23) ähnelt weitestgehend der abstrakten Darstellung eines Verhaltens in Form eines SR-Diagramms – hier entsprechend auf die spezielle Architektur der lokalen Ausführung angepasst.

Damit ist die Grundlage für eine Verhaltensplanung geschaffen. Alle Verhalten haben den gleichen strukturellen Aufbau sowie die gleichen Schnittstellen. Sie können dynamisch instanziiert und entfernt werden und verwalten ihre Informationen lokal. Auf Grund der Nutzung dieses objektorientierten Ansatzes zur Beschreibung ist ebenso eine Vererbung von Basisverhalten realisierbar.

Es bleiben zwei Fragen offen, die im weiteren Verlauf zu klären sind:

- Wie wird die Liste der verfügbaren Verhalten erstellt?
- Wie wählt der Navigator das momentan geeignete Verhalten aus?

In [Röfer, Müller, 1998] wird dazu für einen autonomen Rollstuhl eine Lösung vorgeschlagen, die eine Reihe von Basisverhalten wie Wandverfolgung links/rechts, Einbiegen in Korridor links/rechts und Zentrieren im Korridor implementiert. Die sequenzielle Abfolge dieser Verhalten stützt sich dabei auf die Auswertung von Landmarken im Umkreis des Rollstuhls. Diese wurden in einer Lernfahrt aufgenommen und dienen bei allen nachfolgenden Fahrten als Referenzpunkte. Das Besondere dieser Lösung liegt im Verzicht auf eine explizite Routenbeschreibung in Form von Koordinaten. Der zurückgelegte Weg ergibt sich ausschließlich durch die Abfolge der aktivierten Verhalten.

Ein ähnlicher Ansatz soll für die Roboter der Formation gelten. Sie passen ihre Steuerverhalten entsprechend den örtlichen Gegebenheiten an. Dazu wird dem führenden Roboter eine Liste von Synchronisationspunkten mitgegeben, an denen er entsprechende Steuerkommandos an die nachfolgenden Roboter übermitteln soll. Anhand dieser Kommandos erfolgt bei den nachfolgenden Robotern dann die Instanzierung eines passend hinterlegten Verhaltens. Daraus lassen sich die folgenden Teilaufgaben für die Verhaltensplanung ableiten.

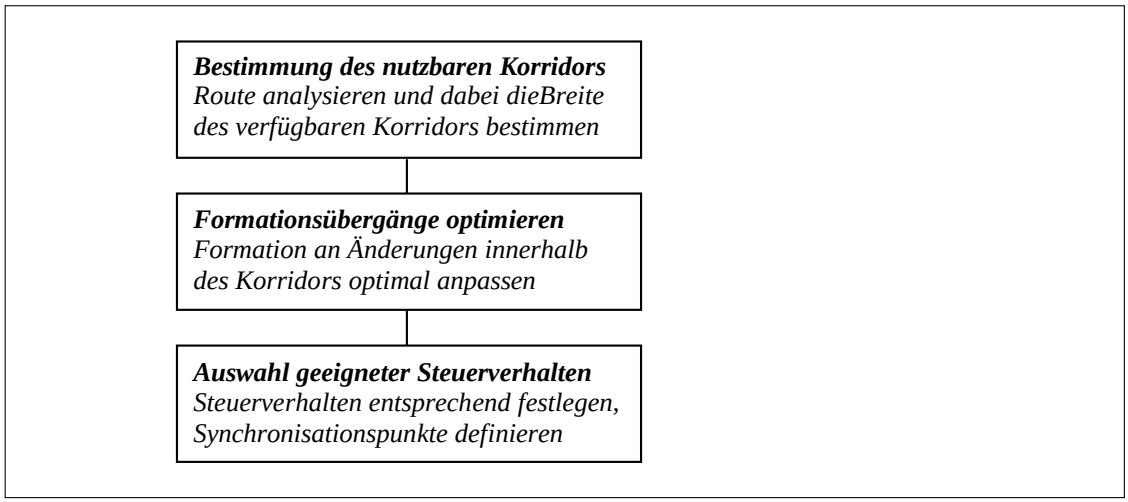


Abbildung 3.24 Teilaufgaben der Verhaltensplanung

Die Bestimmung des nutzbaren Korridors bezieht sich auf die gesamte Formation und bildet somit die Grundlage für die beiden nachfolgenden Schritte. Die Definition von

Synchronisationspunkten betrifft im Wesentlichen den oder die führenden Roboter während schließlich die Auswahl der passenden Steuerverhalten nur die folgenden Roboter betrifft.

3.4.2 Analyse der geplanten Route

Die Analyse der im Verlauf der Missionsplanung bestimmten Route, beginnt mit der Bestimmung des nutzbaren Korridors, dessen Mittellinie die geplante Route darstellt. Dazu wird sequenziell entlang der Routensegmente der kleinste Abstand zu Hindernissen bestimmt. Eine mögliche Lösung besteht darin, den maximalen kollisionsfreien Umkreis zu jeder Stelle s_i entlang der Route zu bestimmen. Dieser Ansatz entspricht genau dem, der bereits bei der Missionsplanung zugrundegelegt wurde. Das dabei vorgestellte Routenplanungsverfahren wählt zwar die Route mit der maximalen Korridorbreite, dabei gehen aber die konkreten lokalen Gegebenheiten verloren. Außerdem können durch die Segmentierung geringfügige Abweichungen der Route entstehen, die somit hier wieder berücksichtigt werden.

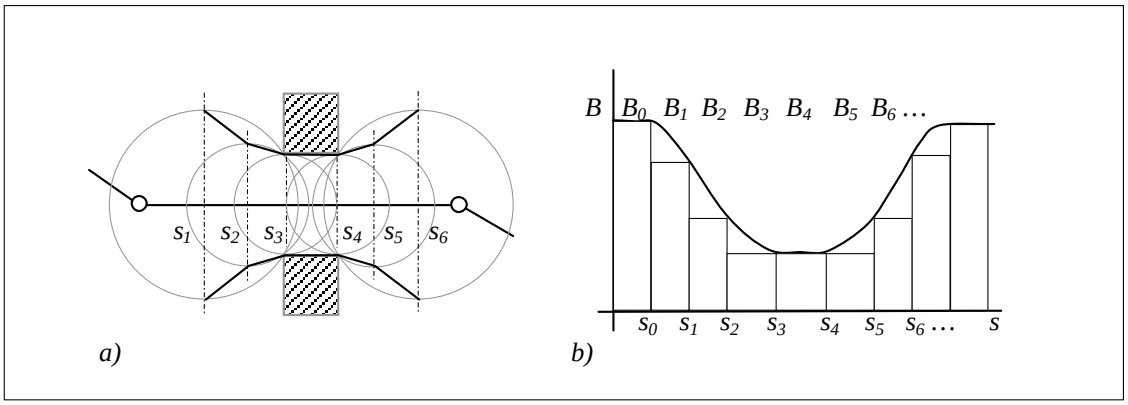


Abbildung 3.25 Analyse der geplanten Route
a) Bestimmung kollisionsfreier Umkreise
b) Korridorbreite B über der Wegstrecke s

Die Suche beginnt sinnvollerweise mit dem Durchmesser der minimal erforderlichen Korridorbreite. Die Route muss an allen Stellen stets mindestens diese Korridorbreite aufweisen. Solange der Durchmesser kollisionsfrei bezüglich bekannter Hindernisse ist, wird er erhöht. Werden Schnittpunkte mit Hindernissen erkannt, so wird der letzte Durchmesser als maximale Korridorbreite für diese Stelle angenommen. Danach wird bei jedem weiteren Schritt entlang der Route stets versucht, zunächst den maximalen Durchmesser wieder zu erreichen. Auf diese Weise folgt der Durchmesser *D* der Kreise an den Stellen s_i genau dem Verlauf der Korridorbreite *B* über den Weg *s*. Abbildung 3.26 stellt den Algorithmus in Pseudocode dar.

An dieser Stelle ist das Ergebnis der Analyse lediglich der Verlauf der Korridorbreite über die Strecke *s* der Route. Durch Wahl eines anderen Routenplanungsverfahrens kann dieser Verlauf unter Umständen schon während der Missionsplanung entstehen oder dieser vorausgehen (Voronoi-Diagramme). Prinzipiell ist die explizite Analyse der Route aber in der Lage, weitere Ergebnisse zu liefern, die nicht direkt in die Routenplanung integriert werden können. So kann beispielsweise auch die relative

Lage der Hindernisse zur geplanten Route abgeschätzt werden. Ein solches Verfahren würde den Robotern zusätzliche Weginformationen, zum Beispiel in Form von *Prädikaten*[5], geben können. Damit ist lokal eine Korrektur der aktuellen Position möglich, wenn keine absolute Position verfügbar ist. Umgekehrt kann aber auch das gezielte Aktivieren von Steuerverhalten durch Auswertung der Prädikate erfolgen. Gerade Letzteres bringt enorme Vorteile, wenn die der Planung zugrunde gelegten Umweltinformationen unscharf sind. So kann ein Routensegment mit einer Sensorkonfiguration verbunden werden, die der lokalen Ausführung den exakten Beginn des Segments signalisiert.

Zusätzliche Informationen über Richtungsänderungen entlang der Route, die eventuelle Formationsänderungen erfordern, sind ebenfalls durch eine solche Analyse erkennbar.

```
- setze Durchmesser D = B_min
- setze n = 0
do
    - nimm Punkt P_n aus Route
    if D an P_n ist kollisionsfrei then
        while D an P_n ist kollisionsfrei do
            - D = D + ε
        - setze B_n = D - ε
    else
        while D an P_n ist nicht kollisionsfrei do
            - D = D - ε
        - setze B_n = D
    - incrementiere n
while n < Länge der Route - 1
```

Abbildung 3.26 Algorithmus zur Routenanalyse in Pseudocode

In der vorliegenden Implementierung beschränkt sich die Routenanalyse nur auf die Bestimmung des Verlaufs der Korridorbreite *B* entlang der geplanten Route. Auf dieser Basis erfolgt nun die Definition von Synchronisationspunkten, die während der Ausführung zur Aktivierung geeigneter Steuerverhalten führen soll.

3.4.3 Optimaler Formationswechsel

Auf Grund unterschiedlicher Korridorbreiten entlang der Route werden Formationswechsel nötig. Dabei ist der Fall, dass der Korridor für die derzeitige Formation zu schmal wird, einsichtig. Aber auch der umgekehrte Fall ist denkbar, wenn die Formation, z.B. bei Suchaufgaben, stets die gesamte Breite des Korridors überstreichen soll. Es existiert also ein Zusammenhang zwischen der Korridorbreite *B* und einem Kriterium K_B, welches für die Auswahl der Formation zugrundegelegt wird. Für den Fall: $K_B = B$ wird die Korridorbreite B selbst als Merkmal bei der Auswahl der Formation verwendet, d.h. es wird stets die Formation verwendet, die den verfügbaren Korridor maximal ausnutzt.

[5] Unter einem Prädikat wird hier die Zuweisung einer Sensorsicht *V* zu einer bestimmten Situation *S* verstanden. Danach ist das Prätikat *P(V,S)* dann erfüllt, wenn sich in der Situation *S* die Sicht *V* einstellt; z.B. WAND(Sensor1,Hindernis).

Im Formationsgraph existieren meist mehrere Formationen, die innerhalb der verfügbaren Korridorbreite eingenommen werden können. Es wird deshalb außerdem eine Kostenfunktion $E = \Sigma e_i$ aufgestellt, die sich aus der Summe aller Teilkosten e_i der gewählten Formationswechsel ergibt. Dabei wird angenommen, dass bei jedem Formationswechsel Kosten hinsichtlich Energie- oder Zeitbedarf enstehen, die minimiert werden sollen. Andere Kostenfunktionen sind vorstellbar, um z.B. die Risiken einzelner Formationswechsel so gering wie möglich zu halten.

Die Informationen über die Kosten/Risiken eines Formationswechsels werden im Formationsgraph als Eigenschaften der Kanten *E* eingetragen, während das Auswahlkriterium, als Eigenschaft der Formation, den Knoten *V* zugeordnet wird. Zur Bestimmung eines optimalen Formationswechsels, bietet sich zunächst die Traversierung des Graphen mittels einer Breitensuche an. Im Ergebnis liegt dann ein Suchbaum vor, der alle möglichen Formationsübergänge beinhaltet. Der optimale Formationswechsel ergibt sich aus der Anwendung der Kostenfunktion auf alle Pfade des Suchbaums und anschließendem Vergleich der Ergebnisse.

Als Erweiterung zur Breitensuche muss bei der Traversierung des Graphen jedoch eine schrittweise *Maskierung* aller nicht einnehmbaren Formationen erfolgen. Dazu wird für jede diskrete Korridorbreite $B_{0..n}$ ein gültiger Formationsgraph gebildet, indem alle Knoten, deren Formationen nicht einnehmbar sind, einschließlich ihrer Kanten entfernt werden. Die so maskierten Formationsgraphen werden durch eine Bildungsvorschrift zu einem gemeinsamen – damit routenabhängigen – Formationsgraphen G_R kombiniert.

Die Bildungsvorschrift zur Verbindung aller zuvor maskierter Formationsgraphen G_0 bis G_n lautet:

$$G_R = \bigcup_{i=0}^{n} (V_i, E_i) \cup \bigcup_{i=0}^{n-1} (\{v_i, v_{i+1}\} : v_i \in V_i \wedge v_{i+1} \exists V_{i+1}). \tag{3.5}$$

Der resultierende Graph beinhaltet daher die Knoten V_i und Kanten E_i aller maskierten Graphen sowie eine Verbindung aller Knoten eines Graphen mit den entsprechenden Knoten des nachfolgenden Graphen, sofern diese nicht maskiert sind.

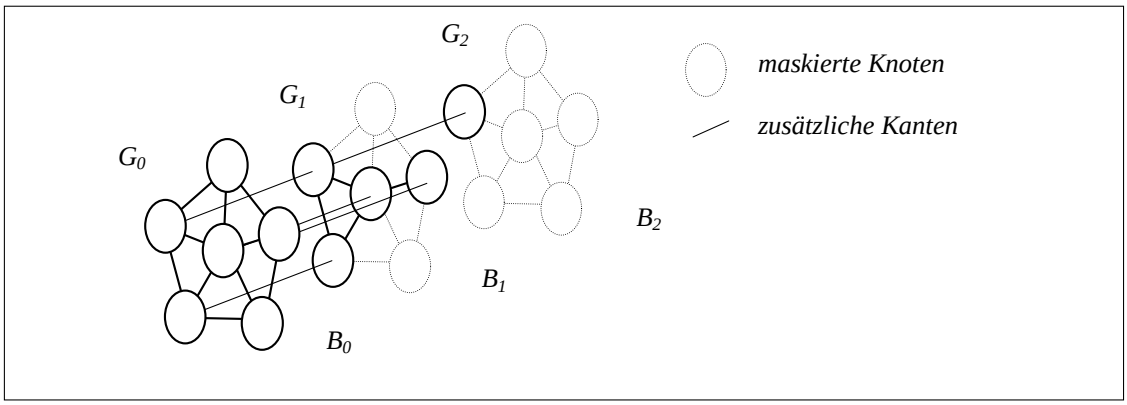


Abbildung 3.27 Darstellung des routenabhängigen Formationsgraphen G_R

Die *Verbindungskanten* e_i erhalten die Teilkosten $e_i=0$, da ein Übergang von Knoten V_n zu V_{n+1} für die Formation keine Änderung und damit weder Zeit, Energie noch Risiko bedeutet. Diese Annahme erklärt sich auch dadurch, dass es keinen Übergangsbereich

zwischen diskreten Korridorbreiten gibt. Ein Korridor B_n kann nur in einer der dort nicht maskierten Formationen befahren werden, nötige Formationswechsel müssen schon im Korridor B_{n-1} erfolgen.

Damit existiert eine Beschreibung aller Formationsübergänge unter Berücksichtigung der verfügbaren Korridorbreite. Die Suche nach dem optimalen Formationswechsel entlang der zuvor berechneten Route reduziert sich somit auf die Suche einer optimalen Verbindung zwischen einem Startknoten S_0 und einem Zielknoten Z_n innerhalb von G_R. Hierzu kann die schon erwähnte Breitensuche (Abbildung 3.27) verwendet werden, aber auch andere Algorithmen, wie Dijkstra's Algorithmus oder A* sind möglich.

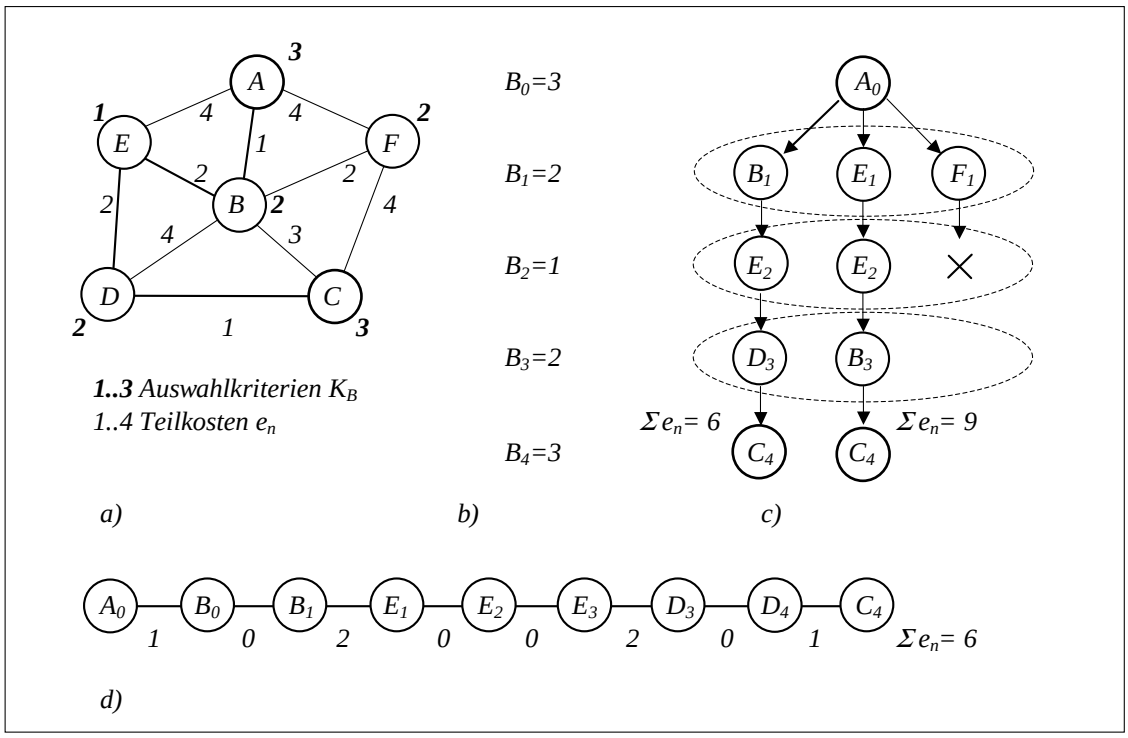


Abbildung 3.28 Breitensuche für optimalen Formationswechsel
a) Formationswechselgraph
b) Korridorbreiten aus Routenanalyse
c) Suchbaum nach Breitensuche (reduziert)
d) Resultierender Formationswechsel

Im Ergebnis liegt ein optimaler Formationswechsel entlang der berechneten Route vor. Damit ist die Menge alle Formationen, die im Verlauf der Mission eingenommen werden, sowie alle Übergänge zwischen diesen Formationen bekannt. Aus diesen beiden Informationen können nun die Steuerverhalten der einzelnen Roboter abgeleitet werden.

3.4.4 Auswahl und Aktivierung der Steuerverhalten

Da die Steuerverhalten der einzelnen Roboter in den Knoten des Formationsgraphen hinterlegt sind, kann schließlich für jeden Roboter dessen optimiertes Verhalten bestimmt werden.

Entsprechend der bereits bekannten Formationswechsel werden zunächst für jeden Roboter die für die Mission tatsächlich benötigten Steuerverhalten in eine Liste vordefinierter Verhalten eingetragen. Für N Roboter entstehen dabei N Listen, die vor Beginn der Mission übertragen und so jeweils dem Navigator der lokalen Ausführung zugänglich gemacht werden (vgl. Abbildung 3.22). Damit sind die Roboter hinsichtlich ihrer Fähigkeiten auf die Ausführung einer Mission spezialisiert. Umgekehrt reduziert sich dadurch das zu übertragende Datenvolumen pro Mission bzw. kann der Formationsgraph bedarfsweise um weitere Formationen und Verhalten ergänzt werden, ohne dass das Änderungen an den Robotern nach sich zieht.

Damit der Navigator das richtige Steuerverhalten auswählen und instanziieren kann, benötigt er eine zusätzliche Beschreibung, wann in welche Formationen zu wechseln ist. Prinzipiell kann dafür auch die zurückgelegte Wegstrecke verwendet werden. Nachfolgend soll jedoch ein Ansatz beschrieben werden, der gegenüber Planungs- und Bewegungsabweichungen toleranter ist. Dabei entscheidet der Navigator auf Grund der vom Beobachter aktuell bereitgestellten Sensorinformationen. Die Formationssequenz wird dazu um Regeln der Form WENN <Bedingung> DANN <Aktion> erweitert. Da im vorliegenden Fall die Formationswechsel streng sequenziell erfolgen, können die Regeln auch als *Transitionen*[6] bezeichnet werden. In Analogie zur Routenplanung sowie der durchgeführten Routenanalyse basieren die Transitionen auf der aktuellen Korridorbreite. Ein Roboter muss also in der Lage sein, den Abstand zu allen für ihn sichtbaren Hindernissen zu bestimmen. Der minimale gemessene Abstand ist dann der Radius des aktuellen Umkreises. Die Transitionen werden als Relation zwischen dem gemessenen Umkreis und einem Grenzwert formuliert.

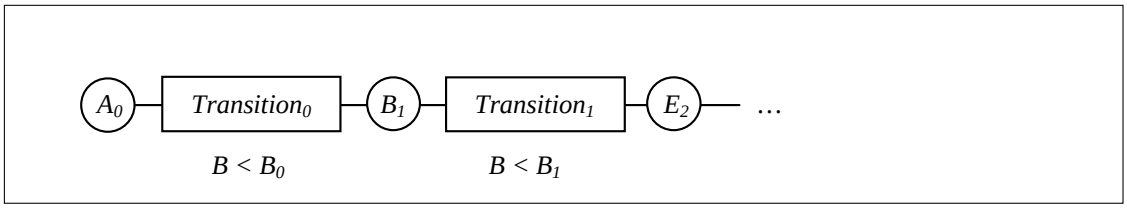


Abbildung 3.29 Erweiterung der Formationssequenz um Transitionen

Durch Vergleich der vom Beobachter gelieferten Entfernungsangaben mit den in den Transitionen angegebenen Grenzwerten ist der Navigator dann in der Lage, eine anstehende Formationsänderung zu erkennen. Neben einer erfüllten Transition, muss aber auch sichergestellt sein, dass die Ausgangsformation momentan besteht. Während der Bewegung in einer Formation sind Positions- und Richtungsabweichungen einzelner Roboter durchaus möglich. Gründe dafür sind Hindernisvermeidung oder Korrekturbewegungen nach einem Richtungswechsel des vorausfahrenden Roboters. Eine Änderung der Steuerverhalten könnte dann zu unerwünschten Bewegungen der Roboter führen.

Die Roboter müssen sich daher zunächst auf einen Formationswechsel verständigen. Dazu richtet der führende Roboter, dessen Transition erfüllt ist, eine Anforderung an alle anderen Roboter. Sofern diese sich an ihren Sollpositionen befinden, bestätigen sie die Anforderung. Erst dann wird der Formationswechsel durchgeführt.

[6] Eine Transition beschreibt den Übergang eines Objektes (Robotersystem) von einem Zustand (Formation) in den nächsten. (UML-Definition)

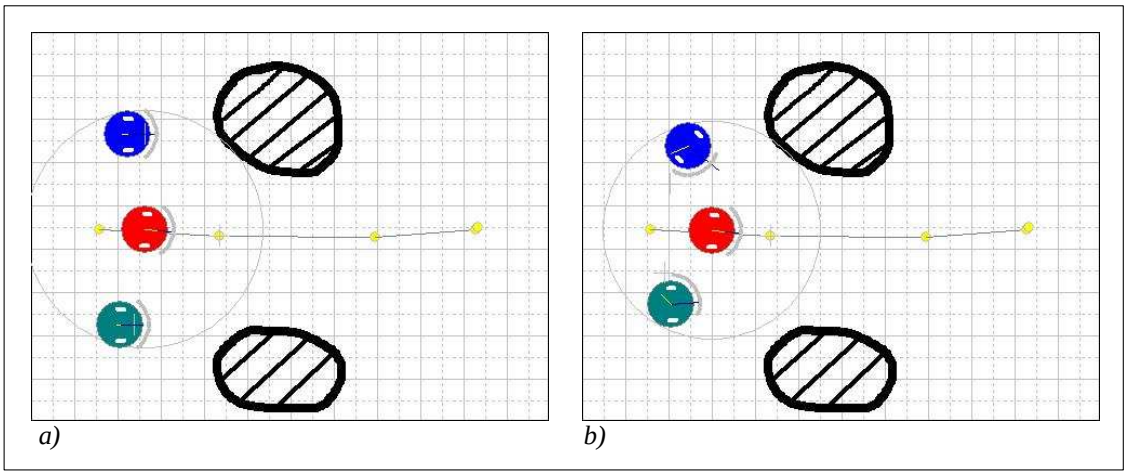


Abbildung 3.30 Wechsel der Formation bei Annäherung an Engstelle
a) Umkreise um R_1 unterschreitet Schwellwert (*Transition*)
b) Beginn des Formationswechsels in Dreiecksformation

Solange nicht alle folgenden Roboter eine Bestätigung ihrer Position gesendet haben, müssen die Roboter, die ihre Positionen bereits erreicht haben, warten. Damit wird sichergestellt, dass die Formation nicht auseinandergezogen wird.

3.4.5 Formulierung eines Abbruchkriteriums

Die korrekte Ausführung einer zuvor geplanten Bewegung beruht auf der sequenziellen Abfolge der Formationen in Bezug zur Umgebung und damit auf der Erkennung der Transitionsbedingungen. Insbesondere durch Abweichungen des vorausfahrenden Roboters von der vorgegebenen Route wäre die sichere Erkennung der erfüllten Transitionsbedingungen fraglich. Ebenso können Veränderungen der Umgebung im Anschluss an eine Planung dazu führen, dass der Missionsplan seine Gültigkeit verliert. Es ist deshalb erforderlich, den Robotern die Möglichkeit zu geben, den Missionplan kontinuierlich mit der aktuellen Situation zu vergleichen. Zusätzlich soll ein Abbruchkriterium dazu führen, dass bei unzulässig großen Abweichungen die Mission vorzeitig beendet und eine Neuplanung initiiert wird.

Die Missionsplanung erfolgte auf der Basis einer erwarteten Korridorbreite B_E. Dabei wurde zunächst der Verlauf von B_E über der Wegstrecke s ermittelt. Dieser Zusammenhang wird dem führenden Roboter nun als zusätzliche Eigenschaft der segmentierten Route zugänglich gemacht. Während der Missionsdurchführung wird ständig der tatsächlich gemessene Verlauf $B_M = f(s)$ in Korrelation zum erwarteten Verlauf gesetzt. Unter der Voraussetzung, dass die Messwertaufnahme in äquidistanten Schritten s_T erfolgt, ergibt sich die Korrelationsfunktion R_m für diskrete Signalverläufe zu:

$$R_m = \frac{1}{n-m} \sum_{i=0}^{n-m} B_E(i) * B_M(i-m). \qquad (3.6)$$

Dabei ist m ein Maß für die Verschiebung der beiden Messreihen um die Weglänge $s_m = m*s_T$. Bei idealer Korrelation der erwarteten mit der gemessenen Korridorbreite hat die Korrelationsfunktion R_m ihr Maximum bei $s_m = m = 0$. Kommt es in Folge einer

Bahnabweichung oder durch ein unbekanntes Hindernis zu einer Änderung des Verlaufs der gemessenen Korridorbreite B_M, verschiebt sich der Scheitelpunkt der Korrelationsfunktion. Durch Wahl eines geeigneten Toleranzbereichs für s_m bzw. m kann lokal die Ausführung der Mission überprüft werden.

Die Korrelation, hier Kreuzkorrelation, ist ein statistisches Verfahren und erfordert daher eine Mindestmenge an diskreten Messpunkten für eine sichere Aussage. Der erwartete Korridorverlauf kann hinreichend genau übertragen oder zumindest interpoliert werden. Die Messwertaufnahme im Roboter beginnt dagegen erst mit dem Start der Mission und benötigt eine Anlaufstrecke. Unter der Annahme, dass mögliche Abweichungen des führenden Roboters von seiner berechneten Bahn nicht schon zu Beginn vorliegen, ist eine Anlaufstrecke unkritisch. Problematisch sind jedoch Hindernisse, die innerhalb der Anlaufstrecke auftreten, da diese dann noch nicht zu einem Abbruch führen.

3.5 Möglichkeiten und Grenzen

Die in den vorangegangenen Abschnitten vorgestellte Umsetzung der Planungsstrategie umfasst alle Schritte von der Modellbildung bis hin zur lokalen Ausführung mit Abbruchkriterium. Für jedes Teilproblem wurde eine Lösungsvariante vorgestellt. Abweichend davon sind auch andere Lösungen, z.B. für die Routenplanung oder die sich anschließende Analyse, denkbar. Insbesondere dabei besteht die Möglichkeit, die einfache sequenzielle Formationsfolge durch eine strukturierte zu ersetzen, die ggf. ein oder mehrere Alternativen enthält. Die Umschaltung erfolgt ebenfalls auf Grund lokaler Gegebenheiten. Erst wenn keine Alternativlösung zum Erfolg führt, wird ein Abbruch initiiert. Ohne die Planungsstrategie also prizipiell ändern zu müssen, sind eine Reihe Erweiterungen denkbar, die auch für komplexere Problemstellungen geeignete Lösungen finden. Nachfolgend soll eine Variante der Planungsstrategie erläutert werden, die die Missions- und die Verhaltensplanung stärker integriert.

3.5.1 Erweiterungen und Optimierungen

In Bezug auf Abbildung 3.10 stellt sich die Frage, ob eine kurze Route mit Formationswechsel im Ergebnis nicht doch günstiger ist, als eine lange Route ohne Formationswechsel? Um darauf eine Antwort geben zu können, ist es erforderlich, die Einzelkosten von Missions- und Verhaltensplanung für jede mögliche Route zu summieren, um schließlich die optimale Lösung auswählen zu können. Die sequenzielle Berechnung aller möglichen Lösungsvarianten ist aber mit einem recht hohen Zeitaufwand verbunden.

Bei Verwendung einer graphenbasierten Routenplanung besteht eine alternative Lösung darin, eine Schätzfunktion $h(v)$ einzubeziehen, die in Abhängigkeit des verfügbaren Korridors den entsprechenden Kanten v_i eines Routengraphen zusätzliche Kosten verleiht. Danach würden Streckenabschnitte geringerer Breite bei der Routensuche zunächst benachteiligt. Wenn aber alle anderen Routen auf Grund ihrer Länge zu höheren Kosten führen, wird schließlich die kürzere Strecke mit geringerer Breite akzeptiert.

Die Grundlage für diese Erweiterung der Missionsplanung bildet ein Routengraph, dessen Kanten neben der Länge auch Informationen über die verfügbare Korridorbreite besitzen. Eine Möglichkeit, einen derartigen Graphen für eine unstrukturierte Umgebung (z.B. Kamerabild) zu gewinnen, besteht in der Bildung eines Voronoi-Diagramms. In Anlehnung an den Algorithmus zur Erweiterung der Hindernisse aus Abschnitt 3.3.3, kann ein Voronoi-Diagramm iterativ gewonnen werden, indem alle Hindernisse solange erweitert werden, bis die Frontlinien aufeinandertreffen. Die dabei gefundenen Punkte haben zu den Hindernissen maximale Abstände, deren Werte sich aus dem jeweiligen Iterationsschritt ergeben.

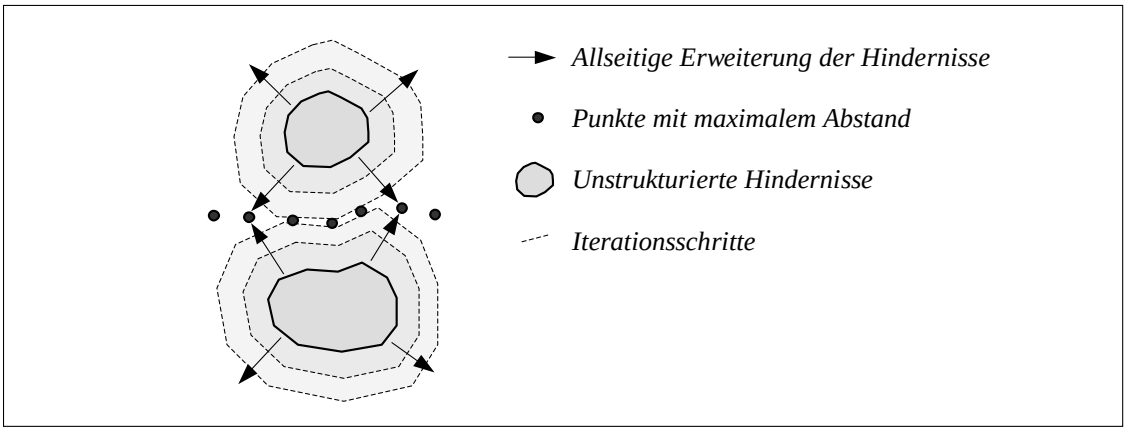


Abbildung 3.31 Bestimmung eines Voronoi-Diagramms mit Zusatzinformationen

Im Anschluss an die Iteration werden benachbarte Punkte so miteinander verbunden, dass sich der gesuchte Routengraph bildet. Vorteil dieses Verfahrens ist neben seiner einfachen Implementierung sowie der Verwendbarkeit für unstrukturierte Umgebungen insbesondere die implizite Bestimmung der Korridorbreite. Die Länge der einzelnen Routensegmente bzw. Kanten ergibt sich im Zuge der Erstellung des Graphen.

3.5.2 Grenzen der Planungsstrategie

Die hier vorgestellte Planungsstrategie betrachtet eine Gruppe von Robotern in der Bewegung als Formation. Dabei wird auf die Beschreibung einer Formation eingegangen, auf die Steuerverhalten, die diese aufrechterhält, sowie auf die Routenplanung und die dazu nötigen Formationswechsel.

Zwei wesentliche Problemstellungen wurden im Verlauf der Erläuterungen jedoch nicht näher betrachtet. Zum einen wurde stets davon ausgegangen, dass sich die Roboter bereits in einer Formation befinden, von der aus weitere Formationen eingenommen bzw. geplant werden können. Ein „ungeordneter“ Grundzustand bedarf aber weiterer Überlegungen, um zunächst eine bekannte, d.h. im Formationsgraph eingetragene, Ausgangsformation einzunehmen. Ein möglicher Lösungsansatz beruht darauf, den Grundzustand mit den bekannten Formationen zu vergleichen und zunächst die Steuerverhalten zu aktivieren, die zum Einnehmen der naheliegendsten Formation führen. Eine Implementierung, bei der eine wahlfreie Positionierung der Roboter allein durch die Aktivierung einiger Grundverhalten, wie „Fahre zum Zielpunkt“ und „Vermeide Kollisionen“ erreicht wird, war noch zu unsicher. Die Roboter drängen sich gegenseitig ab und benötigen dadurch einen großen Freiraum und lange Wege.

Probleme können auch entstehen, wenn die bekannten Formationen nicht zur Lösung der Bewegungsaufgabe ausreichen oder die Lösung dadurch nur unzureichend ist. Eine automatische Erstellung oder Erweiterung des Formationsgraphen wurde zunächst nicht vorgesehen. In dem Fall ist das Ergänzen des Formationsgraphen durch den Anwender nötig. Insbesondere Spezialfälle, wie enge Kurvenradien oder einseitige Hindernisse sollten im Verlauf der Routenanalyse erkannt und z.B. in Form parametrierter Verhaltensobjekte an die Roboter weitergegeben werden. Eine potentielle Lösung wäre die Verbindung von Merkmalen der Route mit Parametern der Verhaltensobjekte. Ein Kurvenradius könnte demnach dazu führen, dass die Formation durch Anpassung der Steuervektoren geeignet „verformt“ wird, während einseitige Hindernisse als kurzfristige Kursabweichung an den führenden Roboter weitergegeben werden.

4 Experimentalplattform

4.1 Verwendete Kleinstroboter

Die für die praktische Erprobung verwendeten Kleinstroboter entstanden parallel zur vorliegenden Arbeit. Die Grundlage bildeten vorangegangene Entwicklungen deren Hardwarekonzepte sowie deren Programmierung in [Altenburg, 2002] umfassend beschrieben sind.

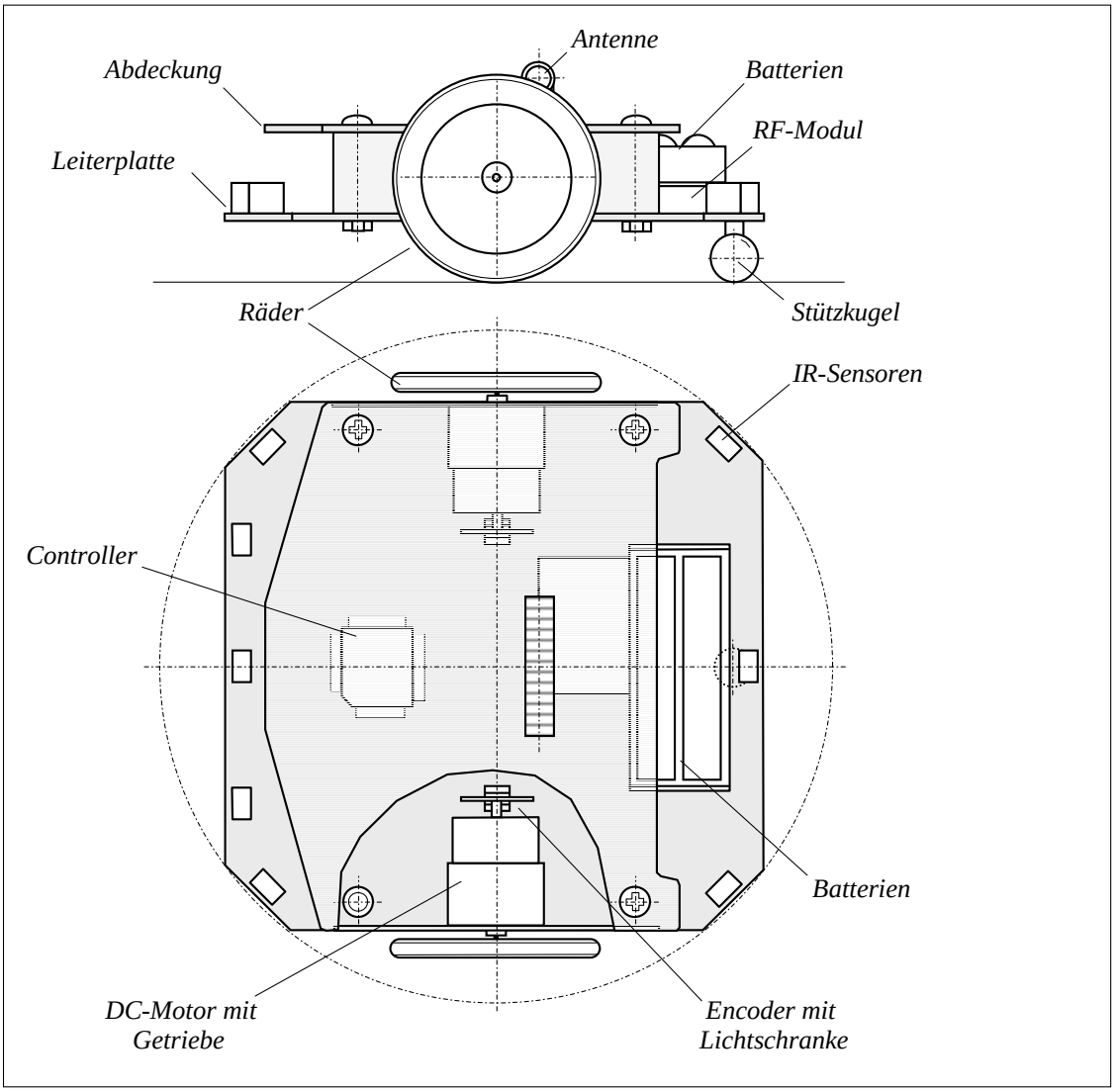


Abbildung 4.1 Kleinstroboter

Verbesserungen ergeben sich im Wesentlichen aus der größeren Anzahl IR-Sensoren, der feineren Auflösung der Radencoder sowie der Verwendung eines leistungsfähigeren Mikrocontrollers.

Prinzipiell handelt es sich um *non-holonome* Roboter mit Differentialantrieb (*differential drive*). Auf Grund der zentralen Lage des Antriebs sowie der Approximation der Form durch einen Zylinder ergeben sich aber kaum kinematische Einschränkungen.

Technische Daten:

Abmessungen (LxBxH):	100 x 105 x 50 mm³ (∅ 120mm)
Antrieb:	
Art:	differential drive
Radstand:	97 mm
Raddurchmesser:	42,5 mm
Inkremente/Umdrehung:	278
Getriebeübersetzung:	139 : 1
Motordrehzahl (max.):	2500 1/min
Sensoren:	
IR-Sensoren:	analog reflex (5 vorn, 3 hinten)
Batteriespannung:	analog
CCD-Kamera:	optional
Mikrocontroller:	Mitsubishi M16 (16bit)
FLASH:	256 kB
RAM:	20 kB
Takt:	16 MHz
Telemetrie:	
Frequenz:	433,92 MHz
Modulation:	FSK
Datenrate (max.):	64 kBit/s halbduplex
Spannungsversorgung:	
Batterien/Akkus:	3 V (2x AA)
Stomaufnahme:	
Steuerelektronik:	32 mA
Motor:	45 mA
RF-Modul:	21 mA
Sonstiges:	
Tonausgabe:	Piezo-Lautsprecher
Debug-Schnittstelle:	Seriell RS232

4.2 Positions- und Lagebestimmung mittels Bilderkennung

Grundsätzlich wird die aktuelle Position jedes Roboters durch einen lokalen Beobachter mitgeführt, der seine Eingangsinformationen aus den Impulsen der Rad-Encoder bezieht. Diese inkrementelle Messung führt aber insbesondere bei häufigem

Wechsel der Drehrichtung der Räder zu einem sich akkumulierenden Digitalisierungsfehler. Daher muss die absolute Position zu Beginn und in Intervallen, die eine hinreichend geringe Abweichung des lokalen Beobachters sichern, von außen vorgegeben werden. Da für eine koordinierte Steuerung auch die Positionen der anderen Roboter von Interesse sind, bietet sich hier die externe bildgestützte Positionsbestimmung mit zyklischer Übertragung an die Roboter an.

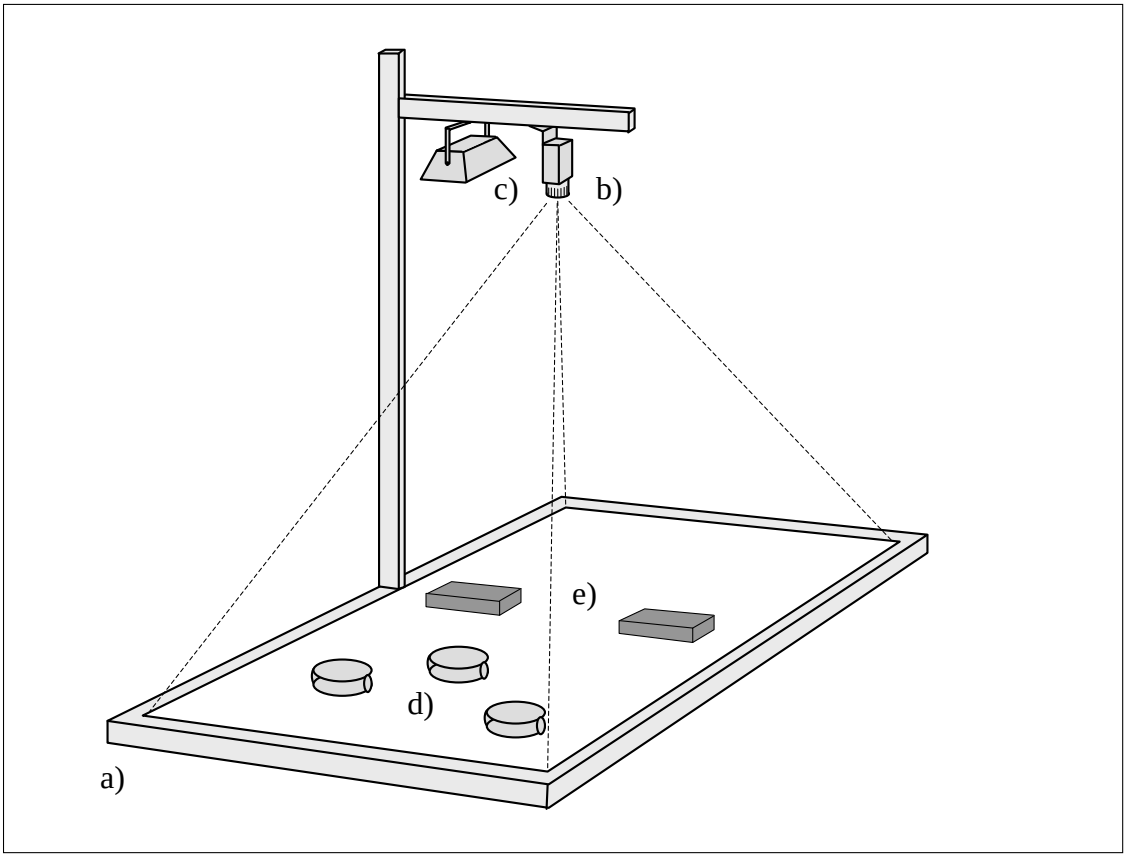


Abbildung 4.2 Anordnung zur Positionsbestimmung
a) Plattform: 1200 x 900 mm² (4:3)
b) Kamera: CCD 752 x 582 (4:3)
c) Beleuchtung: 150W
d) Roboter
e) Hindernisse

Nachdem eine Anfangskonfiguration $K_0 = [x_0,y_0,\phi_0]^T$ übermittelt wurde, kann lokal im Roboter zu jedem Zeitpunkt t dessen aktuelle Konfiguration $K(t) = [x(t),y(t),\phi(t)]^T$ bestimmt werden. Für die Bilderkennung bedeutet das, dass das Gewicht nicht auf der Erkennungsgeschwindigkeit, sondern vielmehr auf der -genauigkeit liegen muss.

Prinzipiell sind zwei Arten der Bilderkennung für vergleichbare Anwendungen üblich. Diese lassen sich in farb- und kantenbasierte Verfahren unterteilen.
Die farbbasierten Verfahren bilden Gruppen von Bildpunkten (auch Farbwolken oder –haufen genannt), die eine gemeinsame Farbe aufweisen und berechnen darin den Schwerpunkt. Für eine eindeutige Positionsbestimmung müssen an einem Roboter dazu mindestens zwei unterschiedliche Farbmarkierungen angebracht werden. Aus den

beiden Schwerpunkten kann dann die Position und Orientierung des Roboters berechnet werden.
Die konturbasierten Verfahren nutzen primär nur die Kanten, die sich durch Helligkeits- bzw. Farbwechsel in einem Bild abzeichnen. Die Kanten werden geometrischen Objekten zugeordnet, die die Bildinformationen zunächst abstrahieren. Die eigentliche Erkennung erfolgt dann anhand der Bildobjekte und ist damit unempfindlicher gegenüber statistischen Schwankungen, wie sie bei der Farbzuordnung auftreten.
Beide Verfahren haben ihre Berechtigung. In Hinblick auf eine möglichst exakte Positionsbestimmung der Roboter, die wiederum Eingangsgröße der Routenplanung ist, wurde das nachfolgend beschriebene konturbasierte Verfahren entworfen und realisiert.

4.2.1 Markierung der Roboter

Die Markierung der Roboter erfolgt durch geometrische Formen für Position und Richtung sowie zusätzlich durch Farbcodes, um eine eindeutige Zuordnung der Roboter treffen zu können. Dabei besitzt jeder Roboter zwei unterschiedlich große Kreisflächen, die sich auf einem schwarzen Untergrund befinden und von denen die größere farbig ausgeführt ist. Da die Grundfläche des Experimentierfeldes nahezu weiß ist, entsteht eine kontrastreiche Markierung.

Die Kreisflächen kodieren die Information über Position und Orientierung dadurch, dass sich auf der gedachten Verbindungslinie der Kreismittelpunkte der Schwerpunkt des Roboters befindet, während die Orientierung sich aus dem Richtungsvektor vom größeren zum kleineren Kreis ergibt. Um beide Kreisflächen möglichst groß ausführen zu können, wurde keiner direkt in den Schwerpunkt des Roboters gelegt. Weiterhin gilt, dass der kleinere Kreis ca. 2/3 der Fläche des größeren Kreises hat, um beiden eine in etwa gleiche Helligkeit zu verleihen.

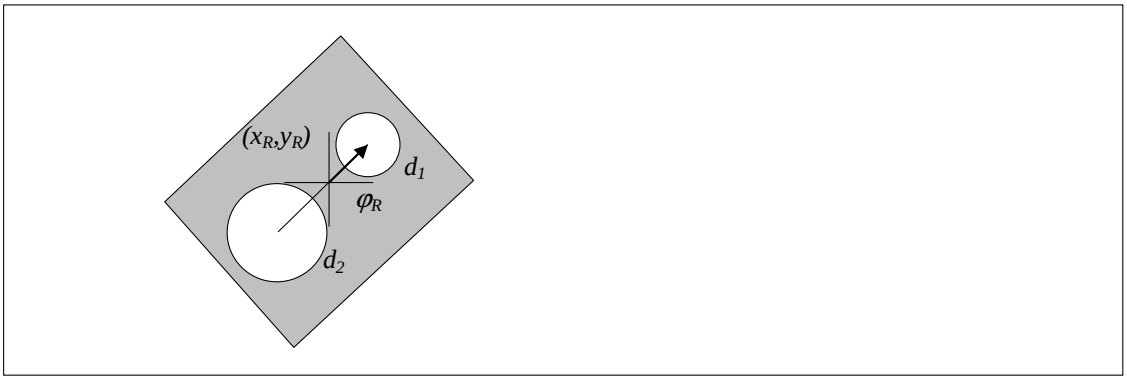


Abbildung 4.3 Markierung eines Roboters (d_1=20mm, d_2=25mm)

Der Farbwert des größeren Kreises ergibt sich durch Filterung nur eines Grundfarbanteils, um einen hohen Kontrast zum Untergrund zu erreichen. Daher besitzt die Markierung von Roboter 1 keinen Rotanteil, die von Roboter 2 keinen Grünanteil und bei Roboter 3 fehlt der Blauanteil. Auf diese Art und Weise können 3 Roboter markiert werden.

4.2.2 Erkennung der Roboter

Der Erkennungsalgorithmus für die Roboter gliedert sich in 6 Schritte:

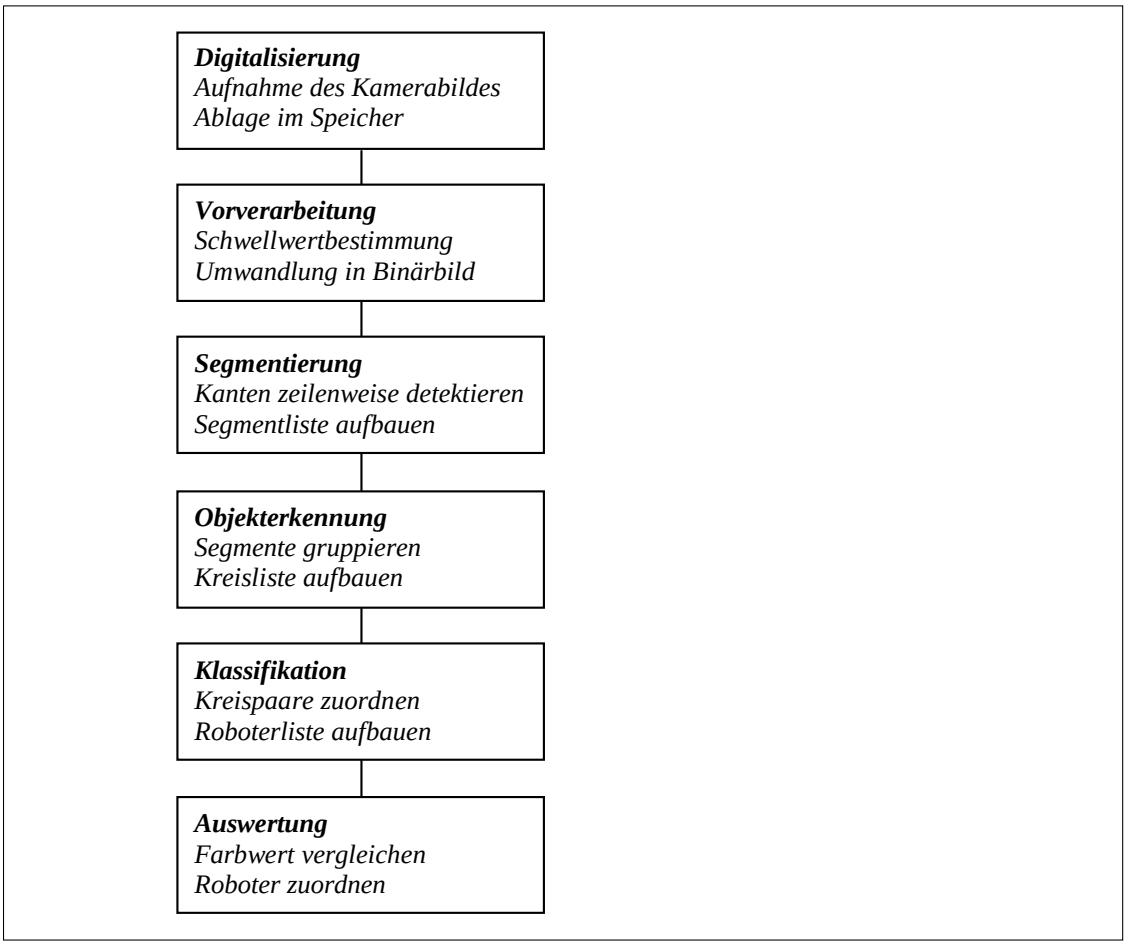


Abbildung 4.4 Schritte der Robotererkennung

Die Digitalisierung des Kamerabildes erfolgt in einer dafür vorgesehenen Framegrabber Karte. Durch Parameter wie Helligkeit, Kontrast und Farbsättigung, kann die Bildaufnahme beeinflusst werden. Die Bilddaten liegen danach zeilenweise im Speicher und können punktweise adressiert werden. Jeder Bildpunkt wird durch seine drei Grundfarben Rot, Grün und Blau (RGB) beschrieben.

In einem ersten Schritt, der *Vorverarbeitung*, wird das Bild in ein Binärbild umgewandelt. Dazu werden die Farbwerte jedes Bildpunktes *P* in einen entsprechenden Grauwert umgerechnet und mit einer Schwelle *T* verglichen. Für *P* gilt demnach:

$$P = \begin{cases} 1, \; falls: 0.3R + 0.59G + 0.11B < T \\ 0, \; sonst \end{cases}. \tag{4.1}$$

Der Bildpunkt *P* wird dann gesetzt (dunkel), wenn dessen Grauwert unterhalb der Schwelle *T* bleibt. Dieser Vergleich wird für jeden Bildpunkt durchgeführt. Die Schwelle *T* wird dabei zunächst vorgegeben, muss aber sinnvollerweise aus einem Histogramm ermittelt werden, in dem alle Graustufen ihrer Häufigkeit nach

eingeordnet werden. Auf diesem Weg können Beleuchtungsschwankungen weitgehend ausgeglichen werden.
Das so gewonnene Binärbild wird nun zeilenweise nach Segmenten durchsucht, die von Dunkel/Hell-Hell/Dunkel-Übergängen begrenzt werden - *Segmentierung*. Unter Einhaltung von Randbedingungen, wie Maximallänge eines Segments, wird diese Suche von einem endlichen Automaten ausgeführt, der dabei eine verkettete Liste aller gefundenen Segmente erstellt.

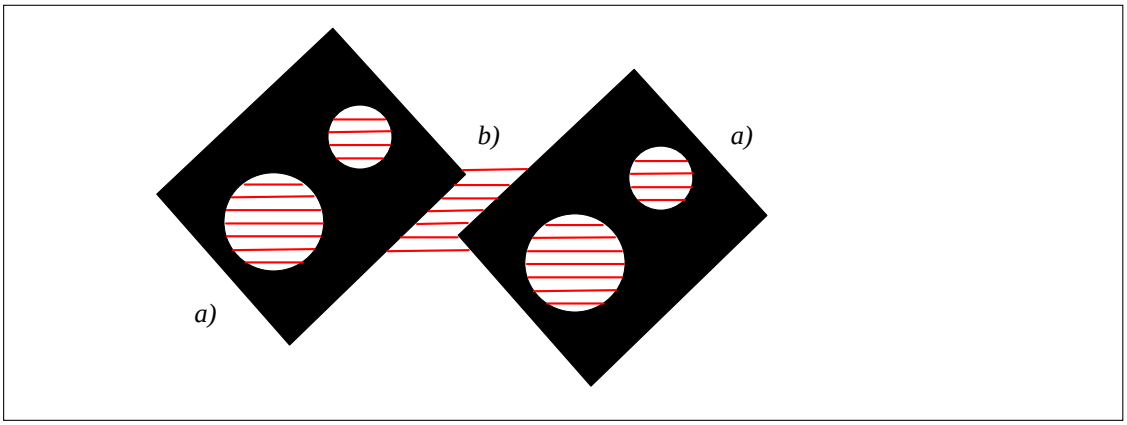


Abbildung 4.5 Segmentierung des Binärbildes
a) erwartete Segmente
b) fehlinterpretierte Segmente

Die Segmentliste $L_S = \{S_0,S_1,...,S_n\}$ enthält neben den erwarteten Segmenten der Kreisflächen a) eine Reihe von Segmenten, die aus den Randbedingungen des Suchalgorithmus' resultieren b) und in einem späteren Prozess eliminiert werden müssen. Die einzelnen Segmente $S = [x_0,x_1,y,c]^T$ sind durch ihren Anfangs- und Endpunkt beschrieben, enthalten jedoch für eine spätere Zuordnung bereits eine Farbinformation.
Unter der Annahme, dass die erwarteten Segmente jeweils Kreisflächen beschreiben, werden die Segmente, deren Mittelpunkte $x_m = (x_1 - x_0) / 2$ innerhalb eines Toleranzbandes $x_m \pm \varepsilon$ übereinander liegen, zu Gruppen zusammengefasst. Für die Segmente einer Gruppe gilt dabei: $y_n - y_{n-1} = 2$. Sofern disese Bedingung nicht eingehalten wird, beginnt eine neue Gruppe. Die Gruppen werden gleichzeitig auf Plausibilität geprüft, d.h. die Anzahl der Segmente innerhalb einer Gruppe muss in einem Erwartungsbereich liegen.
Es hat sich gezeigt, dass durch diese Art der Gruppierung, die in Abbildung 4.4 mit b) bezeichneten Segmente fast vollständig eliminiert werden können. Es konnte aber auch beobachtet werden, dass bei bestimmten Konstellationen der Roboter zueinander gültige Gruppen zwischen den Markierungen entstanden. Besonders die Beleuchtung ist hier kritisch, da die Roboter eine Höhe h_R haben, die bei Verwendung eines Punktstrahlers auf der Grundfläche zu einer Schattenbildung und damit zu unscharfen Konturen der Markierung führt. Die Beleuchtung durch einen angenäherten Lambertschen Strahler wurde auf Grund des konstruktiven Aufwands nicht realisiert.
Für eine eindeutige *Objekterkennung* müssen die gefundenen Segmentgruppen daher noch hinsichtlich ihrer Kontur untersucht werden. Dabei müssen die Endpunkte der Segmente innerhalb eines Toleranzrings *R* liegen. Der Mittelpunkt des Toleranzrings ist $R_m = (x_m, y_m)$ und kann bereits bei der Gruppierung der Segmente durch

Schwerpunktbildung ermittelt werden. Als mittlerer Durchmesser D_m wird das längste Segment der Gruppe angenommen.

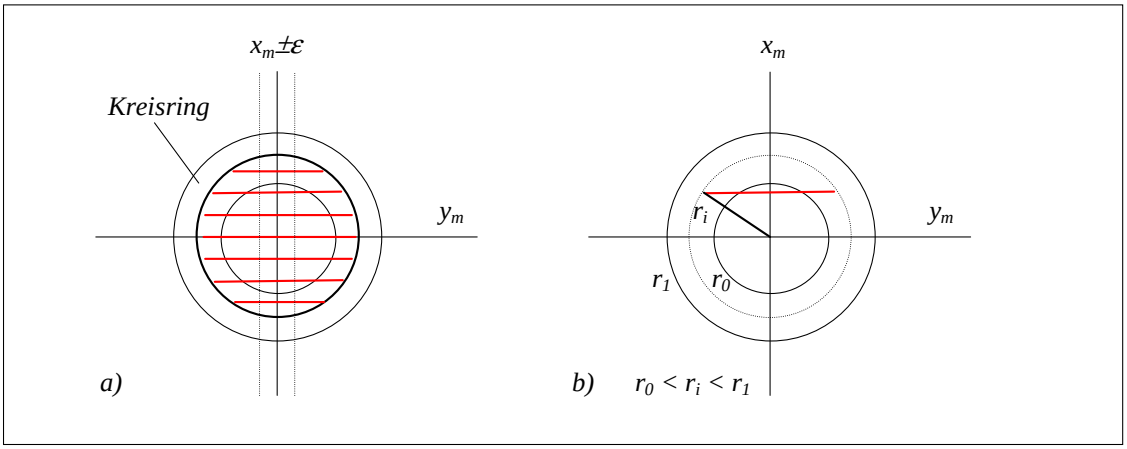


Abbildung 4.6 Kreiserkennung
a) Gruppenkriterium
b) Segmentkriterium

Für jedes Segment S_i innerhalb einer Gruppe wird das Segmentkriterium geprüft. Sofern es von allen Segmenten erfüllt wird, wird der gefundene Kreis in eine Kreisliste $L_K = \{K_0,K_1,...,K_n\}$ aufgenommen. Die Kreise $K = [x_m,y_m,d,c]^T$ sind durch ihren Mittelpunkt, ihren Durchmesser sowie ihren Farbwert beschrieben.
Auf der Basis der Kreisliste kann schließlich eine *Klassifikation* der Roboter durchgeführt werden. Dazu werden diejenigen Kreise zu Paaren zusammengefasst, deren Durchmesser verschieden sind und deren Mittelpunkte einen definierten Abstand haben. Der Abstand L der Kreismittelpunkte unterliegt der Skalierung durch die Bildaufnahme und ist deshalb in Relation zum größeren Kreis definiert.

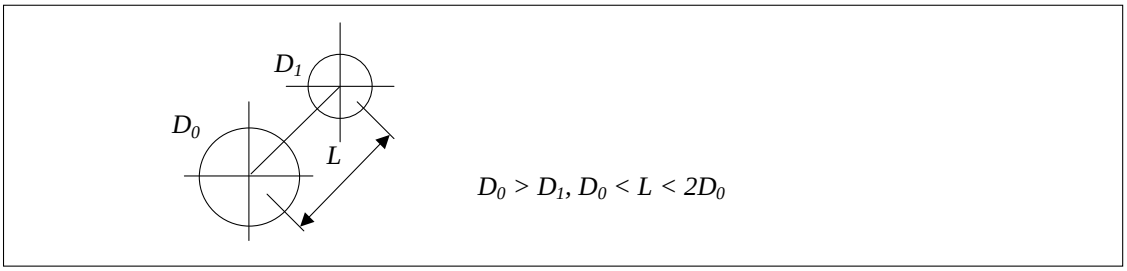


Abbildung 4.7 Roboterklassifikation

Anhand der Infomationen eines solchen Kreispaares kann nun die Konfiguration $K_R = [x_R,y_R,\phi_R]^T$ des Roboters bestimmt werden:

$$\underline{K_R} = \begin{bmatrix} (x_1 - x_0)/2 \\ (y_1 - y_0)/2 \\ \arctan[(y_1 - y_0)/(x_1 - x_0)] \end{bmatrix}. \qquad (4.2)$$

Den letzten Schritt der Robotererkennung bildet die *Auswertung*. Dabei werden die Farbinformationen der Kreise hinsichtlich ihrer Anteile an Rot, Grün und Blau mit den

gesuchten Farbmarkierungen der Roboter verglichen. Die Nummer des Roboters N_R wird dabei nach folgenden Bedingungen gesetzt:

$$N_R = \begin{cases} 1, \; falls: R < G \wedge R < B \\ 2, \; falls: G < R \wedge G < B \\ 3, \; falls: B < R \wedge B < G \\ 0, \; sonst \end{cases}. \tag{4.3}$$

Der Fall, dass die Farbe keinem Roboter zugeordnet werden kann, wird dazu verwendet, um auf eine vollständige Erkennung zu prüfen. Es muss genau einen Roboter mit jeder Nummer geben, daher die Menge der Roboter ist $M_R = \{1,2,3\}$.

4.2.3 Erkennung der Hindernisse

Neben der Position der Roboter ist auch die Lage und Größe möglicher Hindernisse von Bedeutung. Da von einer unstrukturierten Umgebung ausgegangen wird, kommen als Hindernisse beliebig geformte Körper in Betracht. Für den Erkennungsalgorithmus wird daher auf einen Farbvergleich orientiert. Die Hinderniselemente werden so gewählt, dass sie annähernd eine Grundfarbe, z.B. Rot, besitzen.
Damit wird die Erkennung der Roboter nur insoweit beeinflusst, dass alle Hindernisse auch im Binärbild erscheinen, da ihr Grauwert kleiner als die Schwelle T sein wird. Die dadurch entstehenden zusätzlichen Segmente (vgl. Abbildung 4.4 b) stören die Erkennung der Roboter jedoch nicht. Umgekehrt werden die Markierungen der Roboter auch nicht als Hindernisse erkannt, weil hier keine reinen Grundfarben verwendet werden.
Die Hindernisserkennung ist als eigenständiger Algorithmus ausgelegt, da von statischen Hindernissen ausgegangen wird, deren Position und Größe nur im Zusammenhang mit der Routenplanung benötigt werden.

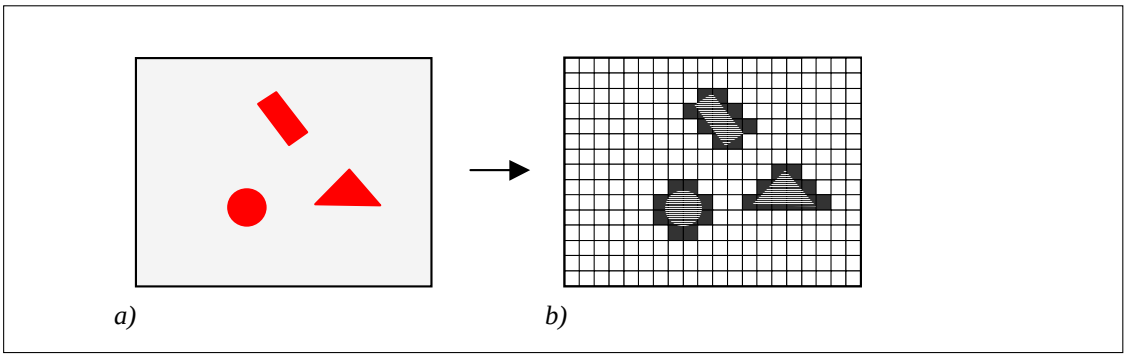


Abbildung 4.8 Diskretisierung der Hindernisse
a) Kamerabild: 752 x 582 (4:3)
b) Hindernisraster: 375 x 280 (4:3)

Das Kamerabild wird mit einem äquidistanten Raster überlagert. Der Suchalgorithmus testet jeden Bildpunkt auf seine Farbe. Hat mindestens ein Bildpunkt in einem Rasterfeld die Farbe des Hindernisses, so wird das Rasterfeld als belegt gekennzeichnet. Auf diesem Weg entsteht eine Hinderniskarte, die für eine Routenplanung geeignet ist.

4.3 Funknetz zur Informationsübertragung

Die Verbindung der Roboter mit der Basisstation erfolgt über ein 433-MHz-Telemetrie-System. Jeder Roboter ist mit einem Sende-/Empfangsmodul ausgestattet, über das ein bidirektionales, paketorientiertes Protokoll im Halbduplex-Verfahren übertragen wird. Alle Roboter nutzen das selbe Frequenzband. Die Funkzelle ist deshalb nach dem Prinzip einer Master/Slave Topologie organisiert. Alle Pakete werden über die Basisstation geleitet, die auch den Netzwerkzugriff steuert. Das Protokoll arbeitet verbindungslos, jede Nachricht (*message*) wird aber durch eine Antwort (*response*) bestätigt. Für die speziellen Anforderungen der Robotersteuerung gibt es neben einer 1:1 (*unicast*) Kommunikation auch eine 1:N (*broadcast*) Kommunikation. Dabei empfangen mehrere Roboter ein Telegramm der Basis und bestätigen es jeweils in einem eigenen Zeitfenster (*timeslice*). Auf diesem Weg können Positionsdaten oder simultan ausführbare Steuerbefehle zeitoptimal bzw. synchron übertragen werden.

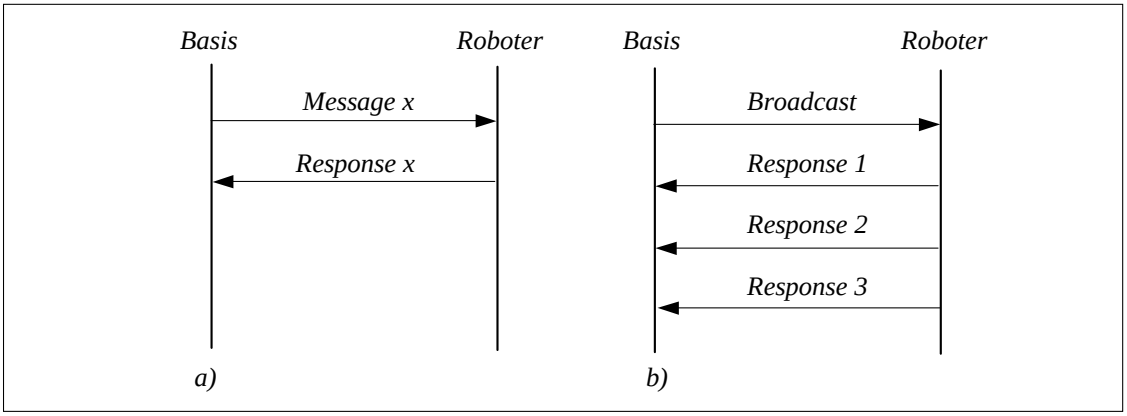


Abbildung 4.9 Kommunikationsarten
a) 1:1 Kommunikation
b) 1:N Kommunikation

Die Nachrichten werden in strukturierten Telegrammen übertragen, die speziell für die verwendete Hardware optimiert sind. Prinzipiell unterscheidet sich der Telegrammaufbau kaum von anderen packetorientierten Netzwerken.

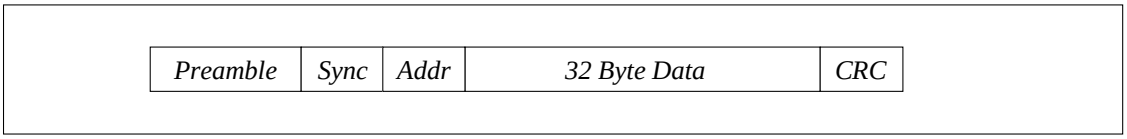


Abbildung 4.10 Telegrammaufbau

Um den Anforderungen batteriebetriebener Kleinstroboter gerecht zu werden, wurden spezielle Eigenschaften implementiert:

- Die Funkmodems der Roboter werden nur alle 50ms aktiviert. Die Präamble ist daher entsprechend lang, ca. 70ms. Der Energiebedarf wird dadurch reduziert.
- Es werden kurze Datensätze mit max. 32 Bytes Nutzdaten übertagen. Dadurch werden kurze Einschaltzeiten erreicht.

- Das Protokoll kann mit einer seriellen Standardschnittstelle (RS232) emuliert werden. Damit entfällt ein zusätzlicher Netzwerkcontroller.
- Das zur Datensicherung verwendete 16Bit CRC-Zeichen[7] wird durch ein Peripheriemodul auf dem Mikrocontroller hardwaremäßig unterstützt.

[7] CRC (Cyclic Redundancy Check) : Verfahren zur Berechnung eines Kontrollwertes, der einem gesendeten Datensatz angefügt wird, um im Empfänger eine Prüfung auf Richtigkeit durchführen zu können. Der CRC-Wert ist der Rest einer Modulodivision des Datensatzes mit einem Generatorpolynom.

5 Softwareumgebung

5.1 Globale Steuerarchitektur

Parallel zum mechanischen Aufbau wurde eine Softwareumgebung geschaffen, die die Grundlage für die Implementierung der vorgestellten Algorithmen bildet. Da im Verlauf einer Entwicklung stets Komponenten erweitert oder gar neu geschaffen werden, war eine offene Strukturierung und die damit verbundene Schaffung von Schnittstellen eines der wesentlichsten Kriterien beim Entwurf. Speziell bei der Bildverarbeitung spielt auch immer die Leistungsfähigkeit eine Rolle. Deshalb wurde auf eine verteilte Lösung mit zwei Rechnern orientiert, die sowohl über die notwendige Rechenleistung als auch über eine schnelle Netzwerkverbindung verfügen. Ein Rechner ist dabei allein für die Bildaufnahme und Auswertung zuständig. Dieser, hier als Image-Server bezeichnete Rechner, hält ständig die aktuellen Positionen der Roboter sowie auf Anforderung auch die Lage der Hindernisse (*Szene*) bereit. Die Bildaufnahme erfolgt über eine CCD-Kamera, die an eine Framegrabber-Karte im Image-Server angeschlossen ist. Der zweite Rechner, auch als Client bezeichnet, ist somit freigestellt von kontinuierlichen rechenintensiven Vorgängen und übernimmt alle Planungs- und Steuerfunktionen.

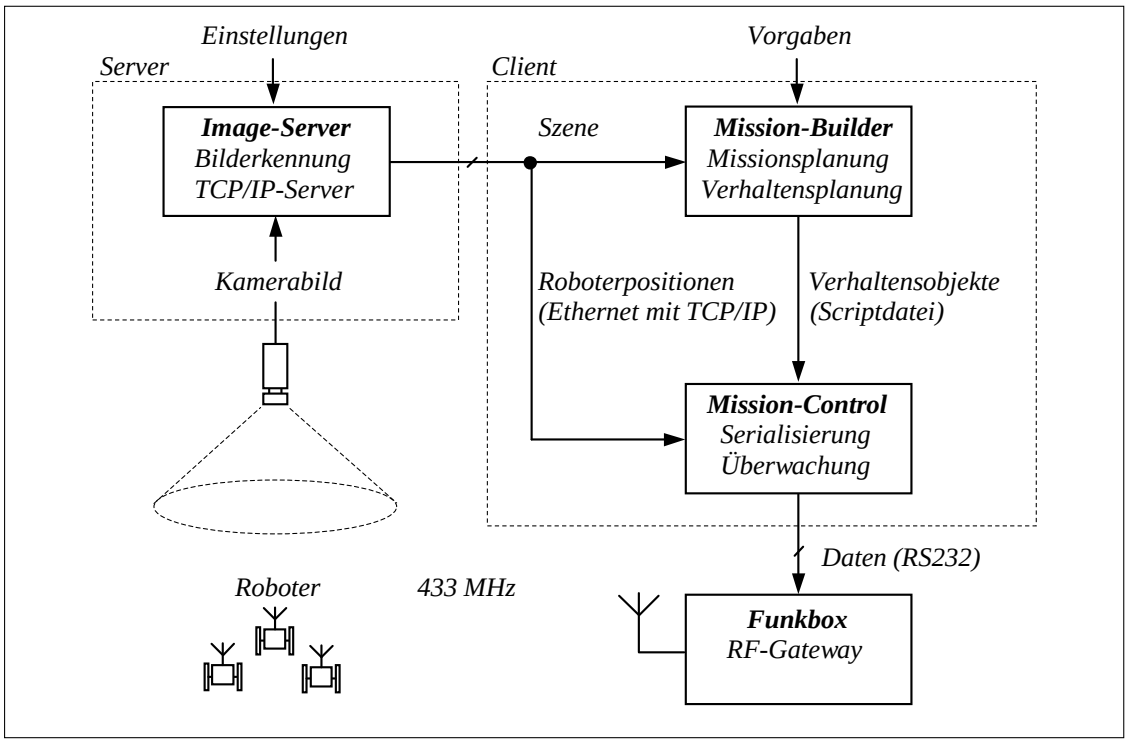


Abbildung 5.1 Globale Steuerarchitektur

Abbildung 5.1 gibt einen Überblick über die Komponenten der globalen Steuerung sowie deren Verbindung und Datenbeziehungen. Der Client-Rechner wird direkt oder

auch über ein lokales Netz mit dem Image-Server verbunden. Als Transportprotokoll wird dabei TCP/IP verwendet, wodurch unter anderem auch die Anbindung mehrerer getrennter Applikationen an den Server ermöglicht wird. Zusätzlich ist an einer der seriellen Schnittstellen (RS232) eine separate *Funkbox*[8] angeschlossen, die eine Protokollumsetzung der Datentelegramme und die eigentliche Funkübertragung zu den Robotern realisiert. Die Ansteuerung der Funkbox erfolgt durch Programmierung der seriellen Schnittstelle[9] des Client-Rechners unter Verwendung eines proprietären Datenprotokolls. Damit ergibt sich ein geschlossener Kreis, der mit der bildgestützten Erkennung der Roboter beginnt und sich mit der Funkübertragung der Steuerbefehle schließt.

Die Aufgabenteilung und -kapselung in entsprechenden Modulen hat sich sehr gut bewährt. Im Verlauf der Entwicklung wurden verschiedene Komponenten geändert. So war etwa die Funkbox in einer ersten Version mit einem eigenen Mikrocontroller ausgestattet, der die Datentelegramme der Roboter empfing und entsprechend der Sync-Sequenz sowie der angefügten CRC-Zeichen filterte und bewertet. In einer zweiten Version konnte diese Aufgabe direkt über die serielle Schnittstelle des Clients emuliert werden. Auch die Bildaufnahme wurde von einer TV-Karte auf eine schnellere Framegrabber-Karte umgestellt. Später wurde dann auch die zunächst verwendete einfache Kamera durch eine hochwertige CCD-Kamera mit maximaler PAL-Auflösung ersetzt. In allen Fällen waren lediglich Änderungen an den unmittelbar betroffenen Modulen nötig.
Eine wesentliche Ergänzung lag in der Einführung eines Simulations-Tools. Mit dem Ziel unterschiedliche Konfigurationen, wie z.B. Positionen der Hindernisse und Roboter oder Softwareänderungen an Planung und Steuerung, schneller testen zu können und gleichzeitig idealisierte Bedingungen zu schaffen, wurde eine Simulationsumgebung für drei Roboter entwickelt. Die wesentlichen Vorteile der Simulation und deren Integration in die hier vorgestellte Steuerungs-Architektur werden später in Abschnitt 5.3 näher erläutert.

Somit umfasst die Softwareumgebung der globalen Steuerungs-Architektur insgesamt vier separate Applikationen/Module:

- eine Bildauswertung mit TCP/IP-Server (*Image-Server*)
- eine Bewegungsplanung (*Mission-Builder*)
- eine Steuerung für 3 Roboter (*Mission-Control*) sowie
- eine Simulation für 3 virtuelle Roboter (*Robot-Lab*).

Diese vier Module sollen nachfolgend genauer betrachtet werden. Die Umsetzung erfolgte zumeist als Native-Code-Applikation für das Betriebsystem Windows®. Speziell für die Simulation wurde jedoch auch MATLAB/Simulink® als Programmier- und Simulationswerkzeug verwendet. Aus diesem Grund musste eine einfache Schnittstelle zur *Virtual Reality Toolbox*[10] geschaffen werden.

[8] Die Funkbox entstand parallel zur Arbeit und besteht aus einem HF-Modul, welches über eine entsprechende Logik an die serielle Schnittstelle des Client-Rechners angeschlossen wird.
[9] Windows® bieten Zugriff auf seriellen Schnittstellen in Form von Lese- und Schreibzugriffen auf virtuelle Dateien (OpenFile, ReadFile, WriteFile, CloseFile).
[10] MATLAB® bietet die Möglichkeit Simulationsergebnisse, z.B. aus Simulink® direkt Virtuellen Objekten zuzuweisen und so zu visualisieren bzw. auch zu animieren. Dieser Teil von MATLAB® wird Virtual Reality Toolbox genannt.

5.2 Software Architektur – Globale Planung

5.2.1 Bilderkennungsmodul: Image-Server

Das Modul Image-Server ist als eigenständige Applikation ausgelegt. Es steuert direkt den Treiber der Framegrabber-Karte an, um so die Möglichkeiten dieser Karte[11] optimal ausnutzen zu können. Dazu werden nach dem Start zunächst Einstellungen vorgenommen, die die Bildaufnahme den gegebenen Lichtverhältnissen anpasst. Zu diesen gehören insbesondere Helligkeit, Kontrast und Farbsättigung. Zur Unterstützung des Einstellvorgangs können unterschiedliche Testroutinen manuell gestartet werden: *Schwarz/Weiss-Bild*, *Binärbild*, *Segmente und Kreise*, *Roboter- und Hindernismarkierung*. Diese Routinen erlauben es, alle Phasen der Bilderkennung zu kontrollieren und die Einstellungen dahingehend zu optimieren. Nach Abschluss der Einrichtung wird dann die eigentliche Server-Applikation gestartet, die kontinuierlich Bilder der Kamera aufnimmt, analysiert und auf Anfrage eines Clients die jeweils letzten Ergebnisse überträgt.

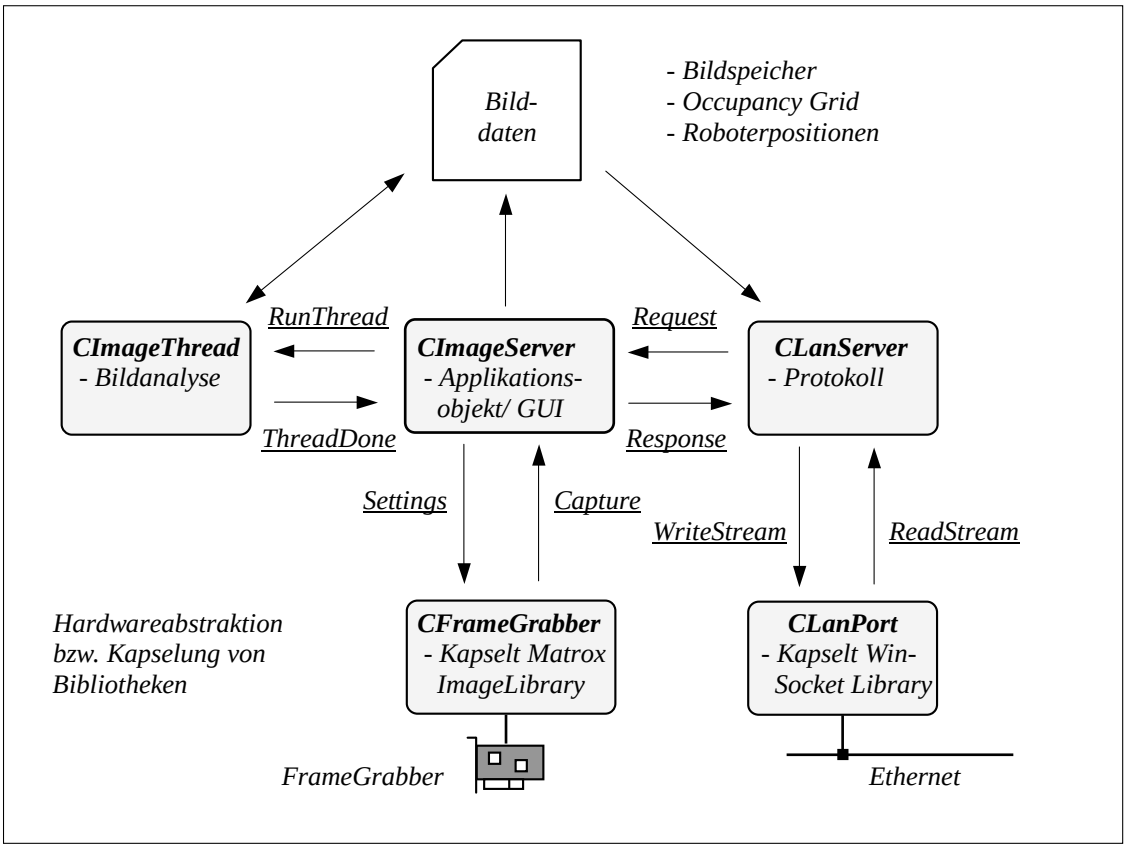


Abbildung 5.2 Objektbeziehungen innerhalb des Moduls Image-Server

Das Modul ist in zwei Bereiche unterteilt – den TCP/IP-Server und die eigentliche Bildauswertung. Letztere läuft in einem eigenen Ausführungskontext (*Thread*), um die Antwort auf Clientanfragen während einer laufenden Auswertung nicht zu verzögern.

[11] Matrox-Framegrabber

Da alle Funktionalität ausschließlich in Objekten (*Instanzen von Klassen*) gekapselt ist, ergibt sich die oben dargestellte Struktur.
Die zentrale Instanz ist die Applikation selbst (*CImageServer*). Sie ist für die Darstellung des Kamerabildes sowie für die Interaktion mit dem Anwender (*Graphical User Interface - GUI*) zuständig und deshalb auch als einzige sichtbar. Parallel dazu existieren die Bildauswertung (*CImageThread*) sowie das eigentliche TCP/IP-Server Objekt (*CLanServer*). Der Server ist als einfacher Kommandointerpreter ausgelegt, der eingehende Anfragen umgehend beantwortet. Das realisierte Protokoll ist proprietär und verwendet ASCII-Kommandoworte zur Anfrage und Binärdaten als Antwort. Die Antwort kann dabei das gesamte Bild, die Position der Roboter oder die Lage der Hindernisse in Form eines Rasterfeldes (*opccupancy grid*) sein. Deshalb ist dem Server ein Objekt (*CLanPort*) unterlagert, das neben dem eigentlichen Zugriff auf das Netzwerk die Sendedaten aufbereitet und umgekehrt die empfangenen Pakete wieder analysiert, um so nur konsistente Daten[12] weiterzuleiten.
Der Ablauf einer Bildauswertung beginnt immer mit der Aktualisierung des Kamerabildes durch Aufruf entsprechender Treiberfunktionen der FrameGrabber-Karte. Danach wird eine Instanz von CImageThread erstellt, die ihrerseits einen eigenen Thread[13] kapselt. Damit läuft die Bildauswertung unabhängig von der Applikation und beeinflusst weder dessen Reaktion auf Bedienereingaben noch die auf Anforderungen an den TCP/IP-Server. Sobald die Bildauswertung abgeschlossen ist, wird die Applikation benachrichtigt. Diese kopiert dann die gewonnenen Daten in einen globalen Datenbereich und beendet die Bildauswertung, indem sie die Instanz von CImageThread wieder entfernt. Dieser Ablauf wiederholt sich zyklisch solange der Server aktiviert bleibt. Für die eingangs erwähnten manuell zu startenden Testroutinen wird der beschriebene Ablauf immer nur genau einmal durchlaufen. Außerdem werden der Bildauswertung bei der Instanziierung dann verschiedene optionale Parameter übertragen. Während im Normalfall nur die lokalisierten Roboter markiert werden, können so auch einzelne Erkennungsschritte, wie gefundene Segmente und Kreise, visualisiert werden.

Im Verlauf der Entwicklung der Bilderkennung hat sich gezeigt, dass es prinzipiell sehr schwer ist, eine Bildauswertung mit den gängigen Mitteln eines Debuggers[14] hinsichtlich ihrer Funktion zu überprüfen. Es kann sichergestellt werden, dass ein Algorithmus korrekt abläuft, aber es ist auf Grund des Datenvolumens ohne weitere Hilfsmittel nicht möglich zu bestimmen, ob Freiheitsgrade innerhalb des Algorithmus' (Kriterien, Grenzen, Abbruchbedingungen) richtig gewählt wurden. Neben statistischen Betrachtungen scheint das Mittel der Wahl eine Visualisierung von Teilfunktionen eines Algorithmus' zu sein. Aus diesem Grund entstanden auch die Testroutinen zuerst und wurden im Verlauf der Entwicklung dann zur eigentlichen Bilderkennung zusammengesetzt.

[12] Die Datensicherung und der Transport der Pakete erfolgt durch das TCP-Protokoll, die Prüfung auf Vollständigkeit der empfangenen Daten liegt jedoch bei der Anwendung (Anwendungsschicht; vgl. [Washburn, Evans, 1997]).
[13] Das Betriebsystem Windows® unterscheidet zwischen Prozessen und Threads. Ein Prozess definiert sich durch Ressourcen (Anwendung) sowie einen Basis-Thread (Ausführungskontext). Weitere Threads können innerhalb eines Prozesses dynamisch erzeugt werden (vgl. [Rößmann, 1997]).
[14] Debugger erlauben i.A. das Definieren von Haltepunkten auf Adressen und/oder Daten sowie die schrittweise Abarbeitung einzelner Threads mit Anzeige von lokalen und globalen Daten.

5.2.2 Planungsmodul: Mission-Builder

Im Modul Mission-Builder wird für die eigentliche Routenplanung neben den Ausgangskonfigurationen der Roboter im Wesentlichen die Lage der Hindernisse benötigt. Eine entsprechende Beschreibung, in Form eines Occupancy Grids, ist zusammen mit den Konfigurationen der Roboter in einer gemeinsamen Datenstruktur (*TScene*[15]) auf dem Image-Server abgelegt. Auf Anfragen mit dem Befehlswort GET_SCENE antwortet der Server mit der Übertragung der Datenstruktur an den Client.

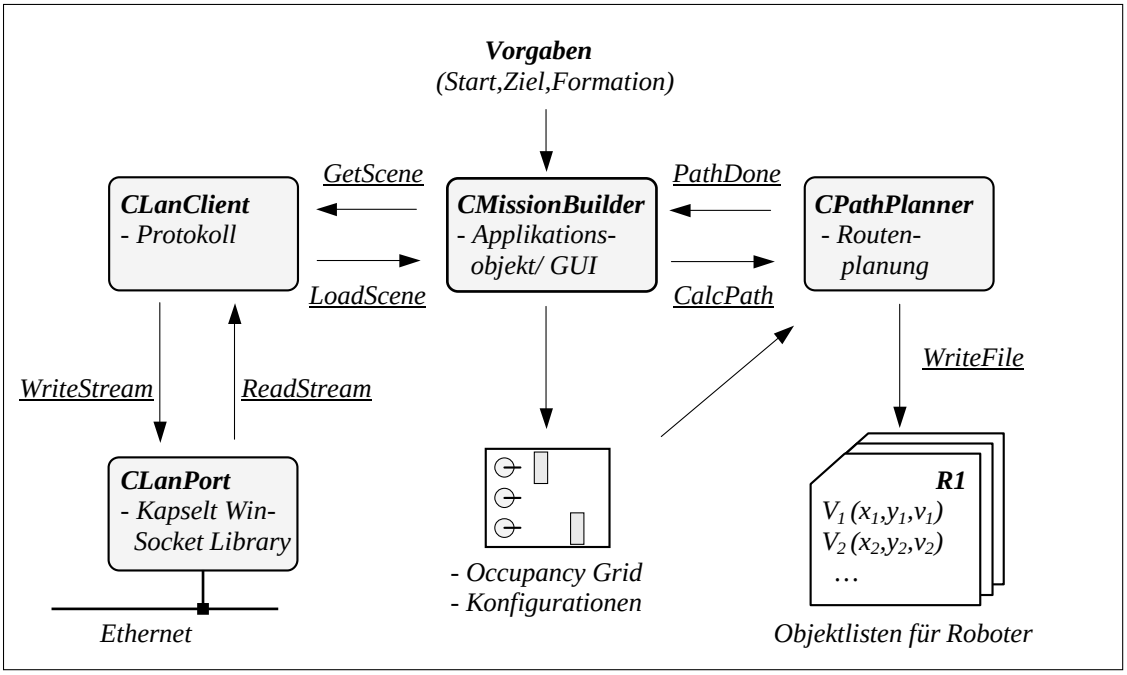


Abbildung 5.3 Objektbeziehungen innerhalb des Moduls Mission-Builder

Die Szene wird zunächst graphisch dargestellt, um Eingaben bezüglich einer Zielposition und -formation zu ermöglichen. Damit besitzt der Routenplanner dann alle Informationen, um eine Route für die gesamte Formation bestimmen zu können. Die Ausgabe der Route erfolgt in Form einer Datei für jeden Roboter $R_1..R_n$. Eine solche Datei enthält dann eine Liste mit Verhaltensobjekten. Die Speicherung als Datei erlaubt die einfache Übergabe an das Modul Mission-Control. Als Format wurde eine strukturierte Textform gewählt, um sowohl eine einfache Kontrolle durchführen zu können, als auch die Möglichkeit einer manuellen Änderung zu haben.

Auf Grund der Tatsache, dass die verwendeten Roboter (vgl. Abschnitt 4.1) nur über eine geringe Speicherkapazität im Bereich weniger Kilobyte (*kB*) verfügen, war die Übertragung der gesamten Liste nicht generell möglich. Außerdem musste auch eine geeignete Beschreibungsform für die Übertragung der Verhalten gefunden werden. Ein Verhalten in Form von Binärcode zu übermitteln, hätte eine Umprogrammierung des

[15] Selbstdefinierte Typbezeichner (*Types*) werden durch Prefix *T* eingeleitet, während Bezeichner für Klassen (*Classes*) entsprechend mit *C* beginnen.

Controllers bei laufendem Programm[16] zur Folge gehabt. Deshalb wurden für die Umsetzung schließlich einige Vereinfachung angenommen:

- Alle planbaren Verhalten sind im Roboter bereits in Form von Funktionen hinterlegt – es erfolgt keine Übertragung neuer Funktionalität
- Ein Verhalten reduziert sich daher auf einen Ordinalwert – dieser stellt einen Index in einer Liste von Verhaltensfunktionen im Roboter dar
- Es wird immer nur ein Verhalten aktiviert, welches sich selbst beenden kann oder durch ein neues abgelöst wird
- Verhalten können durch optionale Parameter spezifiziert werden – mindestens wird aber das Routensegment angegeben, für das das Verhalten gilt.

Damit ergibt sich für die Beschreibung des 1. Verhaltens folgender Eintrag in der Datei:

[Object 0]
Direction = <aktueller Winkel>
Speed = <Sollgeschwindigkeit>
XPosit = <X-Koordinate der aktuellen Position/ Segmentanfang>
YPosit = <Y-Koordinate der aktuellen Position/ Segmentanfang>
XFinish = <X-Koordinate der Zielposition/ Segmentende>
YFinish = <Y-Koordinate der Zielposition/ Segmentende>
ActiveTask = <Verhaltensfunktion>
ParamField0 = <1. optionaler Parameter>
ParamField1 = <2. optionaler Parameter>
ParamField2 = <3. optionaler Parameter>.

Für die in spitzen Klammern angegebenen Werte werden nur Integer-Zahlen verwendet. Damit kann eine solche Struktur sehr effektiv in einen Datensatz serialisiert und über eine Funkschnittstelle übertragen werden. Die Beschreibung weist eine zusätzliche Besonderheit dadurch auf, dass Segmentanfang und -ende immer explizit angegeben werden. Da jedes Segmente S_n immer an das S_{n-1} angrenzt, ist diese Beschreibung eigentlich redundant. Der Roboter führt lokal auch immer die zurückgelegte Wegstrecke mit, sodass er stets seine Position und Ausrichtung kennt. Mindestens einmal, zu Beginn der Route, muss dem Roboter jedoch seine absolute Position von außen übermittelt werden. Da die Auswertung der incrementellen Radencoder zu einer akkumulierenden Abweichung führt, ist es evtl. auch im Verlauf der Fahrt erforderlich, eine Korrektur des lokalen Beobachters vorzunehmen. Um diese beiden Fälle zu verallgemeinern, wurde die aktuelle Position in jeder Übertragung vorgesehen. Vom Planungsmodul wird nur das jeweils erste Objekt einer jeden Verhaltensliste vollständig ausgefüllt. In alle weiteren Objekten werden die Segmentanfänge durch einen Platzhalter (hier Null) markiert und erst unmittelbar vor der Übertragung durch das Modul Mission Control ergänzt.

[16] Controller unterscheiden meist in FLASH- und RAM-Speicher. Daten liegen im RAM, während ausführbare Programme im FLASH abgelegt werden. Während der Programmierung eines FLASHs kann daraus nicht gelesen werden. Es wären zumindest zwei separate FLASH-Bereiche erforderlich.

5.2.3 Steuermodul: Mission-Control

Die letzte Komponente der globalen Steuerarchitektur bildet das Modul Mission-Control. Die wesentliche Aufgabe dieses Moduls besteht zunächst in der Serialisierung der Objektlisten und deren sequenziellen Übertragung an die einzelnen Roboter. Dazu besitzt das Modul Zugriff auf die an der seriellen Schnittstelle angeschlossene Funkbox (*CComPort*). Ein ganz wesentlicher Punkt bildet aber auch die Überwachung und damit verbunden ein Korrektur möglicher Abweichungen von den zuvor geplanten Roboterbahnen. Dazu besitzt das Modul neben der eigentlichen Funkverbindung zu den Robotern auch eine Netzwerkverbindung (*CLanPort*) zum Image-Server, über die es stets die aktuellen Positionen der Roboter abfragen kann.

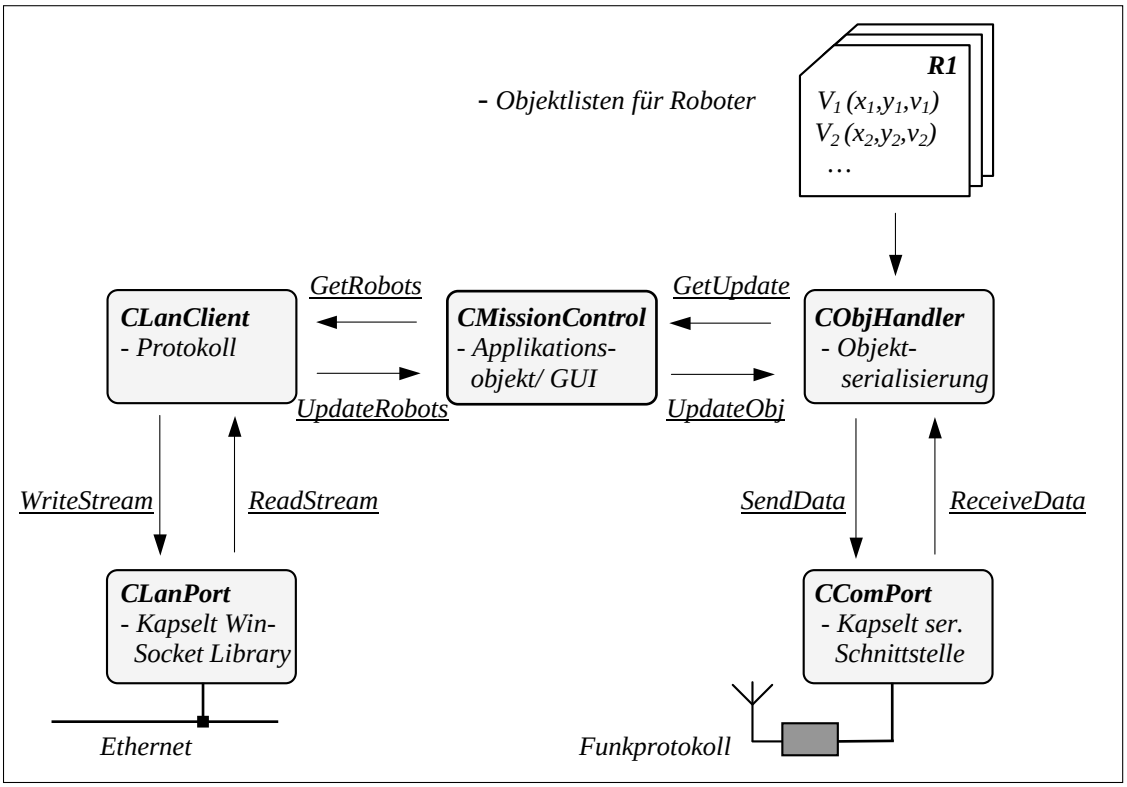


Abbildung 5.4 Objektbeziehungen innerhalb des Moduls Mission-Control

Nachdem die geplanten Objektlisten beim Start des Moduls eingelesen wurden, stellt das Objekt CLanClient eine Netzwerkverbindung zum Image-Server her und das Objekt CComPort initialisiert die serielle Schnittstelle zur Funkbox. Danach beginnt dann der eigentliche Übertragungszyklus. Ein Durchlauf beinhaltet dabei die folgenden Schritte:

- Bestimmen der nächsten Verhaltensobjekte für die Roboter
- Aktuelle Roboterpositionen vom Image-Server anfordern
- Verhaltensobjekte ggf. mit aktuellen Positionen korrigieren
- Verhaltensobjekte sequenziell übertragen: $\rightarrow R_1 ...ok \rightarrow R_2 ...ok \rightarrow R_3 ...ok$
- Startsignal synchron übertragen: $\rightarrow R_1+R_2+R_3 ...ok,ok,ok$
- Warten, dass alle Roboter ihre Zielposition erreicht haben.

Die Besonderheit des hier gewählten Verfahrens ist die zentrale synchrone Steuerung der Roboter. Sie entspricht den technischen Gegebenheiten hinsichtlich Sensorik und

Datenübertragung. Die vom Planungsmodul erstellten Objektlisten benötigen dazu aber einen exakten zeitlichen und räumlichen Bezug zueinander, was über die ursprüngliche Planung von Verhalten hinausgeht und eine Annäherung an die Hardware darstellt. Im Verlauf der Ausführung auftretenden Abweichungen wirken sich damit aber nachteilig aus, insbesondere dann, wenn nur geringe Abstände der Roboter untereinander geplant waren (z.B. Formationswechsel). In [Schiedeck, 2002] wird deshalb eine Erweiterung dieses Moduls vorgeschlagen, bei der die Informationen des Image-Servers mit den erwarteten Positionen verglichen werden, um aus der Differenz Korrekturwerte abzuleiten, die der Abweichung entgegensteuern – eine Art digitale Regelung.

5.3 Software Architektur – Simulation

Die synchrone Steuerung, wie sie im Abschnitt 5.2 beschrieben wurde, stellt in gewisser Weise einen Kompromiss zwischen echter lokaler Ausführung eines zuvor geplanten Verhaltens einerseits und der bloßen Fernsteuerung der Roboter andererseits dar. Die rein lokale Ausführung scheitert an der begrenzten Sensorik, insbesondere der Erkennung der anderen Roboter im Nahbereich (z.B. Folgeverhalten oder Kollisionsvermeidung). Die reine Fernsteuerung entspricht nicht dem hybriden Ansatz der Planungsstrategie.

Es wurden deshalb zwei unterschiedliche Wege hinsichtlich einer besseren Umsetzung der hybriden Planungsstrategie verfolgt:

- Erhöhung der Leistungsfähigkeit der Roboterplattform insgesamt durch lokales Kamerasystem auf den Robotern sowie verbesserte Komponenten der globalen Bilderkennung
- Abstraktion der Roboterplattform und der damit verbundenen technischen Details durch eine Simulationsumgebung mit dem Ziel, die minimalen Anforderungen an eine entsprechende Roboterplattform definieren zu können.

Die geschaffene Simulationsumgebung soll in diesem Abschnitt Gegenstand der Betrachtung sein. Ansätze zur Auswertung von lokalen Kamerabildern werden später in Abschnitt 6.2 unter der Überschrift weiterführende Experimente beschrieben.

5.3.1 Integration in die Steuerungsarchitektur

Prinzipiell erfolgte die Implementierung der Simulationsumgebung als weiteres Modul mit der Bezeichnung *RobotLab*. Da aber auch das Planungsmodul (*CPathPlanner*) sowie die Schnittstelle zum Image-Server (*CLanPort/CLanClient*) schon vollständig integriert wurden, handelt es sich bei der Simulationsumgebung eher um einen eigenständigen Client (vgl. Abbildung 5.1). Damit das Modul auch als einzelne Anwendung genutzt werden kann, wurde zusätzlich eine Eingabe für die Lage der Roboter sowie für die Hindernisse in Form eines graphischen Editors vorgesehen.
Damit sind alle Simulationsschritte, angefangen bei der Gestaltung unterschiedlicher Hindernissituationen über die Routenplanung bis hin zur Formationsfahrt, in einer Testumgebung möglich. Ziel ist natürlich die Erweiterung um eine Schnittstelle zu den realen Robotern, um schließlich die Simulation auch mit der Realität vergleichen zu können.

5.3.2 Abstraktion und Simulationsmodell

Die Güte einer Simulation hängt im Wesentlichen vom zugrundeliegenden Modell ab. Das Problem eines Simulationsmodells sind jedoch die daran vorgenommenen Abstraktionen bzw. Vereinfachungen gegenüber der Realität. Im vorliegenden Fall sind es mehrere Aspekte, die in einem geeigneten Simulationsmodell zusammengefasst werden müssen. Dazu zählt natürlich in erster Linie das Bewegungsmodell der Roboter, aber ebenso ist eine Beschreibung der Umwelt sowie der externen Sensorik des Roboters erforderlich. Letztlich muss auch in der Simulation eine Rückkopplung der Aktorik auf die Sensorik über die Umwelt erfolgen. Abstandssensoren müssen in der Lage sein, Hindernisse der Umgebung detektieren zu können. Allerdings ist es nicht unbedingt erforderlich, interne Sensoren, wie z.B. die Radencoder, zu simulieren. Das Bewegungsmodell abstrahiert diese Sensoren vollständig. Das führt aber wiederum dazu, dass real vorhandene Toleranzen, die solche Radencoder nun einmal besitzen, von vornherein aus der Simulation ausgeschlossen werden. Generell ist eine Simulation deshalb immer idealisiert, da insbesondere die zufälligen Fehler schwer beschreibbar sind.

Mit der Prämisse, die minimalen Anforderungen hinsichtlich Sensorik und Steuerverhalten für Formationsfahrten zu bestimmen, wurden für die Simulation zunächst drei Basisobjekte definiert:

- Umwelt (*CGrid*→Hindernisse)
- Roboter (*CRobot*→Sensorik→Verhalten→Bewegung)
- Beobachter (*CBoard*→Route→Positionen→Kommunikation)

Dabei erfolgt die Beschreibung der Umwelt durch eine Rasterkarte (*occupancy grid*), wie sie schon in Abschnitt 3.2.4 beschrieben wurde. Diese Darstellungsform ist auch im Sinne einer simulierten Sensorik von Vorteil, weil sich beispielsweise die Auswertung eines IR-Tastsensors auf ein lineares Aufsuchen und Vergleichen in einer Matrix zurückführen lässt.
Für die Bewegung der Roboter wurde das Modell einer Punktmasse, die durch Einwirkung eines Steuervektors (virtuelle Kraft) ihren Bewegungszustand ändert, übernommen. Die Beschreibung eines entsprechenden Modells erfolgte in Abschnitt 2.3.1. Die Bestimmung des Steuervektors geschieht hier durch Auswertung der Verhaltensreaktion „Folgen“. Entsprechend dem Formationsmodell aus Abschnitt 3.2.2 wird dazu kontinuierlich ein Zielpunkt berechnet, der sich aus der Position des vorausfahrenden Roboters und eines darauf bezogenen Führungsvektors ergibt. Durch Sensorkontakt mit Hindernissen kann dieser Zielpunkt temporär verlagert werden. Dazu ist ständig ein Verhalten „Hindernisvermeidung“ aktiv, welches nur auf Basis der Sensorinformation einen geeigneten Zielpunkt ermittelt. Diese Steuerung ist damit eine Abwandlung der kombinierten Steuerverhalten, wie sie in Abschnitt 2.3.3 beschrieben sind, mit dem Vorteil, dass die Bewegungsänderung gleichmäßiger erfolgt, als bei reiner Vektorsummation.
Der Beobachter schließlich ist als globaler Datenpool (*black board*) ausgelegt und abstrahiert die Funkübertragung zwischen den Robotern sowie die zur globalen Steuerung. Außer der eigentliche Route in Form von Segmenten werden hier die Positionen der Roboter abgelegt. Jeder Roboter trägt selbst seine aktuelle Position ein und stellt sie somit allen anderen zur Verfügung. Diese Abstraktion vernachlässigt

allerding jegliche Übertragungszeit oder gar Verzögerung durch Wiederholungen eines fehlerhaften Datenpakets.

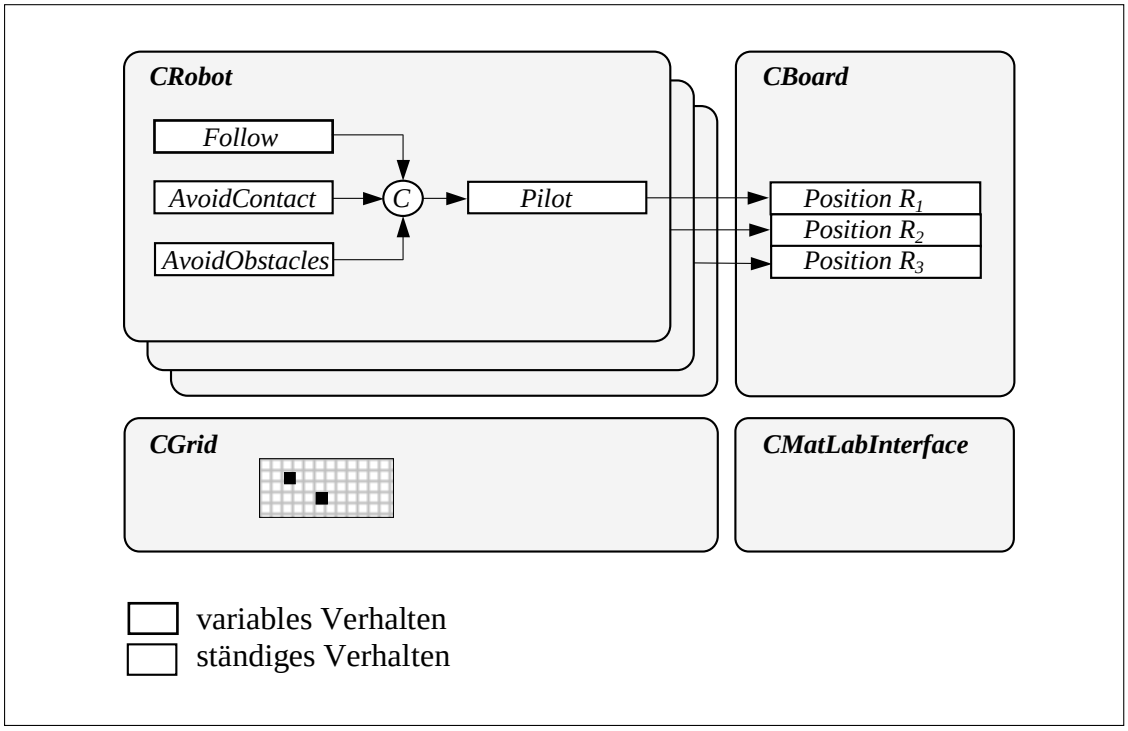


Abbildung 5.5 Simulationsobjekte innerhalb des Moduls RobotLab

Die Simulation läuft in einer Zykluszeit von 20ms ab, in der jeweils die drei simulierten Roboter zur Berechnung ihrer nächsten Position und zu deren Darstellung veranlasst werden. Die Roboter agieren entsprechend ihrer Verhalten und auf der Basis der zuvor geplanten Route. Die Verhältnisse innerhalb der Simulation wurden so weit wie möglich der Realität angepasst. Alle Maße wurde im gleichen Maßstab abgebildet und auch die erreichbaren Geschwindigkeiten entsprechen denen der realen Roboter. Die IR-Sensorik wurde in Abwandlung zu den realen Robotern in einem Kreissegment von 90° lediglich an der Vorderseite nachgebildet. Dieser Bereich wurde in 10 Tastpositionen unterteilt.

Das Modul RobotLab ist als Native-Code-Applikation für das Betriebsystem Windows® realisiert. Der Vorteil liegt in der höheren Verarbeitungsgeschwindigkeit, was insbesondere während der Planung von Vorteil ist. Mit dem Programm MATLAB/Simulink® existiert eine leistungfähige Software, die für modellbasierte Simulationen sowie deren graphische Darstellung und Auswertung geeignet ist. Eine Schnittstelle zu diesem Programm erweitert die Möglichkeiten des Moduls RobotLab im Hinblick auf Darstellung und Auswertung erheblich.

5.3.3 MATLAB®-Schnittstelle

MATLAB® bietet als Software für technische Anwendungen eine eigene Programmiersprache. Die Programme werden dabei in sogenannten *m-Files*[17] formuliert und gespeichert. Die Ausführung dieser Programme erfolgt interpretativ, was sie für aufwändige Iterationen eher ungeeignet erscheinen lässt. Die Möglichkeiten der Programmiersprache sind aber dennoch enorm. Eine Vielzahl von API-Funktionen[18] unterstützen nahezu jedes Problem. Da über m-Files effektiv Dateien gelesen werden können und ebenso ein direkter Zugriff auf die Virtual-Reality-Toolbox besteht, wurde als Verbindung zu MATLAB® eine Datei-Schnittstelle definiert. Darüber können sowohl die Lage der Hindernisse als auch die aktuellen Positionen der Roboter während einer laufenden Simulation ausgetauscht werden. Die Übergabe erfolgt in Form eines strukturierten Textes.

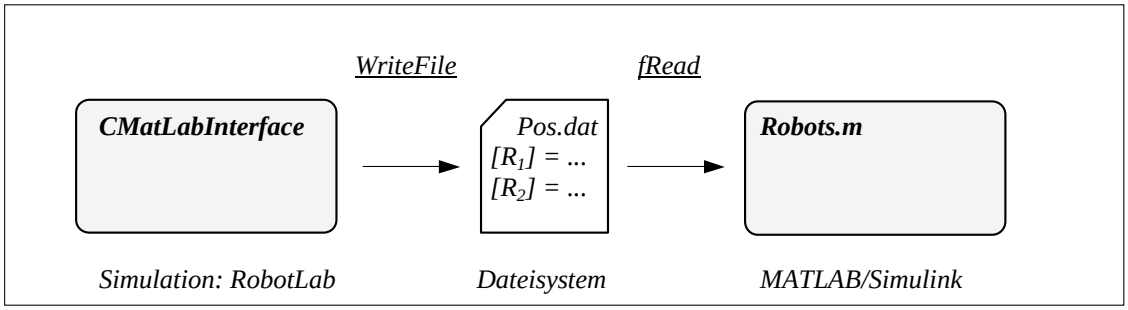


Abbildung 5.6 Dateischnittstelle RobotLab→MATLAB®

Durch das Modul *Robots.m* werden die Positionsdaten der Roboter während einer laufenden Simulation importiert. Der Dateizugriff erfolgt dabei ausreichend schnell, zumal moderne Betriebsysteme Dateien temporär im Hauptspeicher vorhalten[19], um so häufige Zugriffe auf externe Datenträger zu vermeiden. Synchronisationsprobleme bei evtl. gleichzeitigen Zugriff beider Seiten auf die Datei, werden von der schreibenden Seite (*CMatLabInterface*) abgefangen. Eine momentan geöffnete Datei kann nicht geschrieben werden und eine geschriebene Datei wird erst durch einen atomaren Zugriff (Umbenennen) als vorhanden markiert. Die lesende Seite (*Robots.m*) öffnet die Datei, sobald sie vorhanden ist, liest sie und entfernt sie schließlich.

Die Dateischnittstelle ist recht einfach zu implementieren und bietet vielfältige Erweiterungsmöglichkeiten. Das Modul Robots.m kann sowohl in MATLAB® selbst als auch unter Simulink® verwendet werden.
Im vorliegenden Fall hat MATLAB® nur die Aufgabe der erweiterten Darstellung und Auswertung. Auf Grund der Möglichkeiten kann eine Simulation aber auch direkt in Simulink® ablaufen. Dazu wäre eine Umsetzung des Simulationsmodells nötig sowie ein Import der Ergebnisse des Planungsmoduls. Das Planungsmodul sollte sinnvollerweise als Bibliothek[20] eingebunden werden.

[17] Quelltexte mit der Erweiterung *.m
[18] API = Applikation Programmers Interface
[19] Windows® z.B. verwaltet einen File-Cache im RAM, in dem alle Dateien vorläufig erhalten bleiben
[20] MATLAB® unterstütz die Einbindung von Native-Code in Form von DLLs (Dynamic Link Library)

6 Ergebnisse aus Simulation und Experiment

6.1 Simulation

Die Simulation ermöglicht im Vergleich zur realen Roboterplattform eine flexible Variation der Modelle und Zusammenhänge. Beim Versuch, die minimal nötige Konfiguration der Roboter hinsichtlich Sensorik und Kommunikation zu bestimmen, können beispielsweise unterschiedliche Sensorvarianten „konstruiert“ werden, ohne dafür einen Roboter umrüsten zu müssen. Diese Flexibilität hat sich als außerordentlich günstig erwiesen. Zudem zeichnet sich die Simulation dadurch aus, dass eine höhere Wiederholrate bei identischen Bedingungen möglich ist. Während die Roboter für einen neuen Testlauf auf der Plattform zunächst wieder in eine definierte Ausgangslage gebracht werden müssen, bedarf es bei der Simulation nur eines Neustarts.

Aus einer Reihe von Versuchen hat sich ergeben, dass für die Roboter ein binärer IR-Abstandssensor in Fahrtrichtung für die Hindernisvermeidung als ausreichend erscheint. Der als Kreissegment ausgeführte Sensor überstreicht die gesamte Breite des Roboters. Bei Kontakt mit einem Hindernis muss darüber zumindest ein diskreter Winkel zwischen der Fahrtrichtung und dem Hindernis bestimmt werden können. Umgekehrt proportional zu diesem Winkel führt der Roboter dann eine Verlagerung seines temporären Zielpunktes durch. Bei einem direkten Kontakt erfolgt eine Verlagerung um bis zu 90°, während ein seitlicher Kontakt nur zu einer geringen Verlagerung führt. Außerdem wird der Zielpunkt in einen definierten Abstand zur aktuellen Position des Roboter gebracht. Der Roboter versucht dann den neuen Zielpunkt zu erreichen und nimmt erst anschließend den ursprünglichen Zielpunkt wieder auf. In der Simulation wird dieser Sensorbereich als grauer Streifen vor dem Roboter dargestellt. Der gesamte Bereich wurden in 10 diskrete Sensoren unterteilt, die somit eine Winkelauflöung von ca. 10° erlauben.

Hinsichtlich der Kommunikation zwischen den Robotern wurde eine starke Vereinfachung verglichen mit der realen Plattform vorgenommen. Jeder Roboter kennt die Positionen aller anderen Roboter, ohne sich um deren Übertragung kümmern zu müssen. Anhand der eigenen Position in Bezug zu allen anderen, kann jeder Roboter seine Geschwindigkeit reduzieren, um eine bevorstehende Kollision zu vermeiden. Dabei wurde eine Priorität der Roboter untereinander eingeführt. Roboter R_1 hat die höchste, Roboter R_3 die niedrigste Priorität. Der Roboter mit der nierigeren Priorität verringert seine Geschwindigkeit proportional zum Abstand bei Annäherung an einen Roboter höherer Priorität. Verringert sich der Abstand dennoch weiter, bewegt sich der Roboter schließlich auch rückwärts. Ein Ausweichen hat sich als ungünstig erwiesen, da sich damit die angestrebte Formation verändert.

Diesen beiden grundlegenden Verhalten wurden als *AvoidObstacles* und *AvoidContact* bezeichnet und fest implementiert. Zusätzlich wurde ein variables Verhalten *Follow* implementiert, durch das eine gerichtete Bewegung entsteht. Das Verhalten definiert

kontinuierlich einen temporären Zielpunkt, der sich aus der relativen Lage zu einem führenden Roboter ergibt.

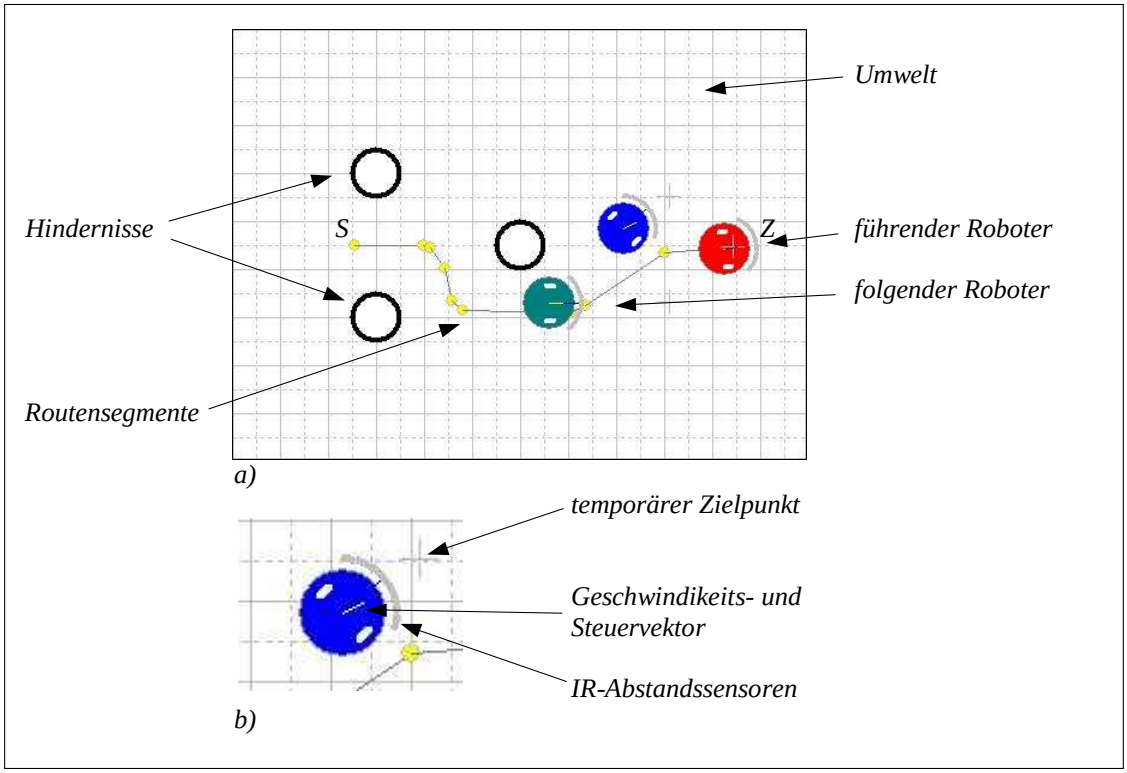


Abbildung 6.1 Simulationsansicht RobotLab
a) Simulationsobjekte
b) Roboterobjekt

Um die Möglichkeiten von MATLAB® hinsichtlich Darstellung und Auswertung nutzen zu können, wurde die in Abschnitt 5.3.3 beschriebene Schnittstelle implementiert.

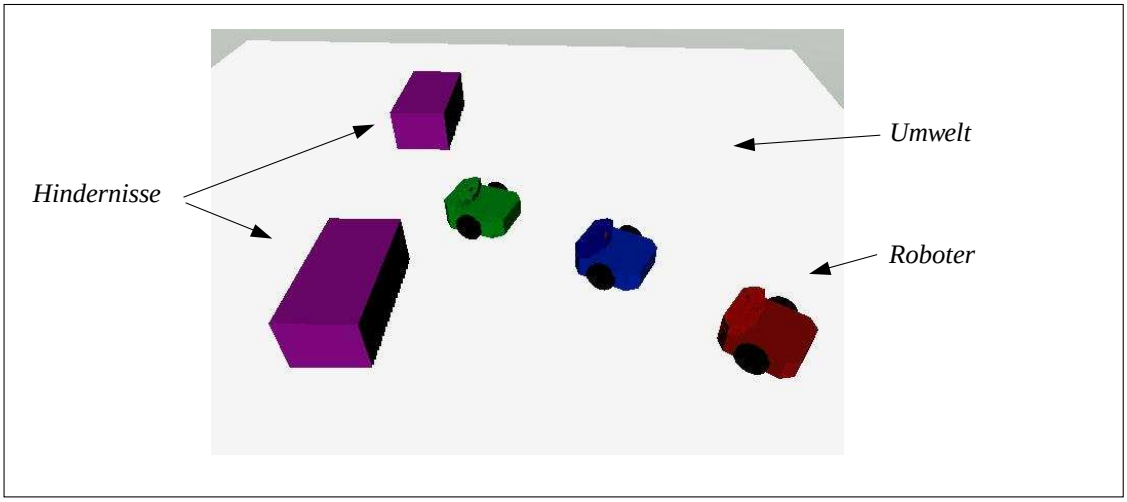


Abbildung 6.2 Simulationsansicht MATLAB®

In einem ersten Schritt wurden nur die Roboterpositionen an die VR-Toolbox übertragen, während die eigentliche Simulation noch in RobotLab erfolgte. Da die Darstellungsmöglichkeiten beachtlich sind, ist es empfehlenswert, auch die Simulation in MATLAB/SimuLink® zu übertragen.

6.1.1 Beispiele für geplante Routen

Die im Rahmen der Missionsplanung gewählte und implementierte Strategie sucht bevorzugt die Routen, die einen möglichst breiten Korridor für die Formation ermöglichen. Dieser Ansatz wird erfüllt, wie die drei unterschiedlichen Fälle aus Abbildung 6.3 zeigen. Bei freier Wahl entscheidet sich das Routensuchverfahren für die breitest mögliche Route. Wird diese versperrt, so wählt das Verfahren die nächst schmalere Route, bis schließlich auch die Passage der Engstelle akzeptiert wird. Diese Strategie weist allerdings zwei Nachteile auf, die in bestimmten Situationen zu extremen Umwegen führen kann:

- Selbst dann, wenn eine kürzere Route mit einem nur unwesentlich schmaleren Korridor existiert, wählt der Algorithmus eine längere Route, sofern die den breiteren Korridor ermöglicht.
- Für die Beurteilung der verfügbaren Korridorbreite wird nur die engste Stelle herangezogen. Damit werden Routen, die für die Formation überwiegend günstig zu fahren sind, aber eben eine Engstelle besitzen, nicht gewählt.

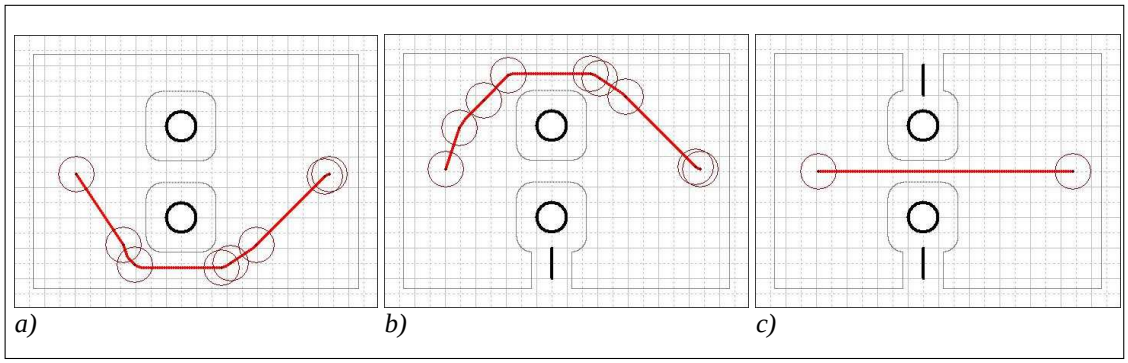


Abbildung 6.3 Beispiele für die gewählte Routenplanungsstrategie
a) Route bei freier Wahl
b) Route bei eingeschränkter Wahl
c) Einzig noch mögliche Route

Das Planungsverfahren selbst hat sich jedoch als sehr robust herausgestellt und liefert, unter Berücksichtigung der Strategie, sehr gute Ergebnisse. Die Routen haben stets einen ausreichenden Abstand zu Hindernissen und verlaufen nahe der Ideallinie.

6.1.2 Formationsfahrten um Hindernisse

Für die Simulation wurden drei unterschiedliche Formationen realisiert: Linie, Dreieck und Reihe. Die Verhaltensplanung musste daher in Abhängigkeit des verfügbaren Korridors nur eines der entsprechenden Verhalten für den jeweiligen Roboter auswählen und die Punkte für erforderliche Formationswechsel bestimmen. Da die drei

Formationen ineinander überführbar sind, bedarf es keiner weiteren Schritte zur Bestimmung von Übergangsformationen. Die so geplanten Missionen können schließlich in der Simulation ablaufen. Dabei hat sich gezeigt, dass insbesondere die Kombination aus Verhaltensplanung und reaktiver Ausführung auch in komplizierten Situationen wie der Passage einer Engstelle zu guten Ergebnissen führt. Eine rein reaktive Lösung führt dagegen oft zu einem Abspalten bzw. „Verlaufen“ einzelner Roboter. Die Routenplanung ist nur insofern von Bedeutung, da die Verhaltensplanung anhand einer konkreten Route erfolgt. Letztlich könnte diese aber auch anderweitig vorgegeben werden.

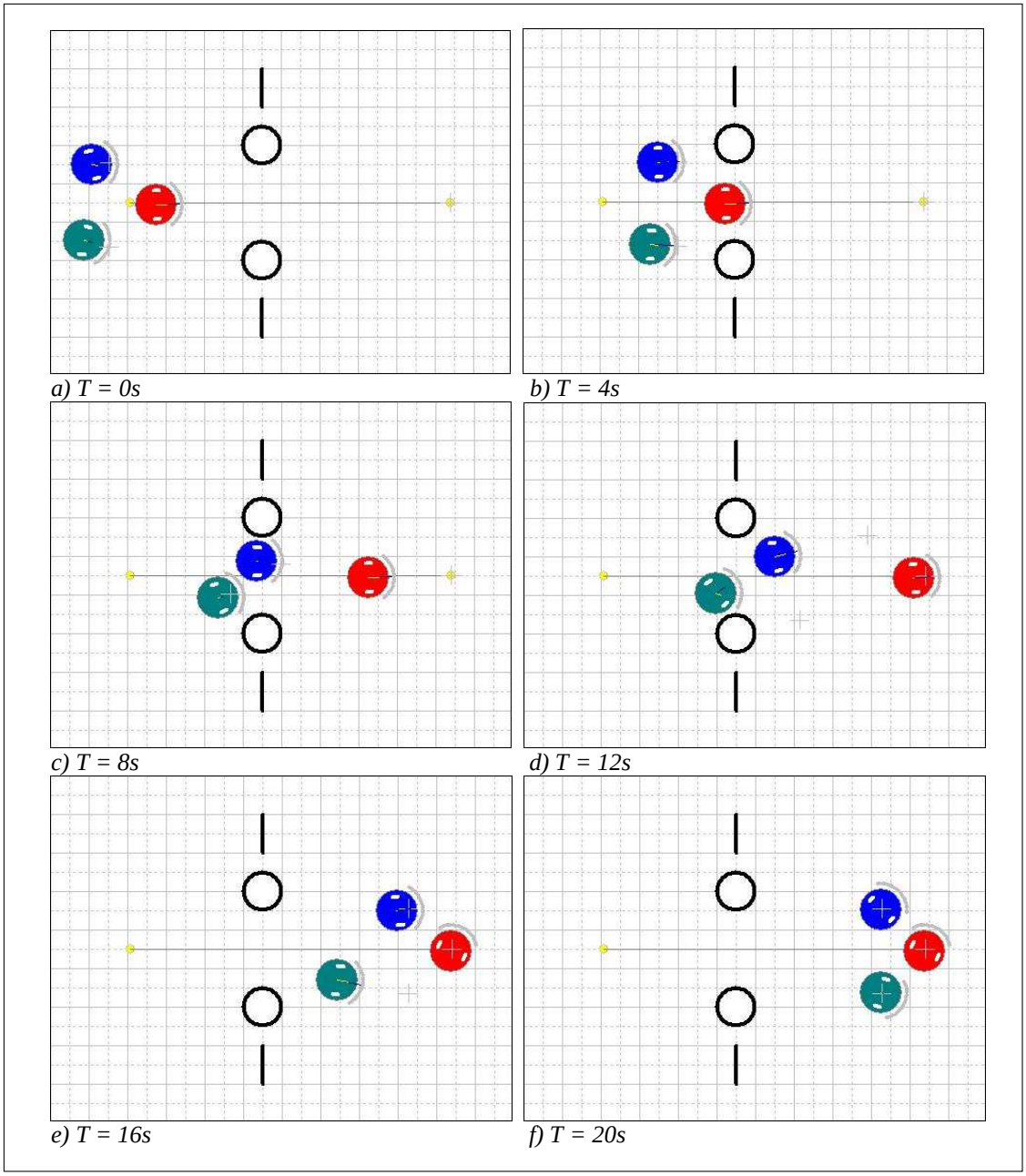


Abbildung 6.4 Simulation eines Formationswechsels an Engstelle
a)..c) Auflösen der Dreieckformation
d)..f) Wiederherstellen der Formation

6.2 Experiment

Im Gegensatz zu reinen Softwarelösungen, wie die der Simulation, sind Experimente auf dem Gebiet der Robotik immer durch ein Zusammenwirken aus Hard- und Software gekennzeichnet. Zur Ansteuerung der verwendeten Hardware sind meist spezielle Softwarekomponenten nötig. Eine Simulation kann die dabei auftretenden Wechselwirkung nur sehr begrenzt widerspiegeln.

Die Plattform, die für die praktischen Versuche entwickelt wurde, umfasst neben den mobilen Kleinstrobotern auch die stationären Komponenten für die kameragestützte Positionsbestimmung. Da dazu ein entsprechend komplexes Softwaremodul erstellt wurde, unterteilen sich die nachfolgend beschriebenen Experimente in Bilderkennung und Bewegungssteuerung. In einem anschließenden Abschnitt werden aufgetretene Probleme in Bezug zu diesen beiden Themenbereichen diskutiert.

6.2.1 Globale Bilderkennung

Ein wesentlicher Bestandteil der praktischen Robotersteuerung ist die kameragestützte globale Positionsbestimmung. Hier wurde, im Gegensatz zu rein farbbasierten Verfahren, eine Variante realisiert, die zunächst nur die Kanten eines Binärbildes auswertet und Farbinformationen letztlich nur zur Identifikation der Roboter verwendet (vgl. Abschnitt 4.2). Für erste Versuche wurde eine Kamera verwendet, deren Bildqualität, bedingt durch eine einfache Optik sowie eine reduzierte Auflösung (¼ PAL), vergleichsweise gering war. Die Erkennung der Kanten ermöglichte dennoch eine sichere Positionsbestimmung. Eine Identifikation der Roboter anhand der Farbmarkierung war dagegen oft nicht möglich. Ursachen dafür waren die geringe Fläche der Farbpunkte in Kombination mit Schwankungen der Beleuchtung (vgl. Abb. 6.5). Die resultierenden Farbwerte konnten dann nicht eindeutig zugeordnet werden.

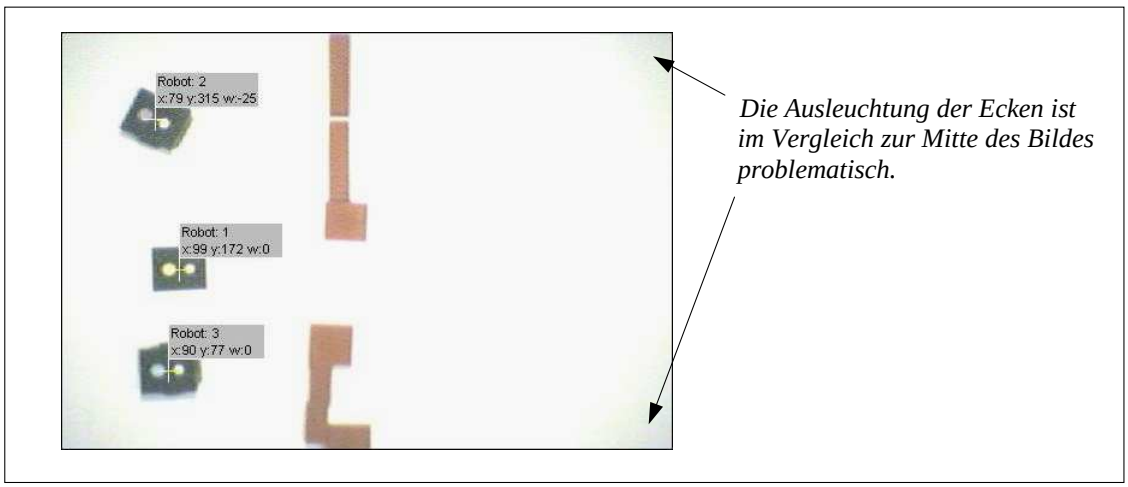


Abbildung 6.5 Kamerasicht mit eingeblendeten Positionsdaten

Um diese Probleme zu umgehen und um die Positions- und Winkelauflösung zu erhöhen, wurde schließlich eine industrielle CCD-Kamera[21] verwendet. Damit konnte eine sichere Positionsbestimmung und Identifikation der Roboter erreicht werden. Die Positionen der Roboter wurden mit einer maximalen Abweichung von $\Delta x_{max}=\pm 2.5mm$, $\Delta y_{max}=\pm 4mm$ und $\Delta \varphi_{max}=\pm 1°$ gemessenen. Damit liegen die Abweichungen noch unter 1%. Aufgrund der hohen Auflösung der Kamera sowie des gewählten Algorithmus zur Kantendetektion, konnten jedoch nur 4-5 Bilder pro Sekunde analysiert werden.
Die Erkennung der rot markierten statischen Hindernisse, die immer nur einmal im Vorfeld der Routenplanung erfolgte, war vergleichsweise problemlos. Hier ergaben mehrere Bildpunkte ein Hindernisraster, wobei bereits ein erkannter Bildpunkt das Raster schon als Hindernis markierte.

6.2.2 Bewegungssteuerung der Roboter

Die Steuerung der Roboter erfolgte nach dem im Abschnitt 5.2.3 beschriebenen Synchronverfahren. Jede Bewegungssequenz wurde zunächst sequenziell in Form von

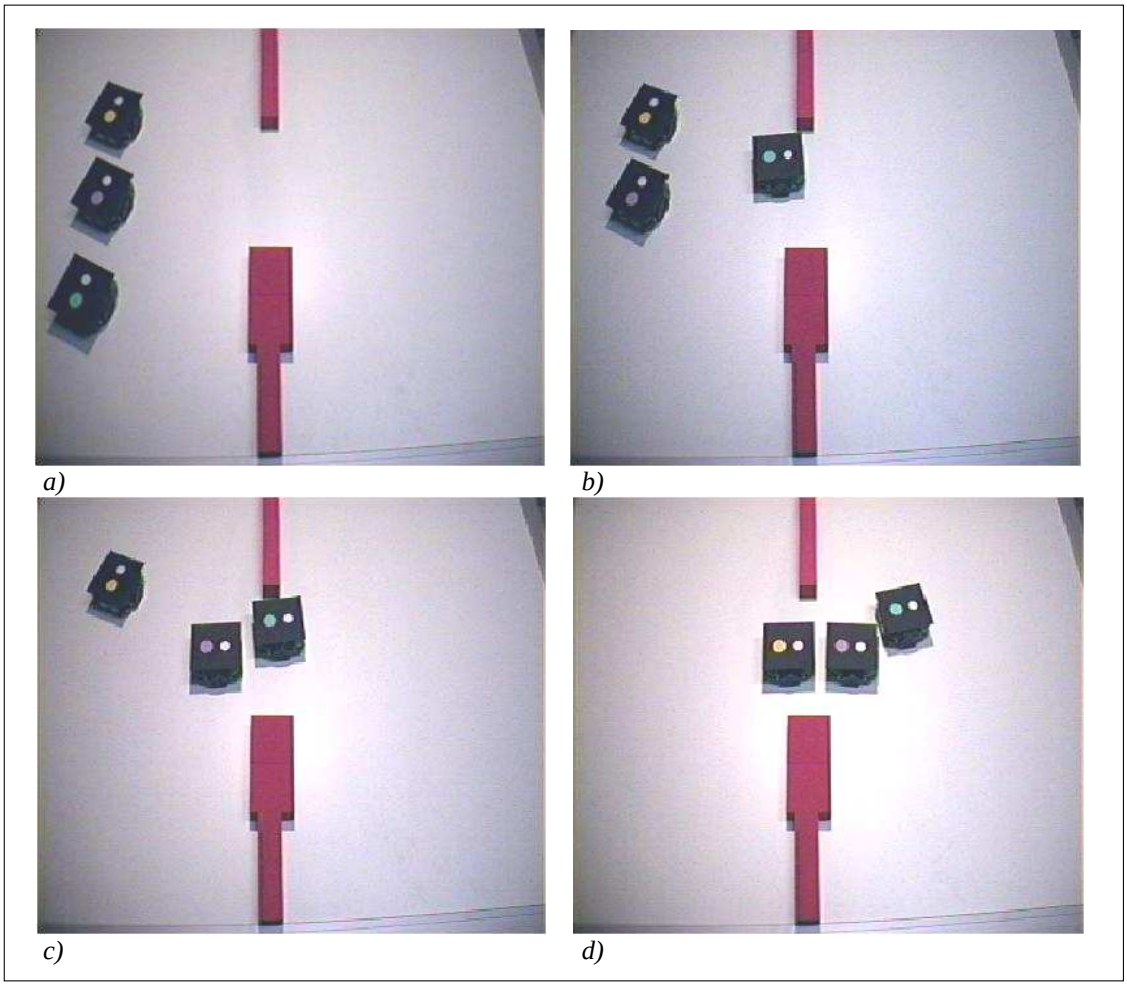


Abbildung 6.6 Formationswechsel an Engstelle
a) Ausgangsformation Linie
b)..d) Formationsübergang

[21] Teli CS-5260DP, 752x582 Pixel, analog FBAS(PAL)

Verhaltensobjekten mit absoluten Koordinaten an die drei Roboter übertragen. Die Roboter berechneten daraus eine Abfolge von Rotation und Translation, um von ihrer lokal mitgeführten Position zur vorgegebenen Zielposition zu gelangen. Schließlich wurden die Roboter synchron gestartet und steuerten ihre zuvor berechneten Zielpunkte an. Die Synchronsteuerung ermöglicht dabei eine Formationsfahrt auch ohne spezielle lokale Sensorik zur Abstandsmessung. Bedingt durch die zyklische Übertragung der einzelnen Verhaltensobjekte, ergibt sich aber ein diskontinuierlicher Bewegungsablauf. Beginnend mit der kameragestützten Erfassung der Roboterpositionen und Hindernisse, über die Planung der Routen bis hin zur Bewegung der Roboter, war es damit bereits möglich, den gesamten Ablauf zu erproben.

Als Konsequenz dieser ersten Experimente sowie der Erkenntnisse der Simulation müssen die Roboter lokal in der Lage sein, die Position eines vorausfahrenden Roboters kontinuierlich zu bestimmen, um davon abgeleitet eigene temporäre Zielpunkte berechnen zu können. Nur so lässt sich ein dynamisches Folgeverhalten realisieren. Für die Vermeidung von Kollisionen mit anderen Robotern ist es außerdem notwendig, auch deren Position zu kennen. Dazu sollten die Roboter ihre lokal mitgeführten Positionen zyklisch veröffentlichen. Schließlich muss jeder Roboter auch noch vor ihm befindliche Hindernisse detektieren und entsprechend ausweichen können.
Um diesen Anforderungen gerecht werden zu können, wurden die Roboter überarbeitet und u.a. mit einem leistungsfähigeren Mikrocontroller ausgestattet, der auch in der Lage ist, aufwändige Berechnungen in Echtzeit auszuführen. Außerdem wurde die Funkschnittstelle verbessert, wodurch höhere Datenraten möglich sind.
Letztlich können die benötigten lokalen Fähigkeiten, insbesondere für das Folgeverhalten, aber nur mit einer geeigneten Sensorik realisiert werden. IR-Sensoren ermöglichen zwar die Erkennung von Objekten, erlauben aber keine Differenzierung zwischen Hindernis und Roboter. Eine interessante Lösungsvariante besteht in der Verwendung einer lokalen Kamera. Grundlegende Experimente dazu werden im folgenden Abschnitt dargestellt.

6.2.3 Weiterführende Experimente

Die Roboter auch lokal mit einer Kamera zu versehen, würde die Möglichkeit einer Sensorintegration ergeben. Es wären gleichermaßen Entfernungsmessungen zu vorausfahrenden Robotern wie auch die Erkennung von Hindernissen möglich. Da eine entsprechende Hardware (Kamera-Chip) verfügbar ist, war zunächst zu klären, ob eine Bildanalyse mit vergleichsweise geringem Rechen- und Speicheraufwand überhaupt möglich ist. Außerdem besitzt so eine kleine Kamera nur eine geringe Auflösung – kann man damit eine Entfernung bestimmen?

Zu diesem Zweck wurde eine einfache CCD-Kamera zunächst über USB[22] an einen Rechner angeschlossen und auf eine geringe Auflösung programmiert. Die eingelesenen Bilder wurden hinsichtlich einer bestimmten Farbe durchsucht. Die gefundenen Bildpunkte wurden gezählt und aus allen Punkten $P(x_i,y_i)$ wurde der Schwerpunkt berechnet.

[22] USB = Universal Serial Bus; serielle Hochgeschwindigkeitsschnittstelle (12 Mbit/s)

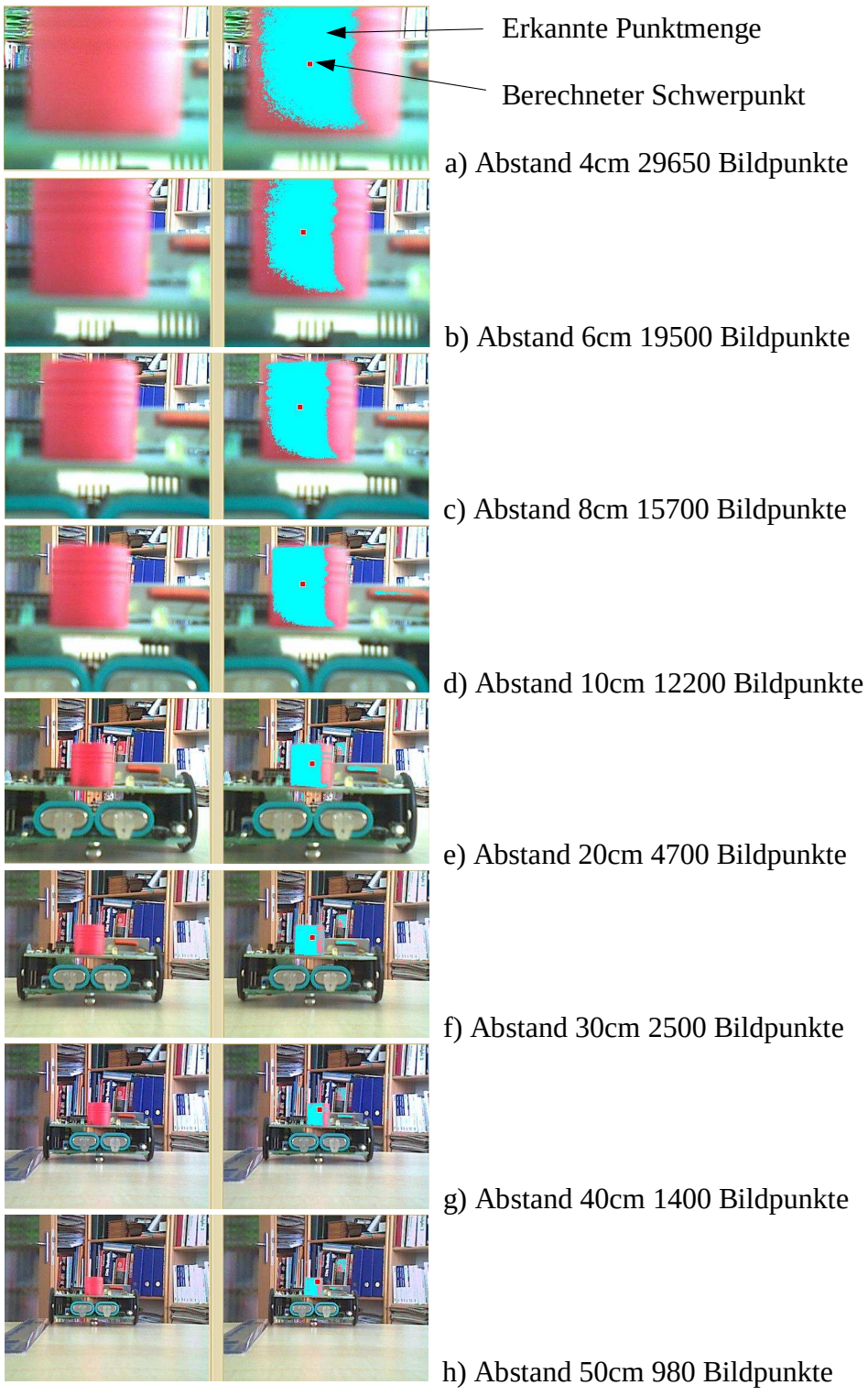


Abbildung 6.7 Testreihe für lokale Bilderkennung
(Kamera: USB[23], Markierung: ∅=2cm, h=2cm, Abstand: d=4..50cm)

Die Testreihe zeigte eindeutig einen reproduzierbaren Zusammenhang zwischen Anzahl der gefundenen Bildpunkte und Abstand der Markierung zur Kamera.

[23] Digitalkamera (USB-WebCam): 352x288, f=6.0mm, ohne Beleuchtung, nur Tageslicht

Insbesondere war auch die Stabilität eines derart einfachen Verfahrens überzeugend. Die Bilderkennung wurde bewusst vor einem „bunten Hintergrund“ durchgeführt, um so Farbstörungen zu simulieren.

6.3 Problemdiskussion

Die hardwareseitige Realisierung und Programmierung eines bisher beschriebenen Multirobotersystems wirft eine Vielzahl von Detailproblemen auf. Jede Lösung ist dabei immer nur eine mögliche Variante, die neben tatsächlichen Gegebenheiten auch durch persönliche Vorstellungen und Erfahrungen geprägt wird. Zeitliche und materielle Beschränkungen führen darüber hinaus auch zu Kompomissen, deren Konsequenzen sich erst spät im weiteren Verlauf der Entwicklung abzeichnen. In diesem Abschnitt sollen deshalb die vorgestellten Ergebnisse und Lösungen hinsichtlich ihrer „Problemstellen“ untersucht werden.

6.3.1 Probleme der Bilderkennung

Als globales Erkennungssystem für die Positionen der Roboter sowie für deren Orientierung ist eine Bilderkennung bei idealisierten Bedingungen hinsichtlich Untergrund, Markierungen der Roboter sowie Beleuchtung Stand der Technik. Bei fußballspielenden Robotern werden mehrere Spieler in zwei Mannschaften unterschieden. Der Vorteil der Bilderkennung liegt hier in der simultanen Erkennung aller Objekte. Zu einem Zeitpunkt t_0 werden gleichzeitig die Positionen aller Markierungen bestimmt.

Eine Bilderkennung ist jedoch prinzipiell sehr anfällig gegenüber Schwankungen der Beleuchtung. Solche Schwankungen entstehen durch wechselnde Grundbeleuchtung, durch seitliche Einstrahlung (z.B. Fenster) oder durch Reflexionen. Ein prinzipielles Problem stellt die Wahl einer geeigneten Grundfläche für die Experimental-Plattform dar. Um einen optimalen Kontrast zu erreichen und Reflexionen zu vermeiden, werden die Roboter zumeist matt schwarz abgedeckt und auf einem hellen Untergrund platziert. Der Untergrund reflektiert dadurch jedoch die Beleuchtung mehr oder weniger direkt in die Kamera. Bei nur einer zentralen Lichtquelle bildet sich ein überdurchschnittlich heller Bereich in der Mitte der Plattform. Zu den Rändern nimmt die Beleuchtungsstärke dann ab. Die Verwendung mehrerer Lichtquellen oder indirekte Beleuchtung führt zu einer gleichmäßigeren Ausleuchtung. Die Oberfläche sollte diffus und wenig reflektierend sein; z.B. Stoff oder Filz.

Ein negativer Effekt ist auch die Schattenbildung der Roboter bei Beleuchtung mit einer Punktquelle. Auf Grund einer entsprechenden Bauhöhe der Roboter ergibt sich ein mit der Position variierender Schatten. Für die Kamera stellt sich der Schatten als die Kontur des Roboters dar. Der Erkennungsalgorithmus sollte deshalb möglichst unanfällig gegenüber Änderungen der äußeren Kontur des Roboters sein. Es dürfen nur Markierungen vermessen werden, die im Inneren der Kontur liegen.

Die Wahl der Kameraposition wird durch die Brennweite (Öffnungswinkel) ihres Objektives bestimmt. Bei großem Öffnungswinkel ergibt sich zwar eine geringere Höhe der Kamera, es entstehen aber Verzerrungen im Randbereich. Einen ähnlichen

Einfluss hat die Blende des Objektives. Bei kleiner Blende ergibt sich eine höhere Tiefenschärfe, es reduziert sich aber die Gesamthelligkeit des Bildes. Befindet sich die Kamera in einem ausreichend großen Abstand von der Plattform, so wird nur eine geringe Tiefenschärfe benötigt.

Nicht zuletzt entscheidet die Wahl des Erkennungsalgorithmus' über die resultierende Stabilität, aber auch über die Geschwindigkeit. Die Verwendung eines Binärbildes und damit die Suche nach bekannten Konturen funktioniert auch bei weniger idealen Bedingungen, ist aber aufwendiger und somit langsamer als reine Farbauswertung. Um eine bestimmte Anzahl von Markierungen unterscheiden zu können, bleibt letztlich nur die Kodierung über Farben. Bei dem in dieser Arbeit beschriebenen Erkennungsalgorithmus war aber gerade die Unterscheidung der Farben im RGB-Modell am anfälligsten. Es empfiehlt sich daher für die Unterscheidung der Farben eher das YUV-Modell zu verwenden. Der Y-Wert stellt die Helligkeit dar, während die Werte U und V für die Übergänge zwischen Rot und Grün bzw. Blau und Gelb stehen. Durch den separaten Helligkeitswert ist ein Farbvergleich mit festen Schwellwerten auch bei wechselnder Beleuchtung noch sicher.

6.3.2 Probleme der Bewegungssteuerung

Die Synchronsteuerung der Roboter basiert auf der sequenziellen Übertragung der nächsten Routensegmente mit anschließendem Startsignal. Die Informationen zur Route werden an jeden Roboter einzeln übermittelt. Das Startsignal wird dagegen mittels einer *Broadcast-Message* an alle Roboter zugleich gesandt. Probleme treten dann auf, wenn ein oder mehrere Roboter gerade diese Message nicht empfangen haben. Bei Verwendung von RF-Kommunikation ist der Verlust eines Datenpaketes durchaus akzeptabel und wird in der Regel durch eine Wiederholung korrigiert. Die Notwendigkeit zur Wiederholung erkennt der Absender im Allgemeinen am Ausbleiben einer entsprechenden Quittung. Ein Broadcast-Telegramm wird von allen adressierten Robotern in einer entsprechenden Zeitscheibe beantwortet (vgl. Abschnitt 4.3). Damit verstreicht zunächst eine gewisse Zeit bis das Programmmodul Mission Control den Übertragungsfehler erkennen kann und bis zur erneuten Übertragung nach Eingang aller anderen Quittungssignale vergeht nochmals etwas Zeit.
Im günstigsten Fall führt der verzögerte Start eines Roboters zu Abweichungen innerhalb der Formation. Die ungünstigste Auswirkung auf den Bewegungsablauf der Roboter sind jedoch Kollisionen, wenn bei geringem Abstand ein vorausfahrender Roboter das Startsignal gerade nicht empfängt.
Eine Erweiterung des Funkprotokolls speziell für derartige Probleme wurde zu Gunsten einer besseren Sensorik und damit höheren Autonomie der Roboter jedoch nicht realisiert.

6.3.3 Weitere Fehlerquellen

In diesem Abschnitt sollen einige Fehlerquellen kurz aufgelistet werden, um für ähnliche und insbesondere nachfolgende Arbeiten einen Einbick in die Details zu geben:

- *Verwendung von Halbbildern der Kamera:* Ein TV-Bild besteht aus zwei Halbbildern. Ein Halbbild umfasst nur die geraden Zeilen, das andere nur die

ungeraden. Bewegung verursacht einen Versatz von Kanten in beiden Bildern. Es empfiehlt sich daher bei der Auswertung mit nur einem Halbbild zu arbeiten, um Erkennungsfehlern vorzubeugen. Damit reduziert sich jedoch die Auflösung in Y-Richtung (Zeilen). Die Abweichungen der gemessenen Roboterwinkel waren deshalb um 0° bzw. um 180° auch größer als um 90° bzw. 270°.

- *Umsetzung von Algorithmen mit Winkelfunktionen:* Die Funktionen sin(), cos(), u.a. beziehen sich nur auf einen Quadranten und müssen daher je nach Winkellage normiert werden. Auf Seiten der Roboter kamen sogar Tabellen zum Einsatz, um zu Gunsten der Rechenzeit auf Fließkomma-Arithmetik verzichten zu können.

7 Zusammenfassung und Ausblick

In der Literatur sowie im Internet finden sich eine Vielzahl von Vorschlägen und Lösungsvarianten für die Steuerung von Robotern. Die weitaus größere Zahl betrachtet dabei stets einen einzelnen Roboter in Wechselwirkung mit seiner Umgebung. Teams von Robotern findet man vorwiegend im Zusammenhang mit dem bekannten Roboterfußball oder zur Lösung von Transportproblemen, bei denen mehrere Roboter innerhalb eines Arbeitsraumes operieren. Vereinzelt finden sich auch Beschreibungen zu Versuchen bei denen emergentes Verhalten demonstriert wird. Kaum findet man hingegen Vorschläge zur Steuerung einer Robotergruppe als Einheit. Dabei eröffnet gerade diese Form der Bewegung bei Verwendung spezialisierter Roboter eine enorme Flexibilität. Es bedarf nicht mehr eines möglichst universellen Roboters, sondern vielmehr unterschiedlich spezialisierter und miteinander kombinierbarer Roboter. Die Fähigkeiten eines Multirobotersystems zur Lösung von Aufgaben auf den Gebieten Manipulation, Wartung, Suche etc. übersteigen die eines einzelnen Roboters dann bei weitem. Mit steigender Zahl der Roboter nimmt jedoch das Problem der Koordination einen immer größeren Stellenwert innerhalb der Bewegungssteuerung ein.

Im Rahmen dieser Arbeit wurde daher eine Strategie erarbeitet, die es erlaubt, eine Gruppe von Robotern unter Einhaltung einer Formation zu steuern. Dabei wird im Wesentlichen auf die Koordination der Roboter bei der Bewältigung möglicher Hindernissituationen eingegangen. Mit dem Ziel, die Gruppe für einen Operator bedienbar zu machen, wird eine hybride Architektur vorgeschlagen. Vorgaben bezüglich einer Start- und Zielkonfigurationen werden von einer globalen Planung zunächst in eine Route für die gesamte Gruppe sowie eine Abfolge von Steuerverhalten für die einzelnen Roboter übersetzt. Jeder Roboter der Formation erhält dann eine für ihn situationsbezogen passende Liste von Verhalten, die im Ergebnis zu einem Übergang von der Start- zur Zielkonfiguration führen. Mindestens ein Roboter erhält zusätzlich eine Weginformation in Form einzelner Routensegmente. Die Aufgabe der Roboter besteht dann in der geeigneten Aktivierung der vorgegebenen Verhalten sowie in der Reaktion auf unerwartete lokale Ereignisse oder möglichen Abweichungen von der Planung. Ein Teil der Planungsstrategie befasst sich deshalb mit der Definition von Abbruchkriterien, bei deren Eintreten eine weitere Verfolgung des ursprünglichen Plans keinen Sinn mehr macht. Damit reduziert sich der Bedienaufwand für den Operator auf die Festlegung von Teilzielen bzw. die Kontrolle und ggf. Korrektur einzelner Roboter. Verglichen mit der simultanen Steuerung von nur drei Robotern ergibt sich eine deutliche Reduzierung der Komplexität.

Neben der Darstellung der Planungsstrategie erfolgte die Beschreibung einer konkreten Realisierung. Dabei wurde insbesondere Wert auf eine vollständige Umsetzung von der Modellierung bis zur Erprobung gelegt. Die dabei eingegangenen Vereinfachungen wurden analysiert und eine verbesserte Lösung vorgeschlagen. Schließlich wurde auch eine Experimentalplattform beschrieben, die im Verlauf dieser Arbeiten entwickelt wurde. Da gerade die praktische Erprobung mit einer Vielzahl von Detailproblemen behaftet ist, wurde eine kurze Analyse der erkannten Schwierigkeiten durchgeführt.

Im Ergebnis bietet die vorliegende Arbeit neben der eigentlichen Planungsstrategie auch Anregungen aus den Bereichen: Bildanalyse, Algorithmen, Softwaredesign und Informationstechnik. Damit zeigt sich ganz deutlich die Stellung der Robotik als Plattform für interdisziplinäre Forschung.

Die Forschungen an kooperativen Robotern stehen noch ganz am Anfang. Einerseits fehlen noch die praktischen Anwendungen, während andererseits die Forschungziele und Erwartungen schon recht hoch gesteckt werden. Realistische Einsatzfälle, bei denen mehrere mobile Roboter nutzbringend als ein System operieren, sind bisher nur bei der Lösung von Transportaufgaben zu finden. Die dabei anfallenden Probleme beziehen sich im Wesentlichen auf die Bahnplanung sowie auf die Lösung von Konfliktsituationen.
Betrachtet man die derzeitige Situation in den Bereichen Manipulation, Erkundung und Wartung, so kommen in den allermeisten Fällen einzelne teleoperierende Roboter zum Einsatz. Wie bereits eingangs erwähnt, eröffnen sich jedoch ganz neue Möglichkeiten, wenn mehrere Roboter zum Einsatz kommen, wobei jeder auf eine Aufgabe spezialisiert ist. Genau genommen gelten hier auch die beiden, bei der ersten Mars-Mission verwendeten Roboter, Pathfinder und Sojourner als Beispiel eines solchen heterogenen Multirobotersystems.

In diesem Sinne stellt die Arbeit einen Beitrag zur Bewegungssteuerung von Multirobotersystemen dar. Der Grundansatz – die Strategie – besteht in der Planung eines geeigneten Bewegungsablaufs und der dazu nötigen Steuerverhalten für eine Gruppe von Robotern mit dem Ziel, die nötigen Bedienhandlungen für einen Operator auf die Vorgabe der Zielkonfiguration zu reduzieren. Dabei wird insbesondere auch die Autonomie der Roboter ausgenutzt, um von einer Online- zu einer zeitdiskreten Offline-Steuerung überzugehen.

Anhand der Ergebnisse der Simulation sowie der ersten praktischen Versuche zeigt sich, dass die hier vorgestellte Planungsstrategie durchaus geeignet erscheint, als Rahmen für unterschiedliche Realisierungsvarianten zu dienen. Es wird aber auch deutlich, dass es bis zu einer praktisch verwertbaren Lösung einer Reihe von weiteren Entwicklungsschritten bedarf. Die im Rahmen dieser Arbeit entstandene Experimentalplattform kann dabei als Ausgangsbasis angesehen werden.

Weitere Schritte wären zunächst die softwareseitige Umsetzung des in Abschnitt 3.5.1 angedachten verbesserten Planungsmoduls sowie dessen Implementierung als Funktionsblock in MATLAB/Simulink®. Damit wäre eine höhere Flexibilität bei der Definition unterschiedlicher Simulationsmodelle gegeben.
Die globale Panung sollte neben den Steuerverhalten auch Vorschläge für lokal nötige Ausweichbewegungen vorgeben. Auf Grund von Toleranzen bei der Erkennung der Hindernisse sowie der Bewegung der Roboter kommt es sonst zu unerwarteten Ausweichbewegungen. Eine mögliche Lösung besteht hier in der Erstellung von Prädikaten. Schließlich fehlt bisher noch ein Algorithmus zur automatischen Generierung der Formationsgraphen. Hier müssen Geometrie und Kinematik der Roboter berücksichtigt werden.

Parallel sollten die Roboter mit einer lokalen Bildauswertung versehen werden, die eine darauf basierende Umsetzung der Verhalten „Folgen“ und „Ausweichen“ erlaubt. Erste Experimente dazu, vgl. Abschnitt 6.2.3, waren sehr vielversprechend. Es bedarf jedoch

einer eingehenden Untersuchung der Realisierbarkeit, der erreichbaren Dynamik und Zuverlässigkeit.

Anhang

A Algorithmen

In der vorliegenden Arbeit wurde mehrfach auf zwei Algorithmen Bezug genommen, die bei der Routensuche bzw. anderen Planungsvorgängen auf der Basis von Graphen (auch Rasterdarstellungen, als eine spezielle Form von Graphen) eine große Rolle spielen. An dieser Stelle soll das Prinzip beider Algorithmen zusammen mit ihrem prinzipiellen Unterschied erläutert werden.

A.1 Dijkstra Algorithmus

Dijkstra's Algorithmus basiert auf einer Breitensuche (BFS) und erweitert diese dahingehend, möglichst effektiv die kürzesten Wege von einem Startknoten *S* zu allen anderen Knoten zu bestimmen. Der Startknoten wird im ersten Schritt auch als *innerer Knoten* bezeichnet. Ausgehend vom Startknoten *S* werden alle Nachbarknoten besucht, um jeweils deren vorläufige kürzeste Abstände zu *S* zu bestimmen. Die Nachbarknoten werden dabei als *Randknoten* bezeichnet, während alle anderen, noch nicht besuchten Knoten, *äußere Knoten* sind. Im nächsten Schritt wird der Randknoten mit dem kürzesten Abstand zu *S* als innerer Knoten angenommen und wiederum alle dessen Nachbarknoten bestimmt. Dabei kann für einen schon bekannten Randknoten ein neuer kürzerer Abstand zu *S* ermittelt werden. In jedem Fall wird deshalb zu einem Randknoten dessen Vorgänger, über den also die kürzeste Route führt, gespeichert. Diese Iteration wird solange fortgesetzt, bis keine Randknoten mehr aufzusuchen sind.

Ein mögliche Art der Implementierung besteht in der Verwendung einer *Suchliste*, in der zunächst alle Knoten v_i des Graphen eingetragen werden. Wenn ein Zielknoten *Z* bekannt ist, wird dieser nicht mit in die Suchliste aufgenommen. Solange noch Knoten in der Suchliste eingetragen sind, wird immer der Knoten *u* entnommen, der den kürzesten Abstand zu *S* hat. Durch Vergleich mit den Nachbarknoten $v_n \in N(u)$ wird dann immer die kürzeste Route für den aktuellen Iterationsschritt bestimmt. Die Funktion *Distanz* liefert dabei den Abstand zwischen einem Knoten *u* und dessen Nachbarknoten v_n. Die kürzeste Route zwischen dem Startknoten *S* und dem Zielknoten *Z* ergibt sich schließlich durch Rückverfolgung der Vorgänger beginnend bei *Z*.

```
- Suchliste = S + [v1..vn] – Z (d.h. Suchliste enthält alle Knoten des Graphen, außer Ziel.)

for ∀Knoten v, ∈ Suchliste do
    - Distanz[v] = unendlich (d.h. maximale Entfernung zu S annehmen)
    - Vorgänger[v] = leer (d.h. noch kein Vorgänger bekannt)

- S.Distanz = 0 (d.h. Startknoten hat Distanz 0)
- S.Vorgänger = leer (d.h. Startknoten hat keinen Vorgänger)

while Suchliste ≠ leer do
    - Sortiere Suchliste nach Distanz
    - Wähle Knoten u aus Suchliste mit Distanz = minimal

    (Prüfe auf neue kürzeste Wege und aktualisiere Vorgänger und Distanz)
    for ∀Knoten v∈ N(u) (d.h. alle Nachbarknoten von u) do
        if v.Distanz > u.Distanz + Distanz(u,v) then
            - v.Distanz = u.Distanz + Distanz(u,v)
            - v.Vorgänger = u

    - Entferne u aus Suchliste

(Alle kürzesten Wege wurden gefunden, gib Weg von S zu Z rückwärts aus)
- Ausgabe Z
- u = Z;
while u ≠ leer do
    - u = u.Vorgänger
    - Ausgabe u
```

Abbildung A.1 Dijkstra Algorithmus in Pseudocode

A.2 A*-Algorithmus

Der A*-Algorithmus ist eine Erweiterung zum Dijkstra Algorithmus. Mit besonderem Hinblick auf die Verwendung als Routenplanungsalgorithmus wurde zusätzlich eine Heuristik zur Abstandsschätzung eines Knotens v_n zum Zielknoten Z eingeführt. Damit wird es im Verlauf der Suche möglich, den Nachbarknoten $v_n \in N(u)$ eines Knotens u zu bevorzugen, der mit der höchsten Wahrscheinlichkeit in Richtung zum Zielknoten Z liegt.
Die Wahl der Schätzfunktion (*Entfernung*(v_n,Z)) hat allerdings einen großen Einfluss auf das Ergebnis der Routenplanung. Wird die Entfernung überschätzt, d.h. die Werte der Schätzfunktion sind im Vergleich zu den realen Distanzen im Graph groß, so wird die Routenplanung beschleunigt, findet u.U. aber nicht die optimale Route. Sind die Werte der Schätzfunktion jedoch klein gegenüber den realen Distanzen, so wird eher die optimale Route gefunden.

```
- Suchliste = S (d.h. nur Startknoten S in Suchliste)
- Warteliste = leer

for ∀Knoten v do
    - v.Distanz = unendlich (d.h. maximale Entfernung zu S annehmen)
    - v.Vorgänger = leer (d.h. kein Vorgänger bekannt)
    - v.Heuristik = Entfernung(v->Z)

- S.Distanz = 0 (d.h. Startknoten hat Distanz 0)
- S.Vorgänger = leer (d.h. Startknoten hat keinen Vorgänger)
- S.Heuristik = 0
- S.Summe = S.Distanz + Entfernung(S,Z)

while Suchliste ≠ leer do
    - Sortiere Suchliste nach Summe
    - Wähle Knoten u aus Suchliste mit Summe = minimal
    - Entferne u aus Suchliste

    if u = Z then WEG_GEFUNDEN

    (Prüfe auf neue kürzeste Wege und aktualisiere Vorgänger und Distanz)
    for ∀Knoten v∈ N(u) (d.h. alle Nachbarknoten von u) do
        if (v.Distanz <= u.Distanz + Distanz(u,v)) and (v∈ Suchliste ∩ Warteliste) then
            - Bearbeite nächstes v (d.h. untersuche nächsten Nachbarknoten)

        - v.Distanz = u.Distanz + Distanz(u,v) (d.h. Aktualisiere reale kürzeste Distanz)
        - v.Summe = v.Distanz + Entfernung(v,Z) (d.h. Aktualisiere geschätzte Entfernung)
        - v.Vorgänger = u

        if v∈ Warteliste then
            - Entfernen von v aus Warteliste (d.h. v wieder verfolgen)

        if v∉ Suchliste then
            - Einfügen von v in Suchliste (d.h. v weiterverfolgen)

    - Einfügen von u in Warteliste (d.h. u evtl. später nochmal verfolgen)

WEG_NICHT_GEFUNDEN: (d.h. es wurde kein Weg gefunden)
Gebe aus: NULL

WEG_GEFUNDEN: (d.h. Weg wurde gefunden, gib den Weg von S zu Z rückwärts aus)
- Ausgabe Z
- u = Z
while u ≠ leer do
    - u = u.Vorgänger
    - Ausgabe u
```

Abbildung A.2 A* Algorithmus in Pseudocode

B Softwareumgebung

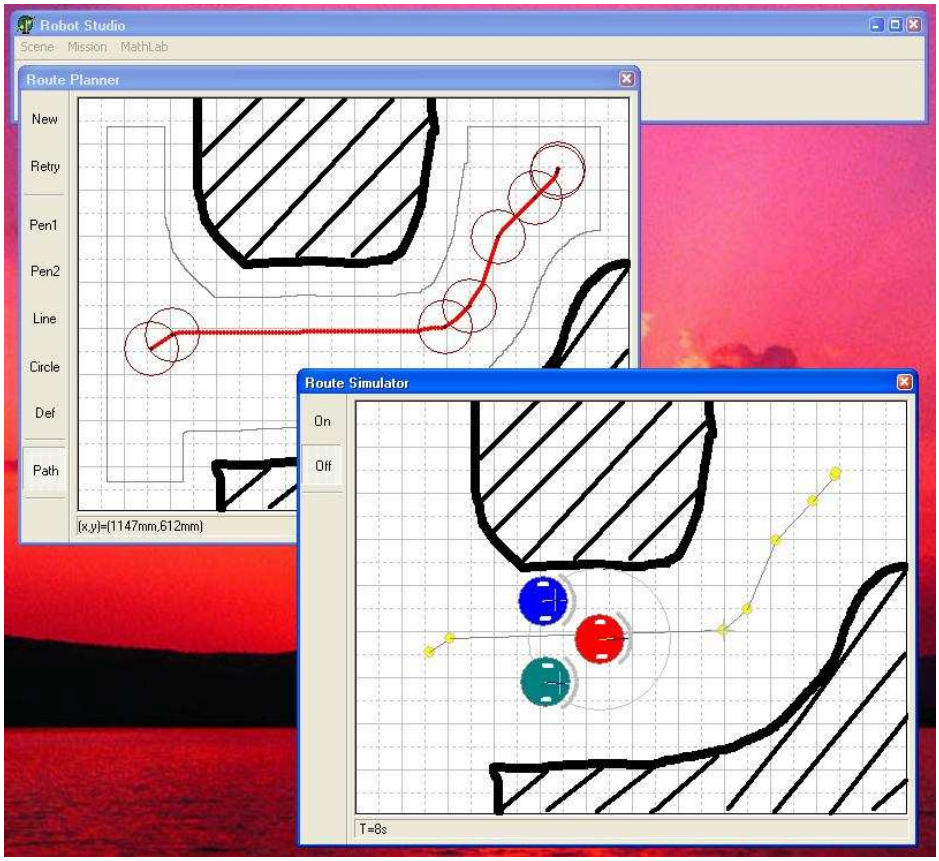


Abbildung B.1 Bildschirmansicht der Simulationsumgebung

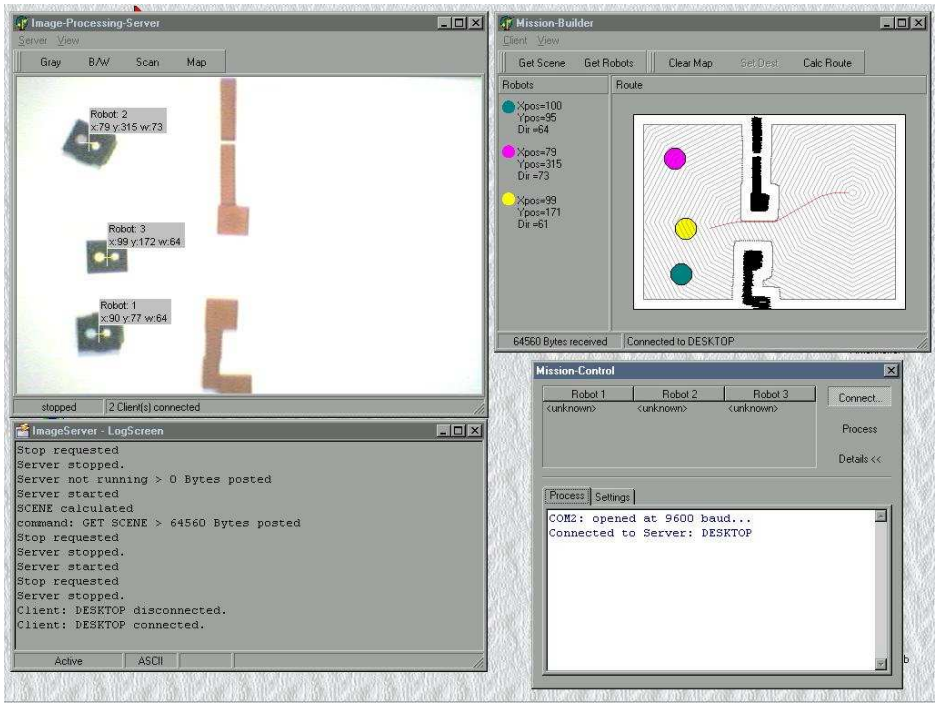


Abbildung B.2 Bildschirmansicht der Robotersteuerung

C Symbole und Abkürzungen

In der vorliegenden Arbeit werden folgende Notationen verwendet: Skalare Größen werden mit kursiven Kleinbuchstaben (z.B. *a,b*) bzeichnet, Vektoren werden zusätzlich unterstrichen (z.B. $\underline{v}$) oder mit Anfangs- und Endpunkt (z.B. $\underline{AB}$) angegeben. Koordinatensysteme werden mit fettgedruckten kursiven Großbuchstaben bezeichnet, während Punkte in ihnen nur durch kursive Großbuchstaben bezeichnet werden.

⇒ Koordinatensysteme:

$\boldsymbol{W}$	Weltkoordinatensystem
$\boldsymbol{R}$	Roboterkoordinatensystem

⇒ Punkte:

A, B, C, D, P, Z

⇒ Vektoren:

$\underline{S}$	Sensorinformation
$\underline{R}$	Aktorreaktion
$\underline{R}_G$	Gesamtreaktion
$\underline{F}$	einwirkende Kraft
$\underline{F}_s$	Steuervektor
$\underline{G}$	Hindernisvektor (*occupancy grid*)
$\underline{OZ}$	Zielvektor (Objekt->Ziel)
$\underline{p}$	Bewegungszustand
$\underline{v}$	Geschwindigkeit
$\underline{v}(t)$	zeitabhängige Geschwindigkeit
$\underline{v}_0$	Anfangsgeschwindigkeit
$\underline{v}_A$	Abstoßungsgeschwindigkeit
$\underline{v}_{LF}$	Führungsvektor
$\underline{v}_R$	Robotergeschwindigkeit
$\underline{v}_Z$	Zielgeschwindigkeit
$\underline{s}$	Position
$\underline{s}(t)$	zeitabhängige Position (Trajektorie)
$\underline{s}_0$	Anfangsposition

⇒ Mengen:

E_B	Menge von Endverhalten
F_B	Menge von Verhaltensfunktionen
R_B	Menge von Verhaltensreaktionen
S	Menge von Sensorinformationen

⇒ Symbole:

δ	Übergangsfunktion eines Automaten
ε	Toleranzschranke
ε_0	Anfangswert für Toleranzschranke
$\varphi(t)$	zeitanhängige Orientierung eines Roboters
φ_L	Orientierung des vorausfahrenden Roboters
φ_{LF}	Winkel zwischen vorausfahrendem und folgendem Roboter

ω_L	Winkelgeschwindigkeit des linken Rades
ω_R	Winkelgeschwindigkeit des rechten Rades
B	Verhalten
C	Konfigurationsraum
C_T	K-Raum-Zeit
D	Durchmesser
E	Kosten
e_i	Teilkosten
F	Freiraum
f_B	Verhaltensfunktion
f_C	Koordinationsfunktion
f_i	Einzelverhalten
f_s	Startverhalten
g_i	Wichtungsfaktoren
H_i	Hindernisse
$K(t)$	zeitabhängige Konfiguration eines Roboters
l	Radabstand
m	Masse
$N(i,j)$	Nachbarschaftselemente
N_O	Orthogonale Nachbarschaftselemente
N_D	Diagonale Nachbarschaftselemente
O	Objekt
R	Roboter
r	Radius
s	Wegstrecke
T	Zeitdauer
t	Zeitpunkt
t_s	Zeitschritt
$x(t)$	zeitanhängige X-Koordinate eines Roboters
$y(t)$	zeitanhängige Y-Koordinate eines Roboters

⇒ <u>Abkürzungen:</u>

A	Automat
A*	Suchalgorithmus
API	ApplicationProgrammers Interface
BFS	Breadth First Search (Traversierung eines Graphen)
CCD	Charged Coupled Device (bildgebender Sensor)
DLL	Dynamic Link Library
SRD	Situations Reaktions Diagramm

Abbildungsverzeichnis

Literaturverzeichnis

[Arkin, 1998*] Ronald C. Arkin: *Behavior Based Robotics*, The MIT Press, Cambridge, Massachusets, London England, 1998

[Asama, Fukuta, Arai u. Endo, 1996] H. Asama, T. Fukuta, T. Arai, I. Endo: *Distributed Autonomous Robotic Systems 2*, Springer-Verlag, Berlin Heidelberg, 1996

[Lueth, Dillmann u. Dario, 1998] T.Lueth, R. Dillmann, P.Dario, H.Wörn: *Distributed Autonomous Robotic Systems 3*, Springer-Verlag, Berlin Heidelberg, 1998

[Inoue, Ota, Hirano, Kurabayashi u. Arai, 1998] K. Inoue, J. Ota, T. Hirano, D. Kurabayashi u. T. Arai: *Iterative Transportation by Cooperative Mobile Robots in Unknown Environment*, University of Tokyo, in [Lueth, Dillmann u. Dario, 1998]

[Martinoli, Mondada, 1998] A. Martinoli, F. Mondada: *Probabilistic Modelling of Bio-Inspired Collective Experiment with Real Robots*, Swiss Federal Institute of Technology, in [Lueth, Dillmann u. Dario, 1998]

[Röfer, Müller, 1998] Th. Röfer, R. Müller: *Navigation and Routemark-Detection of the Bremen Autonomous Wheelchair*, University of Bremen, in [Lueth, Dillmann u. Dario, 1998]

[Arkin, Bekey, 1997] C. Arkin, A. Bekey: *Robot Colonies*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1997

[Bekey, 1998] Edited by George A. Bekey: *Autonomous Agents*, 5, 5 (1998), Kluwer Academic Publishers, Netherlands

[Brooks, 1991*] Rodney A. Brooks: *Intelligence Without Reason*, Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Morgan Kaufmann, 1991

[Kortenkamp, Bonasso u. Murphy, 1998] Edited by David Kortenkamp, R. Peter Bonasso, Robin Murphy: *Artificial Intelligence and Mobile Robots – Case studies of successful Robot Systems*, copublished by The MIT Press, Cambridge, Massachusets, London England, 1998

[Längle, 1997] Thomas Wolfgang Längle: *Verteiltes Steuerungskonzept für komplexe inhomogene Robotersysteme*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 614, VDI Verlag GmbH, Düsseldorf, 1997

[Hoelper, 1997] Ralf Hoelper: *Intelligentes, flexibel konfigurierbares Steuerungssystem für mobile Roboter*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 608, VDI Verlag GmbH, Düsseldorf, 1997

[Kruse, 1998+] Eckhard Kruse: *Bewegungsplanung für mobile Roboter in dynamischen Umgebungen auf Basis automatisch erzeugter statistischer Daten*, Fortschritte in der Robotik Bd. 4,Shaker Verlag, Aachen, 1998

[Azarm, 1997*] Kianoush Azarm: *Ein dezentrales Verfahren zur reaktiven und konfliktfreien Fahrzeugführung in Multiroboterumgebungen*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 644, VDI Verlag GmbH, Düsseldorf, 1997

[Rude, 1995+] Markus Rude: *Koordinierte Kollisionsvermeidung mobiler Roboter mit Hilfe von Kommunikation und Sensorik*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 495, VDI Verlag GmbH, Düsseldorf, 1995

[Langetepe, 2000] Elmar Langetepe: *Design and Analysis of Strategies for Autonomous Systems in Motion Planing*, Berichte aus der Informatik, Shaker Verlag, Aachen, 2000

[Desai, 1998+] Jaydev P. Desai: *Motion Planning and Control of Cooperative Robotic Systems, Dissertation*, Mechanical Engineering and Applied Mechanics, University of Pennsylvania, 1998

[Schmidt, 1995+] Peter Schmidt: *Hard- und Softwareseitige Realisierung einer Steuerung für einen mobilen Roboters mittels Fuzzy-Logik*, Diplomarbeit, Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau, 1995

[Müller, 1998] Dirk Müller: *Ein verhaltensorientierter Ansatz zum flächendeckenden Explorieren und Navigieren eines AMR*, Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern, 1998

[Brock, Montana u. Ceranowicz, 1992] David L. Brock, David J. Montana, Andrew Z. Ceranowicz: *Coordination and Control of Multiple Autonomous Vehicles*, Proc. of the IEEE Conference on Robotics and Automation Nice, France, 1992

[Freund, Rokossa, 1992+] E. Freund, D. Rokossa: *Automatische Bahngenerierung für Transportaufträge bei mobilen Robotern*, in P.Levi, Th.Bräunl, N.Oswald (Hrsg.):Autonome Mobile Systeme 1997, Springer-Verlag, Berlin Heidelberg New York, 1997

[Amendt, 1995] Oliver Amendt: *Neuronale Steuerung eines insektenartigen Schreitroboters*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 483, VDI Verlag GmbH, Düsseldorf, 1995

[Arkin, Bekey, 1997] R.C. Arkin, B.A. Bekey: *Cooperative Mobile Robotics: Antecedents and Directions*, Autonomous Robots 4, 1-23, Kluwer Academic Publishers, Boston, 1997

[Laumond, 1998] J.-P. Laumond (Ed.): *Robot Motion Planning and Control*, Lecture Notes in Control and Information Sciences 229, Springer-Verlag, London, 1998

[Li, 1996[+]] Sheng Li: *Verteilte Steuerung von kooperativen autonomen mobilen Robotern für Transportaufgaben*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 599, VDI Verlag GmbH, Düsseldorf, 1996

[Kinder, 1996] Margit Kinder: *Pfadplanung für Manipulatoren in komplexen Umgebungen mittels generalisierter Pfadspeicherung in Ellipsoidkarten*, Fortschrittsberichte VDI, Reihe 8:Meß-, Steuerung- und Regelungstechnik, Nr. 580, VDI Verlag GmbH, Düsseldorf, 1996

[Oliver, Saptharishi, Dolan, Trebi-Ollennu, Khosla,1999] C. Spence Oliver, Mahesh Saptharishi, John M. Dolan, Ashitey Trebi-Ollennu, Pradeep K. Khosla: *Multi-Robot Path planning by predicting structure in a dynamic environment*, contribution to IFACC 2000, Pittsburgh, Pasadene, 1999

[Schiedeck, 2002[+]] Florian Schiedeck: *Kameragestützte synchrone Steuerung mobiler Roboter auf zuvor berechneten Kursen*, Studienjahresarbeit, Fachbereich Systemanalyse, Technische Universität Ilmenau, 2002

[Siebentritt, 2002[+]] Harry Siebendritt: *Bestimmung der Position und Orientierung von mobilen Robotern in der Ebene anhand von Kamerabildern*, Studienjahresarbeit, Fachbereich Systemanalyse, Technische Universität Ilmenau, 2002

[Eichhorn, 1994] Mike-Joachim Eichhorn: *Gütekriterien und Suchverfahren für die Optimierung von Fuzzy-Systemen*, Diplomarbeit, Fachbereich Systemanalyse, Technische Universität Ilmenau, 1994

[Rechenberg, 1994] Ingo Rechenberg: *Evolutionsstrategie '94*, Friedrich Frommann Verlag, Stuttgart-Bad Cannstatt, 1994

[IASTED, 2001] Editor: M. H. Hamza: *Robotics and Applications, Proceedings of the IASTED International Conference 2001*, ACTA Press, Anaheim, Calgary, Zürich, 2001

[Reynolds, 1987[*]] Reynolds, Craig W.: Flocks, Herds, and Schools: A Distributed Behavioral Model, *ACM SIGGRAPH 1987, Conference Proceedings*, Anaheim California, July 1987 www.cs.toronto.edu/~dt/siggraph97-course/cwr87

[Reynolds, 1988[*]] Reynolds, Craig W.: Not bumping into things, *ACM SIGGRAPH 1988, Conference Proceedings*, Anaheim California, August 1988 www.red3d.com/cwr/nobump/nobump.html

[Reynolds, 1999[*]] Reynolds, Craig W.: Steering Behaviors, *GAME DEVELOPERS CONFERENCE 1999, Conference Proceedings*, Anaheim California, August 1999 www.red3d.com/cwr/steer

[Feilkas, Schnellhammer, 2002] Thomas Feilkas, Christian Schnellhammer: *Steering Behaviors*, Diplomarbeit, Fachbereich Informatik, Fachhochschule Regensburg, 2002

[Liu, Jianbing, 2001] Jiming Liu, Jianbing Wu: *Multi-Agent Robotic Systems*, CRC Press international series on computational intelligence, CRC Press, Boca Raton, London, New York, Washington, 2001

[Dudek, Jenkin, 2000] Dudek Gregory, Jenkin Michael : *Computational Principles of Mobile Robotics*, Cambridge University Press, London, New York, Washington, 2000

[Johnson, 2001] Johnson Steven: *Emergence: the connected lives of ants, brains, cities and Software*, Scribner, New York, 2001

[Hoppen, 1992+] Hoppen Peter: *Autonome Mobile Roboter: Echtzeitnavigation in bekannter und unbekannter Umgebung*, BI-Wissenschaftsverlag, Reihe Informatik, Band 87, Mannheim, Leipzig, Wien, Zürich,1992

[Knieriemen, 1991+] Knieriemen Thomas: *Autonome Mobile Roboter: Sensordateninterpretation und Weltmodellierung zur Navigation in unbekannter Umgebung*, BI-Wissenschaftsverlag, Reihe Informatik, Band 80, Mannheim, Wien, Zürich, 1991

[Altenburg, 2002] Altenburg Jens, Altenburg Uwe: *Mobile Roboter – Vom einfachen Experiment zur künstlichen Intelligenz*, 3. überarbeitete Auflage, Hanser Verlag, München, Wien, 2002

[Lee, 1961] Lee, C.: *An Algorithm for Path Connection and its Applications.*, IRE Trans. Electon. Comput., C-25 (1), 1961, S. 346-365

[Odell, 2002] Odell James: *Objects and Agents compared.*, in Journal of Object Technology (JOT), vol.1, no.1, May-June, 2002, S. 41-53
www.jot.fm/issues/issue 2002 05/column4

[Washburn, Evans, 1997] Washburn Kevin, Evans Jim: *TCP/IP*, 2. überarbeitete Auflage, Addison-Wesley-Longman, Bonn, 1997

[Rößmann, 1997] Rößmann Michael: *Applikationen entwickeln unter Windows NT4.0: Technologie, Design, Programmierung*, Addison-Wesley-Longman, Bonn, 1997

[Reß, Viebeck, 2000] Reß, Harald; Viebeck, Günter: *Datenstrukturen und Algorithmen – objektorientiertes Programmieren in C++*, Hanser Verlag, München, Wien, 2000

* Quellen, die auf die vorliegende Arbeit massgeblichen Einfluss hatten
+ Quellen, die vergleichbare Themen behandeln